The
Connection Machine
System

Paris Reference Manual

Version 5.0
February 1989

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, February, 1989

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142–1214
(617) 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 02142-1214
Internet Electronic Mail:	customer-support@think.com
Usenet Electronic Mail:	harvard!think!customer-support
Telephone:	(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:   bug-connection-machine@think.com
```

Please supplement the automatic report with any further pertinent information.

Chapter 1

Introduction

Paris is a low-level instruction set for programming the Connection Machine computer system. It is the lowest-level protocol by which the actions of Connection Machine processors are directed by the front-end computer. Paris is sometimes referred to as a "macroinstruction set" for the Connection Machine system because it is comparable in power to the (macro)instruction sets of typical sequential processors such as the VAX, and to distinguish it from the "microinstruction set" (microcode) that is executed by the Connection Machine system sequencer and the "nanoinstruction set" that is directly executed by the individual hardware Connection Machine processors.

Paris is intended primarily as a base upon which to build higher-level languages for the Connection Machine system. It provides a large number of operations similar to the machine-level instruction set of an ordinary computer. Paris supports primitive operations on signed and unsigned integers and floating-point numbers, as well as message-passing operations and facilities for transferring data between the Connection Machine processors and the front-end computer.

The Paris user interface consists of a set of macros, functions, and variables to be called from user code. The macros and functions direct the actions of the Connection Machine system by sending macroinstructions to the Connection Machine sequencer, and the variables allow the user program to find out information about the Connection Machine system such as the number of processors available.

Several different versions of the user interface are provided: one for the Lisp programming language, one for C, and one for Fortran. These interfaces are functionally identical; they differ only in conforming to the syntax and data types of one language or the other.

Chapter 2

Virtual Machine Architecture

An important property of the Connection Machine architecture is *scalability.* At present, a single Connection Machine system can have 16,384 or 32,768 or 65,536 physical (hardware) processors, of which any single user can use a portion containing 8,192 or 16,384 or 32,768 or 65,536 processors. (See figure 2.1 for an illustration of 65,536 processors.) In most cases the same software can be executed unchanged on Connection Machine systems (or portions) with different numbers of physical processors; the number of processors affects only the size of the problem that can be handled.

Paris enhances this scalability by presenting to the user an abstract version of the Connection Machine hardware. The most important feature is the *virtual processor* facility, whereby each physical processor is used to simulate some number of virtual processors. A program can be written assuming *any* appropriate number of processors (but not fewer than the number of physical processors); these virtual processors are then mapped onto physical processors. In this way a program can be executed unchanged on Connection Machine systems with different numbers of physical processors, even if it requires a certain minimum number of processors, with an essentially linear trade-off between number of physical processors and execution time. (There is a memory trade-off as well: the memory of a physical processor is divided among the virtual processors it supports.)

For the remainder of this chapter, when we refer to "the Connection Machine" or "the machine" we mean that portion of a Connection Machine system to which the user is attached. For example, if a user is attached to a 16,384 processor portion of a 65,536 processor Connection Machine, the expression "the machine" refers only to the user's 16,384 processors.

The Connection Machine hardware supports two mechanisms for interprocessor communication. The more general mechanism is the *router*, which allows data to be sent from any processor directly to any other processor; indeed, many processors can send data to many other processors simultaneously. The less general mechanism is redundant, but optimizes an important case for speed. It organizes the processors as an n-dimensional grid and allows every processor to send data to its immediate neighbors in the grid. This mechanism is called the *NEWS grid*, from the initials of the four directions in a two-dimensional grid: North, East, West, and South. Using these hardware mechanisms, Paris provides identical virtual mechanisms within the virtual processor framework.

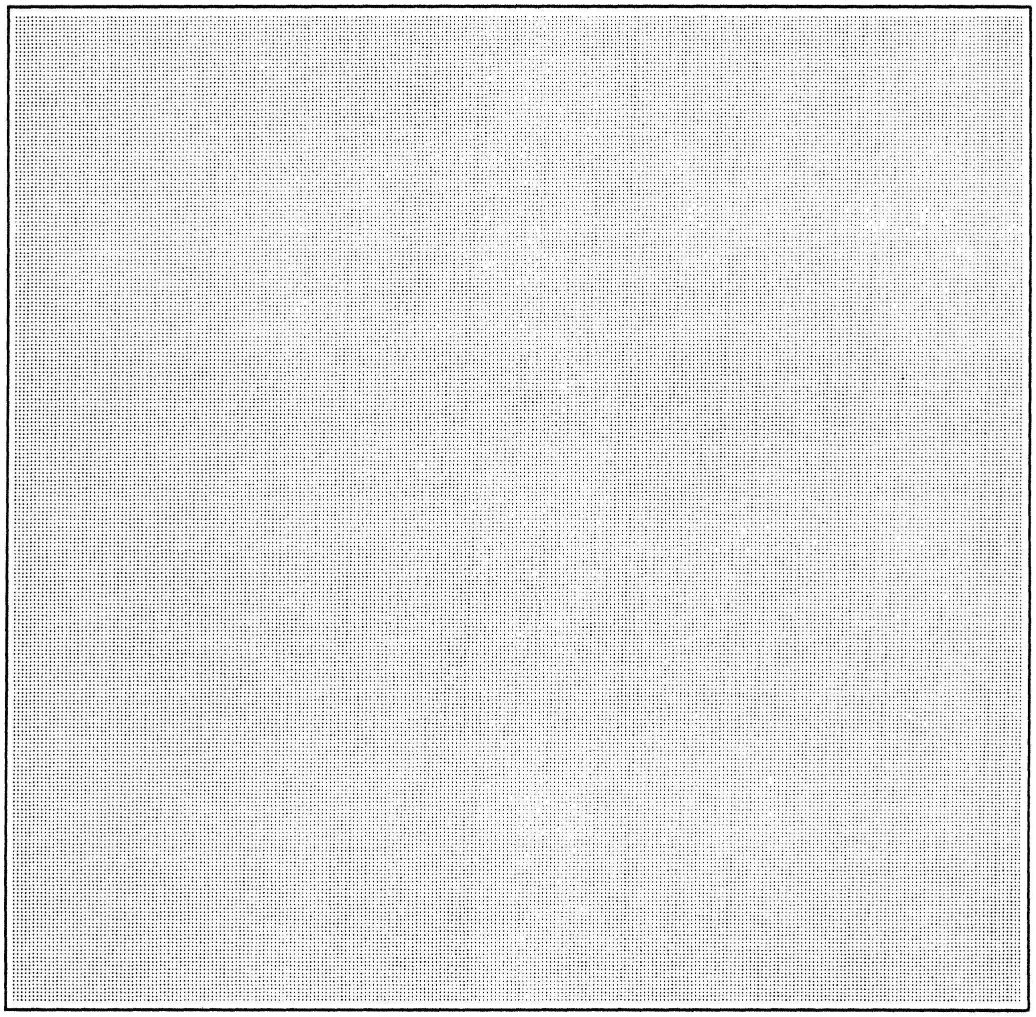


Figure 2.1: 65,536 processors

2.1 Virtual Processors and Virtual Processor Sets

The data parallel programming method associates one processor with each element of a data set. In the virtual processor abstraction provided by Paris, we associate one virtual processor, or VP, with each element of a data set. The set of all virtual processors associated with a data set is called a *virtual processor set*, or *VP set*. For example, consider an image-processing problem that deals with an image of 65,536 pixels, shaped in a 512×128 rectangle. Each pixel is an element of the data set that makes up the image. Thus we would write a program using one VP set of size 65,536: one VP for each pixel.

Because a single problem may be composed of more than one data set, Paris allows for the simultaneous existence of more than one VP set. For example, a text retrieval program might wish to deal with articles at some times, and with words in the articles at other times. This problem is most conveniently modeled with two VP sets, the first corresponding to the data set of all articles (one VP per article) and the second corresponding to the data set of all words (one VP per word).

VP sets are created and deleted through function calls to Paris. The size of a VP set (the number of virtual processors in the VP set) is fixed at the time of the VP set's creation.

Although multiple VP sets may co-exist, only one VP set may be active at any time. This VP set is known as the *current VP set*. All VP sets other than the current VP set are latent; that is, they can not execute any instructions. We say that Paris operates within the current VP set. Paris provides a function CM:set-vp-set for setting the current VP set.

2.2 Mapping VP Sets to the Physical Machine

When a Paris program is run, the virtual processors in the user's program are mapped onto the machine's physical processors. The size of the VP set(s) and the size of the physical machine determine how many virtual processors are assigned to each physical processor. In effect, each Connection Machine processor and its memory are shared among the virtual processors they support.

These concepts are further elaborated in the following sections. The time-slicing of the Connection Machine processors is covered in the section "VP Ratios"; the sharing of physical memory among virtual processors is covered in the section "Fields." Communication and related concepts follow.

2.3 VP Ratios

Let p denote the number of Connection Machine physical processors, and let $|X|$ denote the number of virtual processors in a VP set X.

For each VP set X, each physical processor is assigned the task of simulating $|X|/p$ virtual processors. This number $|X|/p$ is called the *virtual processor ratio*, or *VP ratio*, of VP set X. We denote the VP ratio of VP set X as $vpr(X)$. The virtual processor ratio must always be a power of two.

What exactly does this mean? When the machine is operating within VP set X, each instruction in the user's program is executed $vpr(X)$ times by each physical processor, that is, once for every virtual processor. This is completely transparent to the user. A change of

VP set changes the VP ratio to be that of the newly current VP set; if the program changes from VP set X to VP set Y, each instruction after that will be executed $vpr(Y)$ times.

This method of assigning virtual processors to physical processors "spreads out" a VP set as much as possible; the VP ratio for each VP set is as low as possible. The burden of handling a VP set is shared by the entire physical machine.

As an example, suppose we have two VP sets A and B, where $|A| = 64\text{K}$ and $|B| = 256\text{K}$. Suppose we run our program on a Connection Machine system with 64K physical processors ($p = 64\text{K}$). Then $vpr(a) = 64\text{K}/64\text{K} = 1$, and $vpr(b) = 256\text{K}/64\text{K} = 4$. When executing within VP set A, each instruction is executed once by each physical processor. When executing within VP set B, each instruction is executed four times by each physical processor.

If the same program were to be run on a Connection Machine system with only 16K physical processors ($p = 16\text{K}$), then we would have $vpr(a) = 64\text{K}/16\text{K} = 4$, and $vpr(b) = 256\text{K}/16\text{K} = 16$. When executing within VP set A, each instruction would be executed four times by each physical processor. When executing within VP set B, each instruction would be executed 16 times by each physical processor.

This description of "execute once for each virtual processor" applies most accurately to operations such as arithmetic that can take place within each virtual processor independently of other virtual processors. Operations that perform communication are more complicated, but the idea is the same: each physical processor performs all necessary execution steps on behalf of each virtual processor that is to participate in the operation.

As far as the user is concerned, physical processors are hardly visible. Paris is designed to allow the programmer to think entirely in terms of the virtual processor as the basic unit of computational power.

2.4 Fields

At the time of its creation, a VP set has no associated memory (except for its flags). This is the same as saying that no VP in the VP set has any memory, because the memories of all virtual processors in a VP set are always of the same size and layout. Paris provides functions to allocate and deallocate memory to a VP set.

Memory is handled in units called *fields*. Conceptually, a field is simply some number of consecutive bits. A field can be of any size greater than zero bits. When a field is allocated, it has an initial size specified by the user. When we speak of allocating a field to a VP set, we mean allocating a field to each VP in the VP set.

A field is referenced through a *field-id*. Paris returns a unique field-id for each new field that is allocated, and all Paris calls that require a reference to a field take a field-id as a parameter.

How does this abstraction of fields get mapped into physical Connection Machine memory? Again, the concept of VP ratios is important. Just as a Connection Machine physical processor takes responsibility for $vpr(X)$ virtual processors for each VP set X in the user's program, those same physical processors (more precisely, their memories) take responsibility for the fields of those same virtual processors. A single physical memory contains $vpr(X)$ copies of every field in VP set X, $vpr(Y)$ copies of every field in VP set Y, and so on for every VP set in the user's program.

There are two types of fields: heap fields and stack fields. The distinction between them has to do with the storage management strategy employed in the physical memory supporting the virtual processors. Heap fields are the more flexible of the two, but they also have the higher overhead. Heap fields may be allocated and deallocated in any order. Allocation of heap fields to VP set X may be freely intermixed with allocations to VP set Y, and so on. Deallocations need pay no attention to the VP set to which a field belongs, nor to the order in which other allocations and deallocations were done.

Stack fields may be allocated in any order, without regard to VP set. However, stack fields must be deallocated in the reverse order in which they were allocated. This rule applies globally to all fields in all VP sets. Thus, if a program allocates a field f_1 in VP set A, and then allocates a field f_2 in VP set B, and then allocates a field f_3 in VP set A, they must be deallocated in the order f_3, f_2, f_1.

2.5 Processor Addresses

Paris supports two different sorts of addresses for virtual processors: the *send address*, which is used for general purpose communication among virtual processors, and the *NEWS address*, which describes a VP's position in the n-dimensional grid used to optimize nearest-neighbor communication.

A virtual processor has one send address and one NEWS address at all times. Send addresses and NEWS addresses are specific to a VP set; that is, every VP in a VP set has a unique send address and a unique NEWS address, but it is possible for a VP in another VP set to have the same send address or NEWS address. Since Paris always operates within a single VP set, there is normally no ambiguity as to which VP is meant by a given address. For communication across VP sets, Paris has other means of uniquely identifying the intended destination VP.

2.6 Send Addresses

Send addresses are used as arguments to Paris communication operations to identify virtual processors that are to supply or receieve data. The Paris operation CM:my-send-address allows every VP in a VP set to find out its own send address.

The send address for a VP is composed of two parts, the physical part and the virtual part. The physical part indicates the location in the CM of the physical processor supporting that VP. The virtual part indicates which VP in that VP set on that physical processor is being addressed. The virtual part is in the less significant bits of the send address.

The size (in bits) of a send address for a VP set depends on two things. The physical size of the machine determines the size of the physical part of the send address. The VP ratio for the VP set determines the size of the virtual part.

For example, in a $64\text{K} = 2^{16}$ Connection Machine, the send addresses for VP set Q with $vpr(Q) = 64 = 2^6$ require 22 bits: 16 bits for the physical part, and 6 bits for the virtual part. In this example, send addresses range from 0 to $2^{22} - 1$.

	21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5	4 3 2 1 0
SEND ADDRESS	PHYSICAL PROCESSOR	VP

In this release of Paris, VP ratios must be a power of two. This results in a contiguous address space for send addresses (that is, there are no "holes"). However, this feature is likely to change in the future (thereby allowing a VP ratio to be any integer, not just a power of two). We recommend that no Paris program be written so as to require send addresses to occupy a contiguous range. In particular, we discourage arithmetic on send addresses. Paris provides functions for manipulating send addresses in a "safe" manner. Arithmetic is better done on NEWS addresses; if a total order on all processors is required, please note that a NEWS grid may be one-dimensional.

2.7 NEWS Addresses

A NEWS address is an n-tuple of coordinates $x_0, x_1, \ldots, x_{N-1}$, which specifies a VP's position in an n-dimensional Cartesian-grid geometry. The number of bits required to specify each coordinate depends on the size of that dimension in the geometry. NEWS addresses are treated in more detail below when we discuss geometries.

The Paris operation CM:my-news-coordinate-1L allows every VP in a VP set to find out its own NEWS coordinate along a given axis. Paris also provides functions for producing a send address from a NEWS address, and vice versa. There are a number of variations on these functions to handle only specific dimensions. All addresses are interpreted within the current VP set.

2.8 Communication across VP Sets

Communication across VP sets takes place via the Paris send and get operations and their variants. These operations each accept only a send address as the indicator of the remote VP; NEWS addresses are not allowed. The send address must be of the proper size for the remote VP set; that is, it must have as many bits as are necessary to specify a send address in that VP set, which may be different from the number of bits needed to specify a send address in the current VP set.

We have noted that send addresses are not unique across all VP sets in a program, but that communication across VP sets is unambiguous anyway. This is because every call to a Paris send or get operation also takes a field in a remote VP set as an argument. A field is always associated with exactly one VP set, and this fact allows Paris to determine the remote VP intended as a send destination or a get source.

2.9 Geometries

A *geometry* is an abstract description of an n-dimensional grid of elements. It specifies n, the number of dimensions (also known as the *rank* of the geometry), and it specifies the length of each dimension. There are other aspects of a geometry that may be specified by the Paris user, but we first elaborate on the more basic issues.

The rank of a geometry is an integer between 1 and 31, inclusive. This is the same as saying that a geometry can describe anything from a 1-dimensional grid to a 31-dimensional grid. We number the dimensions of a grid from 0 to the rank minus 1, so we say that a 1-dimensional grid has only dimension 0, a two-dimensional grid has dimensions 0 and 1, etc.

The size of a dimension must be a power of two. The product of the sizes of all dimensions of a geometry specifies the total number of elements in the geometry. For example, a three-dimensional geometry of size $16 \times 512 \times 2$ contains 16,384 elements in all.

Paris provides functions for defining geometries. See section 5.2. A geometry is defined in the abstract, but it has no use until it is associated with a VP set, via another Paris function. Associating a geometry with a VP set defines a "shape," or organization, for the virtual processors of the VP set.

At the time of a VP set's creation, it is associated with some geometry. The geometry specifies the size of the VP set and its conceptual organization in n-space. A VP set is always associated with exactly one geometry, but it may be associated with different geometries over time. Paris provides a function for associating a geometry with a VP set (and implicitly dis-associating the previous one). See section 5.1. In this way, the user can "reshape" a VP set. The only restriction is that all geometries associated with a VP set be of the same total size, since a VP set is not allowed to change size. For example, a VP set originally associated with a $16 \times 512 \times 2$ geometry can later be associated with a 64×256 geometry, since the total number of virtual processors described by both of these geometries is the same (16,384 in this example).

The NEWS address of a virtual processor depends completely on the geometry currently associated with its VP set. Thus, while the send addresses of virtual processors remain constant for the life of a VP set, the NEWS addresses of those same virtual processors can vary as the geometry is changed. When a VP set has a three-dimensional geometry, NEWS addresses for that VP set have three coordinates: x_0, x_1, x_2. When that VP set changes to a two-dimensional geometry, NEWS addresses for that VP set have two coordinates: x_0, x_1.

Given a VP set and given a geometry as we have described it so far (a rank and the size of each dimension), there are many ways for Paris to assign virtual processors to physical processors. However, not all mappings will provide equally efficient communication among the virtual processors of a VP set. Paris allows the user to specify more information than just rank and size of dimensions when creating a geometry. These additional pieces of geometry information we call *ordering* and *weight*, and we discuss them in more detail below.

It should be said, however, that the specification of these properties of a geometry affects only the efficiency of inter-VP communication, and therefore the performance of the program. Choosing suboptimal values will never cause an otherwise correct program to execute in an erroneous manner. Also, for some problems (those involving little or no communication among virtual processors of a VP set) it does not matter how the user specifies these properties. Paris provides a function for creating geometries that does not require specification of ordering or weight information.

Each dimension of a geometry is given an *ordering*. The ordering of a dimension specifies how NEWS coordinates for that dimension are mapped onto physical processors. There are currently two possible orderings: NEWS ordering and send-address ordering. (There may be more in the future.) Different dimensions of a geometry may be given different orderings.

The NEWS ordering specifies the embedding of the grid into the physical (hardware) n-dimensional grid such that processors with adjacent NEWS coordinates are in fact neighbors within the physical grid. The send-address ordering specifies that if processor A has a smaller NEWS coordinate than processor B (in the specified dimension), then A also has a smaller send address than B. Paris functions that provide nearest-neighbor communication (the CM:get-from-news family of functions, for example) perform best with NEWS ordering. Send ordering is useful for applications such as Fast Fourier Transform; under the send ordering, processors that are nearest neighbors within the physical grid have grid coordinates that differ by various powers of two.

What is the weight of a dimension for? Whenever the VP ratio of a VP set is greater than 1, some number of virtual processors are co-resident on a physical processor. If these virtual processors happen to all be in the same dimension of their geometry, communication among them will be even faster than if they were neighbors in the physical NEWS grid. Communication among virtual processors assigned to the 16 physical processors on a Connection Machine chip is also faster than communication between chips, even if the processors concerned are neighbors in the physical NEWS grid.

Paris can lay out virtual processors on physical processors in such a way as to take advantage of intra-processor and intra-chip communication, provided the Paris user knows which dimension(s) of the geometry will sustain the heaviest communication. (By communication, we mean also operations such as scan and spread). Thus, Paris provides an operation for creating geometries with an indication (the *weight*) of which dimension will have the heaviest communication, which will be second heaviest, etc. Paris then maps the virtual processors onto the physical processors in such a way as to favor the dimensions with the heaviest communication.

2.10 Flags

Each Paris virtual processor has an assortment of one-bit flags. These flags are represented as fields that are specially associated with VP sets. These fields are automatically created when the VP set is created by CM:allocate-vp-set.

Many Paris operations store into these flags rather than, or in addition to, storing results into explicitly supplied argument fields. For example, the CM:s-add-2-1L operation adds one signed integer to another, but also stores information into the carry flag and the overflow flag.

The entire set of flags for each virtual processor is as follows.

- The *context-flag* indicates which virtual processors are active within the current VP set. Nearly all Paris operations are *conditional*; the operation is effectively carried out only in those processors whose *context-flag* is 1, and processors whose *context-flag* is 0 are unaffected. Some operations are always unconditional.

- The *test-flag* holds the result of numeric comparisons and other tests, or indicates which operations failed because of bad operands.

- The *carry-flag* holds the carry in and carry out for some integer arithmetic operations. A few operations use the *carry-flag* as an implicit input.

- The *overflow-flag* indicates which operations produced results that the destination field was too small to contain. Many Paris operations can affect the *overflow-flag*.

Chapter 3

Data Formats

A data item always consists of a string of bits having consecutive addresses. Such a bit string is called a *field*. The term *field* is also used to refer to a collection of fields, one for each virtual processor.

Many Paris operations may be regarded as interpreting bit fields as being of particular data types or formats. Currently Paris provides operations that regard the contents of bit fields as structured according to the following data types:

- signed integers, represented in two's-complement format
- unsigned integers, represented in straight binary format
- floating-point numbers, represented in a format close to that specified by IEEE standard 754 for floating-point arithmetic
- send-addresses, which are unsigned integers that label virtual processors for communication purposes
- NEWS coordinates, which are unsigned integers, tuples of which label virtual processors within a Cartesian grid for communication purposes

The Connection Machine system allows unusual flexibility in that the hardware does not enforce any particular length or alignment requirements. Paris supports integers and floating-point numbers of almost any size. (However, certain sizes of floating-point number allow particularly efficient execution by the hardware floating-point accelerator, and certain sizes of integer allow certain other operations to be particularly efficient.)

Most Paris operations operate on fields within a virtual processor, delivering results to other fields within that virtual processor. Frequently we speak of one data item, but really mean to speak of many instances of that data item, one for each selected processor, to be considered or operated on in parallel. For example, when we say that an operation sets a flag when a field has such-and-so value, we mean that a separate decision is made within each processor whether to set that processor's flag, based on the value of the field within that processor.

3.1 Bit Fields

A bit field is specified by a bit address a and a positive length n; the field consists of the bits with addresses a through $a + n - 1$, inclusive. Therefore the address of a field is the same as that of the lowest-addressed bit.

3.2 Signed Integers

A signed integer is specified in the same way as a simple bit field, by a bit address a and a positive length n. The signed integer is represented in two's-complement form, and so a signed integer of length n can take on values in the range $-(2^{(n-1)})$ through $2^{(n-1)} - 1$, inclusive. The least significant bit has address a, and the most significant (sign) bit has address $a + n - 1$.

All arithmetic on signed integers is performed in a strict wraparound mode. As a rule, if the result of an operation overflows the destination field, the *overflow-flag* is set, and the destination receives as many low-order bits of the true result as will fit. For example, using 4-bit signed arithmetic, multiplying 4 by -7 will produce the 4-bit result 4 (and also set the *overflow-flag*), because the two's-complement representation of -28 is ...111111100100, of which the four low-order bits are 0100, or 4. Signed-integer operations that do not overflow leave the *overflow-flag* unchanged.

In order to simplify the Connection Machine microcode, this arbitrary restriction is imposed: the length n may not be zero or one. In addition, certain operations on signed integers cannot handle operands whose length is greater than the value of the variable CM:*maximum-integer-length*; see section 3.6.

3.3 Unsigned Integers

An unsigned integer is specified in the same way as a simple bit field: by a bit address a and a positive length n. The unsigned integer is represented in stright binary form, and so an unsigned integer of length n can take on values in the range 0 through $2^n - 1$, inclusive. The least significant bit has address a, and the most significant bit has address $a + n - 1$.

All arithmetic on unsigned integers is performed in a strict wraparound mode, modulo 2^n. As a rule, if the result of an operation overflows the destination field, the *overflow-flag* is set, and the destination receives as many low-order bits of the true result as will fit. For example, using 4-bit unsigned arithmetic, multiplying 4 by 7 will produce the 4-bit result 12 (and also set the *overflow-flag*), because the two's-complement representation of 28 is ...00000011100, of which the four low-order bits are 1100, or 12. Unsigned-integer operations that do not overflow clear the *overflow-flag*.

Unsigned integers, unlike signed integers, may be of length zero or one as well as of larger sizes. (Note that an unsigned integer of length zero is considered to have the value 0.) However, certain operations on unsigned integers cannot handle operands whose length is greater than the value of the variable CM:*maximum-integer-length*; see section 3.6.

3.4 Floating-Point Numbers

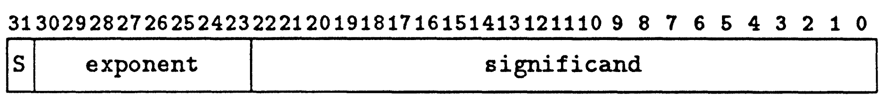
A floating-point data item is specified by three parameters: a bit address a, a significand length s, and an exponent length e. The total number of bits in the representation is $s + e + 1$, and the data item occupies the bits with addresses a through $a + s + e$, inclusive.

The significand occupies bits a through $a + s - 1$, with the least significant bit at address a. A hidden-bit representation is used, and so the significand is normally interpreted as having a 1-bit as its most significant bit implicitly just above the bit at address $a + s - 1$. If the exponent field is all zero-bits, however, then the hidden bit is taken to be 0.

The exponent occupies bits $a + s$ through $a + s + e - 1$, with the least significant bit at address $a + s$. An excess-$(2^{e-1} - 1)$ representation is used.

The sign bit occupies bit $a + s + e$, and is 1 for a negative number and 0 for a positive number. Overall, a sign-magnitude representation is used, so inverting the sign of a floating-point number merely involves flipping the sign bit. Note that there is both a plus zero and a minus zero.

When $s = 23$ and $e = 8$, this is equivalent to the IEEE standard 754 single-precision format, which looks like this:



When $s = 52$ and $e = 11$, the Paris floating-point format is equivalent to IEEE standard 754 double-precision format. The IEEE standard single-extended and double-extended formats can also be accommodated by suitable choices of s and e.

While the Paris floating-point *format* is equivalent to the IEEE standard format, it must be emphasized that the Paris implementation does not support equivalent *operations* at this time.[1] "Soft" underflow (using denormalized numbers for the result) is not supported. Rounding is performed correctly in all cases, using the round-to-nearest mode; the several rounding modes are not supported. The not-a-number (NAN) values are not supported. The standard exceptions and flags are not all supported. It is strongly recommended that a user of Paris always use the IEEE standard formats unless careful analysis of the application (such as a need for speed or additional exponent range) indicates that another format is required and adequate.

The format of a floating-point operand must obey certain restrictions. The length s must be greater than 0 and not greater than CM:*maximum-significand-length*. The length e must be greater than 1 and not greater than CM:*maximum-exponent-length*. See section 3.6. These restrictions are additionally imposed: $e \geq 2$, $s \geq 1$, and $2^{e-1} \geq s + 1$. Values for s and e not satisfying these restrictions will cause unpredictable results.

[1] Thinking Machines Corporation does intend to support all standard IEEE arithmetic operations in a future software release.

3.5 Send Addresses

Every virtual processor in a VP set has an identifiying *send address*, a kind of serial number that distinguishes it from all other virtual processors in that VP set. These addresses are used to perform general interprocessor communication. For example, in the CM:send-1L operation, each virtual processor provides a message and the send address of some other processor, and that message is sent to the specified processor (all such messages effectively being sent in parallel).

The number of bits in a send address depends on the VP set, or rather upon the geometry of that VP set. The function CM:geometry-send-address-length may be used to determine the length in bits of a send address for a given geometry. Suppose that for geometry G this function returns m; then a send address a for a virtual processor in a VP set with geometry G is an unsigned integer such that $0 \leq a < 2^m$. (Programs should not, however, rely on the fact that every integer k such that $0 \leq k < 2^m$ is a valid send address. In a future release of Paris the space of send addresses may contain "holes"; this could occur when the total number of virtual processors in the geometry is not a power of two, an extension that Thinking Machines is contemplating for the future.)

3.6 Configuration Variables

The current configuration of the machine is reflected in a few global variables. Programs may refer to these so they can adapt to various sizes of machine. These variables are set by the cold boot procedure. They should never be set by the user, as there are dependencies among them, which, if violated, will result in errors. Some variables are fixed by the hardware, while others depend on the arrangement of virtual processors set up by the attach or cold boot process. Some variables represent implementation restrictions.

CM:*current-vp-set*

The VP-set-id for the current VP set is always available in this variable. For example, to determine the total number of processors in the current VP set, one might say (in Lisp syntax)

```
(CM:geometry-total-processors
  (CM:vp-set-geometry CM:*current-vp-set*))
```

or (in C syntax)

```
CM_geometry_total_processors(CM_vp_set_geometry(CM_current_vp_set))
```

or (in Fortran syntax)

```
CM_GEOMETRY_TOTAL_PROCESSORS(CM_VP_SET_GEOMETRY(CM_CURRENT_VP_SET))
```

CM:*physical-processors-limit*

The total number of physical processors available for use.

CM:*physical-processors-length*

The base-2 logarithm of the total number of physical processors, that is, the minimum length in bits for an unsigned integer field that can contain the number of any physical processor.

CM:*physical-memory-limit*

The amount of physical memory per physical processor, including memory that is set aside for system use.

CM:*physical-memory-length*

The base-2 logarithm of the amount of physical memory per physical processor.

CM:*maximum-integer-length*

Because of implementation restrictions, a few operations on signed and unsigned integers cannot handle operands longer than the value of CM:*maximum-integer-length*.

Experimentation might reveal that in certain cases some of these operations succeed when applied to operands that are longer than this variable, but that fact is not guaranteed in succeeding software releases.

The value of CM:*maximum-integer-length* is never smaller than 128.

CM:*maximum-significand-length*

Because of implementation restrictions, a few operations on floating-point numbers cannot handle operands with significands longer than a certain size.

Experimentation might reveal that in certain cases some of these operations succeed when applied to operands that are longer than specified by these variables, but that fact is not guaranteed in succeeding software releases.

The value of CM:*maximum-significand-length* is never smaller than 96.

CM:*maximum-exponent-length*

Because of implementation restrictions, a few operations on floating-point numbers cannot handle operands with exponents longer than a certain size.

Experimentation might reveal that in certain cases some of these operations succeed when applied to operands that are longer than specified by these variables, but that fact is not guaranteed in succeeding software releases.

The value of CM:*maximum-exponent-length* is never smaller than 32.

CM:*no-field*

The value of this variable is a dummy field-id suitable for use as an argument to CM:send-1L and related instructions to indicate that no *notify* field is to be used, or to CM:scan-with-... operations to indicate an unused *sbit* argument when the *smode* argument is :none.

Chapter 4

Operation Formats

Paris operations are executed at the direction of a program running in the front-end machine. For each operation there is a function or macro that, when called, causes the Connection Machine hardware to perform the operation.

4.1 Field Id's

Most Paris operations operate on bit fields in the memories of the data processors. A bit field is specified by a *field id*, a data object that serves to identify the field. A Paris operation that allocates memory for a new field will generate and return a new field id; this field id may then be used as an argument to other Paris operations.

For example, in Lisp one might create a new heap field and then unconditionally initialize its contents to 5.0 in the following manner:

```
(let ((fld (CM:allocate-heap-field 32)))     ;Allocate
  (CM:f-move-const-always-1L fld 5.0 23 8)   ;Initialize
  ...)
```

In C the same operation would look like this:

```
{
    CM_field_id_t fld = CM_allocate_heap_field(32);  /* Allocate */
    CM_f_move_const_always_1L(fld, 5.0, 23, 8);      /* Initialize */
    ...
}
```

And in Fortran:

```
C  Declare the variable
      INTEGER FLD
      ...
C  Allocate and initialize
      FLD = CM_ALLOCATE_HEAP_FIELD(32)
      CM_F_MOVE_CONST_ALWAYS_1L(FLD, 5.0, 23, 8)
      ...
```

4.2 Constant Operands

Certain operations accept as an operand a single datum computed within the front end that is broadcast to all of the Connection Machine processors as part of the operation. Such operations have -constant in their names (or -const, in the case of certain compound operations). As a rule, every operation with -constant in its name has a counterpart without -constant in its name.

For example, to CM:f-add-constant-2-1L there corresponds CM:f-add-2-1L. These operations do exactly the same thing except that the first two operands to CM:f-add-2-1L are field id's for fields containing floating-point numbers, whereas CM:f-add-constant-2-1L takes a field id and a front-end floating-point number. This latter value is broadcast to all (active) processors and then used in the same way that a second field would be used by CM:f-add-2-1L. Here are examples of their use in Lisp:

```
(CM:f-add-2-1L x y 23 8)                ;Add field y into field x
(CM:f-add-constant-2-1L x 2.7 23 8)     ;Add 2.7 into field x
```

The same examples in C:

```
CM_f_add_2_1L(x, y, 23, 8);             /* Add field y into field x */
CM_f_add_constant_2_1L(x, 2.7, 23, 8);  /* Add 2.7 into field x */
```

The same examples in Fortran:

```
C  Add field y into field x
      CM_F_ADD_2_1L(X, Y, 23, 8)
C  Add 2.7 into field x
      CM_F_ADD_CONSTANT_2_1L(X, 2.7, 23, 8)
```

4.3 Unconditional Operations

Most Paris operations are conditional: they take place only in processors that have a 1 in the *context-flag*. But sometimes it is necessary to perform operations unconditionally (that is, without respect to the *context-flag*). A number of Paris operations have unconditional versions, generally named by inserting -always in the name of the conditional function. For example, CM:s-move-always-1L is the unconditional equivalent of CM:s-move-1L.

Paris operations that deal directly with the *context-flag* are inherently unconditional. For the sake of brevity, the names of these operations do not contain -always. Any Paris operation that has -context in its name deals with the *context-flag* and is implicitly unconditional despite the fact that -always does not also appear in its name. One example is CM:set-context.

A few other Paris operations also have only unconditional forms but do not have names containing -always. These are typically specialized communications operations whose names are already so long that inserting -always would exceed the limit on the length of a name. One example is CM:u-read-from-news-array-1L.

4.4 Naming Conventions

Lisp, C, and Fortran impose different sets of rules and conventions on how functions and variables are to be named. The description of Paris in this document strikes a compromise among these languages. All names in this document are presented in Lisp syntax, but carefully observing capitalization, to which C is sensitive even though Fortran and Lisp are not. The Paris Dictionary contains a simple set of rules for converting a Lisp name into the corresponding C or Fortran name.

The rest of this section describes the general rules that were used to achieve a regular naming system for Paris operations. It is not necessary to know these rules to use Paris, but a passing familiarity may help you to remember an exact operation name without having to look it up, or to recognize the argument format from the operation name.

The name of every Paris operation begins with CM: (in Lisp) or CM_ (in C and Fortran). It also contains one or more words that are the "main description" of the operation, such as add or send or read-from-news-array.

Between the leading CM: or CM_ and the main operation may be one or more prefixes. The prefix fe- indicates an operation performed entirely on the front end (often such an operation has a parallel counterpart without the fe- prefix). Examples of this correspondence are CM:extract-news-coordinate and CM:fe-extract-news-coordinate. If an fe- prefix is present, it appears before all other prefixes.

Other prefixes indicate the type of data to be operated upon:

f-	floating-point number
s-	signed integer
u-	unsigned integer

For example, CM:f-add-2-1L adds floating-point numbers, whereas CM:s-add-2-1L add signed integers.

If there is more than one type prefix, then the first type applies to the result of the operation, and the other(s) apply to certain source operands, usually the last one(s). For example, CM:s-f-truncate-2-2L produces a signed integer result from a floating-point source.

Some operations include in their names the name of another operation. In this case the embedded operation may have a type prefix. An example is CM:spread-with-f-add-1L. (The name of such an embedded operation is usually preceded by with-, but exceptions occur when this would make names too long, as in CM:multispread-f-multiply-1L, an operation that is not yet implemented but may be in the future.)

There are four groups of *suffixes* for operation names: -constant, -always, number of fields, and number of lengths. They always appear (if at all) in this order.

A number-of-fields suffix is simply a digit (preceded by a hyphen or underscore), such as -3. In many cases there are sets of similar operations differing primarily in their argument format. For example, CM:f-multiply-3-1L takes three fields and stores the floating-point product of the second and third fields into the first field, whereas CM:f-multiply-2-1L takes only two fields, and stores their product back into the first field (thereby overwriting one source value). These two formats are distinguished by a suffix indicating the number of arguments that are fields (in this case -3 or -2). As a rule, this suffix is supplied only if it is necessary to distinguish two or more possible formats.

A number-of-lengths suffix is simply a digit (preceded by a hyphen or underscore) followed by a capital L, such as -3L. This suffix indicates how many length arguments are required. Such arguments indicate the lengths of field arguments. For example, CM:s-add-3-3L takes three field arguments followed by three corresponding length arguments; but CM:s-add-3-1L takes three field arguments and a single length argument that describes the length of all three fields. Note that the format of a floating-point field is described by *two* arguments (significand length and exponent length), but these two arguments are lumped together and counted as a single length. As a rule this suffix always appears in the name of any operation that takes one or more field length arguments.

To summarize, the name of a Paris operation is more or less of this form:

CM:[fe-]{f- | s- | u-}*⟨main name⟩[⟨embedded name⟩][-constant][-always][-m][-nL]

An effort has been made to use full English words in the names of Paris operations. The limitation on the total length of names has made it necessary to use certain abbreviations universally:

divinto	divide into
fe-	front end
f-	floating-point
max	maximum
min	minimum
mod	modulo
rem	remainder
s-	signed integer
subfrom	subtract from
u-	unsigned integer

Some of these are standard abbreviations, of course, used in many programming languages. Paris also uses standard abbreviated names for mathematical operations (tan for the tangent function, for example).

Paris uses certain additional abbreviations in the names of compound operations:

mult	multiply
const	constant
sub	subtract

An example is CM:f-mult-const-sub-const-1L.

4.5 Argument Order

An attempt has been made to keep argument order consistent. The following rules of thumb apply.

Arguments that are fields come first. If there is a destination field it always comes first.

Length fields usually come last. They appear in the same order as the fields to which they apply, but if both integer and floating-point fields appear then the floating-point length arguments appear last. For some complex communication operations, such as scan operations, certain control arguments follow the lengths.

Chapter 5

Instruction Set Overview

This chapter provides a quick guided tour of the entire Paris instruction set, organized by categories of functionally related operations. The names of the operations are presented in the form of charts that bring out the combinatorial structure of the instruction set. Alternatives are stacked vertically between braces, and the symbol $\sim$ indicates a choice that adds no characters to the operation name.

The next chapter, the Paris Dictionary, is organized alphabetically by operation name, and provides detailed descriptions of all the operations.

5.1 VP Sets

These operations create, destroy, and otherwise manipulate VP sets.

$$\text{CM:} \left\{ \begin{array}{l} \text{allocate-vp-set} \\ \text{deallocate-vp-set} \\ \text{physical-vp-set} \\ \text{set-vp-set} \\ \text{set-vp-set-geometry} \\ \text{vp-set-geometry} \end{array} \right\}$$

The operation CM:allocate-vp-set creates a new VP set having a specified geometry (which must be created first). The operation CM:deallocate-vp-set may be used to inform the Paris interface that the user program will not use a VP set any longer.

Of particular importance is CM:set-vp-set, which selects a given VP set as the current VP set.

Given a VP set, the operation CM:vp-set-geometry returns the geometry associated with that VP set.

5.2 Geometries

These operations create, destroy, and otherwise manipulate geometries.

$$\text{CM:} \left\{ \begin{array}{l} \text{create-detailed-geometry} \\ \text{create-geometry} \\ \text{deallocate-geometry} \\ \text{geometry-axis-length} \\ \text{geometry-axis-ordering} \\ \text{geometry-axis-vp-ratio} \\ \text{geometry-coordinate-length} \\ \text{geometry-rank} \\ \text{geometry-send-address-length} \\ \text{geometry-total-processors} \\ \text{geometry-total-vp-ratio} \end{array} \right\}$$

Note the many operations that inquire about the shape of the geometry and various axis attributes.

5.3 Fields

These operations create, destroy, and otherwise manipulate fields.

$$\text{CM:} \left\{ \begin{array}{l} \text{add-offset-to-field-id} \\ \text{allocate-heap-field} \\ \text{allocate-heap-field-vp-set} \\ \text{allocate-stack-field} \\ \text{allocate-stack-field-vp-set} \\ \text{deallocate-heap-field} \\ \text{deallocate-stack-through} \\ \text{is-field-in-heap} \\ \text{is-field-in-stack} \\ \text{is-stack-field-newer} \\ \text{next-stack-field-id} \end{array} \right\}$$

Fields are used to contain data to be operated upon in parallel. Most Paris operations require one or more fields as arguments.

5.4 Copying Fields

These operations simply copy data from one place to another.

$$\text{CM:} \left\{ \begin{array}{l} \text{s-} \\ \text{u-} \\ \text{f-} \end{array} \right\} \text{move} \left\{ \begin{array}{c} \text{-2L} \\ \left\{ \begin{array}{l} \sim \\ \text{-constant} \\ \text{-zero} \end{array} \right\} \left\{ \begin{array}{l} \sim \\ \text{-always} \end{array} \right\} \text{-1L} \end{array} \right\}$$

The two-length versions of the move operations allow for sign-extension (or truncation) of signed integers, zero-extension (or truncation) of unsigned integers, and changes of range or precision for floating-point numbers.

$$\text{CM:} \left\{ \begin{array}{l} \text{move-reversed} \\ \text{swap-2} \end{array} \right\} \text{-1L}$$

The move-reversed operation reverses the order of the bits in a field as it copies them. The swap operation exchanges the contents of two fields.

5.5 Bitwise Boolean Operations

These operations treat fields bit by bit.

$$\text{CM:} \left\{ \begin{array}{l} \text{logand} \\ \text{logior} \\ \text{logxor} \\ \text{logeqv} \\ \text{lognand} \\ \text{lognor} \\ \text{logandc1} \\ \text{logandc2} \\ \text{logorc1} \\ \text{logorc2} \end{array} \right\} \left\{ \begin{array}{c} \sim \\ \text{-constant} \end{array} \right\} \left\{ \begin{array}{l} \text{-2-1L} \\ \text{-3-1L} \end{array} \right\}$$

$$\text{CM:lognot} \left\{ \begin{array}{l} \text{-1-1L} \\ \text{-2-1L} \end{array} \right\}$$

Paris provides all ten non-trivial bitwise boolean operations on two operands, as well as the logical NOT operation that inverts all bits.

5.6 Operations on Flags

Special operations are provided for operating on the flags.

$$\text{CM:} \left\{ \begin{array}{l} \text{load-} \\ \text{store-} \\ \text{clear-} \\ \text{set-} \\ \text{invert-} \\ \text{logand-} \\ \text{logior-} \\ \text{global-logand-} \\ \text{global-logior-} \\ \text{global-count-} \end{array} \right\} \left\{ \begin{array}{l} \text{test} \\ \text{overflow} \end{array} \right\}$$

Flags can be loaded from or stored into another field; cleared to zero or set to one; inverted; or combined with another field via logical AND or OR. One may also determine whether any processor, or all processors, have a flag set, or count the number of processors that have a flag set.

$$\text{CM:clear-all-flags}\left\{\begin{array}{c}\sim\\ \text{-always}\end{array}\right\}$$

For convenience, a special compound operation is provided for clearing all the flags except the context.

$$\text{CM:}\left\{\begin{array}{l}\left\{\begin{array}{l}\text{load-}\\ \text{store-}\\ \text{clear-}\\ \text{set-}\\ \text{invert-}\\ \text{logand-}\\ \text{logior-}\\ \text{global-logand-}\\ \text{global-logior-}\\ \text{global-count-}\end{array}\right\}\text{context}\\ \text{logand-context-with-test}\end{array}\right\}$$

The context flag is distinguished from the others, in that operations on the context flag are always unconditional, while most operations on the other flags are conditional (that is, depend on the state of the context flag).

5.7 Operations on Single Bits

Each of these operations takes exactly one one-bit field as its operand.

$$\text{CM:}\left\{\begin{array}{l}\text{clear-}\\ \text{set-}\\ \text{invert-}\\ \text{global-logand-}\\ \text{global-logior-}\\ \text{global-count-}\end{array}\right\}\text{bit}\left\{\begin{array}{c}\sim\\ \text{-always}\end{array}\right\}$$

These operations on single-bit fields are provided purely for the sake of efficiency. For example,

CM:clear-bit x

has the same effect as

CM:u-move-constant-1L x, 0, 1

but requires only one operand to be processed instead of three. Paris also provides unconditional forms of all these operations.

5.8 Unary Arithmetic Operations

Paris supports most of the unary arithmetic operations one might expect to find in a computer instruction set, as well as a number that are unusual. Most of them are provided in both one-operand and two-operand formats. The one-operand format treats the destination field as also the source operand; the result replaces the input. The two-operand format has a separate source operand, and ignores the previous contents of the destination field. (As a rule, the two-operand format operates correctly if the two operands are the same field, but may be slower than using the one-operand format.)

$$\text{CM:}\left\{\begin{matrix}\left\{\begin{matrix}\text{s-}\\\text{u-}\end{matrix}\right\}\left\{\begin{matrix}\text{negate}\\\text{isqrt}\end{matrix}\right\}\\ \text{s-}\left\{\begin{matrix}\text{abs}\\\text{s-signum}\end{matrix}\right\}\end{matrix}\right\}\left\{\begin{matrix}\text{-1-1L}\\\text{-2-1L}\\\text{-2-2L}\end{matrix}\right\}$$

For signed and unsigned integers there are negation and integer square root. Absolute value and signum are provided for signed operands only, as these operations are degenerate in the unsigned case.

$$\text{CM:}\left\{\begin{matrix}\text{s-}\\\text{u-}\end{matrix}\right\}\left\{\begin{matrix}\text{integer-length}\\\text{logcount}\end{matrix}\right\}\text{-2-2L}$$

The integer-length operation is a modified base-2 logarithm, useful for determining the minimum number of bits required to represent an integer in signed or unsigned form. The logcount operation counts the number of 1-bits in a binary representation (or, in the signed case, it counts the bits that differ from the sign bit).

$$\text{CM:u-}\left\{\begin{matrix}\text{from}\\\text{to}\end{matrix}\right\}\text{-gray-code}\left\{\begin{matrix}\text{-1-1L}\\\text{-2-1L}\end{matrix}\right\}$$

Operations are provided for converting to and from a Gray code representation of binary integers.

$$\text{CM:}\left\{\begin{matrix}\text{f-}\left\{\begin{matrix}\text{s-}\\\text{u-}\end{matrix}\right\}\text{float}\\ \text{s-}\quad\text{f-}\left\{\begin{matrix}\text{floor}\\\text{truncate}\end{matrix}\right\}\end{matrix}\right\}\left\{\text{-2-2L}\right\}$$

Some unary operations take a floating-point operand and produce an integer result, or vice versa. The float operations convert an integer to a floating-point representation. There are several different ways to convert a floating-point number to an integer, reflecting different possible choices for rounding or truncation; floor and truncate provide two such cases.

$$\text{CM:f-}\left\{\begin{array}{l}\text{abs}\\ \text{negate}\\ \text{sqrt}\\ \text{f-floor}\\ \text{f-ceiling}\\ \text{f-truncate}\\ \text{f-signum}\end{array}\right\}\left\{\begin{array}{l}\text{-1-1L}\\ \text{-2-1L}\end{array}\right\}$$

Floating-point absolute value, negation, and square root are provided, as well as truncating and signum operations.

$$\text{CM:f-}\left\{\begin{array}{c}\text{exp}\\ \text{ln}\\ \left\{\begin{array}{c}\sim\\ \text{a}\end{array}\right\}\left\{\begin{array}{l}\text{sin}\\ \text{cos}\\ \text{tan}\\ \text{sinh}\\ \text{cosh}\\ \text{tanh}\end{array}\right\}\end{array}\right\}\left\{\begin{array}{l}\text{-1-1L}\\ \text{-2-1L}\end{array}\right\}$$

Paris provides a complete set of transcendental and trigonometric functions, including hyperbolic functions and their inverses.

5.9 Binary Arithmetic Operations

Paris includes most of the binary arithmetic operations one might expect to find in a computer instruction set, as well as a number that are unusual. Most of them are provided in both two-operand and three-operand formats. The two-operand format treats the destination field as also one of source operands; the result replaces the first input. The three-operand format has two separate source operands, and ignores the previous contents of the destination field. (As a rule, the three-operand format operates correctly if the destination field is the same as one or both source fields, but may be slower than using a two-operand format.)

$$\text{CM:}\left\{\begin{array}{l}\text{s-}\\ \text{u-}\end{array}\right\}\left\{\begin{array}{l}\text{add}\\ \text{subtract}\\ \text{multiply}\\ \text{max}\\ \text{min}\\ \text{truncate}\\ \text{round}\end{array}\right\}\left\{\begin{array}{l}\text{-3-3L}\\ \left\{\begin{array}{c}\sim\\ \text{-constant}\end{array}\right\}\left\{\begin{array}{l}\text{-2-1L}\\ \text{-3-1L}\end{array}\right\}\end{array}\right\}$$

$$\text{CM:}\left\{\begin{array}{l}\text{s-}\\ \text{u-}\end{array}\right\}\left\{\begin{array}{l}\text{rem}\\ \text{mod}\end{array}\right\}\left\{\begin{array}{c}\sim\\ \text{-constant}\end{array}\right\}\left\{\begin{array}{l}\text{-2-1L}\\ \text{-3-1L}\end{array}\right\}$$

For signed and unsigned integers, the usual addition, subtraction, and multiplication operations are provided, as well as max and min operations that store the larger or smaller of the two inputs. There is no single integer division operation; four are provided, with names that reflect the rounding or truncation that must occur when the division is not exact. Conceptually there are four corresponding remainder operations, but only the two most commonly used are provided in Paris: rem, which corresponds to truncate division; and mod, which corresponds to floor division.

$$\text{CM:}\left\{\begin{matrix}\text{s-}\\\text{u-}\end{matrix}\right\}\text{subfrom}\left\{\begin{matrix}\text{-2-1L}\\\text{-constant}\left\{\begin{matrix}\text{-2-1L}\\\text{-3-1L}\end{matrix}\right\}\end{matrix}\right\}$$

Subtraction is not commutative, and so for efficiency the special case of reverse subtraction is provided. (Division is not commutative, either, but is a sufficiently expensive operation that the relative cost of a separate instruction to copy a constant into a temporary field first is small. Paris therefore does not provide integer reverse division operations.)

$$\text{CM:}\left\{\begin{matrix}\text{s-}\\\text{u-}\end{matrix}\right\}\text{add-carry}\left\{\begin{matrix}\text{-3-3L}\\\text{-2-1L}\\\text{-3-1L}\end{matrix}\right\}$$

Paris allows addition and subtraction on integers hundreds of bits long; but in case that is not enough, the usual add-carry and subtract-borrow operations, which use the carry flag as an implicit input, are provided to allow efficient programming of very high precision integer arithmetic.

$$\text{CM:}\left\{\begin{matrix}\text{s-}\\\text{u-}\end{matrix}\right\}\text{add-flags -2-1L}$$

The add-flags operation performs an addition and sets the flags but stores no sum. This is useful in a few specialized situations, such as CORDIC-type calculations.

$$\text{CM:s-s-power}\left\{\begin{matrix}\sim\\\text{-constant}\end{matrix}\right\}\left\{\begin{matrix}\text{-2-1L}\\\text{-3-1L}\end{matrix}\right\}$$

Integer exponentiation operations are provided for signed operands.

$$\text{CM:f-}\left\{\begin{matrix}\left\{\begin{matrix}\text{add}\\\text{subtract}\\\text{multiply}\\\text{divide}\end{matrix}\right\}\left\{\begin{matrix}\sim\\\text{-constant}\\\text{-always}\\\text{-const-always}\end{matrix}\right\}\\\left\{\begin{matrix}\text{max}\\\text{min}\\\text{rem}\\\text{f-power}\end{matrix}\right\}\left\{\begin{matrix}\sim\\\text{-constant}\end{matrix}\right\}\end{matrix}\right\}\left\{\begin{matrix}\text{-2-1L}\\\text{-3-1L}\end{matrix}\right\}$$

For floating-point numbers, the usual addition, subtraction, multiplication, and division operations are provided. Note that there are unconditional versions of these operations in Paris; they can be much faster than the conditional versions when floating-point hardware is used. Also provided are max and min operations that store the larger or smaller of the two inputs, a floating-point remainder operation, and an exponentiation operation.

$$\text{CM:f-}\left\{\begin{array}{l}\text{subfrom}\\ \text{divinto}\end{array}\right\}\left\{\begin{array}{l}\left\{\begin{array}{l}\sim\\ \text{-always}\end{array}\right\}\qquad\qquad \text{-2-1L}\\ \left\{\begin{array}{l}\text{-constant}\\ \text{-const-always}\end{array}\right\}\left\{\begin{array}{l}\text{-2-1L}\\ \text{-3-1L}\end{array}\right\}\end{array}\right\}$$

Subtraction and division are not commutative, and so for efficiency special cases of reverse subtraction and reverse division are provided. (Unlike the integer case, floating-point division is sufficiently fast and sufficiently common that these special cases are worthwhile.)

$$\text{CM:f-}\left\{\begin{array}{l}\text{s}\\ \text{u}\end{array}\right\}\text{power}\left\{\begin{array}{l}\text{-2-2L}\\ \text{-3-2L}\\ \text{-constant-2-1L}\\ \text{-constant-3-1L}\end{array}\right\}$$

Other useful operations include exponentiating to an integer power.

CM:f-atan2-3-1L

A two-input arctangent operation is provided.

5.10 Optimized Floating-Point Computations

Paris supports compound floating-point operations that are functionally identical to sequences of simpler floating-point operations. The compound operations are provided purely for the sake of efficiency; they can be implemented so to exploit floating-point hardware more cleverly.

$$\text{CM:f-}\left\{\begin{array}{l}\text{mult}\left\{\begin{array}{l}\sim\\ \text{-const}\end{array}\right\}\left\{\begin{array}{l}\text{-add}\\ \text{-sub}\end{array}\right\}\left\{\begin{array}{l}\sim\\ \text{-const}\end{array}\right\}\\ \left\{\begin{array}{l}\text{add}\\ \text{sub}\end{array}\right\}\left\{\begin{array}{l}\sim\\ \text{-const}\end{array}\right\}\text{-mult}\left\{\begin{array}{l}\sim\\ \text{-const}\end{array}\right\}\end{array}\right\}\text{-1L}$$

These compound operations perform calculations of the following forms: $xa+b$, $xa-b$, $(x+a)b$, and $(x-a)b$, where x is always a field in memory, and a and b may each be either a field or a constant.

5.11 Arithmetic Comparisons

Paris supports the usual six comparison operations $=$, $\neq$, $<$, $\leq$, $>$, and $\geq$ for integers and floating-point numbers.

$$\text{CM:}\left\{\begin{matrix}\text{s-}\\\text{u-}\end{matrix}\right\}\left\{\begin{matrix}\text{eq}\\\text{ne}\\\text{lt}\\\text{le}\\\text{gt}\\\text{ge}\end{matrix}\right\}\left\{\begin{matrix}\text{-2L}\\\left\{\begin{matrix}\sim\\\text{-constant}\\\text{-zero}\end{matrix}\right\}\text{-1L}\end{matrix}\right\}$$

$$\text{CM:f-}\left\{\begin{matrix}\text{eq}\\\text{ne}\\\text{lt}\\\text{le}\\\text{gt}\\\text{ge}\end{matrix}\right\}\left\{\begin{matrix}\sim\\\text{-constant}\\\text{-zero}\end{matrix}\right\}\text{-1L}$$

Each is available in three forms: compare two fields, compare a field to a constant, and compare a field to zero. The integer operations also allow integer fields of differing length to be compared.

5.12 Pseudo-Random Number Generation

Paris provides a built-in generator of uniformly distributed pseudo-random numbers.

$$\text{CM:}\left\{\begin{matrix}\text{u-}\\\text{f-}\end{matrix}\right\}\text{random -1L}$$

CM:initialize-random-generator

One may generate unsigned integers over a specified range, or floating-point numbers in the range from 0.0 (inclusive) to 1.0 (exclusive).

5.13 Arrays

Often it is convenient to treat a large field as an array of smaller fields. These operations allows each virtual processor to index independently into its own array.

$$\text{CM:}\left\{\begin{matrix}\text{aref}\\\text{aref32}\left\{\begin{matrix}\sim\\\text{-shared}\end{matrix}\right\}\left\{\begin{matrix}\sim\\\text{-always}\end{matrix}\right\}\\\text{aset}\\\text{aset32}\left\{\begin{matrix}\sim\\\text{-shared}\end{matrix}\right\}\end{matrix}\right\}\text{-2L}$$

Three kinds of arrays are supported. An ordinary array is laid out in memory exactly as one would expect: each processor contains its own array elements, concatenated end-to-end to form one large field.

A so-called "fast" array is laid out in such a way that an array element logically belonging to one processor is actually stored in memory belonging to 32 processors. The total amount of memory involved is the same, of course, but because the data is laid out in this peculiar manner ordinary Paris operations (such as CM:f-add-2-1L, for example) cannot properly operate on array elements directly. Only special operations designed to operate on fast arrays can properly fetch or store array elements; however, these special operations are much faster than the corresponding operations on ordinary arrays.

A *shared* array is shared among all the virtual processors occupying a group of 32 physical processors. This can save a great deal of memory, and is useful for lookup tables that are the same for all processors. Of course, care is required when storing into such arrays. In principle this sharing concept could be supported in both ordinary and fast versions, but in fact Paris provides special operations only for fast shared arrays.

Paris also provides, for efficiency, certain compound operations that combine communication with access to a fast array.

5.14 General Communication

The router functions (**send** and **get**) transmit data in a general fashion that allows any processor to communicate directly with any other processor.

$$\text{CM:send}\left\{\begin{array}{l}\sim\\ \text{-with}\left\{\begin{array}{l}\text{-overwrite}\\ \text{-logand}\\ \text{-logior}\\ \text{-logxor}\\ \left\{\begin{array}{l}\text{-s-}\\ \text{-u-}\\ \text{-f-}\end{array}\right\}\left\{\begin{array}{l}\text{add}\\ \text{min}\\ \text{max}\end{array}\right\}\end{array}\right\}\end{array}\right\}\text{-1L}$$

$$\text{CM:send-aset32}\left\{\begin{array}{l}\text{-overwrite}\\ \text{-logior}\\ \quad\text{-u-}\quad\text{add}\end{array}\right\}\text{-2L}$$

$$\text{CM:get}\left\{\begin{array}{l}\text{-1L}\\ \text{-aref32-2L}\end{array}\right\}$$

CM:my-send-address

Every processor within a VP set is identified by an unsigned binary integer called its *send-address*. If processor A is to send a message M to processor B, then procesor A must contain the send-address of processor B as well as the data M to be sent.

For efficiency, Paris includes compound operations that combine general communication with a fast array reference (aref32 or aset32) within the addressed processor.

5.15 NEWS Communication

The NEWS functions (**send-to-news** and **get-from-news**) organize the processors into a multidimensional rectangular grid, and transmit data from every processor to its neighbor along a specified grid axis. The NEWS operations are considerably more efficient, when applicable, than using the general router mechanism.

$$\text{CM:} \left\{ \begin{array}{l} \text{get-from-} \\ \text{send-to-} \end{array} \right\} \text{news} \left\{ \begin{array}{c} \sim \\ \text{-always} \end{array} \right\} \text{-1L}$$

These operations copy data from each processor to the adjacent processor along any NEWS axis.

$$\text{CM:} \left\{ \begin{array}{l} \text{my-news-coordinate} \\ \text{extract-news-coordinate} \\ \text{deposit-news-coordinate} \\ \text{deposit-news-constant} \\ \text{make-news-coordinate} \end{array} \right\} \text{-1L}$$

The operation *my-news-coordinate* stores the NEWS coordinate of each selected processor along a specified NEWS axis into a destination field within that processor.

The operation *extract-news-coordinate* defines the mapping between send-addresses and NEWS coordinates. If g is a geometry, a is an axis number, and s is a send-address, then *extract-news-coordinate*(g, a, s) is the coordinate within geometry g of processor s along the NEWS axis described by a.

A related operation, *deposit-news-coordinate*, may be used to construct a send-address given a set of coordinates by incrementally modifying a send-address one coordinate at a time. If g is a geometry, s is a send-address (for a processor in that geometry), a is an axis number, and c is a coordinate along that axis, then *deposit-news-coordinate*(g, s, a, c) is a new send address s' such that

$$\textit{extract-news-coordinate}(g, a', s') = \begin{cases} c, & \text{if } a' = a \\ \textit{extract-news-coordinate}(g, a', s), & \text{if } a' \neq a \end{cases}$$

In other words, *deposit-news-coordinate*(g, s, a, c) computes a new send-address that has exactly the same NEWS coordinates as s *except* for the coordinate on axis a, which is altered to be c.

Another related operation, *make-news-coordinate*, constructs, 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. If g is a geometry, a is an axis number, and c is a coordinate along a, then *make-news-coordinate*(g, a, c) is s, the send-address of the processor with coordinate c along the NEWS axis a within geometry g and with all other coordinates held at zero. Thus, given a set of zero coordinates of *rank*(g), s',

$$\textit{make-news-coordinate}(g, a, c) = \textit{deposit-news-coordinate}(g, s', a, c) = s$$

In other words, *make-news-coordinate* is the same as *deposit-new-coordinate* except that it does not need a send-address operand.

Frequently it is useful to represent several NEWS coordinate values in a single integer called a *multi-coordinate*. Certain Paris operations, notably the multispread series, take a multi-coordinate as one operand. A multi-coordinate requires no more bits for its representation than a send address.

There are two abstract operations, *extract-multi-coordinate* and *deposit-multi-coordinate*, for accessing and altering multi-coordinates. They are analogous to *extract-news-coordinate* and *deposit-news-coordinate*, the difference being simply that a multi-coordinate contains values for several news coordinates.

Suppose that g is a geometry, A is an axis-set, and s and t are send-addresses, and let

$$s' = \textit{deposit-multi-coordinate}(g, s, A, \textit{extract-multi-coordinate}(g, A, t))$$

Then s' is the same as s except that coordinates for axes in A have been replaced by corresponding coordinates extracted from t. More formally,

$$\textit{extract-news-coordinate}(g, a, s') = \begin{cases} \textit{extract-news-coordinate}(g, a, s), & \text{if } a \notin A \\ \textit{extract-news-coordinate}(g, a, t), & \text{if } a \in A \end{cases}$$

Certain Paris instructions, most notably CM:multispread-copy-1L, require a multi-coordinate as an argument. The simplest way to construct such an argument is to construct a send-address and then use CM:fe-extract-multi-coordinate.

The following routines define the relationship between a processor whose send-address is k and its neighbors in a NEWS grid.

function *news-neighbor*(g, k, *axis*, *direction*) is
  return *news-relative*(g, k, *axis*, *direction*, 1)

function *news-relative*(g, k, *axis*, *direction*, *distance*) is
  case *direction* of
    :upward : let x = (*extract-news-coordinate*(g, *axis*, k) + *distance*)
    :downward : let x = (*extract-news-coordinate*(g, *axis*, k) − *distance*)
  let $x' = x$ mod *geometry-axis-length*(g, *axis*)
  return *deposit-news-coordinate*(g, k, *axis*, x')

5.16 Scan, Reduce, Spread, and Multispread

These operations provide extremely powerful combinations of communication and computation in regular patterns on multidimensional grids.

$$\text{CM:}\left\{\begin{array}{l}\text{scan-with}\\ \text{reduce-with}\\ \text{spread-with}\\ \text{multispread}\end{array}\right\}\left\{\begin{array}{l}\text{-copy}\\ \text{-logand}\\ \text{-logior}\\ \text{-logxor}\\ \left\{\begin{array}{l}\text{-s-}\\ \text{-u-}\\ \text{-f-}\end{array}\right\}\left\{\begin{array}{l}\text{add}\\ \text{min}\\ \text{max}\end{array}\right\}\end{array}\right\}\text{-1L}$$

CM:scan-with-f-multiply -1L

CM:enumerate -1L

In a scan operation, every selected processor receives the result of combining source fields from many processors. The reduce and spread operations are special cases of scans that are particularly useful and can be made especially fast. The multispread operations are a generalization of spread operations.

A scan operation requires that a NEWS axis be specified. The processors are thereby divided into disjoint ordered sets of processors called *scan classes.* Two processors belong to the same scan class if their NEWS coordinates differ only along that axis, and they are ordered by their coordinates along that axis. Only active processors participate in a scan operation; the active processors within a scan class are referred to as the *scan subclass* within that scan class.

Not all the processors in a scan class contribute to the result computed for a given processor. A scan class may be taken whole, or it may be divided into pieces in one of two ways. Each such piece is called a *scan set*, and every processor belongs to just one scan set. The scan set chosen for each processor is controlled by the *smode* operand and by the purpose it assigns to the *sbit* operand.

- If *smode* is :none, then there is no one-bit field, and the *sbit* operand is ignored. The scan set for a processor k is the entire scan subclass for k.

- If *smode* is :segment-bit, then the *sbit* field is a "segment bit." Operationally speaking, a processor (selected or not) is the lowest-addressed processor in a segment if either it is the lowest-addressed processor in its scan class or if its *sbit* field is 1. The segment bit therefore divides a scan class unconditionally (that is, without respect to context) into segments, and a scan operation is done within each segment. There are two remarkable points here. First, the way in which a segment bit divides a scan class does not depend on either the *context-flag* or the direction of the scan. Second, values from one segment never contribute to the result for any processor in another segment.

- If *smode* is :start-bit, then the *sbit* field is a "start bit." Operationally speaking, in each selected processor in which this bit is 1, the scan operation will start over again. The start bit therefore divides a scan subclass into pieces, and a scan operation is done within each piece. These pieces differ from the segments determined by a segment bit. There are three remarkable points here. First, the start bit is examined only in selected processors. Second, the way in which a start bit divides a scan subclass depends on the direction of the scan. Third, for an exclusive scan, a selected processor whose start bit is 1 will receive the identity for the combining operation only if no other selected processor in the same scan subclass precedes it in the ordering; otherwise, it will receive the combined values from all processors in the piece preceding it in the ordering.

A scan operation furthermore behaves as if all the processors were passed over ("scanned") in linear order; therefore the result computed for a given processor may depend only on processors below it in the ordering, or only on processors above it, depending on the direction of the scan. For each processor k, the *direction* and *inclusion* operands determine which processors within the scan set for k can potentially contribute to the result for k. This final, most narrowed set of potential contributors is called the *scan subset* for k.

If *direction* is :upward, then the scan set for processor k will contain only processors below k in the ordering. If *direction* is :downward, then the scan set for k will contain only processors above k in the ordering.

If *inclusion* is :exclusive, then the scan set for processor k will not contain k itself. If *inclusion* is :inclusive, then the scan set for k will contain k itself.

The set of processors whose *source* fields actually do contribute to the *dest* field of processor k is called the *scan subset* for k. This will be a subset of the scan set for k (possibly the entire scan set).

These concepts are embodied in the following pseudo-code routines, which are used in the Paris Dictionary to describe the behavior of scan and other operations. These routines define scan classes in terms of the more general concept of a *hyperplane*, which is any subset of the processors obtained by holding some NEWS coordinates fixed while letting the others range freely over their respective axes. (The *hyperplane* routine is also used in the pseudo-code descriptions of the multispread operations.)

function $\mathit{hyperplane}(g, k, \mathit{axis\text{-}set})$ is
  let $\mathit{other\text{-}axes} = \{\, a \mid 0 \le a < \mathit{rank}(g) \,\} \setminus \mathit{axis\text{-}set}$
  let $c = \mathit{extract\text{-}multi\text{-}coordinate}(g, \mathit{other\text{-}axes}, k)$
  return $\{\, m \mid m \in \mathit{current\text{-}vp\text{-}set} \wedge \mathit{extract\text{-}multi\text{-}coordinate}(g, \mathit{other\text{-}axes}, m) = c \,\}$

function $\mathit{scan\text{-}class}(g, k, \mathit{axis})$ is
  return $\mathit{hyperplane}(g, k, \{\, \mathit{axis} \,\})$

function $\mathit{scan\text{-}subclass}(g, k, \mathit{axis})$ is
  return $\{\, m \mid m \in \mathit{scan\text{-}class}(g, k, \mathit{axis}) \wedge \mathit{context\text{-}flag}[m] = 1 \,\}$

function *scan-set*$(g, k, axis, direction, smode, sbit)$ is
  let $C =$ *scan-subclass*$(g, k, axis)$
  function $coord(s) =$ *extract-news-coordinate*$(g, axis, s)$
  case $(smode)$ of
    (:none) :
      return C
    (:segment-bit) :
      let $Q = \{ m \mid m \in hyperplane(g, k, \{ axis \}) \wedge (sbit[m] = 1 \}$
      return $\{ m \mid m \in C \wedge \neg\exists j : (j \in Q \wedge coord(m) < coord(j) \leq coord(k)) \}$
    (:start-bit) :
      let $Q = \{ m \mid m \in hyperplane(g, k, \{ axis \}) \wedge (sbit[m] = 1 \}$
      case $(direction)$ of
        (:upward) :
          return $\{ m \mid m \in C \wedge \neg\exists j : (j \in (C \cap Q) \wedge coord(m) < coord(j) \leq coord(k)) \}$
        (:downward) :
          return $\{ m \mid m \in C \wedge \neg\exists j : (j \in (C \cap Q) \wedge coord(k) \leq coord(j) < coord(m)) \}$

function *scan-subset*$(g, k, axis, direction, inclusion, smode, sbit)$ is
  let $S =$ *scan-subset*$(g, k, axis, direction, smode, sbit)$
  function $coord(s) =$ *extract-news-coordinate*$(g, axis, s)$
  case $(direction, inclusion)$ of
    (:upward, :exclusive) : return $\{ m \mid m \in S \wedge coord(m) < coord(k) \}$
    (:upward, :inclusive) : return $\{ m \mid m \in S \wedge coord(m) \leq coord(k) \}$
    (:downward, :exclusive) : return $\{ m \mid m \in S \wedge coord(m) > coord(k) \}$
    (:downward, :inclusive) : return $\{ m \mid m \in S \wedge coord(m) \geq coord(k) \}$

The following table shows the results computed for various operand combinations for a scan with unsigned addition over a set of values all of which are 1.

scan-with-u-add		*context-flag*	1	1	1	1	0	0	0	1	1	0	0	1	1	1	0
		sbit	0	0	1	0	0	0	1	0	0	0	0	0	1	0	0
		source	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
direction	*inclusion*	*smode*															
:upward	:exclusive	:none	0	1	2	3				4	5			6	7	8	
:downward	:exclusive	:none	8	7	6	5				4	3			2	1	0	
:upward	:inclusive	:none	1	2	3	4				5	6			7	8	9	
:downward	:inclusive	:none	9	8	7	6				5	4			3	2	1	
:upward	:exclusive	:segment-bit	0	1	0	1				0	1			2	0	1	
:downward	:exclusive	:segment-bit	1	0	1	0				2	1			0	1	0	
:upward	:inclusive	:segment-bit	1	2	1	2				1	2			3	1	2	
:downward	:inclusive	:segment-bit	2	1	2	1				3	2			1	2	1	
:upward	:exclusive	:start-bit	0	1	2	1				2	3			4	5	1	
:downward	:exclusive	:start-bit	2	1	5	4				3	2			1	1	0	
:upward	:inclusive	:start-bit	1	2	1	2				3	4			5	1	2	
:downward	:inclusive	:start-bit	3	2	1	5				4	3			2	1	1	

A spread operation is like a scan, except that rather than producing "intermediate" or "running" results by using scan subsets, every processor gets the result of combining the values from every processor in the scan subclass.

A reduce operation is like a spread, except that instead of storing the result in every processor in the scan subclass, it stores the result into only one specified processor of the scan class.

A multispread operation is like a spread, but allows hyperplanes of any rank, not just of rank 1, to serve as the scan classes. In this manner, for example, a single value within each hyperplane can be replicated throughout its hyperplane.

5.17 Global Reduction Operations

A global operation combines a number of values in much the same manner as a scan or reduce operation, but delivers the result to the front end rather than storing it in a processor field.

$$\text{CM:global}\left\{\begin{array}{l}\text{-logand}\\ \text{-logior}\\ \text{-logxor}\\ \left\{\begin{array}{l}\text{-s-}\\ \text{-u-}\\ \text{-f-}\end{array}\right\}\left\{\begin{array}{l}\text{add}\\ \text{min}\\ \text{max}\end{array}\right\}\\ \text{u-max}\left\{\begin{array}{l}\text{-s-}\\ \text{-u-}\end{array}\right\}\text{-intlen}\end{array}\right\}\text{-1L}$$

All the usual combining operations are provided. In addition, the compound operation max-intlen is provided for efficiency; it is much faster than than a separate integer-length operation followed by a global-max operation.

5.18 Memory Data Transfers

These operations simply transfer data between a field in the processor array and the front end.

$$\text{CM:}\left\{\begin{array}{l}\text{s-}\\ \text{u-}\\ \text{f-}\end{array}\right\}\left\{\begin{array}{l}\text{read-from}\\ \text{write-to}\end{array}\right\}\left\{\begin{array}{l}\text{-processor}\\ \text{-news-array}\end{array}\right\}\text{-1L}$$

The operations read-from-processor and write-to-processor each transfer a single datum (integer or floating-point).

The operations read-from-news-array and write-to-news-array can transfer entire arrays or subarrays. Their implementation is optimized for relatively high throughput.

5.19 The LEDS

One of the most attractive features of a Connection Machine system is the array of blinking lights on the faces of its cabinet.

CM:set-system-leds-mode

This operation specifies whether the lights are to be blinked automatically, or turned on and off under user program control.

$$\text{CM:latch-leds}\left\{\begin{array}{c}\sim\\ \text{-always}\end{array}\right\}$$

These operations turn lights on and off according to the contents of a one-bit data field.

5.20 Front End Operations

Programs that use Paris operations frequently need to perform certain calculations on the front end that are not easily expressed in the host programming language. These operations are provided as part of the Paris library interface.

$$\text{CM:fe-}\left\{\begin{array}{l}\text{from-gray-code}\\ \text{to-gray-code}\\ \text{extract-news-coordinate}\\ \text{extract-multi-coordinate}\\ \text{deposit-news-coordinate}\\ \text{make-news-coordinate}\end{array}\right\}$$

These operations deal primarily with Gray codes and NEWS coordinates.

5.21 Environmental Interface

These operations pertain to allocating, deallocating, initializing, and debugging the Connection Machine.

$$
\text{CM:}\left\{\begin{array}{l}
\text{attach} \\
\text{attached} \\
\text{cold-boot} \\
\text{detach} \\
\text{init} \\
\text{power-up} \\
\text{reset-timer} \\
\text{set-safety-mode} \\
\text{start-timer} \\
\text{stop-timer} \\
\text{time} \\
\text{warm-boot}
\end{array}\right\}
$$

The **attach** operation is used to attach the front end process to a specified portion of all Connection Machine processors.

The **attached** operation returns true if the front end process actually has Connection Machine processors attached for use.

The **cold-boot** operation is used to initialize the Connection Machine hardware allocated to the executing front end.

The **detach** operation frees attached Connection Machine processors from the currect front end process.

The **init** operation is used by the C/Paris and Fortran/Paris interfaces to initialize the Connection Machine hardware.

The **power-up** operation resets the Nexus, causing all front-end computers to become logically detached from the Connection Machine system.

The **set-safety-mode** operation allows the user to specify the level of run-time error checking to be performed by the Paris interface.

The **time** family of operations are used to measure both the execution and the elapsed time taken by other operations.

The **warm-boot** operation is used by the Lisp/Paris interface to reinitialize the Connection Machine system without disturbing user memory.

Chapter 6

The C/Paris Interface

Paris is used as a set of variables, subroutines, and macros within a program that may be written in any one of a number of languages. This chapter explains how to call Paris instructions from C programs.

6.1 C/Paris Header Files

Type specification statements required for programs that access the C/Paris interface are given in the header file named

`/usr/include/cm/paris.h`

This header file contains four kinds of declarations that provide an environment for calling Paris instructions from C.

- Type declarations define new data types (struct types, for example) needed for communication with certain Paris operations.
- Function declarations define the result types of all C/Paris function subprograms.
- Variable declarations define configuration variables that provide access to the state of the Connection Machine system.
- #define statements define symbolic numeric constants to be used as arguments to certain C/Paris subprogram calls.

These declarations are discussed in more detail in the following sections.

6.2 C/Paris Instruction Names and Argument Types

This section describes how to call these instructions from C and what types of arguments to pass them.

The instruction names and other names that appear in this document are spelled in a form acceptable to Lisp (an arbitrary choice in order to have *some* common denominator for the dictionary). Each name is easily converted to the corresponding C name using the following two-part rule:

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

This usually results in a name written in mixed case (some letters uppercase and some lowercase). The name must be written in exactly that way, for C identifiers are case-sensitive. (Although Lisp is *not* case-sensitive, all identifiers appearing in Lisp form in this document are written in mixed case so as to produce the correct C name after applying the conversion rules.)

Chapter 9 describes each of the Paris instructions in terms of its arguments, its effect on operand fields residing in Connection Machine memory, and the result (if any) that it returns to the front end. The same argument name is often used in several different instruction definitions, but arguments with the same name always have the same type (as viewed by the front-end C program). For example, *dest* is used throughout to represent the field-id of a destination field; the field itself may be a floating-point or an integer field, the width of which is specified by other arguments to the instruction, but to the C program the argument is always simply a field-id.

Following is a brief description of the major classes of arguments that can be passed to subprograms of the C/Paris interface.

6.2.1 Id Types

These are values that should be treated as abstract entities, or "black boxes." They are created using special Paris instructions, and their actual values have no significance to the calling C program; they are simply tokens that may be passed to other Paris routines.

vp-set-id

A value representing a virtual processor set. Its C type is CM_vp_set_id_t.

geometry-id

A value representing a geometry with a particular shape. Its C type is CM_geometry_id_t.

field-id

A value representing a field allocated on the CM. Its C type is CM_field_id_t.

6.2.2 Operand Field Addresses

Most Paris operations require one or more field-id's to indicate one or more regions of Connection Machine memory to be processed. Such field-id's are obtained from memory allocation calls. Their C type is CM_field_id_t.

dest, *source*, *source1*, *source2*

These field-ids specify fields to be used as source or destination operands of an instruction.

send-address

This argument specifies a field that itself contains, within each processor, the send address of a processor (possibly the same one, possibly another).

news-coordinate

This argument specifies a field that itself contains, within each processor, the NEWS coordinate of a processor (possibly the same one, possibly another).

notify

A field-id for a 1-bit field to hold a result indicating receipt of a message by a send instruction.

sbit

A field-id for a 1-bit field that indicates how Paris scan operations should divide processors into logical groups.

6.2.3 Immediate Operands

These arguments are scalar values that participate in Paris operations as if they were first copied to every Connection Machine processor and then operated upon as if a field-id had been supplied. Paris operations that take "immediate" operand values of this sort usually have "constant" or "const" in their names.

source-value, source2-value

A (front-end) value or variable to be supplied as input to an instruction on the CM. The type of value passed depends on the instruction to which it is passed. The C type of such an immediate operand is long for a signed integer value, unsigned long for a signed integer value, or double for a floating-point value.

send-address-value

An integer, the send address of a single particular processor. The C type of such an immediate operand is CM_sendaddr_t.

news-coordinate-value An integer, the NEWS coordinate of a single particular processor. The C type of such an immediate operand is unsigned long.

6.2.4 Operand Field Lengths

These are integer values that specify the widths of source and destination operand fields on the CM. Their C type is unsigned.

len, slen, slen1, slen2, dlen

An integer value designating the length (in bits) of a source field that will be treated by the operation as a bit field, a signed integer, or an unsigned integer. It is not unusual for this value to be 32 to match the size of C long variables on the front end, but other lengths may be used as well—longer ones for additional precision, shorter ones for improved speed.

s, *ds*, *ss*

An integer value designating the significand length of a floating-point field. For single-precision (C type float) fields, this value should be 23; for double-precision (C type double) fields, the value should be 52.

e, *de*, *se*

An integer value designating the exponent length of a floating-point field. For single-precision (C type float) fields, this value should be 8; for double-precision (C type double) fields, the value should be 11.

6.2.5 Miscellaneous Signed and Unsigned Values

Both signed and unsigned Paris quantities are represented in C by variables and values whose C type is unsigned long. These are variously referred to, depending on their roles within particular operations, under the following names:
offset, *axis*, *axis-length*, *coordinate*, *rank*, *multi-coordinate*

6.2.6 Bit Sets and Masks

Arguments representing sets taken from universes of up to 31 elements are represented as integer values, where the bit whose value is 2^j is 1 to indicate that element j is in the set. Their C type is unsigned long.

At present, the only universe of interest in Paris is *axis-mask*, the set of axes for a given geometry.

6.2.7 Vectors of Integers

These arguments should be represented as C one-dimensional arrays whose elements are of C type unsigned. The maximum size of these vectors is 31.
axis-vector, *start-vector*, *offset-vector*, *end-vector*, *dimension-vector*

6.2.8 Multi-dimensional Front-end Arrays

Multi-dimensional front-end arrays of any C integer or floating-point type can be transferred to and from CM memory using a single instruction (see section 5.18).
front-end-array

Such an array is passed simply by mentioning the name of the array.

6.2.9 Symbolic Values

The symbolic constants defined in #define statements in the C/Paris header file should be used when supplying values for these arguments:

direction

One of the values CM_upward or CM_downward, indicating the direction of a scan, NEWS, or other instruction.

inclusion

One of the values CM_exclusive or CM_inclusive, indicating the boundaries of a scan instruction.

smode

One of the values CM_none, CM_start_bit, or CM_segment_bit, indicating how a scan operation is to be partitioned.

There are other symbolic values as well, but these are the most important. All names are formed by the standard rule: starting from a Lisp name such as :start-bit, add "CM" to the front and then convert colons and hyphens to underscores, yielding CM_start_bit.

6.3 C/Paris Configuration Variables

The configuration variables provide access to information about the configuration of the Connection Machine system. See section ?? for a list. The C/Paris interface makes these variables accessible through variables declared in the C/Paris header file. They are initialized in an application program by a call to the subroutine CM_init and should not be changed by an application program.

Each configuration variable is a numeric value that is constant over the course of a session (from one cold boot operation to the next), or varies from one Connection Machine configuration to another. For example, CM_physical_processors_limit is a value that depends upon the size of the Connection Machine to which the application is attached.

Numeric values that are constant for a given release of the CM System Software are given in #define statements.

6.4 Calling Paris from C

This section describes how to build C programs that access the Paris instruction set using the C/Paris interface. Such programs must manage the dynamic allocation and deallocation of Connection Machine fields directly. This section describes the form of C main programs and subprograms that call the C/Paris interface, as well as the steps involved in compiling and linking such programs.

The following code fragment illustrates the structure of a C main program that calls Paris instructions.

```
#include <cm/paris.h>
⋮
main() {
  CM_init();
  ⋮
  CM_paris_instruction(...);
  ⋮
  if ( CM_configuration_variable > limit ) ...
```

```
  ⋮
}
```

Note that the call to CM_init is required prior to any other calls to Paris instructions.

The following code fragment illustrates the structure of a C subroutine subprogram that calls Paris instructions.

```
#include <cm/paris.h>
⋮
float test() {
  ⋮
  CM_paris_instruction(...);
  ⋮
  if ( CM_configuration_variable > limit ) ...
  ⋮
}
```

It looks exactly like a main program in its use of Paris, *except* that a subprogram should not call CM_init.

Use the following command to compile and link these program units:

```
% cc main.c test.c -lparis
```

To compile and link these program units for execution under the simulator, use the following cc command:

```
% cc main.c test.c -lparissim
```

Note that there should be no space between the -l option and its argument.

Chapter 7

The Fortran/Paris Interface

Paris is used as a set of variables and subroutines within a program that may be written in any one of a number of languages. This chapter explains how to call Paris instructions from Fortran programs, especially those compiled by VAX Fortran and Sun Fortran.

The Fortran/Paris interface is itself an interface to C/Paris (see chapter 6).

7.1 Fortran/Paris Header Files

Type specification statements required for programs that access the Fortran/Paris interface are given in the header file named

`/usr/include/cm/paris-configuration-fort.h`

This header file contains three kinds of declarations that provide an environment for calling Paris instructions from Fortran.

- Type specification statements define the result types of all Fortran/Paris function subprograms.
- A declaration of a common block named cmval defines configuration variables that provide access to the state of the Connection Machine system.
- PARAMETER statements define symbolic numeric constants to be used as arguments to certain Fortran/Paris subprogram calls.

These declarations are discussed in more detail in the following sections.

7.2 Fortran/Paris Instruction Names and Argument Types

This section describes how to call these instructions from Fortran and what types of arguments to pass them.

The instruction names and other names that appear in this document are spelled in a form acceptable to Lisp (an arbitrary choice in order to have *some* common denominator for the dictionary). Each name is easily converted to the corresponding Fortran name using the following two-part rule:

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

It is also permissible to convert names to entirely uppercase letters if desired, as Fortran identifiers are not case-sensitive.

Chapter 9 describes each of the Paris instructions in terms of its arguments, its effect on operand fields residing in Connection Machine memory, and the result (if any) that it returns to the front end. The same argument name is often used in several different instruction definitions, but arguments with the same name always have the same type (as viewed by the front-end Fortran program). For example, *dest* is used throughout to represent the field-id of a destination field; the field itself may be a floating-point or an integer field, the width of which is specified by other arguments to the instruction, but to the Fortran program the argument is always simply a field-id.

Following is a brief description of the major classes of arguments that can be passed to subprograms of the Fortran/Paris interface.

7.2.1 Id Types

These are integer values that should be treated as abstract entities, or "black boxes." They are created using special Paris instructions, and their actual values have no significance to the calling Fortran program; they are simply tokens that may be passed to other Paris routines. Their Fortran type is INTEGER.

vp-set-id

An integer value representing a virtual processor set.

geometry-id

An integer value representing a geometry with a particular shape.

field-id

An integer value representing a field allocated on the CM.

7.2.2 Operand Field Addresses

Most Paris operations require one or more field-id's to indicate one or more regions of Connection Machine memory to be processed. Such field-id's are obtained from memory allocation calls. Their Fortran type is INTEGER.

dest, *source*, *source1*, *source2*

These field-ids specify fields to be used as source or destination operands of an instruction.

send-address

This argument specifies a field that itself contains, within each processor, the send address of a processor (possibly the same one, possibly another).

news-coordinate

This argument specifies a field that itself contains, within each processor, the NEWS coordinate of a processor (possibly the same one, possibly another).

notify

A field-id for a 1-bit field to hold a result indicating receipt of a message by a send instruction.

sbit

A field-id for a 1-bit field that indicates how Paris scan operations should divide processors into logical groups.

7.2.3 Immediate Operands

These arguments are scalar values that participate in Paris operations as if they were first copied to every Connection Machine processor and then operated upon as if a field-id had been supplied. Paris operations that take "immediate" operand values of this sort usually have "constant" or "const" in their names.

The Fortran type of such an immediate operand is INTEGER for an integer value, or DOUBLE-PRECISION for a floating-point value.

source-value, source2-value

A (front-end) value or variable to be supplied as input to an instruction on the CM. The type of value passed depends on the instruction to which it is passed.

send-address-value

An integer, the send address of a single particular processor.

news-coordinate-value An integer, the NEWS coordinate of a single particular processor.

7.2.4 Operand Field Lengths

These are integer values that specify the widths of source and destination operand fields on the CM. Their Fortran type is INTEGER.

len, *slen*, *slen1*, *slen2*, *dlen*

An integer value designating the length (in bits) of a source field that will be treated by the operation as a bit field, a signed integer, or an unsigned integer. It is not unusual for this value to be 32 to match the size of Fortran INTEGER variables on the front end, but other lengths may be used as well—longer ones for additional precision, shorter ones for improved speed.

s, *ds*, *ss*

An integer value designating the significand length of a floating-point field. For single-precision (Fortran type REAL) fields, this value should be 23; for double-precision (Fortran type DOUBLE PRECISION) fields, the value should be 52.

e, *de*, *se*

An integer value designating the exponent length of a floating-point field. For single-precision (Fortran type REAL) fields, this value should be 8; for double-precision (Fortran type DOUBLE PRECISION) fields, the value should be 11.

7.2.5 Miscellaneous Signed and Unsigned Values

Both signed and unsigned Paris quantities are represented in Fortran by variables and values whose Fortran type is INTEGER. These are variously referred to, depending on their roles within particular operations, under the following names:
offset, *axis*, *axis-length*, *coordinate*, *rank*, *multi-coordinate*

7.2.6 Bit Sets and Masks

Arguments representing sets taken from universes of up to 31 elements are represented as integer values, where the bit whose value is 2^j is 1 to indicate that element j is in the set. Their Fortran type is INTEGER.

At present, the only universe of interest in Paris is *axis-mask*, the set of axes for a given geometry.

7.2.7 Vectors of Integers

These arguments should be represented as Fortran one-dimensional INTEGER arrays. The maximum size of these vectors is 31.
axis-vector, *start-vector*, *offset-vector*, *end-vector*, *dimension-vector*

7.2.8 Multi-dimensional Front-end Arrays

Multi-dimensional front-end arrays of Fortran type LOGICAL, INTEGER, REAL, or DOUBLE PRECISION can be transferred to and from CM memory using a single instruction (see section 5.18).
front-end-array

Such an array is passed simply by mentioning the name of the array.

7.2.9 Symbolic Values

The symbolic constants defined in PARAMETER statements in the Fortran/Paris header file should be used when supplying values for these arguments:

direction

One of the values CM_upward or CM_downward, indicating the direction of a scan, NEWS, or other instruction.

inclusion

One of the values CM_exclusive or CM_inclusive, indicating the boundaries of a scan instruction.

smode

One of the values CM_none, CM_start_bit, or CM_segment_bit, indicating how a scan operation is to be partitioned.

There are other symbolic values as well, but these are the most important. All names are formed by the standard rule: starting from a Lisp name such as :start-bit, add "CM" to the front and then convert colons and hyphens to underscores, yielding CM_start_bit.

7.3 Fortran/Paris Configuration Variables

The configuration variables provide access to information about the configuration of the Connection Machine system. See section ?? for a list. The Fortran/Paris interface makes these variables accessible through variables declared in the common block named cmval, defined by the Fortran/Paris header file. They are initialized in an application program by a call to the subroutine CM_init and should not be changed by an application program.

Each configuration variable is a numeric value that is constant over the course of a session (from one cold boot operation to the next), or varies from one Connection Machine configuration to another. For example, CM_physical_processors_limit is a value that depends upon the size of the Connection Machine to which the application is attached. Most of these configuration variables are declared to be of Fortran type INTEGER.

Numeric values that are constant for a given release of the CM System Software are also given in PARAMETER statements.

7.4 Calling Paris from Fortran

This section describes how to build Fortran programs that access the Paris instruction set using the Fortran/Paris interface. Such programs must manage the dynamic allocation and deallocation of Connection Machine fields directly. This section describes the form of Fortran main programs and subprograms that call the Fortran/Paris interface, as well as the steps involved in compiling and linking such programs.

The following code fragment illustrates the structure of a Fortran main program that calls Paris instructions.

```
      PROGRAM main
C     VAX Fortran or Sun Fortran
      ⋮
      INCLUDE '/usr/include/cm/paris-configuration-fort.h'
      CALL CM_init()
      ⋮
      CALL CM_paris_instruction(...)
      ⋮
      IF ( CM_configuration_variable .GT. limit ) ...
      ⋮
      END
```

Note that the call to CM_init is required prior to any other calls to Paris instructions.

The following code fragment illustrates the structure of a Fortran subroutine subprogram that calls Paris instructions.

```
      SUBROUTINE test
C     VAX Fortran or Sun Fortran
      ⋮
      INCLUDE '/usr/include/cm/paris-configuration-fort.h'
      ⋮
      CALL CM_paris_instruction(...)
      ⋮
      IF ( CM_configuration_variable .GT. limit ) ...
      ⋮
      END
```

It looks exactly like a main program in its use of Paris, *except* that a subprogram should not call CM_init.

Using VAX Fortran, the following command compiles and links these program units to run on the Connection Machine Model 2:

```
% fort main.for test.for -lparis
```

To compile and link these program units for execution under the simulator, use the following fort command:

```
% fort main.for test.for -lparissim
```

Note that there should be no space between the -l option and its argument.

The command to compile and link these program units using the Sun Fortran compiler is quite similar:

```
% f77 main.f test.f -lparis
```

To compile and link these VAX Fortran program units for execution under the simulator, use the following f77 command:

```
% f77 main.f test.f -lparissim
```

Note that there should be no space between the -l option and its argument.

Chapter 8

The Lisp/Paris Interface

Paris is used as a set of variables, subroutines, and macros within a program that may be written in any one of a number of languages. This chapter explains how to call Paris instructions from Lisp programs.

8.1 Lisp/Paris Instruction Names and Argument Types

This section describes how to call these instructions from Lisp and what types of arguments to pass them.

The instruction names and other names that appear in this document are spelled in a form acceptable to Lisp (an arbitrary choice in order to have *some* common denominator for the dictionary).

Although Lisp is *not* case-sensitive, all identifiers appearing in Lisp form in this document are written in mixed case so as to produce the correct C name after applying certain conversion rules. The Lisp programmer may write names entirely in uppercase letters or entirely lowercase letters, if desired.

Chapter 9 describes each of the Paris instructions in terms of its arguments, its effect on operand fields residing in Connection Machine memory, and the result (if any) that it returns to the front end. The same argument name is often used in several different instruction definitions, but arguments with the same name always have the same type (as viewed by the front-end Lisp program). For example, *dest* is used throughout to represent the field-id of a destination field; the field itself may be a floating-point or an integer field, the width of which is specified by other arguments to the instruction, but to the Lisp program the argument is always simply a field-id.

Following is a brief description of the major classes of arguments that can be passed to subprograms of the Lisp/Paris interface.

8.1.1 Id Types

These are values that should be treated as abstract entities, or "black boxes." They are created using special Paris instructions, and their actual values have no significance to the calling Lisp program; they are simply tokens that may be passed to other Paris routines.

vp-set-id

An integer value representing a virtual processor set.

geometry-id

A structure of type CM:geometry-id representing a geometry with a particular shape.

field-id

An integer value representing a field allocated on the CM.

8.1.2 Operand Field Addresses

Most Paris operations require one or more field-id's to indicate one or more regions of Connection Machine memory to be processed. Such field-id's are obtained from memory allocation calls. Their Lisp type is integer.

dest, *source*, *source1*, *source2*

These field-ids specify fields to be used as source or destination operands of an instruction.

send-address

This argument specifies a field that itself contains, within each processor, the send address of a processor (possibly the same one, possibly another).

news-coordinate

This argument specifies a field that itself contains, within each processor, the NEWS coordinate of a processor (possibly the same one, possibly another).

notify

A field-id for a 1-bit field to hold a result indicating receipt of a message by a send instruction.

sbit

A field-id for a 1-bit field that indicates how Paris scan operations should divide processors into logical groups.

8.1.3 Immediate Operands

These arguments are scalar values that participate in Paris operations as if they were first copied to every Connection Machine processor and then operated upon as if a field-id had been supplied. Paris operations that take "immediate" operand values of this sort usually have "constant" or "const" in their names.

The Lisp type of such an immediate operand is integer for an integer value, or float for a floating-point value (any of the several kinds of Common Lisp floating-point numbers may be supplied).

source-value, source2-value

A (front-end) value or variable to be supplied as input to an instruction on the CM. The type of value passed depends on the instruction to which it is passed.

send-address-value

An integer, the send address of a single particular processor.

news-coordinate-value An integer, the NEWS coordinate of a single particular processor.

8.1.4 Operand Field Lengths

These are integer values that specify the widths of source and destination operand fields on the CM. Their Lisp type is integer.

len, *slen*, *slen1*, *slen2*, *dlen*

An integer value designating the length (in bits) of a source field that will be treated by the operation as a bit field, a signed integer, or an unsigned integer. It is not unusual for the programmer to choose this value to match the size of Lisp fixnum variables on the front end, but other lengths may be used as well—longer ones for additional precision, shorter ones for improved speed.

s, *ds*, *ss*

An integer value designating the significand length of a floating-point field. Floating-point numbers of any size are supported, but certain values must be used for good performance on the hardware floating-point accelerator. For single-precision (Lisp type single-float) fields, this value should be 23; for double-precision (Lisp type double-float) fields, the value should be 52.

e, *de*, *se*

An integer value designating the exponent length of a floating-point field. Floating-point numbers of any size are supported, but certain values must be used for good performance on the hardware floating-point accelerator. For single-precision (Lisp type single-float) fields, this value should be 8; for double-precision (Lisp type double-float) fields, the value should be 11.

8.1.5 Miscellaneous Signed and Unsigned Values

Both signed and unsigned Paris quantities are represented in Lisp by variables and values whose Lisp type is integer. These are variously referred to, depending on their roles within particular operations, under the following names:
offset, *axis*, *axis-length*, *coordinate*, *rank*, *multi-coordinate*

8.1.6 Bit Sets and Masks

Arguments representing sets taken from universes of up to 31 elements are represented as integer values, where the bit whose value is 2^j is 1 to indicate that element j is in the set. Their Lisp type is integer.

At present, the only universe of interest in Paris is *axis-mask*, the set of axes for a given geometry.

8.1.7 Vectors of Integers

These arguments should be represented as Lisp vectors (one-dimensional arrays); they may be specialized vectors, capable of holding integers only, or general vectors, capable of holding any Lisp objects but into which only integers happen to have been stored. The maximum size of these vectors is 31.
axis-vector, *start-vector*, *offset-vector*, *end-vector*, *dimension-vector*

8.1.8 Multi-dimensional Front-end Arrays

Multi-dimensional front-end arrays, whether specialized or general, can be transferred to and from CM memory using a single instruction (see section 5.18).
front-end-array

Such an array is passed simply by mentioning the name of the array.

8.1.9 Symbolic Values

These symbolic constants should be used when supplying values for these arguments:

direction

One of the values :upward or :downward, indicating the direction of a scan, NEWS, or other instruction.

inclusion

One of the values :exclusive or :inclusive, indicating the boundaries of a scan instruction.

smode

One of the values :none, :start-bit, or :segment-bit, indicating how a scan operation is to be partitioned.

There are other symbolic values as well, but these are the most important.

8.2 Lisp/Paris Configuration Variables

The configuration variables provide access to information about the configuration of the Connection Machine system. See section ?? for a list. The Lisp/Paris interface makes these variables available. They are initialized in an application program by a call to subroutine CM:cold-boot and should not be changed by an application program.

Each configuration variable is a numeric value that is constant over the course of a session (from one cold boot operation to the next), or varies from one Connection Machine configuration to another. For example, CM:*pysical-processors-limit* is a value that depends upon the size of the Connection Machine to which the application is attached.

8.3 Calling Paris from Lisp

This section describes how to build Lisp programs that access the Paris instruction set using the Lisp/Paris interface. Such programs must manage the dynamic allocation and deallocation of Connection Machine fields directly. This section describes the form of Lisp main programs and subprograms that call the Lisp/Paris interface, as well as the steps involved in compiling and linking such programs.

The following code fragment illustrates the structure of a Lisp function program that calls Paris instructions.

```
(defun test (...)
  :
  (CM:paris-instruction ...)
  :
  (if (> CM:configuration-variable limit) ...)
  :
  )
```

Remember that CM:cold-boot should be called *once* before beginning a computation that uses Paris; it is not appropriate to call CM:cold-boot on entrance to every function.

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).

Lisp and Fortran are not case-sensitive, but C is. In this dictionary the Lisp names are written with both upper-case and lower-case letters, as appropriate, to allow easy translation into C syntax. Lisp also allows names of any length, but Paris names have been limited to 30 characters to satisfy C and Fortran conventions.

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 floating-point destination field.

source 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. If the *source* operand is a NaN, then it is copied unchanged.

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 signed integer destination field.

source — 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 ⟨overflow occurred in processor k⟩ 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.)

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 floating-point destination field.

source 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 *context-flag*[*k*] = 1 then
    dest[*k*] ← $\cos^{-1}$ *source*[*k*]
    if *source*[*k*] < −1 or *source*[*k*] > 1 then
      test-flag[*k*] ← 1
    else
      test-flag[*k*] ← 0

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

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 floating-point destination field.

source 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 $\langle$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.

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 floating-point destination field.

source1 The floating-point first source field.

source2 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 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 (always or *context-flag*[k] = 1) then
  dest[k] ← *source1*[k] + *source2*[k]
  if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

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 signed integer destination field.

source1 The signed integer first source field.

source2 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$[k] \leftarrow$ ⟨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 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer destination field.

source1 The unsigned integer first source field.

source2 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 *context-flag*$[k] = 1$ then
dest$[k] \leftarrow$ *source1*$[k] +$ *source2*$[k]$
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 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. The operation is performed properly in all cases; the constant need not be representable in 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 signed integer destination field.

source1 The signed integer first source field.

source2 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, 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, 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, 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 unsigned integer destination field.

source1 The unsigned integer first source field.

source2 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 *context-flag*$[k] = 1$ then
dest$[k] \leftarrow$ *source1*$[k]$ + *source2*$[k]$ + *carry-flag*$[k]$
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 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 signed integer destination field.
	source1	The signed integer first source field.
	source2	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$[k] \leftarrow$ ⟨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 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 *source1*, *source2*, *len*

Operands

dest	The unsigned integer destination field.
source1	The unsigned integer first source field.
source2	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-const-mult-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-add-mult-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-add-const-mult-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The floating-point destination field.

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

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

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

source3 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 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*$[k]$) $\times$ *source3*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

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 into memory. The various operand formats allow 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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, the newly offset field-id.	
Context	This operation is unconditional. It does not depend on *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 An unsigned integer, the new field-id.

Context This operation is unconditional. It does not depend on *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 specified length in the specified VP set and returns a unique identifier.

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

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

A new field of length *len* is allocated in 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	An unsigned integer, the new field-id.	
Context	This operation is unconditional. It does not depend on *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 specified length in the specified VP set and returns a unique identifier.

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

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

AREF

Fetches an array element specified by a per-processor index and copies it to a fixed destination.

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

Operands	*dest*	The destination field.
	array	The source array field.
	index	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 $\textit{context-flag}[k] = 1$ then
    if $\textit{index}[k] < \textit{index-limit}$ then
      let $p = \textit{index}[k] \times \textit{element-len}$
      $\textit{dest}[k] \leftarrow \textit{array}[k]\langle p : p + \textit{dlen} - 1\rangle$
      $\textit{test-flag}[k] \leftarrow 1$
    else
      $\textit{test-flag}[k] \leftarrow 0$

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* × *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

Fetches an array element specified by a per-processor index and copies it to 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 destination field.

array The source array field.

index 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 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*.

Overlap The fields *array* and *index* may overlap in any manner. However, the *array* and *index* 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 (always or $context\text{-}flag[k] = 1$) then
    if $index[k] < index\text{-}limit$ then
      let $r = geometry\text{-}total\text{-}vp\text{-}ratio(geometry(current\text{-}vp\text{-}set))$
      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
        $dest[k]\langle j\rangle \leftarrow array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle$
    else
      ⟨error⟩

This is a simple form of array reference, for 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 at least

$$\left(\textit{index-limit} + \left\lceil \frac{dlen}{32} \right\rceil - 1\right) \times 32$$

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.

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 $array + i \times 32$, where i is 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 organized in a strange way for fast access. The data within the array area is not organized in the same manner as for CM:aref; instead, the memory of one processor contains data belonging to several other processors, and data belonging to one processor is spread over the memories of several processors. This allows the special indexing hardware to operate more efficiently.

A region of memory set aside for an array of the format required by CM:aref32 should be accessed only through the operations CM:aref32 and CM:aset32, related operations such as CM:get-aref32, or operations that copy the array as a whole from all processors (such as I/O operations).

AREF32-SHARED

Fetches an array element specified by a per-processor index and copies it to a fixed destination. The array is stored in a special format that allows fast access, and 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 destination field.

array The source array field.

index 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 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*.

Overlap The fields *array* and *index* may overlap in any manner. However, the *array* and *index* 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 (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 $m = k \bmod (r \times 32)$
      let $i = \textit{index}[k]$
      let $a = \textit{field-length}(\textit{array})$
      for all j such that $0 \le j < \textit{dlen}$ do
        let $z = i + \left\lfloor \frac{j}{32} \right\rfloor$
        let $q = k - m + (j \bmod 32) \times r + \left\lfloor \frac{z}{a} \right\rfloor$
        let $b = z \bmod a$
        $\textit{dest}[k]\langle j \rangle \leftarrow \textit{array}[q]\langle b \rangle$

else
  ⟨error⟩

This is a simple form of array reference, for arrays stored in the memory of individual processors but accessed in such a way that many processor appear to share a single array. Each processor has an array index stored in the field *index*. This is used to index into an *array*. The length of the array in bits should be at least

$$\left\lceil \frac{\textit{index-limit}}{\textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))} \right\rceil$$

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 element indexed (or a portion of it) is copied into *dest* in all selected processors. Thus different processors may access different elements of the shared array.

A region of memory set aside for an array of the format required by CM:aref32-shared should be accessed only through the operations CM:aref32-shared and CM:aset32-shared, or operations that copy the array as a whole from all processors (such as I/O operations).

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 source field.
array	The destination array field.
index	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

Fetches an array element from a fixed source and copies it to a destination specified by a per-processor index. The 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 source field.

array The destination array field.

index 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.

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*.

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 $context\text{-}flag[k] = 1$ then
    if $index[k] < index\text{-}limit$ then
      let $r = geometry\text{-}total\text{-}vp\text{-}ratio(geometry(current\text{-}vp\text{-}set))$
      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
        $array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle \leftarrow source[k]\langle j \rangle$
    else
      ⟨error⟩

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 at least

$$\left(\textit{index-limit} + \left\lceil \frac{dlen}{32} \right\rceil - 1\right) \times 32$$

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.

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 $array + i \times 32$, where i is the unsigned number stored at *index*, in all selected processors. Even this is not quite accurate, because the array data is organized in a strange way for fast access. The data within the array area is not organized in the same manner as for CM:aref; instead, the memory of one processor contains data belonging to several other processors, and data belonging to one processor is spread over the memories of several processors. This allows the special indexing hardware to operate more efficiently.

A region of memory set aside for an 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:get-aref32, or operations that copy the array as a whole from all processors (such as I/O operations).

ASET32-SHARED

Fetches an array element from a fixed source and copies it to a destination 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 source field.

array — The destination array field.

index — 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.

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*.

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 $context\text{-}flag[k] = 1$ then
    if $index[k] < index\text{-}limit$ then
      let $r = geometry\text{-}total\text{-}vp\text{-}ratio(geometry(current\text{-}vp\text{-}set))$
      let $m = k \bmod (r \times 32)$
      let $i = index[k]$
      let $a = field\text{-}length(array)$
      for all j such that $0 \leq j < dlen$ do
        let $z = i + \left\lfloor \frac{j}{32} \right\rfloor$
        let $q = k - m + (j \bmod 32) \times r + \left\lfloor \frac{z}{a} \right\rfloor$
        let $b = z \bmod a$
        $array[q]\langle b \rangle \leftarrow dest[k]\langle j \rangle$
    else
      ⟨error⟩

ASET32-SHARED

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*. The length of the array in bits should be at least

$$\left\lceil \frac{\textit{index-limit}}{\textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))} \right\rceil$$

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 *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 the shared array. 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.

A region of memory set aside for an array of the format required by CM:aset32-shared should be accessed only through the operations CM:aref32-shared and CM:aset32-shared, or operations that copy the array as a whole from all processors (such as I/O operations).

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 floating-point destination field.

source 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.

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 floating-point destination field.

source 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*] ← $\sinh^{-1}$ *source*[*k*]
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

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

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 floating-point destination field.

source 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
$dest[k] \leftarrow \tan^{-1} source[k]$

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

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 floating-point destination field.
source1	The floating-point y source field.
source2	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.

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
$dest[k] \leftarrow \tan^{-1} \frac{source1[k]}{source2[k]}$
else if $source2[k] < 0$ then
$dest[k] \leftarrow sign(source1[k]) \times \left(\pi - \tan^{-1} \left|\frac{source1[k]}{source2[k]}\right|\right)$
else if $source1[k] = 0 \wedge sign(source2[k]) > 0$ then
$dest[k] \leftarrow sign(source1[k]) \times 0$
else if $source1[k] = 0 \wedge sign(source2[k]) < 0$ then
$dest[k] \leftarrow sign(source1[k]) \times \pi$
else
$dest[k] \leftarrow sign(source1[k]) \times \frac{\pi}{2}$
if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

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.

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 floating-point destination field.

source 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 greater 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$[k] \leftarrow \tanh^{-1}$ *source*$[k]$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
    if *source*$[k] > 1$ then *test-flag*$[k] \leftarrow 1$
    otherwise *test-flag*$[k] \leftarrow 0$

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

ATTACH

Returns the number of physical processors attached.

Formats result ← CM:attach

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

interface The particular bus interface to be used. This argument is optional (actually a keyword argument in the Lisp interface).

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

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

This function is responsible for allocating Connection Machine processors for use by the front end. (To deallocate them, use CM:detach.)

The facility for attaching 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:attach is a function of several arguments. The first argument is optional, while the second is a keyword argument (against the possibility that more keyword arguments may be introduced in the future).

If the *physical-size* argument is not specified, then the smallest possible amount of hardware will be allocated; this will be either 8,192 or 16,384 physical processors. Otherwise the *physical-size* argument must be one of the following:

: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.

:ucc0, :ucc1, :ucc2, or :ucc3 Exactly the specified microcontroller port is to be attached, regardless of whether that port controls 8,192 or 16,384 physical processors. (This option is useful primarily for hardware diagnostic procedures.)

:ucc0-1, :ucc2-3, or :ucc0-3 Exactly the specified microcontroller ports (0 and 1, 2 and 3, or all four) are to be attached, regardless of the number of physical processors involved. (This option is useful primarily for hardware diagnostic procedures.)

(Note: the Lisp/Paris interface on a Symbolics Lisp Machine 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 new forms :8kp, :16kp, :32kp, and :64kp in code to ensure portability. The old forms will continue to be available for convenience in those Lisp implementations that will support them.)

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

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

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.

In the C/Paris and Fortran/Paris interfaces, the detaching operation is performed by a user command cmattach at shell level. See the *Front End Subsystems* manual or the cmattach man page.

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 *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.

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 floating-point destination field.

source 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
    $dest[k] \leftarrow \lceil source[k] \rceil$

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.

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 operations are conditional.
The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*[k] = 1) then
    test-flag[k] ← 0
    overflow-flag[k] ← 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 *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 3.6.
- 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-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 floating-point destination field.

source 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] ← cos *source*[k]

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

F-COSH

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

source 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 $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$

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

CREATE-DETAILED-GEOMETRY

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

Formats result ← CM:create-detailed-geometry *axis-descriptor-array*; *rank*

Operands *axis-descriptor-array* A front-end vector (one-dimensional array) of descriptors for the grid axes. In the Lisp interface, this may be a list of descriptors instead of an array of descriptors, at the user's option.

rank An unsigned integer, the rank (number of dimensions) of the *axis-descriptor-array*.

Result A geometry-id, identifying the newly created geometry.

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

CM:create-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 Lisp definitions of the type of the ordering component and of the descriptor are

```
(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 C definitions of the type of the ordering component and of the descriptor are shown below. The elements of the *axis_descriptor_array* should be pointers to type CM_axis_descriptor_t.

```
typedef enum {CM_news_order, CM_send_order} CM_axis_order_t;

typedef struct CM_axis_descriptor {
  unsigned long length;
  unsigned long weight;
  CM_news_order_t ordering;
  unsigned long on_chip_bits;
  unsigned long off_chip_bits;
} * CM_axis_descriptor_t;
```

(Actually, this structure has other components as well. Code should use the definition of CM_axis_descriptor_t from the cmtypes include file.)

The length component specifies the length of the axis; it must be a power of two. (This restriction may be removed in a future software release.)

The "on-chip-bits" and "off-chip-bits" components for an axis indicate how many physical hypercube dimensions should be used in laying out that axis of the grid. The physical hypercube dimensions are of two kinds: the four that are on-chip, connecting physical processors that are part of the same physical integerated circuit chip, and the rest, which are off-chip. The distinction matters when you're fine-tuning code for speed.

There are implementation restrictions (for the sake of speed) that all the on-chip hypercube dimensions for a given axis must be contiguous and that all the off-chip hypercube dimensions for a given axis must be contiguous. These restrictions are enforced by create-detailed-geometry as it lays out the axes.

If the "bits" components are zero, then values for them are calculated automatically. Such calculations take the specified weights into account. It is assumed that the frequencies of operations along a given axis are proportional to the weight of that axis. (If all weights are zero, it is assumed that all axes are used equally frequently.) For example, if in a given program, for a given geometry, North-South operations occur four times as frequently as East-West operations, then the North-South axis might be assigned a weight of 4 and the East-West axis a weight of 1 (or the weights might equally well be 12 and 3). These weights serve as only a rough but conveniently specified guide to the creation of geometries tuned for performance. For absolutely best tuning of performance, the user should specify all the "bits" components explicitly.

The ordering component 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 useful for specific applications such as FFT. Most operations are about as fast with either ordering, but get-from-news and send-to-news are significantly faster with :news-order. (In the future, other orderings may also be implemented if warranted by performance improvements.)

This operation returns a geometry-id for a newly created geometry. The length of axis j of the resulting geometry will be equal to the length component of *axis-descriptor-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.

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

CREATE-GEOMETRY

Creates a new geometry given the grid axis lengths.

Formats result ← CM:create-geometry *dimension-array*; *rank*

Operands *dimension-array* A front-end vector (one-dimensional array) 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*.

Result A geometry-id, identifying the newly created geometry.

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

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.

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.

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 *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 *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 *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 *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 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 unsigned integer send-address field.

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

coordinate The unsigned integer NEWS coordinate along the specified axis field.

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 33.

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 unconditional. It does not depend on *context-flag*.

Definition Return *deposit-news-coordinate*(*geometry*, *send-address*, *axis*, *coordinate*) where *deposit-news-coordinate* is as defined on page 33.

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.

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 *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.

F-DIVIDE

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

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-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-divide-always-3-1L	*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-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-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-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 floating-point destination field. This is the quotient.

source1 The floating-point first source field. This is the dividend.

source2 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 $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow source1[k]/source2[k]$
if $source2[k] = 0$ then $test\text{-}flag \leftarrow 1$
if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

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 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 segment bit or start bit (a one-bit 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 37.

See section 5.16 on page 34 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*

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 floating-point first source field.

source2 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] ← 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 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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 signed integer first source field.

source2 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer first source field.

source2 — 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 *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 equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 floating-point destination field.

source 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 ⟨overflow occurred in processor k⟩ 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.

EXTRACT-MULTI-COORDINATE

Determines the NEWS multi-coordinate of a processor specified by send-address.

Formats CM:extract-multi-coordinate-1L *geometry*, *dest*, *axis-mask*, *send-address*, *dlen*

Operands

geometry A geometry-id. This geometry determines the NEWS dimensions to be used.

dest The unsigned integer destination field.

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.

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
    let $axis\text{-}set = \{ m \mid 0 \le m < r \wedge (axis\text{-}mask\langle m \rangle = 1) \}$
    $dest[k] \leftarrow extract\text{-}multi\text{-}coordinate(geometry, axis\text{-}set, send\text{-}address)$

where *extract-multi-coordinate* is as defined on page 34.

This function calculates, within each selected processor, 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.

FE-EXTRACT-MULTI-COORDINATE

Calculates, on the front end, the NEWS multi-coordinate of a processor specified by send-address.

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 unconditional. It does not depend on *context-flag*.

Definition Let $axis\text{-}set = \{\, m \mid 0 \leq m < r \wedge (axis\text{-}mask\langle m \rangle = 1)\,\}$
Return $extract\text{-}multi\text{-}coordinate(geometry, axis\text{-}set, send\text{-}address)$
where *extract-multi-coordinate* is as defined on page 34.

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 unsigned integer destination field.

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.

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 33.

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 unconditional. It does not depend on *context-flag*.

Definition Return *extract-news-coordinate*(*geometry*, *axis*, *send-address*)
where *extract-news-coordinate* is as defined on page 33.

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.

FIELD-VP-SET

Returns the VP set associated with a field.

Formats result ← CM:field-vp-set *field*

Operands *field* 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 *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 floating-point destination field.

source 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] ← *source*[k]
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 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 floating-point destination field.

source 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 floating-point destination field.

source 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
$dest[k] \leftarrow \lfloor source[k] \rfloor$

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-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 signed integer destination field.
source	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
    $\mathit{dest}[k] \leftarrow \lfloor \mathit{source}[k] \rfloor$
    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 *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.

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 *context-flag*.

Definition Let $n = \textit{integer-length}(\textit{code})$

Return $\bigoplus_{j=0}^{n-1} \left\lfloor \frac{\textit{code}}{2^j} \right\rfloor$

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 unsigned integer destination field.

source 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-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 floating-point first source field.

source2 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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

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 signed integer first source field.

source2 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-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 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer first source field.

source2 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. The operation is performed properly in all cases; the constant need not be representable in 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 *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-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 *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 *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 *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 *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 ← 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 *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-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 *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 *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 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 destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

len The length of the *dest* and *source* fields.

Overlap The *send-address* and *source* may overlap in any manner. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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$[k] \leftarrow$ *source*[*send-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 witin 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 address within each source processor.

Formats CM:get-aref32-2L *dest*, *send-address*, *array*, *index*, *dlen*, *index-len*, *index-limit*

Operands *dest* The destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

array The source array field.

index The unsigned integer index into the array field. This must be a multiple of 32.

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*.

Overlap The *send-address* and *array* may overlap in any manner. The *dest* field may overlap with *send-address* or *array*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *array* only if within each processor at most one of them will be used.

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 $index[k] < index\text{-}limit$ then
      let $r = geometry\text{-}total\text{-}vp\text{-}ratio(geometry(current\text{-}vp\text{-}set))$
      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
        let $q = send\text{-}address[k] - m \times r + (j \bmod 32) \times r$
        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 destination field.

source 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 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 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 34.

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.

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 ← CM:global-f-add-1L *source*, *s*, *e*

Operands *source* 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 ← CM:global-s-add-1L *source*, *len*

Operands *source* 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 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	←	CM:global-count-bit	*source*
result	←	CM:global-count-bit-always	*source*

Operands — *source* 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 ← 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 $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{flag}[m] = 1 \}$
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 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	←	CM:global-logand-bit	*source*
result	←	CM:global-logand-bit-always	*source*

Operands *source* 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
  return $\left(\bigwedge_{m \in S} \textit{source}[m]\right)$ 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 ← 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 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	←	CM:global-logior-bit	*source*
result	←	CM:global-logior-bit-always	*source*

Operands *source* 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 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 $S = \mathit{current\text{-}vp\text{-}set}$
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
  return $\left(\bigvee_{m \in S} \mathit{source}[m] \right)$ 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 ← 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 $S = \{\, m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{flag}[m] = 1 \,\}$
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 ← CM:global-logxor-1L *source*, *len*

Operands *source* 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 $-2^{len} - 1$ 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 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 $\textit{context-flag}[k] = 1$ 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-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.

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 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.

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 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 $2^{len} - 1$ 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-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 $2^{len} - 1$ is returned.

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 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 -2^{len-1} 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-1L *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 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 $S = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return -2^{len-1} to front end
else
  let $R = \left(\max_{m \in S} \lceil \log_2 (1 + \textit{source}[m]) \rceil \right)$
  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

CM:u-integer-length-1L *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 ← CM:global-f-min-1L *source*, *s*, *e*

Operands	*source*	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 $R = \left(\min_{m \in S} \textit{source}[m]\right)$
  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] ← 1
      else
        test-flag[k] ← 0
  return R to front end

The CM:global-f-min 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.

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 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} 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 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 $2^{len-1} - 1$ is returned.

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 ← CM:global-u-min-1L *source*, *len*

Operands	*source*	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 0 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 *context-flag*[k] = 1 then
      if *source*[k] = R then
        test-flag[k] ← 1
      else
        test-flag[k] ← 0
  return R to front end

The CM:global-u-min operation returns the largest (that is, closest to $-\infty$) 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.

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 floating-point first source field.

source2 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*] ← 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 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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 signed integer first source field.

source2 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer first source field.

source2 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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 *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.

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 unsigned integer destination field.
- *source* — 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 $\textit{context-flag}[k] = 1$ then
    if $\textit{source}[k] \geq 0$ then $\textit{dest}[k] \leftarrow \lceil \log_2(\textit{source}[k] + 1) \rceil$
    else $\textit{dest}[k] \leftarrow \lceil \log_2(-\textit{source}[k]) \rceil$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$
    else $\textit{overflow-flag}[k] \leftarrow 0$

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 unsigned integer destination field.
	source	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-flag*$[k] = 1$ then
    dest$[k] \leftarrow \lceil \log_2(source[k] + 1) \rceil$
    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 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*.

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 *context-flag*.

The pseudo-random generator of numbers used by the operations CM:f-random-1L and cm:u-random-1L is initialized. The seed (a front-end integer, which must be non-zero) determines the initial state.

Note that CM:cold-boot effectively calls CM:initialize-random-generator with a seed based on the date and time of day.

In the Lisp/Paris interface, the *seed* argument is optional; if it is omitted, then a value similarly based on the date and time of day is used.

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 *context-flag*[k] = 1) then
flag[k] $\leftarrow \neg$ *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-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 field-id indicates a field allocated in the heap, and false otherwise.

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

This predicate allows a program to determine 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 field-id indicates a field allocated on the stack, and false otherwise.

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

This predicate allows a program to determine whether a given field has been allocated on the stack (as opposed to the heap).

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 *context-flag*.

This operation compares two stack fields and returns true if the second has been allocated more recently than the first.

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 signed integer destination field.

source 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 *context-flag*$[k] = 1$ then
    if *source*$[k] \geq 0$ then
      dest$[k] \leftarrow \lfloor\sqrt{source}\rfloor$
      test-flag$[k] \leftarrow 0$
    else
      dest$[k] \leftarrow \langle$unpredictable$\rangle$
      test-flag$[k] \leftarrow 1$
    if $\langle$overflow occurred in processor $k\rangle$ then *overflow-flag*$[k] \leftarrow 1$
    else *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 unsigned integer destination field.

source 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*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$
  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 source bit (a one-bit field).

Context The non-always operations are conditional.
The always operations are unconditional.

Definition Let $g = geometry(current\text{-}vp\text{-}set)$
Let $r = geometry\text{-}total\text{-}vp\text{-}ratio(g) \times 16$
Let $n = geometry\text{-}total\text{-}processors / r$
For all m such that $0 \le m < n$ do
if always then
turn on led m if and only if

$$\left(\bigvee_{j=0}^{r-1} source[m \times n + j] \right) = 0$$

else
turn on led m if and only if

$$\left(\bigvee_{j=0}^{r-1} (source[m \times n + j] \wedge context\text{-}flag[m \times n + j]) \right) = 0$$

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 floating-point first source field.

source2 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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 signed integer first source field.

source2 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 $\textit{context-flag}[k] = 1$ then
    if $\textit{source1}[k] \le \textit{source2}[k]$ 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 or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer first source field.

source2 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*$[k] \leq$ *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 or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 floating-point destination field.

source 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-flag*[*k*] = 1 then
dest[*k*] ← ln *source*[*k*]

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 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-overflow *source*

Operands *source* The source bit (a one-bit field).

Context This operation is conditional.

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

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.

LOGAND

Combines two source values with a bitwise logical AND operation, and places the result in the destination field.

Formats

CM:logand-2-1L	*dest/source1*, *source2*, *len*
CM:logand-3-1L	*dest*, *source1*, *source2*, *len*
CM:logand-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logand-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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

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$[k] \leftarrow$ *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 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 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
$flag[k] \leftarrow flag[k] \wedge source[k]$
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 with a bitwise logical AND operation, and places the result in the destination field.

Formats

CM:logandc1-2-1L	*dest*/*source1*, *source2*, *len*
CM:logandc1-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc1-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:logandc1-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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 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
    $dest[k] \leftarrow (\neg source1[k]) \wedge source2[k]$

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 with a bitwise logical AND operation, and places the result in the destination field.

Formats

CM:logandc2-2-1L	*dest/source1, source2, len*
CM:logandc2-3-1L	*dest, source1, source2, len*
CM:logandc2-constant-2-1L	*dest/source1, source2-value, len*
CM:logandc2-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The destination field.

source1 The first source field.

source2 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 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
    $dest[k] \leftarrow source1[k] \wedge (\neg source2[k])$

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 unsigned integer destination field.

source 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 $\langle$overflow occurred in processor $k\rangle$ 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 unsigned integer destination field.
- *source* — 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-flag*[k] = 1 then
dest[k] ← *count-of-one-bits*(*source*[k])
if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
else *overflow-flag*[k] ← 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 with a bitwise logical EQUIVALENCE operation, and places the result in the destination field.

Formats

CM:logeqv-2-1L	*dest/source1*, *source2*, *len*
CM:logeqv-3-1L	*dest*, *source1*, *source2*, *len*
CM:logeqv-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logeqv-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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

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
    $dest[k] \leftarrow \neg(source1[k] \oplus source2[k])$

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 with 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-3-1L	*dest*, *source1*, *source2*, *len*
CM:logior-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logior-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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

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
    $dest[k] \leftarrow source1[k] \vee source2[k]$

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 source bit (a one-bit field).

Context This operation is unconditional.

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

$$context\text{-}flag[k] \leftarrow context\text{-}flag[k] \vee source[k]$$

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 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-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-3-1L	*dest, source1, source2, len*
CM:lognand-constant-2-1L	*dest/source1, source2-value, len*
CM:lognand-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest — The destination field.

source1 — The first source field.

source2 — 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

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
    $\textit{dest}[k] \leftarrow \neg(\textit{source1}[k] \wedge \textit{source2}[k])$

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-3-1L	*dest*, *source1*, *source2*, *len*
CM:lognor-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:lognor-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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 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
$dest[k] \leftarrow \neg(source1[k] \vee source2[k])$

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-2-1L *dest*, *source*, *len*

Operands *dest* The destination field.

source 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 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
$dest[k] \leftarrow \neg source[k]$

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 with a bitwise logical inclusive OR operation, and places the result in the destination field.

Formats

CM:logorc1-2-1L	*dest/source1*, *source2*, *len*
CM:logorc1-3-1L	*dest*, *source1*, *source2*, *len*
CM:logorc1-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logorc1-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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

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$[k] \leftarrow (\neg$*source1*$[k]) \vee$ *source2*$[k]$

Each bit of the *dest* field is cleared if the corresponding bit of the *source1* field is 1 and 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 with a bitwise logical inclusive OR operation, and places the result in the destination field.

Formats

CM:logorc2-2-1L	*dest/source1*, *source2*, *len*
CM:logorc2-3-1L	*dest*, *source1*, *source2*, *len*
CM:logorc2-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logorc2-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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 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
    $dest[k] \leftarrow source1[k] \vee (\neg source2[k])$

Each bit of the *dest* field is cleared if the corresponding bit of the *source1* field is 0 and the corresponding bit of the *source2* field is 1; otherwise it is set.

LOGXOR

Combines two source values with 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-3-1L	*dest*, *source1*, *source2*, *len*
CM:logxor-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logxor-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The destination field.

source1 The first source field.

source2 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

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$[k] \leftarrow$ *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 floating-point first source field.

source2 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$[k] \leftarrow 1$
    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 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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 signed integer first source field.

source2 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 *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 less than the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer first source field.

source2 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] ← 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 the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 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 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 *context-flag*$[k] = 1$ then
dest$[k] \leftarrow$ *make-news-coordinate*(*axis*, *news-coordinate*)

where *make-news-coordinate* is as defined on page 33.

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 unconditional. It does not depend on *context-flag*.

Definition Return *make-news-coordinate*(*axis*, *news-coordinate*)

where *make-news-coordinate* is as defined on page 33.

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.

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 floating-point destination field.

source1 The floating-point first source field.

source2 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] ≥ *source2*[k] then
      dest[k] ← *source1*[k]
      test-flag[k] ← 0
    else
      dest[k] ← *source2*[k]
      test-flag[k] ← 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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 signed integer destination field.

source1 The signed integer first source field.

source2 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer destination field.

source1 The unsigned integer first source field.

source2 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*] ← 0
else
dest[*k*] ← *source2*[*k*]
test-flag[*k*] ← 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. The operation is performed properly in all cases; the constant need not be representable in 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 floating-point destination field.

source1 The floating-point first source field.

source2 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*] ≤ *source2*[*k*] then
      dest[*k*] ← *source1*[*k*]
      test-flag[*k*] ← 0
    else
      dest[*k*] ← *source2*[*k*]
      test-flag[*k*] ← 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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

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 signed integer destination field.

source1 The signed integer first source field.

source2 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer destination field.

source1 The unsigned integer first source field.

source2 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*] ≤ *source2*[*k*] then
      dest[*k*] ← *source1*[*k*]

test-flag[k] ← 0
else
dest[k] ← *source2*[k]
test-flag[k] ← 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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

S-MOD

The residue of one signed integer modulo another is placed in the destination field. Overflow is also computed.

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 signed integer residue field.

source1 The signed integer dividend field.

source2 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 *overflow-flag* is set if either the result cannot be represented in the destination field or the modulus 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
        if *source2*$[k] = 0$ then
            dest$[k] \leftarrow$ ⟨unpredictable⟩
        else
$$dest[k] \leftarrow source1[k] - source2[k] \times \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$
        if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
        else *overflow-flag*$[k] \leftarrow 0$

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 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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

U-MOD

The residue of one unsigned integer modulo another is placed in the destination field. Overflow is also computed.

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 unsigned integer residue field.

source1 The unsigned integer dividend field.

source2 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 *overflow-flag* is set if the modulus 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
    if *source2*$[k] = 0$ then
      dest$[k] \leftarrow$ ⟨unpredictable⟩
    else
      $$dest[k] \leftarrow source1[k] - source2[k] \times \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

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 are constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations.

The value of the destination is unpredictable if the divisor is zero.

The constant operand *source2-value* should be an unsigned integer front-end value. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 floating-point destination field.

source The floating-point source field.

source-value The floating-point source field. 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 $overflow\text{-}flag[k] \leftarrow 0$
  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 signed integer destination field.

source 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 CM:*maximum-integer-length*.

dlen For CM:s-move-1L, 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-move-1L, 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* may overlap in any manner.

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

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: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 unsigned integer destination field.

source 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 non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-move-1L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen For CM:u-move-1L, 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* may overlap in any manner.

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

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] ← *source*[k]
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0

MOVE

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*,

Operands *dest* 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 The fields *dest* and *source* 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
    $dest[k] \leftarrow sign\text{-}value \times (low\text{-}s\text{-}value + 2^{32} \times high\text{-}s\text{-}value) \times 2^{e\text{-}value}$

The three quantities $low\text{-}s\text{-}value + 2^{32} \times high\text{-}s\text{-}value$, *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 a source value into the destination field, reversing the order of the bits.

Formats CM:move-reversed-1L *dest*, *source*, *len*

Operands

dest	The destination field.
source	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 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 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-const-add-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-add-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-const-add-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The floating-point destination field.

source1 The floating-point first source field.

source2 The floating-point second source (multiplier) field.

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

source3 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 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*$[k]$) + *source3*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as floating-point numbers and then a third operand, *source3*, is added to the product. The result is stored into memory. The various operand formats allow 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). The constant is 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-const-sub-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-sub-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-const-sub-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The floating-point destination field.

source1 The floating-point first source field.

source2 The floating-point second source (multiplier) field.

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

source3 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

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*] ← (*source1*[*k*] × *source2*[*k*]) − *source3*[*k*]
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 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 into memory. The various operand formats allow 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). The constant is 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-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 floating-point destination field.

source1 The floating-point first source field.

source2 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-flag*[k] = 1) then
    dest[k] ← *source1*[k] × *source2*[k]
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

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 signed integer destination field.

source1 The signed integer first source field.

source2 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer destination field.

source1 The unsigned integer first source field.

source2 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

$$\begin{array}{l} \text{if } \textit{context-flag}[k] = 1 \text{ then} \\ \quad \textit{dest}[k] \leftarrow \textit{source1}[k] \times \textit{source2}[k] \\ \quad \text{if } \langle \text{overflow occurred in processor } k \rangle \text{ then } \textit{overflow-flag}[k] \leftarrow 1 \\ \quad \text{else } \textit{overflow-flag}[k] \leftarrow 0 \end{array}$$

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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 floating-point destination field.
source	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 $r = \textit{rank}()$

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 *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 ≤ j < rank(geometry(current-vp-set)), in any sequential order, do
  if axis-mask⟨j⟩ = 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 signed integer destination field.

source 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 $r = \textit{rank}()$

let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$

let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.
- *source* 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 $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $r = rank()$
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 *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 $\textit{context-flag}[k] = 1$ 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 $c = \textit{deposit-multi-coordinate}(g, k, \textit{axis-set}, \textit{multi-coordinate})$
    $\textit{dest}[k] \leftarrow \textit{source}[c]$

where *deposit-multi-coordinate* is as defined on page ??.

See section 5.16 on page 34 for a general description of multispread operations.

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 destination field.

source 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 $r = \mathit{rank}()$
    let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \wedge (\mathit{axis\text{-}mask}\langle m\rangle = 1)\,\}$
    let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \wedge \mathit{context\text{-}flag}[m] = 1\,\}$
    $$\mathit{dest}[k] \leftarrow \left(\bigwedge_{m \in C_k} \mathit{source}[m]\right)$$
where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 destination field.
- *source* — 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 $r = \textit{rank}()$
    let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$
    $$\textit{dest}[k] \leftarrow \left(\bigvee_{m \in C_k} \textit{source}[m] \right)$$
where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 destination field.
	source	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 $r = \textit{rank}()$
    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 *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 floating-point destination field.
- *source* 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 $r = \textit{rank}()$
  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 *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 signed integer destination field.
source	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 $r = \textit{rank}()$
  let $\textit{axis-set} = \{ m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \}$
  let $C_k = \{ m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \}$
  $\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m] \right)$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $r = rank()$
let $axis\text{-}set = \{\, m \mid 0 \le m < r \wedge (axis\text{-}mask\langle m\rangle = 1) \,\}$
let $C_k = \{\, m \mid m \in hyperplane(g, k, axis\text{-}set) \wedge context\text{-}flag[m] = 1 \,\}$
$dest[k] \leftarrow \left(\max_{m \in C_k} source[m]\right)$
where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 floating-point destination field.
source	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 $r = \mathit{rank}()$
let $\mathit{axis\text{-}set} = \{\, m \mid 0 \leq m < r \wedge (\mathit{axis\text{-}mask}\langle m \rangle = 1) \,\}$
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 *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 signed integer destination field.

source 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 $r = \textit{rank}()$
    let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$
    $\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m] \right)$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.
source	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 $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $r = rank()$
let $axis\text{-}set = \{ m \mid 0 \leq m < r \wedge (axis\text{-}mask\langle m \rangle = 1) \}$
let $C_k = \{ m \mid m \in hyperplane(g, k, axis\text{-}set) \wedge context\text{-}flag[m] = 1 \}$
$dest[k] \leftarrow \left(\min_{m \in C_k} source[m] \right)$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 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 33.

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 unsigned integer destination 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
    $dest[k] \leftarrow k$

This function stores into the *dest* field, within each selected processor, the send-address of that processor.

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 floating-point first source field.

source2 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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 signed integer first source field.

source2 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 *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 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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer first source field.

source2 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*] ≠ *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 not equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 floating-point destination field.

source 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*] ← −*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 signed integer destination field.
source	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 ⟨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 *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 unsigned integer destination field.

source — 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.

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 *context-flag*.

This function returns the next stack field id to be allocated.

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 *context-flag*.

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 floating-point destination field.

source1 The floating-point base field.

source2 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, or 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*] ≤ 0 then
        dest[*k*] ← 0
        test-flag[*k*] ← 1
      else
        dest[*k*] ← 0
        test-flag[*k*] ← 0
    else if *source1*[*k*] < 0 then

$dest[k] \leftarrow \langle\text{undefined}\rangle$
$test\text{-}flag[k] \leftarrow 1$
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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

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 floating-point destination field.

source1 The floating-point base field.

source2 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 $context\text{-}flag[k] = 1$ then
  if $source2[k] < 0$ then
    $lettemp1_k = 1.0/source1[k]$
    $lettemp2_k = -source2[k]$
  else
    $lettemp1_k = source1[k]$
    $lettemp2_k = source2[k]$
  if $temp2_k\langle 0\rangle = 0$ then
    $dest[k] \leftarrow 1.0$
  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 floating-point destination field.

source1 The floating-point base field.

source2 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 $context\text{-}flag[k] = 1$ then
    $lettemp_k = source1[k]$
    if $(slen2 = 0) \vee (source2[k]\langle 0\rangle = 0)$ then
      $dest[k] \leftarrow 1.0$
    else
      $dest[k] \leftarrow temp_k$
    for j from 1 to $slen2 - 1$ do
      if $source2[k]\langle j : slen2 - 1\rangle \neq 0$ then let $temp_k = temp_k \times temp_k$
      if $source2[k]\langle j\rangle$ then $dest[k] \leftarrow dest[k] \times temp_k$
    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.

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*

Operands

dest The signed integer destination field.

source1 The signed integer base field.

source2 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, 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-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.

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] ← 0
  else if *source2*[k] = 0 then
    dest[k] ← 1
  else
  dest[k] ← $(source1[k])^{source2[k]}$
  if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
  else *overflow-flag*[k] ← 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 are 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 *source1-value* or *source2-value* should be a signed integer front-end value. The operation is performed properly in all cases; the constant need not be representable in 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 *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 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-flag*$[k] = 1$ then

$$dest[k] \leftarrow \frac{\langle \text{pseudo-random choice of some } j,\ +0 \leq j < 2^{len} \rangle}{2^{len}}$$

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).

U-RANDOM

Stores a pseudo-randomly generated unsigned integer into the destination field.

Formats CM:u-random-1L *dest*, *len*, *limit*

Operands

dest 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] ← ⟨pseudo-random choice of some j, $0 \le j < limit$⟩

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 unsigned integer destination field.
source	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*.
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 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 $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] < \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k$
      :downward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] > \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k$
    $\textit{dest}[k] \leftarrow |L_k|$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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.

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. The smallest key has rank 0, the next smallest has rank 1, and so on; the largest key has rank $n - 1$ where n is the number of processors in the scan set. 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.)

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

Operands	*dest*	The unsigned integer destination field.
	source	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*.
	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 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 $L_k = \{\, m \mid m \in S_k \wedge ((source[m] < source[k]) \vee (source[m] = source[k$
      :downward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((source[m] > source[k]) \vee (source[m] = source[k$
      $dest[k] \leftarrow |L_k|$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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.

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. The smallest key has rank 0, the next smallest has rank 1, and so on; the largest key has rank $n-1$ where n is the number of processors in the scan set. 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.)

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 unsigned integer destination field.
source	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*.
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 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 $S_k = \textit{scan-set}(g, k, \textit{axis}, \textit{direction}, \textit{smode}, \textit{sbit})$
    case *direction* of
      :upward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] < \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k$
      :downward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] > \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k$
    $\textit{dest}[k] \leftarrow |L_k|$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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.

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. The smallest key has rank 0, the next smallest has rank 1, and so on; the largest key has rank $n - 1$ where n is the number of processors in the scan set. 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.)

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.

Formats CM:f-read-from-news-array-1L *front-end-array*, *offset-vector*, *start-vector*, *end-vector*, *axis-vector*, *source*, *s*, *e*; *rank*, *dimension-vector*, *element-len*

Operands *front-end-array* A front-end array (possibly multidimensional) of floating-point data.

offset-vector A front-end vector (one-dimensional array) of signed integer subscript offsets for the *front-end-array*.

start-vector A front-end vector (one-dimensional array) of unsigned integer inclusive lower bounds for NEWS indices.

end-vector A front-end vector (one-dimensional array) of unsigned integer exclusive upper bounds for NEWS indices.

axis-vector A front-end vector (one-dimensional array) of unsigned integer numbers indicating NEWS axes.

source 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 $s + e + 1$.

rank An unsigned integer, the rank (number of dimensions) of the *front-end-array*.

dimension-vector A front-end vector (one-dimensional array) of unsigned integer dimensions of the *front-end-array*.

element-len An unsigned integer, the size of an element *front-end-array*, measured in bytes. This must be 4 or 8.

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

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 $0 \le s_0 < (end_0 - start_0)$ 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
        $$\text{let } k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$
        $$\textit{front-end-array}_{offset_0+s_0, offset_1+s_1, \ldots, offset_{rank-1}+s_{rank-1}} \leftarrow source[k_{s_0,s_1,\ldots,s_{rank-1}}]$$

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*. When this operation is invoked from C code, the *element-len* parameter should be the number of bytes in an array element, as determined by the C sizeof operator.

The *source* parameter specifies the memory address within each processor of the field to be copied.

The five vector arguments are one-dimensional front-end arrays of length *rank*. For descriptive purposes let there be a number of indices k_j ($0 \le j < rank$) such that $0 \le k_j < (end_j - start_j)$. Then for all possible combinations of values for these indices, the data in the *source* field of the processor whose send address is

$$\bigvee_{j=0}^{n-1} \textit{make-news-coordinate}(start_j + k_j, axis_j)$$

is copied into the array element whose indices are $offset_j + k_j$ ($\le j < n$). The total number of values transferred is therefore

$$\prod_{j=0}^{n-1} (end_j - start_j)$$

The *dimension-vector* specifies the dimensions of the front end array.

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.

Formats CM:s-read-from-news-array-1L *front-end-array*, *offset-vector*, *start-vector*, *end-vector*, *axis-vector*, *source*, *len*; *rank*, *dimension-vector*, *element-len*

Operands

front-end-array A front-end array (possibly multidimensional) of signed integer data.

offset-vector A front-end vector (one-dimensional array) of signed integer subscript offsets for the *front-end-array*.

start-vector A front-end vector (one-dimensional array) of unsigned integer inclusive lower bounds for NEWS indices.

end-vector A front-end vector (one-dimensional array) of unsigned integer exclusive upper bounds for NEWS indices.

axis-vector A front-end vector (one-dimensional array) of unsigned integer numbers indicating NEWS axes.

source 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*.

rank An unsigned integer, the rank (number of dimensions) of the *front-end-array*.

dimension-vector A front-end vector (one-dimensional array) of unsigned integer dimensions of the *front-end-array*.

element-len An unsigned integer, the size of an element *front-end-array*, measured in bytes. This must be 1, 2, or 4.

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

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 $0 \leq s_0 < (end_0 - start_0)$ 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

$$\text{let } k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0+s_0, offset_1+s_1, \ldots, offset_{rank-1}+s_{rank-1}} \leftarrow source[k_{s_0,s_1,\ldots,s_{rank-1}}]$$

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*. When calling Paris from Lisp the array may be a general S-expression array containing signed integers, or may be a specialized integer-element array (such as the kind called **art-8b** on the Symbolics 3600). When this operation is invoked from C code, the *element-len* parameter should be the number of bytes in an array element, as determined by the C **sizeof** operator.

The *source* parameter specifies the memory address within each processor of the field to be copied.

The five vector arguments are one-dimensional front-end arrays of length *rank*. For descriptive purposes let there be a number of indices k_j $(0 \leq j < rank)$ such that $0 \leq k_j < (end_j - start_j)$. Then for all possible combinations of values for these indices, the data in the *source* field of the processor whose send address is

$$\bigvee_{j=0}^{n-1} \textit{make-news-coordinate}(start_j + k_j, axis_j)$$

is copied into the array element whose indices are $offset_j + k_j$ $(\leq j < n)$. The total number of values transferred is therefore

$$\prod_{j=0}^{n-1} (end_j - start_j)$$

The *dimension-vector* specifies the dimensions of the front end array.

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.

Formats CM:u-read-from-news-array-1L *front-end-array*, *offset-vector*, *start-vector*, *end-vector*, *axis-vector*, *source*, *len*; *rank*, *dimension-vector*, *element-len*

Operands

front-end-array A front-end array (possibly multidimensional) of floating-point data.

offset-vector A front-end vector (one-dimensional array) of signed integer subscript offsets for the *front-end-array*.

start-vector A front-end vector (one-dimensional array) of unsigned integer inclusive lower bounds for NEWS indices.

end-vector A front-end vector (one-dimensional array) of unsigned integer exclusive upper bounds for NEWS indices.

axis-vector A front-end vector (one-dimensional array) of unsigned integer numbers indicating NEWS axes.

len 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 *source* field. The total length of an operand in this format is $s + e + 1$.

rank An unsigned integer, the rank (number of dimensions) of the *front-end-array*.

dimension-vector A front-end vector (one-dimensional array) of unsigned integer dimensions of the *front-end-array*.

element-len An unsigned integer, the size of an element *front-end-array*, measured in bytes. This must be 1, 2, or 4.

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

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

$$\text{let } k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0+s_0, offset_1+s_1,\ldots,offset_{rank-1}+s_{rank-1}} \leftarrow source[k_{s_0,s_1,\ldots,s_{rank-1}}]$$

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.* When this operation is invoked from C code, the *element-len* parameter should be the number of bytes in an array element, as determined by the C sizeof operator.

The *source* parameter specifies the memory address within each processor of the field to be copied.

The five vector arguments are one-dimensional front-end arrays of length *rank.* For descriptive purposes let there be a number of indices k_j $(0 \le j < rank)$ such that $0 \le k_j < (end_j - start_j)$. Then for all possible combinations of values for these indices, the data in the *source* field of the processor whose send address is

$$\bigvee_{j=0}^{n-1} \textit{make-news-coordinate}(start_j + k_j, axis_j)$$

is copied into the array element whose indices are $offset_j + k_j$ $(\le j < n)$. The total number of values transferred is therefore

$$\prod_{j=0}^{n-1} (end_j - start_j)$$

The *dimension-vector* specifies the dimensions of the front end array.

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 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 *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 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*.
Context	This operation is unconditional. It does not depend on *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 unsigned integer source field.

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

Context This operation is unconditional. It does not depend on *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.

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 floating-point destination field.

source 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 36.

See section 5.16 on page 34 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 signed integer destination field.

source 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

$$dest[k] \leftarrow \left(\sum_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 33.

See section 5.16 on page 34 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 destination field.

source 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 $\mathit{context\text{-}flag}[k] = 1$ 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

$$\mathit{dest}[k] \leftarrow \left(\bigwedge_{m \in C_k} \mathit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 destination field.

source 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

$$\textit{dest}[k] \leftarrow \left(\bigvee_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 destination field.

source 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 36.

See section 5.16 on page 34 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 floating-point destination field.

source 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 $g = geometry(current\text{-}vp\text{-}set)$
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 36.

See section 5.16 on page 34 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 signed integer destination field.
- *source* — 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 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 36.

See section 5.16 on page 34 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 floating-point destination field.

source 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 $\mathit{context\text{-}flag}[k] = 1$ 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 36.

See section 5.16 on page 34 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 signed integer destination field.

source 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-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

$$dest[k] \leftarrow \left(\min_{m \in C_k} source[m]\right)$$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 36.

See section 5.16 on page 34 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 floating-point destination field. This is the quotient.

source1 The floating-point first source field. This is the dividend.

source2 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 unaffecteddivision 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*$[k] \neq 0$ then
      let $v =$ *source1*$[k]/$*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
  $dest[k] \leftarrow \langle\text{unpredictable}\rangle$
  $test\text{-}flag[k] \leftarrow 1$
if $\langle$overflow occurred in processor $k\rangle$ 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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

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 signed integer remainder field.

source1 The signed integer dividend field.

source2 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 *overflow-flag* is set if either the result cannot be represented in the destination field or the 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

$$\textit{dest}[k] \leftarrow \textit{sign}(\textit{source1}[k]) \times \left(|\textit{source1}[k]| - |\textit{source2}[k]| \times \left\lfloor \frac{|\textit{source1}[k]|}{|\textit{source2}[k]|} \right\rfloor\right)$$

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

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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer remainder field.

source1 — The unsigned integer dividend field.

source2 — 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 — *overflow-flag* is set if the 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 $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$
    else $\textit{overflow-flag}[k] \leftarrow 0$

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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

RESET-TIMER

For the C/Paris and Fortran/Paris interfaces, resets the timing facility before timing other operations.

Formats CM:reset-timer

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

The function CM:reset-timer is used in the C/Paris and Fortran/Paris interfaces to reset the facility for timing the execution of other operations on the Connection Machine system.

One should first call CM:reset-timer to clear the timing counters. Subsequently one may alternately call CM:start-timer and CM:stop-timer. The amounts of real time and run time between a start and a stop are accumulated into the counters. One may start and stop the clocks repeatedly. Every time CM:stop-timer is called, it returns a structure of type CM_timeval_t that contains time accumulated between all start/stop call pairs since the last call to CM:reset-timer.

The timing facility is provided in the Lisp/Paris interfaces through the CM:time macro.

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 floating-point destination field.
source	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 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 37.

See section 5.16 on page 34 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 signed integer destination field.
source	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 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-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 37.

See section 5.16 on page 34 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 unsigned integer destination field.
source	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 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 $S_k = \mathit{scan\text{-}subset}(g, k, \mathit{axis}, \mathit{direction}, \mathit{inclusion}, \mathit{smode}, \mathit{sbit})$
    if $|S_k| = 0$ then
      $\mathit{dest}[k] \leftarrow 0$
    else

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in S_k} \mathit{source}[m] \right)$$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 destination field.
source	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 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 $context\text{-}flag[k] = 1$ 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
      $dest[k] \leftarrow 000\ldots000$
    else
      case *direction* of
        :upward : let $m' = \min_{m \in S_k} m$
        :downward : let $m' = \max_{m \in S_k} m$
      $dest[k] \leftarrow source[m']$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 destination field.
source	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 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
      $\mathit{dest}[k] \leftarrow 111\ldots111$
    else

$$\mathit{dest}[k] \leftarrow \left(\bigwedge_{m \in S_k} \mathit{source}[m] \right)$$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 destination field.

source — 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 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 *context-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
      $dest[k] \leftarrow 000\ldots000$
    else

$$dest[k] \leftarrow \left(\bigvee_{m \in S_k} source[m] \right)$$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 destination field.
source	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 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 $context\text{-}flag[k] = 1$ 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
      $dest[k] \leftarrow 000\ldots000$
    else

$$dest[k] \leftarrow \left(\bigoplus_{m \in S_k} source[m] \right)$$

where *scan-subset* is as defined on page 37.

SCAN-WITH-LOGXOR

See section 5.16 on page 34 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 floating-point destination field.
source	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 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
      $dest[k] \leftarrow -\infty$
    else
      $$dest[k] \leftarrow \left(\max_{m \in S_k} source[m]\right)$$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 signed integer destination field.

source 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 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-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 37.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 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
      $$dest[k] \leftarrow \left(\max_{m \in S_k} source[m]\right)$$
where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 floating-point destination field.
source	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 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
      $dest[k] \leftarrow +\infty$
    else
      $$dest[k] \leftarrow \left(\min_{m \in S_k} source[m]\right)$$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 signed integer destination field.

source 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 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 37.

See section 5.16 on page 34 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 unsigned integer destination field.
source	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 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
    $$dest[k] \leftarrow \left(\min_{m \in S_k} source[m]\right)$$

where *scan-subset* is as defined on page 37.

See section 5.16 on page 34 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 floating-point destination field.
source	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 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 37.

See section 5.16 on page 34 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 destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 $\textit{notify}[k] \leftarrow 1$

$dest[k] \leftarrow \langle\text{undefined}\rangle$

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 destination array field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

index The unsigned integer index into the array field.

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*.

Overlap The *send-address* and *source* may overlap in any manner. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
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 $0 \le j < dlen$ 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 $0 \le j < dlen$ do
  $array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle \leftarrow sum_k\langle j\rangle$
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 destination array field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

index The unsigned integer index into the array field.

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*.

Overlap The *send-address* and *source* may overlap in any manner. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
  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 $0 \le j < dlen$ 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 destined 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 destination array field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

index The unsigned integer index into the array field.

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*.

Overlap The *send-address* and *source* may overlap in any manner. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors regardless of the value of the *context-flag*.

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

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

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 destination field.
- *source* — 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 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 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 $\textit{context-flag}[k] = 1$) then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    $\textit{dest}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})] \leftarrow \textit{source}[k]$

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-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 floating-point destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source 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 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 signed integer destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The signed integer source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 unsigned integer destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The unsigned integer source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 $\textit{notify}[k] \leftarrow 1$
  $$\textit{dest}[k] \leftarrow \textit{dest}[k] \land \left(\bigwedge_{m \in S_k} \textit{source}[m] \right)$$

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 destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] \lor \left(\bigvee_{m \in S_k} \textit{source}[m] \right)$$

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-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 destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 floating-point destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source 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 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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-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 area should first be set to $-\infty$.

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 signed integer destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The signed integer source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 area should first be set to -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 unsigned integer destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The unsigned integer source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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-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 area should first be set to $2^{len} - 1$.

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 floating-point destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source 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 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 maximum operation. To receive the maximum of only the messages, the destination area should first be set to $+\infty$.

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 signed integer destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The signed integer source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 maximum operation. To receive the maximum of only the messages, the destination area should first be set to $2^{len-1} - 1$.

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 unsigned integer destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The unsigned integer source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 maximum operation. To receive the minimum of only the messages, the destination area should first be set to zero.

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 destination field.

send-address The field containing a send-address that indicates which processor is to receive the message.

source The source field.

len The length of the *dest* and *source* fields.

notify 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. The *dest* field may overlap with *send-address* or *source*, but if it does, then it is forbidden to send a message to a selected processor. In other words, the *dest* may overlap with *send-address* or *source* only if within each processor at most one of them will be used.

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 all processors 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 $\textit{notify}[k] \leftarrow 1$
$\textit{dest}[k] \leftarrow \textit{source}[\textit{choice}(S_k)]$

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-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-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 *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 *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 *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 *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 $\textit{context-flag}[k] = 1$ then
$\textit{flag}[k] \leftarrow 1$

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

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

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 floating-point destination field.

source 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*$[k] < 0$ 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 signed integer destination field.
source	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-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 signed integer destination field.

source 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 $context\text{-}flag[k] = 1$ 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, zero, or positive, respectively.

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 floating-point destination field.

source 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
$dest[k] \leftarrow \sin source[k]$

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

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 floating-point destination field.

source 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*] ← sinh *source*[*k*]
if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

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

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 floating-point destination field.
	source	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 36.

See section 5.16 on page 34 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 signed integer destination field.
source	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 $C_k = scan\text{-}subclass(g, k, axis)$
    $$dest[k] \leftarrow \left(\sum_{m \in C_k} source[m] \right)$$
where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 unsigned integer destination field.
source	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-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 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $c = \mathit{deposit\text{-}news\text{-}coordinate}(g, k, \mathit{axis}, \mathit{coordinate})$
    $\mathit{dest}[k] \leftarrow \mathit{source}[c]$

where *deposit-news-coordinate* is as defined on page 33.

See section 5.16 on page 34 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 destination field.
- *source* 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 36.

See section 5.16 on page 34 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 destination field.
source	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 $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\bigvee_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 destination field.
- *source* — 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 36.

See section 5.16 on page 34 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 floating-point destination field.

source 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 $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $C_k = \mathit{scan\text{-}subclass}(g, k, \mathit{axis})$
    $$\mathit{dest}[k] \leftarrow \left(\max_{m \in C_k} \mathit{source}[m]\right)$$
where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 signed integer destination field.

source 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 36.

See section 5.16 on page 34 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 unsigned integer destination field.

source 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 $\textit{context-flag}[k] = 1$ 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 36.

See section 5.16 on page 34 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 floating-point destination field.
	source	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 $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $C_k = \mathit{scan\text{-}subclass}(g, k, \mathit{axis})$
    $\mathit{dest}[k] \leftarrow \left(\min\limits_{m \in C_k} \mathit{source}[m] \right)$
where *scan-subclass* is as defined on page 36.

See section 5.16 on page 34 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 signed integer destination field.
	source	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 $context\text{-}flag[k] = 1$ 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 36.

See section 5.16 on page 34 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 unsigned integer destination field.
source	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 $\textit{context-flag}[k] = 1$ 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 36.

See section 5.16 on page 34 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.

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 floating-point destination field.

source 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 *context-flag*$[k] = 1$ then
    if *source*$[k] > 0$ then
      dest$[k] \leftarrow \sqrt{source[k]}$
    else if *source*$[k] = \pm 0$ then
      dest$[k] \leftarrow$ *source*$[k]$
    else if : *source* : $[k] < 0$ then
      dest$[k] \leftarrow \langle$unpredictable$\rangle$
      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.

START-TIMER

For the C/Paris and Fortran/Paris interfaces, starts the timer.

Formats CM:start-timer

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

The function CM:start-timer is used in the C/Paris and Fortran/Paris interfaces as part of the facility for timing the execution of other operations on the Connection Machine system. This function starts the accumulation of measured real time and run time.

One should first call CM:reset-timer to clear the timing counters. Subsequently one may alternately call CM:start-timer and CM:stop-timer. The amounts of real time and run time between a start and a stop are accumulated into the counters. One may start and stop the clocks repeatedly. Every time CM:stop-timer is called, it returns a structure of type CM_timeval_t that contains time accumulated between all start/stop call pairs since the last call to CM:reset-timer.

The timing facility is provided in the Lisp/Paris interfaces through the CM:time macro.

STOP-TIMER

For the C/Paris and Fortran/Paris interfaces, stops the timer.

Formats CM:stop-timer

Result — The accumulated timings since the last call to CM:reset-timer. In the C/Paris interface, this is a structure of type CM_timeval_t. In the Fortran/Paris interface, this is a DOUBLE PRECISION array of length 2.

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

The function CM:stop-timer is used in the C/Paris and Fortran/Paris interfaces as part of the facility for timing the execution of other operations on the Connection Machine system. This function stops the accumulation of measured real time and run time.

One should call CM:reset-timer to clear the timing counters. Subsequently one may alternately call CM:start-timer and CM:stop-timer. The amounts of real time and run time between a start and a stop are accumulated into the counters. One may start and stop the clocks repeatedly. Every time CM:stop-timer is called, it returns a structure of type CM_timeval_t that contains time accumulated between all start/stop call pairs since the last call to CM:reset-timer.

The timing facility is provided in the Lisp/Paris interfaces through the CM:time macro.

STORE-CONTEXT

Unconditionally stores the context bit into memory.

Formats CM:store-context *dest*

Operands *dest* 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-overflow *dest*

Operands *dest* The destination bit (a one-bit 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
    $\textit{dest}[k] \leftarrow \textit{flag}[k]$

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

Within each processor, the indicated flag for that processor is stored into memory.

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

Operands

dest The floating-point destination field.

source1 The floating-point first source (minuend) field.

source2 The floating-point second source (subtrahend) field.

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

source3 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

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*] ← (*source1*[*k*] − *source2*[*k*]) × *source3*[*k*]
if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 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 into memory. The various operand formats allow 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). The constant is then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

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.

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 floating-point destination field. This is the difference, the result of the subtraction operation.

source1 The floating-point first source field. This is the minuend.

source2 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 $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow source1[k] - source2[k]$
if $\langle$overflow occurred in processor $k\rangle$ then $overflow\text{-}flag[k] \leftarrow 1$

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 are 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). The constant is then converted, in effect, to the format specified by s and e before the operation is performed.

S-SUBTRACT

The difference of two signed integer source values is placed in the destination field. Carry-out and overflow are 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 signed integer destination field. This is the difference, the result of the subtraction operation.

source1 The signed integer first source field. This is the minuend.

source2 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 is a carry-out from the high-order bit position; otherwise it is cleared. For subtraction, "carry" is equivalent to "not borrow."

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

The operand *source2* is subtracted from *source1*, treated as 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 "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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

U-SUBTRACT

The difference of two unsigned integer source values is placed in the destination field. Carry-out and overflow are 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 unsigned integer destination field. This is the difference, the result of the subtraction operation.

source1 The unsigned integer first source field. This is the minuend.

source2 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 a carry-out from the high-order bit position; otherwise it is cleared. For subtraction, "carry" is equivalent to "not borrow."

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]
carry-flag[k] ← ⟨carry out in processor k⟩
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. 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 "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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

SWAP

Swaps the contents of two bit fields.

Formats CM:swap-2-1L *dest1/source1*, *dest2/source2*, *len*

Operands

dest1 The first destination field.

source1 The first source (same as first destination) field.

dest2 The second destination field.

source2 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 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 $temp1_k = source1[k]$
    let $temp2_k = source2[k]$
    let $dest1[k] \leftarrow temp2_k$
    let $dest2[k] \leftarrow temp1_k$

Each of the two fields is copied into the other so as to exchange their contents.

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 floating-point destination field.

source 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*] ← tan *source*[*k*]
if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

The 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 floating-point destination field.

source 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 ⟨overflow occurred in processor k⟩ 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.

Formats CM:time *expressions*

Context This operation is unconditional. It does not depend on *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 contains a 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.

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.

The timing facility is provided in the C/Paris and Fortran/Paris interfaces through a set of functions CM:reset-timer, CM:start-timer, and CM:stop-timer.

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 unconditional. It does not depend on *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 destination field.

source 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.

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 floating-point destination field.

source 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
$dest[k] \leftarrow sign(source) \times \lfloor |source[k]| \rfloor$

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 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 signed integer destination field.
	source	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
    $\mathit{dest}[k] \leftarrow \mathit{sign}(\mathit{source}) \times \lfloor|\mathit{source}[k]|\rfloor$
  if ⟨overflow occurred in processor k⟩ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$ else $\mathit{overflow\text{-}flag}[k]$

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 signed integer quotient field.

source1 The signed integer dividend field.

source2 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 either the quotient cannot be represented in the destination field or the 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
    $\textit{dest}[k] \leftarrow \textit{sign}(\textit{source1}[k]) \times \textit{sign}(\textit{source2}[k]) \times \left\lfloor \frac{|\textit{source1}[k]|}{|\textit{source2}[k]|} \right\rfloor$
  if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$
  else $\textit{overflow-flag}[k] \leftarrow 0$

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. The operation is performed properly in all cases; the constant need not be representable in 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 unsigned integer quotient field.

source1 The unsigned integer dividend field.

source2 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 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-flag*$[k] = 1$ then

if *source2*$[k] = 0$ then
  $dest[k] \leftarrow$ ⟨unpredictable⟩
else
  $$dest[k] \leftarrow \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$

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. The operation is performed properly in all cases; the constant need not be representable in the number of bits specified by *len*.

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 *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 *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.

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.

Formats CM:f-write-to-news-array-1L *front-end-array*, *offset-vector*, *start-vector*, *end-vector*, *axis-vector*, *dest*, *s*, *e*; *rank*, *dimension-vector*, *element-len*

Operands

front-end-array A front-end array (possibly multidimensional) of floating-point data.

offset-vector A front-end vector (one-dimensional array) of floating-point subscript offsets for the *front-end-array*.

start-vector A front-end vector (one-dimensional array) of unsigned integer inclusive lower bounds for NEWS indices.

end-vector A front-end vector (one-dimensional array) of unsigned integer exclusive upper bounds for NEWS indices.

axis-vector A front-end vector (one-dimensional array) of unsigned integer numbers indicating NEWS axes.

dest 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 $s + e + 1$.

rank An unsigned integer, the rank (number of dimensions) of the *front-end-array*.

dimension-vector A front-end vector (one-dimensional array) of unsigned integer dimensions of the *front-end-array*.

element-len An unsigned integer, the size of an element *front-end-array*, measured in bytes. This must be 4 or 8.

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

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

$$\text{let } s_{\langle i,m\rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

let $k_i = \bigvee_{j=0}^{rank-1} \mathit{make\text{-}news\text{-}coordinate}(axis_j, start_j + s_{i,j})$
$dest[k_i] \leftarrow \mathit{front\text{-}end\text{-}array}_{s_{\langle i,0\rangle}, s_{\langle i,1\rangle}, \ldots, s_{\langle i,rank-1\rangle}}$

Another formulation:

For all s_0 such that $0 \le s_0 < (end_0 - start_0)$ 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
let $k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \mathit{make\text{-}news\text{-}coordinate}(axis_j, start_j + s_j)$
$dest[k_{s_0,s_1,\ldots,s_{rank-1}}] \leftarrow$
$\mathit{front\text{-}end\text{-}array}_{offset_0+s_0, offset_1+s_1, \ldots, offset_{rank-1}+s_{rank-1}}$

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. When this operation is invoked from C code, the *element-len* parameter should be the number of bytes in an array element, as determined by the C sizeof operator.

The *dest* parameter specifies the memory address within each processor of the field into which the data is to be stored.

The five vector arguments are one-dimensional front-end arrays of length *rank*. For descriptive purposes let there be a number of indices k_j $(0 \le j < rank)$ such that $0 \le k_j < (end_j - start_j)$. Then for all possible combinations of values for these indices, the array element whose indices are $offset_j + k_j$ $(\le j < n)$ is copied into the *dest* field of the processor whose send address is

$$\bigvee_{j=0}^{n-1} \mathit{make\text{-}news\text{-}coordinate}(start_j + k_j, axis_j)$$

The total number of values transferred is therefore

$$\prod_{j=0}^{n-1} (end_j - start_j)$$

The *dimension-vector* specifies the dimensions of the front end array.

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.

Formats CM:s-write-to-news-array-1L *front-end-array*, *offset-vector*, *start-vector*, *end-vector*, *axis-vector*, *dest*, *len*; *rank*, *dimension-vector*, *element-len*

Operands *front-end-array* A front-end array (possibly multidimensional) of signed integer data.

offset-vector A front-end vector (one-dimensional array) of signed integer subscript offsets for the *front-end-array*.

start-vector A front-end vector (one-dimensional array) of unsigned integer inclusive lower bounds for NEWS indices.

end-vector A front-end vector (one-dimensional array) of unsigned integer exclusive upper bounds for NEWS indices.

axis-vector A front-end vector (one-dimensional array) of unsigned integer numbers indicating NEWS axes.

dest 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*.

rank An unsigned integer, the rank (number of dimensions) of the *front-end-array*.

dimension-vector A front-end vector (one-dimensional array) of unsigned integer dimensions of the *front-end-array*.

element-len An unsigned integer, the size of an element *front-end-array*, measured in bytes. This must be 1, 2, or 4.

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

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 $0 \le s_0 < (end_0 - start_0)$ 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
          let $k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$
          $dest[k_{s_0,s_1,\ldots,s_{rank-1}}] \leftarrow$
            $\textit{front-end-array}_{offset_0+s_0, offset_1+s_1, \ldots, offset_{rank-1}+s_{rank-1}}$

This operation copies a rectangular subblock of an array in 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. When calling Paris from Lisp the array may be a general S-expression array containing signed integers, or may be a specialized integer-element array (such as the kind called art-8b on the Symbolics 3600). When this operation is invoked from C code, the *element-len* parameter should be the number of bytes in an array element, as determined by the C sizeof operator.

The *dest* parameter specifies the memory address within each processor of the field into which the data is to be stored.

The five vector arguments are one-dimensional front-end arrays of length *rank*. For descriptive purposes let there be a number of indices k_j $(0 \le j < rank)$ such that $0 \le k_j < (end_j - start_j)$. Then for all possible combinations of values for these indices, the array element whose indices are $offset_j + k_j$ $(\le j < n)$ is copied into the *dest* field of the processor whose send address is

$$\bigvee_{j=0}^{n-1} \textit{make-news-coordinate}(start_j + k_j, axis_j)$$

The total number of values transferred is therefore

$$\prod_{j=0}^{n-1} (end_j - start_j)$$

The *dimension-vector* specifies the dimensions of the front end array.

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.

Formats CM:u-write-to-news-array-1L *front-end-array*, *offset-vector*, *start-vector*, *end-vector*, *axis-vector*, *dest*, *len*; *rank*, *dimension-vector*, *element-len*

Operands

front-end-array A front-end array (possibly multidimensional) of unsigned integer data.

offset-vector A front-end vector (one-dimensional array) of signed integer subscript offsets for the *front-end-array*.

start-vector A front-end vector (one-dimensional array) of unsigned integer inclusive lower bounds for NEWS indices.

end-vector A front-end vector (one-dimensional array) of unsigned integer exclusive upper bounds for NEWS indices.

axis-vector A front-end vector (one-dimensional array) of unsigned integer numbers indicating NEWS axes.

dest 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*.

rank An unsigned integer, the rank (number of dimensions) of the *front-end-array*.

dimension-vector A front-end vector (one-dimensional array) of unsigned integer dimensions of the *front-end-array*.

element-len An unsigned integer, the size of an element *front-end-array*, measured in bytes. This must be 1, 2, or 4.

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

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

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

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
  let $k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$
  $dest[k_{s_0,s_1,\ldots,s_{rank-1}}] \leftarrow$
    $\textit{front-end-array}_{offset_0+s_0, offset_1+s_1, \ldots, offset_{rank-1}+s_{rank-1}}$

This operation copies a rectangular subblock of an array in 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. When calling Paris from Lisp the array may be a general S-expression array containing unsigned integers, or may be a specialized integer-element array (such as the kind called art-8b on the Symbolics 3600). When this operation is invoked from C code, the *element-len* parameter should be the number of bytes in an array element, as determined by the C sizeof operator.

The *dest* parameter specifies the memory address within each processor of the field into which the data is to be stored.

The five vector arguments are one-dimensional front-end arrays of length *rank*. For descriptive purposes let there be a number of indices k_j $(0 \leq j < rank)$ such that $0 \leq k_j < (end_j - start_j)$. Then for all possible combinations of values for these indices, the array element whose indices are $offset_j + k_j$ $(\leq j < n)$ is copied into the *dest* field of the processor whose send address is

$$\bigvee_{j=0}^{n-1} \textit{make-news-coordinate}(start_j + k_j, axis_j)$$

The total number of values transferred is therefore

$$\prod_{j=0}^{n-1} (end_j - start_j)$$

The *dimension-vector* specifies the dimensions of the front end array.

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 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 *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 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 *context-flag*.

Definition *dest*[*send-address-value*] ← *source-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 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 *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*.

Appendix A

Changes from Version 4.3 Paris

The Paris instruction set released with Connection Machine System Software Version 5.0 is substantially different from that released with Version 4.3. Nearly all of the previous functionality has been retained, sometimes in slightly altered form. A few operations have been removed, and several categories of new operations have been added. The changes from Version 4.3 to Version 5.0 are summarized here.

First a word about release and version numbers. In the past, Paris releases have born version numbers different from those used for the Connection Machine System Software as a whole. Beginning with Version 5.0, Paris release numbers now correspond to those of the rest of the system software. Thus, when we refer to Paris Version 4.3, we are referring to Paris software that was originally called Paris Release 2, version 7 and that was distributed with Version 4.3 of the system software. The current Paris software is Version 5.0. In conversation, Paris Version 5.0 may sometimes be referred to as Paris Release 3. This should happen less as the new numbering system takes hold.

A.1 All names are alphabetic and limited in length

To allow convenient use of Paris within many programming languages and environments, nearly all Paris instructions have been renamed. Names do not contain any special characters except for colon and hyphen (for the Lisp interface) or underscore (for the C and Fortran interface). For example, what used to be called `cm:+` in the Lisp interface is now called `CM:s-add-2-1L` in the Lisp interface and `CM_s_add_2_1L` in the C and Fortran interface.

All names have been limited to thirty characters.

The new names have been chosen so as to allow the old and the new names to coexist.

A.2 New capitalization conventions accommodate C and Fortran

The prefix "CM" is consistently capitalized, as is the trailing "L" in length specifiers; all other letters are lower case.

Capitalization matters in the C interface. It does not matter in the Lisp or Fortran interface, but for expository consistency this document follows the capitalization conventions even in presenting Lisp code.

A.3 Optional arguments have been eliminated

The Lisp language supports optional arguments, but C and Fortran do not. For the sake of uniformity, operands that were optional in the Lisp interface for Paris Version 4.3 have been made required arguments in the equivalent Paris Version 5.0 instructions.

A.4 New naming conventions reflect new orthogonal attributes

Many binary arithmetic operations now come in three-operand and two-operand forms, according to whether the destination address is explicitly specified separately or is implicitly the same as one source address. Similarly, many unary arithmetic operations now come in two-operand and one-operand forms. In many cases using the form with fewer operands provides a performance advantage; in a few cases the shorter form is provided merely for the sake of symmetry in the instruction set.

Most Paris operations require one or more *length arguments*, indicating the lengths of various memory fields. In most cases the new operation names carry an explicit indication of the number of length operands; this indication always comes last in the same, and consists of the number followed by a capital "L". Sometimes two operations are identical except for the number of length operands; for example, CM:s-add-3-3L takes three separate length specifiers, one for each memory field, whereas CM:s-add-3-1L takes a single length operand specifying the common length of the three memory fields.

Single letters are used to indicate the type of data to be operated upon:

s	signed (two's-complement) integer
u	unsigned integer
f	floating-point

A.5 More instructions have -constant forms

In Version 4.3 not all arithmetic operations had "-constant" forms. In Paris Version 5.0 many more arithmetic operations have "-constant" forms, making the instruction set much more symmetrical.

A.6 Different instructions have -always forms

A number of instructions with the word "always" in their names have been removed; one example is CM:logand-always. Such unconditional instructions were intended primarily for manipulating flag bits, expecially the context bit. They have been replaced by a series of special instructions for operating on the flags (see below).

On the other hand, other unconditional operations have been introduced, especially for floating-point arithmetic. When floating-point hardware is in use, unconditional operations may be significantly faster than the correspondiong conditional versions.

A.7 Special instructions operate on the context and flag bits

Paris Version 5.0 more distinctly separates the flag bits from ordinary memory operands. In Paris Version 4.3, for example, it was possible to use the instruction CM:logand on either a memory operand or a flag. In Version 5.0, CM:logand-2L and related instructions may operate only on memory operands; separate instructions such as CM:logand-overflow are provided to operate on the flags.

All instructions that operate on the context flag are unconditional (despite not having "-always" in their names). All instructions that operate on other flag bits are conditional, except for CM:clear-all-flags-always.

A.8 Irrational and transcendental functions are supported

Ordinary and hyperbolic sine, cosine, and tangent functions are provided, as well as their inverses. The square root, exponential, power, and natural logarithm operations are also provided.

A.9 New arithmetic operations have been added

New operations include integer exponentiation and generation of pseudo-random floating-point numbers.

A.10 Two-result operations have been eliminated

The following operations have been eliminated from the Paris instruction set in Version 5.0:

CM:floor-and-mod
CM:ceiling-and-remainder
CM:truncate-and-rem
CM:round-and-remainder
CM:unsigned-floor-and-mod
CM:unsigned-ceiling-and-remainder
CM:unsigned-truncate-and-rem
CM:unsigned-round-and-remainder

A.11 Special compound floating-point operations improve performance

Special instructions have been added to compute expressions of the form $xy \pm z$ and $(x \pm y)z$, where y and z may each be a memory operand or a constant. Each such instruction is functionally equivalent to a two-instruction sequence containing a multiply and an add instruction, but provides improved performance.

A.12 SEND operations no longer accept a time limit

The optional time-limit operand has been removed from the send operations.

A.13 Cube addresses are now called send addresses

To emphasize that cube addresses really have little to do with the hypercube structure of the router, but rather function primarily as addresses used by the send instructions, the terminology *send address* has been introduced to replace the term *cube address.*

A.14 Generalized NEWS operations support multidimensional grids

Paris Version 4.3 supported a special two-dimensional communication structure called the NEWS grid. Paris Version 5.0 generalizes this (using special hardware on the Connection Machine Model 2) to any number of dimensions, including one dimension. The operations CM:get-from-north, CM:get-from-east, CM:get-from-west, and CM:get-from-south are replaced by a single operation CM:get-from-news. The direction in which to communicate is specified by an additional *axis* operand.

A.15 SCAN and GLOBAL now permit new combining functions

Paris Version 4.3 supported scan operations only for the combining operations max (signed, unsigned, and floating-point) and plus (signed and unsigned). A somewhat larger class of global operations were supported as well.

Paris Version 5.0 rounds out this set of operations to support all the combining operations that may be used with the send operations: add, max, and min for signed, unsigned, and floating-point types; logand, logior, and logxor for bit strings.

In addition, there are the scan-with-copy and scan-with-f-multiply operations. The optimized operations global-u-max-s-intlen and global-u-max-u-intlen are equivalent to an appropriate integer-length operation followed by a global-u-max, but are faster.

An entire set of new operations has been introduced to perform global computations on flags.

A.16 SCAN operations may be applied along NEWS dimensions

In Paris Version 5.0, all scan operations may be applied along a NEWS dimension; the effect is to do a separate scan operation on every row (or column, or whatever) of an array.

A.17 SCAN operations may be partitioned

In Paris Version 5.0, all scan operations allow the set of processors to be partitioned into groups of varying size as indicated by a *segment bit* or *start bit* (these two kinds of indicators have slightly different effects).

A.18 SPREAD operations replicate data efficiently

The new spread operations perform efficient replication of a subplane of a multidimensional array to fill the entire array. The operation spread-with-copy can take any one column of a matrix, for example, and copy it into every column. Other versions of spread involve combining operations. For example, spread-with-f-add can replace every element of a matrix with the sum of all elements in the same row; this is equivalent to performing a scan-with-f-add to form the row sums in the last column, followed by a spread-with-copy (or a downward scan-with-copy) to copy that column back into the other columns, but is faster. For every scan instruction there is a corresponding spread instruction.

The multispread variants allow spreading along several NEWS axes at once.

A.19 REDUCE operations perform reductions efficiently

If the result of a spread operation need not be replicated, but instead may usefully be placed into just one processor of the row or column, then a reduce operation is just the ticket.

A.20 New array instructions allow faster indexing

The new operations aref32 and aset32 allow an array to be stored within each virtual processor and accessed in much the same way as for aref and aset. The new operations store the data in a different manner ("slicewise"); data in such arrays should be accessed *only* through these special instructions or their equivalent. The advantage is that data stored in this special format can be stored and retrieved much more quickly.

Further variants aref32-shared and aset32-shared allow not only fast access but also memory savings by letting many virtual processors share the same array.

A.21 STORE operations have been eliminated

The store operations, which were complex variants of send, have been eliminated. However, new operations that are a compound of of send and aset32 have been introduced; while these do not provide quite as large a variety of combining operations, they are significantly faster.

Similarly, fetch has been replaced by get-aref32.

A.22 Most version 4.3 operations have version 5.0 equivalents

The following table lists all Paris Version 4.3 operations in alphabetical (or rather, ASCII) order, and gives the nearest equivalent in Version 5.0. In most cases the interface and functionality are identical except that the operation has a new name.

cm:*	CM:s-multiply-2-1L
cm:+	CM:s-add-2-1L
cm:+carry	CM:s-add-carry-2-1L
cm:+constant	CM:s-add-constant-2-1L
cm:+flags	CM:s-add-flags-2-1L
cm:-	CM:s-subtract-2-1L
cm:-borrow	CM:s-subtract-borrow-2-1L
cm:-constant	CM:s-subtract-constant-2-1L
cm:/=	CM:s-ne-1L

`cm:/=constant`	CM:s-ne-constant-1L
`cm:<`	CM:s-lt-1L
`cm:<=`	CM:s-le-1L
`cm:<=constant`	CM:s-le-constant-1L
`cm:<constant`	CM:s-lt-constant-1L
`cm:=`	CM:s-eq-1L
`cm:=constant`	CM:s-eq-constant-1L
`cm:>`	CM:s-gt-1L
`cm:>=`	CM:s-ge-1L
`cm:>=constant`	CM:s-ge-constant-1L
`cm:>constant`	CM:s-gt-constant-1L
`cm:abs`	CM:s-abs-2-1L
`cm:add`	CM:s-add-3-3L
`cm:aref`	CM:aref-2L
`cm:aset`	CM:aset-2L
`cm:attach`	No change.
`cm:ceiling`	CM:s-f-ceiling-2-2L
`cm:ceiling-and-remainder`	(No direct equivalent.)
`cm:ceiling-divide`	CM:s-ceiling-3-3L
`cm:cold-boot`	No change.
`cm:compare`	CM:s-compare-3-3L
`cm:cube-from-x-y`	CM:deposit-news-coordinate-1L
`cm:detach`	No change.
`cm:enumerate`	CM:enumerate-1L
`cm:enumerate-and-count`	(No direct equivalent.)
`cm:enumerate-for-rendezvous`	(No direct equivalent.)
`cm:f*`	CM:f-multiply-2-1L
`cm:f+`	CM:f-add-2-1L
`cm:f-`	CM:f-subtract-2-1L
`cm:f/`	CM:f-divide-2-1L
`cm:f/=`	CM:f-ne-1L
`cm:f<`	CM:f-lt-1L
`cm:f<=`	CM:f-le-1L
`cm:f=`	CM:f-eq-1L
`cm:f>`	CM:f-gt-1L
`cm:f>=`	CM:f-ge-1L
`cm:fetch`	CM:get-aref32-2L
`cm:float`	CM:f-s-float-2-2L
`cm:float-abs`	CM:f-abs-2-1L
`cm:float-compare`	CM:f-compare-3-2L
`cm:float-float-signum`	CM:f-f-signum-2-1L
`cm:float-max`	CM:f-max-2-1L

`cm:float-max-scan`	CM:scan-with-f-max-1L
`cm:float-min`	CM:f-min-2-1L
`cm:float-minusp`	CM:f-lt-zero-1L
`cm:float-move`	CM:f-move-1L
`cm:float-move-constant`	CM:f-move-constant-1L
`cm:float-move-decoded-constant`	CM:f-move-decoded-constant-1L
`cm:float-negate`	CM:f-negate-2-1L
`cm:float-new-size`	CM:f-move-2L
`cm:float-plusp`	CM:f-gt-zero-1L
`cm:float-rank`	CM:f-rank-2L
`cm:float-read-array-by-cube-addresses`	(No direct equivalent.)
`cm:float-read-array-by-news-addresses`	CM:f-read-from-news-array-1L
`cm:float-read-from-processor`	CM:f-read-from-processor-1L
`cm:float-signum`	CM:s-f-signum-2-2L
`cm:float-sqrt`	CM:f-sqrt-2-1L
`cm:float-write-array-by-cube-addresses`	(No direct equivalent.)
`cm:float-write-array-by-news-addresses`	CM:f-write-to-news-array-1L
`cm:float-write-to-processor`	CM:f-write-to-processor-1L
`cm:float-zerop`	CM:f-eq-zero-1L
`cm:floor`	CM:s-f-floor-2-2L
`cm:floor-and-mod`	(No direct equivalent.)
`cm:floor-divide`	CM:s-floor-3-3L
`cm:front-end-cube-from-x-y`	CM:fe-deposit-news-coordinate
`cm:front-end-gray-code-from-integer`	CM:fe-to-gray-code
`cm:front-end-integer-from-gray-code`	CM:fe-from-gray-code
`cm:front-end-x-from-cube`	CM:fe-extract-news-coordinate
`cm:front-end-y-from-cube`	CM:fe-extract-news-coordinate
`cm:get`	CM:get-1L
`cm:get-from-east`	CM:get-from-news-1L
`cm:get-from-east-always`	CM:get-from-news-always-1L
`cm:get-from-north`	CM:get-from-news-1L
`cm:get-from-north-always`	CM:get-from-news-always-1L
`cm:get-from-south`	CM:get-from-news-1L
`cm:get-from-south-always`	CM:get-from-news-always-1L
`cm:get-from-west`	CM:get-from-news-1L
`cm:get-from-west-always`	CM:get-from-news-always-1L
`cm:get-stack-limit`	No change.
`cm:get-stack-pointer`	No change.
`cm:get-stack-upper-bound`	No change.
`cm:global-add`	CM:global-s-add-1L
`cm:global-count`	CM:global-count-bit
`cm:global-count-always`	CM:global-count-bit-always
`cm:global-float-max`	CM:global-f-max-1L

cm:global-float-min	CM:global-f-min-1L
cm:global-logand	CM:global-logand-1L
cm:global-logand-always	(No direct equivalent.)
cm:global-logior	CM:global-logior-1L
cm:global-logior-always	(No direct equivalent.)
cm:global-max	CM:global-s-max-1L
cm:global-min	CM:global-s-min-1L
cm:global-unsigned-add	CM:global-u-add-1L
cm:global-unsigned-max	CM:global-u-max-1L
cm:global-unsigned-min	CM:global-u-min-1L
cm:gray-code-from-integer	CM:u-to-gray-code-2-1L
cm:hardware-test-complete	No change.
cm:hardware-test-fast	No change.
cm:initialize-random-number-generator	The argument is no longer optional.
cm:integer-from-gray-code	CM:u-from-gray-code-2-1L
cm:integer-length	CM:s-integer-length-2-2L
cm:isqrt	CM:s-isqrt-2-1L
cm:latch-leds	No change.
cm:latch-leds-always	No change.
cm:logand	CM:logand-2-1L
cm:logand-always	(No direct equivalent.)
cm:logandc1	CM:logandc1-2-1L
cm:logandc1-always	(No direct equivalent.)
cm:logandc2	CM:logandc2-2-1L
cm:logandc2-always	(No direct equivalent.)
cm:logcount	CM:s-logcount-2-2L
cm:logeqv	CM:logeqv-2-1L
cm:logeqv-always	(No direct equivalent.)
cm:logior	CM:logior-2-1L
cm:logior-always	(No direct equivalent.)
cm:lognand	CM:lognand-2-1L
cm:lognand-always	(No direct equivalent.)
cm:lognor	CM:lognor-2-1L
cm:lognor-always	(No direct equivalent.)
cm:lognot	CM:lognot-2-1L
cm:lognot-always	(No direct equivalent.)
cm:logorc1	CM:logorc1-2-1L
cm:logorc1-always	(No direct equivalent.)
cm:logorc2	CM:logorc2-2-1L
cm:logorc2-always	(No direct equivalent.)
cm:logxor	CM:logxor-2-1L
cm:logxor-always	(No direct equivalent.)

`cm:max`	CM:s-max-2-1L
`cm:max-constant`	CM:s-max-constant-2-1L
`cm:max-scan`	CM:scan-with-s-max-1L
`cm:min`	CM:s-min-2-1L
`cm:min-constant`	CM:s-min-constant-2-1L
`cm:minusp`	CM:s-lt-zero-1L
`cm:mod`	CM:s-mod-2-1L
`cm:move`	CM:s-move-1L
`cm:move-always`	CM:s-move-always-1L
`cm:move-constant`	CM:s-move-constant-1L
`cm:move-constant-always`	CM:s-move-constant-always-1L
`cm:move-reversed`	CM:move-reversed-1L
`cm:multiply`	CM:s-multiply-3-3L
`cm:my-cube-address`	CM:my-send-address
`cm:my-x-address`	CM:my-news-coordinate-1L
`cm:my-y-address`	CM:my-news-coordinate-1L
`cm:negate`	CM:s-negate-2-1L
`cm:new-size`	CM:s-move-2L
`cm:plus-scan`	CM:scan-with-s-add-1L
`cm:plusp`	CM:s-gt-zero-1L
`cm:pop-and-discard`	No change.
`cm:power-up`	No change.
`cm:processor-cons`	(No direct equivalent.)
`cm:push-space`	No change.
`cm:rank`	CM:s-rank-2L
`cm:read-array-by-cube-addresses`	(No direct equivalent.)
`cm:read-array-by-news-addresses`	CM:s-read-from-news-array-1L
`cm:read-from-processor`	CM:s-read-from-processor-1L
`cm:rem`	CM:s-rem-2-1L
`cm:reset-stack-pointer`	No change.
`cm:round`	CM:s-f-round-2-2L
`cm:round-and-remainder`	(No direct equivalent.)
`cm:round-divide`	CM:s-round-3-3L
`cm:send`	CM:send-1L
`cm:send-with-add`	CM:send-with-s-add-1L
`cm:send-with-logand`	CM:send-with-logand-1L
`cm:send-with-logior`	CM:send-with-logior-1L
`cm:send-with-logxor`	CM:send-with-logxor-1L
`cm:send-with-max`	CM:send-with-s-max-1L
`cm:send-with-min`	CM:send-with-s-min-1L
`cm:send-with-overwrite`	CM:send-with-overwrite-1L
`cm:send-with-unsigned-max`	CM:send-with-u-max-1L
`cm:send-with-unsigned-min`	CM:send-with-u-min-1L

`cm:set-stack-limit`	No change.
`cm:set-stack-pointer`	No change.
`cm:set-stack-upper-bound`	No change.
`cm:set-system-leds-mode`	No change.
`cm:shift`	CM:s-s-shift-3-3L
`cm:signum`	CM:s-signum-2-2L
`cm:store`	CM:send-aset32-overwrite-2L
`cm:store-with-add`	CM:send-aset32-u-add-2L
`cm:store-with-logand`	(No direct equivalent.)
`cm:store-with-logior`	CM:send-aset32-logior-2L
`cm:store-with-logxor`	(No direct equivalent.)
`cm:store-with-max`	(No direct equivalent.)
`cm:store-with-min`	(No direct equivalent.)
`cm:store-with-overwrite`	CM:send-aset32-overwrite-2L
`cm:store-with-unsigned-max`	(No direct equivalent.)
`cm:store-with-unsigned-min`	(No direct equivalent.)
`cm:subtract`	CM:s-subtract-3-3L
`cm:truncate`	CM:s-f-truncate-2-2L
`cm:truncate-and-rem`	(No direct equivalent.)
`cm:truncate-divide`	CM:s-truncate-3-3L
`cm:u*`	CM:u-multiply-2-1L
`cm:u+`	CM:u-add-2-1L
`cm:u+carry`	CM:u-add-carry-2-1L
`cm:u+constant`	CM:u-add-constant-2-1L
`cm:u+flags`	CM:u-add-flags-2-1L
`cm:u-`	CM:u-subtract-2-1L
`cm:u-borrow`	CM:u-subtract-borrow-2-1L
`cm:u-constant`	CM:u-subtract-constant-2-1L
`cm:u/=`	CM:u-ne-1L
`cm:u/=constant`	CM:u-ne-constant-1L
`cm:u<`	CM:u-lt-1L
`cm:u<=`	CM:u-le-1L
`cm:u<=constant`	CM:u-le-constant-1L
`cm:u<constant`	CM:u-lt-constant-1L
`cm:u=`	CM:u-eq-1L
`cm:u=constant`	CM:u-eq-constant-1L
`cm:u>`	CM:u-gt-1L
`cm:u>=`	CM:u-ge-1L
`cm:u>=constant`	CM:u-ge-constant-1L
`cm:u>constant`	CM:u-gt-constant-1L
`cm:unsigned-add`	CM:u-add-3-3L
`cm:unsigned-ceiling`	CM:u-f-ceiling-2-2L

cm:unsigned-ceiling-and-remainder	(No direct equivalent.)
cm:unsigned-ceiling-divide	CM:u-ceiling-3-3L
cm:unsigned-compare	CM:u-compare-3-3L
cm:unsigned-float	CM:f-u-float-2-2L
cm:unsigned-floor	CM:u-f-floor-2-2L
cm:unsigned-floor-and-mod	(No direct equivalent.)
cm:unsigned-floor-divide	CM:u-floor-3-3L
cm:unsigned-integer-length	CM:u-integer-length-2-2L
cm:unsigned-isqrt	CM:u-isqrt-1-1L
cm:unsigned-logcount	CM:u-logcount-2-2L
cm:unsigned-max	CM:u-max-2-1L
cm:unsigned-max-constant	CM:u-max-constant-2-1L
cm:unsigned-max-scan	CM:scan-with-u-max-1L
cm:unsigned-min	CM:u-min-2-1L
cm:unsigned-min-constant	CM:u-min-constant-2-1L
cm:unsigned-mod	CM:u-mod-2-1L
cm:unsigned-multiply	CM:u-multiply-3-3L
cm:unsigned-negate	CM:u-negate-2-1L
cm:unsigned-new-size	CM:u-move-2-2L
cm:unsigned-plus-scan	CM:scan-with-u-add-1L
cm:unsigned-plusp	CM:u-gt-zero-1L
cm:unsigned-random	CM:u-random-1L
cm:unsigned-rank	CM:u-rank-2L
cm:unsigned-read-array-by-cube-addresses	(No direct equivalent.)
cm:unsigned-read-array-by-news-addresses	CM:u-read-from-news-array-1L
cm:unsigned-read-from-processor	CM:u-read-from-processor-1L
cm:unsigned-rem	CM:u-rem-2-1L
cm:unsigned-round	CM:u-f-round-2-2L
cm:unsigned-round-and-remainder	(No direct equivalent.)
cm:unsigned-round-divide	CM:u-round-3-3L
cm:unsigned-shift	CM:u-s-shift-2-3L
cm:unsigned-subtract	CM:u-subtract-3-3L
cm:unsigned-truncate	CM:u-f-truncate-2-2L
cm:unsigned-truncate-and-rem	(No direct equivalent.)
cm:unsigned-truncate-divide	CM:u-truncate-3-3L
cm:unsigned-write-array-by-cube-addresses	(No direct equivalent.)
cm:unsigned-write-array-by-news-addresses	CM:u-write-to-news-array-1L
cm:unsigned-write-to-processor	CM:u-write-to-processor-1L
cm:unsigned-zerop	CM:u-eq-zero-1L
cm:warm-boot	No change.
cm:write-array-by-cube-addresses	(No direct equivalent.)
cm:write-array-by-news-addresses	CM:s-write-to-news-array-1L
cm:write-to-processor	CM:s-write-to-processor-1L

`cm:x-from-cube`	CM:extract-news-coordinate-1L
`cm:y-from-cube`	CM:extract-news-coordinate-1L
`cm:zerop`	CM:s-eq-zero-1L