The
Connection Machine
System

Paris Reference Manual

Version 6.0
February 1991

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, February 1989
Revised, February 1991

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142–1264
(617) 234–1000/876–1111

Contents

List of Figures

Customer Support

Thinking Machines Customer Support encourages customers to report errors in Connection Machine operation and to suggest improvements in our products.

When reporting an error, please provide as much information as possible to help us identify and correct the problem. A code example that failed to execute, a session transcript, the record of a backtrace, or other such information can greatly reduce the time it takes Thinking Machines to respond to the report.

To contact Thinking Machines Customer Support:

U.S. Mail:	Thinking Machines Corporation Customer Support 245 First Street Cambridge, Massachusetts 02142–1264
Internet Electronic Mail:	customer–support@think.com
Usenet Electronic Mail:	ames!think!customer-support
Telephone:	(617) 234–4000 (617) 876–1111

For Symbolics Users Only

The Symbolics Lisp machine, when connected to the Internet network, provides a special mail facility for automatic reporting of Connection Machine system errors. When such an error occurs, simply press Ctrl-M to create a report. In the mail window that appears, the `To:` field should be addressed as follows:

`To:` customer–support@think.com

Please supplement the automatic report with any further pertinent information.

Part II
Paris Dictionary

Chapter 9

Dictionary of Paris Instructions

9.1 Conventions for Alphabetizing

The operations and variables in this dictionary are ordered alphabetically, but with certain conventions that cause parts of the names to be ignored. The purpose is to ignore "prefixes" and "suffixes" in the name so as to group instructions that have the same main operation name.

- If the name contains a colon (and most do), the colon and any characters preceding it (usually "CM") are ignored.
- If the name begins with "fe-" then those three characters are dropped.
- Similarly, if the name begin with a single letter followed by a hyphen, those two characters are dropped.
- Similarly, if the name contains a single letter (or digit) surrounded by hyphens, each such letter (or digit) and the hyphen following it are dropped.
- Any occurrence of the modifier subsequence "-constant-" or "-const-" or "-always-" is replaced by a single hyphen.
- If the name ends in a hyphen, a digit, and the letter "L" then those three characters are dropped.
- Any asterisks in the name are dropped.

These rules are to be applied repeatedly and in any order until a name is reduced to a form where none of the rules apply.

The running heads on the top outside corners of the dictionary pages show the names with characters dropped according to these rules. Any ties in the ordering are broken by reconsidering letters dropped by the preceding rules.

As an example, CM:s-logcount-2-2L and CM:u-logcount-2-2L appear together (and in that order). As another example, CM:extract-news-coordinate-1L and CM:fe-extract-news-coordinate appear together (and in that order).

9.2 Programming Language Syntax

Paris is not a single language, but rather a library to be used within any of several programming languages, including C, Fortran, and Lisp. These languages have different syntactic conventions for names, operations, and procedure calls. This dictionary strikes a compromise among these conventions that allows straightforward transformations into the specific syntax of any of these languages. See chapters 6, 7, and 8 for information about language-specific aspects of the Paris interface.

9.2.1 Syntax of Names

All names in this dictionary are presented in Lisp syntax (specifically, that of Common Lisp). A simple rule is given below for converting such names to C or Fortran syntax.

Lisp allows names to contain hyphens, asterisks, and colons, among other characters. For the Lisp interface, Paris follows Common Lisp conventions for names:

- Words in a multiword name are separated with hyphens.
- The name of a global variable is surrounded with asterisks.
- Related names are grouped into a single package, indicated by a common prefix ending with a colon. Paris uses the prefix CM: for this purpose. Certain names used as constants, called *keywords*, have a null prefix, and therefore begin with a colon.

These rules are applied in the order given. Examples of names are CM:set-system-leds-mode, CM:s-add-2-1L, :news-order (a keyword), and CM:*maximum-exponent-length* (a global variable).

Fortran and Lisp are not case-sensitive, but C is. Therefore, this dictionary presents Paris instructions names using the upper-case and lower-case letters appropriate for C syntax. Similarly, to satisfy C and Fortran conventions, Paris names are limited to 32 characters (including any suffix and the trailing "L").

The rule for translating a Lisp name to a C or Fortran name has two parts.

- If the Lisp name begins with a colon, first add "CM" to the front.
- Then drop all asterisks, and convert all colons and hyphens to underscores.

Thus the example Lisp names shown above become CM_set_system_leds_mode, CM_s_add_2_1L, CM_news_order, and CM_maximum_exponent_length in C syntax.

For Fortran, this assumes a compiler that accepts 31-character names and permits underscores in names.

9.2.2 Pseudocode Instruction Descriptions

For most of the instructions *two* descriptions of the operation are given. One is in English, and the other is in pseudocode. The pseudocode is written in an *ad hoc* combination of programming constructs, mathematical notation, and occasional dabs of English. For the most part the notation should be self-explanatory, but several features deserve special remarks.

The constructs "let $x = y$" and "$x \leftarrow y$" are superficially similar; each causes x to have the value y. There are two differences, however. First, a "let" statement merely defines a temporary variable for later use in the pseudocode description of that instruction, whereas an arrow assignment represents an actual effect on the CM machine state (usually in the processor memories) that may be detected by subsequent Paris operations. Second, a "let" statement is assumed to give x the precise mathematical value computed for y, whereas an arrow assignment may have to truncate, round, or otherwise approximate the infinitely precise mathematical result before storing it.

When referring to actual machine state, square brackets are used to indicate a particular processor. For example, if *dest* names a field, then *dest*[*k*] refers to the contents of that field within processor k. Actual subscripts are used rather than square brackets for temporary quantities; thus one has "$dest[k] \leftarrow 1$" but "let $S_k = 1$" because the latter does not involve machine state.

Angle brackets are used to select bits within a field (or sometimes within an integer value, to be regarded as a field of bits in binary representation). For example, $dest[k]\langle 0 \rangle$ is the least significant bit of the field *dest* within processor k, and $dest[k]\langle 0 : 3 \rangle$ is the four least significant bits.

Multiplication is always indicated explicitly by the symbol $\times$, never by juxtaposition. The notation $\lfloor x \rfloor$ means the floor of x, the largest integer that is not greater than x; $\lfloor 3.5 \rfloor = 3$ and $\lfloor -3.5 \rfloor = -4$. The notation $\lceil x \rceil$ means the ceiling of x, the smallest integer that is not less than x; $\lceil 3.5 \rceil = 4$ and $\lceil -3.5 \rceil = -3$.

The symbols $\neg$, $\wedge$, $\vee$, and $\oplus$ respectively represent logical (or bitwise, if appropriate) NOT, AND, inclusive OR, and exclusive OR.

The symbols $\cap$ represents set intersection; $\cup$ is set union; $\setminus$ is set difference (thus $A \setminus B$ is the set of elements of A that are not in B); and $\in$ is the set inclusion predicate (and so $x \in A$ is true if x is an element of A).

Other mathematical notations are used freely, including square roots, summation signs, and set notation. The purpose of the pseudocode is to provide a clear explanation of the *results* of an operation, not to provide clues to performance; the particular algorithm shown is not necessarily the one used in the implementation.

F-ABS

Computes, in each selected processor, the absolute value of a floating-point source field and stores it in the destination field.

Formats CM:f-abs-1-1L *dest/source*, *s*, *e*
CM:f-abs-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source*$[k] \geq 0$ then *dest*$[k] \leftarrow$ *source*$[k]$
    else *dest*$[k] \leftarrow -$*source*$[k]$

The absolute value of the *source* operand is placed in the *dest* operand.

For floating-point numbers, absolute value is calculated by changing the sign bit to 0 (positive). All other bits in the number are unchanged.

F-C-ABS

The absolute value of the source field is returned in the destination field.

Formats CM:f-c-abs-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of the *dest* field in this format is $s + e + 1$. The total length of the *source* field in this format is $2(s + e + 1)$.

Overlap The *dest* field must be either identical to *source*, identical to $(source+s+e+1)$, or disjoint from *source*.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow \sqrt{(source[k].real)^2 + (source[k].imag)^2}$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The absolute value of the *source* operand is placed in the *dest* operand.

S-ABS

Computes the absolute value of a signed integer source field and stores it in the destination field.

Formats

CM:s-abs-1-1L	*dest/source*, *len*
CM:s-abs-2-1L	*dest*, *source*, *len*
CM:s-abs-2-2L	*dest*, *source*, *dlen*, *slen*

Operands

dest — The field ID of the signed integer destination field.

source — The field ID of the signed integer source field.

len — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen — The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen — The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source*$[k] \geq 0$ then *dest*$[k] \leftarrow$ *source*$[k]$
    else *dest*$[k] \leftarrow -$*source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The absolute value of the *source* operand is placed in the *dest* operand. (If the length of the *dest* field equals the length n of the *source* field, overflow can occur only if the *source* field contains -2^n. If the length of the *dest* field is greater than the length of the *source* field, then overflow cannot occur.)

C-ACOS

Computes, in each selected processor, the arc cosine of the complex source field and stores it in the complex destination field.

Formats CM:c-acos-1-1L *dest/source*, *s*, *e*
CM:c-acos-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \cos^{-1}$ *source*$[k]$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The arc cosine of the value of the *source* field is stored into the *dest* field.

The following definition of arc cosine determines the range and branch cuts for a complex number z.

$$-i \log \left(z + i\sqrt{1 - z^2}\right)$$

F-ACOS

Computes, in each selected processor, the arc cosine of the floating-point source field and stores it in the floating-point destination field.

Formats CM:f-acos-1-1L *dest/source*, *s*, *e*
CM:f-acos-2-1L *dest*, *source*, *s*, *e*

Operands
dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is less than -1 or greater than 1; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \cos^{-1} \textit{source}[k]$
    if $\textit{source}[k] < -1$ or $\textit{source}[k] > 1$ then
      $\textit{test-flag}[k] \leftarrow 1$
    else
      $\textit{test-flag}[k] \leftarrow 0$

The arc cosine of the value of the *source* field is stored into the *dest* field.

C-ACOSH

Computes, in each selected processor, the arc hyperbolic cosine of the complex source field and stores it in the complex destination field.

Formats CM:c-acosh-1-1L *dest/source*, *s*, *e*
CM:c-acosh-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \cosh^{-1}$ *source*$[k]$
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The arc hyperbolic cosine of the value of the *source* field is stored into the *dest* field.

The following definition of inverse hyperbolic cosine determines the range and branch cuts of a complex number z.

$$\log\left(z + (z+1)\sqrt{\frac{(z-1)}{(z+1)}}\right)$$

F-ACOSH

Computes, in each selected processor, the arc hyperbolic cosine of the floating-point source field and stores it in the floating-point destination field.

Formats CM:f-acosh-1-1L *dest/source, s, e*
CM:f-acosh-2-1L *dest, source, s, e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags

test-flag is set if the *source* is less than 1; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \cosh^{-1}$ *source*$[k]$
    if *source* < 1 then *test-flag*$[k] \leftarrow 1$
    else *test-flag*$[k] \leftarrow 0$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The arc hyperbolic cosine of the value of the *source* field is stored into the *dest* field.

C-ADD

The sum of two complex source values is placed in the destination field.

Formats

CM:c-add-2-1L	*dest*/*source1*, *source2*, *s*, *e*
CM:c-add-always-2-1L	*dest*/*source1*, *source2*, *s*, *e*
CM:c-add-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-add-always-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-add-constant-2-1L	*dest*/*source1*, *source2-value*, *s*, *e*
CM:c-add-const-always-2-1L	*dest*/*source1*, *source2-value*, *s*, *e*
CM:c-add-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:c-add-const-always-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest — The field ID of the complex destination field.

source1 — The field ID of the complex first source field.

source2 — The field ID of the complex second source field.

source2-value — A complex immediate operand to be used as the second source.

s, *e* — The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap — The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags — *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

if (always or $\textit{context-flag}[k] = 1$) then
  $\textit{dest}[k] \leftarrow \textit{source1}[k] + \textit{source2}[k]$
  if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

Two operands, *source1* and *source2*, are added as complex numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-ADD

The sum of two floating-point source values is placed in the destination field.

Formats

CM:f-add-2-1L	*dest/source1, source2, s, e*
CM:f-add-always-2-1L	*dest/source1, source2, s, e*
CM:f-add-3-1L	*dest, source1, source2, s, e*
CM:f-add-always-3-1L	*dest, source1, source2, s, e*
CM:f-add-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-add-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:f-add-constant-3-1L	*dest, source1, source2-value, s, e*
CM:f-add-const-always-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition

For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    $\textit{dest}[k] \leftarrow \textit{source1}[k] + \textit{source2}[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$

Two operands, *source1* and *source2*, are added as floating-point numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-ADD

The sum of two signed integer source values is placed in the destination field. Carry-out and overflow are also computed.

Formats

CM:s-add-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-add-2-1L	*dest/source1, source2, len*
CM:s-add-3-1L	*dest, source1, source2, len*
CM:s-add-constant-2-1L	*dest/source1, source2-value, len*
CM:s-add-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest — The field ID of the signed integer destination field.

source1 — The field ID of the signed integer first source field.

source2 — The field ID of the signed integer second source field.

source2-value — A signed integer immediate operand to be used as the second source.

len — The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen — For CM:s-add-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 — For CM:s-add-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 — For CM:s-add-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

carry-flag is set if there is a carry-out from the high-order bit position; otherwise it is cleared.

overflow-flag is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow$ *source1*$[k]$ + *source2*$[k]$
    carry-flag$[k] \leftarrow \langle$carry out in processor $k\rangle$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

Two operands, *source1* and *source2*, are added as signed integers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *carry-flag* and *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-ADD

The sum of two unsigned integer source values is placed in the destination field. Carry-out and overflow are also computed.

Formats

CM:u-add-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-add-2-1L	*dest/source1, source2, len*
CM:u-add-3-1L	*dest, source1, source2, len*
CM:u-add-constant-2-1L	*dest/source1, source2-value, len*
CM:u-add-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-add-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-add-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-add-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is a carry-out from the high-order bit position; otherwise it is cleared.

overflow-flag is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
$\textit{dest}[k] \leftarrow \textit{source1}[k] + \textit{source2}[k]$
$\textit{carry-flag}[k] \leftarrow \langle\text{carry out in processor } k\rangle$
if $\langle\text{overflow occurred in processor } k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$
else $\textit{overflow-flag}[k] \leftarrow 0$

Two operands, *source1* and *source2*, are added as unsigned integers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *carry-flag* and *overflow-flag* are altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

S-ADD-CARRY

The sum of the *carry-flag* and two signed integer source values is placed in the destination field. Carry-out and overflow are also computed.

Formats

CM:s-add-carry-3-3L *dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-add-carry-2-1L *dest*/*source1*, *source2*, *len*
CM:s-add-carry-3-1L *dest*, *source1*, *source2*, *len*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-add-carry-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-add-carry-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-add-carry-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is a carry-out from the high-order bit position; otherwise it is cleared.

overflow-flag is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow$ *source1*$[k]$ + *source2*$[k]$ + *carry-flag*$[k]$
carry-flag$[k] \leftarrow$ ⟨carry out in processor k⟩
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$

Two operands, *source1* and *source2*, are added as signed integers. The *carry-flag* is used as the carry-in to the low-order bits; the net effect is to compute the sum of *source1*, *source2*, and *carry-flag*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *carry-flag* and *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

U-ADD-CARRY

The sum of the *carry-flag* and two unsigned integer source values is placed in the destination field. Carry-out and overflow are also computed.

Formats

CM:u-add-carry-3-3L *dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-add-carry-2-1L *dest*/*source1*, *source2*, *len*
CM:u-add-carry-3-1L *dest*, *source1*, *source2*, *len*

Operands

dest — The field ID of the unsigned integer destination field.

source1 — The field ID of the unsigned integer first source field.

source2 — The field ID of the unsigned integer second source field.

len — The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen — For CM:u-add-carry-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 — For CM:u-add-carry-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 — For CM:u-add-carry-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is a carry-out from the high-order bit position; otherwise it is cleared.

overflow-flag is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
 if $\mathit{context\text{-}flag}[k] = 1$ then
  $\mathit{dest}[k] \leftarrow \mathit{source1}[k] + \mathit{source2}[k] + \mathit{carry\text{-}flag}[k]$

$carry\text{-}flag[k] \leftarrow \langle \text{carry out in processor } k \rangle$
if $\langle \text{overflow occurred in processor } k \rangle$ then $overflow\text{-}flag[k] \leftarrow 1$
else $overflow\text{-}flag[k] \leftarrow 0$

Two operands, *source1* and *source2*, are added as unsigned integers. The *carry-flag* is used as the carry-in to the low-order bits; the net effect is to compute the sum of *source1*, *source2*, and *carry-flag*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *carry-flag* and *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

S-ADD-FLAGS

The carry-out and overflow are computed for the sum of two signed integer source values. The sum itself is not stored.

Formats CM:s-add-flags-2-1L *source1*, *source2*, *len*

Operands

dest	The field ID of the signed integer destination field.
source1	The field ID of the signed integer first source field.
source2	The field ID of the signed integer second source field.
len	The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap: The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags: *carry-flag* is set if there is a carry-out from the high-order bit position; otherwise it is cleared.

overflow-flag is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context: This operation is conditional. The flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    Compute *source1*$[k]$ + *source2*$[k]$
    carry-flag$[k] \leftarrow \langle$carry out in processor $k\rangle$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

Two operands, *source1* and *source2*, are added as signed integers. The sum is not stored; only the *carry-flag* and *overflow-flag* are affected.

U-ADD-FLAGS

The carry-out and overflow are computed for the sum of two unsigned integer source values. The sum itself is not stored.

Formats CM:u-add-flags-2-1L *dest*, *source1*, *source2*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source1	The field ID of the unsigned integer first source field.
source2	The field ID of the unsigned integer second source field.
len	The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is a carry-out from the high-order bit position; otherwise it is cleared.

overflow-flag is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    Compute *source1*[k] + *source2*[k]
    carry-flag[k] ← ⟨carry out in processor k⟩
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0

Two operands, *source1* and *source2*, are added as unsigned integers. The sum is not stored; only the *carry-flag* and *overflow-flag* are affected.

F-ADD-MULT

Calculates a value $(a + x)b$ and places it in the destination.

Formats

CM:f-add-mult-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-add-mult-always-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-add-const-mult-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-add-const-mult-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-add-mult-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-add-mult-const-always-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-add-const-mult-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-add-const-mult-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source (addend) field.

source2 The field ID of the floating-point second source (augend) field.

source2-value A floating-point immediate operand to be used as the second source (augend).

source3 The field ID of the floating-point third source (multiplier) field.

source3-value A floating-point immediate operand to be used as the third source (multiplier).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

if (always or $\mathit{context\text{-}flag}[k] = 1$) then

$\mathit{dest}[k] \leftarrow (\mathit{source1}[k] + \mathit{source2}[k]) \times \mathit{source3}[k]$

if $\langle$overflow occurred in processor $k\rangle$ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$

ADD-MULT

Two operands *source1* and *source2* are added as floating-point numbers, and then the sum is multiplied by a third operand *source3*. The result is stored in the destination field.

The various formats allow the second source operand to be either a memory field or a constant.

The constant operand *source2-value* or *source3-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

A call to CM:f-add-mult-1L is equivalent to the sequence

CM:f-add-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-multiply-3-1L *dest*, *temp*, *source3*, *s*, *e*

but may be faster.

ADD-OFFSET-TO-FIELD-ID

Returns a new field ID that specifies the same field but possibly a different offset within that field.

Formats result ← CM:add-offset-to-field-id *field-id*, *offset*

Operands	*field-id*	A field ID.
	offset	A signed integer, the number of bits by which to offset the *field-id*.
Result	A field ID, identifying the newly offset field ID.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

Associates a new field ID with the portion of the specified field that begins at the specified bit offset. The size of the field referenced by the new field ID is equal to the size of the original field minus the offset. The offset must be smaller than the size in bits of the original field. Offset fields may themselves have offset fields formed from them.

ALLOCATE-HEAP-FIELD

Allocates a heap field of specified length in the current VP set and returns a unique identifier.

Formats result ← CM:allocate-heap-field *len*

Operands	*len*	An unsigned integer, the length in bits of the field to be allocated.
Result	A field ID, identifying the new field ID.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

A new field of length *len* is allocated in the heap within the current VP set. A field ID for the newly created field is returned.

ALLOCATE-HEAP-FIELD-VP-SET

Allocates a new heap field of the specified length in the specified VP set and returns a unique identifier.

Formats result ← CM:allocate-heap-field-vp-set *len*, *vp-set-id*

Operands	*len*	An unsigned integer, the length in bits of the field to be allocated.
	vp-set-id	A VP set ID. This may specify any VP set, including the current VP set.
Result	A field ID, identifying the new field ID.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

A new field of length *len* is allocated on the heap within the specified VP set. A field ID for the newly created field is returned.

ALLOCATE-STACK-FIELD

Allocates a new stack field of specified length in the current VP set and returns a unique identifier.

Formats result ← CM:allocate-stack-field *len*

Operands	*len*	An unsigned integer, the length, in bits, of the field to be allocated.
Result	A field ID, identifying the new field ID.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

A new field of length *len* is allocated on the stack within the current VP set. A field ID for the newly created field is returned.

ALLOCATE-STACK-FIELD-VP-SET

Allocates a new stack field of the specified length in the specified VP set and returns a unique identifier.

Formats result ← CM:allocate-stack-field-vp-set *len*, *vp-set-id*

Operands	*len*	An unsigned integer, the length in bits of the field to be allocated.
	vp-set-id	A VP set ID. This may specify any VP set, including the current VP set.
Result	A field ID, identifying the new field ID.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

A new field of length *len* is allocated on the stack within the specified VP set. A field ID for the newly created field is returned.

ALLOCATE-VP-SET

Create a new VP set, within which fields may be allocated.

Formats result ← CM:allocate-vp-set *geometry-id*

Operands *geometry-id* A geometry ID.

Result A VP set ID, identifying the newly allocated VP set.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation returns a vp-set-id for a newly created VP set. This may be given to other Paris operations in order to create memory fields in which data may be stored. The size and shape of the VP set is determined by the geometry specified by the *geometry-id*. It is possible to alter the geometry later (by using CM:set-vp-set-geometry), but the total number of virtual processors in the VP set remains forever fixed.

FE-ARRAY-FORMAT

This front-end instruction returns an array format descriptor. An array format descriptor may be passed to any array transfer instruction to specify a front-end array format, although this is not required.

See also CM:fe-packed-array-format and CM:fe-structure-array-format.

Formats result ← CM:fe-array-format *[cm-element-size, array-element-size, stride, ordering]*

Operands *cm-element-size* A signed integer immediate operand to be used as the number of bits each Connection Machine element occupies in the front-end array. This must be a power of two between 1 and 128.

In Lisp/Paris this is a keyword argument. If not specified, it defaults to *array-element-size*. If *array-element-size* is also not specified, *cm-element-size* defaults to the size of the Connection Machine field being read or written.

array-element-size A signed integer immediate operand to be used as the number of bits in each front-end array element. This must be a power of two between 1 and 128.

In Lisp/Paris this is a keyword argument. If not specified, *array-element-size* defaults to the actual front-end element size or, if the front-end array elements are general (i.e., of type t), *array-element-size* defaults to the value of *cm-element-size*.

stride A signed integer immediate operand to be used as the distance, in units of *array-element-size*, between adjacent front-end array elements. This must be either a null value or a positive integer between 1 and 65,535 that obeys the following restrictions. The product (*stride* × *array-element-size*) must be either a multiple of *cm-element-size* or a multiple of 32 bits. If *stride* is specified as a null value (null in C, 0 in Fortran, nil in Lisp), it defaults to the minimum legal value. In Lisp/Paris this is a keyword argument.

ordering The order in which Connection Machine elements are stored in a front-end array. The value of *ordering* must be either a null value or one of: :front-end-order, :lsb-first (least significant bit first), or :msb-first (most significant bit first). (These are CM_front_end_order, CM_lsb_first, or CM_msb_first from C or Fortran.) If specified as a null value (null in C, 0 in Fortran, nil in Lisp), it defaults to :front-end-order, which is the standard ordering for the front end. (Most significant bit first on Suns; least significant bit first on VAXes.) In Lisp/Paris this is a keyword argument.

Result — The array format descriptor specified.

Context — This is a front-end operation. It does not depend on the value of the *context-flag*.

The return value is a format descriptor for arrays; it can be passed to any array transfer instruction as the value of *format*. CM:fe-array-format provides the most generality in specifying an array format for tranfers. More specific descriptors may be obtained with CM:fe-packed-array-format and CM:fe-structure-array-format.

The value of *cm-element-size* defines the unit of measure for the *fe-offset-vector* argument to the CM:read-from-news-array and CM:write-to-news-array instructions.

The value of *array-element-size* defines the unit of measure for the *fe-dimension-vector* argument to the CM:read-from-news-array and CM:write-to-news-array instructions. However, for extended-element array transfers, the unit of measure for the *fe-dimension-vector* argument is (*array-element-size* × *stride*).

If *cm-element-size* is less than *array-element-size*, a packed transfer is specified. That is, multiple Connection Machine array elements are packed into each front-end array element. If *cm-element-size* is greater than *array-element-size*, an extended-element array is specified. That is, more than one front-end array element is used to store each Connection Machine array element.

For most arrays, the value of *stride* is 1. For packed array transfers, *stride* must be 1. For extended-element array transfers, the stride must be large enough to ensure that consecutive elements do not overlap on the front end. To read or write every other (non-packed, non-extended) front-end array element, use a *stride* value of 2.

For a normal (non-packed, non-extended) array transfer, specify *ordering* as a null value.

A packed format with :lsb-first ordering stores the Connection Machine element with the smallest coordinates in the least significant bits of the array element. A packed format with :msb-first ordering stores the CM element with the largest coordinates in the most significant bits of the front-end array.

An extended-element format with :lsb-first ordering stores the low-order bits of the Connection Machine element in the front-end array location with the smallest coordinate. An extended-element format with :msb-first ordering stores the high-order bits of the CM element in the front-end array location with the smallest coordinate.

AREF

Takes array elements specified by a per-processor index and copies them into a fixed destination.

Formats CM:aref-2L *dest, array, index, dlen, index-len, index-limit, element-len*

Operands

dest The field ID of the destination field.

array The field ID of the source array field.

index The field ID of the unsigned integer index into the array field.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*.

element-len An unsigned integer immediate operand to be used as the length of an array element.

Overlap The fields *array* and *index* may overlap in any manner. However, the *array* and *index* fields must not overlap the *dest* field.

Flags *test-flag* is set if the value in the *index* field is less than the *index-limit*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    if $\mathit{index}[k] < \mathit{index\text{-}limit}$ then
      let $p = \mathit{index}[k] \times \mathit{element\text{-}len}$
      $\mathit{dest}[k] \leftarrow \mathit{array}[k]\langle p : p + \mathit{dlen} - 1\rangle$
      $\mathit{test\text{-}flag}[k] \leftarrow 1$
    else
      $\mathit{test\text{-}flag}[k] \leftarrow 0$

This is a simple form of array reference, for arrays stored in the memory of individual processors. Each processor has an array index stored in the field *index*. This is used to index into an *array*, whose length in bits should be *index-limit* $\times$ *element-len*. The element indexed (or a portion of it) is copied into *dest* in all selected processors. Thus different processors may access different elements of their arrays.

More precisely, a field of length *dlen* and starting at address $array + i \times element\text{-}len$, where i is the unsigned number stored at *index*, is copied to *dest* in all selected processors.

The argument *index-limit* is one greater than the largest allowed value of the index. Those processors that have index values greater than or equal to *index-limit* do not alter the value of the destination field; they also clear *test-flag*. All processors in which the index field is less than *index-limit* set *test-flag*. The argument *element-len* is the length of individual elements of the array. Usually this will be the same as *dest-length*, but for certain applications it is worthwhile for it to differ. For example, from an array of 128-bit records one may fetch just one 16-bit component of an indexed record by letting *dlen* be 32, letting *element-len* be 128, and by offsetting the *array* address by the offset within each record of the 16-bit quantity to be fetched. As another example, to extract a 4-character substring from a string of 8-bit characters, one may let *dlen* be 32 and *element-len* be 8.

AREF32

Takes array elements specified by a per-processor index and copies them into a fixed destination. The array is stored in a special format that allows fast access.

Formats

CM:aref32-2L *dest, array, index, dlen, index-len, index-limit*
CM:aref32-always-2L *dest, array, index, dlen, index-len, index-limit*

Operands

dest The field ID of the destination field.

array The field ID of the source array field. This must contain data stored in a special format by either CM:aset32 or CM:transpose32.

index The field ID of the unsigned integer index field. This is used as the per-processor index into the *array*.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This is taken as the *array* element length and must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the *array* extent.

Overlap

The fields *array* and *index* may overlap in any manner. However, the *array* and *index* fields must not overlap the *dest* field.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition

For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    if $\textit{index}[k] < \textit{index-limit}$ then
      let $r = \textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))$
      let $m = \left\lfloor \frac{k}{r} \right\rfloor \bmod 32$
      let $i = \textit{index}[k]$
      for all j such that $0 \le j < \textit{dlen}$ do
        $\textit{dest}[k]\langle j \rangle \leftarrow \textit{array}[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor) \rangle$
    else
      ⟨error⟩

This is a simple form of array reference for parallel arrays whose elements are stored across the memory of individual processors. To each processor belongs an array of extent *index-limit* with elements of length *dlen*.

The *array* element indexed by each active processor is copied into the *dest* field of that processor. Different processors may reference different elements of their arrays. For this reason, this form of array referencing is known as *indirect addressing*.

Each processor has an array index stored in the field *index*. This is used to index into an area of CM memory, *array*, whose allocated length in bits should be at least

$$\left(\mathit{index\text{-}limit} \times \left\lceil \frac{\mathit{dlen}}{32} \right\rceil\right) \times 32$$

The argument *index-limit* is one greater than the largest allowed value of the index. It is an error for any *index* value to equal or exceed this limit.

A field of length *dlen*, and starting at address $array + i \times 32$, where i is the the unsigned number stored at *index*, is copied to *dest* in all selected processors. Even this is not quite accurate, because the array data is not organized in the same manner as for CM:aref. Instead, it is organized in a peculiar way for fast per-processor access. Parallel arrays stored in this format are termed *slicewise parallel arrays*.

Slicewise parallel array data is arranged with successive bits stored in successive processors within groups of 32 virtual processors. Thus, slicewise array data belonging to one processor is spread over the memories of the 32 processors in its group and the memory of each processor holds data belonging to all 32 processors.

A region of memory set aside for a slicewise array of the format required by CM:aref32 should be accessed only through the operations CM:aset32 and CM:aref32, related operations such as CM:get-aref32 and CM:send-aset32-overwrite, or operations that copy the array as a whole from all processors (such as I/O operations). It is also possible to operate on this memory in blocks of 32-bit square matrices with the CM:transpose32 instruction.

AREF32-SHARED

Takes an array element specified by a per-processor index and copies it into to a fixed destination. The source array is stored in a special format that allows fast access, and is accessed in such a way that all the virtual processors within a group of 32 physical processors share the same array.

Formats

CM:aref32-shared-2L	*dest, array, index, dlen, index-len, index-limit*
CM:aref32-shared-always-2L	*dest, array, index, dlen, index-len, index-limit*

Operands

dest The field ID of the destination field.

array The field ID of the source array field. This must be a contiguous region in CM memory. It need not be in the current VP set.

index The field ID of the unsigned integer index field. This is used as the per-processor index into *array*.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This is normally taken as the length of *array elements and must be a multiple of 32. As a special case, dlen may be 8 or 16 and, if so, access into both the source and the destination fields is offset appropriately.*

index-len *The length of the index field. This must be non-negative and no greater than* CM:*maximum-integer-length*.

index-limit *An unsigned integer immediate operand to be used as the exclusive upper bound for the index. This is taken as the extent of array if dlen is a multiple of 32. However, if dlen is 8 or 16, then index-limit is taken as the number of 32-bit elements that would fit into the array field.*

Overlap The fields *array* and *index* may overlap in any manner. However, the *array* and *index* fields must not overlap the *dest* field.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    if $index[k] < index\text{-}limit$ then

for all j such that $0 \leq j < dlen$ do
$dest[k]\langle j \rangle \leftarrow$
$array\left[32\left\lfloor\frac{k}{32r}\right\rfloor + (j \bmod 32)\right]\left\langle index\text{-}limit\left\lfloor\frac{j}{32}\right\rfloor + index[k]\right\rangle$
else
⟨error⟩

where r is the VP ratio, and where j is the bit position in each field.

This is a simple form of array reference for arrays whose elements are stored across the memory of individual processors and accessed in such a way that many processors appear to share a single array of extent *index-limit* with elements of length *dlen*.

The shared array element (or a portion of it) indexed is copied into *dest* in all (selected) processors. Different processors may access different elements of the shared array. For this reason, this form of array referencing is known as *indirect addressing.*

Each processor has an array index stored in the field *index*. This is used to index into *array*. The argument *index-limit* is one greater than the largest allowed value of the index. It is an error for any *index* value to equal or exceed this limit.

The data within the source array area is not organized in the same manner as for CM:aref; instead, it is organized in a peculiar way for fast per-processor access. Shared arrays stored in this format are termed *slicewise shared arrays.*

Slicewise shared array data is arranged with successive bits stored in successive processors, within groups of 32 physical processors. Each 32-bit word of each element is stored separately in processor memories, as follows: The low-order 32 bits of all elements are grouped together across processor memories in a field of length $32 \times$ *index-limit* bits. Similarly, the next 32 bits of all elements are grouped together, and so on, up to the high-order bits of all array elements. This data format allows fast hardware-supported access to the individual elements of a shared array.

A region of memory set aside for an array of the format required by CM:aref32-shared must be contiguous in memory. It must therefore be allocated all at once, at a VP ratio of 1, with a single call to CM:allocate-stack-field or to CM:allocate-heap-field. Alternatively, from Lisp, the memory may be allocated within a with-stack-field form at a VP ratio of 1.

The area of CM memory occupied by *array* should be allocated at a VP ratio of 1 as a field whose length in bits is exactly

$$index\text{-}limit \times \left\lceil \frac{dlen}{32} \right\rceil$$

Shared array memory should be accessed only with the operations CM:aref32-shared and CM:aset32-shared, or with operations that copy the array as a whole from all processors (such as I/O operations). Data in such a region of memory may, however, be reoriented with the CM:transpose32 instruction.

As a special case, if the *dlen* argument is specified as 8 or 16, then each processor accesses one byte or one half-word of a 32-bit element. The *index-limit* argument must be specified as the extent of the array when considered to contain 32-bit elements. Nonetheless, valid *index* values are integers 0 through 2 or 4 times this *index-limit*. The *index* argument may be thought of as consisting of two fields, one that indexes a 32-bit array element and one that indexes an 8- or 16-bit offset into that element. To index bytes, the low 2 bits of *index* specify the offset. To index half-words, the low 1 bit of *index* specifies the offset.

ASET

Stores into an array element specified by a per-processor index a value copied from a fixed source field.

Formats CM:aset-2L *source, array, index, slen, index-len, index-limit, element-len*

Operands

source The field ID of the source field.

array The field ID of the destination array field.

index The field ID of the unsigned integer index into the array field.

slen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*.

element-len An unsigned integer immediate operand to be used as the length of an array element.

Overlap The fields *source* and *index* may overlap in any manner. However, the *source* and *index* fields must not overlap the *array* field.

Flags *test-flag* is set if the value in the *index* field is less than the *index-limit*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then
  if $\textit{index}[k] < \textit{index-limit}$ then
    let $p = \textit{index}[k] \times \textit{element-len}$
    $\textit{array}[k]\langle p : p + \textit{slen} - 1\rangle \leftarrow \textit{source}[k]$
    $\textit{test-flag}[k] \leftarrow 1$
  else
    $\textit{test-flag}[k] \leftarrow 0$

This is a simple form of array modification, for arrays stored in the memory of individual processors. Each processor has an array index stored in the field *index*. This is used to index into an *array*, whose length in bits should be *index-limit* $\times$ *element-len*. The *source* field is copied into the element indexed (or a portion of it) in all selected processors. Thus different processors may modify different elements of their arrays.

More precisely, the *source* field is copied to a field of length *slen* and starting at address *array* $+ i \times$ *element-len*, where i is the unsigned number stored at *index*, in all selected processors.

The argument *index-limit* is one greater than the largest allowed value of the index. Those processors that have index values greater than or equal to *index-limit* do not alter the value of the destination field; they also clear *test-flag*. All processors in which the index field is less than *index-limit* set *test-flag*. The argument *element-len* is the length of individual elements of the array. Usually this will be the same as *dest-length*, but for certain applications it is worthwhile for it to differ. For example, within an array of 128-bit records one may store into just one 16-bit component of an indexed record by letting *slen* be 32, letting *element-len* be 128, and by offsetting the *array* address by the offset within each record of the 16-bit quantity to be modified. As another example, to modify a 4-character substring of a string of 8-bit characters, one may let *slen* be 32 and *element-len* be 8.

ASET32

Copies data from a fixed source to the destination array elements specified by a per-processor index. The destination array is stored in a special format that allows fast access.

Formats CM:aset32-2L *source, array, index, slen, index-len, index-limit*

Operands

source The field ID of the source field.

array The field ID of the destination array field.

index The field ID of the unsigned integer index field. This is used as the per-processor index into *array*.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This is taken as the *array* element length and must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the *array* extent.

Overlap The fields *source* and *index* may overlap in any manner. However, the *source* and *index* fields must not overlap the *array* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
if $\mathit{index}[k] < \mathit{index\text{-}limit}$ then
let $r = \mathit{geometry\text{-}total\text{-}vp\text{-}ratio}(\mathit{geometry}(\mathit{current\text{-}vp\text{-}set}))$
let $m = \left\lfloor \frac{k}{r} \right\rfloor \bmod 32$
let $i = \mathit{index}[k]$
for all j such that $0 \le j < \mathit{slen}$ do
$\mathit{array}[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle \leftarrow \mathit{source}[k]\langle j \rangle$
else
$\langle$error$\rangle$

This is a simple form of array modification for parallel arrays whose elements are stored across the memory of individual processors. To each processor belongs an array of extent *index-limit* with elements of length *slen*.

The *source* field value for each active processor is copied into the indexed array element belonging to that processor. Thus different processors may modify different elements of their arrays. For this reason, this form of array access is known as *indirect addressing.*

Each processor has an array index stored in the field *index*. This is used to index into an area of CM memory, *array*, whose allocated length in bits should be at least

$$\left(\textit{index-limit} \times \left\lceil \frac{slen}{32} \right\rceil\right) \times 32$$

The argument *index-limit* is one greater than the largest allowed value of the index. It is an error for any *index* value to equal or exceed this limit.

In all selected processors, the *source* field is copied to a field of length *slen* and starting at address $array + i \times 32$, where i is the the unsigned number stored at *index*. Even this is not quite accurate, because the data within the destination *array* area is not organized in the same manner as for CM:aset. Instead, it is organized in a peculiar way for fast per-processor access. Parallel arrays stored in this format are termed *slicewise parallel arrays.*

Slicewise parallel array data is arranged with successive bits stored in successive processors within groups of 32 virtual processors. Thus, slicewise array data belonging to one processor is spread over the memories of the 32 processors in its group and the memory of each processor holds data belonging to all 32 processors.

A region of memory set aside for a slicewise array of the format required by CM:aset32 should be accessed only through the operations CM:aref32 and CM:aset32, related operations such as CM:send-aset32-overwrite and CM:get-aref32, or operations that copy the array as a whole from all processors (such as I/O operations). It is also possible to operate on this memory in blocks of 32-bit square matrices with the CM:transpose32 instruction.

ASET32-SHARED

Copies data from a fixed source to the destination array elements specified by a per-processor index. The array is stored in a special format that allows fast access, and is accessed in such a way that all the virtual processors within a group of 32 physical processors share the same array.

Formats CM:aset32-shared-2L *source*, *array*, *index*, *slen*, *index-len*, *index-limit*

Operands

source The field ID of the source field.

array The field ID of the destination array field. This must be contiguous region in CM memory. It need not be in the current VP set.

index The field ID of the unsigned integer index field. This is used as the per-processor index into the *array*.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be a multiple of 32 and is taken as the array element length.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of *array*.

Overlap The fields *source* and *index* may overlap in any manner. However, the *source* and *index* fields must not overlap the *array* field.

Context This operation is conditional, but whether data is copied depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is stored into the field indicated by *array* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{index}[k] < \textit{index-limit}$ then
      for all j such that $0 \le j < \textit{dlen}$ do
        $\textit{array}\left[32\left\lfloor\frac{k}{32r}\right\rfloor + (j \bmod 32)\right]\left\langle \textit{index-limit}\left\lfloor\frac{j}{32}\right\rfloor + \textit{index}[k]\right\rangle$
        $\rightarrow \textit{source}[k]\langle j\rangle$
    else
      ⟨error⟩

where r is the VP ratio, and where j is the bit position in each field.

For any two active virtual processors, k and k', if $index[k] = index[k']$, then either $source[k]$ or $source[k']$ is stored in *dest*, depending upon the implementation.

This is a simple form of array modification for arrays whose elements are stored across the memory of individual processors and accessed in such a way that many processors appear to share a single array of extent *index-limit* with elements of length *slen*.

The *source* field in each selected processor is copied into the array element (or a portion of it) indexed. Different processors may modify different elements of the shared array. For this reason, this form of array referencing is known as *indirect addressing*. If several processors sharing the same array attempt to modify the same element in a single CM:aset32-shared operation, then one of the values is stored and the rest are discarded.

Each processor has an array index stored in the field *index*. This is used to index into *array*. The argument *index-limit* is one greater than the largest allowed value of the index. It is an error for any *index* value to equal or exceed this limit.

The data within the destination array area is not organized in the same manner as for CM:aset; instead, it is organized in a peculiar way for fast per-processor access. Shared arrays stored in this format are termed *slicewise shared arrays*.

Slicewise shared array data is arranged with successive bits stored in successive processors, within groups of 32 physical processors. Each 32-bit word of each element is stored separately in processor memories, as follows: The low-order 32 bits of all elements are grouped together across processor memories in a field of length $32 \times$ *index-limit* bits. Similarly, the next 32 bits of all elements are grouped together, and so on, up to the high-order bits of all array elements. This data format allows fast hardware-supported access to the individual elements of a shared array.

A region of memory set aside for an array of the format required by CM:aset32-shared must be contiguous in memory. It must therefore be allocated all at once, at a VP ratio of 1, with a single call to CM:allocate-stack-field or to CM:allocate-heap-field. Alternatively, from Lisp, the memory may be allocated within a with-stack-field form at a VP ratio of 1.

An area of CM memory occupied by *array* should be allocated at a VP ratio of 1 as a field whose length in bits is exactly

$$\textit{index-limit} \times \left\lceil \frac{slen}{32} \right\rceil$$

Shared array memory should be accessed only with the operations CM:aref32-shared and CM:aset32-shared, or with operations that copy the array as a whole from all processors (such as I/O operations). Data in such a region of memory may, however, be reoriented with the CM:transpose32 instruction.

C-ASIN

Calculates the arc sine of the complex source field values and stores the result in the complex destination field.

Formats CM:c-asin-1-1L *dest/source*, *s*, *e*
CM:c-asin-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \sin^{-1}$ *source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The arc sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

The following definition of arc sine determines the range and branch cuts of a complex number z.

$$-i \log\left(i \times z + \sqrt{1 - z^2}\right)$$

F-ASIN

Calculates the arc sine of the floating-point source field values and stores the result in the floating-point destination field.

Formats CM:f-asin-1-1L *dest/source*, *s*, *e*
CM:f-asin-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is less than -1 or greater than 1; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \sin^{-1} \textit{source}[k]$
    if $\textit{source}[k] < -1$ or $\textit{source}[k] > 1$ then
      $\textit{test-flag}[k] \leftarrow 1$
    otherwise $\textit{test-flag}[k] \leftarrow 0$

The arc sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

C-ASINH

Calculates the arc hyperbolic sine of the complex source field values and stores the result in the complex destination field.

Formats CM:c-asinh-1-1L *dest/source*, *s*, *e*
CM:c-asinh-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*[k] = 1 then
dest[k] $\leftarrow \sinh^{-1}$ *source*[k]

The arc hyperbolic sine of the value of the *source* field is stored into the *dest* field.

The following definition of the inverse hyperbolic sine determines the range and branch cuts for a complex number z.

$$\log\left(z + \sqrt{1 + z^2}\right)$$

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-ASINH

Calculates the arc hyperbolic sine of the floating-point source field values and stores the result in the floating-point destination field.

Formats CM:f-asinh-1-1L *dest/source*, *s*, *e*
CM:f-asinh-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \sinh^{-1}$ *source*$[k]$
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The arc hyperbolic sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

C-ATAN

Calculates the arc tangent of the complx source field values and stores the result in the complex destination field.

Formats CM:c-atan-1-1L *dest/source*, *s*, *e*
CM:c-atan-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

test-flag is set if *source* contains i or $-i$, where i $C(0,1)$; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow \tan^{-1} source[k]$

The arc tangent of the value of the *source* field is stored into the *dest* field.

The following definition for arc tangent determines the range and branch cuts for a complex number *z*.

$$-i\log\left((1+i\times z)\times\sqrt{\frac{1}{(1+z^2)}}\right)$$

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-ATAN

Calculates the arc tangent of the floating-point source field values and stores the result in the floating-point destination field.

Formats CM:f-atan-1-1L *dest*/*source*, *s*, *e*
CM:f-atan-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow \tan^{-1} source[k]$

The arc tangent of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-ATAN2

Calculates the arc tangent of the quotient of two floating-point source fields and stores the result in the floating-point destination field.

Formats CM:f-atan2-3-1L *dest*, *source1*, *source2*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source1	The field ID of the floating-point y source field.
source2	The field ID of the floating-point x source field.
s, *e*	The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

test-flag is set if *source1* and *source2* are both zero; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    if $\mathit{source2}[k] > 0$ then
      $\mathit{dest}[k] \leftarrow \tan^{-1} \frac{\mathit{source1}[k]}{\mathit{source2}[k]}$
    else if $\mathit{source2}[k] < 0$ then
      $\mathit{dest}[k] \leftarrow \mathit{sign}(\mathit{source1}[k]) \times \left(\pi - \tan^{-1} \left|\frac{\mathit{source1}[k]}{\mathit{source2}[k]}\right|\right)$
    else if $\mathit{source1}[k] = 0 \wedge \mathit{sign}(\mathit{source2}[k]) > 0$ then
      $\mathit{dest}[k] \leftarrow \mathit{sign}(\mathit{source1}[k]) \times 0$
    else if $\mathit{source1}[k] = 0 \wedge \mathit{sign}(\mathit{source2}[k]) < 0$ then
      $\mathit{dest}[k] \leftarrow \mathit{sign}(\mathit{source1}[k]) \times \pi$
    else
      $\mathit{dest}[k] \leftarrow \mathit{sign}(\mathit{source1}[k]) \times \frac{\pi}{2}$
    if ⟨overflow occurred in processor k⟩ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$

The arc tangent of the quotient of the *source1* and *source2* fields is stored into the *dest* field. The signs of the source fields are taken into account to produce a result in the correct quadrant of the Cartesian plane.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

C-ATANH

Calculates the arc hyperbolic tangent of the complex source field values and stores the result in the complex destination field.

Formats CM:c-atanh-1-1L *dest/source, s, e*
CM:c-atanh-2-1L *dest, source, s, e*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

test-flag is set if *source* is 1 or -1; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
$\textit{dest}[k] \leftarrow \tanh^{-1} \textit{source}[k]$

The arc hyperbolic tangent of the value of the *source* field is stored into the *dest* field.

The following definition of the arc hyperbolic tangent determines the range and branch cuts for a complex number z.

$$\log\left((1+z)\sqrt{1-\frac{1}{z^2}}\right)$$

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-ATANH

Calculates the arc hyperbolic tangent of the floating-point source field values and stores the result in the floating-point destination field.

Formats CM:f-atanh-1-1L *dest/source*, *s*, *e*
CM:f-atanh-2-1L *dest*, *source*, *s*, *e*

Operands
- *dest* — The field ID of the floating-point destination field.
- *source* — The field ID of the floating-point source field.
- *s*, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags — *test-flag* is set if the magnitude of *source* is greater than or equal to 1; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context — This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \tanh^{-1} \textit{source}[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$
    if $|\textit{source}[k]| \geq 1$ then $\textit{test-flag}[k] \leftarrow 1$
    otherwise $\textit{test-flag}[k] \leftarrow 0$

The arc hyperbolic tangent of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

ATTACH

Attaches the Connection Machine hardware to the front end computer and returns the number of physical processors attached.

This instruction is available only from the Lisp/Paris interface. For Fortran/Paris and C/Paris users, the equivalent functionality is provided by the shell level cmattach command, documented in the *CM System User's Guide.*

Formats result ← CM:attach *[physical-size], [interface], [wait-p]*

Operands

physical-size The number of physical processors to be attached. This argument is an optional argument.

interface The integer indicating a particular bus interface to be used. This is an optional keyword argument and defaults to 0. When specified, the invocation must include the keyword :interface followed by an integer.

wait-p The answer to the question, "Do you want to wait for processors to become available?". This is an optional keyword argument and defaults to nil. When specified, the invocation must include the keyword :wait-p followed by T or NIL.

Result An unsigned integer, the exact number of physical processors allocated.

Context This operation is unconditional. It does not depend on the *context-flag.*

From the Lisp/Paris interface, this function allocates Connection Machine processors for use by the front end. To deallocate the processors, use CM:detach.

In the Lisp/Paris interface, CM:attach is a function of several arguments.

The *physical-size* argument is optional; if no *physical-size* argument is specified, then the smallest possible amount of hardware will be allocated. This default is the smallest number of processors associated with one sequencer, and varies between 8,192 and 16,384 physical processors, depending of site requirements.

If specified, the *physical-size* argument indicates the number of processors desired. It may be any one of the following values:

:8kp or **8192** Exactly 8,192 physical processors are to be allocated.

:16kp or **16384** Exactly 16,384 physical processors are to be allocated.

:32kp or **32768** Exactly 32,768 physical processors are to be allocated.

:64kp or **65536** Exactly 65,536 physical processors are to be allocated.

Alternatively, the *physical-size* argument may specify the sequencer or sequencers desired by using one of the following values: (These options are useful primarily for hardware diagnostic procedures.)

:ucc0, **:ucc1**, **:ucc2**, or **:ucc3** Exactly the specified sequencer (also known as a microcontroller port) is to be attached, regardless of whether that port controls 8,192 or 16,384 physical processors.

:ucc0-1, **:ucc2-3**, or **:ucc0-3** Exactly the specified sequencers (0 and 1, 2 and 3, or all four) are to be attached, regardless of the number of physical processors involved.

The :interface keyword argument is used at sites with more than one Connection Machine. If used, it indicates which Connection Machine is to be attached by specifying the integer value of the interface for the desired Connection Machine.

The :wait-p keyword is used if you want to wait for the requested processors to become available. To quit waiting, type Ctrl-C. (From Gmacs, type Ctrl-C, Ctrl-C; from a Lisp Machine front end, type Ctrl-ABORT.)

The value returned by CM:attach is the number of physical processors that were attached.

An error is signalled if the required number of physical processors or the required set of microcontroller ports is not available.

The
variable CM:*before-attach-initializations* and the variable CM:*after-attach-initializations* contain sets of initialization forms that are respectively evaluated before and after anything else occurs.

Note: On a Symbolics Lisp Machine, the Lisp/Paris interface will also accept :8k, :16k, :32k, and :64k as *physical-size* specifications. However, these are not valid symbols in all Common Lisp implementations—technically speaking, they have the syntax of "potential numbers" in Common Lisp—and therefore users are encouraged to use the forms :8kp, :16kp, :32kp, and :64kp in code to ensure portability. The "k" forms will continue to be available to preserve back-compatibility with existing code that uses them.)

In the C/Paris and Fortran/Paris interfaces, the attaching operation is performed by a user command cmattach at shell level. See the *CM System User's Guide* manual or the cmattach man page for more information.

ATTACHED

Returns true if the front end process has Connection Machine processors attached for use.

Formats result ← CM:attached

Result True if the front end process has Connection Machine processors attached for use, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This predicate allows a program to determine whether there are any Connection Machine processors attached (whether actual hardware or simulated) before it issues other Paris operations.

AVAILABLE-MEMORY

Determines the number of bits of memory, per virtual processor, that remain available for allocation on either the heap or the stack.

Formats result ← CM:available-memory

Result An unsigned integer, the number of bits available.

Context This operation is unconditional. It does not depend on the *context-flag*.

The number of bits available for allocation by either CM:allocate-heap-field or CM:allocate-stack-field is returned to the front end as an integer. The return value represents the number of bits available for each virtual processor in the current VP set.

F-F-CEILING

Determines the smallest integral value that is not less than the floating-point source field value in each selected processor and stores it in the floating-point destination field.

Formats CM:f-f-ceiling-1-1L *dest/source*, *s*, *e*
CM:f-f-ceiling-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow \lceil source[k] \rceil$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $+\infty$, which is stored into the *dest* field as a floating-point-number.

Note that overflow cannot occur.

S-CEILING

The ceiling of the quotient of two signed integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:s-ceiling-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-ceiling-2-1L	*dest/source1, source2, len*
CM:s-ceiling-3-1L	*dest, source1, source2, len*
CM:s-ceiling-constant-2-1L	*dest/source1, source2-value, len*
CM:s-ceiling-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer quotient field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-ceiling-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-ceiling-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-ceiling-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then

$$dest[k] \leftarrow \left\lceil \frac{source1[k]}{source2[k]} \right\rceil$$

if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

else $overflow\text{-}flag[k] \leftarrow 0$

if $source2[k] = 0$ then

$test[k] \leftarrow 1$

else $test[k] \leftarrow 0$

The signed integer *source1* operand is divided by the signed integer *source2* operand. The ceiling of the mathematical quotient is stored into the signed integer memory field *dest.*

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len.*

S-F-CEILING

The floating-point source field values are converted to signed integer values and stored in the destination field.

Formats CM:s-f-ceiling-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the floating-point source field.

len The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow \lceil source[k] \rceil$
    if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $+\infty$. The result is stored into the *dest* field as a signed integer.

U-CEILING

The ceiling of the quotient of two unsigned integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:u-ceiling-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-ceiling-2-1L	*dest*/*source1*, *source2*, *len*
CM:u-ceiling-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-ceiling-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:u-ceiling-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-ceiling-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-ceiling-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-ceiling-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then

$$dest[k] \leftarrow \left\lceil \frac{source1[k]}{source2[k]} \right\rceil$$

if $\langle$overflow occurred in processor $k\rangle$ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$
else $\mathit{overflow\text{-}flag}[k] \leftarrow 0$
if $\mathit{source2}[k] = 0$ then
  $\mathit{test}[k] \leftarrow 1$
else $\mathit{test}[k] \leftarrow 0$

The unsigned integer *source1* operand is divided by the unsigned integer *source2* operand. The ceiling of the mathematical quotient is stored into the unsigned integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-F-CEILING

The floating-point source field values are converted to unsigned integer values and stored in the destination field.

Formats CM:u-f-ceiling-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands	*dest*	The field ID of the unsigned integer destination field.
	source	The field ID of the floating-point source field.
	len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap: The fields *dest* and *source* must not overlap in any manner.

Flags: *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest ← $\lceil$*source*$\rceil$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $+\infty$, which is stored into the *dest* field as an unsigned integer.

CHANGE-FIELD-ALIAS

Changes the referent of the specified field alias.

Formats CM:change-field-alias *alias-id*, *field-id*

Operands *alias-id* An alias field ID. This must be an alias field ID returned by CM:make-field-alias. It need not be in the current VP set.

field-id A field ID. This must be a field id returned by CM:allocate-stack-field or CM:allocate-heap-field; it may *not* be an offset into a field. The field need not be in the current VP set.

Context This operation is unconditional. It does not depend on the *context-flag*.

The alias field ID *alias-id* is made to reference the field identified by *field-id*. This function allows field aliases to be recycled.

After a call to CM:change-field-alias, the field length and the physical length associated with *alias-id* are exactly what they would be if CM:make-field-alias had been called with *field-id*.

An error is signaled if the physical length of the aliased field is not exactly divisible by the VP ratio of the VP set to which *field-id* belongs. (For more on the physical length associated with an alias field see the dictionary entry for CM:make-field-alias.)

The alias field ID can be used in all the same ways as a regular field ID can, with the following exceptions:

- It cannot be passed to CM:deallocate-heap-field.
- It cannot be passed to CM:deallocate-stack-through.

C-F-CIS

Calculates the cosine and sine for the floating-point source field and stores the result in the complex destination field.

Formats CM:c-f-cis-2-1L *dest*, *source*, *s*, *e*

Operands	*dest*	The field ID of the complex destination field.
	source	The field ID of the floating-point source field.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of the *dest* field in this format is $2(s+e+1)$. The total length of the *source* field in this format is $s+e+1$.

Overlap: The *source* field must be either identical to *dest*, identical to $(dest+s+e+1)$, or disjoint from *dest*.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k].real \leftarrow \cos source[k]$
    $dest[k].imag \leftarrow \sin source[k]$

The result is a complex number whose real part is the cosine of the *source* and whose imaginary part is the sine of the *source*. The term cis signifies $\cos + i \sin$.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

CLEAR-ALL-FLAGS

Clears all flags (but not the context bit).

Formats CM:clear-all-flags
CM:clear-all-flags-always

Context The non-always operation is conditional.

The always operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $\textit{context-flag}[k] = 1$) then
$\textit{test-flag}[k] \leftarrow 0$
$\textit{overflow-flag}[k] \leftarrow 0$

Within each processor, all flags for that processor are cleared (but not the context bit).

CLEAR-BIT

Clears a specified memory bit.

Formats CM:clear-bit *dest*
CM:clear-bit-always *dest*

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
    if (always or $context\text{-}flag[k] = 1$) then
        $dest[k] \leftarrow 0$

The destination memory bit is cleared within each selected processor.

CLEAR-CONTEXT

Unconditionally makes all processors inactive.

Formats CM:clear-context

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$context\text{-}flag[k] \leftarrow 0$

Within each processor, the context bit for that processor is unconditionally cleared.

CLEAR-flag

Clears a specified flag bit.

Formats CM:clear-test
CM:clear-overflow

Context This operation is conditional.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
flag$[k] \leftarrow 0$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, the indicated flag for that processor is cleared.

COLD-BOOT

This operation completely resets the state of the hardware allocated to the executing front end, loads microcode, initializes system tables, and clears user memory.

Formats result ← CM:cold-boot *microcode-version*, *dimensions*

Operands *microcode-version* Either :paris or :diagnostics. This specifies which version of the microcode is to be used. This argument is optional (actually a keyword argument in the Lisp interface).

dimensions The dimension information for initializing the NEWS grid. This argument is optional (actually a keyword argument in the Lisp interface).

Result In the Lisp/Paris interface *three* results are returned (as Common Lisp "multiple values"):

An unsigned integer, the number of virtual processors.

An unsigned integer, the number of physical processors.

An unsigned integer, the number of bits available per virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

The facility for cold-booting Connection Machine hardware is provided in different ways in the Lisp/Paris interface (on the one hand) and the C/Paris and Fortran/Paris interfaces (on the other hand).

In the Lisp/Paris interface, CM:cold-boot is a function that accepts optional keyword arguments.

The `:microcode-version` argument specifies what set of microcode is to be loaded into the microcontroller(s). There are two choices for this argument: `:paris` (the default) specifies microcode that interprets the macroinstruction set, and `:diagnostics` specifies special microcode used for hardware maintenance.

The `:dimensions` argument is largely obsolete now that multiple VP sets may be allocated, but it is still supported for the sake of compatibility with previous releases of Paris. The `:dimensions` argument must be an integer, a list of 1 or 2 integers, or unsupplied. (Passing `nil` as the value is the same as not supplying a value.) An integer or a list of one integer specifies the total number of *virtual* processors desired. A list of two integers specifies the desired size of the *virtual* NEWS grid. Each dimension must be a power of two.

If the `:dimensions` argument is unsupplied, then the configuration of virtual processors depends on the most recent CM:cold-boot or CM:attach operation preceding this one. If the most recent such operation was CM:cold-boot, then the same virtual processor configuration set up then will be used this time. If the most recent such operation was CM:attach, then the number of virtual processors will be equal to the number of physical processors, and the virtual NEWS grid will have the same shape as the physical NEWS grid.

Bootstrapping a Connection Machine system includes the following actions:

- Evaluating all initialization forms stored in the variable CM:*before-cold-boot-initializations*. This is done before anything else.
- Loading microcode into the Connection Machine microcontroller and initiating microcontroller execution.
- Clearing and initializing the memory of allocated Connection Machine processors.
- Initializing all of the global configuration variables described in section 3.7.
- Initializing the pseudo-random number generator by effectively invoking the operation CM:initialize-random-number-generator with no seed.
- Initializing the system lights-display mode by effectively invoking the operation CM:set-system-leds-mode with an argument of t.
- Evaluating all initialization forms stored in the variable CM:*after-cold-boot-initializations*. This is done after everything else.

If the cold-booting operation fails, then an error is signalled. If it succeeds, then three values are returned: the number of virtual processors, the number of physical processors, and the number of bits available for the user in each virtual processor. (These are exactly the values of the configuration variables CM:*user-cube-address-limit*, CM:*physical-cube-address-limit*, and CM:*user-memory-address-limit*.

In the C/Paris and Fortran/Paris interfaces, the cold-booting operation is performed by a user command cmcoldboot at shell level. See the *Front End Subsystems* manual.

F-COMPARE

Compares two floating-point source values and stores into the signed integer destination field the result `-1`, `0`, or `1` depending on whether the first source value is less than, equal to, or greater than the second source value.

Formats CM:f-compare-3-2L *dest*, *source1*, *source2*, *dlen*, *s*, *e*

Operands

dest	The field ID of the signed integer destination field.
source1	The field ID of the floating-point first source field.
source2	The field ID of the floating-point second source field.
dlen	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
s, *e*	The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap: The fields *dest* and *source1* must not overlap in any manner. The fields *dest* and *source2* must not overlap in any manner. The fields *source1* and *source2* may overlap in any manner.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

```
if context-flag[k] = 1 then
   if source1[k] < source2[k] then
      dest[k] ← −1
   else if source1[k] > source2[k] then
      dest[k] ← 1
   else
      dest[k] ← 0
```

Two operands are compared as floating-point numbers. The destination receives the signed integer value `-1`, `0`, or `1` depending on whether the first source value is less than, equal to, or greater than the second source value.

S-COMPARE

Compares two signed integer source values and stores into the signed integer destination field the result `-1`, `0`, or `1` depending on whether the first source value is less than, equal to, or greater than the second source value.

Formats CM:s-compare-3-3L *dest, source1, source2, dlen, slen1, slen2*

Operands	*dest*	The field ID of the signed integer destination field.
	source1	The field ID of the signed integer first source field.
	source2	The field ID of the signed integer second source field.
	dlen	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
	slen1	The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
	slen2	The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap: The fields *dest* and *source1* must not overlap in any manner. The fields *dest* and *source2* must not overlap in any manner. The fields *source1* and *source2* may overlap in any manner.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if $source1[k] < source2[k]$ then
      $dest[k] \leftarrow -1$
    else if $source1[k] > source2[k]$ then
      $dest[k] \leftarrow 1$
    else
      $dest[k] \leftarrow 0$

Two operands are compared as signed integers. The destination receives the value `-1`, `0`, or `1` depending on whether the first source value is less than, equal to, or greater than the second source value.

U-COMPARE

Compares two unsigned integer source values and stores into the signed integer destination field the result `-1`, `0`, or `1` depending on whether the first source value is less than, equal to, or greater than the second source value.

Formats CM:u-compare-3-3L *dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source1* must not overlap in any manner. The fields *dest* and *source2* must not overlap in any manner. The fields *source1* and *source2* may overlap in any manner.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then
  if $\textit{source1}[k] < \textit{source2}[k]$ then
    $\textit{dest}[k] \leftarrow -1$
  else if $\textit{source1}[k] > \textit{source2}[k]$ then
    $\textit{dest}[k] \leftarrow 1$
  else
    $\textit{dest}[k] \leftarrow 0$

Two operands are compared as unsigned integers. The destination receives the signed integer value `-1`, `0`, or `1` depending on whether the first source value is less than, equal to, or greater than the second source value.

COMPRESS-HEAP

Invokes the heap compression mechanism on demand.

Formats CM:compress-heap

Context This operation is unconditional. It does not depend on the *context-flag*.

Heap compression removes heap memory fragmentation.

By default, the configuration variable CM:*heap-compression-enabled* is T (true), causing automatic heap compression whenever the stack and heap try to grow into each other. Therefore, under normal circumstances it not necessary to use the CM:compress-heap instruction.

Automatic heap compression can, however, make performance calculations unpredictable. To ensure deterministic performance, set CM:*heap-compression-enabled* to NIL (false, 0), arrange data structures to avoid fragmentation where possible, and explicitly invoke CM:compress-heap as necessary.

The variable CM:*heap-compression-messages-enabled* determines whether a message is issued when heap compression occurs. By default, this value is T (true, 1) and heap compression messages are issued. If this variable is NIL (false, 0), heap compression occurs without report.

C-CONJUGATE

The conjugate of the complex *source* field is placed in the complex *dest* field.

Formats CM:c-conjugate-1-1L *dest/source, s, e*
CM:c-conjugate-2-1L *dest, source, s, e*

Operands

dest	The field ID of the complex destination field.
source	The field ID of the complex source field.
s, e	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\mathit{context\text{-}flag}[k] = 1$ then
  $\mathit{dest}[k].\mathit{real} \leftarrow \mathit{source}[k].\mathit{real}$
  $\mathit{dest}[k].\mathit{imag} \leftarrow -\mathit{source}[k].\mathit{imag}$

Given a complex number C the conjugate C' consists of a real part equal to the real part of C and an imaginary part equal to the negation of the imaginary part of C. The conjugate of the complex *source* field is placed in the *dest* field.

C-COS

Calculates the cosine of the complex source field and stores the result in the complex destination field.

Formats CM:c-cos-1-1L *dest/source*, *s*, *e*
CM:c-cos-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \cos$ *source*$[k]$
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The cosine of the value of the complex *source* field is stored into the complex *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-COS

Calculates, in each selected processor, the cosine of the floating-point source field value and stores it in the floating-point destination field.

Formats CM:f-cos-1-1L *dest/source*, *s*, *e*
CM:f-cos-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
$\mathit{dest}[k] \leftarrow \cos \mathit{source}[k]$

The cosine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

C-COSH

Calculates, in each selected processor, the hyperbolic cosine of the complex source field value and stores it in the complex destination field.

Formats CM:c-cosh-1-1L *dest/source*, *s*, *e*
CM:c-cosh-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
$\textit{dest}[k] \leftarrow \cosh \textit{source}[k]$
if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$

The hyperbolic cosine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-COSH

Calculates the hyperbolic cosine of the floating-point source field and stores it in the floating-point destination field.

Formats CM:f-cosh-1-1L *dest/source*, *s*, *e*
CM:f-cosh-2-1L *dest*, *source*, *s*, *e*

Operands	*dest*	The field ID of the floating-point destination field.
	source	The field ID of the floating-point source field.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags: *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \cosh$ *source*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The hyperbolic cosine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

CREATE-DETAILED-GEOMETRY

Creates a new geometry given detailed information about how the grid is laid out.

For most applications, the simpler CM:create-geometry instruction is recommended over this one. Use CM:create-detailed-geometry only to tune the performance of an application with stable, known inter-processor communication patterns. (See also CM:intern-geometry and CM:intern-detailed-geometry).

Formats result ← CM:create-detailed-geometry *axis-descriptor-array, [rank]*

Operands *axis-descriptor-array* A front-end vector of descriptors for the grid axes.

In the C interface, the elements of the *axis-descriptor-array* must be of type CM_axis_descriptor_t, that is, they must be pointers to structures of type CM_axis_descriptor.

In the Lisp interface, the *axis-descriptor-array* may be either a list of descriptors or an array of descriptors.

rank An unsigned integer, the rank (number of dimensions) of the geometry being created. This must be between 1 and CM:*max-geometry-rank*, inclusive. This argument is not provided when calling Paris from Lisp.

Result A geometry ID, identifying the newly created geometry. This is of type CM_geometry_id_t in C, of type CM:geometry-id in Lisp, and an integer in Fortran.

Context This operation is unconditional. It does not depend on the *context-flag*.

CM:create-detailed-geometry takes an array of axis descriptors, one for each axis. The operation returns a geometry ID, which may then be used to create a VP set or to respecify the geometry of an existing VP set.

Each axis descriptor specified by CM:axis-descriptor-array is a structure describing one NEWS axis in some detail. Most of the descriptor components are unsigned integers, but the value of the *ordering* component is different. From Lisp, the *ordering* component must be either :news-order, :send-order, or :framebuffer-order. From C or Fortran, it must be either CM_news_order, CM_send_order, or CM_framebuffer_order.

The C definitions of the type of the ordering component and of the axis descriptor are shown below. Notice that the elements of the *axis_descriptor_array* must be pointers to type struct CM_axis_descriptor.

```
typedef enum {CM_news_order, CM_send_order } CM_axis_order_t;
typedef struct CM_axis_descriptor {
  unsigned length;
  unsigned weight;
  CM_axis_order_t ordering;
  unsigned char on_chip_bits;
  unsigned char off_chip_bits;
} * CM_axis_descriptor_t;
```

Actually, this structure has other components as well. C code should use the definition of CM_axis_descriptor from the cmtypes.h include file.

The Fortran/Paris interface defines CM_axis_descriptor as an array:

```
INTEGER RANK,DESCRIPTOR_ARRAY(7,RANK)
```

The elements of each Fortran axis descriptor are defined such that:

$DESCRIPTOR\_ARRAY(1, I)$ is the length of axis I
$DESCRIPTOR\_ARRAY(2, I)$ is the weight of axis I
$DESCRIPTOR\_ARRAY(3, I)$ is the ordering of axis I
$DESCRIPTOR\_ARRAY(4, I)$ is the on-chip bits of axis I
$DESCRIPTOR\_ARRAY(6, I)$ is the off-chip bits of axis I

Thus CM:axis-descriptor-array is, in Fortran, an array of axis descriptor arrays.

The Lisp definitions of the type of the ordering component and of the axis descriptor are shown below.

```
(deftype cm:axis-order () '(member :news-order :send-order))
(defstruct CM:axis-descriptor
  (length 0) (weight 0) (ordering :news-order)
  (on-chip-bits 0) (off-chip-bits 0))
```

The *axis-descriptor-array* operand must be created by first making one axis descriptor for each axis and then using these to assign values to the array elements. An example in C is given below. Notice that *axis1* and *axis2* are *pointers* to axis descriptor structures and that the descriptor structures are zeroed before any values are assigned.

```
CM_geometry_id_t  my_geometry;
CM_axis_descriptor_t  my_geometry_axes[2];
CM_axis_descriptor_t  axis1, axis2;
```

```
axis1 = (cm_axis_descriptor_t)malloc(sizeof(struct CM_axis_descriptor));
axis2 = (cm_axis_descriptor_t)malloc(sizeof(struct CM_axis_descriptor));
bzero(axis1, sizeof(struct CM_axis_descriptor));
bzero(axis2, sizeof(struct CM_axis_descriptor));
axis1->length = 128;
axis2->length = 256;
axis1->weight = 5;
axis2->weight = 10;
axis1->ordering = CM_news_order;
axis2->ordering = CM_news_order;

my_geometry_axes[0] = axis1;
my_geometry_axes[1] = axis2;
my_geometry = CM_create_detailed_geometry(my_geometry_axes, 2);
```

The following example specifies the same axes, descriptor array, and geometry in Lisp. Notice that the constructor CM:make-axis-descriptor is used.

```
(setq my-geometry-axes make-array(2))
(setq axis1
 (CM:make-axis-descriptor :length 128 :weight 5
  :ordering :news-order))
(setq axis2
 (CM:make-axis-descriptor :length 256 :weight 10
  :ordering :news-order)))
(setf (aref my-geometry-axes 0) axis1)
(setf (aref my-geometry-axis 1) axis2)
(setq my-geometry (CM:make-detailed-geometry my-geometry-axes 2)
```

Once the geometry has been created, the user may destroy the descriptors and the array used to provide axis information. All necessary information is copied out of these structures as the geometry is created.

The "length" component of an axis descriptor specifies the length of the axis; it must be a power of two.

The "weight" component of the axis descriptors specifies the relative frequency of interprocessor communication along different axes. For instance, in the above example it is assumed that communication occurs about half as often along *axis1*, which is given a weight of 5, as along *axis2*, which is given a weight of 10. Only the relative values of the weight components matter. The same communication traffic could be specified with weights of 1 and 2, or of 3 and 6. If all weights are 1, it is assumed that all axes are used equally frequently.

Given a set of weight components, Paris lays out the hypercube grid for optimal performance. Virtual processors are mapped onto the physical hypercube in a pattern that exploits the fact that communication is especially rapid among virtual processors within the same physical processor and among virtual processors within the same physical chip.

The "ordering" component of an axis descriptor specifies how NEWS coordinates are mapped onto physical processors for that axis. The value :news-order specifies the usual embedding of the grid into the hypercube such that processors with adjacent NEWS coordinates are in fact neighbors within the hypercube. The value :send-order specifies that, if processor A has a smaller NEWS coordinate than processor B, then A also has a smaller send-address than B. This ordering is rarely used. However, :send-order ordering *is* useful for specific applications such as FFT. The value :framebuffer-order is provided solely for creating VP sets that are used as image buffers (for details, see chapter 1 of the *Generic Display Interface Reference Manual*).

If the "weight" components are all 1, then the mapping of virtual to physical processors can be specified with the "on-chip-bits" and "off-chip-bits" components of the axis descriptors. This is not recommended. To tune performance for communication, use the weight component.

CREATE-GEOMETRY

Creates a new geometry given the grid axis lengths. See also CM:intern-geometry.

Formats result ← CM:create-geometry *dimension-array*, *[rank]*

Operands *dimension-array* A front-end vector of unsigned integer lengths of the grid axes. In the Lisp interface, this may be a list of dimension lengths instead of an array of dimension lengths, at the user's option.

rank An unsigned integer, the rank (number of dimensions) of the *dimension-array*. This must be between 1 and CM:*max-geometry-rank*, inclusive. This argument is not provided when calling Paris from Lisp.

Result A geometry ID, identifying the newly created geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

The *dimension-array* must be a one-dimensional array of nonnegative integers; each must be a power of 2. The product of all these integers must be a multiple of the number of physical processors attached for use by this process.

This operation returns a geometry ID for a newly created geometry whose dimensions are specified by the *dimension-array*. The length of axis j of the resulting geometry will be equal to *dimension-array*[j]. Such a geometry ID may then be used to create a VP set, or to respecify the geometry of an existing VP set.

The geometry will be laid out so as to optimize performance under the assumption that the axes are used equally frequently for NEWS communication. The operation CM:create-detailed-geometry may be used instead to get more precise control over layout for performance tuning.

Once the geometry has been created, the user may destroy the array used to provide the dimension information. All necessary information is copied out of this array as the geometry is created.

CROSS-VP-MOVE

Copies data from a source field with a particular shape and orientation to a destination field with the same shape, but possibly with a different orientation within the CM. The source and destination VP sets are not required to have matching dimensionality along all axes. However, every source axis selected for inclusion in this copying operation must be mapped to a destination axis of the same length. The source field must be in the current VP set; the destination field may be in a different VP set.

Formats

CM:cross-vp-move-1L	*dest*, *source*, *axis-mapping*, *source-axis-coords*, *dest-axis-coords*, *len*
CM:cross-vp-move-always-1L	*dest*, *source*, *axis-mapping*, *source-axis-coords*, *dest-axis-coords*, *len*

Operands

dest The field ID of the dest field. This is in the destination VP set.

source The field ID of the source field. This is in the current VP set.

axis-mapping A front-end vector of unsigned integer values. The set of valid values also includes the null value CM:*cvpm-indexed*.

This vector defines how the source axes are mapped to the destination axes during data transfer. The length of this vector is equal to the number of axes in the source VP set. Thus, axis-mapping element 0 corresponds to *source* axis 0, and so forth. The value of each vector element should indicate to which destination axis the corresponding source axis is mapped.

For any source axis that is *not* to be copied, give the corresponding axis-mapping element the value CM:*cvpm-indexed*; treatment of such axes is further specified by the next argument.

source-axis-coords A front-end vector of unsigned integer values.. The set of valid values also includes the null value CM:*cvpm-mapped*.

This vector defines what *source* data is copied by the operation. The length of this vector is equal to the number of axes in the source VP set. Thus, source-axis-coords element 0 corresponds to *source* axis 0, and so forth. Any source axis that is mapped in the axis-mapping vector should have a source-axis-coords value of CM:*cvpm-mapped*; the shape of the data to be copied is described by these mapped axes.

The remaining, unmapped, source-axis-coords elements should be integers, each of which indexes a specific point along its corresponding *source* axis; these coordinates describe the location of the source data to be copied.

dest-axis-coords A front-end vector of unsigned integer values.. The set of valid values also includes the null value CM:*cvpm-mapped*.

This vector defines where within the destination VP set the *source* data is transferred. The length of this vector is equal to the number of axes in the destination VP set. Thus, dest-axis-coords element 0 corresponds to *dest* axis 0, and so forth. Any destination axis that is mapped in the axis-mapping vector should have a dest-axis-coords value of CM:*cvpm-mapped*; the final orientation of the copied data is described by these mapped axes.

The remaining, unmapped, dest-axis-coords elements should be integers, each of which indexes a specific point along its corresponding *dest* axis; these coordinates describe the final location of the copied data.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap For d, e, s, and t, the fields *s*, *o*, *u*, *r*, *c*, and *e* must be either nonoverlapping or identical.

Context This operation is conditional.

Data values of *len* bits each are copied from the *source* field into the *dest* field, where the *source* field is in the current VP set and the *dest* field may be in the same or a different VP set. During this operation, the copied data is *moved* from one orientation within the Connection Machine – dictated by the layout of the participating *source* axes – into another orientation dictated by the layout of the participating *dest* axes.

The three vector arguments determine *what* source data is copied, *where* within the destination geometry it is put, and *how* it is moved or reoriented within the CM during this process.

The *source-axis-coords* vector specifies *what* source data is copied. It contains one element for each source geometry axis such that element 0 corresponds to axis 0, and so forth. It is not necessary to copy all the source data: along each axis, either one point or all points may be included in the shape that is copied. For example, to copy a 2-dimensional shape from a 3-dimensional geometry, we include two entire axes and one point along the third axis.

To include all the data along a particular source axis, specify the corresponding *source-axis-coords* value as CM:*cvpm-mapped* – meaning this axis is mapped in its entirety to some destination axis. The shape of the source data to copy is defined by the lengths of the axes specified as mapped. The exact mapping is given by the *axis-mapping* vector. To include only one point along a particular source axis, specify the corresponding *source-axis-coords* value as an unsigned integer between 0 and one less than the extent of the axis.

The *dest-axis-coords* vector specifies *where* in the destination to put the source data. This vector is analogous to *source-axis-coords* in that it specifies which destination axes recieve data and where along the remaining axes the copying is carried out. There must be one *dest-axis-coords* element for each destination geometry axis and each element value must be either an integer or CM:*cvpm-mapped*.

To transfer data to an entire axis, specify the corresponding *dest-axis-coords* value as CM:*cvpm-mapped*. To transfer data only at a specific coordinate along an axis, specify an integer value. In *dest-axis-coords* and *source-axis-coords*, the number and lengths of the axes specifed as mapped must exactly match. For example, when copying a 2-dimensional shape from a 3-dimensional VP set into a 2-dimensional VP set, the *source-axis-coords* will include two mapped axes and one coordinate while the *dest-axis-coords* will include two mapped axes and no coordinates.

The *axis-mapping* vector specifies how the copied data is reoriented as it is transferred from the source geometry to the destination geometry. As discribed above, the *source-axis-coords* and *dest-axis-coords* vectors each specify certain *source* and *dest* axes as "mapped." The *axis-mapping* vector determines which source axis is mapped to which destination axis. It contains one element for each source geometry axis such that element 0 corresponds to source axis 0 and so forth. Each element value is either an integer or CM:*cvpm-indexed*.

For each source axis that is *not* mapped to a destination axis, give the corresponding *axis-mapping* element the value CM:*cvpm-indexed* – meaning that this axis is indexed. The *source-axis-coords* vector gives coordinates from which data along an indexed axis is copied. For each source axis that is mapped to a destination axis, give the corresponding *axis-mapping* element an unsigned integer value indicating which destination axis is to recieve data from this source axis. Each pair of mapped axes must be of the same length.

Note: Proper execution of this instruction requires that the lengths of the source and destination axes not be changed between invocations. Be especially careful if a CM:set-vp-set-geometry call changes the geometry of either the source or destination VP set between invocation of CM:cross-vp-set-move-1L.

The code fragment below demonstrates copying a 2-dimensional shape from a 3-dimensional source geometry into a 2-dimensional destination geometry. Source axes 0 and 1 are copied from coordinate i along source axis 2. Source axis 0 maps to destination axis 1 and source axis 1 maps to destination axis 0.

DEALLOCATE-GEOMETRY

Declare that a geometry will no longer be used.

Formats CM:deallocate-geometry *geometry-id*

Operands *geometry-id* A geometry ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

By this operation a user program declares that a geometry will no longer be used. The system is permitted to reclaim any and all resources associated with that geometry. It is an error for the user program to give the specified geometry ID as an argument to any Paris operation once it has been deallocated.

It is an error to deallocate a geometry that is still in use by some VP set.

DEALLOCATE-HEAP-FIELD

Declare that a heap field will no longer be used.

Formats CM:deallocate-heap-field *heap-field-id*

Operands *heap-field-id* A field ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

By this operation a user program declares that a field will no longer be used. The system is permitted to reclaim any and all resources associated with that field, in particular the memory that it occupied. It is an error for the user program to give the specified field ID as an argument to any Paris operation once it has been deallocated.

DEALLOCATE-STACK-THROUGH

Declare that a stack field and all fields allocated more recently than it will no longer be used.

Formats CM:deallocate-stack-through *stack-field-id*

Operands *stack-field-id* A field ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

By this operation a user program declares that the specified field on the stack, and all fields allocated more recently than it, will no longer be used. (Note that any fields allocated more recently than the specified field are necessarily closer to the top of the stack.) The system is permitted to reclaim any and all resources associated with those fields, in particular the memory that they occupied. It is an error for the user program to give the field ID of a deallocated field as an argument to any Paris operation.

DEALLOCATE-VP-SET

Declare that a VP set will no longer be used.

Formats CM:deallocate-vp-set *vp-set-id*

Operands *vp-set-id* A VP set ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

By this operation a user program declares that a VP set will no longer be used. The system is permitted to reclaim any and all resources associated with that VP set. It is an error for the user program to give the specified VP set ID as an argument to any Paris operation once it has been deallocated.

It is an error to deallocate a VP set for which there are still fields that have not yet been deallocated. The user should first deallocate all fields belonging to that VP set, except the flags, which are deallocated automatically when the VP set is deallocated.

DEPOSIT-NEWS-COORDINATE

Modifies a send address to reflect a specific NEWS coordinate.

Formats

CM:deposit-news-coordinate-1L *geometry, dest/send-address, axis, coordinate, slen*

CM:deposit-news-constant-1L *geometry, dest/send-address, axis, coordinate-value, slen*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

dest The field ID of the unsigned integer destination field. (In the instruction formats currently provided, the *dest* field is always the same as the *send-address* source field. The length of this field is implicitly the same as *geometry-send-address-length*(*geometry*).)

send-address The field ID of the unsigned integer send address field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

coordinate The field ID of the unsigned integer NEWS coordinate. field. This specifies the position along the corrsponding axis of the processor whose send address is to be calculated.

coordinate-value An unsigned integer immediate operand to be used as the NEWS coordinate along the specified axis.

slen The length of the *coordinate* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap For CM:deposit-news-coordinate-1L, the *coordinate* field must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \textit{deposit-news-coordinate}(\textit{geometry}, \textit{send-address}, \textit{axis}, \textit{coordinate})$

where *deposit-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates equal to those for the processor identified by *send-address*.

FE-DEPOSIT-NEWS-COORDINATE

Calculates on the front end the modification of a send address to reflect a specific NEWS coordinate.

Formats result ← CM:fe-deposit-news-coordinate *geometry*, *send-address*, *axis*, *coordinate*

Operands *geometry* A geometry ID. This geometry determines the NEWS dimensions to be used.

send-address An unsigned integer immediate operand to be used as the send address of some processor.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along the specified axis.

Result An unsigned integer, the send address of the processor whose coordinate along the specified axis is *coordinate* and whose coordinate along all other axes equals those of *send-address.*

Context This operation is performed on the front end. It does not depend on the CM *context-flag.*

Definition Return *deposit-news-coordinate*(*geometry*, *send-address*, *axis*, *coordinate*) where *deposit-news-coordinate* is as defined on page 40.

This function calculates, entirely on the front end, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates equal to those for the processor identified by *send-address.*

DETACH

Detaches the specified front-end computer from the Connection Machine hardware previously allocated for and attached to it.

This instruction is available only from the Lisp/Paris interface. For Fortran/Paris and C/Paris users, the equivalent functionality is provided by the shell level cmdetach command, documented in the *CM System User's Guide*.

Formats CM:detach *front-end-name*, *suppress-confirmation*

Operands *front-end-name* The name of a front end, or a list of a front end name and a bus-interface specifier. This argument is optional.

suppress-confirmation The confirmation suppression flag. This argument is optional. If supplied and not false, then the interactive query and prompt requesting confirmation of the detach operation is suppressed.

Context This operation is unconditional. It does not depend on the *context-flag*.

The facility for detaching Connection Machine hardware is provided in different ways in the Lisp/Paris interface (on the one hand) and the C/Paris and Fortran/Paris interfaces (on the other hand).

In the Lisp/Paris interface, CM:detach is a function of two arguments. The arguments are optional.

In most normal use no argument is specified. In this case the front end executing the call to CM:detach releases all Connection Machine hardware to which it had been attached, resetting relevant parts of the Nexus so that the front end can no longer issue macroinstructions to the Connection Machine system. (An error is signalled if in fact no hardware had been attached in the first place.) This use of CM:detach is the normal way of releasing attached hardware and will not disrupt users on other front ends.

If a *front-end-name* argument is specified, it must be the name of a front end that is connected to the same Connection Machine system (that is, Nexus) as the front end executing the call, or perhaps a list of a front end name and a small integer identifying a bus interface on that front end. A front end name may be either a string or a symbol. Examples (assuming, for the sake of exposition, that front end computers are named after Shakespearean characters):

```
(detach 'hamlet)          ;Detach front end named Hamlet
```

```
(detach "lear" t)          ;Detach front end named Lear, and don't confirm
(detach '(desdemona 1))   ;Detach bus interface 1 of front end Desdemona
```

Specifying the name of the front end that is executing the call has the same effect as specifying no argument; the front end is gracefully detached. But specifying the name of some other front end forcibly detaches that other front end, possibly disrupting any ongoing interaction with the Connection Machine system. The external communications network is used to send a message to the detached front end to inform its user that it has been forcibly detached.

There are two sets of initialization forms, kept in the variables CM:*before-detach-initializations* and CM:*after-detach-initializations*, that are evaluated before and after anything else occurs.

In the C/Paris and Fortran/Paris interfaces, the detaching operation is performed by a user command cmdetach at shell level. See the *Front End Subsystems* manual or the cmdetach man page.

C-DIVIDE

The quotient of two complex source values is placed in the destination field. **Note:** Integer division is performed by the round, truncate, rem, and mod operations.

Formats

CM:c-divide-2-1L	*dest/source1, source2, s, e*
CM:c-divide-always-2-1L	*dest/source1, source2, s, e*
CM:c-divide-3-1L	*dest, source1, source2, s, e*
CM:c-divide-always-3-1L	*dest, source1, source2, s, e*
CM:c-divide-constant-2-1L	*dest/source1, source2-value, s, e*
CM:c-divide-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:c-divide-constant-3-1L	*dest, source1, source2-value, s, e*
CM:c-divide-const-always-3-1L	*dest, source1, source2-value, s, e*
CM:c-divinto-2-1L	*dest/source2, source1, s, e*
CM:c-divinto-always-2-1L	*dest/source2, source1, s, e*
CM:c-divinto-constant-2-1L	*dest/source2, source1-value, s, e*
CM:c-divinto-const-always-2-1L	*dest/source2, source1-value, s, e*
CM:c-divinto-constant-3-1L	*dest, source2, source1-value, s, e*
CM:c-divinto-const-always-3-1L	*dest, source2, source1-value, s, e*

Operands

dest The field ID of the complex destination field. This is the quotient.

source1 The field ID of the complex first source field. This is the dividend.

source2 The field ID of the complex second source field. This is the divisor.

source1-value A complex immediate operand to be used as the first source.

source2-value A complex immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if division by zero occurs; otherwise it is unaffected.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if (always or *context-flag*$[k] = 1$) then
dest$[k] \leftarrow$ *source1*$[k]/$*source2*$[k]$
if *source2*$[k] = 0$ then *test-flag*$[k] \leftarrow 1$
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The *source1* operand is divided by the *source2* operand, treating both as complex numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-DIVIDE

The quotient of two floating-point source values is placed in the destination field.

Note: Integer division is performed by the round, truncate, rem, and mod operations.

Formats

CM:f-divide-2-1L	*dest/source1, source2, s, e*
CM:f-divide-always-2-1L	*dest/source1, source2, s, e*
CM:f-divide-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-divide-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:f-divinto-2-1L	*dest/source2, source1, s, e*
CM:f-divinto-always-2-1L	*dest/source2, source1, s, e*
CM:f-divinto-constant-2-1L	*dest/source2, source1-value, s, e*
CM:f-divinto-const-always-2-1L	*dest/source2, source1-value, s, e*
CM:f-divide-3-1L	*dest, source1, source2, s, e*
CM:f-divide-always-3-1L	*dest, source1, source2, s, e*
CM:f-divide-constant-3-1L	*dest, source1, source2-value, s, e*
CM:f-divide-const-always-3-1L	*dest, source1, source2-value, s, e*
CM:f-divinto-constant-3-1L	*dest, source2, source1-value, s, e*
CM:f-divinto-const-always-3-1L	*dest, source2, source1-value, s, e*

Operands

dest The field ID of the floating-point destination field. This is the quotient.

source1 The field ID of the floating-point first source field. This is the dividend.

source2 The field ID of the floating-point second source field. This is the divisor.

source1-value A floating-point immediate operand to be used as the first source.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if division by zero occurs; otherwise it is unaffected.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flags may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\mathit{context\text{-}flag}[k] = 1$) then
    $\mathit{dest}[k] \leftarrow \mathit{source1}[k]/\mathit{source2}[k]$
    if $\mathit{source2}[k] = 0$ then $\mathit{test\text{-}flag} \leftarrow 1$
    if $\langle$overflow occurred in processor $k\rangle$ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$

The *source1* operand is divided by the *source2* operand, treating both as floating-point numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

ENUMERATE

The destination field in every selected processor receives the number of processors below or above it in some ordering of the processors.

Formats CM:enumerate-1L *dest, axis, len, direction, inclusion, smode, sbit*

Operands

dest	The field ID of the unsigned integer destination field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The *sbit* field must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $context\text{-}flag[k] = 1$ then
  let $S_k = scan\text{-}subset(k, axis, len, direction, inclusion, smode, sbit)$
  $dest[k] \leftarrow |S_k|$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:enumerate-1L operation stores into the *dest* field of each selected processor the size of the scan subset for that processor. This means that every processor within a scan set of size N will receive a different integer in the range 0 to $N-1$ (for an exclusive enumeration) or in the range 1 to N (for an inclusive enumeration).

A call to CM:enumerate-1L is equivalent to the sequence below, but may be faster.

CM:u-move-constant-1L *temp*, 1, *len*
CM:scan-with-u-add-1L *dest*, *temp*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*
CM:u-subtract-constant-1L *dest*, 1, *len*

C-EQ

Compares two complex source values. The *test-flag* is set if they are equal, and otherwise it is cleared.

Formats

CM:c-eq-1L	*source1, source2, s, e*
CM:c-eq-constant-1L	*source1, source2-value, s, e*
CM:c-eq-zero-1L	*source1, s, e*

Operands

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source. For CM:c-eq-zero-1L, this implicitly has the value zero.

s, e The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k]$ = *source2*$[k]$
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as complex numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-EQ

Compares two floating-point source values. The *test-flag* is set if they are equal, and otherwise is cleared.

Formats

CM:f-eq-1L	*source1*, *source2*, *s*, *e*
CM:f-eq-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-eq-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-eq-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] = *source2*[k]
      test-flag[k] $\leftarrow$ 1
    else
      test-flag[k] $\leftarrow$ 0

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-EQ

Compares two signed integer source values. The *test-flag* is set if they are equal, and otherwise is cleared.

Formats

CM:s-eq-1L	*source1*, *source2*, *len*
CM:s-eq-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-eq-constant-1L	*source1*, *source2-value*, *len*
CM:s-eq-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-eq-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k]$ = *source2*$[k]$ then
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-EQ

Compares two unsigned integer source values. The *test-flag* is set if they are equal, and otherwise is cleared.

Formats

CM:u-eq-1L	*source1*, *source2*, *len*
CM:u-eq-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-eq-constant-1L	*source1*, *source2-value*, *len*
CM:u-eq-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-eq-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source1}[k] = \textit{source2}[k]$ then
      $\textit{test-flag}[k] \leftarrow 1$
    else
      $\textit{test-flag}[k] \leftarrow 0$

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-EXP

The exponent of the complex source field is stored in the complex destination field.

Formats CM:c-exp-1-1L *dest/source*, *s*, *e*
CM:c-exp-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \exp$ *source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The value e^s is stored into the *dest* field, where s is the value of the *source* field, and e is the base of the natural logarithms; $e \approx 2.718281828\ldots$

F-EXP

Calculates, in each selected processor, the exponential function e^x of the floating-point source field and stores it in the floating-point destination field.

Formats CM:f-exp-1-1L *dest/source*, *s*, *e*
CM:f-exp-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source*$[k] = +\infty$ then
      dest$[k] \leftarrow +\infty$
    else if *source*$[k] = -\infty$ then
      dest$[k] \leftarrow +0$
    else
      dest$[k] \leftarrow \exp$ *source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

Call the value of the *source* field s; the value e^s is stored into the *dest* field, where $e \approx 2.718281828\ldots$ is the base of the natural logarithms.

FE-EXTRACT-MULTI-COORDINATE

Calculates, on the front end, the NEWS multi-coordinate of a processor specified by send-address. A multi-coordinate is needed in order to use the CM:multispread-copy-1L instruction.

Formats result ← CM:fe-extract-multi-coordinate *geometry*, *axis-mask*, *send-address*

Operands *geometry* A geometry ID. This geometry determines the NEWS dimensions to be used.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

send-address An unsigned integer immediate operand to be used as the send address of some processor.

Result An unsigned integer, the NEWS multi-coordinate of the specified processor along the specified axes.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Let $\textit{axis-set} = \{ m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \}$
Return *extract-multi-coordinate*(*geometry*, *axis-set*, *send-address*)
where *extract-multi-coordinate* is as defined on page 44.

This function calculates, entirely on the front end, the NEWS multi-coordinate of a processor along specified NEWS axes. The axes are indicated by the *axis-mask* argument; the processor is identified by its send-address.

EXTRACT-NEWS-COORDINATE

Determines the NEWS coordinate of a processor specified by send-address.

Formats CM:extract-news-coordinate-1L *geometry*, *dest*, *axis*, *send-address*, *dlen*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

dest The field ID of the unsigned integer destination field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

send-address The field ID of the send address field. For each processor, this identifies the send address of some other processor.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow extract\text{-}news\text{-}coordinate(geometry, axis, send\text{-}address)$

where *extract-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the NEWS coordinate of a processor along a specified NEWS axis. The axis is indicated by the *axis* argument; the processor is identified by its send-address.

FE-EXTRACT-NEWS-COORDINATE

Calculates, on the front end, the NEWS coordinate of a processor specified by send-address.

Formats result ← CM:fe-extract-news-coordinate *geometry, axis, send-address*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

send-address An unsigned integer immediate operand to be used as the send address of some processor.

Result An unsigned integer, the NEWS coordinate of the specified processor along the specified axis.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Return *extract-news-coordinate*(*geometry*, *axis*, *send-address*)

where *extract-news-coordinate* is as defined on page 40.

This function calculates, entirely on the front end, the NEWS coordinate of a processor along a specified NEWS axis. The axis is indicated by the *axis* argument; the processor is identified by its send-address.

DEALLOCATE-FFT-SETUP

Deallocates a front-end setup descriptor that has been used to prepare information for execution of an FFT routine.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. A more efficient set of FFT routines are included in the CM Scientific Subroutines Library.

Formats CMSSL:deallocate-fft-setup *setup-id*

Operands *setup-id* The ID of the FFT setup descriptor to be deallocated.

Context This is a front-end operation. It does not depend on the value of the *context-flag*.

This routine may be used to remove an FFT setup descriptor when it is no longer needed. The setup-id argument must have been obtained by a call to CMSSL:c-fft-setup.

An fft setup descriptor occupies memory both on the front end and on the Connection Machine. It is therefore wise to free this space by calling CMSSL:deallocate-fft-setup after completion of all FFT routines that use the specified setup descriptor.

C-C-FFT

The Discrete Fourier Transform of the complex source field is calculated using a Fast Fourier Transform (FFT) algorithm. The complex result is stored in the destination field.

A Fourier transform routine converts (possibly multidimensional) sequences between the time or space domain and the frequency domain. This type of transform has a variety of useful applications. For example, an FFT can be used to filter discrete signals, to smooth input data or output images, to interpolate or extrapolate from a given data set, to measure the correlation between two samples, or to multiply polynomials and extremely large integers.

The Fast Fourier Transform is called a fast transform because it exhibits $O(N \log N)$ complexity, where O is the order of complexity and N is the length of the input sequence. By comparison, the Discrete Fourier Transform exhibits only $O(N^2)$ complexity.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix c-c- to signify that single-precision complex operands are involved. A more efficient set of FFT routines are included in the CM Scientific Subroutines Library.

Formats CMSSL:c-c-fft *dest, source, setup, ops, source-bit-order, dest-bit-order, source-cm-order, dest-cm-order, scale*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

setup The setup-id. This must be a setup ID returned by CMSSL:c-fft-setup. The geometry information of the setup must be identical to that of the source and destination fields.

ops A front-end vector of operation identifiers. Each element specifies whether the corresponding source axis is transformed and, if so, by what method. Valid vector element values are :f-xform (FFT_f_xform in C; 1 in Fortran) for a forward transform, :i-xform (FFT_i_xfrom in C; 2 in Fortran) for an inverse transform, and :nop (FFT_nop in C; 0 in Fortran) for no transform.

source-bit-order A front-end vector of input bit orderings. Each element identifies the bit ordering of the corresponding source axis and must be either :normal or :bit-reversed. (The corresponding values are are FFT_normal and FFT_bit_reversed in C, and 0 and 1 in Fortran, respectively.)

dest-bit-order A front-end vector of output bit orderings. Each element identifies the bit ordering of the corresponding destination axis and must be either :normal or :bit-reversed. (The corresponding values are are FFT_normal and FFT_bit_reversed in C, and 0 and 1 in Fortran, respectively.)

source-cm-order A front-end vector of input orderings. Each element declares the addressing mode of the corresponding source axis and must be one of the following: :send-order, :news-order, or :default. (The corresponding values are FFT_send_order, FFT_news_order, and FFT_default in C, and 1, 2, and 0 in Fortran, respectively.)

A value of :default causes the current ordering of an axis to be used.

dest-cm-order A front-end vector of output orderings. Each element declares the addressing mode of the corresponding destination axis and must be one of the following: :send-order, :news-order, or :default. (The corresponding values are FFT_send_order, FFT_news_order, and FFT_default in C, and 1, 2, and 0 in Fortran, respectively.)

A value of :default causes the current ordering of an axis to be used.

scale A front-end vector of output scaling methods. Each element specifies whether the corresponding destination axis is rescaled and, if so, by what method. Valid values are :noscale for no rescaling, :scale-sqrt for scaling by the inverse square root of the FFT, and :scale-n for scaling by the inverse of the size of the FFT. (The corresponding values are FFT_noscale, FFT_scale_sqrt, and FFT_scale_n in C, and 0, 1, and 2 in Fortran, respectively.)

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. FFT performance is slightly better if the two fields are identical.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

$$dest[k] \leftarrow FFT(source[k])$$

The Discrete Fourier Transform of the *source* field is stored in the *dest* field. A multidimensional transform is computed by performing the transform across each dimension in sequence.

The source and destination fields must either belong to the same VP set or to VP sets of identical shape and size.

The *ops*, *source-bit-order*, *dest-bit-order*, *source-cm-order*, *dest-cm-order*, and *scale* arguments are one-dimensional front-end arrays. The length of each is equal to the rank of the setup geometry.

By convention, a Fast Fourier Transform operation reverses the order of the data bits when storing the result in the destination. The vectors *source-bit-order* and *dest-bit-order* specify whether the source and destination data are treated as normal or as bit-reversed.

Along any given dimension of the data's geometry, the Connection Machine FFT instruction is most efficient for data arranged in send order. Many FFT applications do not depend on the order of the data elements. The *dest-cm-order* and *source-cm-order* arguments are therefore provided to permit the most efficient execution possible along each dimension.

C/Paris code that calls the Paris FFT routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris FFT.

Fortran/Paris code should include the line

```
  INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris FFT.

C-FFT-SETUP

Allocates a front-end setup descriptor for use with the CMSSL:fft Fast Fourier Transform routines and returns a setup ID.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix c- to signify that single-precision complex operands are involved. A more efficient set of FFT routines are included in the CM Scientific Subroutines Library.

Formats result ← CMSSL:c-fft-setup *geometry-id*

Operands	*geometry*	A geometry ID.
Result	The ID of the newly created FFT setup descriptor.	
Context	This is a front-end operation. It does not depend on the value of the *context-flag*.	

This routine computes information needed to perform a Fast Fourier Transform (FFT), stores it in an FFT setup descriptor, and return the setup-id.

In Lisp/Paris, a setup ID is a structure of type CMSSL:fft-setup. In C/Paris, it is a pointer to a structure of type FFT_fft_setup_t. In Fortran/Paris it is an integer.

The *geometry* argument must be a geometry ID returned by a call to CM:create-geometry, CM:create-detailed-geometry, intern-geometry, or intern-detailed-geometry.

The returned setup ID is a valid value for the *setup* argument to any CMSSL FFT routine if the following requirement is obeyed. The geometries of the FFT source and destination fields must be identical to that of the setup geometry.

This routine must be reinvoked whenever the geometry of an FFT source field VP set is changed. CMSSL:c-fft-setup allocates memory both on the front end and on the CM. To free this memory, use CMSSL:deallocate-fft-setup.

C/Paris code that calls the Paris FFT routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris FFT.

Fortran/Paris code should include the line

```
INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris FFT.

FIELD-VP-SET

Returns the VP set associated with a field.

Formats result ← CM:field-vp-set *field*

Operands *field* The field ID of the field.

Result A VP set ID, identifying the VP set to which the field belongs.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *vp-set*(*field*)

This operation may be used to determine the VP set with which any given field is associated. The field need not belong to the current VP set.

F-S-FLOAT

Converts a signed integer field into a floating-point number field.

Formats CM:f-s-float-2-2L *dest*, *source*, *slen*, *s*, *e*

Operands	*dest*	The field ID of the floating-point destination field.
	source	The field ID of the signed integer source field.
	slen	The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
	s, *e*	The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Overlap: The fields *dest* and *source* must not overlap in any manner.

Flags: *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow$ *source*$[k]$
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The *source* field, treated as a signed integer, is converted to a floating-point number, which is stored into the *dest* field.

F-U-FLOAT

Converts an unsigned integer field into a floating-point number field.

Formats CM:f-u-float-2-2L *dest*, *source*, *slen*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the unsigned integer source field.
slen	The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
s, *e*	The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow$ *source*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The *source* field, treated as an unsigned integer, is converted to a floating-point number, which is stored into the *dest* field.

F-F-FLOOR

In each selected processor, calculates the largest integer that is not greater than a specified floating-point value and stores the result as a floating-point field.

Formats CM:f-f-floor-1-1L *dest/source*, *s*, *e*
CM:f-f-floor-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
$\mathit{dest}[k] \leftarrow \lfloor \mathit{source}[k] \rfloor$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $-\infty$, which is stored into the *dest* field as a floating-point number.

Note that overflow cannot occur.

S-FLOOR

The floor of the quotient of two signed integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:s-floor-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-floor-2-1L	*dest*/*source1*, *source2*, *len*
CM:s-floor-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-floor-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:s-floor-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the signed integer quotient field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-floor-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-floor-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-floor-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*[k] = 1 then
$dest[k] \leftarrow \left\lfloor \dfrac{source1[k]}{source2[k]} \right\rfloor$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
else *overflow-flag*[k] ← 0
if *source2*[k] = 0 then
test[k] ← 1
else *test*[k] ← 0

The signed integer *source1* operand is divided by the signed integer *source2* operand. The floor of the mathematical quotient is stored into the signed integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

S-F-FLOOR

Calculates, in each selected processsor, the largest integer that is not greater than a specified floating-point value and stores the result as a signed integer field.

Formats

CM:s-f-floor-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the floating-point source field.
len	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \lfloor \textit{source}[k] \rfloor$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$
    else $\textit{overflow-flag}[k] \leftarrow 0$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $-\infty$, which is stored into the *dest* field as a signed integer.

U-FLOOR

The floor of the quotient of two unsigned integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:u-floor-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-floor-2-1L	*dest/source1, source2, len*
CM:u-floor-3-1L	*dest, source1, source2, len*
CM:u-floor-constant-2-1L	*dest/source1, source2-value, len*
CM:u-floor-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

dlen For CM:s-floor-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:s-floor-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:s-floor-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then

$$dest[k] \leftarrow \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$

> if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] $\leftarrow 1$
>   else *overflow-flag*[k] $\leftarrow 0$
> if *source2*[k] $= 0$ then
>   *test*[k] $\leftarrow 1$
> else *test*[k] $\leftarrow 0$

The unsigned integer *source1* operand is divided by the unsigned integer *source2* operand. The floor of the mathematical quotient is stored into the unsigned integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-F-FLOOR

Converts floating-point source field values into unsigned integers by rounding towards $-\infty$.

Formats CM:u-f-floor-2-2L *dest, source, dlen, s, e*

Operands	*dest*	The field ID of the unsigned integer destination field.
	source	The field ID of the floating-point source field.
	len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
	s, e	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.
Overlap		The fields *dest* and *source* must not overlap in any manner.
Flags		*overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.
Context		This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    $dest \leftarrow \lfloor source \rfloor$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $-\infty$. The result is stored into the *dest* field as an unsigned integer.

FE-FROM-GRAY-CODE

Calculates, on the front end, the Gray code representation of a specified integer.

Formats result ← CM:fe-from-gray-code *code*

Operands *code* An unsigned integer immediate operand to be used as the Gray encoding, represented as a nonnegative integer.

Result An unsigned integer, the nonnegative integer represented by *code*.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = \mathit{integer\text{-}length}(\mathit{code})$

Return $\bigoplus_{j=0}^{n-1} \left\lfloor \frac{\mathit{code}}{2^j} \right\rfloor$

This function calculates, entirely on the front end, the integer represented by a bit-string encoding *code* in a particular reflected binary Gray code.

Note that the binary value 0 is always equivalent to a Gray code string that is all 0-bits.

U-FROM-GRAY-CODE

Converts a bit string representing a Gray-coded integer value to the usual unsigned binary representation.

Formats CM:u-from-gray-code-1-1L *dest/source, len*
CM:u-from-gray-code-2-1L *dest, source, len*

Operands *dest* The field ID of the unsigned integer destination field.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
for j from $len - 1$ to 0 do

$$dest[k]\langle j\rangle \leftarrow \left(\bigoplus_{i=j}^{len-1} source[k]\langle i\rangle\right)$$

The *source* operand is considered to be a value in a particular reflected binary Gray code. The position of that value in the standard Gray code sequence is calculated as an unsigned binary integer. This is done as follows: bit i of the result is 1 if and only if all the bit positions of the source to the left of (and including) bit i contain an odd number of 1's.

Note that a Gray code string that is all 0-bits is always equivalent to the binary value 0.

F-GE

Compares two floating-point source values. The *test-flag* is set if the first is greater than or equal to the second, and otherwise is cleared.

Formats

CM:f-ge-1L	*source1*, *source2*, *s*, *e*
CM:f-ge-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-ge-zero-1L	*source1*, *s*, *e*

Operands

source1 — The field ID of the floating-point first source field.

source2 — The field ID of the floating-point second source field.

source2-value — A floating-point immediate operand to be used as the second source. For CM:f-ge-zero-1L, this implicitly has the value zero.

s, *e* — The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
    if *context-flag*$[k] = 1$ then
        if *source1*$[k] \geq$ *source2*$[k]$
            test-flag$[k] \leftarrow 1$
        else
            test-flag$[k] \leftarrow 0$

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than or equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-GE

Compares two signed integer source values. The *test-flag* is set if the first is greater than or equal to the second, and otherwise is cleared.

Formats

CM:s-ge-1L	*source1*, *source2*, *len*
CM:s-ge-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-ge-constant-1L	*source1*, *source2-value*, *len*
CM:s-ge-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-ge-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $source1[k] \geq source2[k]$ then
      $test\text{-}flag[k] \leftarrow 1$
    else
      $test\text{-}flag[k] \leftarrow 0$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-GE

Compares two unsigned integer source values. The *test-flag* is set if the first is greater than or equal to the second, and otherwise is cleared.

Formats

CM:u-ge-1L	*source1*, *source2*, *len*
CM:u-ge-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-ge-constant-1L	*source1*, *source2-value*, *len*
CM:u-ge-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-ge-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \geq$ *source2*$[k]$ then
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

GEOMETRY-AXIS-LENGTH

Returns the length of one axis of a geometry.

Formats result ← CM:geometry-axis-length *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose length is desired.

Result An unsigned integer, the length of the indicated axis.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *axis-descriptors*(*geometry-id*)[*axis*].*length*

This operation returns the length of the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-AXIS-OFF-CHIP-BITS

Returns the number of off-chip bits that are allocated for the specified NEWS axis within the off-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-off-chip-bits *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose off-chip bits count is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the count of the off-chip bits associated with the specified *axis*. If *axis* has no off-chip bits, the result is 0.

Context This operation is unconditional. It does not depend on the *context-flag*.

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates how many of the off-chip bits within the off-chip bits partition are used in the send addresses of virtual processors that lie along the specified dimension.

Note that the integer returned does not indicate the total number of all off-chip bits within the send address but the number of off-chip bits used for a particular dimension.

GEOMETRY-AXIS-OFF-CHIP-POS

Returns the starting position for the off-chip bits that are allocated for the specified NEWS axis within the off-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-off-chip-pos *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose off-chip bits position is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the location in the send address of the first off-chip bit associated with the specified axis. This is zero-based; the first location is numbered 0.

Context This operation is unconditional. It does not depend on the *context-flag.*

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates where, within the off-chip bits partition, the off-chip bits for the specified dimension lie.

Note that the integer returned does not indicate the absolute position of all off-chip bits within the send address but the position of the off-chip bits for a particular dimension relative to the start of all off-chip bits in an address.

GEOMETRY-AXIS-ON-CHIP-BITS

Returns the number of on-chip bits that are allocated for the specified NEWS axis within the on-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-on-chip-bits *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose on-chip bits count is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the count of the on-chip bits associated with the specified *axis*. If *axis* has no on-chip bits, the result is 0.

Context This operation is unconditional. It does not depend on the *context-flag*.

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates how many of the on-chip bits within the on-chip bits partition are used in the send addresses of virtual processors that lie along the specified dimension.

Note that the integer returned does not indicate the total number of all on-chip bits within the send address but the number of on-chip bits used for a particular dimension.

GEOMETRY-AXIS-ON-CHIP-POS

Returns the starting position for the on-chip bits that are allocated for the specified NEWS axis within the on-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-on-chip-pos *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose on-chip bits position is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the location in the send address of the first on-chip bit associated with the specified axis. This is zero-based; the first location is numbered 0.

Context This operation is unconditional. It does not depend on the *context-flag*.

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates where, within the on-chip bits partition, the on-chip bits for the specified dimension lie.

Note that the integer returned does not indicate the absolute position of all on-chip bits within the send address but the position of the on-chip bits for a particular dimension relative to the start of all on-chip bits in an address.

GEOMETRY-AXIS-ORDERING

Returns the ordering of one axis of a geometry.

Formats result ← CM:geometry-axis-ordering *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose ordering is desired.

Result The ordering of the specified axis (either :news-order or :send-order).

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *axis-descriptors*(*geometry-id*)[*axis*].*ordering*

This operation returns the ordering of the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-AXIS-VP-RATIO

Returns the VP ratio of one axis of a geometry.

Formats result ← CM:geometry-axis-vp-ratio *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose VP ratio is desired.

Result An unsigned integer, the VP ratio of the indicated axis.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *axis-descriptors*(*geometry-id*)[*axis*].*vp-ratio*

This operation returns the VP ratio of the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-COORDINATE-LENGTH

Returns the number of bits needed to represent a NEWS coordinate.

Formats result ← CM:geometry-coordinate-length *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose coordinate length is desired.

Result An unsigned integer, the number of bits required to represent a coordinate for the indicated axis.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *integer-length*(*axis-descriptors*(*geometry-id*)[*axis*].*length* − 1)

This operation returns the number of bits required to represent (as an unsigned integer) a NEWS coordinate for the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-RANK

Returns the number of axes for a geometry.

Formats result ← CM:geometry-rank *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the rank (number of axes) of the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *rank*(*geometry*)

This operation returns the number of grid axes for the geometry specified by the *geometry-id*.

GEOMETRY-SEND-ADDRESS-LENGTH

Returns the number of bits needed to represent a send-address.

Formats result $\leftarrow$ CM:geometry-send-address-length *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the number of bits required to represent a send-address for a processor in the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = rank(geometry\text{-}id)$

Return $\sum_{j=0}^{n-1} integer\text{-}length(axis\text{-}descriptors(geometry\text{-}id)[j].length - 1)$

This operation returns the number of bits required to represent a send-address for a virtual processor in any VP set whose geometry is the one specified by the *geometry-id*. This will be equal to the sum of the numbers of bits needed to represent NEWS coordinates for all the axes.

GEOMETRY-SERIAL-NUMBER

Assigns a unique number to the specified geometry.

Formats result ← CM:geometry-serial-number *geometry-id*

Operands *geometry-id* A geometry ID. This geometry ID must be obtained by calling CM:create-geometry or CM:create-detailed-geometry.

Result The serial number that uniquely identifies the geometry.

Context This operation is unconditional. It does not depend on the *context-flag.*

A unique number, the serial number, is assigned to the specified geometry. This facilitates geometry-based caching; geometry serial numbers are useful as hash table keys.

Note that geometry ID's are not unique identifiers. After a geometry is deallocated, its ID may be reused for another geometry. In contrast, geometry serial numbers are guaranteed to be unique.

GEOMETRY-TOTAL-PROCESSORS

Returns the number of virtual processors for a geometry.

Formats result ← CM:geometry-total-processors *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the total number of processors in the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = rank(geometry\text{-}id)$

Return $\prod_{j=0}^{n-1} axis\text{-}descriptors(geometry\text{-}id)[j].length$

This operation returns the total number of virtual processors in any VP set whose geometry is the one specified by the *geometry-id*. This will be equal to the product of the lengths of all the axes.

GEOMETRY-TOTAL-VP-RATIO

Returns the total VP ratio for a specified geometry.

Formats result ← CM:geometry-total-vp-ratio *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the number of virtual processors represented within each physical processor for the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = rank(geometry\text{-}id)$

Return $\prod_{j=0}^{n-1} axis\text{-}descriptor(geometry\text{-}id)[j].vp\text{-}ratio$

This operation returns the total VP ratio for a specified geometry. This is equal to the total number of virtual processors for the geometry, divided by the total number of physical processors.

GET

Each selected processor gets a message from a specified source processor, possibly itself. A source processor may supply messages even if it is not selected. Messages are all retrieved from the same memory address within each source processor, and all the source processors may be in a VP set different from the VP set of the destination processors.

Formats CM:get-1L *dest*, *send-address*, *source*, *len*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates from which processor a message is retrieved.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

Overlap The *send-address* and *dest* may overlap in any manner. Similarly, the *send-address* and *source* may overlap in any manner. However, it is forbidden for the *dest* and *source* to overlap.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
$\mathit{dest}[k] \leftarrow \mathit{source}[\mathit{send\text{-}address}[k]]$

For every selected processor p_d, a message *length* bits long is sent to p_d from the processor p_s whose send-address is in the field *send-address* in the memory of processor p_d. The message is taken from the *source* field within processor p_s and is stored into the field at location *dest* within processor p_d. Although the *send-address* operand is a field in the VP set of the destination processors, its value must specify a valid send address for *source*, which may belong to a different VP set.

Note that more than one selected processor may request data from the same source processor p_s, in which case the same data is sent to each of the requesting processors.

GET-AREF32

Each selected processor gets a message from a specified array field within any specified source processor (possibly itself). A source processor may supply messages even if it is not selected. Messages are all retrieved from the same memory address within each source processor.

Formats CM:get-aref32-2L *dest*, *send-address*, *array*, *index*, *dlen*, *index-len*, *index-limit*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates from which processor a message is retrieved.

array The field ID of the source array field. This must be stored in the special format required by CM:aref32.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *dlen*.

dlen The length of the *dest* field.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of *array*.

Overlap The *send-address* and *array* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *array* and *dest* to overlap.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{index}[k] < \textit{index-limit}$ then
      let $r = \textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))$
      let $m = \left\lfloor \frac{k}{r} \right\rfloor \bmod 32$
      let $i = \textit{index}[k]$
      for all j such that $0 \leq j < \textit{dlen}$ do
        let $q = \textit{send-address}[k] - m \times r + (j \bmod 32) \times r$

$$\text{let } b = i + \left\lfloor \frac{j}{32} \right\rfloor$$
$$dest[k]\langle j\rangle \leftarrow array[q]\langle b\rangle$$

else

⟨error⟩

For every selected processor p_d, a message *length* bits long is sent to p_d from the processor p_s whose send-address is in the field *send-address* in the memory of processor p_d. The message is taken from the *array* field within processor p_s as if by the operation aref32 and is stored into the field at location *dest* within processor p_d.

Note that more than one selected processor may request data from the same source processor p_s, possibly from different locations within the *array*. Note also that in each case the array element to be sent from processor p_s to processor p_d is determined by the value of *index* within p_d, not the value within p_s.

GET-FROM-NEWS

Each processor gets a message from a specified neighbor processor.

Formats CM:get-from-news-1L *dest, source, axis, direction, len*
CM:get-from-news-always-1L *dest, source, axis, direction, len*

Operands	*dest*	The field ID of the destination field.
	source	The field ID of the source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	direction	Either :upward or :downward.
	len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Note that in the conditional case the storing of data depends only on the *context-flag* of the processor receiving the data, not on the *context-flag* of the processor from which the data is obtained.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    $\textit{dest}[k] \leftarrow \textit{source}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})]$

where *news-neighbor* is as defined on page 40.

The *dest* field in each processor receives the contents of the *source* field of that processor's neighbor along the NEWS axis specified by *axis* in the direction specified by *direction*.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

GET-FROM-POWER-TWO

Each processor gets a message from a processor that is a specified distance away in the NEWS grid. The distance must be a power of two.

Formats

CM:get-from-power-two-1L	*dest*, *source*, *axis*, *log-2-distance*, *direction*, *len*
CM:get-from-power-two-always-1L	*dest*, *source*, *axis*, *log-2-distance*, *direction*, *len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

log-2-distance An unsigned integer immediate operand to be used as the base 2 logarithm of *distance*, where *distance* must be a power of 2.

direction Either :upward or :downward.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Note that in the conditional case data storage depends only on the *context-flag* of the processor receiving the data, not on the *context-flag* of the processor from which the data is obtained.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    $\textit{dest}[k] \leftarrow \textit{source}[\textit{news-relative}(g, k, \textit{axis}, \textit{direction}, \textit{log-2-distance})]$

where *news-relative* is defined in the NEWS Communication section of the Instruction Set Overview chapter.

The *dest* field in each processor receives the contents of the *source* field of that processor's relative along the NEWS axis specified by *axis*, in the direction specified by *direction*, and at the distance specified by *log-2-distance*.

The immediate operand *log-2-distance*, is $\log_2$ *distance*, where *distance* is the distance, along axis $axis$, between each destination processor and the source processor from which it retrieves data. In terms of this operand, *distance* is $2^{log\text{-}2\text{-}distance}$.

If *direction* is :upward then each processor retrieves data from a relative whose NEWS coordinate is ($coordinate + distance \bmod axis\text{-}length$). For most processors, this means getting from a processor whose coordinate is greater. The GET wraps around however; the processor whose coordinate is greatest retrieves data from the processor whose coordinate is ($0 + distance$).

If *direction* is :downward then each processor retrieves data from a relative whose NEWS coordinate is ($coordinate - distance \bmod axis\text{-}length$). For most processors, this means getting from a processor whose coordinate is less. The GET wraps around however; the processor whose coordinate is zero retrieves data from the processor whose coordinate is ($max\text{-}coordinate(axis) - distance$).

GLOBAL-C-ADD

The sum of the values in the complex source field is returned to the front end as a complex number.

Formats result ← CM:global-c-add-1L *source*, s, e

Operands *source* The field ID of the complex source field.

s, e The significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(s + e + 1)$.

Result A complex number, the sum of the *source* field.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $P = \{ m \mid 0 \leq m < \text{CM:*user-send-address-limit*} \}$
Let $S = \{ m \mid m \in P \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $+0$ to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-c-add-1L operation sums the *source* field values from all selected processors, treated as complex numbers. The sum is sent to the front-end computer as a complex number and returned as the result of the operation. If there are no selected processors, then the value $+0$ is returned.

GLOBAL-F-ADD

One floating-point number is examined in every selected processor, and the sum of all these fields is returned to the front end as a floating-point number.

Formats result $\leftarrow$ CM:global-f-add-1L *source*, s, e

Operands	*source*	The field ID of the floating-point source field.
	s, e	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.
Result	A floating-point number, the sum of the *source* fields.	
Overlap	There are no constraints, because overlap is not possible.	
Context	This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.	

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $+0$ to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-f-add operation sums the *source* fields, treated as floating-point numbers, in all selected processors. The sum is sent to the front-end computer as a floating-point number and returned as the result of the operation. If there are no selected processors, then the value $+0$ is returned.

GLOBAL-S-ADD

One signed integer is examined in every selected processor, and the sum of all these fields is returned to the front end as a signed integer.

Formats result $\leftarrow$ CM:global-s-add-1L *source*, *len*

Operands
source The field ID of the signed integer source field.
len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Result A signed integer, the sum of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \,\}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m]\right)$ to front end

The CM:global-s-add operation sums the *source* fields, treated as signed integers, in all selected processors. The sum is sent to the front-end computer as a signed integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned.

GLOBAL-U-ADD

One unsigned integer is examined in every selected processor, and the sum of all these fields is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-add-1L *source*, *len*

Operands	*source*	The field ID of the unsigned integer source field.
	len	The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
Result		An unsigned integer, the sum of the *source* fields.
Overlap		There are no constraints, because overlap is not possible.
Context		This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-u-add operation sums the *source* fields, treated as unsigned integers, in all selected processors. The sum is sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned.

GLOBAL-COUNT-BIT

One bit is examined in every selected processor, and the count of bits that are 1 is delivered to the front end.

Formats

result	$\leftarrow$	CM:global-count-bit	*source*
result	$\leftarrow$	CM:global-count-bit-always	*source*

Operands *source* The field ID of the source bit (a one-bit field).

Result An unsigned integer, the number of 1 bits.

Overlap There are no constraints, because overlap is not possible.

Context The non-always operations are conditional. The result returned depends only upon processors whose *context-flag* is 1.

The always operations are unconditional. The result returned does not depend on the *context-flag*.

Definition

If always then
  let $S = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{source}[m] = 1 \,\}$
else
  let $S = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{source}[m] = 1 \,\}$
return $|S|$ to front end

The CM:global-count-bit operation sums the one-bit *bit-source* fields in all selected processors; in other words, it returns a count of how many processors have a 1-bit in that field. The count is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned.

Using CM:global-count-bit is identical in effect to using CM:global-unsigned-add on a one-bit field, but may be faster.

GLOBAL-COUNT-CONTEXT

Returns the number of active processors.

Formats result $\leftarrow$ CM:global-count-context

Context This operation is unconditional.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
Return $|S|$ to front end

The number of processors whose context bit is 1 is returned to the front end.

GLOBAL-COUNT-flag

Returns the number of processors that have a specified flag set.

Formats CM:global-count-test
CM:global-count-overflow

Context This operation is conditional.

Definition Let $S = \{\, m \mid m \in \mathit{current\text{-}vp\text{-}set} \wedge \mathit{context\text{-}flag}[m] = 1 \wedge \mathit{flag}[m] = 1 \,\}$
Return $|S|$ to front end

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

The number of processors for which the specified flag is 1 is returned to the front end.

GLOBAL-LOGAND

One field is examined in every selected processor, and the bitwise logical AND of all these fields is returned to the front end as an unsigned integer.

Formats result ← CM:global-logand-1L *source*, *len*

Operands *source* The field ID of the source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical AND of all the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $2^{len} - 1$ to front end
else
  return $\left(\bigwedge_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-logand operation combines the *source* fields in all selected processors by performing bitwise logical AND operations. A bit is 1 in the result field if the corresponding bit is a 1 in *all* of the fields to be combined. The resulting combined field is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value $-2^{\text{len}} - 1$ is returned, representing a field of length *len* containing all ones.

GLOBAL-LOGAND-BIT

One memory bit is examined in each processor; 1 is returned if they are all 1, 0 if any is zero.

Formats result $\leftarrow$ CM:global-logand-bit *source*
result $\leftarrow$ CM:global-logand-bit-always *source*

Operands *source* The field ID of the source field.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical AND of all the *source* bits.

Overlap There are no constraints, because overlap is not possible.

Context The non-always operations are conditional. The result returned depends only upon processors whose *context-flag* is 1.

The always operations are unconditional. The result returned does not depend on the *context-flag*.

Definition If always then
let $S = \textit{current-vp-set}$
else
let $S = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \,\}$
If $|S| = 0$ then
return 1 to front end
else

$$\text{return } \left(\bigwedge_{m \in S} \textit{source}[m] \right) \text{ to front end}$$

The CM:global-logand-bit operation combines the *source* bits in all selected processors by performing a bitwise logical AND operation. The result is 1 if all the examined bits are 1; otherwise the result is 0. The result is sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 1 is returned.

Using CM:global-logand-bit is identical in effect to using CM:global-logand on a one-bit field, but may be faster.

GLOBAL-LOGAND-CONTEXT

Return 1 if all processors are active, 0 if any processor is inactive.

Formats result $\leftarrow$ CM:global-logand-context

Context This operation is unconditional.

Definition Return $\left(\bigwedge_{m \in \textit{current-vp-set}} \textit{context-flag}[m] \right)$ to front end

If all processors are active, then 1 is returned to the front end; otherwise 0 is returned.

GLOBAL-LOGAND-flag

Return 1 if a specified flag is set in all processors, 0 if it is clear in any processor.

Formats CM:global-logand-test
CM:global-logand-overflow

Context This operation is conditional.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{flag}[m] = 1 \}$
If $|S| = 0$ then
return 0 to front end
else
return $\left(\bigwedge_{m \in S} \textit{flag}[m] \right)$ to front end
where *flag* is *test-flag* or *overflow-flag*, as appropriate.

If all processors have the indicated flag set, then 1 is returned to the front end; otherwise 0 is returned.

GLOBAL-LOGIOR

One field is examined in every selected processor, and the bitwise logical inclusive OR of all these fields is returned to the front end as an unsigned integer.

Formats result ← CM:global-logior-1L *source*, *len*

Operands *source* The field ID of the source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical INCLUSIVE OR of all the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
return 0 to front end
else
return $\left(\bigvee_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-logior operation combines the *source* fields in all selected processors by performing bitwise logical INCLUSIVE OR operations. A bit is 1 in the result field if the corresponding bit is a 1 in *any* of the fields to be combined. The resulting combined field is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned, representing a field of length *len* containing all zeros.

GLOBAL-LOGIOR-BIT

One memory bit is examined in each processor; 1 is returned if any is 1, 0 if they are all zero.

Formats result $\leftarrow$ CM:global-logior-bit *source*
result $\leftarrow$ CM:global-logior-bit-always *source*

Operands *source* The field ID of the source field.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical OR of all the *source* bits.

Overlap There are no constraints, because overlap is not possible.

Context The non-always operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

The always operation is unconditional. The result returned does not depend on the *context-flag*.

Definition If always then
let $S = \mathit{current\text{-}vp\text{-}set}$
else
let $S = \{\, m \mid m \in \mathit{current\text{-}vp\text{-}set} \wedge \mathit{context\text{-}flag}[m] = 1 \,\}$
If $|S| = 0$ then
return 0 to front end
else

$$\text{return } \left(\bigvee_{m \in S} \mathit{source}[m] \right) \text{ to front end}$$

The CM:global-logior-bit operation combines the *source* bits in all selected processors by performing a bitwise logical inclusive OR operation. The result is 1 if any examined bit is 1; otherwise the result is 0. The result is sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned.

Using CM:global-logior-bit is identical in effect to using CM:global-logior on a one-bit field, but may be faster.

GLOBAL-LOGIOR-CONTEXT

Return 1 if any processor is active, 0 if no processors are active.

Formats result $\leftarrow$ CM:global-logior-context

Context This operation is unconditional.

Definition Return $\left(\bigvee_{m \in \mathit{current\text{-}vp\text{-}set}} \mathit{context\text{-}flag}[m] \right)$ to front end

If any processor has its context bit set, then 1 is returned to the front end; otherwise 0 is returned.

GLOBAL-LOGIOR-flag

Return 1 if a specified flag is set in any processor, 0 if it is clear in all processors.

Formats CM:global-logior-test
CM:global-logior-overflow

Context This operation is conditional.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\bigvee_{m \in S} \textit{flag}[m] \right)$ to front end
where *flag* is *test-flag* or *overflow-flag*, as appropriate.

If any processor has the indicated flag set, then 1 is returned to the front end; otherwise 0 is returned.

GLOBAL-LOGXOR

One field is examined in every selected processor, and the bitwise exclusive OR of all these fields is returned to the front end as an unsigned integer.

Formats result $\leftarrow$ CM:global-logxor-1L *source*, *len*

Operands *source* The field ID of the source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical exclusive OR of all the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\bigoplus_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-logxor operation combines the *source* fields in all selected processors by performing bitwise logical EXCLUSIVE OR operations. A bit is 1 in the result field if the corresponding bit is a 1 in *an odd number* of the fields to be combined. The resulting combined field is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned, representing a field of length *len* containing all zeros.

GLOBAL-F-MAX

One floating-point number is examined in every selected processor, and the largest of all these integers (that is, the one closest to $+\infty$) is returned to the front end as a floating-point number.

Formats result $\leftarrow$ CM:global-f-max-1L *source*, *s*, *e*

Operands

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Result A floating-point number, the largest of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor equals the maximum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \,\}$
If $|S| = 0$ then
  return $-\infty$ to front end
else
  let $R = \left(\max_{m \in S} \textit{source}[m]\right)$
  For every virtual processor k in the *current-vp-set* do
    if *context-flag*$[k] = 1$ then
      if *source*$[k] = R$ then
        test-flag$[k] \leftarrow 1$
      else
        test-flag$[k] \leftarrow 0$
  return R to front end

The CM:global-f-max operation returns the largest (that is, closest to $+\infty$) of the floating-point *source* fields of all selected processors. This largest value is sent to the front-end computer as a floating-point number and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value $-\infty$ is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-S-MAX

One signed integer is examined in every selected processor, and the largest of all these integers (that is, the one closest to $+\infty$) is returned to the front end as a signed integer.

Formats result $\leftarrow$ CM:global-s-max-1L *source*, *len*

Operands	*source*	The field ID of the signed integer source field.
	len	The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
Result	A signed integer, the largest of the *source* fields.	
Overlap	There are no constraints, because overlap is not possible.	
Flags	*test-flag* is set if the value in a particular processor equals the maximum; otherwise it is cleared.	
Context	This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.	

Definition

Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return -2^{len-1} to front end
else
  let $R = \left(\max_{m \in S} \textit{source}[m] \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-s-max operation returns the largest (that is, closest to $+\infty$) of the signed-integer *source* fields of all selected processors. This largest value is sent to the front-end computer as a signed integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value -2^{len-1} is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-U-MAX

One unsigned integer is examined in every selected processor, and the largest of all these integers is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-max-1L *source*, *len*

Operands

source The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the largest of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor equals the maximum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \,\}$

If $|S| = 0$ then
  return 0 to front end
else
  let $R = \left(\max_{m \in S} \textit{source}[m]\right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-u-max operation returns the largest of the unsigned-integer *source* fields of all selected processors. This largest value is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value 0 is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-U-MAX-S-INTLEN

One signed integer is examined in every selected processor, and the largest *length* of all these integers is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-max-s-intlen-1L *source*, *len*

Operands *source* The field ID of the signed integer source field.

len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the length of the *source* field value of greatest length.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor has a length equal to the maximum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  let $R = \left(\max_{m \in S} \left\lceil \log_2 \left(\frac{1}{2} + \left| \frac{1}{2} + \textit{source}[m] \right| \right) \right\rceil \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-u-max-s-intlen operation computes the integer-length of each signed integer *source* value. The largest length is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value 0 is returned.

A call to CM:global-u-max-s-intlen-1L is equivalent to the sequence

CM:s-integer-length-2-2L *temp*, *source*, *len*, *len*
CM:global-u-max-1L *temp*, *len*

but may be faster.

GLOBAL-U-MAX-U-INTLEN

One unsigned integer is examined in every selected processor, and the largest *length* of all these integers is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-max-u-intlen-1L *source*, *len*

Operands *source* The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the length of the *source* field value of greatest length.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor has a length equal to the maximum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  let $R = \left(\max\limits_{m \in S} \lceil \log_2 (1 + \textit{source}[m]) \rceil \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-u-max-u-intlen operation computes the integer-length of each unsigned integer *source* value. The largest length is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value 0 is returned.

A call to CM:global-u-max-u-intlen-1L is equivalent to the sequence

GLOBAL-MAX-INTLEN

CM:u-integer-length-2-2L *temp*, *source*, *len*, *len*
CM:global-u-max-1L *temp*, *len*

but may be faster.

GLOBAL-F-MIN

One floating-point number is examined in every selected processor, and the smallest of all these integers (that is, the one closest to $-\infty$) is returned to the front end as a floating-point number.

Formats result $\leftarrow$ CM:global-f-min-1L *source*, *s*, *e*

Operands	*source*	The field ID of the floating-point source field.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.
Result	A floating-point number, the smallest of the *source* fields.	
Overlap	There are no constraints, because overlap is not possible.	
Flags	*test-flag* is set if the value in a particular processor equals the minimum; otherwise it is cleared.	
Context	This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.	

Definition Let $S = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \,\}$

If $|S| = 0$ then
  return $+\infty$ to front end
else
  let $R = \left(\min\limits_{m \in S} \textit{source}[m]\right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-f-min operation returns the smallest (that is, closest to $-\infty$) of the floating-point *source* fields of all selected processors. This smallest value is sent to the front-end computer as a floating-point number and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value $+\infty$ is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-S-MIN

One signed integer is examined in every selected processor, and the smallest of all these integers (that is, the one closest to $-\infty$) is returned to the front end as a signed integer.

Formats result $\leftarrow$ CM:global-s-min-1L *source*, *len*

Operands *source* The field ID of the signed integer source field.

len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Result A signed integer, the smallest of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor equals the minimum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $2^{len-1} - 1$ to front end
else
  let $R = \left(\min_{m \in S} \textit{source}[m] \right)$ to front end
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-s-min operation returns the smallest (that is, closest to $-\infty$) of the signed-integer *source* fields of all selected processors. This smallest value is sent to the front-end computer as a signed integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value $2^{len-1} - 1$ is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-U-MIN

One unsigned integer is examined in every selected processor, and the smallest of all these integers is returned to the front end as an unsigned integer.

Formats result $\leftarrow$ CM:global-u-min-1L *source*, *len*

Operands	*source*	The field ID of the unsigned integer source field.
	len	The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
Result	An unsigned integer, the smallest of the *source* fields.	
Overlap	There are no constraints, because overlap is not possible.	
Flags	*test-flag* is set if the value in a particular processor equals the minimum; otherwise it is cleared.	
Context	This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.	

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $2^len - 1$ to front end
else
  let $R = \left(\min_{m \in S} \textit{source}[m]\right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-u-min operation returns the smallest (that is, closest to zero) of the unsigned-integer *source* fields of all selected processors. This smallest value is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value $2^{len} - 1$ is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

F-GT

Compares two floating-point source values. The *test-flag* is set if the first is strictly greater than the second, and otherwise is cleared.

Formats

CM:f-gt-1L	*source1*, *source2*, *s*, *e*
CM:f-gt-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-gt-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-gt-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] >$ *source2*$[k]$
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ is not greater than -0.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-GT

Compares two signed integer source values. The *test-flag* is set if the first is strictly greater than the second, and otherwise is cleared.

Formats

CM:s-gt-1L	*source1*, *source2*, *len*
CM:s-gt-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-gt-constant-1L	*source1*, *source2-value*, *len*
CM:s-gt-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-gt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] > *source2*[k] then
      test-flag[k] ← 1
    else
      test-flag[k] ← 0

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly

required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-GT

Compares two unsigned integer source values. The *test-flag* is set if the first is strictly greater than the second, and otherwise is cleared.

Formats

CM:u-gt-1L	*source1*, *source2*, *len*
CM:u-gt-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-gt-constant-1L	*source1*, *source2-value*, *len*
CM:u-gt-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-gt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] >$ *source2*$[k]$ then
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than the second operand and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

F-IEEE-TO-VAX

Converts the floating-point source field values from IEEE floating-point format to VAX floating-point format and stores the result in the destination field.

Formats CM:f-ieee-to-vax-1L *vax-dest*, *ieee-source*, *len*

Operands

vax-dest The field ID of the floating-point destination field.

ieee-source The field ID of the floating-point source field.

len The length of the *vax-dest* and *ieee-source* fields. The value of *len* must be either 32 or 64.

Overlap The fields *vax-dest* and *ieee-source* may overlap in any manner.

Flags *overflow-flag* is set if the ieee-source cannot be represented in the destination field; otherwise it is cleared. If *ieee-source* represents ∞ or NaN, then *vax-dest* is set to the "undefined variable" value in VAX format and the *overflow-flag* is cleared. If *ieee-source* represents -0.0, it is converted to VAX 0.0 and the *overflow-flag* is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

The Connection Machine operates internally on floating point data in IEEE format whereas the VAX uses a VAX floating-point format. In each active processor, this function converts a floating-point field in standard IEEE format to a field in VAX format.

The value of *len* specifies the precision of *vax-dest*. If *len* is specified as 32, then VAX 'F' format is used. If *len* is specified as 64, then VAX 'D' format is used.

VAX and IEEE floating-point formats are incompatible, so there are a number of potential inaccuracies in the translation. In general, if the conversion is accurate then the overflow flag is cleared; if inaccurate, then the overflow flag is set. See the flags description above.

This instruction is useful for rapidly converting floating-point data to VAX format, even if a VAX front end is not being used. For example, if data is to be transferred from a file in the CM file system to a VAX, CM:f-ieee-to-vax-1L should be called before writing the data file.

All Paris CM to front end data transfer functions automatically convert the data to the appropriate front-end format so it is not necessary to call CM:ieee-to-vax before calling, for instance, one of the read-from-news-array instructions.

To convert data back to IEEE floating-point format, see the definition of CM:f-vax-to-ieee-1L.

INIT

For the C/Paris and Fortran/Paris interfaces only. Makes various machine parameters available and performs a warm boot operation.

Formats CM:init

Context This operation is unconditional. It does not depend on the *context-flag*.

The facility for initializing Connection Machine hardware is provided in different ways in the Lisp/Paris interface (on the one hand) and the C/Paris and Fortran/Paris interfaces (on the other hand).

In the Lisp/Paris interface, there is no CM:init operation. Part of the work done by CM:init is performed by CM:cold-boot, and the remainder by CM:warm-boot.

In the C/Paris and Fortran/Paris interfaces, CM:init makes available to the user program various machine parameters that are initialized by the cmattach and cmcoldboot shell commands. It also performs all the functions of CM:warm-boot.

Every C or Fortran program that uses Paris should call CM:init before invoking any other Paris operations.

INITIALIZE-RANDOM-GENERATOR

Formats CM:initialize-random-generator *seed*

Operands *seed* An unsigned integer immediate operand to be used as the seed value for initializing the pseudo-random number generator.

Context This operation is unconditional. It does not depend on the *context-flag*.

Explicitly initializes the pseudo-random generator of numbers used by the Paris random number generator operations CM:f-random-1L and cm:u-random-1L. The seed (a front-end integer, which must be non-zero) determines the initial state.

If it has not been explicitly initialized by a call to this operation, the Paris random number generator is automaticaly initialized the first time it is called. Automatic initialization uses a seed based on the date and time.

In the Lisp/Paris interface, the *seed* argument is optional; if it is omitted, then a value based on the date and time of day is used.

Note: Less simple but more flexible random number generation routines are provided as part of the CM Scientific Subroutines Library (CMSSL). For instance, the CMSSL random number generators may be checkpointed to guard against accidental interuptions.

S-INTEGER-LENGTH

The minimum number of bits, minus one, needed to represent a signed integer value is placed in the destination field.

Formats CM:s-integer-length-2-2L *dest, source, dlen, slen*

Operands
- *dest* — The field ID of the unsigned integer destination field.
- *source* — The field ID of the signed integer source field.
- *dlen* — The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
- *slen* — The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source*$[k] \geq 0$ then *dest*$[k] \leftarrow \lceil \log_2(source[k] + 1) \rceil$
    else *dest*$[k] \leftarrow \lceil \log_2(-source[k]) \rceil$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The *dest* field receives, as an *unsigned* integer, the result of the computation

$$\begin{array}{ll} \lceil \log_2(s+1) \rceil & \text{if } s \geq 0 \\ \lceil \log_2(-s) \rceil & \text{if } s < 0 \end{array}$$

where s is the source value. This quantity is one less than the minimum number of bits required to represent s as a signed number, and will therefore be strictly less than *slen*.

U-INTEGER-LENGTH

The minimum number of bits needed to represent an unsigned integer value is placed in the destination field.

Formats CM:u-integer-length-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    $\mathit{dest}[k] \leftarrow \lceil \log_2(\mathit{source}[k] + 1) \rceil$
    if $\langle$overflow occurred in processor $k\rangle$ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$
    else $\mathit{overflow\text{-}flag}[k] \leftarrow 0$

The *dest* field receives, as an unsigned integer, the value $\lceil \log_2(s+1) \rceil$, where s is the source value. This quantity is the minimum number of bits required to represent s as an unsigned number, and will therefore be no greater than *slen*.

INTERN-DETAILED-GEOMETRY

Returns an interned geometry given detailed information about how the grid is laid out.

Formats result ← CM:intern-detailed-geometry *axis-descriptor-array, [rank]*

Operands *axis-descriptor-array* A front-end vector of descriptors for the grid axes. In the C interface, the elements of the *axis-descriptor-array* must be of type CM_axis_descriptor_t, that is, they must be pointers to structures of type CM_axis_descriptor.

In the Lisp interface, the *axis-descriptor-array* may be either a list of descriptors or an array of descriptors.

rank An unsigned integer, the rank (number of dimensions) of the *axis-descriptor-array*. This must be in between 1 and CM:*max-geometry-rank*, inclusive. This argument is not provided when calling Paris from Lisp.

Result A geometry ID, identifying the existing or newly created interned geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

By using interned geometries, modules that require identical geometries can use identical geometries – without having to keep track of the geometryID's.

CM:intern-detailed-geometry takes an array of descriptors. Each descriptor describes one NEWS axis in some detail. Most of the components are unsigned integers, but the value of the *ordering* component must be either :news-order or :send-order. The CM:create-detailed-geometry dictionary entry defines the type of the ordering component and of the descriptor for each language interface.

CM:intern-detailed-geometry is identical to CM:create-detailed-geometry with this exception: it returns an *interned* geometryID. A list of interned geometries is maintained and whenever CM:intern-detailed-geometry or intern-geometry is called, a previously interned geometry is returned if one exists that matches the specifications of the call, otherwise a new geometry is created and added to the list.

An interned geometryID is a geometryID returned by CM:intern-detailed-geometry or by CM:intern-geometry; a geometryID returned by CM:create-detailed-geometry or by CM:create-geometry may *not* be interned.

CM:create-detailed-geometry returns a unique, uninterned geometryID each time it is called. In contrast, CM:intern-detailed-geometry returns an existing interned geometryID if it can. If there is an interned geometry with an axis descriptor array that matches the supplied *axis-descriptor-array*, it is returned. Otherwise, CM:intern-detailed-geometry returns a new interned geometryID. The returned geometryID may be used to create a VP set or to respecify the geometry of an existing VP set.

Once the interned geometry has been created, the user may destroy the array created to provide the dimension information. All necessary information is copied from this array when the geometry is created.

INTERN-GEOMETRY

Returns an interned geometry given grid axis lengths.

Formats result ← CM:intern-geometry *dimension-array*; *[rank]*

Operands *dimension-array* A front-end vector of unsigned integer lengths of the grid axes. In the Lisp interface, this may be a list of dimension lengths instead of an array of dimension lengths, at the user's option.

rank An unsigned integer, the rank (number of dimensions) of the *dimension-array*. This must be in between 1 and CM:*max-geometry-rank*, inclusive. This argument is not provided when calling Paris from Lisp.

Result A geometry ID, identifying the existing or newly created interned geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

By using interned geometries, codes that require identical geometries can use identical geometries – without having to keep track of the geometryID's.

CM:intern-geometry is identical to CM:create-geometry with this exception: it returns an *interned* geometryID. An interned geometryID is a geometryID returned by CM:intern-geometry or by CM:intern-detailed-geometry; a geometryID returned by CM:create-geometry or by CM:create-detailed-geometry may *not* be interned.

CM:create-geometry returns a unique, uninterned geometryID each time it is called. In contrast, CM:intern-geometry returns an existing interned geometryID if it can. If there is a geometry, created by CM:intern-geometry and with dimensions that match those specified in *dimension-array*, it is returned. Otherwise, CM:intern-geometry returns a new interned geometryID. The returned geometryID may be used to create a VP set or to respecify the geometry of an existing VP set.

The *dimension-array* must be a one-dimensional array of nonnegative integers; each must be a power of two. The product of all these integers must be a multiple of the number of physical processors attached for use by this process.

The geometry is laid out so as to optimize performance under the assumption that the axes are used equally frequently for NEWS communication. The operations CM:create-detailed-geometry or CM:intern-detailed-geometry may be used instead to more precisely control layout for performance tuning.

Once the interned geometry has been created, the user may destroy the array used to provide the dimension information. All necessary information is copied out of this array when the geometry is created.

INTERN-IDENTICAL-VP-SET

Returns an interned VP set, within which fields may be allocated.

Formats result ← CM:intern-identical-vp-set *geometry-id*

Operands *geometry-id* A geometry ID.

Result A VP set ID, identifying the existing or newly allocated interned VP set.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation returns a VP set ID for an *interned* VP set. An interned VP set is a VP set referenced by a VP set ID returned by CM:intern-identical-vp-set. VP set interning allows different modules to reference identical VP sets and reduces VP set memory management overhead.

CM:intern-identical-vp-set returns an existing, interned VP set ID if there is an existing, interned VP set whose geometry is identical to the geometry specified by *geometry-id*. Otherwise, CM:intern-identical-vp-set returns a new, interned VP set ID.

Once a VP set has been created as interned, it may never be uninterned. Similarly, an uninterned VP set (created for instance with CM:create-vp-set) may never become interned.

An interned VP set may be used in the same ways as an uninterned VP set. For instance, it may be given to other Paris operations in order to create memory fields in which data may be stored. It may also be deallocated with CM:deallocate-vp-set.

INVERT-CONTEXT

Unconditionally makes all active processors inactive and vice versa.

Formats CM:invert-context

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do

$$context\text{-}flag[k] \leftarrow \neg context\text{-}flag[k]$$

Within each processor, the context bit for that processor is unconditionally inverted.

INVERT-flag

Inverts a specified flag bit.

Formats CM:invert-test
CM:invert-test-always
CM:invert-overflow
CM:invert-overflow-always

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $\mathit{context\text{-}flag}[k] = 1$) then
$\mathit{flag}[k] \leftarrow \neg \mathit{flag}[k]$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, the indicated flag for that processor is inverted.

IS-FIELD-AN-ALIAS

Returns true if the specified field ID is an alias field ID, false otherwise.

Formats result ← CM:is-field-an-alias *field-id*

Operands	*field-id*	A field ID.
Result	True if *field-id* is an alias field ID, and false otherwise.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

This operation tests whether the provided field ID is an alias field ID created with CM:make-field-alias, as opposed to a regular field ID created with a field allocation instruction such as CM:allocate-stack-field.

IS-FIELD-IN-HEAP

Returns true if the specified field is a heap field, false otherwise.

Formats result ← CM:is-field-in-heap *field-id*

Operands *field-id* A field ID.

Result True if the fieldID indicates a field allocated in the heap, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This instruction allows a program to test whether a given field has been allocated in the heap (as opposed to the stack).

IS-FIELD-IN-STACK

Returns true if the specified field is a stack field, false otherwise.

Formats result ← CM:is-field-in-stack *field-id*

Operands *field-id* A field ID.

Result True if the fieldID indicates a field allocated on the stack, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This instruction allows a program to test whether a given field has been allocated on the stack (as opposed to the heap).

IS-FIELD-VALID

Returns true if the specified field ID corresponds to a currently allocated CM field ID, false otherwise.

Formats result ← CM:is-field-valid *field-id*

Operands *field* ID A field ID.

Result True if *field-id* is a valid field ID, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This instruction allows a program to test whether the provided field ID is valid. Valid field ID's are assigned and returned by operations such as CM:allocate-stack-field, CM:allocate-heap-field, CM:add-offset-to-field-id, and CM:make-field-alias.

IS-STACK-FIELD-NEWER

Formats result ← CM:is-stack-field-newer *stack-query-field*, *stack-base-field*

Operands *stack-query-field* A field ID. The field must be in the stack.

stack-base-field A field ID. The field must be in the stack.

Result True if the *stack-query-field* has been allocated more recently than the *stack-base-field*, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation compares two stack fields and returns true if the second has been allocated more recently than the first.

IS-VP-SET-VALID

Returns true if the specified VP set ID corresponds to a currently allocated VP set, false otherwise.

Formats result ← CM:is-vp-set-valid *vp-set*

Operands *field* ID A VP set ID.

Result True if *vp-set-id* is a valid VP set ID, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This instruction allows a program to test whether the provided VP set ID is valid. Valid VP set ID's are assigned and returned by CM:allocate-vp-set.

S-ISQRT

The integer square root of a signed integer source field is placed in the destination field. This is the largest integer not larger than the true mathematical square root.

Formats

CM:s-isqrt-1-1L *dest/source*, *len*
CM:s-isqrt-2-1L *dest*, *source*, *len*
CM:s-isqrt-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest — The field ID of the signed integer destination field.

source — The field ID of the signed integer source field.

len — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen — The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen — The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags

test-flag is set if the *source* value is negative; otherwise it is cleared.

overflow-flag is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:s-isqrt-2-2L.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source}[k] \geq 0$ then
      $\textit{dest}[k] \leftarrow \lfloor\sqrt{\textit{source}}\rfloor$
      $\textit{test-flag}[k] \leftarrow 0$
    else
      $\textit{dest}[k] \leftarrow \langle\text{unpredictable}\rangle$
      $\textit{test-flag}[k] \leftarrow 1$
    if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$
    else $\textit{overflow-flag}[k] \leftarrow 0$
  as appropriate.

If the *source* value is non-negative, then the integer square root of that value (the largest integer not greater than the mathematical square root) is placed in the destination, and *test-flag* is cleared. Otherwise the *test-flag* is set and an unpredictable value is placed in the *dest* field.

U-ISQRT

The integer square root of an unsigned integer source field is placed in the destination field. This is the largest integer not larger than the true mathematical square root.

Formats

CM:u-isqrt-1-1L	*dest*/*source*, *len*
CM:u-isqrt-2-1L	*dest*, *source*, *len*
CM:u-isqrt-2-2L	*dest*, *source*, *dlen*, *slen*

Operands

dest — The field ID of the unsigned integer destination field.

source — The field ID of the unsigned integer source field.

len — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen — The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen — The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags — *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:u-isqrt-2-2L.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    $dest[k] \leftarrow \lfloor\sqrt{source}\rfloor$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0
  as appropriate.

The integer square root of the *source* value (the largest integer not greater than the mathematical square root) is placed in the destination.

LATCH-LEDS

Uses a one-bit field to turn the front-panel lights on or off.

Formats CM:latch-leds *source*
CM:latch-leds-always *source*

Operands *source* The field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition Let $g = \textit{geometry}(\textit{current-vp-set})$
Let $r = \textit{geometry-total-vp-ratio}(g) \times 16$
Let $n = \textit{geometry-total-processors}/r$
For all m such that $0 \le m < n$ do
  if always then
    turn on led m if and only if

$$\left(\bigvee_{j=0}^{r-1} \textit{source}[m \times n + j]\right) = 0$$

  else
    turn on led m if and only if

$$\left(\bigvee_{j=0}^{r-1} (\textit{source}[m \times n + j] \wedge \textit{context-flag}[m \times n + j])\right) = 0$$

The specified 1-bit field is read from every selected processor (or every processor, for the always version) and used to determine which LEDs should be illuminated. There is one LED associated with each group of 16 physical processors; each physical processor has some number of virtual processors. Two virtual processors belong to the same group if their virtual processor numbers agree in their $\log_2 n$ most significant bits, where n is the total number of LEDs. A LED is illuminated if every selected virtual processor in the group has a 0 in the selected *source* field (that is, the fields are combined for each group by a logical NOR operation).

Note that the pattern will actually persist in the lights only if CM:set-system-leds-mode has been called with the argument nil (in the Lisp/Paris interface) or 0 (in the C/Paris or Fortran/Paris interface); otherwise the Connection Machine system software will present other patterns in the lights.

F-LE

Compares two floating-point source values. The *test-flag* is set if the first is less than or equal to the second, and otherwise is cleared.

Formats

CM:f-le-1L	*source1, source2, s, e*
CM:f-le-constant-1L	*source1, source2-value, s, e*
CM:f-le-zero-1L	*source1, s, e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-le-zero-1L, this implicitly has the value zero.

s, e The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \leq$ *source2*$[k]$
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is less than or equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-LE

Compares two signed integer source values. The *test-flag* is set if the first is less than or equal to the second, and otherwise is cleared.

Formats

CM:s-le-1L	*source1*, *source2*, *len*
CM:s-le-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-le-constant-1L	*source1*, *source2-value*, *len*
CM:s-le-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-le-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \leq$ *source2*$[k]$ then
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is less than or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly

required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len.*

U-LE

Compares two unsigned integer source values. The *test-flag* is set if the first is less than or equal to the second, and otherwise is cleared.

Formats

CM:u-le-1L	*source1*, *source2*, *len*
CM:u-le-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-le-constant-1L	*source1*, *source2-value*, *len*
CM:u-le-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-le-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

```
if context-flag[k] = 1 then
   if source1[k] ≤ source2[k] then
      test-flag[k] ← 1
   else
      test-flag[k] ← 0
```

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is less than or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-LN

The natural logarithm of the complex source field values is placed in the complex destination field.

Formats
CM:c-ln-1-1L *dest/source*, s, e
CM:c-ln-2-1L *dest*, *source*, s, e

Operands

dest	The field ID of the complex destination field.
source	The field ID of the complex source field.
s, e	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap
The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags
test-flag is set if the *source* is zero; otherwise it is cleared.

Context
This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition
For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \ln$ *source*$[k]$

The value $\ln s$ is stored into the *dest* field, where s is the value of the *source* field. This is the natural logarithm to the base $e \approx 2.718281828\ldots$.

F-LN

The natural logarithm of the floating-point source field values are placed in the floating-point destination field.

Formats CM:f-ln-1-1L *dest/source, s, e*
CM:f-ln-2-1L *dest, source, s, e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is non-positive; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow \ln source[k]$
if $source[k] < 0$ then
$test[k] \leftarrow 1$
else $test[k] \leftarrow 0$

Call the value of the *source* field s. The value $\ln s$ is stored into the *dest* field; this is the natural logarithm to the base $e \approx 2.718281828\ldots$

LOAD-CONTEXT

Unconditionally reads a bit from memory and loads it into the context bit.

Formats CM:load-context *source*

Operands *source* The field ID of the source bit (a one-bit field).

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$context\text{-}flag[k] \leftarrow source[k]$

Within each processor, a bit is read from memory and unconditionally loaded into the context bit for that processor.

LOAD-flag

Reads a bit from memory and loads it into a flag.

Formats

CM:load-test	*source*
CM:load-test-always	*source*
CM:load-overflow	*source*
CM:load-overflow-always	*source*

Operands *source* The field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $\mathit{context\text{-}flag}[k] = 1$) then
$\mathit{flag}[k] \leftarrow \mathit{source}[k]$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, a bit is read from memory and loaded into the indicated flag for that processor.

F-LOG2

The base two logarithm of the floating-point source field is placed in the floating-point destination field.

Formats CM:f-log2-1-1L *dest/source*, *s*, *e*
CM:f-log2-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
$\textit{dest}[k] \leftarrow \log_2 \textit{source}[k]$

The value $\log_2 s$ is stored into the *dest* field, where s is the value of the *source* field. This is the logarithm to the base two of the floating-point source field.

F-LOG10

The base ten logarithm of the floating-point source field is placed in the floating-point destination field.

Formats CM:f-log10-1-1L *dest/source*, *s*, *e*
CM:f-log10-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \log_{10}$ *source*$[k]$

The value $\log_{10} s$ is stored into the *dest* field, where s is the value of the *source* field. This is the logarithm to the base ten of the floating-point source field.

LOGAND

Combines two source values using a bitwise logical AND operation, and places the result in the destination field.

Formats

CM:logand-2-1L	*dest/source1, source2, len*
CM:logand-always-2-1L	*dest/source1, source2, len*
CM:logand-constant-2-1L	*dest/source1, source2-value, len*
CM:logand-const-always-2-1L	*dest/source1, source2-value, len*
CM:logand-3-1L	*dest, source1, source2, len*
CM:logand-always-3-1L	*dest, source1, source2, len*
CM:logand-constant-3-1L	*dest, source1, source2-value, len*
CM:logand-const-always-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow source1[k] \wedge source2[k]$

Each bit of the *dest* field is set if both of the corresponding bits of the *source1* and *source2* fields are 1, and is cleared if either of the corresponding bits of the *source1* and *source2* fields is 0.

LOGAND-CONTEXT

Reads a bit from memory; if it is zero, the context bit is cleared, unconditionally.

Formats CM:logand-context *source*

Operands *source* The field ID of the source bit (a one-bit field).

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$context\text{-}flag[k] \leftarrow context\text{-}flag[k] \wedge source[k]$

Within each processor, a bit is read from memory and is "anded" into the context bit for that processor.

LOGAND-CONTEXT-WITH-TEST

If the test flag is zero, the context bit is cleared.

Formats CM:logand-context-with-test

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do

$$\textit{context-flag}[k] \leftarrow \textit{context-flag}[k] \wedge \textit{test-flag}[k]$$

Within each processor, the test flag is "anded" into the context bit for that processor.

LOGAND-flag

Reads a bit from memory; if it is zero, a specified flag is cleared.

Formats

CM:logand-test	*source*
CM:logand-test-always	*source*
CM:logand-overflow	*source*
CM:logand-overflow-always	*source*

Operands *source* The field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $\mathit{context\text{-}flag}[k] = 1$) then
$\mathit{flag}[k] \leftarrow \mathit{flag}[k] \wedge \mathit{source}[k]$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, a bit is read from memory and is "anded" into the indicated flag for that processor.

LOGANDC1

Combines the second source and the bitwise logical NOT of the first source using a bitwise logical AND operation. Places the result in the destination field.

Formats

CM:logandc1-2-1L	*dest/source1*, *source2*, *len*
CM:logandc1-always-2-1L	*dest/source1*, *source2*, *len*
CM:logandc1-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logandc1-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logandc1-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc1-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc1-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logandc1-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow (\neg source1[k]) \wedge source2[k]$

Each bit of the *dest* field is set if the corresponding bit of the *source1* field is 0 and the corresponding bit of the *source2* field is 1; otherwise it is cleared.

LOGANDC2

Combines the first source and the bitwise logical NOT of the second source using a bitwise logical AND operation. Places the result in the destination field.

Formats

CM:logandc2-2-1L	*dest*/*source1*, *source2*, *len*
CM:logandc2-always-2-1L	*dest*/*source1*, *source2*, *len*
CM:logandc2-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:logandc2-const-always-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:logandc2-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc2-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc2-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logandc2-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow source1[k] \wedge (\neg source2[k])$

Each bit of the *dest* field is set if the corresponding bit of the *source1* field is 1 and the corresponding bit of the *source2* field is 0; otherwise it is cleared.

S-LOGCOUNT

The destination field receives a count of the number of bits that differ from the sign bit in a two's-complement binary representation of a signed integer source value. For nonnegative values, this is a count of 1 bits.

Formats CM:s-logcount-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the signed integer source field.
dlen	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
slen	The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source*$[k] \geq 0$ then *dest*$[k] \leftarrow$ *count-of-one-bits*(*source*$[k]$)
    else *dest*$[k] \leftarrow$ *count-of-one-bits*($\neg$*source*$[k]$)
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The *dest* field receives, as an *unsigned* integer, a count of the number of bits in the two's-complement representation of the signed source value that are different from the sign bit of that value.

U-LOGCOUNT

The destination field receives a count of the number of 1 bits in the binary represenation of an unsigned integer source value.

Formats CM:u-logcount-2-2L *dest*, *source*, *dlen*, *slen*

Operands
- *dest* — The field ID of the unsigned integer destination field.
- *source* — The field ID of the unsigned integer source field.
- *dlen* — The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
- *slen* — The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap — The fields *dest* and *source* must not overlap in any manner.

Flags — *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow count\text{-}of\text{-}one\text{-}bits(source[k])$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$
    else $overflow\text{-}flag[k] \leftarrow 0$

The *dest* field receives, as an unsigned integer, a count of the number of bits in the binary representation of the unsigned source value.

LOGEQV

Combines two source values using a bitwise logical EQUIVALENCE operation, and places the result in the destination field.

Formats

CM:logeqv-2-1L	*dest*/*source1*, *source2*, *len*
CM:logeqv-always-2-1L	*dest*/*source1*, *source2*, *len*
CM:logeqv-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:logeqv-const-always-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:logeqv-3-1L	*dest*, *source1*, *source2*, *len*
CM:logeqv-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logeqv-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logeqv-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow \neg(source1[k] \oplus source2[k])$

Each bit of the *dest* field is set where corresponding bits of the *source1* and *source2* fields are alike, and is cleared where corresponding bits of the *source1* and *source2* fields differ.

LOGIOR

Combines two source values using a bitwise logical inclusive OR operation, and places the result in the destination field.

Formats

CM:logior-2-1L	*dest/source1*, *source2*, *len*
CM:logior-always-2-1L	*dest/source1*, *source2*, *len*
CM:logior-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logior-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logior-3-1L	*dest*, *source1*, *source2*, *len*
CM:logior-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logior-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logior-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow source1[k] \vee source2[k]$

Each bit of the *dest* field is set if either of the corresponding bits of the *source1* and *source2* fields is 1, and is cleared if both of the corresponding bits of the *source1* and *source2* fields are 0.

LOGIOR-CONTEXT

Reads a bit from memory; if it is one, the context bit is set, unconditionally.

Formats CM:logior-context *source*

Operands *source* The field ID of the source bit (a one-bit field).

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$$\textit{context-flag}[k] \leftarrow \textit{context-flag}[k] \vee \textit{source}[k]$$

Within each processor, a bit is read from memory and is "ored" into the context bit for that processor.

LOGIOR-flag

Reads a bit from memory; if it is 1, a specified flag is set.

Formats

CM:logior-test	*source*
CM:logior-test-always	*source*
CM:logior-overflow	*source*
CM:logior-overflow-always	*source*

Operands *source* The field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $flag[k] \leftarrow flag[k] \vee source[k]$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, a bit is read from memory and is "ored" into the indicated flag for that processor.

LOGNAND

Combines two source values with a bitwise logical NAND operation, and places the result in the destination field.

Formats

CM:lognand-2-1L	*dest/source1*, *source2*, *len*
CM:lognand-always-2-1L	*dest/source1*, *source2*, *len*
CM:lognand-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:lognand-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:lognand-3-1L	*dest*, *source1*, *source2*, *len*
CM:lognand-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:lognand-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:lognand-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow \neg(source1[k] \wedge source2[k])$

Each bit of the *dest* field is set if either of the corresponding bits of the *source1* and *source2* fields is 0, and is cleared if both of the corresponding bits of the *source1* and *source2* fields are 1.

LOGNOR

Combines two source values with a bitwise logical NOR operation, and places the result in the destination field.

Formats

CM:lognor-2-1L	*dest/source1, source2, len*
CM:lognor-always-2-1L	*dest/source1, source2, len*
CM:lognor-constant-2-1L	*dest/source1, source2-value, len*
CM:lognor-const-always-2-1L	*dest/source1, source2-value, len*
CM:lognor-3-1L	*dest, source1, source2, len*
CM:lognor-always-3-1L	*dest, source1, source2, len*
CM:lognor-constant-3-1L	*dest, source1, source2-value, len*
CM:lognor-const-always-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow \neg(source1[k] \vee source2[k])$

Each bit of the *dest* field is set if both of the corresponding bits of the *source1* and *source2* fields are 0, and is cleared if either of the corresponding bits of the *source1* and *source2* fields is 1.

LOGNOT

Copies a source field, inverts all the bits, and places them in the destination field.

Formats

CM:lognot-1-1L	*dest/source, len*
CM:lognot-always-1-1L	*dest/source, len*
CM:lognot-always-2-1L	*dest, source, len*
CM:lognot-2-1L	*dest, source, len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow \neg source[k]$

Each bit of the *dest* field is set to the inverse of the corresponding bit of the *source* field.

LOGORC1

Combines the second source and the bitwise logical NOT of the first source using a bitwise logical inclusive OR operation. Places the result in the destination field.

Formats

CM:logorc1-2-1L	*dest/source1, source2, len*
CM:logorc1-always-2-1L	*dest/source1, source2, len*
CM:logorc1-constant-2-1L	*dest/source1, source2-value, len*
CM:logorc1-const-always-2-1L	*dest/source1, source2-value, len*
CM:logorc1-3-1L	*dest, source1, source2, len*
CM:logorc1-always-3-1L	*dest, source1, source2, len*
CM:logorc1-constant-3-1L	*dest, source1, source2-value, len*
CM:logorc1-const-always-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow (\neg source1[k]) \vee source2[k]$

Each bit of the *dest* field is cleared if the corresponding bit of the *source1* field is 1 and if the corresponding bit of the *source2* field is 0; otherwise it is set.

LOGORC2

Combines the first source and the bitwise logical NOT of the second source using a bitwise logical inclusive OR operation. Places the result in the destination field.

Formats

CM:logorc2-2-1L	*dest/source1, source2, len*
CM:logorc2-always-2-1L	*dest/source1, source2, len*
CM:logorc2-constant-2-1L	*dest/source1, source2-value, len*
CM:logorc2-const-always-2-1L	*dest/source1, source2-value, len*
CM:logorc2-3-1L	*dest, source1, source2, len*
CM:logorc2-always-3-1L	*dest, source1, source2, len*
CM:logorc2-constant-3-1L	*dest, source1, source2-value, len*
CM:logorc2-const-always-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow source1[k] \vee (\neg source2[k])$

Each bit of the *dest* field is cleared if the corresponding bit of the *source1* field is 0 and if the corresponding bit of the *source2* field is 1; otherwise it is set.

LOGXOR

Combines two source values using a bitwise logical exclusive OR operation, and places the result in the destination field.

Formats

CM:logxor-2-1L	*dest/source1*, *source2*, *len*
CM:logxor-always-2-1L	*dest/source1*, *source2*, *len*
CM:logxor-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logxor-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logxor-3-1L	*dest*, *source1*, *source2*, *len*
CM:logxor-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logxor-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logxor-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    $dest[k] \leftarrow source1[k] \oplus source2[k]$

Each bit of the *dest* field is set where corresponding bits of the *source1* and *source2* fields differ, and is cleared where corresponding bits of the *source1* and *source2* fields are alike.

F-LT

Compares two floating-point source values. The *test-flag* is set if the first is strictly less than the second, and otherwise is cleared.

Formats

CM:f-lt-1L	*source1*, *source2*, *s*, *e*
CM:f-lt-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-lt-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-lt-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

```
if context-flag[k] = 1 then
   if source1[k] < source2[k]
      test-flag[k] ← 1
   else
      test-flag[k] ← 0
```

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is less than the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; -0 is not less than $+0$.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-LT

Compares two signed integer source values. The *test-flag* is set if the first is strictly less than the second, and otherwise is cleared.

Formats

CM:s-lt-1L	*source1, source2, len*
CM:s-lt-2L	*source1, source2, slen1, slen2*
CM:s-lt-constant-1L	*source1, source2-value, len*
CM:s-lt-zero-1L	*source1, len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-lt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\textit{context-flag}[k] = 1$ then
  if $\textit{source1}[k] < \textit{source2}[k]$ then
   $\textit{test-flag}[k] \leftarrow 1$
  else
   $\textit{test-flag}[k] \leftarrow 0$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is less than the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly

required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-LT

Compares two unsigned integer source values. The *test-flag* is set if the first is strictly less than the second, and otherwise is cleared.

Formats

CM:u-lt-1L	*source1*, *source2*, *len*
CM:u-lt-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-lt-constant-1L	*source1*, *source2-value*, *len*
CM:u-lt-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-lt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] <$ *source2*$[k]$ then
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is less than the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

MAKE-FIELD-ALIAS

Creates a new field ID that points to an existing field.

Formats result ← CM:make-field-alias *field-id*

Operands	*field-id*	A field ID. This must be a field ID returned by CM:allocate-stack-field or CM:allocate-heap-field; it may *not* be an offset into a field. The field need not be in the current VP set.
Result	A field ID, identifying the alias field ID. This ID initially resides in the current VP set.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

The return value is a *field alias*. It is a new field ID that identifies the same area of memory as does *field-id*.

The field identified by *field-id* can be in a VP set other than the current VP set. The returned alias field ID initially resides in the current VP set. The alias field ID can be used in all the same ways as a regular field ID can, with the following exceptions:

- It cannot be passed to CM:deallocate-heap-field.
- It cannot be passed to CM:deallocate-stack-through.

Associated with a field alias is a *physical length*: the number of bits that the field occupies in each physical processor. Also associated with a field alias is a *field length*: the number of bits the field occupies in each virtual processor. The physical length is equal to the field length multiplied by the VP ratio of the current VP set. It is an error if the physical length is not exactly divisible by the VP ratio of the current VP set.

It is possible for the field length of an alias field to be different from the field length of the original field. This is the case when make-field-alias is called on a field in a VP set that has a VP ratio different from the VP ratio of the current VP set. Suppose, for example, the current VP ratio is 32. If we make an alias for a 32-bit field that resides in a VP set with a VP ratio of 1, the resulting alias field is a 1 bit field (in a VP ratio of 32).

MAKE-NEWS-COORDINATE

Determine the send-address of a processor with the specified NEWS coordinate.

Formats CM:make-news-coordinate-1L *geometry, dest, axis, news-coordinate, slen*

Operands

geometry A geometry ID. This determines the NEWS dimensions to be used.

dest The field ID of the unsigned integer destination, to receive the send address of the processor whose coordinate along the specified axis is *news-coordinate* and whose coordinate along all other axes is a zero field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

news-coordinate The field ID of the unsigned integer NEWS coordinate along the specified axis field.

slen The length of the *news-coordinate* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\textit{context-flag}[k] = 1$ then
 $\textit{dest}[k] \leftarrow \textit{make-news-coordinate}(\textit{axis}, \textit{news-coordinate})$

where *make-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates zero.

FE-MAKE-NEWS-COORDINATE

Calculates, entirely on the front end, the send-address of the processor with the specified coordinate along the specified NEWS axis and with all other coordinates zero.

Formats result ← CM:fe-make-news-coordinate *geometry, axis, news-coordinate*

Operands

geometry A geometry ID. This determines the NEWS dimensions to be used.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

news-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along the specified axis.

Result An unsigned integer, the send address of the processor whose coordinate along the specified axis is *news-coordinate* and whose coordinate along all other axes is zero.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Return *make-news-coordinate*(*axis*, *news-coordinate*)

where *make-news-coordinate* is as defined on page 40.

This function calculates, entirely on the front end, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates zero.

C-MATRIX-MULTIPLY

Computes matrix multiplication using three single-precision complex operands and stores the result in the last.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix c- to signify that single-precision complex operands are used. A more efficient version of this operation is included in the CM Scientific Subroutines Library.

Formats CMSSL:c-matrix-multiply *source1*, *source2*, *dest*/*source3*

Operands	*dest*	The field ID of the complex destination field.
	source1	The field ID of the complex first source field.
	source2	The field ID of the complex second source field.
	source3	The field ID of the complex third source field.
Overlap	The fields *source1*, *source2*, and *dest/source3* must not overlap in any manner.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

The calculation $dest \leftarrow source3 + source1 \times source2$ is performed on three conforming matrices, represented as CM fields.

The operands *source1*, *source2*, and *dest/source3* must be fields of 64-bit single-precision complex values whose real and imaginary parts are 32-bit floating-point values.

All three operands may belong to separate VP sets if the geometries of those VP sets obey the following rule:

- The *source1* dimensions are $n \times m$
- The *source2* dimensions are $m \times p$
- The *dest/source3* dimensions are $n \times p$

where n, m, and p are each powers of two. Otherwise, all three operands must belong to the same square VP set.

The matrix multiply is performed using Cannon's systolic algorithm, which can be summarized in three steps:

1. The *source1* and *source2* matrices are aligned so the elements in each processor have conforming indices for matrix multiplication. In terms of data motion, this implies aligning the diagonal entries of the *source1* matrix to the first column and aligning the diagonal entries of the *source2* matrix to the first row.

2. The systolic part of the algorithm involves local multiplication of *source1* and *source2* elements followed by nearest neighbor data moves that simulate the inner product.

3. The *source1* and *source2* matrices are aligned back to the original form supplied by the calling program.

In order to exploit the full potential of the floating-point hardware, a block version of the algorithm is implemented. See the Thinking Machines technical report entitled "Matrix Multiplication on the Connection Machine" for details.

The CM matrix multiplication operation performs best for square matrices and at high VP ratios.

C/Paris code that calls the Paris matrix multiplication routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris matrix multiplication routine.

Fortran/Paris code should include the line

```
  INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris matrix multiplication routine.

S-MATRIX-MULTIPLY

Computes matrix multiplication using three single-precision floating-point operands and stores the result in the last.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix s- to signify that single-precision floating-point operands are used. A more efficient version of this operation is included in the CM Scientific Subroutines Library.

Formats CMSSL:s-matrix-multiply *source1*, *source2*, *dest*/*source3*

Operands	*dest*	The field ID of the floating-point destination field.
	source1	The field ID of the floating-point first source field.
	source2	The field ID of the floating-point second source field.
	source3	The field ID of the floating-point third source field.
Overlap		The fields *source1*, *source2*, and *dest/source3* must not overlap in any manner.
Context		This operation is unconditional. It does not depend on the *context-flag*.

The calculation $dest \leftarrow source3 + source1 \times source2$ is performed on three conforming matrices, represented as CM fields.

The operands *source1*, *source2*, and *dest/source3* must be fields of 32-bit single-precision floating-point values.

All three operands may belong to separate VP sets if the geometries of those VP sets obey the following rule:

- The *source1* dimensions are $n \times m$
- The *source2* dimensions are $m \times p$
- The *dest/source3* dimensions are $n \times p$

where n, m, and p are each powers of two. Otherwise, all three operands must belong to the same square VP set.

The matrix multiply is performed using Cannon's systolic algorithm, which can be summarized in three steps:

1. The *source1* and *source2* matrices are aligned so the elements in each processor have conforming indices for matrix multiplication. In terms of data motion, this implies aligning the diagonal entries of the *source1* matrix to the first column and aligning the diagonal entries of the *source2* matrix to the first row.

2. The systolic part of the algorithm involves local multiplication of *source1* and *source2* elements, followed by nearest neighbor data moves that simulate the inner product.

3. The *source1* and *source2* matrices are aligned back to the original form supplied by the calling program.

In order to exploit the full potential of the floating-point hardware, a block version of the algorithm is implemented. See the Thinking Machines technical report entitled "Matrix Multiplication on the Connection Machine" for details.

The CM matrix multiplication routine performs best for square matrices and at high VP ratios.

C/Paris code that calls the Paris matrix multiplication routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris matrix multiplication routine.

Fortran/Paris code should include the line

```
  INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris matrix multiplication routine.

F-MAX

Two floating-point values are compared. The larger is placed in the destination field.

Formats

CM:f-max-2-1L	*dest/source1, source2, s, e*
CM:f-max-3-1L	*dest, source1, source2, s, e*
CM:f-max-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-max-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \geq$ *source2*$[k]$ then
      dest$[k] \leftarrow$ *source1*$[k]$
      test-flag$[k] \leftarrow 0$
    else
      dest$[k] \leftarrow$ *source2*$[k]$
      test-flag$[k] \leftarrow 1$

Two operands are compared as floating-point numbers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The larger of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-MAX

Two signed integer values are compared. The larger (the one closer to $+\infty$) is placed in the destination field.

Formats

CM:s-max-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-max-2-1L	*dest/source1, source2, len*
CM:s-max-3-1L	*dest, source1, source2, len*
CM:s-max-constant-2-1L	*dest/source1, source2-value, len*
CM:s-max-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-max-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-max-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-max-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \geq$ *source2*$[k]$ then
      dest$[k] \leftarrow$ *source1*$[k]$
      test-flag$[k] \leftarrow 0$
    else
      dest$[k] \leftarrow$ *source2*$[k]$
      test-flag$[k] \leftarrow 1$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The larger of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-MAX

Two unsigned integer values are compared. The larger is placed in the destination field.

Formats

CM:u-max-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-max-2-1L	*dest/source1, source2, len*
CM:u-max-3-1L	*dest, source1, source2, len*
CM:u-max-constant-2-1L	*dest/source1, source2-value, len*
CM:u-max-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-max-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-max-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-max-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
if *source1*$[k] \geq$ *source2*$[k]$ then
dest$[k] \leftarrow$ *source1*$[k]$
test-flag$[k] \leftarrow 0$
else
dest$[k] \leftarrow$ *source2*$[k]$
test-flag$[k] \leftarrow 1$

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The larger of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

F-MIN

Two floating-point values are compared. The smaller is placed in the destination field.

Formats

CM:f-min-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-min-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-min-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-min-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source1 — The field ID of the floating-point first source field.

source2 — The field ID of the floating-point second source field.

source2-value — A floating-point immediate operand to be used as the second source.

s, *e* — The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap — The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags — *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source1}[k] \leq \textit{source2}[k]$ then
      $\textit{dest}[k] \leftarrow \textit{source1}[k]$
      $\textit{test-flag}[k] \leftarrow 0$
    else
      $\textit{dest}[k] \leftarrow \textit{source2}[k]$
      $\textit{test-flag}[k] \leftarrow 1$

Two operands are compared as floating-point numbers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The smaller of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-MIN

Two signed integer values are compared. The smaller (the one closer to $-\infty$) is placed in the destination field.

Formats

CM:s-min-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-min-2-1L	*dest/source1, source2, len*
CM:s-min-3-1L	*dest, source1, source2, len*
CM:s-min-constant-2-1L	*dest/source1, source2-value, len*
CM:s-min-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-min-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-min-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-min-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \leq$ *source2*$[k]$ then
      dest$[k] \leftarrow$ *source1*$[k]$
      test-flag$[k] \leftarrow 0$
    else
      dest$[k] \leftarrow$ *source2*$[k]$
      test-flag$[k] \leftarrow 1$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The smaller of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-MIN

Two unsigned integer values are compared. The smaller is placed in the destination field.

Formats

CM:u-min-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-min-2-1L	*dest*/*source1*, *source2*, *len*
CM:u-min-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-min-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:u-min-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest — The field ID of the unsigned integer destination field.

source1 — The field ID of the unsigned integer first source field.

source2 — The field ID of the unsigned integer second source field.

source2-value — An unsigned integer immediate operand to be used as the second source.

len — The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen — For CM:u-min-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 — For CM:u-min-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 — For CM:u-min-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \leq$ *source2*$[k]$ then
      dest$[k] \leftarrow$ *source1*$[k]$

$test\text{-}flag[k] \leftarrow 0$
else
$dest[k] \leftarrow source2[k]$
$test\text{-}flag[k] \leftarrow 1$

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The smaller of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

F-MOD

One floating-point source field is divided by another and the residue is placed in the destination field. Overflow is also computed.

This operation's name is derived from the term modulus; the destination field receives the the residue of taking one source field *modulus* another source field.

Formats

CM:f-mod-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-mod-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-mod-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-mod-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field. This is the quotient.

source1 The field ID of the floating-point first source field. This is the dividend.

source2 The field ID of the floating-point second source field. This is the divisor.

source2-value A floating-point immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if division by zero occurs; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source2*$[k] = 0$ then
      dest$[k] \leftarrow \langle$unpredictable$\rangle$

$test\text{-}flag[k] \leftarrow 1$
else
$$dest[k] \leftarrow source1[k] - source2[k] \times \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$
$test\text{-}flag[k] \leftarrow 0$
if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

The residue resulting from the reduction of the floating-point *source1* operand divided by the *source2* operand is stored in the *dest* field. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-MOD

One signed integer source field is divided by another and the residue is placed in the destination field. Overflow is also computed.

This operation's name is derived from the term modulus; the destination field receives the the residue of taking one source field *modulus* another source field.

Formats

CM:s-mod-2-1L	*dest/source1*, *source2*, *len*
CM:s-mod-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-mod-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:s-mod-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the signed integer residue field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer modulus (divisor) field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the modulus (divisor) is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then
  if $\textit{source2}[k] = 0$ then
    $\textit{dest}[k] \leftarrow \langle\text{unpredictable}\rangle$
  else
    $$\textit{dest}[k] \leftarrow \textit{source1}[k] - \textit{source2}[k] \times \left\lfloor \frac{\textit{source1}[k]}{\textit{source2}[k]} \right\rfloor$$
  if ⟨divisor was zero in processor k⟩ then $\textit{test-flag}[k] \leftarrow 1$
  else $\textit{test-flag}[k] \leftarrow 0$

The residue resulting from the reduction of the signed integer *source1* modulo the signed integer *source2* operand is stored into the *dest* field. The result always has the same sign as the *source2* operand. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

If the divisor is zero occurs, then the *test-flag* is set and the value of the destination is unpredictable

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-MOD

One unsigned integer source field is divided by another and the residue is placed in the destination field. Overflow is also computed.

This operation's name is derived from the term modulus; the destination field receives the the residue of taking one source field *modulus* another source field.

Formats

CM:u-mod-2-1L	*dest/source1, source2, len*
CM:u-mod-3-1L	*dest, source1, source2, len*
CM:u-mod-constant-2-1L	*dest/source1, source2-value, len*
CM:u-mod-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer residue field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer modulus (divisor) field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the modulus (divisor) is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source2}[k] = 0$ then
      $\textit{dest}[k] \leftarrow \langle\text{unpredictable}\rangle$
    else
      $\textit{dest}[k] \leftarrow \textit{source1}[k] - \textit{source2}[k] \times \left\lfloor \frac{\textit{source1}[k]}{\textit{source2}[k]} \right\rfloor$
    if ⟨divisor was zero in processor k⟩ then $\textit{test-flag}[k] \leftarrow 1$
    else $\textit{test-flag}[k] \leftarrow 0$

The residue resulting from the reduction of the unsigned integer *source1* modulo the unsigned integer *source2* operand is stored into the *dest* field. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

If the divisor is zero occurs, then the *test-flag* is set and the value of the destination is unpredictable

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-MOVE

Copies a complex source value into the destination field.

Formats

CM:c-move-2L	*dest*, *source*, *ds*, *de*, *ss*, *se*
CM:c-move-1L	*dest*, *source*, *s*, *e*
CM:c-move-always-1L	*dest*, *source*, *s*, *e*
CM:c-move-constant-1L	*dest*, *source-value*, *s*, *e*
CM:c-move-const-always-1L	*dest*, *source-value*, *s*, *e*
CM:c-move-zero-1L	*dest*, *s*, *e*
CM:c-move-zero-always-1L	*dest*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

source-value The field ID of the complex source field. For CM:c-move-zero-1L and CM:c-move-zero-always-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

ds, *de* For CM:c-move-2L, the significand and exponent lengths for the *dest* field. The total length of an operand in this format is $2(ds + de + 1)$.

ss, *se* For CM:c-move-2L, the significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(ss + se + 1)$.

Overlap The fields *dest* and *source* may overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:c-move-2L.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ *source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

else *overflow-flag*[k] $\leftarrow 0$
as appropriate.

The *source* field or value is copied into the *dest* field.

However, overlapping fields are not handled carefully and should be avoided.

F-MOVE

Copies a floating-point source value into the destination field.

Formats

CM:f-move-2L	*dest*, *source*, *ds*, *de*, *ss*, *se*
CM:f-move-1L	*dest*, *source*, *s*, *e*
CM:f-move-always-1L	*dest*, *source*, *s*, *e*
CM:f-move-constant-1L	*dest*, *source-value*, *s*, *e*
CM:f-move-const-always-1L	*dest*, *source-value*, *s*, *e*
CM:f-move-zero-1L	*dest*, *s*, *e*
CM:f-move-zero-always-1L	*dest*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source — The field ID of the floating-point source field.

source-value — A floating-point immediate operand to be used as the source. This should be of type double-float in Lisp/Paris and will be coerced if necessary. For CM:f-move-zero-1L and CM:f-move-zero-always-1L, this implicitly has the value zero.

s, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

ds, *de* — For CM:f-move-2L, the significand and exponent lengths for the *dest* field. The total length of an operand in this format is $ds + de + 1$.

ss, *se* — For CM:f-move-2L, the significand and exponent lengths for the *source* field. The total length of an operand in this format is $ss + se + 1$.

Overlap The fields *dest* and *source* may overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:f-move-2L.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow source[k]$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

else *overflow-flag*$[k] \leftarrow 0$
as appropriate.

The *source* field or value is copied into the *dest* field.

Overlapping fields are handled carefully. The operation behaves as if the entire *source* field were first copied to a temporary buffer not overlapping either the *source* or *dest* field, and then the temporary buffer copied to the *dest* field.

S-MOVE

Copies a signed integer source value into the destination field.

Formats

CM:s-move-2L	*dest, source, dlen, slen*
CM:s-move-1L	*dest, source, len*
CM:s-move-always-1L	*dest, source, len*
CM:s-move-constant-1L	*dest, source-value, len*
CM:s-move-const-always-1L	*dest, source-value, len*
CM:s-move-zero-1L	*dest, len*
CM:s-move-zero-always-1L	*dest, len*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

source-value A signed integer immediate operand to be used as the source. For CM:s-move-zero-1L and CM:s-move-zero-always-1L, this implicitly has the value zero.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than the maximum Paris field length.

dlen For CM:s-move-1L, the length of the *dest* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

slen For CM:s-move-1L, the length of the *source* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

Overlap The fields *dest* and *source* may overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:s-move-2L.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ *source*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The *source* field or value is copied into the *dest* field. For CM:s-move-2L, if *slen* is less than *dlen* then the source value, regarded as a bit field, is padded at the most significent end with copies of the most significant source bit (sign extension), and if *slen* is greater than *dlen* then truncation occurs and overflow may be detected.

Overlapping fields are handled carefully. The operation behaves as if the entire *source* field were first copied to a temporary buffer not overlapping either the *source* or *dest* field, and then the temporary buffer copied to the *dest* field.

U-MOVE

Copies an unsigned integer source value into the destination field.

Formats

CM:u-move-2L	*dest, source, dlen, slen*
CM:u-move-1L	*dest, source, len*
CM:u-move-always-1L	*dest, source, len*
CM:u-move-constant-1L	*dest, source-value, len*
CM:u-move-const-always-1L	*dest, source-value, len*
CM:u-move-zero-1L	*dest, len*
CM:u-move-zero-always-1L	*dest, len*

Operands

dest — The field ID of the unsigned integer destination field.

source — The field ID of the unsigned integer source field.

source-value — An unsigned integer immediate operand to be used as the source. For CM:u-move-zero-1L and CM:u-move-zero-always-1L, this implicitly has the value zero.

len — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than the maximum Paris field length.

dlen — For CM:u-move-1L, the length of the *dest* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

slen — For CM:u-move-1L, the length of the *source* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

Overlap — The fields *dest* and *source* may overlap in any manner.

Flags — *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:u-move-2L.

Context — The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow source[k]$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$
    else $overflow\text{-}flag[k] \leftarrow 0$

The *source* field or value is copied into the *dest* field. For CM:u-move-2L, if *slen* is less than *dlen* then the source value, regarded as a bit field, is padded at the most significent end with zero bits, and if *slen* is greater than *dlen* then truncation occurs and overflow may be detected.

Overlapping fields are handled carefully. The operation behaves as if the entire *source* field were first copied to a temporary buffer not overlapping either the *source* or *dest* field, and then the temporary buffer copied to the *dest* field.

F-MOVE-DECODED-CONSTANT

Copies a decoded immediate floating-point source value into the destination field.

Formats CM:f-move-decoded-constant-1L *dest, low-s-value, high-s-value, e-value, sign-value, s, e*

Operands

dest The field ID of the floating-point destination field.

low-s-value An unsigned integer immediate operand to be used as the low 32 bits of the integer significand.

high-s-value An unsigned integer immediate operand to be used as the high bits of the integer significand.

e-value A signed integer immediate operand to be used as the integer exponent.

sign-value A signed integer immediate operand to be used as the integer sign. This must be either 1 or -1.

s, e The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \textit{sign-value} \times (\textit{low-s-value} + 2^{32} \times \textit{high-s-value}) \times 2^{\textit{e-value}}$

The three quantities $\textit{low-s-value} + 2^{32} \times \textit{high-s-value}$, *e-value*, and *sign-value* are three integers that together describe a floating-point value. (This is the same decoded form that is used by such Common Lisp operations as integer-decode-float.) This floating-point value is copied into the *dest* field.

In the Lisp interface one may use a "bignum" as the *low-s-value* and always pass zero for the *high-s-value*. In the C interface, however, it is not possible to pass an integer of more than 32 bits. The *high-s-value* operand provides a way around this difficulty that works compatibly in either language.

MOVE-REVERSED

Copies the source values into the destination field, reversing the order of the bits.

Formats

CM:move-reversed-1L	*dest*, *source*, *len*
CM:move-reversed-always-1L	*dest*, *source*, *len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    for j from 0 to $len - 1$ do
      $dest[k]\langle j\rangle \leftarrow source[k]\langle len - j - 1\rangle$

The *source* field or value is copied into the *dest* field, with the order of the bits reversed; that is, the least significant bit of the *source* field is copied into the most significant bit of the *dest* field, and so on.

F-MULT-ADD

Calculates a value $xa + b$ and places it in the destination.

Formats

CM:f-mult-add-1L	*dest, source1, source2, source3, s, e*
CM:f-mult-add-always-1L	*dest, source1, source2, source3, s, e*
CM:f-mult-const-add-1L	*dest, source1, source2-value, source3, s, e*
CM:f-mult-const-add-always-1L	*dest, source1, source2-value, source3, s, e*
CM:f-mult-add-const-1L	*dest, source1, source2, source3-value, s, e*
CM:f-mult-add-const-always-1L	*dest, source1, source2, source3-value, s, e*
CM:f-mult-const-add-const-1L	*dest, source1, source2-value, source3-value, s, e*
CM:f-mult-const-add-const-a-1L	*dest, source1, source2-value, source3-value, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source (multiplier) field.

source2-value A floating-point immediate operand to be used as the second source (multiplier).

source3 The field ID of the floating-point third source (augend) field.

source3-value A floating-point immediate operand to be used as the third source (augend).

s, e The significand and exponent lengths for the *dest, source1, source2,* and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1, source2,* and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow (source1[k] \times source2[k]) + source3[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as floating-point numbers and then a third operand, *source3*, is added to the product. The result is stored in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-mult-add-1L is equivalent to the sequence

CM:f-multiply-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-add-3-1L *dest*, *temp*, *source3*, *s*, *e*

but may be faster.

F-MULT-SUB

Calculates a value $xa - b$ and places it in the destination.

Formats

CM:f-mult-sub-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-sub-always-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-const-sub-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-const-sub-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-sub-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-sub-const-always-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-const-sub-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-mult-const-sub-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source (multiplier) field.

source2-value A floating-point immediate operand to be used as the second source (multiplier).

source3 The field ID of the floating-point third source (subtrahend) field.

source3-value A floating-point immediate operand to be used as the third source (subtrahend).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    $\textit{dest}[k] \leftarrow (\textit{source1}[k] \times \textit{source2}[k]) - \textit{source3}[k]$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as floating-point numbers and then a third operand, *source3*, is subtracted from the product. The result is stored in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-mult-sub-1L is equivalent to the sequence

CM:f-multiply-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-subtract-3-1L *dest*, *temp*, *source3*, *s*, *e*

but may be faster.

F-MULT-SUBF

Calculates a value $b - xa$ and places it in the destination.

Formats

CM:f-mult-subf-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-subf-always-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-const-subf-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-const-subf-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-subf-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-subf-const-always-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-const-subf-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-mult-const-subf-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source (multiplier) field.

source2-value A floating-point immediate operand to be used as the second source (multiplier).

source3 The field ID of the floating-point third source (minuend) field.

source3-value A floating-point immediate operand to be used as the third source (minuend).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ *source3*$[k] - ($*source1*$[k] \times$ *source2*$[k])$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

Two operands *source1* and *source2* are multiplied as floating-point numbers and the product is subtracted from a third operand, *source3*. The result is stored in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-mult-subf-1L is equivalent to the sequence

CM:f-multiply-3-1L	*temp*, *source1*, *source2*, *s*, *e*
CM:f-subtract-3-1L	*dest*, *source3*, *temp*, *s*, *e*

but may be faster.

C-MULTIPLY

The product of two complex source values is placed in the destination field.

Formats

CM:c-multiply-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-multiply-always-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-multiply-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-multiply-always-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-multiply-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-multiply-const-always-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-multiply-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:c-multiply-const-always-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow source1[k] \times source2[k]$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as complex numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-MULTIPLY

The product of two floating-point source values is placed in the destination field.

Formats

CM:f-multiply-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-multiply-always-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-multiply-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-multiply-always-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-multiply-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-multiply-const-always-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-multiply-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:f-multiply-const-always-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow source1[k] \times source2[k]$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as floating-point numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-MULTIPLY

The product of two signed integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:s-multiply-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-multiply-2-1L	*dest*/*source1*, *source2*, *len*
CM:s-multiply-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-multiply-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:s-multiply-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-multiply-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-multiply-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-multiply-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the product cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow$ *source1*$[k] \times$ *source2*$[k]$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$

Two operands, *source1* and *source2*, are multiplied as signed integers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-MULTIPLY

The product of two unsigned integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:u-multiply-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-multiply-2-1L	*dest*/*source1*, *source2*, *len*
CM:u-multiply-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-multiply-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:u-multiply-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-multiply-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-multiply-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-multiply-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do

if *context-flag*$[k] = 1$ then
  $dest[k] \leftarrow source1[k] \times source2[k]$
  if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
  else *overflow-flag*$[k] \leftarrow 0$

Two operands, *source1* and *source2*, are multiplied as unsigned integers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

MULTISPREAD-C-ADD

The destination field in every selected processor receives the sum of the complex floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-c-add-1L *dest*, *source*, *axis-mask*, *s*, *e*

Operands	*dest*	The field ID of the complex destination field.
	source	The field ID of the complex source field.
	axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
let $r = \mathit{rank}(g)$
let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \wedge (\mathit{axis\text{-}mask}\langle m\rangle = 1)\,\}$
let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \wedge \mathit{context\text{-}flag}[m] = 1\,\}$

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \mathit{source}[m]\right)$$

where *hyperplane* is as defined on 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-c-add operation combines *source* fields by performing complex floating-point addition.

A call to CM:multispread-c-add-1L is equivalent to the sequence

for all integers j, $0 \le j < \mathit{rank}(\mathit{geometry}(\mathit{current\text{-}vp\text{-}set}))$, in any sequential order, do
  if $\mathit{axis\text{-}mask}\langle j\rangle = 1$ then
    CM:spread-with-c-add-1L *dest*, *source*, j, *s*, *e*

but may be faster.

MULTISPREAD-F-ADD

The destination field in every selected processor receives the sum of the floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-f-add-1L *dest*, *source*, *axis-mask*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\mathit{context\text{-}flag}[k] = 1$ then

let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$

let $r = \mathit{rank}(g)$

let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \wedge (\mathit{axis\text{-}mask}\langle m \rangle = 1)\,\}$

let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \wedge \mathit{context\text{-}flag}[m] = 1\,\}$

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \mathit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-f-add operation combines *source* fields by performing floating-point addition.

A call to CM:multispread-f-add-1L is equivalent to the sequence

CM:f-move-zero-always-1L *temp*, *s*, *e*
CM:f-move-1L *temp*, *source*, *s*, *e*
CM:store-context *ctemp*
CM:set-context

for all integers j, $0 \leq j < rank(geometry(current\text{-}vp\text{-}set))$, in any sequential order, do
  if $axis\text{-}mask\langle j \rangle = 1$ then
    CM:spread-with-f-add-1L *temp*, *temp*, j, *s*, *e*
CM:load-context *ctemp*
CM:f-move-1L *dest*, *temp*, *s*, *e*

but may be faster.

MULTISPREAD-S-ADD

The destination field in every selected processor receives the sum of the signed integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-s-add-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $r = rank(g)$
let $axis\text{-}set = \{\, m \mid 0 \leq m < r \wedge (axis\text{-}mask\langle m\rangle = 1) \,\}$
let $C_k = \{\, m \mid m \in hyperplane(g, k, axis\text{-}set) \wedge context\text{-}flag[m] = 1 \,\}$

$$dest[k] \leftarrow \left(\sum_{m \in C_k} source[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-s-add operation combines *source* fields by performing signed integer addition.

MULTISPREAD-U-ADD

The destination field in every selected processor receives the sum of the unsigned integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-u-add-1L *dest, source, axis-mask, len*

Operands

	dest	The field ID of the unsigned integer destination field.
	source	The field ID of the unsigned integer source field.
	axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
	len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $r = \textit{rank}(g)$

let $\textit{axis-set} = \{\, m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$

let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-u-add operation combines *source* fields by performing unsigned integer addition.

MULTISPREAD-COPY

The destination field in every selected processor receives a copy of the source value from a particular value within its scan subclass.

Formats CM:multispread-copy-1L *dest*, *source*, *axis-mask*, *len*, *multi-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

multi-coordinate An unsigned integer, the multi-coordinate indicating which element of each hyperplane is to be replicated throughout that hyperplane.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $r = \mathit{rank}(g)$
    let $\mathit{axis\text{-}set} = \{\, m \mid 0 \leq m < r \wedge (\mathit{axis\text{-}mask}\langle m \rangle = 1)\,\}$
    let $c = \mathit{deposit\text{-}multi\text{-}coordinate}(g, k, \mathit{axis\text{-}set}, \mathit{multi\text{-}coordinate})$
    $\mathit{dest}[k] \leftarrow \mathit{source}[c]$

where *deposit-multi-coordinate* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations.

To construct a multi-coordinate, construct a send-address and provide it as an argument to CM:fe-extract-multi-coordinate.

MULTISPREAD-LOGAND

The destination field in every selected processor receives the bitwise logical AND of the source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-logand-1L *dest*, *source*, *axis-mask*, *len*

Operands	*dest*	The field ID of the destination field.
	source	The field ID of the source field.
	axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
	len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $r = \textit{rank}(g)$
let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m\rangle = 1) \,\}$
let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$

$$\textit{dest}[k] \leftarrow \left(\bigwedge_{m \in C_k} \textit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-logand operation combines *source* fields by performing bitwise logical AND operations.

MULTISPREAD-LOGIOR

The destination field in every selected processor receives the bitwise logical inclusive OR of the source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-logior-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\mathit{context\text{-}flag}[k] = 1$ then

let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$

let $r = \mathit{rank}(g)$

let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \land (\mathit{axis\text{-}mask}\langle m \rangle = 1) \,\}$

let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \land \mathit{context\text{-}flag}[m] = 1 \,\}$

$$\mathit{dest}[k] \leftarrow \left(\bigvee_{m \in C_k} \mathit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-logior operation combines *source* fields by performing bitwise logical inclusive OR operations.

MULTISPREAD-LOGXOR

The destination field in every selected processor receives the bitwise logical exclusive OR of the source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-logxor-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $r = \textit{rank}(g)$

let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1)\,\}$

let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1\,\}$

$$\textit{dest}[k] \leftarrow \left(\bigoplus_{m \in C_k} \textit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations.

MULTISPREAD-F-MAX

The destination field in every selected processor receives the largest of the floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-f-max-1L *dest*, *source*, *axis-mask*, *s*, *e*

Operands	*dest*	The field ID of the floating-point destination field.
	source	The field ID of the floating-point source field.
	axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m\rangle = 1)\,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1\,\}$
    $\textit{dest}[k] \leftarrow \left(\max\limits_{m \in C_k} \textit{source}[m]\right)$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-f-max operation combines *source* fields by performing a floating-point maximum operation.

MULTISPREAD-S-MAX

The destination field in every selected processor receives the largest of the signed integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-s-max-1L *dest*, *source*, *axis-mask*, *len*

Operands	*dest*	The field ID of the signed integer destination field.
	source	The field ID of the signed integer source field.
	axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
	len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $r = \textit{rank}(g)$

let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1)\,\}$

let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1\,\}$

$\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m]\right)$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-s-max operation combines *source* fields by performing a signed integer maximum operation.

MULTISPREAD-U-MAX

The destination field in every selected processor receives the largest of the unsigned integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-u-max-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\mathit{context\text{-}flag}[k] = 1$ then
  let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
  let $r = \mathit{rank}(g)$
  let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \wedge (\mathit{axis\text{-}mask}\langle m \rangle = 1)\,\}$
  let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \wedge \mathit{context\text{-}flag}[m] = 1\,\}$
  $\mathit{dest}[k] \leftarrow \left(\max\limits_{m \in C_k} \mathit{source}[m]\right)$
 where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-u-max operation combines *source* fields by performing an unsigned integer maximum operation.

MULTISPREAD-F-MIN

The destination field in every selected processor receives the smallest of the floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-f-min-1L *dest, source, axis-mask, s, e*

Operands

- *dest* — The field ID of the floating-point destination field.
- *source* — The field ID of the floating-point source field.
- *axis-mask* — An unsigned integer, the mask indicating a set of NEWS axes.
- *s, e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $r = \textit{rank}(g)$
let $\textit{axis-set} = \{\, m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m \rangle = 1)\,\}$
let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1\,\}$
$\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m]\right)$
where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-f-min operation combines *source* fields by performing a floating-point minimum operation.

MULTISPREAD-S-MIN

The destination field in every selected processor receives the smallest of the signed integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-s-min-1L *dest, source, axis-mask, len*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the signed integer source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\mathit{context\text{-}flag}[k] = 1$ then

let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$

let $r = \mathit{rank}(g)$

let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \wedge (\mathit{axis\text{-}mask}\langle m\rangle = 1)\,\}$

let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \wedge \mathit{context\text{-}flag}[m] = 1\,\}$

$$\mathit{dest}[k] \leftarrow \left(\min_{m \in C_k} \mathit{source}[m]\right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-s-min operation combines *source* fields by performing a signed integer minimum operation.

MULTISPREAD-U-MIN

The destination field in every selected processor receives the smallest of the unsigned integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-u-min-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\mathit{context\text{-}flag}[k] = 1$ then

let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$

let $r = \mathit{rank}(g)$

let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \land (\mathit{axis\text{-}mask}\langle m \rangle = 1)\,\}$

let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \land \mathit{context\text{-}flag}[m] = 1\,\}$

$\mathit{dest}[k] \leftarrow \left(\min\limits_{m \in C_k} \mathit{source}[m]\right)$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-u-min operation combines *source* fields by performing an unsigned integer minimum operation.

MY-NEWS-COORDINATE

Stores the NEWS coordinate of each selected processor along a specified NEWS axis into a destination field within that processor.

Formats CM:my-news-coordinate-1L *dest*, *axis*, *dlen*

Operands	*dest*	The field ID of the unsigned integer destination field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	dlen	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\textit{context-flag}[k] = 1$ then
  let $g = \textit{geometry}(\textit{current-vp-set})$
  $\textit{dest}[k] \leftarrow \textit{extract-news-coordinate}(g, \textit{axis}, k)$

where *extract-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the NEWS coordinate of that processor along a specified NEWS axis.

MY-SEND-ADDRESS

Stores the send-address of each selected processor into a destination field in that processor.

Formats CM:my-send-address *dest*

Operands *dest* The field ID of the unsigned integer destination field. This must be no less than the value returned by **CM:geometry-send-address-length**.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow k$

This function stores into the *dest* field, within each selected processor, the send-address of that processor.

C-NE

Compares two complex source values. The *test-flag* is set if they are not equal; otherwise it is cleared.

Formats

CM:c-ne-1L	*source1*, *source2*, *s*, *e*
CM:c-ne-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:c-ne-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source. For CM:c-ne-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \neq$ *source2*$[k]$
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as complex numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is not equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-NE

Compares two floating-point source values. The *test-flag* is set if they are not equal, and otherwise is cleared.

Formats

CM:f-ne-1L	*source1*, *source2*, *s*, *e*
CM:f-ne-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-ne-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-ne-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] \neq$ *source2*$[k]$
      test-flag$[k] \leftarrow 1$
    else
      test-flag$[k] \leftarrow 0$

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is not equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-NE

Compares two signed integer source values. The *test-flag* is set if they are not equal, and otherwise is cleared.

Formats

CM:s-ne-1L	*source1*, *source2*, *len*
CM:s-ne-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-ne-constant-1L	*source1*, *source2-value*, *len*
CM:s-ne-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-ne-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source1}[k] \neq \textit{source2}[k]$ then
      $\textit{test-flag}[k] \leftarrow 1$
    else
      $\textit{test-flag}[k] \leftarrow 0$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is not equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-NE

Compares two unsigned integer source values. The *test-flag* is set if they are not equal, and otherwise is cleared.

Formats

CM:u-ne-1L	*source1*, *source2*, *len*
CM:u-ne-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-ne-constant-1L	*source1*, *source2-value*, *len*
CM:u-ne-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-ne-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] $\neq$ *source2*[k] then
      test-flag[k] $\leftarrow$ 1
    else
      test-flag[k] $\leftarrow$ 0

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is not equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-NEGATE

Copies a complex number with both signs inverted.

Formats CM:c-negate-1-1L *dest/source*, *s*, *e*
CM:c-negate-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
$\textit{dest}[k].\textit{real} \leftarrow -\textit{source}[k].\textit{real}$
$\textit{dest}[k].\textit{imag} \leftarrow -\textit{source}[k].\textit{imag}$

A copy of the *source* operand, with both sign bits inverted, is placed in the *dest* operand.

F-NEGATE

Copies a floating-point number with its sign inverted.

Formats CM:f-negate-1-1L *dest/source*, *s*, *e*
CM:f-negate-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow -\textit{source}[k]$

A copy of the *source* operand, with its sign bit inverted, is placed in the *dest* operand. This is done even if the operand is a NaN, whether a signalling NaN or a quiet NaN.

This operation therefore differs from the operation of subtracting a floating-point number from the constant zero when the operand is ± 0 or a NaN.

S-NEGATE

Computes the negative (that is, the additive inverse) of a signed integer source field and places it in the destination field.

Formats CM:s-negate-1-1L *dest/source*, *len*
CM:s-negate-2-1L *dest*, *source*, *len*
CM:s-negate-2-2L *dest*, *source*, *dlen*, *slen*

Operands *dest* The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow -$*source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The negative of the *source* operand is placed in the *dest* operand. If overflow occurs, then the *overflow-flag* is set. (If the length of the *dest* field equals the length n of the *source* field, overflow can occur only if the *source* field contains -2^n. If the length of the *dest* field is greater than the length of the *source* field, then overflow cannot occur.)

U-NEGATE

The "negative" (that is, the unsigned additive inverse) of an unsigned integer source field is placed in the destination field. This is an unsigned value that, when added to the original source field, will produce zero (possibly with overflow).

Formats

CM:u-negate-1-1L *dest/source*, *len*
CM:u-negate-2-1L *dest*, *source*, *len*
CM:u-negate-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest — The field ID of the unsigned integer destination field.

source — The field ID of the unsigned integer source field.

len — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen — The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen — The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. Overflow occurs whenever the source value is non-zero.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] $\leftarrow -$*source*[k]
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] $\leftarrow 1$
    else *overflow-flag*[k] $\leftarrow 0$

The negative of the *source* operand is placed in the *dest* operand. If overflow occurs, then the *dest* field will contain a value equal to $2^{len} - source$. This operation matches the functionality of the unary "-" operator on unsigned integers in the C language.

F-NEWS-ADD

The sum of two floating-point source values (one from a NEWS neighbor) is placed in the destination field.

Formats

CM:f-news-add-2-1L	*dest, source, axis, direction, s, e*
CM:f-news-add-always-2-1L	*dest, source, axis, direction, s, e*
CM:f-news-add-3-1L	*dest, source1, source2, axis, direction, s, e*
CM:f-news-add-always-3-1L	*dest, source1, source2, axis, direction, s, e*
CM:f-news-add-const-3-1L	*dest, source1, source2-value, axis, direction, s, e*
CM:f-news-add-const-a-3-1L	*dest, source1, source2-value, axis, direction, s, e*

Operands

dest — The field ID of the floating-point destination field.

source — The field ID of the floating-point source field.

source1 — The field ID of the floating-point first source field.

source2 — The field ID of the floating-point second source field.

source2-value — A floating-point immediate operand to be used as the second source.

axis — An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction — Either :upward or :downward.

s, e — The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
$\textit{dest}[k] \leftarrow \textit{source1}[k] + \textit{source2}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})]$
if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

where *news-neighbor* is is defined in the NEWS Communication section of the Instruction Set Overview Chapter.

Two source operands are added as floating-point numbers and the result is stored in *dest*. The various operand formats allow source operands to be either memory fields or constants. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The instructions with two operands take *source* from a NEWS neighbor, sum it with *dest* and store the result back in *dest*.

For the instructions CM:f-news-add-3-1L and CM:f-news-add-always-3-1L, *source2* is taken from a NEWS neighbor.

The instructions CM:f-news-add-const-3-1L and CM:f-news-add-const-a-3-1L take *source1* is from a NEWS neighbor. Note that the *a* in CM:f-news-add-const-a-3-1L stands for "always."

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-add-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-add-3-1L *dest*, *source1*, *temp*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-ADD-MULT

Calculates the value $(a+x)b$, where one of the operands is taken from a NEWS neighbor, and places the result in the destination.

Formats

CM:f-news-add-mult-4-1L *dest*, *source1*, *source2*, *source3*, *axis*, *direction*, *s*, *e*
CM:f-news-add-const-mult-4-1L *dest*, *source1*, *source2-value*, *source3*, *axis*, *direction*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

source3 A floating-point immediate operand to be used as the third source.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s+e+1$.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1. Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    $\textit{dest}[k] \leftarrow (\textit{source1} + \textit{source2}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})]) \times \textit{source3}[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$

The sum of two source operands is multiplied by the value of a third source operand. The result is stored in *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The CM:f-news-add-mult-4-1L instruction takes *source2* from a NEWS neighbor. For the CM:f-news-add-const-mult-4-1L instruction, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-add-mult is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-add-mult-1L *souce1*, *temp*, *source3*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-MULT

The product of two floating-point source values (one from a NEWS neighbor) is placed in the destination field.

Formats

CM:f-news-mult-2-1L	*dest, source, axis, direction, s, e*
CM:f-news-mult-always-2-1L	*dest, source, axis, direction, s, e*
CM:f-news-mult-3-1L	*dest, source1, source2, axis, direction, s, e*
CM:f-news-mult-always-3-1L	*dest, source1, source2, axis, direction, s, e*
CM:f-news-mult-const-3-1L	*dest, source1, source2-value, axis, direction, s, e*
CM:f-news-mult-const-a-3-1L	*dest, source1, source2-value, axis, direction, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*. Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then
  let $g = geometry(current\text{-}vp\text{-}set)$
  $dest[k] \leftarrow source1[k] \times source2[news\text{-}neighbor(g, k, axis, direction)]$
  if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

Two source operands are multiplied as floating-point numbers. The result is stored in *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The instructions with two operands take *source* from a NEWS neighbor, multiply it with *dest*, and store the result back in *dest*.

For the instructions CM:f-news-mult-3-1L and CM:f-news-mult-always-3-1L, *source2* is taken from a NEWS neighbor.

For the instructions CM:f-news-mult-const-3-1L and CM:f-news-mult-const-a-3-1L, *source1* is taken from a NEWS neighbor. Note that the *a* in CM:f-news-mul-const-always-3-1L stands for "always." This is necessary to meet the 31 character limit on instruction names.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-mult-3-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-multiply-3-1L *dest*, *source1*, *temp*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-MULT-ADD

The product of two floating-point source values (one from a NEWS neighbor) is added to yet another floating-point source value; the result is placed in the destination field.

Formats

CM:f-news-mult-add-4-1L	*dest*, *source1*, *source2*, *source3*, *axis*, *direction*, *s*, *e*
CM:f-news-mult-const-add-4-1L	*dest*, *source1*, *source2-value*, *source3*, *axis*, *direction*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point multiplicand field.

source2 The field ID of the floating-point multiplier field. These values may be taken from a NEWS neighbor.

source2-value A floating-point immediate operand to be used as the multiplier.

source3 The field ID of the floating-point addend field. These values may be taken from a NEWS neighbor.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then

let $g = geometry(current\text{-}vp\text{-}set)$
$dest[k] \leftarrow source1[k] \times source2[news\text{-}neighbor(g, k, axis, direction)] + source3[k]$
if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

Two operands are multiplied as floating-point numbers; to the product is added a third operand. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

For CM:f-news-mult-add-4-1L, *source2* is taken from a NEWS neighbor.

For CM:f-news-mult-const-add-4-1L, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* or *source3-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-mult-add-4-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-multiply-3-1L *temp*, *source1*, *temp*, s, e
CM:f-add-3-1L *dest*, *temp*, *source3*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-MULT-SUB

From the product of two floating-point source values (one from a NEWS neighbor) is subtracted yet another floating-point source value; the result is placed in the destination field.

Formats CM:f-news-mult-sub-4-1L *dest, source1, source2, source3, axis, direction, s, e*
CM:f-news-mult-const-sub-4-1L *dest, source1, source2-value, source3, axis, direction, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point multiplicand field.

source2 The field ID of the floating-point multiplier field.

source2-value A floating-point immediate operand to be used as the multiplier.

source3 The field ID of the floating-point subtrahend field.

source3-value A floating-point immediate operand to be used as the subtrahend.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    $\textit{dest}[k] \leftarrow \textit{source1}[k] \times \textit{source2}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})] - \textit{source3}[k]$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as floating-point numbers; from the product is subtracted a third operand, *source3*. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

For CM:f-news-mult-sub-4-1L, *source2* is taken from a NEWS neighbor.

For and CM:f-news-mult-const-sub-4-1L, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* or *source3-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-mult-sub-4-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-multiply-3-1L *temp*, *source1*, *temp*, s, e
CM:f-subtract-3-1L *dest*, *temp*, *source3*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-SUB

The difference of two floating-point source values (one from a NEWS neighbor) is placed in the destination field.

Formats

CM:f-news-sub-2-1L	*dest, source, axis, direction, s, e*
CM:f-news-sub-always-2-1L	*dest, source, axis, direction, s, e*
CM:f-news-sub-3-1L	*dest, source1, source2, axis, direction, s, e*
CM:f-news-sub-always-3-1L	*dest, source1, source2, axis, direction, s, e*
CM:f-news-sub-const-3-1L	*dest, source1, source2-value, axis, direction, s, e*
CM:f-news-sub-const-a-3-1L	*dest, source1, source2-value, axis, direction, s, e*

Operands

dest The field ID of the floating-point destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the floating-point first source field) field. This is the minuend.

source2 The field ID of the floating-point second source field. This is the subtrahend.

source2-value A floating-point immediate operand to be used as the second source.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
let $g =$ *geometry*(*current-vp-set*)
dest$[k] \leftarrow$ *source1*$[k]$ – *source2*[*news-neighbor*(g, k, *axis*, *direction*)]
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The operands are treated as as floating-point numbers and one is subtracted from another. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The instructions with two operands take *source* from a NEWS neighbor, subtract it from *dest*, and store the result stored back in *dest*.

For the instructions CM:f-news-sub-3-1L and CM:f-news-sub-always-3-1L, *source2* is obtained from a NEWS neighbor.

For the instructions CM:f-news-sub-const-3-1L and CM:f-news-sub-const-a-3-1L, *source2* is a constant and *source1* is obtained from a NEWS neighbor. Note that the *a* in CM:f-news-sub-const-a-3-1L stands for "always."

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-sub-3-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-subtract-3-1L *dest*, *source1*, *temp*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-SUB-MULT

Calculates the value $(a - x)b$, when one of the operands is taken from a NEWS neighbor, and places the result in the destination.

Formats

CM:f-news-sub-mult-4-1L *dest, source1, source2, source3, axis, direction, s, e*
CM:f-news-sub-const-mult-4-1L *dest, source1, source2-value, source3, axis, direction, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

source3 The field ID of the floating-point third source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    $\textit{dest}[k] \leftarrow (\textit{source1} - \textit{source2}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})]) \times \textit{source3}[k]$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The difference of two operands is multiplied by the value of a third operand. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The CM:f-news-sub-mult-4-1L instruction takes *source2* from a NEWS neighbor. For the CM:f-news-sub-const-mult-4-1L instruction, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-sub-mult-4-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-sub-mult-1L *dest*, *source1*, *temp source3*, s, e

but is faster at high VP ratios and requires little temporary memory.

NEXT-STACK-FIELD-ID

Determines the next stack field id that would be returned by a call to CM:allocate-stack-field.

Formats result ← CM:next-stack-field-id

Operands None.

Result An unsigned integer, the field ID that will be returned by the next invocation of CM:allocate-stack-field.

Context This operation is unconditional. It does not depend on the *context-flag*.

This function returns the next stack field id to be allocated.

FE-PACKED-ARRAY-FORMAT

This front-end instruction returns an array format descriptor for a packed front-end array format. A format descriptor may be used as the *format* argument to any array transfer instruction, although this is not required.

See also CM:fe-array-format and CM:fe-structure-array-format.

Formats result ← CM:fe-packed-array-format *cm-element-size*, *[array-element-size]*

Operands *cm-element-size* A signed integer immediate operand to be used as the number of bits each Connection Machine element occupies in the front-end array. This must be a power of two between 1 and 128.

array-element-size A signed integer immediate operand to be used as the number of bits in each front-end array element. This must be a power of two between 1 and 128.

In Lisp/Paris, this argument is optional. If not specified, it defaults to the actual front-end element size or, if the front-end array elements are general (i.e., of type t), *array-element-size* defaults to the value of *cm-element-size*.

Result The array format descriptor specified.

Context This is a front-end operation. It does not depend on the value of the *context-flag*.

The return value is a format descriptor for packed arrays; it can be passed to any array transfer instruction. In this format, multiple Connection Machine array elements are packed into each front-end array element during array transfers in either direction between the Connection Machine and the front-end computer.

By using this instruction, it is also possible to specify an extended-element front-end array format. In an extended-element format, each CM element is stored in multiple front-end array elements.

The value of *cm-element-size* defines the unit of measure for the *fe-offset-vector* argument to the CM:read-from-news-array and CM:write-to-news-array instructions.

The value of *array-element-size* defines the unit of measure for the argument *fe-dimension-vector* to the CM:read-from-news-array and CM:write-to-news-array instructions.

The number of Connection Machine elements packed into each front-end array element is the ratio of *array-element-size* to *cm-element-size*. If *array-element-size* is larger than *cm-element-size*, multiple Connection Machine elements are packed into each front-end array element. Alternatively, if *array-element-size* is smaller than *cm-element-size*, each CM element is stored in more than one front-end array element.

The ordering of the packing defaults to the standard ordering for the front end. For example, on a VAX the Connection Machine element with the smallest coordinates is put into the least significant bits of the front-end array element. On a Sun, the Connection Machine element with the largest coordinates is put into the least significant bits of the front-end array element.

F-C-PHASE

Calculates the phase of the complex source field and puts the result in the floating-point destination field.

Formats CM:f-c-phase-2-1L *dest*, *source*, s, e

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the complex source field.
s, e	The significand and exponent lengths for the *dest* and *source* fields. The total length of the *dest* field in this format is $s + e + 1$. The total length of the *source* field in this format is $2(s + e + 1)$.

Overlap: The *dest* field must be either identical to *source*, identical to $(source{+}s{+}e{+}1)$, or disjoint from *source*.

Flags: *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    $\mathit{dest}[k] \leftarrow \mathit{atan2}(\mathit{source}[k].\mathit{imag}, \mathit{source}[k].\mathit{real})$
    if $\langle$overflow occurred in processor $k\rangle$ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$

The phase of a number is the angle part of its polar representation as a complex number.

PHYSICAL-VP-SET

Returns a VP set that has one virtual processor for each physical processor.

Formats result ← CM:physical-vp-set

Operands None.

Result A VP set ID, identifying the VP set whose VP ratio is 1.

Context This operation is unconditional. It does not depend on the *context-flag*.

C-C-POWER

Raises a complex number to a complex power.

Formats

CM:c-c-power-2-1L	*dest/source1, source2, s, e*
CM:c-c-power-3-1L	*dest, source1, source2, s, e*
CM:c-c-power-constant-2-1L	*dest/source1, source2-value, s, e*
CM:c-c-power-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected. *test-flag* is set if zero is raised to a non-positive power; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

$\quad \textit{dest}[k] \leftarrow \textit{source1}[k]^{\textit{source2}[k]}$

$\quad$ if $\textit{source1}[k] = 0.0$ and $\textit{source2}[k].\textit{real} \leq 0.0$

$\quad$ and $\textit{source2}[k].\textit{imag} = 0.0$ then

$\quad\quad \textit{test-flag}[k] \leftarrow 1$

$\quad$ else $\textit{test-flag}[k] \leftarrow 0$

$\quad$ if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent), using exp and ln operations.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

C-F-POWER

Raises a complex number to a floating-point power.

Formats

CM:c-f-power-2-1L	*dest/source1, source2, s, e*
CM:c-f-power-3-1L	*dest, source1, source2, s, e*
CM:c-f-power-constant-2-1L	*dest/source1, source2-value, s, e*
CM:c-f-power-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest and source1* and *source2* fields. The total length of the *dest and source1* field in this format is $2(s+e+1)$. The total length of the *source2* field in this format is $s+e+1$.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected. *test-flag* is set if zero is raised to a non-positive power; otherwise it is cleared.

Context

This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow source1[k]^{source2[k]}$
    if $source1[k] = 0.0$ and $source2[k].real \leq 0.0$
    and $source2[k].imag = 0.0$ then
      $test\text{-}flag[k] \leftarrow 1$
    else $test\text{-}flag[k] \leftarrow 0$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent), using exp and ln operations.

The result is stored into the memory field *dest.* The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

C-S-POWER

Raises a complex number to a signed integer power.

Formats

CM:c-s-power-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:c-s-power-2-2L	*dest*/*source1*, *source2*, *slen2*, *s*, *e*
CM:c-s-power-constant-2-1L	*dest*/*source1*, *source2-value*, *s*, *e*
CM:c-s-power-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex base field.

source2 The field ID of the signed integer exponent field.

source2-value A signed integer immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $2(s+e+1)$.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected. *test-flag* is set if zero is raised to a negative power; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow$ *source1*$[k]^{source2[k]}$
    if *source1*$[k] = 0.0$ and *source2*$[k] < 0$ then
      test-flag$[k] \leftarrow 1$
    else *test-flag*$[k] \leftarrow 0$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent), using repeated multiplications.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

C-U-POWER

Raises a complex number to an unsigned integer power.

Formats

CM:c-u-power-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:c-u-power-2-2L	*dest/source1*, *source2*, *slen2*, *s*, *e*
CM:c-u-power-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-u-power-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex base field.

source2 The field ID of the unsigned integer exponent field.

source2-value An unsigned integer immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $2(s+e+1)$.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{desk}[k] \leftarrow \textit{source1}[k]^{\textit{source2}[k]}$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent), using repeated multiplications.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

F-F-POWER

Raises a floating-point number to a floating-point power.

Formats

CM:f-f-power-2-1L	*dest/source1, source2, s, e*
CM:f-f-power-3-1L	*dest, source1, source2, s, e*
CM:f-f-power-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-f-power-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest — The field ID of the floating-point destination field.

source1 — The field ID of the floating-point base field.

source2 — The field ID of the floating-point exponent field.

source2-value — A floating-point immediate operand to be used as the exponent.

s, e — The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap — The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags — *test-flag* is set if the base is negative and the exponent is non-zero, or if the base is zero and the exponent is non-positive; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context — This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] = 0$ then
      if *source2*$[k] \leq 0$ then
        dest$[k] \leftarrow 0$
        test-flag$[k] \leftarrow 1$
      else
        dest$[k] \leftarrow 0$
        test-flag$[k] \leftarrow 0$
    else if *source1*$[k] < 0$ then

if $source2[k] = 0$ then
  $dest[k] \leftarrow 1.0$
  $test[k] \leftarrow 0$
else
  $dest[k] \leftarrow \langle\text{undefined}\rangle$
  $test\text{-}flag[k] \leftarrow 1$
else
  $dest[k] \leftarrow \exp(source2[k] \times \ln source1[k])$
  $test\text{-}flag[k] \leftarrow 0$
  if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent).

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-S-POWER

Raises a floating-point number to a signed integer power.

Formats

CM:f-s-power-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:f-s-power-2-2L	*dest*/*source1*, *source2*, *slen2*, *s*, *e*
CM:f-s-power-constant-2-1L	*dest*/*source1*, *source2-value*, *s*, *e*
CM:f-s-power-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source1 — The field ID of the floating-point base field.

source2 — The field ID of the signed integer exponent field.

source2-value — A signed integer immediate operand to be used as the second source.

s, *e* — The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 — The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap — The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags — *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

```
if context-flag[k] = 1 then
  if source2[k] < 0 then
    let temp1_k = 1.0/source1[k]
    let temp2_k = -source2[k]
  else
    let temp1_k = source1[k]
    let temp2_k = source2[k]
  if temp2_k⟨0⟩ = 0 then
    dest[k] ← 1.0
  else
```

$dest[k] \leftarrow temp1_k$
for j from 1 to $slen2 - 1$ do
if $temp2_k\langle j : slen2 - 1\rangle \neq 0$ then let $temp1_k = temp1_k \times temp1_k$
if $temp2_k\langle j\rangle$ then $dest[k] \leftarrow dest[k] \times temp1_k$
if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent).

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

F-U-POWER

Raises a floating-point number to an unsigned integer power.

Formats

CM:f-u-power-3-2L	*dest, source1, source2, slen2, s, e*
CM:f-u-power-2-2L	*dest/source1, source2, slen2, s, e*
CM:f-u-power-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-u-power-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point base field.

source2 The field ID of the unsigned integer exponent field.

source2-value An unsigned integer immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $\textit{temp}_k = \textit{source1}[k]$
    if $(\textit{slen2} = 0) \vee (\textit{source2}[k]\langle 0\rangle = 0)$ then
      $\textit{dest}[k] \leftarrow 1.0$
    else
      $\textit{dest}[k] \leftarrow \textit{temp}_k$
    for j from 1 to $\textit{slen2} - 1$ do
      if $\textit{source2}[k]\langle j : \textit{slen2} - 1\rangle \neq 0$ then let $\textit{temp}_k = \textit{temp}_k \times \textit{temp}_k$
      if $\textit{source2}[k]\langle j\rangle$ then $\textit{dest}[k] \leftarrow \textit{dest}[k] \times \textit{temp}_k$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent).

The result is stored into the memory field *dest.* The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

S-S-POWER

Raises a signed integer to a signed integer power.

Formats

CM:s-s-power-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-s-power-2-1L	*dest/source1*, *source2*, *len*
CM:s-s-power-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-s-power-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:s-s-power-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:s-s-power-constant-3-2L	*dest*, *source1*, *source2-value*, *dlen*, *slen*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer base field.

source2 The field ID of the signed integer exponent field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-s-power-3-3L and CM:s-s-power-constant-3-2L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen For CM:s-s-power-constant-3-2L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-s-power-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-s-power-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if zero is raised to a negative power; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if *context-flag*$[k] = 1$ then
  if *source2*$[k] < 0$ then
    if *source1*$[k] = 1$ then *dest*$[k] \leftarrow 1$
    else *dest*$[k] \leftarrow 0$
  else if *source2*$[k] = 0$ then
    dest$[k] \leftarrow 1$
  else
  dest$[k] \leftarrow (\mathit{source1}[k])^{\mathit{source2}[k]}$
  if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
  else *overflow-flag*$[k] \leftarrow 0$

The *source1* field (the base) is raised to the power *source2* (the exponent). If the exponent is negative, the result is always 0; if the exponent is zero, the result is always 1.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

S-U-POWER

Raises a signed integer to a unsigned integer power.

Formats

CM:s-u-power-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-u-power-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:s-u-power-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:s-u-power-constant-3-2L	*dest*, *source1*, *source2-value*, *dlen*, *slen1*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer base field.

source2 The field ID of the unsigned integer exponent field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-u-power-3-3L and CM:s-u-power-constant-3-2L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-u-power-3-3L and CM:s-u-power-constant-3-2L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-u-power-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, *source1* must be either disjoint from or identical to the *dest* field while *source2* must be disjoint from the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if *context-flag*$[k] = 1$ then
  if *source2*$[k] = 0$ then
    $dest[k] \leftarrow 1$
  else
  $dest[k] \leftarrow (source1[k])^{source2[k]}$
  if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
  else *overflow-flag*$[k] \leftarrow 0$

The *source1* field (the base) is raised to the power *source2* (the exponent). If the exponent is zero, the result is always 1.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-S-POWER

Raises a unsigned integer to a signed integer power.

Formats

CM:u-s-power-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-s-power-constant-2-1L	*dest/source1, source2-value, len*
CM:u-s-power-constant-3-1L	*dest, source1, source2-value, len*
CM:u-s-power-constant-3-2L	*dest, source1, source2-value, dlen, slen1*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer base field.

source2 The field ID of the signed integer exponent field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:u-s-power-3-3L and CM:u-s-power-constant-3-2L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-s-power-3-3L and CM:u-s-power-constant-3-2L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-s-power-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, *source1* must be either disjoint from or identical to the *dest* field while *source2* must be disjoint from the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if zero is raised to a negative power; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
test-flag$[k] \leftarrow 0$
if *source1*$[k] = 0$ then
test-flag$[k] \leftarrow 1$
if *source2*$[k] < 0$ then
$dest[k] \leftarrow \left\lfloor 1 \div source1[k]^{|source2[k]|} \right\rfloor$
else if *source2*$[k] = 0$ then
$dest[k] \leftarrow 1$
else
$dest[k] \leftarrow (source1[k])^{source2[k]}$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$

The *source1* field (the base) is raised to the power *source2* (the exponent). If the exponent is negative, the result is the truncation of the reciprocal of *source1* raised to the absolute value of *source2*. If the exponent is zero, the result is always 1.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit. If, in any particular processor, an attempt is made to raise zero to a negative power, the test flag in that processor is set.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-U-POWER

Raises an unsigned integer to an unsigned integer power.

Formats

CM:u-u-power-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-u-power-2-1L	*dest/source1, source2, len*
CM:u-u-power-3-1L	*dest, source1, source2, len*
CM:u-u-power-constant-2-1L	*dest/source1, source2-value, len*
CM:u-u-power-constant-3-1L	*dest, source1, source2-value, len*
CM:u-u-power-constant-3-2L	*dest, source1, source2-value, dlen, slen1*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer base field.

source2 The field ID of the unsigned integer exponent field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-u-power-3-3L and CM:u-u-power-constant-3-2L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-u-power-3-3L and CM:u-u-power-constant-3-2L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-u-power-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
if *source2*$[k] = 0$ then
$dest[k] \leftarrow 1$
else
$dest[k] \leftarrow (source1[k])^{source2[k]}$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$

The *source1* field (the base) is raised to the power *source2* (the exponent). If the exponent is zero, the result is always 1.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

POWER-UP

This operation resets the Nexus, causing all front-end computers to become logically detached from the Connection Machine system.

Formats CM:power-up

Context This operation is unconditional. It does not depend on the *context-flag*.

This function resets the state of the Nexus, causing all front-end computers to become logically detached from the Connection Machine system. When a Connection Machine system is first powered up or is to be completely reset for other reasons, this is the first operation to perform. Any of the front-end computers may be used to do it.

If users on other front-end computers are actively using the Connection Machine system, their computations will be disrupted. Normally all the front-end computers are connected not only through the Connection Machine Nexus but also through some sort of communications network; a front end that executes CM:power-up will attempt to send messages through this network to the other front-end computers on the same Nexus indicating that a CM:power-up operation is being performed.

F-RANDOM

Stores a pseudo-randomly generated floating-point number into the destination field.

Formats CM:f-random-1L *dest*, *s*, *e*

Operands	*dest*	The field ID of the floating-point destination field.
	s, *e*	The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.
Context	This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.	

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then

$$dest[k] \leftarrow \frac{\langle \text{pseudo-random choice of some } j,\ +0 \leq j < 2^{len} \rangle}{2^{len}}$$

where *len* is the length of the destination field.

Into the destination field of each selected processor is stored a floating-point number pseudo-randomly chosen from a uniform distribution between zero (inclusive) and one (exclusive).

The seed for the Paris random number generator is automaticaly initialized the first time the random number generator is called. A value derived from the system clock is used. It is nonetheless possible to explicitly initialize the random number generator by call CM:initialize-random-generator.

Note: Less simple but more flexible random number generation routines are provided as part of the CM Scientific Subroutines Library (CMSSL). For instance, the CMSSL random number generators may be checkpointed to guard against accidental interuptions.

U-RANDOM

Stores a pseudo-randomly generated unsigned integer into the destination field.

Formats CM:u-random-1L *dest, len, limit*

Operands

dest	The field ID of the unsigned integer destination field.
len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
limit	An unsigned integer immediate operand to be used as the exclusive upper bound on values to be generated.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \langle$pseudo-random choice of some j, $0 \leq j < limit\rangle$

The *dest* field in each selected processor receives a pseudo-randomly chosen from a uniform distribution ranging from zero (inclusive) to the specified limit (exclusive).

F-RANK

The destination field in every selected processor receives the rank of that processor's key among all keys in the scan set for that processor.

Formats

CM:f-rank-2L *dest*, *source*, *axis*, *dlen*, *s*, *e*, *direction*, *smode*, *sbit*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the floating-point source field. This is the sort key.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
dlen	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be no larger than the value returned by CM:geometry-coordinate-length.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.
direction	Either :upward or :downward.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *source* and *sbit* fields must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-set}(g, k, \textit{axis}, \textit{direction}, \textit{smode}, \textit{sbit})$
    case *direction* of
      :upward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] < \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k] \wedge m <$
      :downward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] > \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k] \wedge m >$
    $\textit{dest}[k] \leftarrow |L_k|$

where *scan-set* is as defined on page 44.

See section 5.20 on page 42 for a general description of scan sets and the effect of the *axis*, *direction*, *smode*, and *sbit* operands.

This operation determines the ordering necessary to sort the *source* fields within each scan set. It does not not actually move the data so as to sort it, but merely indicates where the data should be moved so as to sort it. A stable ranking is guaranteed. That is, two identical keys will be ranked in the order in which they occur in the *source* field.

In more detail: The *dest* field in each selected processor receives, as an unsigned integer, the rank of that processor's key within the set of keys in the scan set for that processor. This rank may be used to calculate a send address a CM:send operation may then be used to put the data into sorted order. (An advantage of decoupling the rank determination from the reordering process is that the data to be moved may be much larger than the key that determines the ordering, and indeed it may be desirable to reorder the other data but not the key itself. In this way ranking and reordering each need operate only on the relevant data.)

The way in which the rank operation uses scan sets has one unusual twist: A rank that is partitioned into scan sets restarts the rank *ordering* within each scan set (or segment). However, the rank *indices* assigned are not restarted within each scan set.

Specifically, along the entire *axis* specified, only one processor receives a rank index of 0. Rank indices in the first scan set (segment) begin at 0 and run through $n-1$, where n is the number of active processors in the scan set; ranks in the second segment begin at n; and so forth. Thus, the smallest key in the first scan set has rank 0, the next smallest has rank 1; the smallest key in the second scan set has rank n, the next smallest has rank $n+1$, and so on. Within each scan set the ranking index assigned to any given processor determines the rank of that processor's key value relative to the keys of all other active processors within that scan set. The non-repeating indices produce correctly sorted values when used by a send operation either along the entire axis (the scan subclass) or within one or more segments (the scan sets).

This operation was originally documented to result in a set of indexes that restart at 0 for each segment. To obtain that effect use the following strategy:

1) Use the rank function.

2) Set the context bit on for processors with segment bits and then call CM:my-news-address.

3) Use a segmented copy-scan operation to copy the NEWS address within each segment.

4) Subtract the results of the segmented copy scan from the results of the rank ordering.

S-RANK

The destination field in every selected processor receives the rank of that processor's key among all keys in the scan set for that processor.

Formats CM:s-rank-2L *dest*, *source*, *axis*, *dlen*, *slen*, *direction*, *smode*, *sbit*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the signed integer source field. This is the sort key.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
dlen	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be no larger than the value returned by CM:geometry-coordinate-length.
slen	The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *source* and *sbit* fields must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $S_k = scan\text{-}set(g, k, axis, direction, smode, sbit)$
    case *direction* of
      :upward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((source[m] < source[k]) \vee (source[m] = source[k] \wedge m <$
      :downward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((source[m] > source[k]) \vee (source[m] = source[k] \wedge m >$
    $dest[k] \leftarrow |L_k|$

where *scan-set* is as defined on page 44.

See section 5.20 on page 42 for a general description of scan sets and the effect of the *axis*, *direction*, *smode*, and *sbit* operands.

This operation determines the ordering necessary to sort the *source* fields within each scan set. It does not not actually move the data so as to sort it, but merely indicates where the data should be moved so as to sort it. A stable ranking is guaranteed. That is, two identical keys will be ranked in the order in which they occur in the *source* field.

In more detail: The *dest* field in each selected processor receives, as an unsigned integer, the rank of that processor's key within the set of keys in the scan set for that processor. This rank may be used to calculate a send address a CM:send operation may then be used to put the data into sorted order. (An advantage of decoupling the rank determination from the reordering process is that the data to be moved may be much larger than the key that determines the ordering, and indeed it may be desirable to reorder the other data but not the key itself. In this way ranking and reordering each need operate only on the relevant data.)

The way in which the rank operation uses scan sets has one unusual twist: A rank that is partitioned into scan sets restarts the rank *ordering* within each scan set (or segment). However, the rank *indices* assigned are not restarted within each scan set.

Specifically, along the entire *axis* specified, only one processor receives a rank index of 0. Rank indices in the first scan set (segment) begin at 0 and run through $n - 1$, where n is the number of active processors in the scan set; ranks in the second segment begin at n; and so forth. Thus, the smallest key in the first scan set has rank 0, the next smallest has rank 1; the smallest key in the second scan set has rank n, the next smallest has rank $n + 1$, and so on. Within each scan set the ranking index assigned to any given processor determines the rank of that processor's key value relative to the keys of all other active processors within that scan set. The non-repeating indices produce correctly sorted values when used by a send operation either along the entire axis (the scan subclass) or within one or more segments (the scan sets).

U-RANK

The destination field in every selected processor receives the rank of that processor's key among all keys in the scan set for that processor.

Formats CM:u-rank-2L *dest*, *source*, *axis*, *dlen*, *slen*, *direction*, *smode*, *sbit*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field. This is the sort key.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be no larger than the value returned by CM:geometry-coordinate-length.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $S_k = \mathit{scan\text{-}set}(g, k, \mathit{axis}, \mathit{direction}, \mathit{smode}, \mathit{sbit})$
    case *direction* of
      :upward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\mathit{source}[m] < \mathit{source}[k]) \vee (\mathit{source}[m] = \mathit{source}[k] \wedge m <$
      :downward:

$$\text{let } L_k = \{\, m \mid m \in S_k \wedge ((\mathit{source}[m] > \mathit{source}[k]) \vee (\mathit{source}[m] = \mathit{source}[k] \wedge m > k),$$
$$\mathit{dest}[k] \leftarrow |L_k|$$

where *scan-set* is as defined on page 44.

See section 5.20 on page 42 for a general description of scan sets and the effect of the *axis*, *direction*, *smode*, and *sbit* operands.

This operation determines the ordering necessary to sort the *source* fields within each scan set. It does not not actually move the data so as to sort it, but merely indicates where the data should be moved so as to sort it. A stable ranking is guaranteed. That is, two identical keys will be ranked in the order in which they occur in the *source* field.

In more detail: The *dest* field in each selected processor receives, as an unsigned integer, the rank of that processor's key within the set of keys in the scan set for that processor. This rank may be used to calculate a send address a CM:send operation may then be used to put the data into sorted order. (An advantage of decoupling the rank determination from the reordering process is that the data to be moved may be much larger than the key that determines the ordering, and indeed it may be desirable to reorder the other data but not the key itself. In this way ranking and reordering each need operate only on the relevant data.)

The way in which the rank operation uses scan sets has one unusual twist: A rank that is partitioned into scan sets restarts the rank *ordering* within each scan set (or segment). However, the rank *indices* assigned are not restarted within each scan set.

Specifically, along the entire *axis* specified, only one processor receives a rank index of 0. Rank indices in the first scan set (segment) begin at 0 and run through $n-1$, where n is the number of active processors in the scan set; ranks in the second segment begin at n; and so forth. Thus, the smallest key in the first scan set has rank 0, the next smallest has rank 1; the smallest key in the second scan set has rank n, the next smallest has rank $n+1$, and so on. Within each scan set the ranking index assigned to any given processor determines the rank of that processor's key value relative to the keys of all other active processors within that scan set. The non-repeating indices produce correctly sorted values when used by a send operation either along the entire axis (the scan subclass) or within one or more segments (the scan sets).

C-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as complex numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:c-read-from-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, source, s, e, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of complex data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers specifying NEWS axes.

source The field ID of the complex source field.

s, e The significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(s + e + 1)$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Complex number values are copied from the Connection Machine processors to the specified *front-end-array.*

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank*.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if $cm\text{-}axis\text{-}vector[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, from C or Fortran, one of the following predefined complex *format* values may be used: CM_complex_float_single or CM_complex_float_double. For complex data types in C, two front-end elements are used for each Connection Machine element.

When calling Paris from Lisp, the *format* parameter is a keyword argument; for complex transfers, only arrays of type t may be used.

Definition For all i such that $0 \le i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \le m < rank$ do

$$\text{let } s_{\langle i,m\rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$\textit{front-end-array}_{s_{\langle i,0\rangle}, s_{\langle i,1\rangle}, \ldots, s_{\langle i,rank-1\rangle}} \leftarrow source[k_i]$$

Another formulation:

For all s_0 such that $0 \le s_0 < (end_0 - start_0)$ do

for all s_1 such that $0 \le s_1 < (end_1 - start_1)$ do

for all s_2 such that $0 \le s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \le s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{\textit{offset-vector}_0 + s_0, \textit{offset-vector}_1 + s_1, \ldots, \textit{offset-vector}_{rank-1} + s_{rank-1}} \leftarrow source[k_{s_0, s_1, \ldots, s_{rank-1}}]$$

F-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as floating-point numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:f-read-from-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, source, s, e, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of floating-point data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes.

source The field ID of the floating-point source field.

s, e The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Floating-point number values are transferred from the Connection Machine processors to the specified *array.*

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank.*

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* $\times$ *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if $cm\text{-}axis\text{-}vector[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined floatingpoint *format* values may be used. These are CM_float_single or CM_float_double from C or Fortran, and :float-single or :float-double from Lisp.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type and size of the Connection Machine field.

Definition For all i such that $0 \le i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \le m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$\textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}} \leftarrow source[k_i]$$

Another formulation:

For all s_0 such that $0 \le s_0 < (end_0 - start_0)$ do
  for all s_1 such that $0 \le s_1 < (end_1 - start_1)$ do
    for all s_2 such that $0 \le s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \le s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}} \leftarrow source[k_{s_0, s_1, \ldots, s_{rank-1}}]$$

S-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as signed integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:s-read-from-news-array-1L *front-end-array*, *fe-offset-vector*, *cm-start-vector*, *cm-end-vector*, *cm-axis-vector*, *source*, *len*, *[fe-rank*, *fe-dimension-vector*, *format]*

Operands

front-end-array A front-end array (possibly multidimensional) of signed integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes.

source The field ID of the signed integer source field.

len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Signed integer values are transferred from the Connection Machine processors to the specified *array*.

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

When calling Paris from Lisp, the array may be either a general array (of type t) containing signed integers, or a specialized integer-element array (such as an array of type (unsigned-byte 8)).

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank*.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if $cm\text{-}axis\text{-}vector[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined signed *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined signed format keywords are :8-bit, :16-bit, :32-bit, :2-bit-packed, and :4-bit-packed.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type and size of the Connection Machine field.

Definition For all i such that $0 \leq i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$\textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}} \leftarrow source[k_i]$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}} \leftarrow source[k_{s_0, s_1, \ldots, s_{rank-1}}]$$

U-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as unsigned integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:u-read-from-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, source, len, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of unsigned integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes.

source The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Unsigned integer values are transferred from the Connection Machine processors to the specified *array*.

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank*.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if $cm\text{-}axis\text{-}vector[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined unsigned *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_1_bit_packed, CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined unsigned format keywords are :8-bit, :16-bit, :32-bit, :1-bit-packed, :2-bit-packed, and :4-bit-packed.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the CM field.

Definition For all i such that $0 \leq i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$\textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}} \leftarrow source[k_i]$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do

for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do

for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}} \leftarrow source[k_{s_0, s_1, \ldots, s_{rank-1}}]$$

C-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as a complex number and returns it to the front end.

Formats result ← CM:c-read-from-processor-1L *send-address-value*, *source*, *len*

Operands

send-address-value An immediate operand, the send address of a single particular processor.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(s + e + 1)$.

Result A complex number, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as a floating-point number to the front end.

F-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as a floating-point number and returns it to the front end.

Formats result ← CM:f-read-from-processor-1L *send-address-value*, *source*, *s*, *e*

Operands *send-address-value* An immediate operand, the send address of a single particular processor.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Result A floating-point number, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as a floating-point number to the front end.

S-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as a signed integer and returns it to the front end.

Formats result ← CM:s-read-from-processor-1L *send-address-value*, *source*, *len*

Operands *send-address-value* An immediate operand, the send address of a single particular processor.

source The field ID of the signed integer source field.

len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Result A signed integer, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as a signed integer to the front end.

U-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as an unsigned integer and returns it to the front end.

Formats result ← CM:u-read-from-processor-1L *send-address-value*, *source*, *len*

Operands

send-address-value An immediate operand, the send address of a single particular processor.

source The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as an unsigned integer to the front end.

C-RECIPROCAL

Calculates the reciprocal of a complex number.

Formats CM:c-reciprocal-1-1L *dest/source, s, e*
CM:c-reciprocal-2-1L *dest, source, s, e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating point overflow occurs; otherwise it is unaffected.

test-flag is set if division by zero occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
$\mathit{dest}[k] \leftarrow \frac{1}{\mathit{source}[k]}$

A reciprocal of the complex *source* field is place in the complex *dest* field.

REDUCE-WITH-C-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the complex source fields from all the selected processors in that scan class.

Formats CM:reduce-with-c-add-1L *dest, source, axis, s, e, to-coordinate*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36 of the *Paris Reference Manual.*

See section 5.16 beginning on page 34 for a general description of reduce operations. The CM:reduce-with-c-add operation combines *source* fields by performing complex addition.

The operation CM:reduce-with-c-add-1L differs from CM:spread-with-c-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-F-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the floating-point source fields from all the selected processors in that scan class.

Formats CM:reduce-with-f-add-1L *dest*, *source*, *axis*, *s*, *e*, *to-coordinate*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-f-add operation combines *source* fields by performing floating-point addition.

The operation CM:reduce-with-f-add-1L differs from CM:spread-with-f-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-S-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the signed integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-s-add-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

- *dest* — The field ID of the signed integer destination field.
- *source* — The field ID of the signed integer source field.
- *axis* — An unsigned integer immediate operand to be used as the number of a NEWS axis.
- *len* — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
- *to-coordinate* — An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-s-add operation combines *source* fields by performing signed integer addition.

The operation CM:reduce-with-s-add-1L differs from CM:spread-with-s-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-U-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the unsigned integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-u-add-1L *dest, source, axis, len, to-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $C_k = scan\text{-}subclass(g, k, axis)$
if $extract\text{-}news\text{-}coordinate(g, axis, k) = to\text{-}coordinate$ then

$$dest[k] \leftarrow \left(\sum_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-u-add operation combines *source* fields by performing unsigned integer addition.

The operation CM:reduce-with-u-add-1L differs from CM:spread-with-u-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-COPY

Within each scan class one particular processor (if it is selected) receives a copy of the source value from a particular value within its scan subclass.

Formats CM:reduce-with-copy-1L *dest, source, axis, len, to-coordinate, from-coordinate*

Operands
- *dest* The field ID of the unsigned integer destination field.
- *source* The field ID of the unsigned integer source field.
- *axis* An unsigned integer immediate operand to be used as the number of a NEWS axis.
- *len* The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
- *to-coordinate* An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.
- *from-coordinate* An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class is to be read.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $c = \textit{deposit-news-coordinate}(g, k, \textit{axis}, \textit{from-coordinate})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then
      $\textit{dest}[k] \leftarrow \textit{source}[c]$

where *deposit-news-coordinate* is as defined on page 40.

See section 5.20 on page 42 for a general description of reduce operations.

REDUCE-WITH-LOGAND

Within each scan class one particular processor (if it is selected) receives the bitwise logical AND of the source fields from all the selected processors in that scan class.

Formats CM:reduce-with-logand-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\bigwedge_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-logand operation combines *source* fields by performing bitwise logical AND operations.

The operation CM:reduce-with-logand-1L differs from CM:spread-with-logand-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-LOGIOR

Within each scan class one particular processor (if it is selected) receives the bitwise logical inclusive OR of the source fields from all the selected processors in that scan class.

Formats CM:reduce-with-logior-1L *dest, source, axis, len, to-coordinate*

Operands

dest	The field ID of the destination field.
source	The field ID of the source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
to-coordinate	An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $C_k = scan\text{-}subclass(g, k, axis)$
if $extract\text{-}news\text{-}coordinate(g, axis, k) = to\text{-}coordinate$ then

$$dest[k] \leftarrow \left(\bigvee_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-logior operation combines *source* fields by performing bitwise logical inclusive OR operations.

The operation CM:reduce-with-logior-1L differs from CM:spread-with-logior-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-LOGXOR

Within each scan class one particular processor (if it is selected) receives the bitwise logical exclusive OR of the source fields from all the selected processors in that scan class.

Formats CM:reduce-with-logxor-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\bigoplus_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations.

The operation CM:reduce-with-logxor-1L differs from CM:spread-with-logxor-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-F-MAX

Within each scan class one particular processor (if it is selected) receives the largest of the floating-point source fields from all the selected processors in that scan class.

Formats CM:reduce-with-f-max-1L *dest, source, axis, s, e, to-coordinate*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $context\text{-}flag[k] = 1$ then
  let $g = geometry(current\text{-}vp\text{-}set)$
  let $C_k = scan\text{-}subclass(g, k, axis)$
  if $extract\text{-}news\text{-}coordinate(g, axis, k) = to\text{-}coordinate$ then
   $dest[k] \leftarrow \left(\max_{m \in C_k} source[m]\right)$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-f-max operation combines *source* fields by performing an floating-point maximum operation.

The operation CM:reduce-with-f-max-1L differs from CM:spread-with-f-max-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-S-MAX

Within each scan class one particular processor (if it is selected) receives the largest of the signed integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-s-max-1L *dest, source, axis, len, to-coordinate*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$
    if $extract\text{-}news\text{-}coordinate(g, axis, k) = to\text{-}coordinate$ then

$$dest[k] \leftarrow \left(\max_{m \in C_k} source[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-s-max operation combines *source* fields by performing a signed integer maximum operation.

The operation CM:reduce-with-s-max-1L differs from CM:spread-with-s-max-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-U-MAX

Within each scan class one particular processor (if it is selected) receives the largest of the unsigned integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-u-max-1L *dest, source, axis, len, to-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-u-max operation combines *source* fields by performing an unsigned integer maximum operation.

The operation CM:reduce-with-u-max-1L differs from CM:spread-with-u-max-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-F-MIN

Within each scan class one particular processor (if it is selected) receives the smallest of the floating-point source fields from all the selected processors in that scan class.

Formats CM:reduce-with-f-min-1L *dest*, *source*, *axis*, *s*, *e*, *to-coordinate*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-f-min operation combines *source* fields by performing an floating-point minimum operation.

The operation CM:reduce-with-f-min-1L differs from CM:spread-with-f-min-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-S-MIN

Within each scan class one particular processor (if it is selected) receives the smallest of the signed integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-s-min-1L *dest, source, axis, len, to-coordinate*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
let $C_k = \mathit{scan\text{-}subclass}(g, k, \mathit{axis})$
if $\mathit{extract\text{-}news\text{-}coordinate}(g, \mathit{axis}, k) = \mathit{to\text{-}coordinate}$ then

$$\mathit{dest}[k] \leftarrow \left(\min_{m \in C_k} \mathit{source}[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-s-min operation combines *source* fields by performing a signed integer minimum operation.

The operation CM:reduce-with-s-min-1L differs from CM:spread-with-s-min-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-U-MIN

Within each scan class one particular processor (if it is selected) receives the smallest of the unsigned integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-u-min-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-u-min operation combines *source* fields by performing an unsigned integer minimum operation.

The operation CM:reduce-with-u-min-1L differs from CM:spread-with-u-min-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

F-REM

The remainder from dividing one floating-point source value by another is placed in the destination field.

Formats

CM:f-rem-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-rem-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-rem-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-rem-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field. This is the quotient.

source1 The field ID of the floating-point first source field. This is the dividend.

source2 The field ID of the floating-point second source field. This is the divisor.

source2-value A floating-point immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

test-flag is set if division by zero occurs; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source2}[k] \neq 0$ then
      let $v = \textit{source1}[k]/\textit{source2}[k]$
      if $v > \left\lfloor v + \frac{1}{2} \right\rfloor$ then
        let $n = \lfloor v \rfloor$
      else if $v < \left\lfloor v + \frac{1}{2} \right\rfloor$ then

let $n = \lceil v \rceil$
else if $even(\lfloor v \rfloor)$ then
let $n = \lfloor v \rfloor$
else
let $n = \lceil v \rceil$
$dest[k] \leftarrow source1[k] - source2[k] \times n$
else
$dest[k] \leftarrow$ ⟨unpredictable⟩
$test\text{-}flag[k] \leftarrow 1$
if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The remainder from the *source1* operand when divided by the *source2* operand is calculated treating both as floating-point numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-REM

The remainder from the truncating division of one signed integer by another is placed in the destination field. Overflow is also computed.

Formats

CM:s-rem-2-1L	*dest/source1, source2, len*
CM:s-rem-3-1L	*dest, source1, source2, len*
CM:s-rem-constant-2-1L	*dest/source1, source2-value, len*
CM:s-rem-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer remainder field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $source2[k] = 0$ then
      $dest[k] \leftarrow \langle\text{unpredictable}\rangle$
    else
      $$dest[k] \leftarrow sign(source1[k]) \times \left(|source1[k]| - |source2[k]| \times \left\lfloor \frac{|source1[k]|}{|source2[k]|} \right\rfloor \right)$$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$
    else $overflow\text{-}flag[k] \leftarrow 0$

The remainder resulting from the truncating division of the signed integer *source1* by the signed integer *source2* operand is stored into the *dest* field. The result always has the same sign as the *source1* operand. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The value of the destination is unpredictalbe if the divisor is zero.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-REM

The remainder from the truncating division of one unsigned integer by another is placed in the destination field. Overflow is also computed.

Formats

CM:u-rem-2-1L	*dest/source1, source2, len*
CM:u-rem-3-1L	*dest, source1, source2, len*
CM:u-rem-constant-2-1L	*dest/source1, source2-value, len*
CM:u-rem-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer remainder field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    if $\mathit{source2}[k] = 0$ then
      $\mathit{dest}[k] \leftarrow \langle\text{unpredictable}\rangle$
    else
      $\mathit{dest}[k] \leftarrow \mathit{source1}[k] - \mathit{source2}[k] \times \left\lfloor \frac{\mathit{source1}[k]}{\mathit{source2}[k]} \right\rfloor$
    if $\langle$overflow occurred in processor $k\rangle$ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$
    else $\mathit{overflow\text{-}flag}[k] \leftarrow 0$

The remainder resulting from the truncating division of the unsigned integer *source1* by the unsigned integer *source2* operand is stored into the *dest* field. For unsigned integers this is of course the same as the mod operation.

The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The value of the destination is unpredictable if the divisor is zero.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

REMOVE-FIELD-ALIAS

Removes the specified alias field ID from the field to which it refers, leaving the field intact.

Formats CM:remove-field-alias *alias-id*

Operands *alias-id* An alias field ID. This must be an alias field ID returned by CM:make-field-alias.

Context This operation is unconditional. It does not depend on the *context-flag*.

Removing an alias field ID does not affect the memory field to which it refers.

F-F-ROUND

Rounds each source field value to the nearest integer value and stores the result as a floating-point number in the destination field.

Formats CM:f-f-round-1-1L *dest/source*, *s*, *e*
CM:f-f-round-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow sign(source) \times round(source[k])$

The *source* field, treated as a floating-point number, is rounded to the nearest integer and the result is stored in the *dest* field as a floating-point number.

If the *source* field value is exactly midway between two integers, then it is rounded to the even integer.

S-ROUND

The quotient of two signed integer source values, rounded to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:s-round-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-round-2-1L	*dest/source1, source2, len*
CM:s-round-3-1L	*dest, source1, source2, len*
CM:s-round-constant-2-1L	*dest/source1, source2-value, len*
CM:s-round-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer quotient field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
let $v = \dfrac{source1[k]}{source2[k]}$
if $v > \left\lfloor v + \frac{1}{2} \right\rfloor$ then
$dest[k] \leftarrow \lfloor v \rfloor$
else if $v < \left\lfloor v + \frac{1}{2} \right\rfloor$ then
$dest[k] \leftarrow \lceil v \rceil$
else if $even(\lfloor v \rfloor)$ then
$dest[k] \leftarrow \lfloor v \rfloor$
else
$dest[k] \leftarrow \lceil v \rceil$
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The signed integer *source1* operand is divided by the signed integer *source2* operand. The mathematical quotient, rounded to the nearest integer (or to whichever of two equally near neighbors is even) is stored into the signed integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

S-F-ROUND

Converts floating-point source field values to signed integer values by rounding to the nearest integer.

Formats CM:s-f-round-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands	*dest*	The field ID of the signed integer destination field.
	source	The field ID of the floating-point source field.
	len	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap: The fields *dest* and *source* must not overlap in any manner.

Flags: *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    let $v = \mathit{source}[k]$
    if $v > \left\lfloor v + \frac{1}{2} \right\rfloor$ then
      $\mathit{dest}[k] \leftarrow \lfloor v \rfloor$
    else if $v < \left\lfloor v + \frac{1}{2} \right\rfloor$ then
      $\mathit{dest}[k] \leftarrow \lceil v \rceil$
    else if $\mathit{even}(\lfloor v \rfloor)$ then
      $\mathit{dest}[k] \leftarrow \lfloor v \rfloor$
    else
      $\mathit{dest}[k] \leftarrow \lceil v \rceil$
    if $\langle$overflow occurred in processor $k\rangle$ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$

The *source* field, treated as a floating-point number, is rounded to the nearest integer (to the nearest even integer if its value is equal to an integer plus $\frac{1}{2}$). The result is stored into the *dest* field as a signed integer.

U-ROUND

The quotient of two unsigned integer source values, rounded to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:u-round-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-round-2-1L	*dest/source1, source2, len*
CM:u-round-3-1L	*dest, source1, source2, len*
CM:u-round-constant-2-1L	*dest/source1, source2-value, len*
CM:u-round-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then

$$
\begin{array}{l}
\text{let } v = \dfrac{\mathit{source1}[k]}{\mathit{source2}[k]} \\
\text{if } v > \left\lfloor v + \frac{1}{2} \right\rfloor \text{ then} \\
\quad \mathit{dest}[k] \leftarrow \lfloor v \rfloor \\
\text{else if } v < \left\lfloor v + \frac{1}{2} \right\rfloor \text{ then} \\
\quad \mathit{dest}[k] \leftarrow \lceil v \rceil \\
\text{else if } \mathit{even}(\lfloor v \rfloor) \text{ then} \\
\quad \mathit{dest}[k] \leftarrow \lfloor v \rfloor \\
\text{else} \\
\quad \mathit{dest}[k] \leftarrow \lceil v \rceil \\
\text{if } \langle \text{overflow occurred in processor } k \rangle \text{ then } \mathit{overflow\text{-}flag}[k] \leftarrow 1
\end{array}
$$

The unsigned integer *source1* operand is divided by the unsigned integer *source2* operand. The mathematical quotient, rounded to the nearest integer (or to whichever of two equally near neighbors is even) is stored into the unsigned integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-F-ROUND

Converts the floating-point source field values to unsigned integer values, which are stored in the destination field.

Formats CM:u-f-round-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the floating-point source field.

len The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $dest > \lfloor source \rfloor$ then
      $dest \leftarrow \lfloor source \rfloor$
    else if $dest < \lfloor source \rfloor$ then
      $dest \leftarrow \lceil source \rceil$
    else if $even(\lfloor source \rfloor)$ then
      $dest \leftarrow \lfloor source \rfloor$
    else
      $dest \leftarrow \lceil source \rceil$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The *source* field, treated as a floating-point number, is rounded to the nearest integer (to the nearest even integer if its value is equal to an integer plus $\frac{1}{2}$), which is stored into the *dest* field as an unsigned integer.

F-S-SCALE

In each selected processor, multiplies a floating-point number by a specified power of two and stores the result in the destination.

Formats

CM:f-s-scale-2-2L	*dest/source1*, *source2*, *slen2*, *s*, *e*
CM:f-s-scale-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:f-s-scale-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-s-scale-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field. This is the quantity to be scaled.

source2 The field ID of the signed integer second source field. This is the base-2 logarithm of the scale factor.

source2-value A signed integer immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow \left\lfloor source1[k] \times 2^{source2[k]} \right\rfloor$
if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The operand *source1* is scaled by the power of two specified by *source2*. (This is faster than an equivalent multiplication by a power of two.)

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

F-U-SCALE

Multiplies a floating-point number by a specified power of two and stores the result into the destination.

Formats

CM:f-u-scale-2-2L	*dest/source1*, *source2*, *slen2*, *s*, *e*
CM:f-u-scale-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:f-u-scale-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-u-scale-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field. This is the quantity to be scaled.

source2 The field ID of the unsigned integer second source field. This is the base-2 logarithm of the scale factor.

source2-value An unsigned integer immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \left\lfloor \textit{source1}[k] \times 2^{\textit{source2}[k]} \right\rfloor$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The operand *source1* is scaled by the power of two specified by *source2*. (This is faster than an equivalent multiplication by a power of two.)

The result is stored into the memory field *dest.* The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

SCAN-WITH-C-ADD

The destination field in every selected processor receives the sum of the complex source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-c-add-1L *dest, source, axis, s, e, direction, inclusion, smode, sbit*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then
  let $g = geometry(current\text{-}vp\text{-}set)$
  let $S_k = scan\text{-}subset(g, k, axis, direction, inclusion, smode, sbit)$
  if $|S_k| = 0$ then
    $dest[k] \leftarrow 0$
  else

$$dest[k] \leftarrow \left(\sum_{m \in S_k} source[m] \right)$$

where *scan-subset* is as defined on page 36 of the *Paris Reference Manual.*

See the section beginning on 34 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-c-add operation combines *source* fields by performing complex addition. If the scan subset for a selected processor is empty, then the complex value $+0.0$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-ADD

The destination field in every selected processor receives the sum of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-f-add-1L *dest*, *source*, *axis*, *s*, *e*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
if $|S_k| = 0$ then
$\textit{dest}[k] \leftarrow 0$
else

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-add operation combines *source* fields by performing floating-point addition. If the scan subset for a selected processor is empty, then the floating-point value +0.0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-S-ADD

The destination field in every selected processor receives the sum of the signed integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-s-add-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the signed integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 0$
    else

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-s-add operation combines *source* fields by performing signed integer addition. If the scan subset for a selected processor is empty, then the signed integer value 0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-U-ADD

The destination field in every selected processor receives the sum of the unsigned integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-u-add-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 0$
    else

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-u-add operation combines *source* fields by performing unsigned integer addition. If the scan subset for a selected processor is empty, then the unsigned integer value 0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-COPY

The destination field in every selected processor receives the *first* source field from the processors below or above it in some ordering of the processors.

Formats CM:scan-with-copy-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the destination field.
source	The field ID of the source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $S_k = \mathit{scan\text{-}subset}(g, k, \mathit{axis}, \mathit{direction}, \mathit{inclusion}, \mathit{smode}, \mathit{sbit})$
    if $|S_k| = 0$ then
      $\mathit{dest}[k] \leftarrow 000\ldots000$
    else
      case *direction* of
        :upward : let $m' = \min_{m \in S_k} m$
        :downward : let $m' = \max_{m \in S_k} m$
      $\mathit{dest}[k] \leftarrow \mathit{source}[m']$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-copy operation stores into each processor k the *source* field value from the first processor in the scan subset for processor k (where "first" means the processor with lowest address for an upward scan, or with highest address for a downward scan). Generally speaking, the net effect is to propagate a value from the first processor in a group to all the other processors in the group, although variations on this effect are provided by the various possibilities for the *inclusion* and *smode* arguments.

If the scan subset for a selected processor is empty, then the *dest* field for that processor is set to all zero bits. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-LOGAND

The destination field in every selected processor receives the bitwise logical AND of the source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-logand-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow \texttt{111...111}$
    else

$$\textit{dest}[k] \leftarrow \left(\bigwedge_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:**scan-with-logand** operation combines *source* fields by performing bitwise logical AND operations. If the scan subset for a selected processor is empty, then the unsigned integer value $-2^{\text{len}} - 1$ (all ones) is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-LOGIOR

The destination field in every selected processor receives the bitwise logical inclusive OR of the source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-logior-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 000\ldots000$
    else

$$\textit{dest}[k] \leftarrow \left(\bigvee_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-logior operation combines *source* fields by performing bitwise logical inclusive OR operations. If the scan subset for a selected processor is empty, then the unsigned integer value 0 (all zero bits) is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-LOGXOR

The destination field in every selected processor receives the bitwise logical exclusive OR of the source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-logxor-1L *dest, source, axis, len, direction, inclusion, smode, sbit*

Operands

dest	The field ID of the destination field.
source	The field ID of the source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 000\ldots000$
    else

$$\textit{dest}[k] \leftarrow \left(\bigoplus_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations. If the scan subset for a selected processor is empty, then the unsigned integer value 0 (all zero bits) is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-MAX

The destination field in every selected processor receives the largest of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-f-max-1L *dest*, *source*, *axis*, *s*, *e*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$

if $|S_k| = 0$ then

$\textit{dest}[k] \leftarrow -\infty$

else

$$\textit{dest}[k] \leftarrow \left(\max_{m \in S_k} \textit{source}[m]\right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-max operation combines *source* fields by performing an floating-point maximum operation. If the scan subset for a selected processor is empty, then the floating-point value $-\infty$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-S-MAX

The destination field in every selected processor receives the largest of the signed integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-s-max-1L *dest, source, axis, len, direction, inclusion, smode, sbit*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the signed integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $S_k = scan\text{-}subset(g, k, axis, direction, inclusion, smode, sbit)$
if $|S_k| = 0$ then
$dest[k] \leftarrow -2^{len-1}$
else
$$dest[k] \leftarrow \left(\max_{m \in S_k} source[m]\right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-s-max operation combines *source* fields by performing a signed integer maximum operation. If the scan subset for a selected processor is empty, then the signed integer value -2^{len-1} is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-U-MAX

The destination field in every selected processor receives the largest of the unsigned integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-u-max-1L *dest, source, axis, len, direction, inclusion, smode, sbit*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
if $|S_k| = 0$ then
$\textit{dest}[k] \leftarrow 0$
else

$$\textit{dest}[k] \leftarrow \left(\max_{m \in S_k} \textit{source}[m]\right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-u-max operation combines *source* fields by performing an unsigned integer maximum operation. If the scan subset for a selected processor is empty, then the unsigned integer value 0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-MIN

The destination field in every selected processor receives the smallest of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-f-min-1L *dest*, *source*, *axis*, *s*, *e*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow +\infty$
    else
      $$\textit{dest}[k] \leftarrow \left(\min_{m \in S_k} \textit{source}[m]\right)$$
where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-min operation combines *source* fields by performing an floating-point minimum operation. If the scan subset for a selected processor is empty, then the floating-point value $+\infty$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-S-MIN

The destination field in every selected processor receives the smallest of the signed integer source fields from processors below or above it in some ordering of the processors.

Formats

CM:scan-with-s-min-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the signed integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap

The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context

This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $S_k = scan\text{-}subset(g, k, axis, direction, inclusion, smode, sbit)$
    if $|S_k| = 0$ then
      $dest[k] \leftarrow 2^{len-1} - 1$
    else
      $$dest[k] \leftarrow \left(\min_{m \in S_k} source[m]\right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-s-min operation combines *source* fields by performing a signed integer minimum operation. If the scan subset for a selected processor is empty, then the signed integer value $2^{len-1} - 1$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-U-MIN

The destination field in every selected processor receives the smallest of the unsigned integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-u-min-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

$$
\begin{array}{l}
\text{if } \textit{context-flag}[k] = 1 \text{ then} \\
\quad \text{let } g = \textit{geometry}(\textit{current-vp-set}) \\
\quad \text{let } S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit}) \\
\quad \text{if } |S_k| = 0 \text{ then} \\
\quad\quad \textit{dest}[k] \leftarrow 2^{len} - 1 \\
\quad \text{else} \\
\quad\quad \textit{dest}[k] \leftarrow \left(\min_{m \in S_k} \textit{source}[m] \right)
\end{array}
$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-u-min operation combines *source* fields by performing an unsigned integer minimum operation. If the scan subset for a selected processor is empty, then the unsigned integer value $2^{len} - 1$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-MULTIPLY

The destination field in every selected processor receives the product of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats

CM:scan-with-f-multiply-1L *dest*, *source*, *axis*, *s*, *e*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap

The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context

This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 1$
    else

$$\textit{dest}[k] \leftarrow \left(\prod_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-multiply operation combines *source* fields by performing floating-point multiplication. If the scan subset for a selected processor is empty, then the floating-point value 1.0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SEND

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. If a processor receives more than one message, then the message data received by that processor will be unpredictable.

Formats CM:send-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is stored into the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else if $|S_k| = 1$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \textit{source}[\textit{choice}(S_k)]$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \langle\text{undefined}\rangle$

where the *choice* function arbitrarily but deterministically chooses an element from a set.

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d. Note that, although the *send-address* operand is a field in the current VP set, its value must specify a valid send address for *dest*, which may belong to a different VP set.

The CM:send operation combines multiple incoming messages in an unpredictable manner. This operation may be used when the programmer can guarantee that no processor will receive more than one message. Using this operation when it is appropriate may speed message delivery. The destination area need not be prepared.

SEND-ASET32-U-ADD

Sends a message from every selected processor to a specified destination processor and stores it there, as if by **aset32**, in an array. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected. All incoming messages are combined with the destination array element using unsigned integer addition.

Formats CM:send-aset32-u-add-2L *array*, *send-address*, *source*, *index*, *slen*, *index-len*, *index-limit*

Operands

array The field ID of the destination array field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *slen*.

slen The length of the *source* field. This must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of the destination array.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is combined with the field indicated by *array* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do

  let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

  for every processor k' in S_k do

    if $\textit{index}[k'] < \textit{index-limit}$ then

      let $r = \textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))$

let $m = \left\lfloor \frac{k}{r} \right\rfloor \bmod 32$
let $i = index[k']$
for all j such that $0 \leq j < dlen$ do
  let $temp_k\langle j\rangle = array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle$
let $sum_k = temp_k + source[k']$
for all j such that $0 \leq j < dlen$ do
  $array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle \leftarrow sum_k\langle j\rangle$
else
  ⟨error⟩

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into an array element within processor p_d. Note that in each case the array element to be modified in processor p_d is determined by the value of *index* within p_s, not the value within p_d.

The CM:send-aset32-u-add operation combines incoming messages with unsigned integer addition. To receive the sum of only the messages, the destination *array* should first be cleared in all processors that might receive a message.

SEND-ASET32-LOGIOR

Sends a message from every selected processor to a specified destination processor and stores it there, as if by **aset32**, in an array. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected. All incoming messages are combined with the destination array element using bitwise logical inclusive OR.

Formats CM:send-aset32-logior-2L *array*, *send-address*, *source*, *index*, *slen*, *index-len*, *index-limit*

Operands

array The field ID of the destination array field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *slen*.

slen The length of the *source* field. This must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of the destination array.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is combined with the field indicated by *array* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do
  let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
  for every processor k' in S_k do
    if $\textit{index}[k'] < \textit{index-limit}$ then
      let $r = \textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))$

let $m = \left\lfloor \frac{k}{r} \right\rfloor \bmod 32$
let $i = index[k']$
for all j such that $0 \leq j < dlen$ do
  let $q = k - m \times r + (j \bmod 32) \times r$
  let $b = 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)$
  $array[q]\langle b \rangle \leftarrow array[q]\langle b \rangle \vee source[k']\langle j \rangle$
else
  $\langle$error$\rangle$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into an array element within processor p_d. Note that in each case the array element to be modified in processor p_d is determined by the value of *index* within p_s, not the value within p_d.

The CM:send-aset32-logior operation combines incoming messages with a bitwise logical inclusive OR operation. To receive the logical inclusive OR of only the messages, the destination *array* should first be cleared in all processors that might receive a message.

SEND-ASET32-OVERWRITE

Sends a message from every selected processor to a specified destination processor and stores it there, as if by **aset32**, in an array. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected. If a processor receives more than one message destinated for the same array element, then one is stored in that array element and the rest are discarded.

Formats

CM:send-aset32-overwrite-2L *array, send-address, source, index, slen, index-len, index-limit*

Operands

array The field ID of the destination array field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *slen*.

slen The length of the *source* field. This must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of the destination array.

Overlap

The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context

This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is combined with the field indicated by *array* regardless of the *context-flag* of the receiving processor.

Definition

For every virtual processor k in the *current-vp-set* do
  let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
  let $k' = \textit{choice}(S_k)$
  if $\textit{index}[k'] < \textit{index-limit}$ then
    let $r = \textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))$

let $m = \left\lfloor \frac{k}{r} \right\rfloor \bmod 32$
let $i = index[k']$
for all j such that $0 \le j < dlen$ do
$array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle \leftarrow source[k']\langle j\rangle$
else
⟨error⟩

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into an array element within processor p_d. Note that in each case the array element to be modified in processor p_d is determined by the value of *index* within p_s, not the value within p_d.

The CM:send-aset32-overwrite operation will store one of the messages sent to a particular array element, discarding all other messages as well as the original contents of that array element in the receiving processor.

SEND-TO-NEWS

Each processor sends a message to a neighboring processor along a specified NEWS axis.

Formats

CM:send-to-news-1L	*dest, source, axis, direction, len*
CM:send-to-news-always-1L	*dest, source, axis, direction, len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional, but whether data is copied depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is stored into the field indicated by *dest* regardless of the *context-flag* of the receiving processor.

Note that in the conditional case the storing of data depends only on the *context-flag* of the processor sending the data, not on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
 if (always or $context\text{-}flag[k] = 1$) then
  let $g = geometry(current\text{-}vp\text{-}set)$
  $dest[news\text{-}neighbor(g, k, axis, direction)] \leftarrow source[k]$

The *source* field in each processor is stored into the *dest* field of that processor's neighbor along the NEWS axis specified by *axis* in the direction specified by *direction*.

If *direction* is :upward then each processor stores data into the neighbor whose NEWS coordinate is one greater, with the processor whose coordinate is greatest storing data into the processor whose coordinate is zero.

If *direction* is :downward then each processor stores data into the neighbor whose NEWS coordinate is one less, with the processor whose coordinate is zero storing data into the processor whose coordinate is greatest.

SEND-TO-QUEUE32

Sends a message from every selected processor to a specified destination processor and stores it there, as if by **aset32**, in a queue. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors.

Formats CM:send-to-queue32-1L *dest*, *send-address*, *source*, *slen*, *index-limit*

Operands

dest The field ID of the queue field. The length of this field must accommodate 32 bits for the *queue.count* subfield, plus *index* − *limit* × *slen* bits for the *queue.elements* subfield, where *index-limit* is the number of queue elements in each processor.

send address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

slen The length of the *source* field. This is also the length of each queue element. It is currently restricted to 32 bits.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for a zero-based index into *queue.elements*. The value of this argument must be at least 1 and should never exceed the number of elements that can be stored in the queue.

Overlap The fields *send-address* and *source* may overlap in any manner. No overlap with the *dest* field is allowed.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is queued in the field indicated by *dest* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k\,\}$

let T_k be a sub-set of S_k where $|T_k| = \min(|S_k| + \textit{queue.count}, \textit{index-limit})$

for i from *queue.count* to $\textit{queue.count} + |T_k| - 1$ do

$\textit{queue.elements}[i] \leftarrow T_k[i]$

$\textit{queue.count} \leftarrow \textit{queue.count} + |S_k|$

Note that if $(|S_k| + \textit{queue.count} > \textit{index-limit})$ then there is some choice in picking the elements of T_k.

The destination field is treated as two subfields: *queue.count* and *queue.elements*. *Queue.count* is 32 bits long and records the number of enqueued messages. *Queue.elements* stores the enqueued messages; it is formatted as a slicewise array (accessed using aref32 and aset32), and starts at an offset of 32 bits from the start of the destination field. Its length is a multiple of the message length: at least *index-limit* $\times$ *slen* and possibly greater.

The *index-limit* argument specifies the maximum number of elements that any processor's *queue.elements* subfield may accumulate. If any processor receives more messages than this specified number, the queue overflows and messages are lost. If a *queue.elements* subfield overflows, the *queue.count* subfield for that processor nonetheless accurately reflects the number of messages received.

For any given communication pattern, both the order of message queueing and the selection of messages preserved or discarded in case of queue overflow are deterministic. That is, the order and selection of enqueued messages can be predictably reproduced from one invocation to the next.

This determinism is especially important for applications that use successive CM:send-to-queue32-1L calls to send large data structures by breaking up them up into chunks of length *slen*. By holding the *send-address* argument constant, such applications can send successive chunks of *slen* bits each to corresponding queues.

To prepare an empty queue for a CM:send-to-queue-1L instruction, the *queue.count* subfield should be set to zero. From Lisp/Paris, this is done by executing the following code in the destination context:

```
(let   ((zeros (allocate-stack-field 32))
  (context-hold (allocate-stack-field 1)))
  (cm:move-constant-always zeros 0 32)
  (cm:store-context context-hold)
  (cm:set-context)
  (cm:aset32-2L zeros queue zeros 32 32 1)
  (cm:load-context context-hold)
)
```

The CM:send-to-queue32-1L operation is conditional on the context of the source field; the set of queues that will *receive* messages is independent of the currently active set. To zero the *queue.count* subfield in only those queues that are to receive messages, execute the following code in the source context:

```
(let   ((zeros (allocate-stack-field 32)))
  (cm:move-constant-always zeros 0 32)
  (cm:send-aset32-overwrite-2L queue dest zeros zeros 32 32 1)
)
```

After the CM:send-to-queue32 operation, the local count can be retrieved by executing the following code in the destination context:

```
(let   ((zeros (allocate-stack-field 32)))
  (count-field (allocate-stack-field 32))
  )
  (cm:move-constant-always zeros 0 32)
  (cm:aref32-2L count-field queue zeros 32 32 1)
)
```

The $i^(th)$ message can be retrieved from *queue.elements* by executing the following code in the destination context:

```
(let   ((index (allocate-stack-field 32))
  (data-field (allocate-stack-field message-length))
  )
  (cm:move-constant-always index i 32)
  (cm:aref32-2L data-field (+ 32 queue) index len 32 queue-size)
)
```

Note that *queue.elements* is offset from the queue field by 32 bits.

An artificially small queue size may be used by passing CM:send-to-queue-1L an index-limit value that is less than the number of elements of length slen that could be stored in the *queue.elements* portion of the destination field. If this is done, the queues will be partially filled. However, the correct queue size should always be used as the index-limit argument to CM:aref32-2L when reading elements from the queue.

SEND-WITH-C-ADD

Sends a message from every selected processor to a destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using complex addition.

Formats CM:send-with-c-add-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest	The field ID of the complex destination field.
send-address	The field ID of the send address field. For each processor, this indicates to which processor a message is sent.
source	The field ID of the complex source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.
notify	The field ID of the notification bit (a one-bit field).

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition Let $P = \{\, m \mid 0 \le m \le \text{CM:*user-send-address-limit*} \,\}$
For every virtual processor k in *vp-set*(*dest*) do
let $S_k = \{\, m \mid m \in P \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose absolute send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:**send-with-c-add** operation adds incoming messages to the *dest* field, treating all quantities as complex numbers. To receive the sum of only the messages, the destination area should initially be set to zero in all processors that might receive a message.

SEND-WITH-F-ADD

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using floating-point addition.

Formats CM:send-with-f-add-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest The field ID of the floating-point destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-f-add operation adds incoming messages together with the *dest* field as floating-point numbers. To receive the sum of only the messages, the destination area should first be set to zero in all processors that might receive a message.

SEND-WITH-S-ADD

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using signed integer addition.

Formats CM:send-with-s-add-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the signed integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-s-add operation adds incoming messages into the *dest* field as signed integers. Carry-out and arithmetic overflow are not detected. To receive the sum of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-U-ADD

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using unsigned integer addition.

Formats CM:send-with-u-add-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the unsigned integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the unsigned integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
$$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-u-add operation adds incoming messages into the *dest* field as unsigned integers. Carry-out and arithmetic overflow are not detected. To receive the sum of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-LOGAND

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using bitwise logical AND.

Formats CM:send-with-logand-1L *dest, send-address, source, len, notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] \wedge \left(\bigwedge_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-logand operation will combine all messages and the original contents of the destination field with a bitwise logical AND operation. To receive the logical AND of only the messages, the destination area should first be set to all-ones in all processors that might receive a message.

SEND-WITH-LOGIOR

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using bitwise logical inclusive OR.

Formats CM:send-with-logior-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap

The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context

This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
if $|S_k| = 0$ then
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] \vee \left(\bigvee_{m \in S_k} \textit{source}[m] \right)$$

SEND-WITH-LOGIOR

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-logior operation combines incoming messages with a bitwise logical inclusive OR operation. To receive the logical inclusive OR of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-LOGXOR

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using bitwise logical exclusive OR.

Formats CM:send-with-logxor-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap

The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context

This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $$\textit{dest}[k] \leftarrow \textit{dest}[k] \oplus \left(\bigoplus_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-logxor operation is similar but combines incoming messages with a bitwise logical EXCLUSIVE OR operation. To receive the logical EXCLUSIVE OR of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-F-MAX

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a floating-point maximum operation.

Formats CM:send-with-f-max-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest The field ID of the floating-point destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \mathit{current\text{-}vp\text{-}set} \land \mathit{context\text{-}flag}[m] = 1 \land \mathit{send\text{-}address}[m] = k \,\}$
if $|S_k| = 0$ then
  if $\mathit{notify}[k] \not\equiv$ CM:*no-field* then $\mathit{notify}[k] \leftarrow 0$
else
  if $\mathit{notify}[k] \not\equiv$ CM:*no-field* then $\mathit{notify}[k] \leftarrow 1$
  $$\mathit{dest}[k] \leftarrow \max\left(\mathit{dest}[k], \max_{m \in S_k} \mathit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-f-max operation combines incoming messages with the *dest* field using floating-point maximum operations. The *test-flag* is not affected by the maximum operation.

To receive the maximum of only the messages, the destination field should first be set to the smallest possible value: $-\infty$.

SEND-WITH-S-MAX

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a signed integer maximum operation.

Formats CM:send-with-s-max-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the signed integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \max\left(\textit{dest}[k], \max_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-s-max operation combines incoming messages with the *dest* field using signed integer maximum operations. The *test-flag* is not affected by the maximum operation.

To receive the maximum of only the messages, the destination field should first be set to athe smallest possible value: -2^{len-1}.

SEND-WITH-U-MAX

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using an unsigned integer maximum operation.

Formats CM:send-with-u-max-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the unsigned integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the unsigned integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
$$\textit{dest}[k] \leftarrow \max\left(\textit{dest}[k], \max_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-u-max operation combines incoming messages with the *dest* field using unsigned integer maximum operations. The *test-flag* is not affected by the maximum operation.

To receive the maximum of only the messages, the destination field should first be set to the smallest possible value: zero.

SEND-WITH-F-MIN

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a floating-point minimum operation.

Formats CM:send-with-f-min-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest The field ID of the floating-point destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \min\left(\textit{dest}[k], \min_{m \in S_k} \textit{source}[m]\right)$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-f-min operation combines incoming messages with the *dest* field using floating-point minimum operations. The *test-flag* is not affected by the minimum operation.

To receive the minimum of only the messages, the destination field should first be set to the largest value possible: $+\infty$.

SEND-WITH-S-MIN

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a signed integer minimum operation.

Formats CM:send-with-s-min-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the signed integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \min\left(\textit{dest}[k], \min_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-s-min operation combines incoming messages with the *dest* field using signed integer minimum operations. The *test-flag* is not affected by the minimum operation.

To receive the minimum of only the messages, the destination field should first be set to the largest possible value: $2^{len-1} - 1$.

SEND-WITH-U-MIN

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using an unsigned integer minimum operation.

Formats CM:send-with-u-min-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the unsigned integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the unsigned integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \min\left(\textit{dest}[k], \min_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-u-min operation combines incoming messages with the *dest* field using unsigned integer minimum operations. The *test-flag* is not affected by the minimum operation.

To receive the minimum of only the messages, the destination field should first be set to the largest possible value: $2^{len} - 1$.

SEND-WITH-OVERWRITE

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. If a processor receives more than one message, then one is delivered and the rest are discarded.

Formats CM:send-with-overwrite-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is stored into the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \textit{source}[\textit{choice}(S_k)]$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-overwrite operation will store one of the messages sent, discarding all other messages as well as the original contents of the *dest* field in the receiving processor.

SET-BIT

Sets a specified memory bit.

Formats CM:set-bit *dest*
CM:set-bit-always *dest*

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow 1$

The destination memory bit is set within each selected processor.

SET-CONTEXT

Unconditionally makes all processors active.

Formats CM:set-context

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$context\text{-}flag[k] \leftarrow 1$

Within each processor, the context bit for that processor is unconditionally set.

SET-FIELD-ALIAS-VP-SET

Sets the VP set of the specified alias fieldID to the specified VP set.

Formats CM:set-field-alias-vp-set *alias-id*, *vp-set*

Operands	*alias-id*	An alias field ID. This must be an alias fieldID returned by CM:make-field-alias. This alias id need not be in the current VP set.
	vp-set	A VP set ID. This need not be the current VP set.
Context	This operation is unconditional. It does not depend on the *context-flag*.	

This function sets the VP set of *alias-id* to *vp-set*.

An error is signaled if the physical length of the aliased field is not exactly divisible by the VP ratio of *vp-set*. (See the definitions of CM:make-field-alias for more information about the physical length of an aliased field.)

SET-SAFETY-MODE

Formats CM:set-safety-mode *safety-mode*

Operands *safety-mode* An unsigned integer, the safety level. Currently only the values 0 and 1 are meaningful.

Context This operation is unconditional. It does not depend on the *context-flag*.

The safety mode is set to the specified value. A non-zero value indicates that the Paris interface should perform various extra error checks and consistency checks that may be helpful in detecting bugs in user programs. Of course, the price of these error checks is reduced execution speed.

SET-SYSTEM-LEDS-MODE

Formats CM:set-system-leds-mode *leds-mode*

Operands *leds-mode* Either :leds-off, :leds-on, :leds-throb, :leds-diagnostics, :leds-perfmon, :leds-sync, or :leds-blink-sync.

Context This operation is unconditional. It does not depend on the *context-flag*.

The lights on the front and back of the Connection Machine system cabinet can be controlled in a variety of ways. The **cm:set-system-leds-mode** operation selects what information will be displayed in the lights. If the specified *leds-mode* is :leds-off, then all the lights are turned off, and thereafter the user operations **cm:latch-leds** and **cm:latch-leds-always** may be used to control the lights. Other values for *leds-mode* select one of the system-supplied display modes. (The operations **cm:latch-leds** and **cm:latch-leds-always** may still be used when in a system-supplied display mode, but the user-specified pattern is unlikely to persist as it may be immediately altered by the system, depending on the mode.)

The names of the possible modes shown above are for the C/Paris and Fortran/Paris interfaces. Through an accident of history, the names for the leds modes are different in the Lisp/Paris interface:

C and Fortran	Lisp
CM_leds_off	nil
CM_leds_on	t
CM_leds_throb	:throb
CM_leds_diagnostics	:diagnostics
CM_leds_perfmon	:performance-monitor
CM_leds_sync	:synch
CM_leds_blink_sync	:blink-and-synch

C'est la vie.

SET-VP-SET

Declares a specified VP set to be current.

Formats CM:set-vp-set *vp-set-id*

Operands *vp-set-id* A VP set ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *current-vp-set* ← *vp-set-id*

The VP set specified by the *vp-set-id* becomes the current VP set. Most Paris operations implicitly operate within the virtual processors of the current VP set.

SET-VP-SET-GEOMETRY

Alters the geometry of an existing VP set.

Formats CM:set-vp-set-geometry *vp-set-id*, *geometry-id*

Operands *vp-set-id* A VP set ID.

geometry-id A geometry ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

The VP set specified by the *vp-set-id* is altered so that its geometry is that specified by the *geometry-id*. The new geometry must have the same total number of elements (product of axis lengths) as the old geometry.

SET-flag

Sets a specified flag bit.

Formats CM:set-test
CM:set-overflow

Context This operation is conditional.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
flag$[k] \leftarrow 1$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, the indicated flag for that processor is set.

S-S-SHIFT

Shifts a signed integer by an amount specified by a signed integer.

Formats CM:s-s-shift-2-2L *dest/source1, source2, dlen, slen2*
CM:s-s-shift-constant-3-2L *dest, source1, source2-value, dlen, slen1*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field. This is the quantity to be shifted.

source2 The field ID of the signed integer second source field. This is the shift distance (positive for a left shift, negative for a right shift).

source2-value A signed integer immediate operand to be used as the second source. The same shift distance is applied to each *source1* value.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-s-shift-2-2L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-s-shift-constant-3-2L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
$dest[k] \leftarrow \left\lfloor source1[k] \times 2^{source2[k]} \right\rfloor$
if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$
else $\textit{overflow-flag}[k] \leftarrow 0$

The operand *source1* is shifted by the number of bit positions specified by *source2*, where a positive shift distance indicates a left shift (that is, a shift toward more significant bit positions) and a negative shift distance indicates a right shift (that is, a shift toward less significant bit positions). A left shift introduces zero bits into the vacated (least significant) bit positions; a right shift introduces copies of the sign bit into the vacated (most significant) bit positions. This operation is sometimes called an *arithmetic shift.*

The result is stored into the memory field *dest.* The various operand formats allow the second source operand to be either a memory field or a constant. In the non-constant version the destination field initially contains one source operand.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *dlen.*

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

U-S-SHIFT

Shifts an unsigned integer by an amount specified by a signed integer.

Formats

CM:u-s-shift-2-2L *dest/source1, source2, dlen, slen2*
CM:u-s-shift-constant-3-2L *dest, source1, source2-value, dlen, slen1*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field. This is the quantity to be shifted.

source2 The field ID of the signed integer second source field. This is the shift distance (positive for a left shift, negative for a right shift.)

source2-value A signed integer immediate operand to be used as the second source. The same shift distance is applied to each *source1* value.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-s-shift-2-2L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:u-s-shift-constant-3-2L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $context\text{-}flag[k] = 1$ then
  $dest[k] \leftarrow \left\lfloor source1[k] \times 2^{source2[k]} \right\rfloor$
  if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$
  else $overflow\text{-}flag[k] \leftarrow 0$

The operand *source1* is shifted by the number of bit positions specified by *source2*, where a positive shift distance indicates a left shift (that is, a shift toward more significant bit positions) and a negative shift distance indicates a right shift (that is, a shift toward less significant bit positions). Zero-valued bits are introduced into the vacated bit positions (least significant for a left shift, most significant for a right shift). This operation is sometimes called a *logical shift*.

The result is stored into the memory field *dest*. The various operand formats allow the second source operand to be either a memory field or a constant. In the non-constant version, the destination field initially contains one source operand.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *dlen*.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

C-C-SIGNUM

The signum of the complex source field is stored in the complex destination field.

Formats CM:c-c-signum-1-1L *dest/source*, *s*, *e*
CM:c-c-signum-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
$\mathit{dest}[k] \leftarrow \mathit{signum}(\mathit{source}[k])$

The signum of a complex number is a complex number of the same phase but with unit magnitude, unless the numer is a complex zero, in which case the result is a complex zero.

F-F-SIGNUM

Determines whether the floating-point source field is negative, minus zero, plus zero, or positive and places the value `-1.0`, `+0.0`, -0.0, or `1.0` in the destination field accordingly.

Formats

CM:f-f-signum-1-1L *dest/source*, *s*, *e*
CM:f-f-signum-2-1L *dest*, *source*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source — The field ID of the floating-point source field.

s, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $source[k] < 0$ then $dest[k] \leftarrow -1.0$
    else if $source[k] > 0$ then $dest[k] \leftarrow 1.0$
    else $dest[k] \leftarrow source[k]$

The signum function of the *source* operand is placed in the *dest* operand. The result is `-1.0`, `-0.0`, `+0.0`, or `1.0` thus indicating whether the source value is negative, minus zero, plus zero, or positive, respectively. If the *source* operand is a NaN, then it is copied unchanged.

S-F-SIGNUM

Determines whether the floating-point source field is negative, zero, or positive and places the value -1, 0, or 1 in the destination field accordingly.

Formats CM:s-f-signum-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the floating-point source field.
dlen	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $source[k] < 0$ then $dest[k] \leftarrow -1$
    else if $source[k] > 0$ then $dest[k] \leftarrow 1$
    else $dest[k] \leftarrow 0$

The signum function of the *source* operand is placed in the *dest* operand. The result is -1, 0, or 1 according to whether the source value is negative (but non-zero), zero ($+0$ or -0), or positive (but non-zero), respectively.

S-S-SIGNUM

Determines whether the signed integer source field is negative, zero, or positive and places the value `-1`, `0`, or `1` in the destination field accordingly.

Formats

CM:s-s-signum-1-1L *dest/source*, *len*
CM:s-s-signum-2-1L *dest*, *source*, *len*
CM:s-s-signum-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context

This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source}[k] < 0$ then $\textit{dest}[k] \leftarrow -1$
    else if $\textit{source}[k] > 0$ then $\textit{dest}[k] \leftarrow 1$
    else $\textit{dest}[k] \leftarrow 0$

The signum function of the *source* operand is placed in the *dest* operand. The result is `-1`, `0`, or `1` according to whether the source value is negative, zero, or positive, respectively.

C-SIN

The sine of the complex source field is placed in the complex destination field.

Formats CM:c-sin-1-1L *dest*/*source*, *s*, *e*
CM:c-sin-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \sin \textit{source}[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$

The sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-SIN

Calculates the floating-point sine of the source field values and stores the result in the floating-point destination field.

Formats CM:f-sin-1-1L *dest/source, s, e*
CM:f-sin-2-1L *dest, source, s, e*

Operands
dest The field ID of the floating-point destination field.
source The field ID of the floating-point source field.
s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \sin$ *source*$[k]$

The sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

C-SINH

The hyperbolic sine of the complex source field is placed in the complex destination field.

Formats CM:c-sinh-1-1L *dest/source*, *s*, *e*
CM:c-sinh-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the complex destination field.
source	The field ID of the complex source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \sinh$ *source*$[k]$

The hyperbolic sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-SINH

Calculates the floating-point hyperbolic sine of the source field values and stores the result in the floating-point destination field.

Formats CM:f-sinh-1-1L *dest/source*, *s*, *e*
CM:f-sinh-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags: *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \sinh \textit{source}[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$

The hyperbolic sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

SPREAD-FROM-PROCESSOR

A single source processor is specified. A copy of its source field value is spread to every (selected) processor in the destination field. Neither the destination nor the source field needs to be in the current VP set.

Formats

CM:spread-from-processor-1L *dest*, *send-address-value*, *source*, *len*
CM:spread-from-processor-a-1L *dest*, *send-address-value*, *source*, *len*

Operands

dest The field ID of the destination field.

send-address-value An unsigned integer immediate operand to be used as the the send address of the processor whose *source* value is to be spread.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operations are conditional.

The always operations are unconditional.
For this instruction, -a is used instead of the standard -always suffix to indicate unconditional operation.

Definition For every virtual processor k in *vp-set*(*dest*) do
if (always or $\textit{context-flag}[k] = 1$) then
$\textit{dest}[k] \leftarrow \textit{source}[\textit{send-address-value}]$

The value of the source field in the processor specified by *send-address-value* is spread to all (selected) processors in the destination field. The source and destination fields may reside in different VP sets.

SPREAD-WITH-C-ADD

The destination field in every selected processor receives the sum of the complex source fields from processors below or above it in some ordering of the processors.

Formats CM:spread-with-c-add-1L *dest*, *source*, *axis*, *s*, *e*

Operands	*dest*	The field ID of the complex destination field.
	source	The field ID of the complex source field.
	axis	An unsigned integer immediate operand to be used as the the number of a NEWS axis.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\mathit{context\text{-}flag}[k] = 1$ then

let $C_k = \mathit{scan\text{-}subclass}(k, \{\, \mathit{axis} \,\})$

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \mathit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36 of the *Paris Reference Manual.*

See the section beginning on page 36 for a general description of spread operations. The CM:spread-with-c-add operation combines *source* fields by performing complex addition.

A call to CM:spread-with-c-add-1L is equivalent to the sequence

CM:scan-with-c-add-1L *dest*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *source*, *axis*, $2 \times (s+e+1)$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-F-ADD

The destination field in every selected processor receives the sum of the floating-point source fields from all processors in its scan subclass.

Formats CM:spread-with-f-add-1L *dest*, *source*, *axis*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-f-add operation combines *source* fields by performing floating-point addition.

A call to CM:spread-with-f-add-1L is equivalent to the sequence

CM:scan-with-f-add-1L *temp*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, $s + e + 1$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-S-ADD

The destination field in every selected processor receives the sum of the signed integer source fields from all processors in its scan subclass.

Formats CM:spread-with-s-add-1L *dest*, *source*, *axis*, *len*

Operands	*dest*	The field ID of the signed integer destination field.
	source	The field ID of the signed integer source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-s-add operation combines *source* fields by performing signed integer addition.

A call to CM:spread-with-s-add-1L is equivalent to the sequence

CM:scan-with-s-add-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-U-ADD

The destination field in every selected processor receives the sum of the unsigned integer source fields from all processors in its scan subclass.

Formats CM:spread-with-u-add-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$

$$dest[k] \leftarrow \left(\sum_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-u-add operation combines *source* fields by performing unsigned integer addition.

A call to CM:spread-with-u-add-1L is equivalent to the sequence

CM:scan-with-u-add-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-COPY

The destination field in every selected processor receives a copy of the source value from a particular value within its scan subclass.

Formats CM:spread-with-copy-1L *dest, source, axis, len, coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class is to be replicated.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $c = \textit{deposit-news-constant}(g, k, \textit{axis}, \textit{coordinate-value})$
    $\textit{dest}[k] \leftarrow \textit{source}[c]$

where *deposit-news-constant* is defined in the dictionary entry for CM:deposit-news-coordinate.

See section 5.20 on page 42 for a general description of spread operations.

SPREAD-WITH-LOGAND

The destination field in every selected processor receives the bitwise logical AND of the source fields from all processors in its scan subclass.

Formats CM:spread-with-logand-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the destination field.
source	The field ID of the source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $C_k = scan\text{-}subclass(g, k, axis)$

$$dest[k] \leftarrow \left(\bigwedge_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-logand operation combines *source* fields by performing bitwise logical AND operations.

A call to CM:spread-with-logand-1L is equivalent to the sequence

CM:scan-with-logand-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-LOGIOR

The destination field in every selected processor receives the bitwise logical inclusive OR of the source fields from all processors in its scan subclass.

Formats CM:spread-with-logior-1L *dest*, *source*, *axis*, *len*

Operands	*dest*	The field ID of the destination field.
	source	The field ID of the source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $C_k = scan\text{-}subclass(g, k, axis)$

$$dest[k] \leftarrow \left(\bigvee_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-logior operation combines *source* fields by performing bitwise logical inclusive OR operations.

A call to CM:spread-with-logior-1L is equivalent to the sequence

CM:scan-with-logior-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-LOGXOR

The destination field in every selected processor receives the bitwise logical exclusive OR of the source fields from all processors in its scan subclass.

Formats CM:spread-with-logxor-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the destination field.
source	The field ID of the source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\bigoplus_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations.

A call to CM:spread-with-logxor-1L is equivalent to the sequence

CM:scan-with-logxor-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-F-MAX

The destination field in every selected processor receives the largest of the floating-point source fields from all processors in its scan subclass.

Formats CM:spread-with-f-max-1L *dest*, *source*, *axis*, *s*, *e*

Operands	*dest*	The field ID of the floating-point destination field.
	source	The field ID of the floating-point source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$
    $dest[k] \leftarrow \left(\max_{m \in C_k} source[m]\right)$
where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-f-max operation combines *source* fields by performing an floating-point maximum operation.

A call to CM:spread-with-f-max-1L is equivalent to the sequence

CM:scan-with-f-max-1L *temp*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, $s + e + 1$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-S-MAX

The destination field in every selected processor receives the largest of the signed integer source fields from all processors in its scan subclass.

Formats CM:spread-with-s-max-1L *dest*, *source*, *axis*, *len*

Operands	*dest*	The field ID of the signed integer destination field.
	source	The field ID of the signed integer source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    $$\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m]\right)$$
  where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-s-max operation combines *source* fields by performing a signed integer maximum operation.

A call to CM:spread-with-s-max-1L is equivalent to the sequence

CM:scan-with-s-max-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-U-MAX

The destination field in every selected processor receives the largest of the unsigned integer source fields from all processors in its scan subclass.

Formats CM:spread-with-u-max-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $C_k = scan\text{-}subclass(g, k, axis)$

$$dest[k] \leftarrow \left(\max_{m \in C_k} source[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-u-max operation combines *source* fields by performing an unsigned integer maximum operation.

A call to CM:spread-with-u-max-1L is equivalent to the sequence

CM:scan-with-u-max-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-F-MIN

The destination field in every selected processor receives the smallest of the floating-point source fields from all processors in its scan subclass.

Formats CM:spread-with-f-min-1L *dest*, *source*, *axis*, *s*, *e*

Operands	*dest*	The field ID of the floating-point destination field.
	source	The field ID of the floating-point source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\textit{context-flag}[k] = 1$ then
  let $g = \textit{geometry}(\textit{current-vp-set})$
  let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
  $$\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m]\right)$$
 where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-f-min operation combines *source* fields by performing an floating-point minimum operation.

A call to CM:spread-with-f-min-1L is equivalent to the sequence

CM:scan-with-f-min-1L *temp*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, $s + e + 1$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-S-MIN

The destination field in every selected processor receives the smallest of the signed integer source fields from all processors in its scan subclass.

Formats CM:spread-with-s-min-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the signed integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*[k] = 1 then
let $g = geometry(current\text{-}vp\text{-}set)$
let $C_k = scan\text{-}subclass(g, k, axis)$

$$dest[k] \leftarrow \left(\min_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-s-min operation combines *source* fields by performing a signed integer minimum operation.

A call to CM:spread-with-s-min-1L is equivalent to the sequence

CM:scan-with-s-min-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-U-MIN

The destination field in every selected processor receives the smallest of the unsigned integer source fields from all processors in its scan subclass.

Formats CM:spread-with-u-min-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$
    $dest[k] \leftarrow \left(\min_{m \in C_k} source[m]\right)$
where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-u-min operation combines *source* fields by performing an unsigned integer minimum operation.

A call to CM:spread-with-u-min-1L is equivalent to the sequence

CM:scan-with-u-min-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

C-SQRT

Calculates the square root of the complex source field and places it in the complex destination field.

Formats CM:c-sqrt-1-1L *dest*/*source*, *s*, *e*
CM:c-sqrt-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
$dest[k] \leftarrow \sqrt{source}$

In each selected processor, the square root of the *source* field value is placed in the *dest* field.

F-SQRT

Calculates the floating-point square root of the source field values and stores the result in the floating-point destination field.

Formats

CM:f-sqrt-1-1L *dest/source*, *s*, *e*
CM:f-sqrt-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is negative and non-zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    if $\mathit{source}[k] > 0$ then
      $\mathit{dest}[k] \leftarrow \sqrt{\mathit{source}[k]}$
    else if $\mathit{source}[k] = \pm 0$ then
      $\mathit{dest}[k] \leftarrow \mathit{source}[k]$
    else if $: \mathit{source} : [k] < 0$ then
      $\mathit{dest}[k] \leftarrow \langle\text{unpredictable}\rangle$
      $\mathit{test}[k] \leftarrow 1$

If the *source* value is non-negative, then the square root of that value is placed in the destination. The square root of -0 is defined to be -0.

If the *source* operand is a NaN, then it is copied to the *dest* field unchanged.

STORE-CONTEXT

Unconditionally stores the context bit into memory.

Formats CM:store-context *dest*

Operands *dest* The field ID of the destination bit (a one-bit field).

Context This operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
$dest[k] \leftarrow context\text{-}flag[k]$

Within each processor, the context bit for that processor is unconditionally stored into memory.

STORE-flag

Conditionally stores a flag bit into memory.

Formats

CM:store-test	*dest*
CM:store-test-always	*dest*
CM:store-overflow	*dest*
CM:store-overflow-always	*dest*

Operands *dest* The field ID of the destination bit (a one-bit field).

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    $\mathit{dest}[k] \leftarrow \mathit{flag}[k]$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, the indicated flag for that processor is stored into memory.

FE-STRUCTURE-ARRAY-FORMAT

This instruction returns an array format descriptor for a particular slot in an array of structures. A format descriptor may be passed to any array transfer instruction to specify a front-end array format, although this is not required. See also CM:fe-array-format and CM:fe-packed-array-format.

This instruction is not provided for the Lisp interface to Paris.

Formats result ← CM:fe-structure-array-format *cm-element-byte-size*, *structure-byte-size*

Operands *cm-element-byte-size* A signed integer immediate operand to be used as the number of bytes each Connection Machine element occupies in the front-end array. This must be a power of two between 1 and 16.

structure-byte-size A signed integer immediate operand to be used as the length of the front-end structure in bytes. This may be any positive integer.

Result The array format descriptor specified.

Context This is a front-end operation. It does not depend on the value of the *context-flag*.

The return value is a format descriptor for a front-end array of structures. Such a format descriptor can be passed to any of the CM array transfer instructions in order to allow transfers in either direction between CM fields and a front-end array of structures. If this is done, one CM element per selected processor is copied into, or receives data from, the specified slot across an array of structures on the front end.

Values for both *cm-element-byte-size* and *cm-structure-byte-size* may be obtained by calls to sizeof(...).

The value of *cm-element-byte-size* specifies the length of the structure slot in bytes. It also defines the unit of measure for the *fe-offset-vector* argument to the CM:read-from-news-array and CM:write-to-news-array instructions.

The value of *structure-byte-size* specifies the length of the entire stucture in bytes. It also defines the unit of measure for the argument *fe-dimension-vector* to the CM:read-from-news-array and CM:write-to-news-array instructions.

If a slot other than the first slot in the front-end structure is the destination of a CM:read-from-news-array or the source for a CM:write-to-news-array transfer instruction, then a pointer to that slot must be provided as the value of *front-end-array*. This is a bit tricky. The pointer must identify the location of the chosen slot in the structure that is the first element of the array of structures.

Here is an example in C.

```
#define n_foos 256

/* declare array of structure foo */
struct foo { int a; double b; char c; } fooarray[n_foos];

/* declare the format */
CM_array_format_t foo_format;

/* declare an offset for the 'b' slot of struct foo */
/* this is a pointer to a double - b is a double */
double *bslot_pointer;

/* lots of other declarations etc. in here */
...

/* create format descriptor for foo.b */
foo_format = CM_structure_array_format(sizeof(double), sizeof(struct foo));

/* create pointer offset to slot b of struct foo */
bslot_pointer = &fooarray[0].b;

/* store src-field values in slot b of each foo struct in foo_array */
/* all variables xxxx_vector should be self explanatory */

CM_f_read_from_news_array_1L(bslot_pointer, offset_vector,
                             start_vector, end_vector, axis_vector,
                             src_field, 23, 8, rank,
                             dimension_vector, foo_format);
```

Slot b of each foo structure in the array foo_array receives a copy of the value stored in the corresponding CM *src-field* processor.

The value of bslot_pointer is a pointer to the b slot of the first foo structure in foo_array. Given this starting place, foo_format indicates how many bytes must be skipped between b slots.

For further examples, refer to the manual entitled *Introduction to Programming in C/Paris.*

F-SUBF-CONST-MULT

Calculates a value $(b - a)x$ and places it in the destination.

Formats

CM:f-subf-const-mult-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-subf-const-mult-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-subf-const-mult-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-subf-const-mult-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source (subtrahend) field.

source2-value A floating-point immediate operand to be used as the second source (minuend).

source3 The field ID of the floating-point third source (multiplier) field.

source3-value A floating-point immediate operand to be used as the third source (multiplier).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow (source2\text{-}value[k] - source1[k]) \times source3[k]$
 if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

The operand *source1* is subtracted from *source2-value*, treating them as floating-point numbers, and then the difference is multiplied by a third operand *source3*. The result is stored in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-subf-const-mult-1L is equivalent to the sequence

CM:f-subfrom-constant-3-1L *dest*, *source1*, *source2-value*, *s*, *e*
CM:f-multiply-3-1L *dest*, *dest*, *source3*, *s*, *e*

but may be faster.

F-SUB-MULT

Calculates a value $(x - a)b$ and places it in the destination.

Formats

CM:f-sub-mult-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-sub-mult-always-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-sub-const-mult-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-sub-const-mult-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-sub-mult-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-sub-mult-const-always-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-sub-const-mult-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-sub-const-mult-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source (minuend) field.

source2 The field ID of the floating-point second source (subtrahend) field.

source2-value A floating-point immediate operand to be used as the second source (subtrahend).

source3 The field ID of the floating-point third source (multiplier) field.

source3-value A floating-point immediate operand to be used as the third source (multiplier).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition

For every virtual processor k in the *current-vp-set* do
 if (always or $context\text{-}flag[k] = 1$) then
  $dest[k] \leftarrow (source1[k] - source2[k]) \times source3[k]$
 if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

The operand *source2* is subtracted from *source1*, treating them as floating-point numbers, and then the difference is multiplied by a third operand *source3*. The result is stored in the destination field.

The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

A call to CM:f-sub-mult-1L is equivalent to the sequence

CM:f-subtract-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-multiply-3-1L *dest*, *temp*, *source3*, *s*, *e*

but may be faster.

C-SUBTRACT

The difference of two complex source values is placed in the destination field.

Formats

CM:c-subtract-2-1L	*dest/source1, source2, s, e*
CM:c-subtract-always-2-1L	*dest/source1, source2, s, e*
CM:c-subtract-3-1L	*dest, source1, source2, s, e*
CM:c-subtract-always-3-1L	*dest, source1, source2, s, e*
CM:c-subtract-constant-2-1L	*dest/source1, source2-value, s, e*
CM:c-subtract-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:c-subtract-constant-3-1L	*dest, source1, source2-value, s, e*
CM:c-subtract-const-always-3-1L	*dest, source1, source2-value, s, e*
CM:c-subfrom-2-1L	*dest/source2, source1, s, e*
CM:c-subfrom-always-2-1L	*dest/source2, source1, s, e*
CM:c-subfrom-constant-2-1L	*dest/source2, source1-value, s, e*
CM:c-subfrom-const-always-2-1L	*dest/source2, source1-value, s, e*
CM:c-subfrom-constant-3-1L	*dest, source2, source1-value, s, e*
CM:c-subfrom-const-always-3-1L	*dest, source2, source1-value, s, e*

Operands

dest The field ID of the complex destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the complex first source field. This is the minuend.

source2 The field ID of the complex second source field. This is the subtrahend.

source1-value A complex immediate operand to be used as the first source.

source2-value A complex immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
$\textit{dest}[k] \leftarrow \textit{source1}[k] - \textit{source2}[k]$
if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$

The operand *source2* is subtracted from *source1*, treated as as complex numbers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The constant operand *source1-value* or *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-SUBTRACT

The difference of two floating-point source values is placed in the destination field.

Formats

CM:f-subtract-2-1L	*dest/source1, source2, s, e*
CM:f-subtract-always-2-1L	*dest/source1, source2, s, e*
CM:f-subtract-3-1L	*dest, source1, source2, s, e*
CM:f-subtract-always-3-1L	*dest, source1, source2, s, e*
CM:f-subtract-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-subtract-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:f-subtract-constant-3-1L	*dest, source1, source2-value, s, e*
CM:f-subtract-const-always-3-1L	*dest, source1, source2-value, s, e*
CM:f-subfrom-2-1L	*dest/source2, source1, s, e*
CM:f-subfrom-always-2-1L	*dest/source2, source1, s, e*
CM:f-subfrom-constant-2-1L	*dest/source2, source1-value, s, e*
CM:f-subfrom-const-always-2-1L	*dest/source2, source1-value, s, e*
CM:f-subfrom-constant-3-1L	*dest, source2, source1-value, s, e*
CM:f-subfrom-const-always-3-1L	*dest, source2, source1-value, s, e*

Operands

dest The field ID of the floating-point destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the floating-point first source field. This is the minuend.

source2 The field ID of the floating-point second source field. This is the subtrahend.

source1-value A floating-point immediate operand to be used as the first source.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or *context-flag*$[k] = 1$) then
dest$[k] \leftarrow$ *source1*$[k]$ − *source2*$[k]$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The operand *source2* is subtracted from *source1*, treated as as floating-point numbers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The constant operand *source1-value* or *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-SUBTRACT

The difference of two signed integer source values is placed in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats

CM:s-subtract-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-subtract-2-1L	*dest/source1, source2, len*
CM:s-subtract-3-1L	*dest, source1, source2, len*
CM:s-subtract-constant-2-1L	*dest/source1, source2-value, len*
CM:s-subtract-constant-3-1L	*dest, source1, source2-value, len*
CM:s-subfrom-2-1L	*dest/source2, source1, len*
CM:s-subfrom-constant-2-1L	*dest/source2, source1-value, len*
CM:s-subfrom-constant-3-1L	*dest, source2, source1-value, len*

Operands

dest The field ID of the signed integer destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the signed integer first source field. This is the minuend.

source2 The field ID of the signed integer second source field. This is the subtrahend.

source1-value A signed integer immediate operand to be used as the first source.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-subtract-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-subtract-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-subtract-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is equivalent to "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow$ *source1*$[k]$ – *source2*$[k]$
    if $\langle$no borrow needed in processor $k\rangle$ then *carry-flag*$[k] \leftarrow 1$
    else *carry-flag*$[k] \leftarrow 0$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The operand *source2* is subtracted from *source1*, treated as as signed integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The various operand formats allow the first and second source operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source1-value* or *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-SUBTRACT

The difference of two unsigned integer source values is placed in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats

CM:u-subtract-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-subtract-2-1L	*dest/source1*, *source2*, *len*
CM:u-subfrom-2-1L	*dest/source2*, *source1*, *len*
CM:u-subtract-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-subtract-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:u-subfrom-constant-2-1L	*dest/source2*, *source1-value*, *len*
CM:u-subtract-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:u-subfrom-constant-3-1L	*dest*, *source2*, *source1-value*, *len*

Operands

dest The field ID of the unsigned integer destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the unsigned integer first source field. This is the minuend.

source2 The field ID of the unsigned integer second source field. This is the subtrahend.

source1-value An unsigned integer immediate operand to be used as the first source.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-subtract-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-subtract-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-subtract-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is equivalent to "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] ← *source1*[k] – *source2*[k]
    if ⟨no borrow needed in processor k⟩ then *carry-flag*[k] ← 1
    else *carry-flag*[k] ← 0
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0

The operand *source2* is subtracted from *source1*, treated as as unsigned integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source1-value* or *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

S-SUBTRACT-BORROW

In each selected processor, computes the difference of two signed integer source values and places it in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats CM:s-subtract-borrow-3-1L *dest*, *source1*, *source2*, *len*

Operands

dest The field ID of the signed integer destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the signed integer first source field. This is the minuend.

source2 The field ID of the signed integer second source field. This is the subtrahend.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is interpreted as "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

$\textit{dest}[k] \leftarrow \textit{source1}[k] - \textit{source2}[k] + (\textit{carry-flag}[k] - 1)$

if ⟨no borrow needed in processor k⟩ then $\textit{carry-flag}[k] \leftarrow 1$

else $\textit{carry-flag}[k] \leftarrow 0$

if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

else $\textit{overflow-flag}[k] \leftarrow 0$

The operand *source2* is subtracted from *source1*, treated as signed integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

U-SUBTRACT-BORROW

In each selected processor, computes the difference of two unsigned integer source values and places it in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats CM:u-subtract-borrow-3-1L *dest*, *source1*, *source2*, *len*

Operands

dest — The field ID of the unsigned integer destination field. This is the difference, the result of the subtraction operation.

source1 — The field ID of the unsigned integer first source field. This is the minuend.

source2 — The field ID of the unsigned integer second source field. This is the subtrahend.

len — The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is equivalent to "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] $\leftarrow$ *source1*[k] $-$ *source2*[k] $+$ (*carry-flag*[k] $-$ 1)
    if $\langle$no borrow needed in processor $k\rangle$ then *carry-flag*[k] $\leftarrow$ 1
    else *carry-flag*[k] $\leftarrow$ 0
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*[k] $\leftarrow$ 1
    else *overflow-flag*[k] $\leftarrow$ 0

The operand *source2* is subtracted from *source1*, treated as as unsigned integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

SWAP

Swaps the contents of two bit fields.

Formats CM:swap-2-1L *dest1/source1, dest2/source2, len*
CM:swap--always-2-1L *dest1/source1, dest2/source2, len*

Operands

dest1	The field ID of the first destination field.
source1	The field ID of the first source (same as first destination) field.
dest2	The field ID of the second destination field.
source2	The field ID of the second source (same as second destination) field.
len	The length of the *dest1*, *source1*, *dest2*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *dest1* and *dest2* must not overlap in any manner.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
let $temp1_k = source1[k]$
let $temp2_k = source2[k]$
let $dest1[k] \leftarrow temp2_k$
let $dest2[k] \leftarrow temp1_k$

Each of the two provided fields is copied into the other so as to exchange their contents.

C-TAN

Calculates the complex tangent of the source field values and stores the result in the complex destination field.

Formats CM:c-tan-1-1L *dest/source*, *s*, *e*
CM:c-tan-2-1L *dest*, *source*, *s*, *e*

Operands	*dest*	The field ID of the complex destination field.
	source	The field ID of the complex source field.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags: *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \tan$ *source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The tangent of the value of the *source* field is stored into the *dest* field.

F-TAN

Calculates the floating-point tangent of the source field values and stores the result in the floating-point destination field.

Formats CM:f-tan-1-1L *dest/source*, *s*, *e*
CM:f-tan-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \tan$ *source*$[k]$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The tangent of the value of the *source* field is stored into the *dest* field.

C-TANH

Calculates the complex hyperbolic tangent of the source field values and stores the result in the complex destination field.

Formats CM:c-tanh-1-1L *dest/source*, *s*, *e*
CM:c-tanh-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the complex destination field.
source	The field ID of the complex source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \tanh$ *source*

The hyperbolic tangent of the value of the *source* field is stored into the *dest* field.

F-TANH

Calculates the floating-point hyperbolic tangent of the source field values and stores the result in the floating-point destination field.

Formats CM:f-tanh-1-1L *dest/source*, *s*, *e*
CM:f-tanh-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow$ tanh *source*
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

The hyperbolic tangent of the value of the *source* field is stored into the *dest* field.

TIME

Times other operations and reports both the total amount of time elapsed and the amount of time spent executing on the Connection Machine system.

This instruction is available only from the Lisp/Paris interface. For Fortran/Paris and C/Paris users, the equivalent functionality is provided by the CM:timer- series of functions – which may also be used from Lisp. The CM:timer- functions are documented in this dictionary and also in the *CM System User's Guide.*

Formats CM:time *form, [return-statistics-p]*

Operands

form — The a Lisp, Lisp/Paris, or *Lisp form to be timed. This must be a single Lisp expression. To time more than one expression, enclose them in a progn form.

return-statistics-p — The answer to the question, "Do you want timing statistics returned as the value of the macro?". This is an optional keyword argument and defaults to NIL. When specified, the invocation must include the keyword :return-statistics-p followed by T or NIL.

Context — This operation is unconditional. It does not depend on the *context-flag.*

The CM:time facility is a Lisp macro, not a function. It is used in the Lisp/Paris interface to time the execution of other operations on the Connection Machine system.

A call to the CM:time macro may contain a single Lisp expression; this is executed in the normal manner, but before the value is returned, timing information is printed out as for the Common Lisp `time` macro.

Specifying a NIL value to the :return-statistics-p (the default) causes the statistics to be displayed on standard output.

Specifying a T value to the :return-statistics-p causes the statistics to be returned as two floating-point values in a list that is the return value of the macro call.

The first number reported is elapsed time during execution on both the front-end computer and the Connection Machine system. In addition, timing information related to Connection Machine system performance is printed. The second number reported is the amount of that time that the Connection Machine system was actually executing instructions (not waiting for the front end). For optimal performance, the programmer strives to obtain the maximum percentage of Connection Machine utilization possible.

For further information about timing code from the Lisp/Paris interface, see the *CM System User's Guide* chaper entitled "In The Lisp Environment."

TIME

The timing facility is provided in the C/Paris and Fortran/Paris interfaces through a set of functions whose names all begin with CM:timer-.

TIMER

The timing facility. A set of instructions that together determine how much time any part of a program takes to execute on the Connection Machine.

Formats	CM:timer-clear	*timer*
	CM:timer-start	*timer*
	CM:timer-stop	*timer*
	CM:timer-print	*timer*
	CM:timer-read-starts	*timer*
	CM:timer-read-elapsed	*timer*
	CM:timer-read-cm-busy	*timer*
	CM:timer-read-cm-idle	*timer*
	CM:timer-read-run-state	*timer*
	CM:timer-set-starts	*timer*, *int*

Operands	*timer*	The integer used to identify the timer being used.. This must be an unsigned integer immediate operand between 0 (inclusive) and CM*max-number-of-timers* (exclusive).
	int	For CM:timer-set-starts, the start number to which the specified timer is to be reset.

Context This operation is unconditional. It does not depend on the *context-flag.*

To activate multiple timers, assign each an integer identifier. Nested calls to different timers is permitted. Each timer can record timings of up to 43 hours, with microsecond precision.

Four basic operations are required in order to use this timing facility. Use them in the following order:

CM:timer-clear

Sets the total elapsed time, total CM busy time, and number of starts for *timer* to zero.

CM:timer-start

Starts the clock running for *timer.* Elapsed time (also known as wall time) and CM busy time are accumulated. Number of starts is incremented.

CM:timer-stop

Stops the clock running for timer. The specified timer's state variables for CM elapsed time and CM busy time are updated. A subsequent call to CM:timer-start – without an intervening call to CM:timer-clear – restarts the timer and *adds* to the accumulated elapsed and busy values for this timer.

CM:timer-print

Prints information about *timer*, including, but not limited to: the number of starts, the total elapsed time, and the total time that the Connection Machine was busy while this timer was active.

To use a timer, first invoke CM:timer-clear to zero the timer values. Then, call CM:timer-start and CM:timer-stop any number of times. Finally call CM:timer-print.

For each timer, state variables for CM elapsed time and CM busy time are maintained. Elapsed time records how much time has elapsed between each pair of CM:timer-start and CM:timer-stop calls that have been made since CM:timer-clear was last called for *timer*. CM busy time records the total time the CM has spent being active between each pair of CM:timer-start and CM:timer-stop calls that have been made since CM:timer-clear was last called for *timer*.

The following five functions return state values for a specified timer:

CM:timer-read-starts

Returns an unsigned integer, the number of times CM:timer-start has been called for this timer.

CM:timer-read-elapsed

Returns the total elapsed time, in seconds, accumulated while *timer* was running.

CM:timer-read-cm-busy

Returns the total CM busy time, in seconds, accumulated while *timer* was running.

CM:timer-read-cm-idle

Returns the total CM idle time, in seconds, accumulated while *timer* was running. CM idle time is equal to total elapsed time minus the CM busy time.

CM:timer-read-run-state

Returns **TRUE** (or t or 1) if and only if *timer* is running. Otherwise, returns **FALSE** (or nil or 0).

One further operation is provided to reset the number of starts for the specified timer:

CM:timer-set-starts

Sets the number of starts for *timer* to the specified integer value. This is useful in code that stops a timer to query it and then restarts the same timer. CM:timer-set-starts can be used to set the number of starts to 1 less than the actual number of starts before restarting the timer. In this way, querying a timer does not change the number of starts ultimately recorded.

For a detailed guide to using the new timing facility, including information about conditions that affect timing accuracy, see the *CM System User's Guide.*

FE-TO-GRAY-CODE

Converts, on the front end, a nonnegative integer into a bit string representing a Gray-coded integer value.

Formats result ← CM:fe-to-gray-code *integer*

Operands *integer* An unsigned integer immediate operand to be used as the nonnegative integer.

Result An unsigned integer, the Gray code equivalent of *integer*.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Return $integer \oplus \left\lfloor \frac{integer}{2} \right\rfloor$

This function calculates, entirely on the front end, a bit-string encoding in a particular reflected binary Gray code. The position of that value in the standard Gray code sequence is equal to the specified *integer*.

Note that the binary value 0 is always equivalent to a Gray code string that is all 0-bits.

U-TO-GRAY-CODE

Converts an unsigned binary integer to a bit string representing a Gray-coded integer value.

Formats CM:u-to-gray-code-1-1L *dest/source, len*
CM:u-to-gray-code-2-1L *dest, source, len*

Operands

dest The field ID of the destination field.

source The field ID of the unsigned integer source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k]\langle len-1\rangle \leftarrow source[k]\langle len-1\rangle$
    for j from $len-2$ to 0 do
      $dest[k]\langle j\rangle \leftarrow source[k]\langle j\rangle \oplus source[k]\langle j+1\rangle$

The *source* operand is an unsigned binary integer, and is converted to a bit-string value in a particular reflected binary Gray code. The position of that value in the standard Gray code sequence is the *source*.

Note that the binary value 0 is always equivalent to a Gray code string that is all 0-bits.

TRANSPOSE32

Within each cluster of 32 physical processors, for every group of 32 virtual processors in such a cluster, copies one 32-bit field to another. During this copying operation, transposes the data as a 32-by-32 bit matrix. Thus, each virtual processor receives one bit from the source value of each virtual processor in its group of 32.

Formats CM:transpose32-1-1L *dest/source, len*
CM:transpose32-2-1L *dest, source, len*

Operands

source The field ID of the source field.

dest The field ID of the destination field.

len The length of the *source* and *dest* fields. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be a multiple of 32.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. The fields *dest* and *source* may overlap in any manner.

Context This operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
for all j such that $0 \le j < \textit{dlen}$ do
$\textit{dest}[k]\langle j \rangle \leftarrow$
$\textit{source}\left[32r\left\lfloor \frac{k}{32r} \right\rfloor + (k \bmod r) + r(j \bmod 32)\right]\left\langle 32\left\lfloor \frac{j}{32} \right\rfloor + \frac{k \bmod 32}{r} \right\rangle$

where r is the value of CM:*virtual-to-physical-processor-ratio* and j is the bit position in each field.

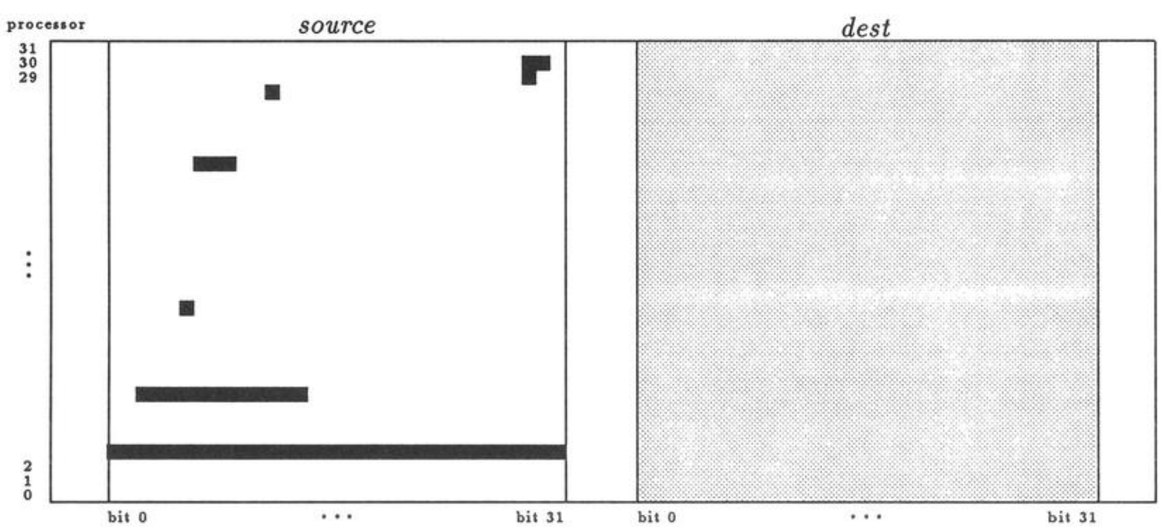
This instruction copies each 32-bit field to the corresponding 32-bit field within each virtual processor. In the course of copying the bits, it "transposes" them so that a 32-bit value lying entirely within the *source* field of one virtual processor is made to occupy a memory slice, that is, one bit in each of 32 virtual processors. The opposite is also true: the 32-bit value that ends up in the *dest* field of a virtual processor is made up of one bit from each of 32 virtual processors. Transposed data is said to be stored in a *slicewise* format.

For the purposes of this instruction, the physical processors are divided into clusters of 32. Two processors are in the same cluster if their physical processor numbers agree in all but the five least significant bits.

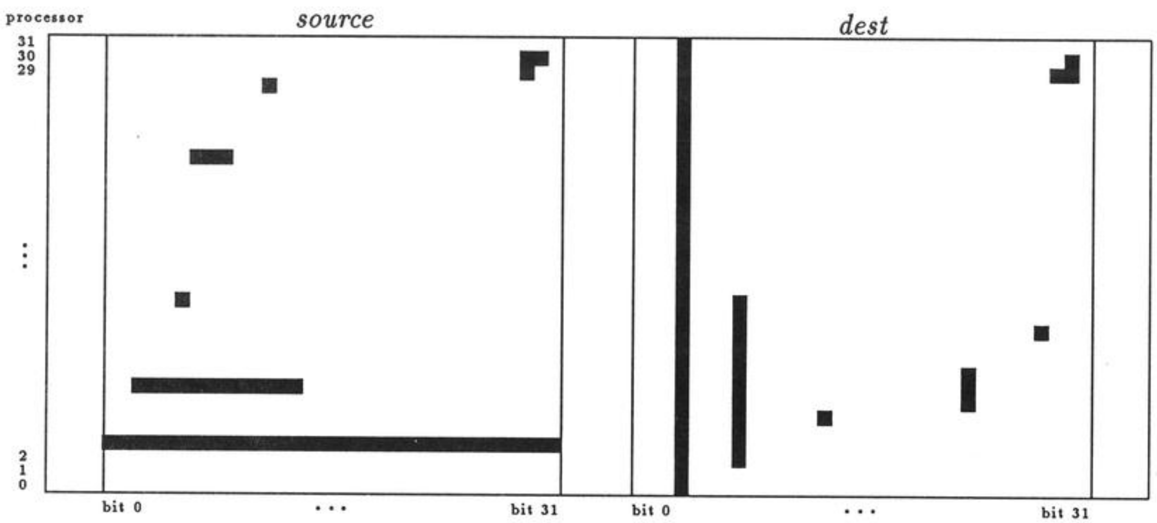
The virtual processors are similarly divided into groups of 32; a group of virtual processors consists of one virtual processor from each physical processor of a cluster, such that the virtual processors occupy the same physical memory locations within their respective physical processors. Thus, two virtual processors are in the same group if their virtual processor numbers agree in all but bit positions n through $n+4$, where n is the number of virtual processors bits in each physical processor.

The CM:transpose32 operation may then be understood as taking the 32 32-bit *source* values from a group of 32 virtual processors as the rows of a 32-by-32 bit matrix, and then storing the columns of this matrix into the *dest* fields of these same virtual processors.

The process may be understood pictorially. Suppose that before the operation the memory of a group of 32 virtual processors looks like this:



Then, after the CM:transpose32 operation, it will look like this:



Knowledge of the internal details of Connection Machine VP memory layout is required to use this instruction properly on *source* values represented in more than 32-bits.

This instruction reorients processor data into a slicewise format that permits rapid, indirect field addressing. A memory region containing transposed data may be viewed either as a single, *shared slicewise array* or as a set of *parallel slicewise arrays.* (See the CM:aref32 and CM:aref32-shared dictionary entries for a description of these data formats.) Viewed as a shared slicewise array, this is especially useful for quickly constructing lookup tables.

Transposition is reversed by applying the CM:transpose32 instruction to a field already stored in the slicewise format. To preserve the correlation between processors and data, this instruction should not be used on slicewise data that was orginally stored by providing CM:aset32 or CM:aset32-shared with an *index-limit* other than 32.

F-F-TRUNCATE

Rounds each source field value to the largest integral value not greater than that value and stores the result as a floating-point number in the destination field.

Formats CM:f-f-truncate-1-1L *dest/source*, *s*, *e*
CM:f-f-truncate-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow sign(source) \times \lfloor |source[k]| \rfloor$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of zero, which is stored into the *dest* field as a floating-point number.

S-F-TRUNCATE

Rounds each floating-point source field value to the largest integer not greater than that value and stores the result as a signed integer in the destination field.

Formats CM:s-f-truncate-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands	*dest*	The field ID of the signed integer destination field.
	source	The field ID of the floating-point source field.
	len	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap: The fields *dest* and *source* must not overlap in any manner.

Flags: *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \textit{sign}(\textit{source}) \times \lfloor|\textit{source}[k]|\rfloor$
  if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$ else $\textit{overflow-flag}[k] \leftarrow 0$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of zero, which is stored into the *dest* field as a signed integer.

S-TRUNCATE

The quotient of two signed integer source values, rounded toward zero to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:s-truncate-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-truncate-2-1L	*dest/source1, source2, len*
CM:s-truncate-3-1L	*dest, source1, source2, len*
CM:s-truncate-constant-2-1L	*dest/source1, source2-value, len*
CM:s-truncate-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer quotient field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-truncate-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-truncate-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-truncate-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if divisor is zero; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

$$
\begin{array}{l}
\text{if } \textit{context-flag}[k] = 1 \text{ then} \\
\quad \text{if } \textit{source2}[k] = 0 \text{ then} \\
\quad\quad \textit{dest}[k] \leftarrow \langle \text{unpredictable} \rangle \\
\quad \text{else} \\
\quad\quad \textit{dest}[k] \leftarrow \textit{sign}(\textit{source1}[k]) \times \textit{sign}(\textit{source2}[k]) \times \left\lfloor \dfrac{|\textit{source1}[k]|}{|\textit{source2}[k]|} \right\rfloor \\
\quad \text{if } \langle \text{overflow occurred in processor } k \rangle \text{ then } \textit{overflow-flag}[k] \leftarrow 1 \\
\quad \text{else } \textit{overflow-flag}[k] \leftarrow 0
\end{array}
$$

The signed integer *source1* operand is divided by the signed integer *source2* operand. The mathematical quotient is truncated towards zero and stored into the signed integer memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-TRUNCATE

The quotient of two unsigned integer source values, rounded toward zero to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:u-truncate-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-truncate-2-1L	*dest*/*source1*, *source2*, *len*
CM:u-truncate-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-truncate-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:u-truncate-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-truncate-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-truncate-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-truncate-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if divisor is zero; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
if *source2*$[k] = 0$ then
dest$[k] \leftarrow \langle$unpredictable$\rangle$
else
$$dest[k] \leftarrow \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$
if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$

The unsigned integer *source1* operand is divided by the unsigned integer *source2* operand. The floor of the mathematical quotient is stored into the unsigned integer memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-F-TRUNCATE

Rounds each source field value to the largest integer not greater than that value and stores the result as an unsigned integer in the destination field.

Formats CM:u-f-truncate-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands	*dest*	The field ID of the unsigned integer destination field.
	source	The field ID of the floating-point source field.
	len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.
Overlap		The fields *dest* and *source* must not overlap in any manner.
Flags		*overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.
Context		This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    $dest \leftarrow sign(source) \times \lfloor |source| \rfloor$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of zero, and the result is stored into the *dest* field as an unsigned integer.

F-VAX-TO-IEEE

Converts the floating-point source field values from VAX floating-point format to IEEE floating-point format and stores the result in the destination field.

Formats CM:f-vax-to-ieee-1L *ieee-dest*, *vax-source*, *len*

Operands *ieee-dest* The field ID of the floating-point destination field.

vax-source The field ID of the floating-point source field.

len The length of the *vax-source* and *ieee-dest* fields. The value of *len* must be either 32 or 64.

Overlap The fields *ieee-dest* and *vax-source* may overlap in any manner.

Flags *overflow-flag* is set if the vax-source cannot be represented in the destination field; otherwise it is cleared. If *vax-source* is the VAX "undefined variable", the IEEE destination is set to NaN(all 1's) and the *overflow-flag* is cleared. VAX double precision format uses three more mantissa bits than the IEEE double precision format uses. These bits are simply dropped during the conversion. The *overflow-flag* is always cleared for double-precision conversion.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

The CM operates internally on floating point data in IEEE format whereas the VAX uses a VAX floating-point format. In each active processor, this function converts a floating-point field in VAX format to a field in standard IEEE format.

The value of *len* specifies the precision of *vax-source*. If *len* is specified as 32, then VAX 'F' format is used. If *len* is specified as 64, then VAX 'D' format is used.

VAX and IEEE floating-point formats are incompatible, so there are a number of potential inaccuracies in the translation. These are described in the flags description above.

This instruction is useful for rapidly converting floating-point data from VAX to IEEE format. For example, if data is transferred from a VAX to a file in the CM file system, CM:f-vax-to-ieee-1L should be called after reading the data file.

All Paris front end to CM data transfer functions automatically convert the data from the front-end format appropriately so it is not necessary to call CM:vax-to-ieee before calling, for instance, one of the write-to-news-array instructions.

To convert data back to VAX floating-point format, see the definition of CM:f-ieee-to-vax-1L.

VP-SET-GEOMETRY

Returns the geometry associated with a given VP set.

Formats result ← CM:vp-set-geometry *vp-set-id*

Operands *vp-set-id* A VP set ID.

Result A geometry ID, identifying the current geometry of the specified VP set.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *geometry*(*vp-set-id*)

The geometry associated with the specified VP set is returned.

WARM-BOOT

This operation is used by the Lisp/Paris interface to reinitialize the Connection Machine system without disturbing user memory.

Formats CM:warm-boot

Context This operation is unconditional. It does not depend on the *context-flag.*

This operation clears error status indicators for the attached Connection Machine hardware. It also clears the IFIFO and OFIFO in the bus interface and possibly loads fresh microcode into the attached microcontroller(s). The user memory areas in the Connection Machine system are not disturbed, but are checked for errors; any memory errors are reported. Certain system memory areas in the Connection Machine system are reinitialized, but the state of the pseudo-random number generator is not altered and the system lights-display mode is not altered. The intent is to recover from an error condition while preserving as much of the machine state as possible.

The facility for warm-booting Connection Machine hardware is provided in different ways in the Lisp/Paris interface (on the one hand) and the C/Paris and Fortran/Paris interfaces (on the other hand).

In the Lisp/Paris interface, CM:warm-boot is a function.

This operation takes no arguments and returns no values. It signals an error if the warm-boot process was not successful.

There are two sets of initializations, kept in the variables CM:*before-warm-boot-initializations* and CM:*after-warm-boot-initializations*, that are evaluated before and after anything else occurs.

In the C/Paris and Fortran/Paris interfaces, there is no CM:warm-boot operation. Instead, a related operation called CM:init is used.

C-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both source and destination values are treated as complex numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:c-write-to-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, dest, s, e, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of complex data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must be of length *fe-rank*.

dest The field ID of the complex destination field. Must have length equal to the rank of the *dest* geometry.

s, e The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $2(s + e + 1)$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array in the front end into a similarly shaped subblock of the NEWS grid. Complex number values are transferred from the specified *front-end-array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which the data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine destination field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the destination field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, from C or Fortran, one of the following predefined complex *format* values may be used: CM_complex_float_single or CM_complex_float_double. For complex data types in C, two front-end elements are used for each Connection Machine element.

When calling Paris from Lisp, the *format* parameter is a keyword argument; for complex transfers only arrays of type t may be used

Definition For all i such that $0 \le j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \le m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$dest[k_i] \leftarrow \textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$$

Another formulation:

For all s_0 such that $0 \le s_0 < (end_0 - start_0)$ do
for all s_1 such that $0 \le s_1 < (end_1 - start_1)$ do
for all s_2 such that $0 \le s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \le s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$dest[k_{s_0,s_1,\ldots,s_{rank-1}}] \leftarrow \textit{front-end-array}_{\textit{offset}_0 + s_0, \textit{offset}_1 + s_1, \ldots, \textit{offset}_{rank-1} + s_{rank-1}}$$

F-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both source and destination values are treated as floating-point numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:f-write-to-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, dest, s, e, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of floating-point data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must have length equal to the rank of the *dest* geometry.

dest The field ID of the floating-point destination field.

s, e The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array in the front end into a similarly shaped subblock of the NEWS grid. Floating-point number values are transferred from the specified *array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which the data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine destination field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the destination field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined floating-point *format* values may be used. These are CM_float_single or CM_float_double from C or Fortran, and :float-single or :float-double from Lisp.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the Connection Machine field.

Definition For all i such that $0 \leq j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

let $s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$

let $k_i = \bigvee_{j=0}^{rank-1} make\text{-}news\text{-}coordinate(axis_j, start_j + s_{i,j})$

$dest[k_i] \leftarrow front\text{-}end\text{-}array_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

let $k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} make\text{-}news\text{-}coordinate(axis_j, start_j + s_j)$

$dest[k_{s_0, s_1, \ldots, s_{rank-1}}] \leftarrow$
$front\text{-}end\text{-}array_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}}$

S-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both the source and destination values are treated as signed integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:s-write-to-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, dest, len, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of signed integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must have length equal to the rank of the *dest* geometry.

dest The field ID of the signed integer destination field.

len The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array from the front end into a similarly shaped subblock of the NEWS grid. Signed integer values are transferred from the specified *array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which the data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

When calling Paris from Lisp, the array may be either a general array (of type t) containing signed integers, or a specialized integer-element array (such as an array of type (unsigned-byte 8)).

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine destination field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the destination field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined signed *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_1_bit_packed, CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined signed format keywords are :8-bit, :16-bit, :32-bit, :1-bit-packed, :2-bit-packed, and :4-bit-packed.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the Connection Machine field.

Definition For all i such that $0 \leq j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$dest[k_i] \leftarrow \textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$dest[k_{s_0, s_1, \ldots, s_{rank-1}}] \leftarrow \textit{front-end-array}_{\textit{offset}_0 + s_0, \textit{offset}_1 + s_1, \ldots, \textit{offset}_{rank-1} + s_{rank-1}}$$

U-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both the source and destination values are treated as unsigned integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:u-write-to-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, dest, len, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of unsigned integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must have length equal to the rank of the *dest* geometry.

dest The field ID of the unsigned integer dest field.

len The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array from the front end into a similarly shaped subblock of the NEWS grid. Unsigned integer values are transferred from the specified *array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* $\times$ *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined unsigned *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_1_bit_packed, CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined unsigned format keywords are :8-bit, :16-bit, :32-bit, :1-bit-packed, :2-bit-packed, and :4-bit-packed.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the Connection Machine field.

Definition For all i such that $0 \leq j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$dest[k_i] \leftarrow \textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$dest[k_{s_0,s_1,\ldots,s_{rank-1}}] \leftarrow \textit{front-end-array}_{offset_0+s_0, offset_1+s_1, \ldots, offset_{rank-1}+s_{rank-1}}$$

C-WRITE-TO-PROCESSOR

Stores an immediate complex number operand value into the destination field of a single specified processor.

Formats CM:c-write-to-processor-1L *send-address-value*, *dest*, *source-value*, *len*

Operands *send-address-value* An immediate operand, the send address of a single particular processor.

dest The field ID of the complex destination field.

source-value A complex immediate operand to be used as the source.

s, e The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $2(s + e + 1)$.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *dest*[*send-address-value*] $\leftarrow$ *source-value*

The specified *source-value*, a complex number, is stored into the *dest* field of the processor whose send address is the immediate operand *send-address-value*.

The constant operand *source-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary).

F-WRITE-TO-PROCESSOR

Stores an immediate floating-point number operand value into the destination field of a single specified processor.

Formats CM:f-write-to-processor-1L *send-address-value*, *dest*, *source-value*, *s*, *e*

Operands *send-address-value* An immediate operand, the send address of a single particular processor.

dest The field ID of the floating-point destination field.

source-value A floating-point immediate operand to be used as the source.

s, *e* The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *dest*[*send-address-value*] ← *source-value*

The specified *source-value*, a floating-point number, is stored into the *dest* field of the processor whose send address is the immediate operand *send-address-value*.

S-WRITE-TO-PROCESSOR

Stores an immediate signed integer operand value into the destination field of a single specified processor.

Formats CM:s-write-to-processor-1L *send-address-value*, *dest*, *source-value*, *len*

Operands

send-address-value An immediate operand, the send address of a single particular processor.

dest The field ID of the signed integer destination field.

source-value A signed integer immediate operand to be used as the source.

len The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition $dest[send\text{-}address\text{-}value] \leftarrow source\text{-}value$

The specified *source-value*, a signed integer, is stored into the *dest* field of the processor whose send address is the immediate operand *send-address-value*.

U-WRITE-TO-PROCESSOR

Stores an immediate unsigned integer operand value into the destination field of a single specified processor.

Formats CM:u-write-to-processor-1L *send-address-value*, *dest*, *source-value*, *len*

Operands

send-address-value An immediate operand, the send address of a single particular processor.

dest The field ID of the unsigned integer destination field.

source-value An unsigned integer immediate operand to be used as the source.

len The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *dest*[*send-address-value*] ← *source-value*

The specified *source-value*, an unsigned integer, is stored into the *dest* field of the processor whose send address is the immediate operand *send-address-value*.