The
Connection Machine
System

Paris Reference Manual

Version 6.0
February 1991

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, February 1989
Revised, February 1991

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

Contents

1 Introduction 1

2 Virtual Machine Architecture 3
- 2.1 Virtual Processors and Virtual Processor Sets 5
- 2.2 Mapping VP Sets to the Physical Machine 5
- 2.3 VP Ratios 5
- 2.4 Fields 6
- 2.5 Processor Addresses 7
- 2.6 Send Addresses 7
- 2.7 NEWS Addresses 8
- 2.8 Communication across VP Sets 8
- 2.9 Geometries 8
- 2.10 Flags 10

3 Data Formats 13
- 3.1 Bit Fields 14
- 3.2 Signed Integers 14
- 3.3 Unsigned Integers 14
- 3.4 Floating-Point Numbers 15
- 3.5 Complex Floating-Point Numbers 16
- 3.6 Send Addresses 16
- 3.7 Configuration Variables 16

4 Operation Formats 19
- 4.1 Field Id's 19
- 4.2 Constant Operands 20
- 4.3 Unconditional Operations 20
- 4.4 Naming Conventions 21
- 4.5 Argument Order 23

5 Instruction Set Overview 25
- 5.1 VP Sets 25
- 5.2 Geometries 26
- 5.3 Interned Geometries and VP Sets 26
- 5.4 Fields 27

5.5 Copying Fields 27
5.6 Field Aliasing 28
5.7 Bitwise Boolean Operations 28
5.8 Operations on Flags 29
5.9 Operations on Single Bits 30
5.10 Unary Arithmetic Operations 30
5.11 Binary Arithmetic Operations 32
5.12 Optimized Floating-Point Computations 36
5.13 Arithmetic Comparisons 37
5.14 Pseudo-Random Number Generation 37
5.15 Arrays 38
5.16 General Communication 38
5.17 NEWS Communication 39
5.18 Power of Two NEWS 41
5.19 NEWS with Floating-Point Combiners 41
5.20 Scan, Reduce, Spread, and Multispread 42
5.21 Global Reduction Operations 46
5.22 Memory Data Transfers 46
5.23 Sorting 47
5.24 Timing Paris Code 47
5.25 The LEDS 48
5.26 Front End Operations 48
5.27 Environmental Interface 48

6 The C/Paris Interface **51**
6.1 C/Paris Header Files 51
6.2 C/Paris Instruction Names and Argument Types 51
6.2.1 Id Types 52
6.2.2 Operand Field Addresses 52
6.2.3 Immediate Operands 53
6.2.4 Operand Field Lengths 53
6.2.5 Miscellaneous Signed and Unsigned Values 54
6.2.6 Bit Sets and Masks 54
6.2.7 Vectors of Integers 54
6.2.8 Multi-dimensional Front-end Arrays 54
6.2.9 Symbolic Values 54
6.3 C/Paris Configuration Variables 55
6.4 Calling Paris from C 55

7 The Fortran/Paris Interface **57**
7.1 Fortran/Paris Header Files 57
7.2 Fortran/Paris Instruction Names and Argument Types 57
7.2.1 Id Types 58
7.2.2 Operand Field Addresses 58
7.2.3 Immediate Operands 59

7.2.4 Operand Field Lengths 59
7.2.5 Miscellaneous Signed and Unsigned Values 60
7.2.6 Bit Sets and Masks 60
7.2.7 Vectors of Integers 60
7.2.8 Multi-dimensional Front-end Arrays 60
7.2.9 Symbolic Values 60
7.3 Fortran/Paris Configuration Variables 61
7.4 Calling Paris from Fortran 61

8 The Lisp/Paris Interface 63
8.1 Lisp/Paris Instruction Names and Argument Types 63
8.1.1 Id Types 63
8.1.2 Operand Field Addresses 64
8.1.3 Immediate Operands 64
8.1.4 Operand Field Lengths 65
8.1.5 Miscellaneous Signed and Unsigned Values 65
8.1.6 Bit Sets and Masks 65
8.1.7 Vectors of Integers 66
8.1.8 Multi-dimensional Front-end Arrays 66
8.1.9 Symbolic Values 66
8.2 Lisp/Paris Configuration Variables 66
8.3 Calling Paris from Lisp 67

9 Dictionary of Paris Instructions 69
9.1 Conventions for Alphabetizing 69
9.2 Programming Language Syntax 70
9.2.1 Syntax of Names 70
9.2.2 Pseudocode Instruction Descriptions 70
F-ABS 73
F-C-ABS 74
S-ABS 75
C-ACOS 76
F-ACOS 77
C-ACOSH 78
F-ACOSH 79
C-ADD 80
F-ADD 81
S-ADD 83
U-ADD 85
S-ADD-CARRY 87
U-ADD-CARRY 89
S-ADD-FLAGS 91
U-ADD-FLAGS 92
F-ADD-MULT 93
ADD-OFFSET-TO-FIELD-ID 95

ALLOCATE-HEAP-FIELD 96
ALLOCATE-HEAP-FIELD-VP-SET 97
ALLOCATE-STACK-FIELD 98
ALLOCATE-STACK-FIELD-VP-SET 99
ALLOCATE-VP-SET 100
FE-ARRAY-FORMAT 101
AREF 103
AREF32 105
AREF32-SHARED 107
ASET 110
ASET32 112
ASET32-SHARED 114
C-ASIN 116
F-ASIN 117
C-ASINH 118
F-ASINH 119
C-ATAN 120
F-ATAN 121
F-ATAN2 122
C-ATANH 124
F-ATANH 125
ATTACH 126
ATTACHED 128
AVAILABLE-MEMORY 129
F-F-CEILING 131
S-CEILING 132
S-F-CEILING 134
U-CEILING 135
U-F-CEILING 137
CHANGE-FIELD-ALIAS 138
C-F-CIS 139
CLEAR-ALL-FLAGS 140
CLEAR-BIT 141
CLEAR-CONTEXT 142
CLEAR-flag 143
COLD-BOOT 144
F-COMPARE 146
S-COMPARE 147
U-COMPARE 148
COMPRESS-HEAP 149
C-CONJUGATE 150
C-COS 151
F-COS 152
C-COSH 153
F-COSH 154

CREATE-DETAILED-GEOMETRY 155
CREATE-GEOMETRY 159
CROSS-VP-MOVE 160
DEALLOCATE-GEOMETRY 163
DEALLOCATE-HEAP-FIELD 164
DEALLOCATE-STACK-THROUGH 165
DEALLOCATE-VP-SET 166
DEPOSIT-NEWS-COORDINATE 167
FE-DEPOSIT-NEWS-COORDINATE 168
DETACH 169
C-DIVIDE 171
F-DIVIDE 173
ENUMERATE 175
C-EQ 177
F-EQ 178
S-EQ 179
U-EQ 180
C-EXP 182
F-EXP 183
FE-EXTRACT-MULTI-COORDINATE 184
EXTRACT-NEWS-COORDINATE 185
FE-EXTRACT-NEWS-COORDINATE 186
DEALLOCATE-FFT-SETUP 187
C-C-FFT 188
C-FFT-SETUP 191
FIELD-VP-SET 193
F-S-FLOAT 194
F-U-FLOAT 195
F-F-FLOOR 196
S-FLOOR 197
S-F-FLOOR 199
U-FLOOR 200
U-F-FLOOR 202
FE-FROM-GRAY-CODE 203
U-FROM-GRAY-CODE 204
F-GE 205
S-GE 206
U-GE 208
GEOMETRY-AXIS-LENGTH 210
GEOMETRY-AXIS-OFF-CHIP-BITS 211
GEOMETRY-AXIS-OFF-CHIP-POS 212
GEOMETRY-AXIS-ON-CHIP-BITS 213
GEOMETRY-AXIS-ON-CHIP-POS 214
GEOMETRY-AXIS-ORDERING 215
GEOMETRY-AXIS-VP-RATIO 216

GEOMETRY-COORDINATE-LENGTH 217
GEOMETRY-RANK 218
GEOMETRY-SEND-ADDRESS-LENGTH 219
GEOMETRY-SERIAL-NUMBER 220
GEOMETRY-TOTAL-PROCESSORS 221
GEOMETRY-TOTAL-VP-RATIO 222
GET 223
GET-AREF32 224
GET-FROM-NEWS 226
GET-FROM-POWER-TWO 227
GLOBAL-C-ADD 229
GLOBAL-F-ADD 230
GLOBAL-S-ADD 231
GLOBAL-U-ADD 232
GLOBAL-COUNT-BIT 233
GLOBAL-COUNT-CONTEXT 234
GLOBAL-COUNT-flag 235
GLOBAL-LOGAND 236
GLOBAL-LOGAND-BIT 237
GLOBAL-LOGAND-CONTEXT 238
GLOBAL-LOGAND-flag 239
GLOBAL-LOGIOR 240
GLOBAL-LOGIOR-BIT 241
GLOBAL-LOGIOR-CONTEXT 242
GLOBAL-LOGIOR-flag 243
GLOBAL-LOGXOR 244
GLOBAL-F-MAX 245
GLOBAL-S-MAX 247
GLOBAL-U-MAX 248
GLOBAL-U-MAX-S-INTLEN 249
GLOBAL-U-MAX-U-INTLEN 251
GLOBAL-F-MIN 253
GLOBAL-S-MIN 255
GLOBAL-U-MIN 256
F-GT 257
S-GT 258
U-GT 260
F-IEEE-TO-VAX 263
INIT 264
INITIALIZE-RANDOM-GENERATOR 265
S-INTEGER-LENGTH 266
U-INTEGER-LENGTH 267
INTERN-DETAILED-GEOMETRY 268
INTERN-GEOMETRY 270
INTERN-IDENTICAL-VP-SET 272

INVERT-CONTEXT 273
INVERT-flag 274
IS-FIELD-AN-ALIAS 275
IS-FIELD-IN-HEAP 276
IS-FIELD-IN-STACK 277
IS-FIELD-VALID 278
IS-STACK-FIELD-NEWER 279
IS-VP-SET-VALID 280
S-ISQRT 281
U-ISQRT 283
LATCH-LEDS 285
F-LE 286
S-LE 287
U-LE 289
C-LN 291
F-LN 292
LOAD-CONTEXT 293
LOAD-flag 294
F-LOG2 295
F-LOG10 296
LOGAND 297
LOGAND-CONTEXT 298
LOGAND-CONTEXT-WITH-TEST 299
LOGAND-flag 300
LOGANDC1 301
LOGANDC2 302
S-LOGCOUNT 303
U-LOGCOUNT 304
LOGEQV 305
LOGIOR 306
LOGIOR-CONTEXT 307
LOGIOR-flag 308
LOGNAND 309
LOGNOR 310
LOGNOT 311
LOGORC1 312
LOGORC2 313
LOGXOR 314
F-LT 315
S-LT 316
U-LT 318
MAKE-FIELD-ALIAS 321
MAKE-NEWS-COORDINATE 322
FE-MAKE-NEWS-COORDINATE 323
C-MATRIX-MULTIPLY 324

S-MATRIX-MULTIPLY 326
F-MAX 328
S-MAX 330
U-MAX 332
F-MIN 334
S-MIN 336
U-MIN 338
F-MOD 340
S-MOD 342
U-MOD 344
C-MOVE 346
F-MOVE 348
S-MOVE 350
U-MOVE 352
F-MOVE-DECODED-CONSTANT 354
MOVE-REVERSED 355
F-MULT-ADD 356
F-MULT-SUB 358
F-MULT-SUBF 360
C-MULTIPLY 362
F-MULTIPLY 364
S-MULTIPLY 366
U-MULTIPLY 368
MULTISPREAD-C-ADD 370
MULTISPREAD-F-ADD 371
MULTISPREAD-S-ADD 373
MULTISPREAD-U-ADD 374
MULTISPREAD-COPY 375
MULTISPREAD-LOGAND 376
MULTISPREAD-LOGIOR 377
MULTISPREAD-LOGXOR 378
MULTISPREAD-F-MAX 379
MULTISPREAD-S-MAX 380
MULTISPREAD-U-MAX 381
MULTISPREAD-F-MIN 382
MULTISPREAD-S-MIN 383
MULTISPREAD-U-MIN 384
MY-NEWS-COORDINATE 385
MY-SEND-ADDRESS 386
C-NE 387
F-NE 388
S-NE 389
U-NE 390
C-NEGATE 392
F-NEGATE 393

S-NEGATE 394
U-NEGATE 395
F-NEWS-ADD 396
F-NEWS-ADD-MULT 398
F-NEWS-MULT 400
F-NEWS-MULT-ADD 402
F-NEWS-MULT-SUB 404
F-NEWS-SUB 406
F-NEWS-SUB-MULT 408
NEXT-STACK-FIELD-ID 410
FE-PACKED-ARRAY-FORMAT 411
F-C-PHASE 413
PHYSICAL-VP-SET 414
C-C-POWER 415
C-F-POWER 417
C-S-POWER 419
C-U-POWER 421
F-F-POWER 422
F-S-POWER 424
F-U-POWER 426
S-S-POWER 428
S-U-POWER 430
U-S-POWER 432
U-U-POWER 434
POWER-UP 436
F-RANDOM 437
U-RANDOM 438
F-RANK 439
S-RANK 441
U-RANK 443
C-READ-FROM-NEWS-ARRAY 445
F-READ-FROM-NEWS-ARRAY 448
S-READ-FROM-NEWS-ARRAY 451
U-READ-FROM-NEWS-ARRAY 454
C-READ-FROM-PROCESSOR 457
F-READ-FROM-PROCESSOR 458
S-READ-FROM-PROCESSOR 459
U-READ-FROM-PROCESSOR 460
C-RECIPROCAL 461
REDUCE-WITH-C-ADD 462
REDUCE-WITH-F-ADD 463
REDUCE-WITH-S-ADD 464
REDUCE-WITH-U-ADD 465
REDUCE-WITH-COPY 466
REDUCE-WITH-LOGAND 467

REDUCE-WITH-LOGIOR 468
REDUCE-WITH-LOGXOR 469
REDUCE-WITH-F-MAX 470
REDUCE-WITH-S-MAX 471
REDUCE-WITH-U-MAX 472
REDUCE-WITH-F-MIN 473
REDUCE-WITH-S-MIN 474
REDUCE-WITH-U-MIN 475
F-REM 476
S-REM 478
U-REM 480
REMOVE-FIELD-ALIAS 482
F-F-ROUND 483
S-ROUND 484
S-F-ROUND 486
U-ROUND 487
U-F-ROUND 489
F-S-SCALE 491
F-U-SCALE 493
SCAN-WITH-C-ADD 495
SCAN-WITH-F-ADD 497
SCAN-WITH-S-ADD 499
SCAN-WITH-U-ADD 501
SCAN-WITH-COPY 503
SCAN-WITH-LOGAND 505
SCAN-WITH-LOGIOR 507
SCAN-WITH-LOGXOR 509
SCAN-WITH-F-MAX 511
SCAN-WITH-S-MAX 513
SCAN-WITH-U-MAX 515
SCAN-WITH-F-MIN 517
SCAN-WITH-S-MIN 519
SCAN-WITH-U-MIN 521
SCAN-WITH-F-MULTIPLY 523
SEND 525
SEND-ASET32-U-ADD 527
SEND-ASET32-LOGIOR 529
SEND-ASET32-OVERWRITE 531
SEND-TO-NEWS 533
SEND-TO-QUEUE32 534
SEND-WITH-C-ADD 537
SEND-WITH-F-ADD 539
SEND-WITH-S-ADD 541
SEND-WITH-U-ADD 543
SEND-WITH-LOGAND 545

SEND-WITH-LOGIOR 547
SEND-WITH-LOGXOR 549
SEND-WITH-F-MAX 551
SEND-WITH-S-MAX 553
SEND-WITH-U-MAX 555
SEND-WITH-F-MIN 557
SEND-WITH-S-MIN 559
SEND-WITH-U-MIN 561
SEND-WITH-OVERWRITE 563
SET-BIT 565
SET-CONTEXT 566
SET-FIELD-ALIAS-VP-SET 567
SET-SAFETY-MODE 568
SET-SYSTEM-LEDS-MODE 569
SET-VP-SET 570
SET-VP-SET-GEOMETRY 571
SET-flag 572
S-S-SHIFT 573
U-S-SHIFT 575
C-C-SIGNUM 577
F-F-SIGNUM 578
S-F-SIGNUM 579
S-S-SIGNUM 580
C-SIN 581
F-SIN 582
C-SINH 583
F-SINH 584
SPREAD-FROM-PROCESSOR 585
SPREAD-WITH-C-ADD 586
SPREAD-WITH-F-ADD 587
SPREAD-WITH-S-ADD 588
SPREAD-WITH-U-ADD 589
SPREAD-WITH-COPY 590
SPREAD-WITH-LOGAND 591
SPREAD-WITH-LOGIOR 592
SPREAD-WITH-LOGXOR 593
SPREAD-WITH-F-MAX 594
SPREAD-WITH-S-MAX 595
SPREAD-WITH-U-MAX 596
SPREAD-WITH-F-MIN 597
SPREAD-WITH-S-MIN 598
SPREAD-WITH-U-MIN 599
C-SQRT 600
F-SQRT 601
STORE-CONTEXT 602

STORE-flag 603
FE-STRUCTURE-ARRAY-FORMAT 604
F-SUBF-CONST-MULT 606
F-SUB-MULT 608
C-SUBTRACT 610
F-SUBTRACT 612
S-SUBTRACT 614
U-SUBTRACT 616
S-SUBTRACT-BORROW 618
U-SUBTRACT-BORROW 620
SWAP 622
C-TAN 623
F-TAN 624
C-TANH 625
F-TANH 626
TIME 627
TIMER 629
FE-TO-GRAY-CODE 632
U-TO-GRAY-CODE 633
TRANSPOSE32 634
F-F-TRUNCATE 637
S-F-TRUNCATE 638
S-TRUNCATE 639
U-TRUNCATE 641
U-F-TRUNCATE 643
F-VAX-TO-IEEE 645
VP-SET-GEOMETRY 646
WARM-BOOT 647
C-WRITE-TO-NEWS-ARRAY 648
F-WRITE-TO-NEWS-ARRAY 651
S-WRITE-TO-NEWS-ARRAY 655
U-WRITE-TO-NEWS-ARRAY 659
C-WRITE-TO-PROCESSOR 662
F-WRITE-TO-PROCESSOR 663
S-WRITE-TO-PROCESSOR 664
U-WRITE-TO-PROCESSOR 665

List of Figures

2.1 65,536 processors 4

Customer Support

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

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

To contact Thinking Machines Customer Support:

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

For Symbolics Users Only

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

`To:` customer–support@think.com

Please supplement the automatic report with any further pertinent information.

Part I
Paris Concepts

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.

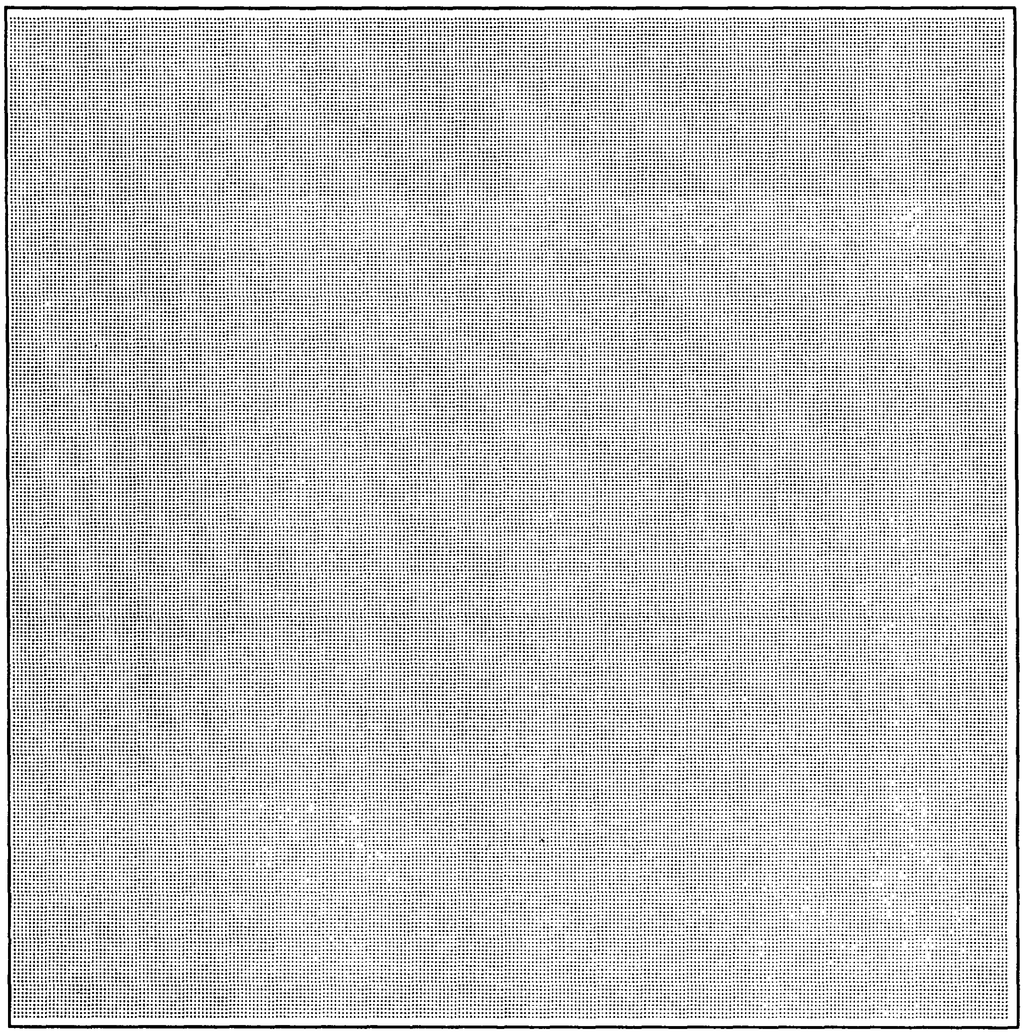


Figure 2.1: 65,536 processors

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
- complex floating-point numbers, represented as two floating-point numbers, the real part and the imaginary part
- 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 *within each processor* a separate decision is made: 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 3.7.

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 3.7.

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:

31	30 29 28 27 26 25 24 23	22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
S	exponent	significand

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 3.7. 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 Complex Floating-Point Numbers

A complex floating-point data item is specified by three parameters exactly like those for a floating-point data item: a bit address a, a significand length s, and an exponent length e. The data item consists of two consecutive floating-point data items, with the real part at address a and the imaginary part at address $a + s + e + 1$. The total number of bits in the representation is $2(s + e + 1)$, and the data item occupies the bits with addresses a through $a + 2(s + e) + 1$, inclusive.

3.6 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.)

3.7 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. **Note:** Also see the dictionary entry for CM:available-memory, which indicates how much Connection Machine memory is available for user programs.

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:*heap-compression-enabled*

When this variable is true (T, 1), automatic heap compression is enabled. See the dictionary entry for CM:compress-heap for information on explicit heap compression.

CM:*heap-compression-messages-enabled*

This variable determines whether a message is issued when heap compression occurs.

CM:*max-number-of-timers*

This represents the maximum number of timers that can be allocated by any one program using the CM:timer- functions.

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 is limited to 32 characters and 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:

c-	complex number
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. It tells how many source and destination arguments an instruction requires. The destination arguments are fields; the source arguements are fields, or in some cases constants. 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. (Note that "field-like" arguments, such as the constant used in place of a field in CM:f-multiply-constant-2-1L, are included in the number-of-fields count.)

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*][-*n*L]

An effort has been made to use full English words in the names of Paris operations. The 32-character limitation on the total length of names has made it necessary to use certain abbreviations universally:

c-	complex floating-point
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
a	always

An example is CM:f-mult-const-sub-const-a-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

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

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

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

CM: {
create-detailed-geometry
create-geometry
deallocate-geometry
geometry-axis-length
geometry-axis-off-chip-bits
geometry-axis-off-chip-pos
geometry-axis-on-chip-bits
geometry-axis-on-chip-pos
geometry-axis-ordering
geometry-axis-vp-ratio
geometry-coordinate-length
geometry-rank
geometry-send-address-length
geometry-total-processors
geometry-total-vp-ratio
}

These operations create, destroy, and otherwise manipulate geometries. Note the many operations that inquire about the shape of the geometry and various axis attributes.

5.3 Interned Geometries and VP Sets

Paris supports a special class of geometry and VP set objects: *interned* objects. The interning facility is especially useful to compiler writers because interned objects may be accessed by description rather than by ID and are automatically reused as needed.

CM: {
intern-geometry
intern-detailed-geometry
intern-identical-vp-set
}

These operations create interned geometries and VP sets.

Note that interned geometries and VP sets are substantively different kinds of objects from their uninterned couterparts. For instance, a geometry created with CM:create-geometry is never interchangeable with a geometry created with CM:intern-geometry.

5.4 Fields

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

These operations create, destroy, and otherwise manipulate fields. Fields are used to contain data to be operated upon in parallel. Most Paris operations require one or more fields as arguments.

CM:available-memory

This instruction indicates the number of bits of memory, per virtual processor, currently available for allocation on either the heap or stack.

CM:compress-heap

Automatic heap compression is enabled by default. Programmers can control heap compression explicitly by setting the configuration variable CM:*heap-compression-enabled* to NIL (false, 0) and then calling the above instruction to control fragmentation.

5.5 Copying Fields

A number of operations are provided simply to copy data from one place to another.

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

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.

$$\text{CM:}\left\{\begin{array}{ll} \text{move-reversed} & \left\{\begin{array}{c}\sim\\ \text{-always}\end{array}\right\} \\ \text{swap} & \left\{\begin{array}{c}\sim\\ \text{-always}\end{array}\right\}\text{-2}\end{array}\right\}\text{-1L}$$

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.

$$\text{CM:cross-vp-move}\left\{\begin{array}{c}\sim\\ \text{-always}\end{array}\right\}\text{-1L}$$

The cross-vp-move instruction copies all or a portion of one multidimensional block of data from the current VP set into a similarly shaped region in another VP set.

5.6 Field Aliasing

$$\text{CM:}\left\{\begin{array}{l}\text{change-field-alias}\\ \text{is-field-an-alias}\\ \text{make-field-alias}\\ \text{remove-field-alias}\\ \text{set-field-alias-vp-set}\end{array}\right\}$$

These operations create, destroy, and manipulate field aliases. A *field alias* is a field ID that references a field already referenced by at least one other field ID. By using field aliases, it is possible to reference the same Connection Machine memory field from within different VP sets.

5.7 Bitwise Boolean Operations

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

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

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

5.8 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.

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

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).

5.9 Operations on Single Bits

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

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

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.

5.10 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.)

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.

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

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).

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

A shift instruction performs an arithmetic shift by a specified number of bit positions. Paris supports shifts on either signed or unsigned source fields.

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

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

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

These Paris instructions support converting floating-point numbers between the IEEE format used in the Connection Machine system and VAX floating-point format.

$$\text{CM:f-}\left\{\begin{array}{l}\text{ieee-to-vax}\\ \text{vax-to-ieee}\end{array}\right\}\text{-1L}$$

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.

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

Floating-point and complex absolute value, negation, and square root are provided.

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

Floating-point floor, ceiling, truncation, rounding, and signum operations are available.

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

Complex signum, conjugate, and reciprocal operations are provided.

$$\text{CM:c-}\left\{\begin{array}{l}\text{c-signum}\\ \text{c-conjugate}\\ \text{c-reciprocal}\end{array}\right\}\left\{\begin{array}{l}\text{-1-1L}\\ \text{-2-1L}\end{array}\right\}$$

These two unary operations on complex operands yield floating-point destination values. One calculates the absolute value and the other calculates the phase of each complex source value.

$$\text{CM:f-c-}\left\{\begin{array}{l}\text{abs}\\ \text{phase}\end{array}\right\}\ \text{-2-1L}$$

For both floating-point and complex numbers, Paris provides a complete set of transcendental and trigonometric functions, including hyperbolic functions and their inverses.

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

In addition, the cis instruction is available. It yields a complex field in which the real part is the cosine of the floating-point source and the imaginary part is the sine of the source.

CM:c-f-cis-2-1L

5.11 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.)

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 by the signed and unsigned round and truncate instructions, whose names reflect the rounding or truncation that must occur when integer 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.

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

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

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.)

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

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. Since the add-carry and subtract-borrow instructions take the *carry-flag* as input as well as setting it upon completion, these instructions can be chained. (The one exception to this rule are the -add-carry-3-3L instructions, which do not set the *carry-flag* because it is unclear what carry means in the 3L case.)

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

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.

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

For floating-point and complex numbers, the usual addition, subtraction, multiplication, and division operations are provided. Note that there are unconditional versions of these operations in Paris; these can be much faster than the conditional versions when floating-point hardware is used.

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

For floating-point numbers, max and min operations are provided, along with floating-point remainder and modulo division operations, and a floating-point exponentiation instruction.

$$\text{CM:f} \left\{\begin{matrix}\text{-max}\\ \text{-min}\\ \text{-mod}\\ \text{-rem}\\ \text{-f-power}\end{matrix}\right\} \left\{\begin{matrix}\sim\\ \text{-constant}\end{matrix}\right\} \left\{\begin{matrix}\text{-2-1L}\\ \text{-3-1L}\end{matrix}\right\}$$

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

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

Other useful floating-point operations include scaling, as well as exponentiating to an integer power.

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

Paris supports integer exponentiation instructions for both signed and unsigned operands.

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

Exponentiation of complex number is supported for powers of any data type.

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

The **exp** operations calculate e^s for complex operands and 2^s for floating-point operands, where s is the value of the *source* field and e is the base of the natural logarithms.

$$\texttt{CM:}\left\{\begin{array}{l}\texttt{c}\\ \texttt{f}\end{array}\right\}\texttt{-exp}\left\{\begin{array}{l}\texttt{-1-1L}\\ \texttt{-2-1L}\end{array}\right\}$$

Instructions are provided that calculate the base 2 or base 10 logarithm of a floating-point source field or the natural logarithm of a complex source field.

$$\text{CM:}\left\{\begin{array}{l}\text{f-log2}\\ \text{f-log10}\\ \text{c-ln}\end{array}\right\}\left\{\begin{array}{l}\text{-1-1L}\\ \text{-2-1L}\end{array}\right\}$$

A two-input arctangent operation is provided.

CM:f-atan2-3-1L

5.12 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.

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.

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

Note: Where using the term -always in an unconditional instruction name would cause the name to exceed the 32 character limit for Paris instruction names, the implementation uses the term -a instead. In the above chart, this is the case only for instructions that contain const twice. An example is CM:f-sub-const-mult-const-a-1L.

These compound instructions combine floating-point multiplication with reverse subtraction in a variety of ways. The unconditional versions may be faster than the conditional versions. (Note that the name CM:subf-const-mult-const-a-1L uses -a instead of -always in order to stay within the 32-character Paris operation name length limit.)

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

5.13 Arithmetic Comparisons

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

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

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

$$\text{CM:c-}\left\{\begin{matrix}\text{eq}\\ \text{ne}\end{matrix}\right\}\left\{\begin{matrix}\sim\\ \text{-constant}\\ \text{-zero}\end{matrix}\right\}\text{-1L}$$

5.14 Pseudo-Random Number Generation

Paris provides a built-in generator of uniformly distributed pseudo-random numbers. Use these instructions to generate unsigned integers over a specified range, or floating-point numbers in the range from 0.0 (inclusive) to 1.0 (exclusive).

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

CM:initialize-random-generator

5.15 Arrays

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

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

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 *slicewise* 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 slicewise array elements directly. Only special operations designed to operate on slicewise arrays can properly fetch or store slicewise array elements. Examples are CM:aref32-2L and CM:aset32-2L. 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.

5.16 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.

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

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

CM:send-to-queue32-1L

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

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.

5.17 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.

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

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

The instructions in the chart below all work with NEWS coordinates.

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

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.

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 $x = (\textit{extract-news-coordinate}(g, axis, k) + distance)$
    :**downward** : let $x = (\textit{extract-news-coordinate}(g, axis, k) - distance)$
  let $x' = x \bmod \textit{geometry-axis-length}(g, axis)$
  return $\textit{deposit-news-coordinate}(g, k, axis, x')$

5.18 Power of Two NEWS

One special-purpose instruction performs near-neighbor communication between processors that are separated by a particular distance. That distance must be a power of two, measured in intervening processors and inclusive of the source processor.

$$\text{CM:get-from-power-two}\left\{\begin{matrix}\sim\\ \text{-always}\end{matrix}\right\}\text{-1L}$$

5.19 NEWS with Floating-Point Combiners

A series of special-case combining operations that use NEWS communication are supported. These instructions calculate a form of binary addition, subtraction, and multiplication in which one operand is retrieved from a NEWS neighbor of the destination field.

$$\text{CM:f-news}\left\{\begin{array}{llll}
\left\{\begin{matrix}\text{-add}\\ \text{-sub}\\ \text{-mult}\end{matrix}\right\} & \left\{\begin{matrix}\sim\\ \text{-always}\end{matrix}\right\} & \left\{\begin{matrix}\text{-2-1L}\\ \text{-3-1L}\end{matrix}\right\} & \\
\left\{\begin{matrix}\text{-add-const}\\ \text{-sub-const}\end{matrix}\right\} & \left\{\begin{matrix}\sim\\ \text{-a}\end{matrix}\right\} & \text{-3-1L} & \\
\left\{\begin{matrix}\sim\\ \text{-const}\end{matrix}\right\} & \text{-mult-4-1L} & & \\
\text{-mult-const} & \left\{\begin{matrix}\sim\\ \text{-a}\end{matrix}\right\} & \text{-4-1L} & \\
\text{-mult} & \left\{\begin{matrix}\sim\\ \text{-const}\end{matrix}\right\} & \left\{\begin{matrix}\text{-add}\\ \text{-sub}\end{matrix}\right\} & \text{-4-1L}
\end{array}\right\}$$

5.20 Scan, Reduce, Spread, and Multispread

The spread-from-processor operation provides a simple way to take the value found in one processor and replicate it throughout the machine.

$$\text{CM:spread-from-processor-}\left\{\begin{array}{c}\sim\\ \text{a-}\end{array}\right\}\text{1L}$$

Extending this idea, the following operations provide extremely powerful combinations of communication and computation in regular patterns on multidimensional grids.

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

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 and enumerate operations generalize the 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 one axis, and they are ordered by their coordinates along that axis. Only active processors participate in a scan operation; all scan and scan-like operations are conditional. The set of active processors along a NEWS axis is called the *scan subclass.*

The scan result computed for a given processor may be produced by combining values from all processors within a scan subclass. That is, all active processors along a specified axis may contribute to the result for each processor along that axis. However – and more usefully – a scan subclass may be divided into pieces called *scan sets*, such that each 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 :segment-bit, then the *sbit* field is interpreted as a "segment bit."

The segment bit divides a scan class unconditionally (that is, without respect to context) into segments, and a separate scan operation is done within each segment. Operationally speaking, a processor (active 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 value is 1.

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 is interpreted as 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, or scan set. 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. In an upward scan, a processor with a start bit of 1 is the first participant in a scan set that includes its neighbor with the next higher coordinate along the specified NEWS axis; in a downward scan, the same processor begins a scan set that includes its neighbors with lower NEWS axis coordinates.

  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. (Exclusive scans are described below.)

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

A scan operation furthermore behaves as if all the processors in the specified scan set were passed over ("scanned") in linear order; therefore the result computed for a given processor, k, depends only on processors below it in the ordering, or only on processors above it, depending on the direction of the scan. The *direction* and *inclusion* operands determine which processors within the scan set 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 subset for processor k will contain only processors below k in the ordering. If *direction* is :downward, then the scan subset for k will contain only processors above k in the ordering.

If *inclusion* is :exclusive, then the scan subset for processor k will not contain k itself. If *inclusion* is :inclusive, then the scan subset 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 the scan, spread, reduce, rank, and multispread operations.

Consider representing several NEWS coordinate values in a single integer called a *multi-coordinate*. We can define two 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,

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

The Paris instruction CM:multispread-copy-1L actually requires a multi-coordinate as an argument and the instruction CM:fe-extract-multi-coordinate constructs a multi-coordinate. Beyond this, the notion of a multi-coordinate providess a useful conceptual building block in the following pseudo-code definitions.

Now we can 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.

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

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

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

function *scan-set*$(g, k, axis, direction, smode, sbit)$ is
  let $C = \textit{scan-subclass}(g, k, axis)$
  function $coord(s) = \textit{extract-news-coordinate}(g, axis, s)$
  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 $S = \textit{scan-set}(g, k, axis, direction, smode, sbit)$
  function $coord(s) = \textit{extract-news-coordinate}(g, axis, s)$
  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 $\{\, m \mid m \in S \wedge coord(m) \geq coord(k) \,\}$

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

A reduce operation is like a spread, except that instead of storing the result in every active processor in the scan class, 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.

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 0 1 1 0 0 1 1 1 0
		sbit	0 0 1 0 0 0 1 0 0 0 0 0 0 1 0 0
		source	1 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

5.21 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.

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

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.

5.22 Memory Data Transfers

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

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

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

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.

5.23 Sorting

Paris provides operations for sorting data based on integer or floating-point keys.

$$\text{CM:}\left\{\begin{array}{l}\text{f-}\\\text{s-}\\\text{u-}\end{array}\right\}\text{rank-2-L}$$

The rank operation does not actually put records into sorted order. Instead, it produces ranking information from which appropriate send addresses can be calculated; a send operation can then be used to put the records in order. This allows the ranking operation to deal only with sort keys and not with entire records.

5.24 Timing Paris Code

A set of instructions beginning with CM:timer- provide a timing facility with microsecond precision.

$$\text{CM:timer-}\left\{\begin{array}{l}\text{clear}\\\text{start}\\\text{stop}\\\text{print}\\\text{read-starts}\\\text{read-elapsed}\\\text{read-cm-busy}\\\text{read-cm-idle}\\\text{read-run-state}\\\text{set-starts}\end{array}\right\}$$

From the Lisp/Paris interface, this timing facility is incorporated in the macro CM:time, which may be wrapped around code in order to time it.

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

CM:set-system-leds-mode

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

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

5.26 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; they deal primarily with Gray codes and NEWS coordinates.

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

5.27 Environmental Interface

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

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

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 IDs to indicate one or more regions of Connection Machine memory to be processed. Such field IDs 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 5.22).

front-end-array A pointer to a front-end 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 3.7 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 -lm
```

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 IDs to indicate one or more regions of Connection Machine memory to be processed. Such field IDs 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 must be INTEGER for an integer value, and 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 5.22).

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 3.7 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 -lparisfort -lparis
```

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

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

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

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 5.22).
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 3.7 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.

Part II

Paris Dictionary

Chapter 9

Dictionary of Paris Instructions

9.1 Conventions for Alphabetizing

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

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

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

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

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

9.2 Programming Language Syntax

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

9.2.1 Syntax of Names

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

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

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

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

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

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

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

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

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

9.2.2 Pseudocode Instruction Descriptions

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

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

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

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

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

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

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

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

F-ABS

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

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

Operands

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

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

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

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

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

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

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

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

F-C-ABS

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

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

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

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

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

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

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

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

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

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

S-ABS

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

Formats

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

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

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

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

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

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

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

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

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source*[k] ≥ 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.)

C-ACOS

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

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

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

source The field ID of the complex source field.

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

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

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

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

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

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

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

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

F-ACOS

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

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

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

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

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

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

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

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

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

C-ACOSH

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

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

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

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

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

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

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

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

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

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

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

F-ACOSH

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

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

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

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

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

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

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

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

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

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

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

C-ADD

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

Formats

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

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

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

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

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

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

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

Definition

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

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

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

F-ADD

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

Formats

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

Operands

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

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

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

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

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

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

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

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

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

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

ADD

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

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

S-ADD

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

Formats

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

Operands

dest The field ID of the signed integer destination field.

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

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

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

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

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

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

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

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

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

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

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

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow$ *source1*$[k]$ + *source2*$[k]$
    carry-flag$[k] \leftarrow$ ⟨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. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-ADD

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

Formats

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

Operands

dest The field ID of the unsigned integer destination field.

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

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

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

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

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

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

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

Overlap

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

Flags

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

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

Context

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

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

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

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

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

S-ADD-CARRY

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

Formats

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

Operands

dest The field ID of the signed integer destination field.

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

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

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

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

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

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

Overlap

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

Flags

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

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

Context

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

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

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

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

U-ADD-CARRY

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

Formats

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

Operands

dest The field ID of the unsigned integer destination field.

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

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

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

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

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

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

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

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

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

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

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

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

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

S-ADD-FLAGS

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

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

Operands

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

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

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

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

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

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    Compute *source1*$[k]$ + *source2*$[k]$
    carry-flag$[k] \leftarrow$ ⟨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 *dest*, *source1*, *source2*, *len*

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

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

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

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

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

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

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

F-ADD-MULT

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

Formats

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

Operands

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

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

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

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

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

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

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

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

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

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

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

Definition For every virtual processor k in the *current-vp-set* do
 if (always or $context\text{-}flag[k] = 1$) then
  $dest[k] \leftarrow (source1[k] + source2[k]) \times source3[k]$
  if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

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 in the destination field.

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

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

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

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

but may be faster.

ADD-OFFSET-TO-FIELD-ID

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

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

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

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

ALLOCATE-HEAP-FIELD

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

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

Operands *len* An unsigned integer, the length in bits of the field to be allocated.

Result A field ID, identifying the new field ID.

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

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

ALLOCATE-HEAP-FIELD-VP-SET

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

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

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

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

ALLOCATE-STACK-FIELD

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

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

Operands *len* An unsigned integer, the length, in bits, of the field to be allocated.

Result A field ID, identifying the new field ID.

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

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

ALLOCATE-STACK-FIELD-VP-SET

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

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

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

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

ALLOCATE-VP-SET

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

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

Operands	*geometry-id*	A geometry ID.
Result	A VP set ID, identifying the newly allocated VP set.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

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

FE-ARRAY-FORMAT

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

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

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

Operands

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

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

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

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

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

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

Result The array format descriptor specified.

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

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

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

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

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

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

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

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

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

AREF

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

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

Operands

dest The field ID of the destination field.

array The field ID of the source array field.

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

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

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

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

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

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

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

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

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

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

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

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

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

AREF32

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

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

Operands

dest The field ID of the destination field.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

AREF32-SHARED

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

Formats

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

Operands

dest The field ID of the destination field.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ASET

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

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

Operands

source The field ID of the source field.

array The field ID of the destination array field.

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

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

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

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

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

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

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

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

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

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

This is a simple form of array modification, for arrays stored in the memory of individual processors. Each processor has an array index stored in the field *index*. This is used to index into an *array*, whose length in bits should be *index-limit* × *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* × *element-len*, where *i* is the unsigned number stored at *index*, in all selected processors.

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

ASET32

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

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

Operands

source The field ID of the source field.

array The field ID of the destination array field.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ASET32-SHARED

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

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

Operands

source The field ID of the source field.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

C-ASIN

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

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

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

source The field ID of the complex source field.

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

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

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

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

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

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

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

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

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

F-ASIN

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

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

Operands

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

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

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

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

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

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

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

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

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

C-ASINH

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

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

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

source The field ID of the complex source field.

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

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

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

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

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

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

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

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

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

F-ASINH

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

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

Operands

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

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

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

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

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

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

C-ATAN

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

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

Operands
dest The field ID of the complex destination field.

source The field ID of the complex source field.

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

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

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

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

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

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

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

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

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

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

F-ATAN

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

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

Operands

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

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

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

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

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

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

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

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

F-ATAN2

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

Formats

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

Operands

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

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

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

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

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

Definition

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

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

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

C-ATANH

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

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

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

source The field ID of the complex source field.

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

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

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

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

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

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

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

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

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

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

F-ATANH

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

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

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

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

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

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

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

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

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

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

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

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

ATTACH

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

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

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

Operands

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ATTACHED

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

Formats result ← CM:attached

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

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

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

AVAILABLE-MEMORY

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

Formats result ← CM:available-memory

Result An unsigned integer, the number of bits available.

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

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

F-F-CEILING

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

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

Operands

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

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

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

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

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

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

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

Note that overflow cannot occur.

S-CEILING

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

Formats

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

Operands

dest The field ID of the signed integer quotient field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

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

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

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

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

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

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

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

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

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

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
$dest[k] \leftarrow \left\lceil \frac{source1[k]}{source2[k]} \right\rceil$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$
if *source2*$[k] = 0$ then
test$[k] \leftarrow 1$
else *test*$[k] \leftarrow 0$

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

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

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

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

S-F-CEILING

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

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

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

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

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

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

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-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 floating-point number, is rounded to the nearest integer in the direction of $+\infty$. The result is stored into the *dest* field as a signed integer.

U-CEILING

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

Formats

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

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

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

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

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

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

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

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

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

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

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

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

if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$
if *source2*$[k] = 0$ then
  test$[k] \leftarrow 1$
else *test*$[k] \leftarrow 0$

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

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

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

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

U-F-CEILING

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

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

Operands

dest The field ID of the unsigned integer destination field.

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

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

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

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

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

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

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

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

CHANGE-FIELD-ALIAS

Changes the referent of the specified field alias.

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

Operands	*alias-id*	An alias field ID. This must be an alias field ID returned by CM:make-field-alias. It need not be in the current VP set.
	field-id	A field ID. This must be a field id returned by CM:allocate-stack-field or CM:allocate-heap-field; it may *not* be an offset into a field. The field need not be in the current VP set.
Context		This operation is unconditional. It does not depend on the *context-flag*.

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

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

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

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

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

C-F-CIS

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

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

Operands

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

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

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

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

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

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

CLEAR-ALL-FLAGS

Clears all flags (but not the context bit).

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

Context The non-always operation is conditional.

The always operation is unconditional.

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

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

CLEAR-BIT

Clears a specified memory bit.

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

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

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

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

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

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

COLD-BOOT

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

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

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

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

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

An unsigned integer, the number of virtual processors.

An unsigned integer, the number of physical processors.

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

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

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

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

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

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

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

Bootstrapping a Connection Machine system includes the following actions:

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

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

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

F-COMPARE

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

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

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

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

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

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

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

S-COMPARE

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

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

Operands

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

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

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

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

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

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

U-COMPARE

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

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

Operands

dest The field ID of the signed integer destination field.

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

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

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

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

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

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

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

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

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

COMPRESS-HEAP

Invokes the heap compression mechanism on demand.

Formats CM:compress-heap

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

Heap compression removes heap memory fragmentation.

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

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

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

C-CONJUGATE

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

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

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

source The field ID of the complex source field.

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

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

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

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

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

C-COS

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

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

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

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

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

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

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

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

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

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

F-COS

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

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

Operands

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

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

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

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

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

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

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

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

C-COSH

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

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

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

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

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

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

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

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

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

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

F-COSH

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

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

Operands

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

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

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

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

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

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

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

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

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

CREATE-DETAILED-GEOMETRY

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

The Fortran/Paris interface defines CM_axis_descriptor as an array:

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

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

DESCRIPTOR_ARRAY(1,*I*) is the length of axis *I*
DESCRIPTOR_ARRAY(2,*I*) is the weight of axis *I*
DESCRIPTOR_ARRAY(3,*I*) is the ordering of axis *I*
DESCRIPTOR_ARRAY(4,*I*) is the on-chip bits of axis *I*
DESCRIPTOR_ARRAY(6,*I*) is the off-chip bits of axis *I*

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

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

```
(deftype cm:axis-order () '(member :news-order :send-order))

(defstruct CM:axis-descriptor
  (length 0) (weight 0) (ordering :news-order)
  (on-chip-bits 0) (off-chip-bits 0))
```

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

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

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

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

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

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

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

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

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

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

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

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

CREATE-GEOMETRY

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

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

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

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

Result A geometry ID, identifying the newly created geometry.

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

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

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

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

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

CROSS-VP-MOVE

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

Formats

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

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

Operands

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

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

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

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

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

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

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

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

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

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

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

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

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

Context This operation is conditional.

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

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

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

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

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

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

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

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

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

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

DEALLOCATE-GEOMETRY

Declare that a geometry will no longer be used.

Formats CM:deallocate-geometry *geometry-id*

Operands *geometry-id* A geometry ID.

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

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

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

DEALLOCATE-HEAP-FIELD

Declare that a heap field will no longer be used.

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

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

Context This operation is unconditional. It does not depend on the *context-flag*.

By this operation a user program declares that a field will no longer be used. The system is permitted to reclaim any and all resources associated with that field, in particular the memory that it occupied. It is an error for the user program to give the specified field ID as an argument to any Paris operation once it has been deallocated.

DEALLOCATE-STACK-THROUGH

Declare that a stack field and all fields allocated more recently than it will no longer be used.

Formats CM:deallocate-stack-through *stack-field-id*

Operands *stack-field-id* A field ID.

Context This operation is unconditional. It does not depend on the *context-flag.*

By this operation a user program declares that the specified field on the stack, and all fields allocated more recently than it, will no longer be used. (Note that any fields allocated more recently than the specified field are necessarily closer to the top of the stack.) The system is permitted to reclaim any and all resources associated with those fields, in particular the memory that they occupied. It is an error for the user program to give the field ID of a deallocated field as an argument to any Paris operation.

DEALLOCATE-VP-SET

Declare that a VP set will no longer be used.

Formats CM:deallocate-vp-set *vp-set-id*

Operands *vp-set-id* A VP set ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

By this operation a user program declares that a VP set will no longer be used. The system is permitted to reclaim any and all resources associated with that VP set. It is an error for the user program to give the specified VP set ID as an argument to any Paris operation once it has been deallocated.

It is an error to deallocate a VP set for which there are still fields that have not yet been deallocated. The user should first deallocate all fields belonging to that VP set, except the flags, which are deallocated automatically when the VP set is deallocated.

DEPOSIT-NEWS-COORDINATE

Modifies a send address to reflect a specific NEWS coordinate.

Formats

CM:deposit-news-coordinate-1L *geometry*, *dest*/*send-address*, *axis*, *coordinate*, *slen*

CM:deposit-news-constant-1L *geometry*, *dest*/*send-address*, *axis*, *coordinate-value*, *slen*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

dest The field ID of the unsigned integer destination field. (In the instruction formats currently provided, the *dest* field is always the same as the *send-address* source field. The length of this field is implicitly the same as *geometry-send-address-length*(*geometry*).)

send-address The field ID of the unsigned integer send address field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

coordinate The field ID of the unsigned integer NEWS coordinate. field. This specifies the position along the corrsponding axis of the processor whose send address is to be calculated.

coordinate-value An unsigned integer immediate operand to be used as the NEWS coordinate along the specified axis.

slen The length of the *coordinate* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap For CM:deposit-news-coordinate-1L, the *coordinate* field must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] ← *deposit-news-coordinate*(*geometry*, *send-address*, *axis*, *coordinate*)

where *deposit-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates equal to those for the processor identified by *send-address*.

FE-DEPOSIT-NEWS-COORDINATE

Calculates on the front end the modification of a send address to reflect a specific NEWS coordinate.

Formats result ← CM:fe-deposit-news-coordinate *geometry*, *send-address*, *axis*, *coordinate*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

send-address An unsigned integer immediate operand to be used as the send address of some processor.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along the specified axis.

Result An unsigned integer, the send address of the processor whose coordinate along the specified axis is *coordinate* and whose coordinate along all other axes equals those of *send-address*.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Return *deposit-news-coordinate*(*geometry*, *send-address*, *axis*, *coordinate*)

where *deposit-news-coordinate* is as defined on page 40.

This function calculates, entirely on the front end, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates equal to those for the processor identified by *send-address*.

DETACH

Detaches the specified front-end computer from the Connection Machine hardware previously allocated for and attached to it.

This instruction is available only from the Lisp/Paris interface. For Fortran/Paris and C/Paris users, the equivalent functionality is provided by the shell level cmdetach command, documented in the *CM System User's Guide.*

Formats CM:detach *front-end-name*, *suppress-confirmation*

Operands *front-end-name* The name of a front end, or a list of a front end name and a bus-interface specifier. This argument is optional.

suppress-confirmation The confirmation suppression flag. This argument is optional. If supplied and not false, then the interactive query and prompt requesting confirmation of the detach operation is suppressed.

Context This operation is unconditional. It does not depend on the *context-flag*.

The facility for detaching Connection Machine hardware is provided in different ways in the Lisp/Paris interface (on the one hand) and the C/Paris and Fortran/Paris interfaces (on the other hand).

In the Lisp/Paris interface, CM:detach is a function of two arguments. The arguments are optional.

In most normal use no argument is specified. In this case the front end executing the call to CM:detach releases all Connection Machine hardware to which it had been attached, resetting relevant parts of the Nexus so that the front end can no longer issue macroinstructions to the Connection Machine system. (An error is signalled if in fact no hardware had been attached in the first place.) This use of CM:detach is the normal way of releasing attached hardware and will not disrupt users on other front ends.

If a *front-end-name* argument is specified, it must be the name of a front end that is connected to the same Connection Machine system (that is, Nexus) as the front end executing the call, or perhaps a list of a front end name and a small integer identifying a bus interface on that front end. A front end name may be either a string or a symbol. Examples (assuming, for the sake of exposition, that front end computers are named after Shakespearean characters):

```
(detach 'hamlet)          ;Detach front end named Hamlet
```

```
(detach "lear" t)          ;Detach front end named Lear, and don't confirm
(detach '(desdemona 1))    ;Detach bus interface 1 of front end Desdemona
```

Specifying the name of the front end that is executing the call has the same effect as specifying no argument; the front end is gracefully detached. But specifying the name of some other front end forcibly detaches that other front end, possibly disrupting any ongoing interaction with the Connection Machine system. The external communications network is used to send a message to the detached front end to inform its user that it has been forcibly detached.

There are two sets of initialization forms, kept in the variables CM:*before-detach-initializations* and CM:*after-detach-initializations*, that are evaluated before and after anything else occurs.

In the C/Paris and Fortran/Paris interfaces, the detaching operation is performed by a user command cmdetach at shell level. See the *Front End Subsystems* manual or the cmdetach man page.

C-DIVIDE

The quotient of two complex source values is placed in the destination field. **Note:** Integer division is performed by the **round**, **truncate**, **rem**, and **mod** operations.

Formats

CM:c-divide-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-divide-always-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-divide-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-divide-always-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-divide-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-divide-const-always-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-divide-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:c-divide-const-always-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:c-divinto-2-1L	*dest/source2*, *source1*, *s*, *e*
CM:c-divinto-always-2-1L	*dest/source2*, *source1*, *s*, *e*
CM:c-divinto-constant-2-1L	*dest/source2*, *source1-value*, *s*, *e*
CM:c-divinto-const-always-2-1L	*dest/source2*, *source1-value*, *s*, *e*
CM:c-divinto-constant-3-1L	*dest*, *source2*, *source1-value*, *s*, *e*
CM:c-divinto-const-always-3-1L	*dest*, *source2*, *source1-value*, *s*, *e*

Operands

dest — The field ID of the complex destination field. This is the quotient.

source1 — The field ID of the complex first source field. This is the dividend.

source2 — The field ID of the complex second source field. This is the divisor.

source1-value — A complex immediate operand to be used as the first source.

source2-value — A complex immediate operand to be used as the second source.

s, *e* — The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

test-flag is set if division by zero occurs; otherwise it is unaffected.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if (always or *context-flag*[k] = 1) then
dest[k] ← *source1*[k]/*source2*[k]
if *source2*[k] = 0 then *test-flag*[k] ← 1
if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The *source1* operand is divided by the *source2* operand, treating both as complex numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-DIVIDE

The quotient of two floating-point source values is placed in the destination field.

Note: Integer division is performed by the round, truncate, rem, and mod operations.

Formats

CM:f-divide-2-1L	*dest/source1, source2, s, e*
CM:f-divide-always-2-1L	*dest/source1, source2, s, e*
CM:f-divide-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-divide-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:f-divinto-2-1L	*dest/source2, source1, s, e*
CM:f-divinto-always-2-1L	*dest/source2, source1, s, e*
CM:f-divinto-constant-2-1L	*dest/source2, source1-value, s, e*
CM:f-divinto-const-always-2-1L	*dest/source2, source1-value, s, e*
CM:f-divide-3-1L	*dest, source1, source2, s, e*
CM:f-divide-always-3-1L	*dest, source1, source2, s, e*
CM:f-divide-constant-3-1L	*dest, source1, source2-value, s, e*
CM:f-divide-const-always-3-1L	*dest, source1, source2-value, s, e*
CM:f-divinto-constant-3-1L	*dest, source2, source1-value, s, e*
CM:f-divinto-const-always-3-1L	*dest, source2, source1-value, s, e*

Operands

dest The field ID of the floating-point destination field. This is the quotient.

source1 The field ID of the floating-point first source field. This is the dividend.

source2 The field ID of the floating-point second source field. This is the divisor.

source1-value A floating-point immediate operand to be used as the first source.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

test-flag is set if division by zero occurs; otherwise it is unaffected.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flags may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*[k] = 1) then
    dest[k] ← *source1*[k]/*source2*[k]
    if *source2*[k] = 0 then *test-flag* ← 1
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The *source1* operand is divided by the *source2* operand, treating both as floating-point numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

ENUMERATE

The destination field in every selected processor receives the number of processors below or above it in some ordering of the processors.

Formats CM:enumerate-1L *dest*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the unsigned integer destination field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The *sbit* field must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $S_k = scan\text{-}subset(k, axis, len, direction, inclusion, smode, sbit)$
    $dest[k] \leftarrow |S_k|$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:enumerate-1L operation stores into the *dest* field of each selected processor the size of the scan subset for that processor. This means that every processor within a scan set of size N will receive a different integer in the range 0 to $N-1$ (for an exclusive enumeration) or in the range 1 to N (for an inclusive enumeration).

A call to CM:enumerate-1L is equivalent to the sequence below, but may be faster.

ENUMERATE

CM:u-move-constant-1L *temp*, 1, *len*
CM:scan-with-u-add-1L *dest*, *temp*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*
CM:u-subtract-constant-1L *dest*, 1, *len*

C-EQ

Compares two complex source values. The *test-flag* is set if they are equal, and otherwise it is cleared.

Formats

CM:c-eq-1L	*source1*, *source2*, *s*, *e*
CM:c-eq-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:c-eq-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source. For CM:c-eq-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do

```
if context-flag[k] = 1 then
   if source1[k] = source2[k]
      test-flag[k] ← 1
   else
      test-flag[k] ← 0
```

Two operands are compared as complex numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; +0 and −0 are considered to be equal.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

F-EQ

Compares two floating-point source values. The *test-flag* is set if they are equal, and otherwise is cleared.

Formats

CM:f-eq-1L	*source1*, *source2*, *s*, *e*
CM:f-eq-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-eq-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-eq-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] = *source2*[*k*]
      test-flag[*k*] ← 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). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-EQ

Compares two signed integer source values. The *test-flag* is set if they are equal, and otherwise is cleared.

Formats

CM:s-eq-1L	*source1*, *source2*, *len*
CM:s-eq-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-eq-constant-1L	*source1*, *source2-value*, *len*
CM:s-eq-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-eq-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do

```
if context-flag[k] = 1 then
   if source1[k] = source2[k] then
      test-flag[k] ← 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 equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-EQ

Compares two unsigned integer source values. The *test-flag* is set if they are equal, and otherwise is cleared.

Formats

CM:u-eq-1L	*source1*, *source2*, *len*
CM:u-eq-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-eq-constant-1L	*source1*, *source2-value*, *len*
CM:u-eq-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-eq-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *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 equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-EXP

The exponent of the complex source field is stored in the complex destination field.

Formats

CM:c-exp-1-1L *dest/source*, *s*, *e*
CM:c-exp-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    dest[*k*] ← exp *source*[*k*]
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

The value e^s is stored into the *dest* field, where *s* is the value of the *source* field, and *e* is the base of the natural logarithms; $e \approx 2.718281828\ldots$

F-EXP

Calculates, in each selected processor, the exponential function e^x of the floating-point source field and stores it in the floating-point destination field.

Formats CM:f-exp-1-1L *dest*/*source*, *s*, *e*
CM:f-exp-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source*[k] = $+\infty$ then
      dest[k] ← $+\infty$
    else if *source*[k] = $-\infty$ then
      dest[k] ← $+0$
    else
      dest[k] ← exp *source*[k]
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

Call the value of the *source* field s; the value e^s is stored into the *dest* field, where $e \approx 2.718281828\ldots$ is the base of the natural logarithms.

FE-EXTRACT-MULTI-COORDINATE

Calculates, on the front end, the NEWS multi-coordinate of a processor specified by send-address. A multi-coordinate is needed in order to use the CM:multispread-copy-1L instruction.

Formats result ← CM:fe-extract-multi-coordinate *geometry*, *axis-mask*, *send-address*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

send-address An unsigned integer immediate operand to be used as the send address of some processor.

Result An unsigned integer, the NEWS multi-coordinate of the specified processor along the specified axes.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Let $\textit{axis-set} = \{\, m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m\rangle = 1)\,\}$
Return *extract-multi-coordinate*(*geometry*, *axis-set*, *send-address*)
where *extract-multi-coordinate* is as defined on page 44.

This function calculates, entirely on the front end, the NEWS multi-coordinate of a processor along specified NEWS axes. The axes are indicated by the *axis-mask* argument; the processor is identified by its send-address.

EXTRACT-NEWS-COORDINATE

Determines the NEWS coordinate of a processor specified by send-address.

Formats CM:extract-news-coordinate-1L *geometry, dest, axis, send-address, dlen*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

dest The field ID of the unsigned integer destination field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

send-address The field ID of the send address field. For each processor, this identifies the send address of some other processor.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if *context-flag*[k] = 1 then
  dest[k] ← *extract-news-coordinate*(*geometry*, *axis*, *send-address*)

where *extract-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the NEWS coordinate of a processor along a specified NEWS axis. The axis is indicated by the *axis* argument; the processor is identified by its send-address.

FE-EXTRACT-NEWS-COORDINATE

Calculates, on the front end, the NEWS coordinate of a processor specified by send-address.

Formats result ← CM:fe-extract-news-coordinate *geometry*, *axis*, *send-address*

Operands

geometry A geometry ID. This geometry determines the NEWS dimensions to be used.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

send-address An unsigned integer immediate operand to be used as the send address of some processor.

Result An unsigned integer, the NEWS coordinate of the specified processor along the specified axis.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Return *extract-news-coordinate*(*geometry*, *axis*, *send-address*)

where *extract-news-coordinate* is as defined on page 40.

This function calculates, entirely on the front end, the NEWS coordinate of a processor along a specified NEWS axis. The axis is indicated by the *axis* argument; the processor is identified by its send-address.

DEALLOCATE-FFT-SETUP

Deallocates a front-end setup descriptor that has been used to prepare information for execution of an FFT routine.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. A more efficient set of FFT routines are included in the CM Scientific Subroutines Library.

Formats CMSSL:deallocate-fft-setup *setup-id*

Operands *setup-id* The ID of the FFT setup descriptor to be deallocated.

Context This is a front-end operation. It does not depend on the value of the *context-flag*.

This routine may be used to remove an FFT setup descriptor when it is no longer needed. The setup-id argument must have been obtained by a call to CMSSL:c-fft-setup.

An fft setup descriptor occupies memory both on the front end and on the Connection Machine. It is therefore wise to free this space by calling CMSSL:deallocate-fft-setup after completion of all FFT routines that use the specified setup descriptor.

C-C-FFT

The Discrete Fourier Transform of the complex source field is calculated using a Fast Fourier Transform (FFT) algorithm. The complex result is stored in the destination field.

A Fourier transform routine converts (possibly multidimensional) sequences between the time or space domain and the frequency domain. This type of transform has a variety of useful applications. For example, an FFT can be used to filter discrete signals, to smooth input data or output images, to interpolate or extrapolate from a given data set, to measure the correlation between two samples, or to multiply polynomials and extremely large integers.

The Fast Fourier Transform is called a fast transform because it exhibits $O(N \log N)$ complexity, where O is the order of complexity and N is the length of the input sequence. By comparison, the Discrete Fourier Transform exhibits only $O(N^2)$ complexity.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix c-c- to signify that single-precision complex operands are involved. A more efficient set of FFT routines are included in the CM Scientific Subroutines Library.

Formats CMSSL:c-c-fft *dest*, *source*, *setup*, *ops*, *source-bit-order*, *dest-bit-order*, *source-cm-order*, *dest-cm-order*, *scale*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

setup The setup-id. This must be a setup ID returned by CMSSL:c-fft-setup. The geometry information of the setup must be identical to that of the source and destination fields.

ops A front-end vector of operation identifiers. Each element specifies whether the corresponding source axis is transformed and, if so, by what method. Valid vector element values are :f-xform (FFT_f_xform in C; 1 in Fortran) for a forward transform, :i-xform (FFT_i_xfrom in C; 2 in Fortran) for an inverse transform, and :nop (FFT_nop in C; 0 in Fortran) for no transform.

source-bit-order A front-end vector of input bit orderings. Each element identifies the bit ordering of the corresponding source axis and must be either :normal or :bit-reversed. (The corresponding values are are FFT_normal and FFT_bit_reversed in C, and 0 and 1 in Fortran, respectively.)

dest-bit-order A front-end vector of output bit orderings. Each element identifies the bit ordering of the corresponding destination axis and must be either :normal or :bit-reversed. (The corresponding values are are FFT_normal and FFT_bit_reversed in C, and 0 and 1 in Fortran, respectively.)

source-cm-order A front-end vector of input orderings. Each element declares the addressing mode of the corresponding source axis and must be one of the following: :send-order, :news-order, or :default. (The corresponding values are FFT_send_order, FFT_news_order, and FFT_default in C, and 1, 2, and 0 in Fortran, respectively.)

A value of :default causes the current ordering of an axis to be used.

dest-cm-order A front-end vector of output orderings. Each element declares the addressing mode of the corresponding destination axis and must be one of the following: :send-order, :news-order, or :default. (The corresponding values are FFT_send_order, FFT_news_order, and FFT_default in C, and 1, 2, and 0 in Fortran, respectively.)

A value of :default causes the current ordering of an axis to be used.

scale A front-end vector of output scaling methods. Each element specifies whether the corresponding destination axis is rescaled and, if so, by what method. Valid values are :noscale for no rescaling, :scale-sqrt for scaling by the inverse square root of the FFT, and :scale-n for scaling by the inverse of the size of the FFT. (The corresponding values are FFT_noscale, FFT_scale_sqrt, and FFT_scale_n in C, and 0, 1, and 2 in Fortran, respectively.)

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. FFT performance is slightly better if the two fields are identical.

Context This operation is unconditional. It does not depend on the *context-flag.*

Definition For every virtual processor k in the *current-vp-set* do

$$dest[k] \leftarrow FFT(source[k])$$

The Discrete Fourier Transform of the *source* field is stored in the *dest* field. A multi-dimensional transform is computed by performing the transform across each dimension in sequence.

The source and destination fields must either belong to the same VP set or to VP sets of identical shape and size.

The *ops*, *source-bit-order*, *dest-bit-order*, *source-cm-order*, *dest-cm-order*, and *scale* arguments are one-dimensional front-end arrays. The length of each is equal to the rank of the setup geometry.

By convention, a Fast Fourier Transform operation reverses the order of the data bits when storing the result in the destination. The vectors *source-bit-order* and *dest-bit-order* specify whether the source and destination data are treated as normal or as bit-reversed.

Along any given dimension of the data's geometry, the Connection Machine FFT instruction is most efficient for data arranged in send order. Many FFT applications do not depend on the order of the data elements. The *dest-cm-order* and *source-cm-order* arguments are therefore provided to permit the most efficient execution possible along each dimension.

C/Paris code that calls the Paris FFT routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris FFT.

Fortran/Paris code should include the line

```
  INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris FFT.

C-FFT-SETUP

Allocates a front-end setup descriptor for use with the CMSSL:fft Fast Fourier Transform routines and returns a setup ID.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix c- to signify that single-precision complex operands are involved. A more efficient set of FFT routines are included in the CM Scientific Subroutines Library.

Formats result ← CMSSL:c-fft-setup *geometry-id*

Operands *geometry* A geometry ID.

Result The ID of the newly created FFT setup descriptor.

Context This is a front-end operation. It does not depend on the value of the *context-flag.*

This routine computes information needed to perform a Fast Fourier Transform (FFT), stores it in an FFT setup descriptor, and return the setup-id.

In Lisp/Paris, a setup ID is a structure of type CMSSL:fft-setup. In C/Paris, it is a pointer to a structure of type FFT_fft_setup_t. In Fortran/Paris it is an integer.

The *geometry* argument must be a geometry ID returned by a call to CM:create-geometry, CM:create-detailed-geometry, intern-geometry, or intern-detailed-geometry.

The returned setup ID is a valid value for the *setup* argument to any CMSSL FFT routine if the following requirement is obeyed. The geometries of the FFT source and destination fields must be identical to that of the setup geometry.

This routine must be reinvoked whenever the geometry of an FFT source field VP set is changed. CMSSL:c-fft-setup allocates memory both on the front end and on the CM. To free this memory, use CMSSL:deallocate-fft-setup.

C/Paris code that calls the Paris FFT routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris FFT.

Fortran/Paris code should include the line

```
INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris FFT.

FIELD-VP-SET

Returns the VP set associated with a field.

Formats result ← CM:field-vp-set *field*

Operands *field* The field ID of the field.

Result A VP set ID, identifying the VP set to which the field belongs.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *vp-set*(*field*)

This operation may be used to determine the VP set with which any given field is associated. The field need not belong to the current VP set.

F-S-FLOAT

Converts a signed integer field into a floating-point number field.

Formats CM:f-s-float-2-2L *dest*, *source*, *slen*, *s*, *e*

Operands

- *dest* — The field ID of the floating-point destination field.
- *source* — The field ID of the signed integer source field.
- *slen* — The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
- *s*, *e* — The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Overlap — The fields *dest* and *source* must not overlap in any manner.

Flags — *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    dest[*k*] ← *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 field ID of the floating-point destination field.

source The field ID of the unsigned integer source field.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

s, *e* The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    dest[*k*] ← *source*[*k*]
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

The *source* field, treated as an unsigned integer, is converted to a floating-point number, which is stored into the *dest* field.

F-F-FLOOR

In each selected processor, calculates the largest integer that is not greater than a specified floating-point value and stores the result as a floating-point field.

Formats CM:f-f-floor-1-1L *dest/source*, *s*, *e*
CM:f-f-floor-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *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-FLOOR

The floor of the quotient of two signed integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:s-floor-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-floor-2-1L	*dest/source1*, *source2*, *len*
CM:s-floor-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-floor-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:s-floor-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the signed integer quotient field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-floor-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-floor-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-floor-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
$dest[k] \leftarrow \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$
if $source2[k] = 0$ then
$test[k] \leftarrow 1$
else $test[k] \leftarrow 0$

The signed integer *source1* operand is divided by the signed integer *source2* operand. The floor of the mathematical quotient is stored into the signed integer memory field *dest.*

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len.*

S-F-FLOOR

Calculates, in each selected processsor, the largest integer that is not greater than a specified floating-point value and stores the result as a signed integer field.

Formats CM:s-f-floor-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

- *dest* — The field ID of the signed integer destination field.
- *source* — The field ID of the floating-point source field.
- *len* — The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
- *s*, *e* — The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] ← $\lfloor$*source*[k]$\rfloor$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $-\infty$, which is stored into the *dest* field as a signed integer.

U-FLOOR

The floor of the quotient of two unsigned integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:u-floor-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-floor-2-1L	*dest/source1, source2, len*
CM:u-floor-3-1L	*dest, source1, source2, len*
CM:u-floor-constant-2-1L	*dest/source1, source2-value, len*
CM:u-floor-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

dlen For CM:s-floor-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:s-floor-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:s-floor-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then

$$dest[k] \leftarrow \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$

if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] $\leftarrow$ 1
  else *overflow-flag*[k] $\leftarrow$ 0
if *source2*[k] = 0 then
  test[k] $\leftarrow$ 1
else *test*[k] $\leftarrow$ 0

The unsigned integer *source1* operand is divided by the unsigned integer *source2* operand. The floor of the mathematical quotient is stored into the unsigned integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-F-FLOOR

Converts floating-point source field values into unsigned integers by rounding towards $-\infty$.

Formats CM:u-f-floor-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands
- *dest* — The field ID of the unsigned integer destination field.
- *source* — The field ID of the floating-point source field.
- *len* — The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
- *s*, *e* — The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap — The fields *dest* and *source* must not overlap in any manner.

Flags — *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest $\leftarrow \lfloor source \rfloor$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of $-\infty$. The result is stored into the *dest* field as an unsigned integer.

FE-FROM-GRAY-CODE

Calculates, on the front end, the Gray code representation of a specified integer.

Formats result ← CM:fe-from-gray-code *code*

Operands *code* An unsigned integer immediate operand to be used as the Gray encoding, represented as a nonnegative integer.

Result An unsigned integer, the nonnegative integer represented by *code*.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = \mathit{integer\text{-}length}(\mathit{code})$

Return $\bigoplus_{j=0}^{n-1} \left\lfloor \frac{\mathit{code}}{2^j} \right\rfloor$

This function calculates, entirely on the front end, the integer represented by a bit-string encoding *code* in a particular reflected binary Gray code.

Note that the binary value 0 is always equivalent to a Gray code string that is all 0-bits.

U-FROM-GRAY-CODE

Converts a bit string representing a Gray-coded integer value to the usual unsigned binary representation.

Formats CM:u-from-gray-code-1-1L *dest*/*source*, *len*
CM:u-from-gray-code-2-1L *dest*, *source*, *len*

Operands *dest* The field ID of the unsigned integer destination field.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
for j from $len - 1$ to 0 do

$$dest[k]\langle j\rangle \leftarrow \left(\bigoplus_{i=j}^{len-1} source[k]\langle i\rangle\right)$$

The *source* operand is considered to be a value in a particular reflected binary Gray code. The position of that value in the standard Gray code sequence is calculated as an unsigned binary integer. This is done as follows: bit i of the result is 1 if and only if all the bit positions of the source to the left of (and including) bit i contain an odd number of 1's.

Note that a Gray code string that is all 0-bits is always equivalent to the binary value 0.

F-GE

Compares two floating-point source values. The *test-flag* is set if the first is greater than or equal to the second, and otherwise is cleared.

Formats

CM:f-ge-1L	*source1*, *source2*, *s*, *e*
CM:f-ge-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-ge-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-ge-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] $\geq$ *source2*[*k*]
      test-flag[*k*] $\leftarrow$ 1
    else
      test-flag[*k*] $\leftarrow$ 0

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than or equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-GE

Compares two signed integer source values. The *test-flag* is set if the first is greater than or equal to the second, and otherwise is cleared.

Formats

CM:s-ge-1L	*source1, source2, len*
CM:s-ge-2L	*source1, source2, slen1, slen2*
CM:s-ge-constant-1L	*source1, source2-value, len*
CM:s-ge-zero-1L	*source1, len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-ge-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if *context-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. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-GE

Compares two unsigned integer source values. The *test-flag* is set if the first is greater than or equal to the second, and otherwise is cleared.

Formats

CM:u-ge-1L	*source1*, *source2*, *len*
CM:u-ge-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-ge-constant-1L	*source1*, *source2-value*, *len*
CM:u-ge-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-ge-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] ≥ *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 greater than or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

GEOMETRY-AXIS-LENGTH

Returns the length of one axis of a geometry.

Formats result ← CM:geometry-axis-length *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose length is desired.

Result An unsigned integer, the length of the indicated axis.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *axis-descriptors*(*geometry-id*)[*axis*].*length*

This operation returns the length of the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-AXIS-OFF-CHIP-BITS

Returns the number of off-chip bits that are allocated for the specified NEWS axis within the off-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-off-chip-bits *geometry-id*, *axis*

Operands

geometry-id A geometry ID.

axis An unsigned integer, the number of the axis whose off-chip bits count is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the count of the off-chip bits associated with the specified *axis*. If *axis* has no off-chip bits, the result is 0.

Context This operation is unconditional. It does not depend on the *context-flag*.

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates how many of the off-chip bits within the off-chip bits partition are used in the send addresses of virtual processors that lie along the specified dimension.

Note that the integer returned does not indicate the total number of all off-chip bits within the send address but the number of off-chip bits used for a particular dimension.

GEOMETRY-AXIS-OFF-CHIP-POS

Returns the starting position for the off-chip bits that are allocated for the specified NEWS axis within the off-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-off-chip-pos *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose off-chip bits position is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the location in the send address of the first off-chip bit associated with the specified axis. This is zero-based; the first location is numbered 0.

Context This operation is unconditional. It does not depend on the *context-flag.*

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates where, within the off-chip bits partition, the off-chip bits for the specified dimension lie.

Note that the integer returned does not indicate the absolute position of all off-chip bits within the send address but the position of the off-chip bits for a particular dimension relative to the start of all off-chip bits in an address.

GEOMETRY-AXIS-ON-CHIP-BITS

Returns the number of on-chip bits that are allocated for the specified NEWS axis within the on-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-on-chip-bits *geometry-id*, *axis*

Operands

geometry-id A geometry ID.

axis An unsigned integer, the number of the axis whose on-chip bits count is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the count of the on-chip bits associated with the specified *axis*. If *axis* has no on-chip bits, the result is 0.

Context This operation is unconditional. It does not depend on the *context-flag*.

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates how many of the on-chip bits within the on-chip bits partition are used in the send addresses of virtual processors that lie along the specified dimension.

Note that the integer returned does not indicate the total number of all on-chip bits within the send address but the number of on-chip bits used for a particular dimension.

GEOMETRY-AXIS-ON-CHIP-POS

Returns the starting position for the on-chip bits that are allocated for the specified NEWS axis within the on-chip bits portion of a send address associated with the specified geometry.

Formats result ← CM:geometry-axis-on-chip-pos *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose on-chip bits position is desired. This must be between 0 and the rank of the geometry minus one. Note that VP set geometry dimensions are zero-based; the first axis is numbered 0.

Result An unsigned integer, the location in the send address of the first on-chip bit associated with the specified axis. This is zero-based; the first location is numbered 0.

Context This operation is unconditional. It does not depend on the *context-flag*.

The send addresses associated with a particular geometry are partitioned into three portions: off-chip bits, on-chip bits, and VP bits.

The off-chip bits identify one CM chip. The on-chip bits identify one physical processor on that CM chip. The VP bits give an offset in the memory of the physical processor and thus identify a virtual processor within that physical processor.

Within each partition, a certain number of bits are used for each dimension of the geometry. This instruction indicates where, within the on-chip bits partition, the on-chip bits for the specified dimension lie.

Note that the integer returned does not indicate the absolute position of all on-chip bits within the send address but the position of the on-chip bits for a particular dimension relative to the start of all on-chip bits in an address.

GEOMETRY-AXIS-ORDERING

Returns the ordering of one axis of a geometry.

Formats result ← CM:geometry-axis-ordering *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose ordering is desired.

Result The ordering of the specified axis (either :news-order or :send-order).

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *axis-descriptors*(*geometry-id*)[*axis*].*ordering*

This operation returns the ordering of the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-AXIS-VP-RATIO

Returns the VP ratio of one axis of a geometry.

Formats result ← CM:geometry-axis-vp-ratio *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose VP ratio is desired.

Result An unsigned integer, the VP ratio of the indicated axis.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *axis-descriptors*(*geometry-id*)[*axis*].*vp-ratio*

This operation returns the VP ratio of the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-COORDINATE-LENGTH

Returns the number of bits needed to represent a NEWS coordinate.

Formats result ← CM:geometry-coordinate-length *geometry-id*, *axis*

Operands *geometry-id* A geometry ID.

axis An unsigned integer, the number of the axis whose coordinate length is desired.

Result An unsigned integer, the number of bits required to represent a coordinate for the indicated axis.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *integer-length*(*axis-descriptors*(*geometry-id*)[*axis*].*length* − 1)

This operation returns the number of bits required to represent (as an unsigned integer) a NEWS coordinate for the specified axis of the geometry specified by the *geometry-id*.

GEOMETRY-RANK

Returns the number of axes for a geometry.

Formats result ← CM:geometry-rank *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the rank (number of axes) of the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag.*

Definition Return *rank*(*geometry*)

This operation returns the number of grid axes for the geometry specified by the *geometry-id.*

GEOMETRY-SEND-ADDRESS-LENGTH

Returns the number of bits needed to represent a send-address.

Formats result ← CM:geometry-send-address-length *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the number of bits required to represent a send-address for a processor in the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = rank(geometry\text{-}id)$

Return $\sum_{j=0}^{n-1} integer\text{-}length(axis\text{-}descriptors(geometry\text{-}id)[j].length - 1)$

This operation returns the number of bits required to represent a send-address for a virtual processor in any VP set whose geometry is the one specified by the *geometry-id*. This will be equal to the sum of the numbers of bits needed to represent NEWS coordinates for all the axes.

GEOMETRY-SERIAL-NUMBER

Assigns a unique number to the specified geometry.

Formats result ← CM:geometry-serial-number *geometry-id*

Operands *geometry-id* A geometry ID. This geometry ID must be obtained by calling CM:create-geometry or CM:create-detailed-geometry.

Result The serial number that uniquely identifies the geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

A unique number, the serial number, is assigned to the specified geometry. This facilitates geometry-based caching; geometry serial numbers are useful as hash table keys.

Note that geometry ID's are not unique identifiers. After a geometry is deallocated, its ID may be reused for another geometry. In contrast, geometry serial numbers are guaranteed to be unique.

GEOMETRY-TOTAL-PROCESSORS

Returns the number of virtual processors for a geometry.

Formats result ← CM:geometry-total-processors *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the total number of processors in the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = rank(geometry\text{-}id)$

Return $\prod_{j=0}^{n-1} axis\text{-}descriptors(geometry\text{-}id)[j].length$

This operation returns the total number of virtual processors in any VP set whose geometry is the one specified by the *geometry-id*. This will be equal to the product of the lengths of all the axes.

GEOMETRY-TOTAL-VP-RATIO

Returns the total VP ratio for a specified geometry.

Formats result ← CM:geometry-total-vp-ratio *geometry-id*

Operands *geometry-id* A geometry ID.

Result An unsigned integer, the number of virtual processors represented within each physical processor for the specified geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Let $n = rank(geometry\text{-}id)$

Return $\prod_{j=0}^{n-1} axis\text{-}descriptor(geometry\text{-}id)[j].vp\text{-}ratio$

This operation returns the total VP ratio for a specified geometry. This is equal to the total number of virtual processors for the geometry, divided by the total number of physical processors.

GET

Each selected processor gets a message from a specified source processor, possibly itself. A source processor may supply messages even if it is not selected. Messages are all retrieved from the same memory address within each source processor, and all the source processors may be in a VP set different from the VP set of the destination processors.

Formats CM:get-1L *dest*, *send-address*, *source*, *len*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates from which processor a message is retrieved.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

Overlap The *send-address* and *dest* may overlap in any manner. Similarly, the *send-address* and *source* may overlap in any manner. However, it is forbidden for the *dest* and *source* to overlap.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *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 within any specified source processor (possibly itself). A source processor may supply messages even if it is not selected. Messages are all retrieved from the same memory address within each source processor.

Formats CM:get-aref32-2L *dest*, *send-address*, *array*, *index*, *dlen*, *index-len*, *index-limit*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates from which processor a message is retrieved.

array The field ID of the source array field. This must be stored in the special format required by CM:aref32.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *dlen*.

dlen The length of the *dest* field.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of *array*.

Overlap The *send-address* and *array* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *array* and *dest* to overlap.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $index[k] < index\text{-}limit$ then
      let $r = geometry\text{-}total\text{-}vp\text{-}ratio(geometry(current\text{-}vp\text{-}set))$
      let $m = \lfloor \frac{k}{r} \rfloor \bmod 32$
      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 field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Note that in the conditional case the storing of data depends only on the *context-flag* of the processor receiving the data, not on the *context-flag* of the processor from which the data is obtained.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    $\textit{dest}[k] \leftarrow \textit{source}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})]$
where *news-neighbor* is as defined on page 40.

The *dest* field in each processor receives the contents of the *source* field of that processor's neighbor along the NEWS axis specified by *axis* in the direction specified by *direction*.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

GET-FROM-POWER-TWO

Each processor gets a message from a processor that is a specified distance away in the NEWS grid. The distance must be a power of two.

Formats

CM:get-from-power-two-1L	*dest*, *source*, *axis*, *log-2-distance*, *direction*, *len*
CM:get-from-power-two-always-1L	*dest*, *source*, *axis*, *log-2-distance*, *direction*, *len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

log-2-distance An unsigned integer immediate operand to be used as the base 2 logarithm of *distance*, where *distance* must be a power of 2.

direction Either :upward or :downward.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Note that in the conditional case data storage depends only on the *context-flag* of the processor receiving the data, not on the *context-flag* of the processor from which the data is obtained.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    let $g = geometry(current\text{-}vp\text{-}set)$
    $dest[k] \leftarrow source[news\text{-}relative(g, k, axis, direction, log\text{-}2\text{-}distance)]$

where *news-relative* is defined in the NEWS Communication section of the Instruction Set Overview chapter.

The *dest* field in each processor receives the contents of the *source* field of that processor's relative along the NEWS axis specified by *axis*, in the direction specified by *direction*, and at the distance specified by *log-2-distance*.

GET-FROM-POWER-TWO

The immediate operand *log-2-distance*, is $\log_2$ *distance*, where *distance* is the distance, along axis *axis*, between each destination processor and the source processor from which it retrieves data. In terms of this operand, *distance* is $2^{log\text{-}2\text{-}distance}$.

If *direction* is :upward then each processor retrieves data from a relative whose NEWS coordinate is (*coordinate* + *distance* mod *axis-length*). For most processors, this means getting from a processor whose coordinate is greater. The GET wraps around however; the processor whose coordinate is greatest retrieves data from the processor whose coordinate is (0 + *distance*).

If *direction* is :downward then each processor retrieves data from a relative whose NEWS coordinate is (*coordinate* − *distance* mod *axis-length*). For most processors, this means getting from a processor whose coordinate is less. The GET wraps around however; the processor whose coordinate is zero retrieves data from the processor whose coordinate is (*max-coordinate*(*axis*) − *distance*).

GLOBAL-C-ADD

The sum of the values in the complex source field is returned to the front end as a complex number.

Formats result ← CM:global-c-add-1L *source*, *s*, *e*

Operands *source* The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(s + e + 1)$.

Result A complex number, the sum of the *source* field.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $P = \{ m \mid 0 \leq m < \text{CM:*user-send-address-limit*} \}$
Let $S = \{ m \mid m \in P \land \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $+0$ to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-c-add-1L operation sums the *source* field values from all selected processors, treated as complex numbers. The sum is sent to the front-end computer as a complex number and returned as the result of the operation. If there are no selected processors, then the value $+0$ is returned.

GLOBAL-F-ADD

One floating-point number is examined in every selected processor, and the sum of all these fields is returned to the front end as a floating-point number.

Formats result ← CM:global-f-add-1L *source*, *s*, *e*

Operands	*source*	The field ID of the floating-point source field.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Result: A floating-point number, the sum of the *source* fields.

Overlap: There are no constraints, because overlap is not possible.

Context: This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return +0 to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-f-add operation sums the *source* fields, treated as floating-point numbers, in all selected processors. The sum is sent to the front-end computer as a floating-point number and returned as the result of the operation. If there are no selected processors, then the value +0 is returned.

GLOBAL-S-ADD

One signed integer is examined in every selected processor, and the sum of all these fields is returned to the front end as a signed integer.

Formats result $\leftarrow$ CM:global-s-add-1L *source*, *len*

Operands	*source*	The field ID of the signed integer source field.
	len	The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
Result	A signed integer, the sum of the *source* fields.	
Overlap	There are no constraints, because overlap is not possible.	
Context	This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.	

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-s-add operation sums the *source* fields, treated as signed integers, in all selected processors. The sum is sent to the front-end computer as a signed integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned.

GLOBAL-U-ADD

One unsigned integer is examined in every selected processor, and the sum of all these fields is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-add-1L *source*, *len*

Operands *source* The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the sum of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\sum_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-u-add operation sums the *source* fields, treated as unsigned integers, in all selected processors. The sum is sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned.

GLOBAL-COUNT-BIT

One bit is examined in every selected processor, and the count of bits that are 1 is delivered to the front end.

Formats result ← CM:global-count-bit *source*
result ← CM:global-count-bit-always *source*

Operands *source* The field ID of the source bit (a one-bit field).

Result An unsigned integer, the number of 1 bits.

Overlap There are no constraints, because overlap is not possible.

Context The non-always operations are conditional. The result returned depends only upon processors whose *context-flag* is 1.

The always operations are unconditional. The result returned does not depend on the *context-flag*.

Definition If always then
let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{source}[m] = 1 \}$
else
let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{source}[m] = 1 \}$
return $|S|$ to front end

The CM:global-count-bit operation sums the one-bit *bit-source* fields in all selected processors; in other words, it returns a count of how many processors have a 1-bit in that field. The count is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned.

Using CM:global-count-bit is identical in effect to using CM:global-unsigned-add on a one-bit field, but may be faster.

GLOBAL-COUNT-CONTEXT

Returns the number of active processors.

Formats result ← CM:global-count-context

Context This operation is unconditional.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \}$
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 field ID of the source field.
	len	The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
Result		An unsigned integer to be regarded as a vector of bits, the bitwise logical AND of all the *source* fields.
Overlap		There are no constraints, because overlap is not possible.
Context		This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $2^{len} - 1$ to front end
else
  return $\left(\bigwedge_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-logand operation combines the *source* fields in all selected processors by performing bitwise logical AND operations. A bit is 1 in the result field if the corresponding bit is a 1 in *all* of the fields to be combined. The resulting combined field is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value $-2^{\text{len}} - 1$ is returned, representing a field of length *len* containing all ones.

GLOBAL-LOGAND-BIT

One memory bit is examined in each processor; 1 is returned if they are all 1, 0 if any is zero.

Formats

result	←	CM:global-logand-bit	*source*
result	←	CM:global-logand-bit-always	*source*

Operands *source* The field ID of the source field.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical AND of all the *source* bits.

Overlap There are no constraints, because overlap is not possible.

Context The non-always operations are conditional. The result returned depends only upon processors whose *context-flag* is 1.

The always operations are unconditional. The result returned does not depend on the *context-flag*.

Definition

If always then
  let $S = current\text{-}vp\text{-}set$
else
  let $S = \{ m \mid m \in current\text{-}vp\text{-}set \wedge context\text{-}flag[m] = 1 \}$
If $|S| = 0$ then
  return 1 to front end
else
  return $\left(\bigwedge_{m \in S} source[m] \right)$ 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 $\left(\bigwedge_{m \in \mathit{current\text{-}vp\text{-}set}} \mathit{context\text{-}flag}[m] \right)$ to front end

If all processors are active, then 1 is returned to the front end; otherwise 0 is returned.

GLOBAL-LOGAND-flag

Return 1 if a specified flag is set in all processors, 0 if it is clear in any processor.

Formats CM:global-logand-test
CM:global-logand-overflow

Context This operation is conditional.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{flag}[m] = 1 \}$
If $|S| = 0$ then
return 0 to front end
else
return $\left(\bigwedge_{m \in S} \textit{flag}[m] \right)$ to front end
where *flag* is *test-flag* or *overflow-flag*, as appropriate.

If all processors have the indicated flag set, then 1 is returned to the front end; otherwise 0 is returned.

GLOBAL-LOGIOR

One field is examined in every selected processor, and the bitwise logical inclusive OR of all these fields is returned to the front end as an unsigned integer.

Formats result ← CM:global-logior-1L *source*, *len*

Operands *source* The field ID of the source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical INCLUSIVE OR of all the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\bigvee_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-logior operation combines the *source* fields in all selected processors by performing bitwise logical INCLUSIVE OR operations. A bit is 1 in the result field if the corresponding bit is a 1 in *any* of the fields to be combined. The resulting combined field is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned, representing a field of length *len* containing all zeros.

GLOBAL-LOGIOR-BIT

One memory bit is examined in each processor; 1 is returned if any is 1, 0 if they are all zero.

Formats

result	←	CM:global-logior-bit	*source*
result	←	CM:global-logior-bit-always	*source*

Operands *source* The field ID of the source field.

Result An unsigned integer to be regarded as a vector of bits, the bitwise logical OR of all the *source* bits.

Overlap There are no constraints, because overlap is not possible.

Context The non-always operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

The always operation is unconditional. The result returned does not depend on the *context-flag*.

Definition

If always then
  let $S = \textit{current-vp-set}$
else
  let $S = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \}$
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-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 $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{flag}[m] = 1 \}$
If $|S| = 0$ then
 return 0 to front end
else
 return $\left(\bigvee_{m \in S} \textit{flag}[m] \right)$ to front end
where *flag* is *test-flag* or *overflow-flag*, as appropriate.

If any processor has the indicated flag set, then 1 is returned to the front end; otherwise 0 is returned.

GLOBAL-LOGXOR

One field is examined in every selected processor, and the bitwise exclusive OR of all these fields is returned to the front end as an unsigned integer.

Formats result ← CM:global-logxor-1L *source*, *len*

Operands

source	The field ID of the source field.
len	The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result — An unsigned integer to be regarded as a vector of bits, the bitwise logical exclusive OR of all the *source* fields.

Overlap — There are no constraints, because overlap is not possible.

Context — This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  return $\left(\bigoplus_{m \in S} \textit{source}[m] \right)$ to front end

The CM:global-logxor operation combines the *source* fields in all selected processors by performing bitwise logical EXCLUSIVE OR operations. A bit is 1 in the result field if the corresponding bit is a 1 in *an odd number* of the fields to be combined. The resulting combined field is then sent to the front-end computer as an unsigned integer and returned as the result of the operation. If there are no selected processors, then the value 0 is returned, representing a field of length *len* containing all zeros.

GLOBAL-F-MAX

One floating-point number is examined in every selected processor, and the largest of all these integers (that is, the one closest to $+\infty$) is returned to the front end as a floating-point number.

Formats result $\leftarrow$ CM:global-f-max-1L *source*, *s*, *e*

Operands *source* The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Result A floating-point number, the largest of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor equals the maximum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $-\infty$ to front end
else
  let $R = \left(\max_{m \in S} \textit{source}[m] \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ 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.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-S-MAX

One signed integer is examined in every selected processor, and the largest of all these integers (that is, the one closest to $+\infty$) is returned to the front end as a signed integer.

Formats result $\leftarrow$ CM:global-s-max-1L *source*, *len*

Operands	*source*	The field ID of the signed integer source field.
	len	The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
Result		A signed integer, the largest of the *source* fields.
Overlap		There are no constraints, because overlap is not possible.
Flags		*test-flag* is set if the value in a particular processor equals the maximum; otherwise it is cleared.
Context		This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \}$

If $|S| = 0$ then
  return -2^{len-1} to front end
else
  let $R = \left(\max_{m \in S} \textit{source}[m] \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-s-max operation returns the largest (that is, closest to $+\infty$) of the signed-integer *source* fields of all selected processors. This largest value is sent to the front-end computer as a signed integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value -2^{len-1} is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-U-MAX

One unsigned integer is examined in every selected processor, and the largest of all these integers is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-max-1L *source*, *len*

Operands

source The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the largest of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor equals the maximum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$

If $|S| = 0$ then
  return 0 to front end
else
  let $R = \left(\max_{m \in S} \textit{source}[m] \right)$
  For every virtual processor k in the *current-vp-set* do
    if *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-max operation returns the largest of the unsigned-integer *source* fields of all selected processors. This largest value is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value 0 is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-U-MAX-S-INTLEN

One signed integer is examined in every selected processor, and the largest *length* of all these integers is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-max-s-intlen-1L *source*, *len*

Operands

source The field ID of the signed integer source field.

len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the length of the *source* field value of greatest length.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor has a length equal to the maximum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  let $R = \left(\max_{m \in S} \left\lceil \log_2 \left(\frac{1}{2} + \left| \frac{1}{2} + \textit{source}[m] \right| \right) \right\rceil \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-u-max-s-intlen operation computes the integer-length of each signed integer *source* value. The largest length is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value 0 is returned.

A call to CM:global-u-max-s-intlen-1L is equivalent to the sequence

GLOBAL-MAX-INTLEN

CM:s-integer-length-2-2L *temp*, *source*, *len*, *len*
CM:global-u-max-1L *temp*, *len*

but may be faster.

GLOBAL-U-MAX-U-INTLEN

One unsigned integer is examined in every selected processor, and the largest *length* of all these integers is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-max-u-intlen-1L *source*, *len*

Operands	*source*	The field ID of the unsigned integer source field.
	len	The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result: An unsigned integer, the length of the *source* field value of greatest length.

Overlap: There are no constraints, because overlap is not possible.

Flags: *test-flag* is set if the value in a particular processor has a length equal to the maximum; otherwise it is cleared.

Context: This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return 0 to front end
else
  let $R = \left(\max_{m \in S} \lceil \log_2 (1 + \textit{source}[m]) \rceil \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-u-max-u-intlen operation computes the integer-length of each unsigned integer *source* value. The largest length is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value 0 is returned.

A call to CM:global-u-max-u-intlen-1L is equivalent to the sequence

GLOBAL-MAX-INTLEN

CM:u-integer-length-2-2L *temp*, *source*, *len*, *len*
CM:global-u-max-1L *temp*, *len*

but may be faster.

GLOBAL-F-MIN

One floating-point number is examined in every selected processor, and the smallest of all these integers (that is, the one closest to $-\infty$) is returned to the front end as a floating-point number.

Formats result ← CM:global-f-min-1L *source*, *s*, *e*

Operands	*source*	The field ID of the floating-point source field.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Result A floating-point number, the smallest of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor equals the minimum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \}$

If $|S| = 0$ then
  return $+\infty$ to front end
else
  let $R = \left(\min_{m \in S} \textit{source}[m]\right)$
  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 smallest (that is, closest to $-\infty$) of the floating-point *source* fields of all selected processors. This smallest value is sent to the front-end computer as a floating-point number and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value $+\infty$ is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-S-MIN

One signed integer is examined in every selected processor, and the smallest of all these integers (that is, the one closest to $-\infty$) is returned to the front end as a signed integer.

Formats result ← CM:global-s-min-1L *source*, *len*

Operands	*source*	The field ID of the signed integer source field.
	len	The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
Result	A signed integer, the smallest of the *source* fields.	
Overlap	There are no constraints, because overlap is not possible.	
Flags	*test-flag* is set if the value in a particular processor equals the minimum; otherwise it is cleared.	
Context	This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.	

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $2^{len-1} - 1$ to front end
else
  let $R = \left(\min_{m \in S} \textit{source}[m] \right)$ to front end
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-s-min operation returns the smallest (that is, closest to $-\infty$) of the signed-integer *source* fields of all selected processors. This smallest value is sent to the front-end computer as a signed integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value $2^{len-1} - 1$ is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

GLOBAL-U-MIN

One unsigned integer is examined in every selected processor, and the smallest of all these integers is returned to the front end as an unsigned integer.

Formats result ← CM:global-u-min-1L *source*, *len*

Operands *source* The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the smallest of the *source* fields.

Overlap There are no constraints, because overlap is not possible.

Flags *test-flag* is set if the value in a particular processor equals the minimum; otherwise it is cleared.

Context This operation is conditional. The result returned depends only upon processors whose *context-flag* is 1.

Definition Let $S = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \}$
If $|S| = 0$ then
  return $2^len - 1$ to front end
else
  let $R = \left(\min_{m \in S} \textit{source}[m] \right)$
  For every virtual processor k in the *current-vp-set* do
    if $\textit{context-flag}[k] = 1$ then
      if $\textit{source}[k] = R$ then
        $\textit{test-flag}[k] \leftarrow 1$
      else
        $\textit{test-flag}[k] \leftarrow 0$
  return R to front end

The CM:global-u-min operation returns the smallest (that is, closest to zero) of the unsigned-integer *source* fields of all selected processors. This smallest value is sent to the front-end computer as an unsigned integer and returned as the result of the operation. In addition, the *test-flag* is set in every selected processor whose field is equal to the finally computed value, and is cleared in all other selected processors. If there are no selected processors, then the value $2^{len} - 1$ is returned.

In the Lisp/Paris interface, this function returns two values; the second value is T if no processors are selected and nil if any processors are selected.

F-GT

Compares two floating-point source values. The *test-flag* is set if the first is strictly greater than the second, and otherwise is cleared.

Formats

CM:f-gt-1L	*source1*, *source2*, *s*, *e*
CM:f-gt-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-gt-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-gt-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do

```
if context-flag[k] = 1 then
   if source1[k] > source2[k]
      test-flag[k] ← 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). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-GT

Compares two signed integer source values. The *test-flag* is set if the first is strictly greater than the second, and otherwise is cleared.

Formats

CM:s-gt-1L	*source1*, *source2*, *len*
CM:s-gt-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-gt-constant-1L	*source1*, *source2-value*, *len*
CM:s-gt-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-gt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] > *source2*[*k*] then
      test-flag[*k*] ← 1
    else
      test-flag[*k*] ← 0

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is greater than the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly

required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-GT

Compares two unsigned integer source values. The *test-flag* is set if the first is strictly greater than the second, and otherwise is cleared.

Formats

CM:u-gt-1L	*source1*, *source2*, *len*
CM:u-gt-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-gt-constant-1L	*source1*, *source2-value*, *len*
CM:u-gt-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-gt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is greater than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] > *source2*[k] then
      test-flag[k] ← 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 greater than the second operand and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

F-IEEE-TO-VAX

Converts the floating-point source field values from IEEE floating-point format to VAX floating-point format and stores the result in the destination field.

Formats CM:f-ieee-to-vax-1L *vax-dest*, *ieee-source*, *len*

Operands

vax-dest The field ID of the floating-point destination field.

ieee-source The field ID of the floating-point source field.

len The length of the *vax-dest* and *ieee-source* fields. The value of *len* must be either 32 or 64.

Overlap The fields *vax-dest* and *ieee-source* may overlap in any manner.

Flags *overflow-flag* is set if the ieee-source cannot be represented in the destination field; otherwise it is cleared. If *ieee-source* represents ∞ or NaN, then *vax-dest* is set to the "undefined variable" value in VAX format and the *overflow-flag* is cleared. If *ieee-source* represents −0.0, it is converted to VAX 0.0 and the *overflow-flag* is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

The Connection Machine operates internally on floating point data in IEEE format whereas the VAX uses a VAX floating-point format. In each active processor, this function converts a floating-point field in standard IEEE format to a field in VAX format.

The value of *len* specifies the precision of *vax-dest*. If *len* is specified as 32, then VAX 'F' format is used. If *len* is specified as 64, then VAX 'D' format is used.

VAX and IEEE floating-point formats are incompatible, so there are a number of potential inaccuracies in the translation. In general, if the conversion is accurate then the overflow flag is cleared; if inaccurate, then the overflow flag is set. See the flags description above.

This instruction is useful for rapidly converting floating-point data to VAX format, even if a VAX front end is not being used. For example, if data is to be transferred from a file in the CM file system to a VAX, CM:f-ieee-to-vax-1L should be called before writing the data file.

All Paris CM to front end data transfer functions automatically convert the data to the appropriate front-end format so it is not necessary to call CM:ieee-to-vax before calling, for instance, one of the read-from-news-array instructions.

To convert data back to IEEE floating-point format, see the definition of CM:f-vax-to-ieee-1L.

INIT

For the C/Paris and Fortran/Paris interfaces only. Makes various machine parameters available and performs a warm boot operation.

Formats CM:init

Context This operation is unconditional. It does not depend on the *context-flag*.

The facility for initializing Connection Machine hardware is provided in different ways in the Lisp/Paris interface (on the one hand) and the C/Paris and Fortran/Paris interfaces (on the other hand).

In the Lisp/Paris interface, there is no CM:init operation. Part of the work done by CM:init is performed by CM:cold-boot, and the remainder by CM:warm-boot.

In the C/Paris and Fortran/Paris interfaces, CM:init makes available to the user program various machine parameters that are initialized by the cmattach and cmcoldboot shell commands. It also performs all the functions of CM:warm-boot.

Every C or Fortran program that uses Paris should call CM:init before invoking any other Paris operations.

INITIALIZE-RANDOM-GENERATOR

Formats CM:initialize-random-generator *seed*

Operands *seed* An unsigned integer immediate operand to be used as the seed value for initializing the pseudo-random number generator.

Context This operation is unconditional. It does not depend on the *context-flag*.

Explicitly initializes the pseudo-random generator of numbers used by the Paris random number generator operations CM:f-random-1L and cm:u-random-1L. The seed (a front-end integer, which must be non-zero) determines the initial state.

If it has not been explicitly initialized by a call to this operation, the Paris random number generator is automaticaly initialized the first time it is called. Automatic initialization uses a seed based on the date and time.

In the Lisp/Paris interface, the *seed* argument is optional; if it is omitted, then a value based on the date and time of day is used.

Note: Less simple but more flexible random number generation routines are provided as part of the CM Scientific Subroutines Library (CMSSL). For instance, the CMSSL random number generators may be checkpointed to guard against accidental interuptions.

S-INTEGER-LENGTH

The minimum number of bits, minus one, needed to represent a signed integer value is placed in the destination field.

Formats CM:s-integer-length-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the signed integer source field.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ 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 field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-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*.

INTERN-DETAILED-GEOMETRY

Returns an interned geometry given detailed information about how the grid is laid out.

Formats result ← CM:intern-detailed-geometry *axis-descriptor-array*, *[rank]*

Operands *axis-descriptor-array* A front-end vector of descriptors for the grid axes. In the C interface, the elements of the *axis-descriptor-array* must be of type CM_axis_descriptor_t, that is, they must be pointers to structures of type CM_axis_descriptor.

In the Lisp interface, the *axis-descriptor-array* may be either a list of descriptors or an array of descriptors.

rank An unsigned integer, the rank (number of dimensions) of the *axis-descriptor-array*. This must be in between 1 and CM:*max-geometry-rank*, inclusive. This argument is not provided when calling Paris from Lisp.

Result A geometry ID, identifying the existing or newly created interned geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

By using interned geometries, modules that require identical geometries can use identical geometries – without having to keep track of the geometryID's.

CM:intern-detailed-geometry takes an array of descriptors. Each descriptor describes one NEWS axis in some detail. Most of the components are unsigned integers, but the value of the *ordering* component must be either :news-order or :send-order. The CM:create-detailed-geometry dictionary entry defines the type of the ordering component and of the descriptor for each language interface.

CM:intern-detailed-geometry is identical to CM:create-detailed-geometry with this exception: it returns an *interned* geometryID. A list of interned geometries is maintained and whenever CM:intern-detailed-geometry or intern-geometry is called, a previously interned geometry is returned if one exists that matches the specifications of the call, otherwise a new geometry is created and added to the list.

An interned geometryID is a geometryID returned by CM:intern-detailed-geometry or by CM:intern-geometry; a geometryID returned by CM:create-detailed-geometry or by CM:create-geometry may *not* be interned.

CM:create-detailed-geometry returns a unique, uninterned geometryID each time it is called. In contrast, CM:intern-detailed-geometry returns an existing interned geometryID if it can. If there is an interned geometry with an axis descriptor array that matches the supplied *axis-descriptor-array*, it is returned. Otherwise, CM:intern-detailed-geometry returns a new interned geometryID. The returned geometryID may be used to create a VP set or to respecify the geometry of an existing VP set.

Once the interned geometry has been created, the user may destroy the array created to provide the dimension information. All necessary information is copied from this array when the geometry is created.

INTERN-GEOMETRY

Returns an interned geometry given grid axis lengths.

Formats result ← CM:intern-geometry *dimension-array*; *[rank]*

Operands *dimension-array* A front-end vector of unsigned integer lengths of the grid axes. In the Lisp interface, this may be a list of dimension lengths instead of an array of dimension lengths, at the user's option.

rank An unsigned integer, the rank (number of dimensions) of the *dimension-array*. This must be in between 1 and CM:*max-geometry-rank*, inclusive. This argument is not provided when calling Paris from Lisp.

Result A geometry ID, identifying the existing or newly created interned geometry.

Context This operation is unconditional. It does not depend on the *context-flag*.

By using interned geometries, codes that require identical geometries can use identical geometries – without having to keep track of the geometryID's.

CM:intern-geometry is identical to CM:create-geometry with this exception: it returns an *interned* geometryID. An interned geometryID is a geometryID returned by CM:intern-geometry or by CM:intern-detailed-geometry; a geometryID returned by CM:create-geometry or by CM:create-detailed-geometry may *not* be interned.

CM:create-geometry returns a unique, uninterned geometryID each time it is called. In contrast, CM:intern-geometry returns an existing interned geometryID if it can. If there is a geometry, created by CM:intern-geometry and with dimensions that match those specified in *dimension-array*, it is returned. Otherwise, CM:intern-geometry returns a new interned geometryID. The returned geometryID may be used to create a VP set or to respecify the geometry of an existing VP set.

The *dimension-array* must be a one-dimensional array of nonnegative integers; each must be a power of two. The product of all these integers must be a multiple of the number of physical processors attached for use by this process.

The geometry is laid out so as to optimize performance under the assumption that the axes are used equally frequently for NEWS communication. The operations CM:create-detailed-geometry or CM:intern-detailed-geometry may be used instead to more precisely control layout for performance tuning.

Once the interned geometry has been created, the user may destroy the array used to provide the dimension information. All necessary information is copied out of this array when the geometry is created.

INTERN-IDENTICAL-VP-SET

Returns an interned VP set, within which fields may be allocated.

Formats result ← CM:intern-identical-vp-set *geometry-id*

Operands *geometry-id* A geometry ID.

Result A VP set ID, identifying the existing or newly allocated interned VP set.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation returns a VP set ID for an *interned* VP set. An interned VP set is a VP set referenced by a VP set ID returned by CM:intern-identical-vp-set. VP set interning allows different modules to reference identical VP sets and reduces VP set memory management overhead.

CM:intern-identical-vp-set returns an existing, interned VP set ID if there is an existing, interned VP set whose geometry is identical to the geometry specified by *geometry-id*. Otherwise, CM:intern-identical-vp-set returns a new, interned VP set ID.

Once a VP set has been created as interned, it may never be uninterned. Similarly, an uninterned VP set (created for instance with CM:create-vp-set) may never become interned.

An interned VP set may be used in the same ways as an uninterned VP set. For instance, it may be given to other Paris operations in order to create memory fields in which data may be stored. It may also be deallocated with CM:deallocate-vp-set.

INVERT-CONTEXT

Unconditionally makes all active processors inactive and vice versa.

Formats CM:invert-context

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$context\text{-}flag[k] \leftarrow \neg context\text{-}flag[k]$

Within each processor, the context bit for that processor is unconditionally inverted.

INVERT-flag

Inverts a specified flag bit.

Formats CM:invert-test
CM:invert-test-always
CM:invert-overflow
CM:invert-overflow-always

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
if (always or *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-AN-ALIAS

Returns true if the specified field ID is an alias field ID, false otherwise.

Formats result ← CM:is-field-an-alias *field-id*

Operands *field-id* A field ID.

Result True if *field-id* is an alias field ID, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation tests whether the provided field ID is an alias field ID created with CM:make-field-alias, as opposed to a regular field ID created with a field allocation instruction such as CM:allocate-stack-field.

IS-FIELD-IN-HEAP

Returns true if the specified field is a heap field, false otherwise.

Formats result ← CM:is-field-in-heap *field-id*

Operands *field-id* A field ID.

Result True if the fieldID indicates a field allocated in the heap, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This instruction allows a program to test whether a given field has been allocated in the heap (as opposed to the stack).

IS-FIELD-IN-STACK

Returns true if the specified field is a stack field, false otherwise.

Formats result ← CM:is-field-in-stack *field-id*

Operands	*field-id*	A field ID.
Result	True if the fieldID indicates a field allocated on the stack, and false otherwise.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

This instruction allows a program to test whether a given field has been allocated on the stack (as opposed to the heap).

IS-FIELD-VALID

Returns true if the specified field ID corresponds to a currently allocated CM field ID, false otherwise.

Formats result ← CM:is-field-valid *field-id*

Operands	*field* ID A field ID.
Result	True if *field-id* is a valid field ID, and false otherwise.
Context	This operation is unconditional. It does not depend on the *context-flag*.

This instruction allows a program to test whether the provided field ID is valid. Valid field ID's are assigned and returned by operations such as CM:allocate-stack-field, CM:allocate-heap-field, CM:add-offset-to-field-id, and CM:make-field-alias.

IS-STACK-FIELD-NEWER

Formats result ← CM:is-stack-field-newer *stack-query-field*, *stack-base-field*

Operands *stack-query-field* A field ID. The field must be in the stack.

stack-base-field A field ID. The field must be in the stack.

Result True if the *stack-query-field* has been allocated more recently than the *stack-base-field*, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation compares two stack fields and returns true if the second has been allocated more recently than the first.

IS-VP-SET-VALID

Returns true if the specified VP set ID corresponds to a currently allocated VP set, false otherwise.

Formats result ← CM:is-vp-set-valid *vp-set*

Operands *field* ID A VP set ID.

Result True if *vp-set-id* is a valid VP set ID, and false otherwise.

Context This operation is unconditional. It does not depend on the *context-flag*.

This instruction allows a program to test whether the provided VP set ID is valid. Valid VP set ID's are assigned and returned by CM:allocate-vp-set.

S-ISQRT

The integer square root of a signed integer source field is placed in the destination field. This is the largest integer not larger than the true mathematical square root.

Formats

CM:s-isqrt-1-1L *dest/source, len*
CM:s-isqrt-2-1L *dest, source, len*
CM:s-isqrt-2-2L *dest, source, dlen, slen*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *test-flag* is set if the *source* value is negative; otherwise it is cleared.

overflow-flag is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:s-isqrt-2-2L.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if *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] $\leftarrow$ ⟨unpredictable⟩
      test-flag[k] $\leftarrow$ 1
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] $\leftarrow$ 1
    else *overflow-flag*[k] $\leftarrow$ 0
  as appropriate.

ISQRT

If the *source* value is non-negative, then the integer square root of that value (the largest integer not greater than the mathematical square root) is placed in the destination, and *test-flag* is cleared. Otherwise the *test-flag* is set and an unpredictable value is placed in the *dest* field.

U-ISQRT

The integer square root of an unsigned integer source field is placed in the destination field. This is the largest integer not larger than the true mathematical square root.

Formats

CM:u-isqrt-1-1L *dest/source*, *len*
CM:u-isqrt-2-1L *dest*, *source*, *len*
CM:u-isqrt-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest — The field ID of the unsigned integer destination field.

source — The field ID of the unsigned integer source field.

len — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen — The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen — The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:u-isqrt-2-2L.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \lfloor\sqrt{source}\rfloor$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    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 field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition Let $g = geometry(current\text{-}vp\text{-}set)$
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 $0 \leq m < n$ 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 field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-le-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] ≤ *source2*[*k*]
      test-flag[*k*] ← 1
    else
      test-flag[*k*] ← 0

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is less than or equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; +0 and −0 are considered to be equal.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-LE

Compares two signed integer source values. The *test-flag* is set if the first is less than or equal to the second, and otherwise is cleared.

Formats

CM:s-le-1L	*source1*, *source2*, *len*
CM:s-le-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-le-constant-1L	*source1*, *source2-value*, *len*
CM:s-le-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-le-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $source1[k] \leq source2[k]$ then
      $test\text{-}flag[k] \leftarrow 1$
    else
      $test\text{-}flag[k] \leftarrow 0$

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is less than or equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly

required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-LE

Compares two unsigned integer source values. The *test-flag* is set if the first is less than or equal to the second, and otherwise is cleared.

Formats

CM:u-le-1L	*source1*, *source2*, *len*
CM:u-le-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-le-constant-1L	*source1*, *source2-value*, *len*
CM:u-le-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-le-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than or equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] $\leq$ *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. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-LN

The natural logarithm of the complex source field values is placed in the complex destination field.

Formats CM:c-ln-1-1L *dest/source*, *s*, *e*
CM:c-ln-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then
dest[*k*] ← ln *source*[*k*]

The value $\ln s$ is stored into the *dest* field, where *s* is the value of the *source* field. This is the natural logarithm to the base $e \approx 2.718281828\ldots$.

F-LN

The natural logarithm of the floating-point source field values are placed in the floating-point destination field.

Formats CM:f-ln-1-1L *dest/source*, *s*, *e*
CM:f-ln-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is non-positive; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    dest[*k*] ← ln *source*[*k*]
    if *source*[*k*] < 0 then
      test[*k*] ← 1
    else *test*[*k*] ← 0

Call the value of the *source* field *s*. The value $\ln s$ is stored into the *dest* field; this is the natural logarithm to the base $e \approx 2.718281828\ldots$

LOAD-CONTEXT

Unconditionally reads a bit from memory and loads it into the context bit.

Formats CM:load-context *source*

Operands *source* The field ID of the source bit (a one-bit field).

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$context\text{-}flag[k] \leftarrow source[k]$

Within each processor, a bit is read from memory and unconditionally loaded into the context bit for that processor.

LOAD-flag

Reads a bit from memory and loads it into a flag.

Formats

CM:load-test	*source*
CM:load-test-always	*source*
CM:load-overflow	*source*
CM:load-overflow-always	*source*

Operands *source* The field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    flag$[k] \leftarrow$ *source*$[k]$

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, a bit is read from memory and loaded into the indicated flag for that processor.

F-LOG2

The base two logarithm of the floating-point source field is placed in the floating-point destination field.

Formats CM:f-log2-1-1L *dest/source*, *s*, *e*
CM:f-log2-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \log_2$ *source*$[k]$

The value $\log_2 s$ is stored into the *dest* field, where s is the value of the *source* field. This is the logarithm to the base two of the floating-point source field.

F-LOG10

The base ten logarithm of the floating-point source field is placed in the floating-point destination field.

Formats CM:f-log10-1-1L *dest/source*, *s*, *e*
CM:f-log10-2-1L *dest*, *source*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags: *test-flag* is set if the *source* is zero; otherwise it is cleared.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then
dest[*k*] $\leftarrow \log_{10}$ *source*[*k*]

The value $\log_{10} s$ is stored into the *dest* field, where *s* is the value of the *source* field. This is the logarithm to the base ten of the floating-point source field.

LOGAND

Combines two source values using a bitwise logical AND operation, and places the result in the destination field.

Formats

CM:logand-2-1L	*dest/source1, source2, len*
CM:logand-always-2-1L	*dest/source1, source2, len*
CM:logand-constant-2-1L	*dest/source1, source2-value, len*
CM:logand-const-always-2-1L	*dest/source1, source2-value, len*
CM:logand-3-1L	*dest, source1, source2, len*
CM:logand-always-3-1L	*dest, source1, source2, len*
CM:logand-constant-3-1L	*dest, source1, source2-value, len*
CM:logand-const-always-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $\textit{context-flag}[k] = 1$) then
$\textit{dest}[k] \leftarrow \textit{source1}[k] \land \textit{source2}[k]$

Each bit of the *dest* field is set if both of the corresponding bits of the *source1* and *source2* fields are 1, and is cleared if either of the corresponding bits of the *source1* and *source2* fields is 0.

LOGAND-CONTEXT

Reads a bit from memory; if it is zero, the context bit is cleared, unconditionally.

Formats CM:logand-context *source*

Operands *source* The field ID of the source bit (a one-bit field).

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do

$$\textit{context-flag}[k] \leftarrow \textit{context-flag}[k] \wedge \textit{source}[k]$$

Within each processor, a bit is read from memory and is "anded" into the context bit for that processor.

LOGAND-CONTEXT-WITH-TEST

If the test flag is zero, the context bit is cleared.

Formats CM:logand-context-with-test

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do

$$\textit{context-flag}[k] \leftarrow \textit{context-flag}[k] \wedge \textit{test-flag}[k]$$

Within each processor, the test flag is "anded" into the context bit for that processor.

LOGAND-flag

Reads a bit from memory; if it is zero, a specified flag is cleared.

Formats

CM:logand-test	*source*
CM:logand-test-always	*source*
CM:logand-overflow	*source*
CM:logand-overflow-always	*source*

Operands *source* The field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
if (always or *context-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 using a bitwise logical AND operation. Places the result in the destination field.

Formats

CM:logandc1-2-1L	*dest/source1*, *source2*, *len*
CM:logandc1-always-2-1L	*dest/source1*, *source2*, *len*
CM:logandc1-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logandc1-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logandc1-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc1-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc1-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logandc1-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition

For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow (\neg source1[k]) \wedge source2[k]$

Each bit of the *dest* field is set if the corresponding bit of the *source1* field is 0 and the corresponding bit of the *source2* field is 1; otherwise it is cleared.

LOGANDC2

Combines the first source and the bitwise logical NOT of the second source using a bitwise logical AND operation. Places the result in the destination field.

Formats

CM:logandc2-2-1L	*dest/source1*, *source2*, *len*
CM:logandc2-always-2-1L	*dest/source1*, *source2*, *len*
CM:logandc2-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logandc2-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logandc2-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc2-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logandc2-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logandc2-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow source1[k] \wedge (\neg source2[k])$

Each bit of the *dest* field is set if the corresponding bit of the *source1* field is 1 and the corresponding bit of the *source2* field is 0; otherwise it is cleared.

S-LOGCOUNT

The destination field receives a count of the number of bits that differ from the sign bit in a two's-complement binary representation of a signed integer source value. For nonnegative values, this is a count of 1 bits.

Formats CM:s-logcount-2-2L *dest*, *source*, *dlen*, *slen*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the signed integer source field.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source*$[k] \geq 0$ then *dest*$[k] \leftarrow$ *count-of-one-bits*(*source*$[k]$)
    else *dest*$[k] \leftarrow$ *count-of-one-bits*($\neg$*source*$[k]$)
    if $\langle$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 field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-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 using a bitwise logical EQUIVALENCE operation, and places the result in the destination field.

Formats

CM:logeqv-2-1L	*dest/source1, source2, len*
CM:logeqv-always-2-1L	*dest/source1, source2, len*
CM:logeqv-constant-2-1L	*dest/source1, source2-value, len*
CM:logeqv-const-always-2-1L	*dest/source1, source2-value, len*
CM:logeqv-3-1L	*dest, source1, source2, len*
CM:logeqv-always-3-1L	*dest, source1, source2, len*
CM:logeqv-constant-3-1L	*dest, source1, source2-value, len*
CM:logeqv-const-always-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow \neg(source1[k] \oplus source2[k])$

Each bit of the *dest* field is set where corresponding bits of the *source1* and *source2* fields are alike, and is cleared where corresponding bits of the *source1* and *source2* fields differ.

LOGIOR

Combines two source values using a bitwise logical inclusive OR operation, and places the result in the destination field.

Formats

CM:logior-2-1L	*dest/source1*, *source2*, *len*
CM:logior-always-2-1L	*dest/source1*, *source2*, *len*
CM:logior-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logior-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logior-3-1L	*dest*, *source1*, *source2*, *len*
CM:logior-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logior-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logior-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    $\textit{dest}[k] \leftarrow \textit{source1}[k] \vee \textit{source2}[k]$

Each bit of the *dest* field is set if either of the corresponding bits of the *source1* and *source2* fields is 1, and is cleared if both of the corresponding bits of the *source1* and *source2* fields are 0.

LOGIOR-CONTEXT

Reads a bit from memory; if it is one, the context bit is set, unconditionally.

Formats CM:logior-context *source*

Operands *source* The field ID of the source bit (a one-bit field).

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do

$$\textit{context-flag}[k] \leftarrow \textit{context-flag}[k] \vee \textit{source}[k]$$

Within each processor, a bit is read from memory and is "ored" into the context bit for that processor.

LOGIOR-flag

Reads a bit from memory; if it is 1, a specified flag is set.

Formats

CM:logior-test	*source*
CM:logior-test-always	*source*
CM:logior-overflow	*source*
CM:logior-overflow-always	*source*

Operands *source* The field ID of the source bit (a one-bit field).

Context The non-always operations are conditional.

The always operations are unconditional.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$flag[k] \leftarrow flag[k] \lor source[k]$
where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, a bit is read from memory and is "ored" into the indicated flag for that processor.

LOGNAND

Combines two source values with a bitwise logical NAND operation, and places the result in the destination field.

Formats

CM:lognand-2-1L	*dest/source1*, *source2*, *len*
CM:lognand-always-2-1L	*dest/source1*, *source2*, *len*
CM:lognand-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:lognand-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:lognand-3-1L	*dest*, *source1*, *source2*, *len*
CM:lognand-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:lognand-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:lognand-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow \neg(source1[k] \wedge source2[k])$

Each bit of the *dest* field is set if either of the corresponding bits of the *source1* and *source2* fields is 0, and is cleared if both of the corresponding bits of the *source1* and *source2* fields are 1.

LOGNOR

Combines two source values with a bitwise logical NOR operation, and places the result in the destination field.

Formats

CM:lognor-2-1L	*dest*/*source1*, *source2*, *len*
CM:lognor-always-2-1L	*dest*/*source1*, *source2*, *len*
CM:lognor-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:lognor-const-always-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:lognor-3-1L	*dest*, *source1*, *source2*, *len*
CM:lognor-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:lognor-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:lognor-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context

The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition

For every virtual processor k in the *current-vp-set* do
 if (always or *context-flag*[k] = 1) then
  $dest[k] \leftarrow \neg(source1[k] \vee source2[k])$

Each bit of the *dest* field is set if both of the corresponding bits of the *source1* and *source2* fields are 0, and is cleared if either of the corresponding bits of the *source1* and *source2* fields is 1.

LOGNOT

Copies a source field, inverts all the bits, and places them in the destination field.

Formats

CM:lognot-1-1L	*dest/source*, *len*
CM:lognot-always-1-1L	*dest/source*, *len*
CM:lognot-always-2-1L	*dest*, *source*, *len*
CM:lognot-2-1L	*dest*, *source*, *len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or *context-flag*[k] = 1) then
dest[k] $\leftarrow \neg$*source*[k]

Each bit of the *dest* field is set to the inverse of the corresponding bit of the *source* field.

LOGORC1

Combines the second source and the bitwise logical NOT of the first source using a bitwise logical inclusive OR operation. Places the result in the destination field.

Formats

CM:logorc1-2-1L	*dest*/*source1*, *source2*, *len*
CM:logorc1-always-2-1L	*dest*/*source1*, *source2*, *len*
CM:logorc1-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:logorc1-const-always-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:logorc1-3-1L	*dest*, *source1*, *source2*, *len*
CM:logorc1-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logorc1-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logorc1-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if (always or $context\text{-}flag[k] = 1$) then
$dest[k] \leftarrow (\neg source1[k]) \lor source2[k]$

Each bit of the *dest* field is cleared if the corresponding bit of the *source1* field is 1 and if the corresponding bit of the *source2* field is 0; otherwise it is set.

LOGORC2

Combines the first source and the bitwise logical NOT of the second source using a bitwise logical inclusive OR operation. Places the result in the destination field.

Formats

CM:logorc2-2-1L	*dest/source1*, *source2*, *len*
CM:logorc2-always-2-1L	*dest/source1*, *source2*, *len*
CM:logorc2-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logorc2-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logorc2-3-1L	*dest*, *source1*, *source2*, *len*
CM:logorc2-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logorc2-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logorc2-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

if (always or $\mathit{context\text{-}flag}[k] = 1$) then

$\mathit{dest}[k] \leftarrow \mathit{source1}[k] \vee (\neg \mathit{source2}[k])$

Each bit of the *dest* field is cleared if the corresponding bit of the *source1* field is 0 and if the corresponding bit of the *source2* field is 1; otherwise it is set.

LOGXOR

Combines two source values using a bitwise logical exclusive OR operation, and places the result in the destination field.

Formats

CM:logxor-2-1L	*dest/source1*, *source2*, *len*
CM:logxor-always-2-1L	*dest/source1*, *source2*, *len*
CM:logxor-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:logxor-const-always-2-1L	*dest/source1*, *source2-value*, *len*
CM:logxor-3-1L	*dest*, *source1*, *source2*, *len*
CM:logxor-always-3-1L	*dest*, *source1*, *source2*, *len*
CM:logxor-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:logxor-const-always-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the destination field.

source1 The field ID of the first source field.

source2 The field ID of the second source field.

source2-value An unsigned integer immediate operand to be regarded as a vector of bits and used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    $\textit{dest}[k] \leftarrow \textit{source1}[k] \oplus \textit{source2}[k]$

Each bit of the *dest* field is set where corresponding bits of the *source1* and *source2* fields differ, and is cleared where corresponding bits of the *source1* and *source2* fields are alike.

F-LT

Compares two floating-point source values. The *test-flag* is set if the first is strictly less than the second, and otherwise is cleared.

Formats

CM:f-lt-1L *source1*, *source2*, *s*, *e*
CM:f-lt-constant-1L *source1*, *source2-value*, *s*, *e*
CM:f-lt-zero-1L *source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-lt-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] < *source2*[*k*]
      test-flag[*k*] ← 1
    else
      test-flag[*k*] ← 0

Two operands are compared as floating-point numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is less than the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; -0 is not less than $+0$.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-LT

Compares two signed integer source values. The *test-flag* is set if the first is strictly less than the second, and otherwise is cleared.

Formats

CM:s-lt-1L	*source1*, *source2*, *len*
CM:s-lt-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-lt-constant-1L	*source1*, *source2-value*, *len*
CM:s-lt-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-lt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *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 less than the second operand, and is cleared otherwise.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly

required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-LT

Compares two unsigned integer source values. The *test-flag* is set if the first is strictly less than the second, and otherwise is cleared.

Formats

CM:u-lt-1L	*source1*, *source2*, *len*
CM:u-lt-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-lt-constant-1L	*source1*, *source2-value*, *len*
CM:u-lt-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-lt-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is less than *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] < *source2*[k] then
      test-flag[k] ← 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. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

MAKE-FIELD-ALIAS

Creates a new field ID that points to an existing field.

Formats result ← CM:make-field-alias *field-id*

Operands *field-id* A field ID. This must be a field ID returned by CM:allocate-stack-field or CM:allocate-heap-field; it may *not* be an offset into a field. The field need not be in the current VP set.

Result A field ID, identifying the alias field ID. This ID initially resides in the current VP set.

Context This operation is unconditional. It does not depend on the *context-flag*.

The return value is a *field alias*. It is a new field ID that identifies the same area of memory as does *field-id*.

The field identified by *field-id* can be in a VP set other than the current VP set. The returned alias field ID initially resides in the current VP set. The alias field ID can be used in all the same ways as a regular field ID can, with the following exceptions:

- It cannot be passed to CM:deallocate-heap-field.
- It cannot be passed to CM:deallocate-stack-through.

Associated with a field alias is a *physical length*: the number of bits that the field occupies in each physical processor. Also associated with a field alias is a *field length*: the number of bits the field occupies in each virtual processor. The physical length is equal to the field length multiplied by the VP ratio of the current VP set. It is an error if the physical length is not exactly divisible by the VP ratio of the current VP set.

It is possible for the field length of an alias field to be different from the field length of the original field. This is the case when make-field-alias is called on a field in a VP set that has a VP ratio different from the VP ratio of the current VP set. Suppose, for example, the current VP ratio is 32. If we make an alias for a 32-bit field that resides in a VP set with a VP ratio of 1, the resulting alias field is a 1 bit field (in a VP ratio of 32).

MAKE-NEWS-COORDINATE

Determine the send-address of a processor with the specified NEWS coordinate.

Formats CM:make-news-coordinate-1L *geometry*, *dest*, *axis*, *news-coordinate*, *slen*

Operands

geometry A geometry ID. This determines the NEWS dimensions to be used.

dest The field ID of the unsigned integer destination, to receive the send address of the processor whose coordinate along the specified axis is *news-coordinate* and whose coordinate along all other axes is a zero field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

news-coordinate The field ID of the unsigned integer NEWS coordinate along the specified axis field.

slen The length of the *news-coordinate* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow make\text{-}news\text{-}coordinate(axis, news\text{-}coordinate)$

where *make-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates zero.

FE-MAKE-NEWS-COORDINATE

Calculates, entirely on the front end, the send-address of the processor with the specified coordinate along the specified NEWS axis and with all other coordinates zero.

Formats result ← CM:fe-make-news-coordinate *geometry*, *axis*, *news-coordinate*

Operands *geometry* A geometry ID. This determines the NEWS dimensions to be used.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

news-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along the specified axis.

Result An unsigned integer, the send address of the processor whose coordinate along the specified axis is *news-coordinate* and whose coordinate along all other axes is zero.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Return *make-news-coordinate*(*axis*, *news-coordinate*)

where *make-news-coordinate* is as defined on page 40.

This function calculates, entirely on the front end, the send-address of a processor that has a specified coordinate along a specified NEWS axis, with all other coordinates zero.

C-MATRIX-MULTIPLY

Computes matrix multiplication using three single-precision complex operands and stores the result in the last.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix c- to signify that single-precision complex operands are used. A more efficient version of this operation is included in the CM Scientific Subroutines Library.

Formats CMSSL:c-matrix-multiply *source1*, *source2*, *dest*/*source3*

Operands	*dest*	The field ID of the complex destination field.
	source1	The field ID of the complex first source field.
	source2	The field ID of the complex second source field.
	source3	The field ID of the complex third source field.
Overlap	The fields *source1*, *source2*, and *dest/source3* must not overlap in any manner.	
Context	This operation is unconditional. It does not depend on the *context-flag*.	

The calculation $dest \leftarrow source3 + source1 \times source2$ is performed on three conforming matrices, represented as CM fields.

The operands *source1*, *source2*, and *dest/source3* must be fields of 64-bit single-precision complex values whose real and imaginary parts are 32-bit floating-point values.

All three operands may belong to separate VP sets if the geometries of those VP sets obey the following rule:

- The *source1* dimensions are $n \times m$
- The *source2* dimensions are $m \times p$
- The *dest/source3* dimensions are $n \times p$

where n, m, and p are each powers of two. Otherwise, all three operands must belong to the same square VP set.

The matrix multiply is performed using Cannon's systolic algorithm, which can be summarized in three steps:

1. The *source1* and *source2* matrices are aligned so the elements in each processor have conforming indices for matrix multiplication. In terms of data motion, this implies aligning the diagonal entries of the *source1* matrix to the first column and aligning the diagonal entries of the *source2* matrix to the first row.

2. The systolic part of the algorithm involves local multiplication of *source1* and *source2* elements followed by nearest neighbor data moves that simulate the inner product.

3. The *source1* and *source2* matrices are aligned back to the original form supplied by the calling program.

In order to exploit the full potential of the floating-point hardware, a block version of the algorithm is implemented. See the Thinking Machines technical report entitled "Matrix Multiplication on the Connection Machine" for details.

The CM matrix multiplication operation performs best for square matrices and at high VP ratios.

C/Paris code that calls the Paris matrix multiplication routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris matrix multiplication routine.

Fortran/Paris code should include the line

```
INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris matrix multiplication routine.

S-MATRIX-MULTIPLY

Computes matrix multiplication using three single-precision floating-point operands and stores the result in the last.

Note: For historical reasons, this operation uses the prefix CMSSL: in place of the standard CM: Paris instruction prefix. It also uses the prefix s- to signify that single-precision floating-point operands are used. A more efficient version of this operation is included in the CM Scientific Subroutines Library.

Formats CMSSL:s-matrix-multiply *source1*, *source2*, *dest/source3*

Operands		
	dest	The field ID of the floating-point destination field.
	source1	The field ID of the floating-point first source field.
	source2	The field ID of the floating-point second source field.
	source3	The field ID of the floating-point third source field.

Overlap The fields *source1*, *source2*, and *dest/source3* must not overlap in any manner.

Context This operation is unconditional. It does not depend on the *context-flag*.

The calculation *dest* $\leftarrow$ *source3* + *source1* $\times$ *source2* is performed on three conforming matrices, represented as CM fields.

The operands *source1*, *source2*, and *dest/source3* must be fields of 32-bit single-precision floating-point values.

All three operands may belong to separate VP sets if the geometries of those VP sets obey the following rule:

- The *source1* dimensions are $n \times m$
- The *source2* dimensions are $m \times p$
- The *dest/source3* dimensions are $n \times p$

where n, m, and p are each powers of two. Otherwise, all three operands must belong to the same square VP set.

The matrix multiply is performed using Cannon's systolic algorithm, which can be summarized in three steps:

1. The *source1* and *source2* matrices are aligned so the elements in each processor have conforming indices for matrix multiplication. In terms of data motion, this implies aligning the diagonal entries of the *source1* matrix to the first column and aligning the diagonal entries of the *source2* matrix to the first row.

2. The systolic part of the algorithm involves local multiplication of *source1* and *source2* elements, followed by nearest neighbor data moves that simulate the inner product.

3. The *source1* and *source2* matrices are aligned back to the original form supplied by the calling program.

In order to exploit the full potential of the floating-point hardware, a block version of the algorithm is implemented. See the Thinking Machines technical report entitled "Matrix Multiplication on the Connection Machine" for details.

The CM matrix multiplication routine performs best for square matrices and at high VP ratios.

C/Paris code that calls the Paris matrix multiplication routine must include the line

```
#include <cm/cmtypes.h>
```

at the top of the main program file. This declares all C/Paris functions and symbolic constants, including those for the Paris matrix multiplication routine.

Fortran/Paris code should include the line

```
  INCLUDE '/usr/include/cm/cmssl-paris-fort.h'
```

at the top of any program unit that calls the Paris matrix multiplication routine.

F-MAX

Two floating-point values are compared. The larger is placed in the destination field.

Formats

CM:f-max-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-max-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-max-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-max-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

    if context-flag[k] = 1 then
        if source1[k] ≥ 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). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-MAX

Two signed integer values are compared. The larger (the one closer to $+\infty$) is placed in the destination field.

Formats

CM:s-max-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-max-2-1L	*dest/source1*, *source2*, *len*
CM:s-max-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-max-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:s-max-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-max-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-max-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-max-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

test-flag is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context

This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

```
if context-flag[k] = 1 then
  if source1[k] ≥ 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 signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The larger of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len.*

U-MAX

Two unsigned integer values are compared. The larger is placed in the destination field.

Formats

CM:u-max-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-max-2-1L	*dest/source1, source2, len*
CM:u-max-3-1L	*dest, source1, source2, len*
CM:u-max-constant-2-1L	*dest/source1, source2-value, len*
CM:u-max-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-max-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-max-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-max-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
if $source1[k] \geq source2[k]$ then
$dest[k] \leftarrow source1[k]$
$test\text{-}flag[k] \leftarrow 0$
else
$dest[k] \leftarrow source2[k]$
$test\text{-}flag[k] \leftarrow 1$

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The larger of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

F-MIN

Two floating-point values are compared. The smaller is placed in the destination field.

Formats

CM:f-min-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-min-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-min-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-min-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source1 — The field ID of the floating-point first source field.

source2 — The field ID of the floating-point second source field.

source2-value — A floating-point immediate operand to be used as the second source.

s, *e* — The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap — The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags — *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor *k* in the *current-vp-set* do
  if *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). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-MIN

Two signed integer values are compared. The smaller (the one closer to $-\infty$) is placed in the destination field.

Formats

CM:s-min-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-min-2-1L	*dest/source1*, *source2*, *len*
CM:s-min-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-min-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:s-min-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest — The field ID of the signed integer destination field.

source1 — The field ID of the signed integer first source field.

source2 — The field ID of the signed integer second source field.

source2-value — A signed integer immediate operand to be used as the second source.

len — The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen — For CM:s-min-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 — For CM:s-min-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 — For CM:s-min-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap — The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags — *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] $\leq$ *source2*[k] then
      dest[k] $\leftarrow$ *source1*[k]
      test-flag[k] $\leftarrow$ 0
    else
      dest[k] $\leftarrow$ *source2*[k]
      test-flag[k] $\leftarrow$ 1

Two operands are compared as signed integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The smaller of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-MIN

Two unsigned integer values are compared. The smaller is placed in the destination field.

Formats

CM:u-min-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-min-2-1L	*dest*/*source1*, *source2*, *len*
CM:u-min-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-min-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:u-min-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-min-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-min-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-min-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the value placed in the *dest* field is not equal to *source1*; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $source1[k] \le source2[k]$ 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 smaller of the two values is copied to the *dest* field. The *test-flag* is set or cleared to indicate which operand was copied; if the two source operands are equal, then the *test-flag* is cleared.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

F-MOD

One floating-point source field is divided by another and the residue is placed in the destination field. Overflow is also computed.

This operation's name is derived from the term modulus; the destination field receives the the residue of taking one source field *modulus* another source field.

Formats

CM:f-mod-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-mod-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-mod-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-mod-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field. This is the quotient.

source1 The field ID of the floating-point first source field. This is the dividend.

source2 The field ID of the floating-point second source field. This is the divisor.

source2-value A floating-point immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if division by zero occurs; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source2*[k] = 0 then
      dest[k] ← ⟨unpredictable⟩

$test\text{-}flag[k] \leftarrow 1$
else
$$dest[k] \leftarrow source1[k] - source2[k] \times \left\lfloor \frac{source1[k]}{source2[k]} \right\rfloor$$
$test\text{-}flag[k] \leftarrow 0$
if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The residue resulting from the reduction of the floating-point *source1* operand divided by the *source2* operand is stored in the *dest* field. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-MOD

One signed integer source field is divided by another and the residue is placed in the destination field. Overflow is also computed.

This operation's name is derived from the term modulus; the destination field receives the the residue of taking one source field *modulus* another source field.

Formats

CM:s-mod-2-1L	*dest/source1, source2, len*
CM:s-mod-3-1L	*dest, source1, source2, len*
CM:s-mod-constant-2-1L	*dest/source1, source2-value, len*
CM:s-mod-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer residue field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer modulus (divisor) field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the modulus (divisor) is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *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 ⟨divisor was zero in processor k⟩ then *test-flag*$[k] \leftarrow 1$
    else *test-flag*$[k] \leftarrow 0$

The residue resulting from the reduction of the signed integer *source1* modulo the signed integer *source2* operand is stored into the *dest* field. The result always has the same sign as the *source2* operand. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

If the divisor is zero occurs, then the *test-flag* is set and the value of the destination is unpredictable

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-MOD

One unsigned integer source field is divided by another and the residue is placed in the destination field. Overflow is also computed.

This operation's name is derived from the term modulus; the destination field receives the the residue of taking one source field *modulus* another source field.

Formats

CM:u-mod-2-1L	*dest/source1*, *source2*, *len*
CM:u-mod-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-mod-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:u-mod-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the unsigned integer residue field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer modulus (divisor) field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the modulus (divisor) is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *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 ⟨divisor was zero in processor k⟩ then *test-flag*$[k] \leftarrow 1$
    else *test-flag*$[k] \leftarrow 0$

The residue resulting from the reduction of the unsigned integer *source1* modulo the unsigned integer *source2* operand is stored into the *dest* field. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

If the divisor is zero occurs, then the *test-flag* is set and the value of the destination is unpredictable

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-MOVE

Copies a complex source value into the destination field.

Formats

CM:c-move-2L	*dest, source, ds, de, ss, se*
CM:c-move-1L	*dest, source, s, e*
CM:c-move-always-1L	*dest, source, s, e*
CM:c-move-constant-1L	*dest, source-value, s, e*
CM:c-move-const-always-1L	*dest, source-value, s, e*
CM:c-move-zero-1L	*dest, s, e*
CM:c-move-zero-always-1L	*dest, s, e*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

source-value The field ID of the complex source field. For CM:c-move-zero-1L and CM:c-move-zero-always-1L, this implicitly has the value zero.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

ds, de For CM:c-move-2L, the significand and exponent lengths for the *dest* field. The total length of an operand in this format is $2(ds + de + 1)$.

ss, se For CM:c-move-2L, the significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(ss + se + 1)$.

Overlap The fields *dest* and *source* may overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:c-move-2L.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ *source*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

else *overflow-flag*[*k*] ← 0
as appropriate.

The *source* field or value is copied into the *dest* field.

However, overlapping fields are not handled carefully and should be avoided.

F-MOVE

Copies a floating-point source value into the destination field.

Formats

CM:f-move-2L	*dest*, *source*, *ds*, *de*, *ss*, *se*
CM:f-move-1L	*dest*, *source*, *s*, *e*
CM:f-move-always-1L	*dest*, *source*, *s*, *e*
CM:f-move-constant-1L	*dest*, *source-value*, *s*, *e*
CM:f-move-const-always-1L	*dest*, *source-value*, *s*, *e*
CM:f-move-zero-1L	*dest*, *s*, *e*
CM:f-move-zero-always-1L	*dest*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

source-value A floating-point immediate operand to be used as the source. This should be of type double-float in Lisp/Paris and will be coerced if necessary. For CM:f-move-zero-1L and CM:f-move-zero-always-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

ds, *de* For CM:f-move-2L, the significand and exponent lengths for the *dest* field. The total length of an operand in this format is $ds + de + 1$.

ss, *se* For CM:f-move-2L, the significand and exponent lengths for the *source* field. The total length of an operand in this format is $ss + se + 1$.

Overlap The fields *dest* and *source* may overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:f-move-2L.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow source[k]$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

else *overflow-flag*[*k*] ← 0
as appropriate.

The *source* field or value is copied into the *dest* field.

Overlapping fields are handled carefully. The operation behaves as if the entire *source* field were first copied to a temporary buffer not overlapping either the *source* or *dest* field, and then the temporary buffer copied to the *dest* field.

S-MOVE

Copies a signed integer source value into the destination field.

Formats

CM:s-move-2L	*dest, source, dlen, slen*
CM:s-move-1L	*dest, source, len*
CM:s-move-always-1L	*dest, source, len*
CM:s-move-constant-1L	*dest, source-value, len*
CM:s-move-const-always-1L	*dest, source-value, len*
CM:s-move-zero-1L	*dest, len*
CM:s-move-zero-always-1L	*dest, len*

Operands

dest — The field ID of the signed integer destination field.

source — The field ID of the signed integer source field.

source-value — A signed integer immediate operand to be used as the source. For CM:s-move-zero-1L and CM:s-move-zero-always-1L, this implicitly has the value zero.

len — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than the maximum Paris field length.

dlen — For CM:s-move-1L, the length of the *dest* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

slen — For CM:s-move-1L, the length of the *source* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

Overlap The fields *dest* and *source* may overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:s-move-2L.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*[k] = 1) then
    dest[k] ← *source*[k]
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0

The *source* field or value is copied into the *dest* field. For CM:s-move-2L, if *slen* is less than *dlen* then the source value, regarded as a bit field, is padded at the most significent end with copies of the most significant source bit (sign extension), and if *slen* is greater than *dlen* then truncation occurs and overflow may be detected.

Overlapping fields are handled carefully. The operation behaves as if the entire *source* field were first copied to a temporary buffer not overlapping either the *source* or *dest* field, and then the temporary buffer copied to the *dest* field.

U-MOVE

Copies an unsigned integer source value into the destination field.

Formats

CM:u-move-2L	*dest*, *source*, *dlen*, *slen*
CM:u-move-1L	*dest*, *source*, *len*
CM:u-move-always-1L	*dest*, *source*, *len*
CM:u-move-constant-1L	*dest*, *source-value*, *len*
CM:u-move-const-always-1L	*dest*, *source-value*, *len*
CM:u-move-zero-1L	*dest*, *len*
CM:u-move-zero-always-1L	*dest*, *len*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

source-value An unsigned integer immediate operand to be used as the source. For CM:u-move-zero-1L and CM:u-move-zero-always-1L, this implicitly has the value zero.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than the maximum Paris field length.

dlen For CM:u-move-1L, the length of the *dest* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

slen For CM:u-move-1L, the length of the *source* field. This must be no smaller than 2 but no greater than the maximum Paris field length.

Overlap The fields *dest* and *source* may overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared. This can occur only for CM:u-move-2L.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ *source*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The *source* field or value is copied into the *dest* field. For CM:u-move-2L, if *slen* is less than *dlen* then the source value, regarded as a bit field, is padded at the most significent end with zero bits, and if *slen* is greater than *dlen* then truncation occurs and overflow may be detected.

Overlapping fields are handled carefully. The operation behaves as if the entire *source* field were first copied to a temporary buffer not overlapping either the *source* or *dest* field, and then the temporary buffer copied to the *dest* field.

F-MOVE-DECODED-CONSTANT

Copies a decoded immediate floating-point source value into the destination field.

Formats CM:f-move-decoded-constant-1L *dest*, *low-s-value*, *high-s-value*, *e-value*, *sign-value*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

low-s-value An unsigned integer immediate operand to be used as the low 32 bits of the integer significand.

high-s-value An unsigned integer immediate operand to be used as the high bits of the integer significand.

e-value A signed integer immediate operand to be used as the integer exponent.

sign-value A signed integer immediate operand to be used as the integer sign. This must be either 1 or -1.

s, *e* The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Overlap There are no constraints, because overlap is not possible.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *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 the source values into the destination field, reversing the order of the bits.

Formats CM:move-reversed-1L *dest*, *source*, *len*
CM:move-reversed-always-1L *dest*, *source*, *len*

Operands *dest* The field ID of the destination field.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context The non-always operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $\textit{context-flag}[k] = 1$) then
    for j from 0 to $len - 1$ do
      $dest[k]\langle j\rangle \leftarrow source[k]\langle len - j - 1\rangle$

The *source* field or value is copied into the *dest* field, with the order of the bits reversed; that is, the least significant bit of the *source* field is copied into the most significant bit of the *dest* field, and so on.

F-MULT-ADD

Calculates a value $xa + b$ and places it in the destination.

Formats

CM:f-mult-add-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-add-always-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-const-add-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-const-add-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-add-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-add-const-always-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-const-add-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-mult-const-add-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source1 — The field ID of the floating-point first source field.

source2 — The field ID of the floating-point second source (multiplier) field.

source2-value — A floating-point immediate operand to be used as the second source (multiplier).

source3 — The field ID of the floating-point third source (augend) field.

source3-value — A floating-point immediate operand to be used as the third source (augend).

s, *e* — The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition

For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow (source1[k] \times source2[k]) + source3[k]$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as floating-point numbers and then a third operand, *source3*, is added to the product. The result is stored in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-mult-add-1L is equivalent to the sequence

CM:f-multiply-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-add-3-1L *dest*, *temp*, *source3*, *s*, *e*

but may be faster.

F-MULT-SUB

Calculates a value $xa - b$ and places it in the destination.

Formats

CM:f-mult-sub-1L	*dest, source1, source2, source3, s, e*
CM:f-mult-sub-always-1L	*dest, source1, source2, source3, s, e*
CM:f-mult-const-sub-1L	*dest, source1, source2-value, source3, s, e*
CM:f-mult-const-sub-always-1L	*dest, source1, source2-value, source3, s, e*
CM:f-mult-sub-const-1L	*dest, source1, source2, source3-value, s, e*
CM:f-mult-sub-const-always-1L	*dest, source1, source2, source3-value, s, e*
CM:f-mult-const-sub-const-1L	*dest, source1, source2-value, source3-value, s, e*
CM:f-mult-const-sub-const-a-1L	*dest, source1, source2-value, source3-value, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source (multiplier) field.

source2-value A floating-point immediate operand to be used as the second source (multiplier).

source3 The field ID of the floating-point third source (subtrahend) field.

source3-value A floating-point immediate operand to be used as the third source (subtrahend).

s, e The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
    if (always or *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 in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-mult-sub-1L is equivalent to the sequence

CM:f-multiply-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-subtract-3-1L *dest*, *temp*, *source3*, *s*, *e*

but may be faster.

F-MULT-SUBF

Calculates a value $b - xa$ and places it in the destination.

Formats

CM:f-mult-subf-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-subf-always-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-mult-const-subf-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-const-subf-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-mult-subf-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-subf-const-always-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-mult-const-subf-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-mult-const-subf-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source (multiplier) field.

source2-value A floating-point immediate operand to be used as the second source (multiplier).

source3 The field ID of the floating-point third source (minuend) field.

source3-value A floating-point immediate operand to be used as the third source (minuend).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ *source3*$[k]$ − (*source1*$[k] \times$ *source2*$[k]$)
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

Two operands *source1* and *source2* are multiplied as floating-point numbers and the product is subtracted from a third operand, *source3*. The result is stored in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-mult-subf-1L is equivalent to the sequence

CM:f-multiply-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-subtract-3-1L *dest*, *source3*, *temp*, *s*, *e*

but may be faster.

C-MULTIPLY

The product of two complex source values is placed in the destination field.

Formats

CM:c-multiply-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-multiply-always-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-multiply-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-multiply-always-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-multiply-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-multiply-const-always-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-multiply-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:c-multiply-const-always-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-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 complex numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

F-MULTIPLY

The product of two floating-point source values is placed in the destination field.

Formats

CM:f-multiply-2-1L	*dest/source1, source2, s, e*
CM:f-multiply-always-2-1L	*dest/source1, source2, s, e*
CM:f-multiply-3-1L	*dest, source1, source2, s, e*
CM:f-multiply-always-3-1L	*dest, source1, source2, s, e*
CM:f-multiply-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-multiply-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:f-multiply-constant-3-1L	*dest, source1, source2-value, s, e*
CM:f-multiply-const-always-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ *source1*$[k] \times$ *source2*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

Two operands, *source1* and *source2*, are multiplied as floating-point numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-MULTIPLY

The product of two signed integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:s-multiply-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-multiply-2-1L	*dest/source1, source2, len*
CM:s-multiply-3-1L	*dest, source1, source2, len*
CM:s-multiply-constant-2-1L	*dest/source1, source2-value, len*
CM:s-multiply-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-multiply-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-multiply-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-multiply-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if the product cannot be represented in the destination field; otherwise it is cleared.

Context

This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*[k] = 1 then
dest[k] ← *source1*[k] × *source2*[k]
if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
else *overflow-flag*[k] ← 0

Two operands, *source1* and *source2*, are multiplied as signed integers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-MULTIPLY

The product of two unsigned integer source values is placed in the destination field. Overflow is also computed.

Formats

CM:u-multiply-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-multiply-2-1L	*dest/source1, source2, len*
CM:u-multiply-3-1L	*dest, source1, source2, len*
CM:u-multiply-constant-2-1L	*dest/source1, source2-value, len*
CM:u-multiply-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-multiply-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-multiply-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-multiply-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the sum cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do

if *context-flag*[k] = 1 then
  dest[k] ← *source1*[k] × *source2*[k]
  if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
  else *overflow-flag*[k] ← 0

Two operands, *source1* and *source2*, are multiplied as unsigned integers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

MULTISPREAD-C-ADD

The destination field in every selected processor receives the sum of the complex floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-c-add-1L *dest*, *source*, *axis-mask*, *s*, *e*

Operands	*dest*	The field ID of the complex destination field.
	source	The field ID of the complex source field.
	axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
	s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$
    $$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$
where *hyperplane* is as defined on 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-c-add operation combines *source* fields by performing complex floating-point addition.

A call to CM:multispread-c-add-1L is equivalent to the sequence

for all integers j, $0 \le j < \textit{rank}(\textit{geometry}(\textit{current-vp-set}))$, in any sequential order, do
  if $\textit{axis-mask}\langle j \rangle = 1$ then
    CM:spread-with-c-add-1L *dest*, *source*, j, *s*, *e*

but may be faster.

MULTISPREAD-F-ADD

The destination field in every selected processor receives the sum of the floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-f-add-1L *dest*, *source*, *axis-mask*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\mathit{context\text{-}flag}[k] = 1$ then

let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$

let $r = \mathit{rank}(g)$

let $\mathit{axis\text{-}set} = \{\, m \mid 0 \leq m < r \land (\mathit{axis\text{-}mask}\langle m \rangle = 1) \,\}$

let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \land \mathit{context\text{-}flag}[m] = 1 \,\}$

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \mathit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-f-add operation combines *source* fields by performing floating-point addition.

A call to CM:multispread-f-add-1L is equivalent to the sequence

CM:f-move-zero-always-1L *temp*, *s*, *e*
CM:f-move-1L *temp*, *source*, *s*, *e*
CM:store-context *ctemp*
CM:set-context

for all integers j, $0 \leq j < rank(geometry(current\text{-}vp\text{-}set))$, in any sequential order, do
  if $axis\text{-}mask\langle j \rangle = 1$ then
    CM:spread-with-f-add-1L *temp*, *temp*, j, *s*, *e*
CM:load-context *ctemp*
CM:f-move-1L *dest*, *temp*, *s*, *e*

but may be faster.

MULTISPREAD-S-ADD

The destination field in every selected processor receives the sum of the signed integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-s-add-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the signed integer source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{ m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \}$
    let $C_k = \{ m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \}$
    $$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$
where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-s-add operation combines *source* fields by performing signed integer addition.

MULTISPREAD-U-ADD

The destination field in every selected processor receives the sum of the unsigned integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-u-add-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{\, m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m\rangle = 1)\,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1\,\}$

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m]\right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-u-add operation combines *source* fields by performing unsigned integer addition.

MULTISPREAD-COPY

The destination field in every selected processor receives a copy of the source value from a particular value within its scan subclass.

Formats CM:multispread-copy-1L *dest*, *source*, *axis-mask*, *len*, *multi-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

multi-coordinate An unsigned integer, the multi-coordinate indicating which element of each hyperplane is to be replicated throughout that hyperplane.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$
    let $c = \textit{deposit-multi-coordinate}(g, k, \textit{axis-set}, \textit{multi-coordinate})$
    $\textit{dest}[k] \leftarrow \textit{source}[c]$

where *deposit-multi-coordinate* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations.

To construct a multi-coordinate, construct a send-address and provide it as an argument to CM:fe-extract-multi-coordinate.

MULTISPREAD-LOGAND

The destination field in every selected processor receives the bitwise logical AND of the source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-logand-1L *dest, source, axis-mask, len*

Operands *dest* The field ID of the destination field.

source The field ID of the source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{\, m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m \rangle = 1)\,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1\,\}$
    $$\textit{dest}[k] \leftarrow \left(\bigwedge_{m \in C_k} \textit{source}[m] \right)$$
where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-logand operation combines *source* fields by performing bitwise logical AND operations.

MULTISPREAD-LOGIOR

The destination field in every selected processor receives the bitwise logical inclusive OR of the source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-logior-1L *dest*, *source*, *axis-mask*, *len*

Operands *dest* The field ID of the destination field.

source The field ID of the source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $r = \textit{rank}(g)$
let $\textit{axis-set} = \{\, m \mid 0 \le m < r \land (\textit{axis-mask}\langle m \rangle = 1) \,\}$
let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \land \textit{context-flag}[m] = 1 \,\}$

$$\textit{dest}[k] \leftarrow \left(\bigvee_{m \in C_k} \textit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-logior operation combines *source* fields by performing bitwise logical inclusive OR operations.

MULTISPREAD-LOGXOR

The destination field in every selected processor receives the bitwise logical exclusive OR of the source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-logxor-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest	The field ID of the destination field.
source	The field ID of the source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\mathit{context\text{-}flag}[k] = 1$ then
let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
let $r = \mathit{rank}(g)$
let $\mathit{axis\text{-}set} = \{\, m \mid 0 \le m < r \wedge (\mathit{axis\text{-}mask}\langle m \rangle = 1)\,\}$
let $C_k = \{\, m \mid m \in \mathit{hyperplane}(g, k, \mathit{axis\text{-}set}) \wedge \mathit{context\text{-}flag}[m] = 1\,\}$

$$\mathit{dest}[k] \leftarrow \left(\bigoplus_{m \in C_k} \mathit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations.

MULTISPREAD-F-MAX

The destination field in every selected processor receives the largest of the floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-f-max-1L *dest*, *source*, *axis-mask*, *s*, *e*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $r = \textit{rank}(g)$
let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$
let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$
$\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m] \right)$
where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-f-max operation combines *source* fields by performing a floating-point maximum operation.

MULTISPREAD-S-MAX

The destination field in every selected processor receives the largest of the signed integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-s-max-1L *dest, source, axis-mask, len*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $context\text{-}flag[k] = 1$ then

let $g = geometry(current\text{-}vp\text{-}set)$

let $r = rank(g)$

let $axis\text{-}set = \{ m \mid 0 \leq m < r \wedge (axis\text{-}mask\langle m \rangle = 1) \}$

let $C_k = \{ m \mid m \in hyperplane(g, k, axis\text{-}set) \wedge context\text{-}flag[m] = 1 \}$

$dest[k] \leftarrow \left(\max_{m \in C_k} source[m] \right)$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-s-max operation combines *source* fields by performing a signed integer maximum operation.

MULTISPREAD-U-MAX

The destination field in every selected processor receives the largest of the unsigned integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-u-max-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1)\,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1\,\}$
    $\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m] \right)$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-u-max operation combines *source* fields by performing an unsigned integer maximum operation.

MULTISPREAD-F-MIN

The destination field in every selected processor receives the smallest of the floating-point source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-f-min-1L *dest*, *source*, *axis-mask*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis-mask An unsigned integer, the mask indicating a set of NEWS axes.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $r = \textit{rank}(g)$

let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \,\}$

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 *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-f-min operation combines *source* fields by performing a floating-point minimum operation.

MULTISPREAD-S-MIN

The destination field in every selected processor receives the smallest of the signed integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-s-min-1L *dest*, *source*, *axis-mask*, *len*

Operands

- *dest* — The field ID of the signed integer destination field.
- *source* — The field ID of the signed integer source field.
- *axis-mask* — An unsigned integer, the mask indicating a set of NEWS axes.
- *len* — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $r = \textit{rank}(g)$
    let $\textit{axis-set} = \{\, m \mid 0 \le m < r \wedge (\textit{axis-mask}\langle m\rangle = 1) \,\}$
    let $C_k = \{\, m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \,\}$
    $\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m] \right)$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-s-min operation combines *source* fields by performing a signed integer minimum operation.

MULTISPREAD-U-MIN

The destination field in every selected processor receives the smallest of the unsigned integer source fields from all processors in the same hyperplane through the NEWS grid.

Formats CM:multispread-u-min-1L *dest*, *source*, *axis-mask*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis-mask	An unsigned integer, the mask indicating a set of NEWS axes.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $r = \textit{rank}(g)$

let $\textit{axis-set} = \{ m \mid 0 \leq m < r \wedge (\textit{axis-mask}\langle m \rangle = 1) \}$

let $C_k = \{ m \mid m \in \textit{hyperplane}(g, k, \textit{axis-set}) \wedge \textit{context-flag}[m] = 1 \}$

$$\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m] \right)$$

where *hyperplane* is as defined on page 44.

See section 5.20 on page 42 for a general description of multispread operations. The CM:multispread-u-min operation combines *source* fields by performing an unsigned integer minimum operation.

MY-NEWS-COORDINATE

Stores the NEWS coordinate of each selected processor along a specified NEWS axis into a destination field within that processor.

Formats CM:my-news-coordinate-1L *dest*, *axis*, *dlen*

Operands

dest The field ID of the unsigned integer destination field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    let g = *geometry*(*current-vp-set*)
    dest$[k] \leftarrow$ *extract-news-coordinate*$(g, axis, k)$

where *extract-news-coordinate* is as defined on page 40.

This function calculates, within each selected processor, the NEWS coordinate of that processor along a specified NEWS axis.

MY-SEND-ADDRESS

Stores the send-address of each selected processor into a destination field in that processor.

Formats CM:my-send-address *dest*

Operands *dest* The field ID of the unsigned integer destination field. This must be no less than the value returned by CM:geometry-send-address-length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] ← k

This function stores into the *dest* field, within each selected processor, the send-address of that processor.

C-NE

Compares two complex source values. The *test-flag* is set if they are not equal; otherwise it is cleared.

Formats

CM:c-ne-1L	*source1*, *source2*, *s*, *e*
CM:c-ne-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:c-ne-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source. For CM:c-ne-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] ≠ *source2*[*k*]
      test-flag[*k*] ← 1
    else
      test-flag[*k*] ← 0

Two operands are compared as complex numbers. The first operand is a memory field; the second is a memory field or an immediate value. The *test-flag* is set if the first operand is not equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

F-NE

Compares two floating-point source values. The *test-flag* is set if they are not equal, and otherwise is cleared.

Formats

CM:f-ne-1L	*source1*, *source2*, *s*, *e*
CM:f-ne-constant-1L	*source1*, *source2-value*, *s*, *e*
CM:f-ne-zero-1L	*source1*, *s*, *e*

Operands

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source. For CM:f-ne-zero-1L, this implicitly has the value zero.

s, *e* The significand and exponent lengths for the *source1* and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source1*[*k*] ≠ *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 not equal to the second operand, and is cleared otherwise. Note that comparisons ignore the sign of zero; $+0$ and -0 are considered to be equal.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

S-NE

Compares two signed integer source values. The *test-flag* is set if they are not equal, and otherwise is cleared.

Formats

CM:s-ne-1L	*source1*, *source2*, *len*
CM:s-ne-2L	*source1*, *source2*, *slen1*, *slen2*
CM:s-ne-constant-1L	*source1*, *source2-value*, *len*
CM:s-ne-zero-1L	*source1*, *len*

Operands

source1 The field ID of the signed integer first source field.

source2 The field ID of the signed integer second source field.

source2-value A signed integer immediate operand to be used as the second source. For CM:s-ne-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *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. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-NE

Compares two unsigned integer source values. The *test-flag* is set if they are not equal, and otherwise is cleared.

Formats

CM:u-ne-1L	*source1*, *source2*, *len*
CM:u-ne-2L	*source1*, *source2*, *slen1*, *slen2*
CM:u-ne-constant-1L	*source1*, *source2-value*, *len*
CM:u-ne-zero-1L	*source1*, *len*

Operands

source1 The field ID of the unsigned integer first source field.

source2 The field ID of the unsigned integer second source field.

source2-value An unsigned integer immediate operand to be used as the second source. For CM:u-ne-zero-1L, this implicitly has the value zero.

len The length of the *source1* and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner.

Flags *test-flag* is set if *source1* is not equal to *source2*; otherwise it is cleared.

Context This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    if *source1*[k] $\neq$ *source2*[k] then
      test-flag[k] $\leftarrow$ 1
    else
      test-flag[k] $\leftarrow$ 0

Two operands are compared as unsigned integers. Operand *source1* is always a memory field; operand *source2* is a memory field or an immediate value. The *test-flag* is set if the first operand is not equal to the second operand, and is cleared otherwise.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

C-NEGATE

Copies a complex number with both signs inverted.

Formats CM:c-negate-1-1L *dest/source*, *s*, *e*
CM:c-negate-2-1L *dest*, *source*, *s*, *e*

Operands
dest The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
$dest[k].real \leftarrow -source[k].real$
$dest[k].imag \leftarrow -source[k].imag$

A copy of the *source* operand, with both sign bits inverted, is placed in the *dest* operand.

F-NEGATE

Copies a floating-point number with its sign inverted.

Formats CM:f-negate-1-1L *dest/source*, *s*, *e*
CM:f-negate-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *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 field ID of the signed integer destination field.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] $\leftarrow$ $-$*source*[k]
    if ⟨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 field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags

overflow-flag is set if the result cannot be represented in the destination field; otherwise it is cleared. Overflow occurs whenever the source value is non-zero.

Context

This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] ← −*source*[k]
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0

The negative of the *source* operand is placed in the *dest* operand. If overflow occurs, then the *dest* field will contain a value equal to 2^{len} − *source*. This operation matches the functionality of the unary "-" operator on unsigned integers in the C language.

F-NEWS-ADD

The sum of two floating-point source values (one from a NEWS neighbor) is placed in the destination field.

Formats

CM:f-news-add-2-1L	*dest*, *source*, *axis*, *direction*, *s*, *e*
CM:f-news-add-always-2-1L	*dest*, *source*, *axis*, *direction*, *s*, *e*
CM:f-news-add-3-1L	*dest*, *source1*, *source2*, *axis*, *direction*, *s*, *e*
CM:f-news-add-always-3-1L	*dest*, *source1*, *source2*, *axis*, *direction*, *s*, *e*
CM:f-news-add-const-3-1L	*dest*, *source1*, *source2-value*, *axis*, *direction*, *s*, *e*
CM:f-news-add-const-a-3-1L	*dest*, *source1*, *source2-value*, *axis*, *direction*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context

The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
let g = *geometry*(*current-vp-set*)
dest$[k] \leftarrow$ *source1*$[k]$ + *source2*[*news-neighbor*(g, k, *axis*, *direction*)]
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

where *news-neighbor* is is defined in the NEWS Communication section of the Instruction Set Overview Chapter.

Two source operands are added as floating-point numbers and the result is stored in *dest*. The various operand formats allow source operands to be either memory fields or constants. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The instructions with two operands take *source* from a NEWS neighbor, sum it with *dest* and store the result back in *dest*.

For the instructions CM:f-news-add-3-1L and CM:f-news-add-always-3-1L, *source2* is taken from a NEWS neighbor.

The instructions CM:f-news-add-const-3-1L and CM:f-news-add-const-a-3-1L take *source1* is from a NEWS neighbor. Note that the *a* in CM:f-news-add-const-a-3-1L stands for "always."

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-add-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-add-3-1L *dest*, *source1*, *temp*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-ADD-MULT

Calculates the value $(a + x)b$, where one of the operands is taken from a NEWS neighbor, and places the result in the destination.

Formats CM:f-news-add-mult-4-1L *dest, source1, source2, source3, axis, direction, s, e*
CM:f-news-add-const-mult-4-1L *dest, source1, source2-value, source3, axis, direction, s, e*

Operands *dest* The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

source3 A floating-point immediate operand to be used as the third source.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1. Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
$\textit{dest}[k] \leftarrow (\textit{source1} + \textit{source2}[\textit{news-neighbor}(g, k, \textit{axis}, \textit{direction})]) \times \textit{source3}[k]$
if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The sum of two source operands is multiplied by the value of a third source operand. The result is stored in *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The CM:f-news-add-mult-4-1L instruction takes *source2* from a NEWS neighbor. For the CM:f-news-add-const-mult-4-1L instruction, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

A call to CM:f-news-add-mult is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-add-mult-1L *souce1*, *temp*, *source3*, *s*, *e*

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-MULT

The product of two floating-point source values (one from a NEWS neighbor) is placed in the destination field.

Formats

CM:f-news-mult-2-1L	*dest*, *source*, *axis*, *direction*, *s*, *e*
CM:f-news-mult-always-2-1L	*dest*, *source*, *axis*, *direction*, *s*, *e*
CM:f-news-mult-3-1L	*dest*, *source1*, *source2*, *axis*, *direction*, *s*, *e*
CM:f-news-mult-always-3-1L	*dest*, *source1*, *source2*, *axis*, *direction*, *s*, *e*
CM:f-news-mult-const-3-1L	*dest*, *source1*, *source2-value*, *axis*, *direction*, *s*, *e*
CM:f-news-mult-const-a-3-1L	*dest*, *source1*, *source2-value*, *axis*, *direction*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*. Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor *k* in the *current-vp-set* do

if *context-flag*[k] = 1 then
  let g = *geometry*(*current-vp-set*)
  dest[k] ← *source1*[k] × *source2*[*news-neighbor*($g, k,$ *axis*, *direction*)]
  if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

Two source operands are multiplied as floating-point numbers. The result is stored in *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The instructions with two operands take *source* from a NEWS neighbor, multiply it with *dest*, and store the result back in *dest*.

For the instructions CM:f-news-mult-3-1L and CM:f-news-mult-always-3-1L, *source2* is taken from a NEWS neighbor.

For the instructions CM:f-news-mult-const-3-1L and CM:f-news-mult-const-a-3-1L, *source1* is taken from a NEWS neighbor. Note that the *a* in CM:f-news-mul-const-always-3-1L stands for "always." This is necessary to meet the 31 character limit on instruction names.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-mult-3-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-multiply-3-1L *dest*, *source1*, *temp*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-MULT-ADD

The product of two floating-point source values (one from a NEWS neighbor) is added to yet another floating-point source value; the result is placed in the destination field.

Formats

CM:f-news-mult-add-4-1L	*dest*, *source1*, *source2*, *source3*, *axis*, *direction*, *s*, *e*
CM:f-news-mult-const-add-4-1L	*dest*, *source1*, *source2-value*, *source3*, *axis*, *direction*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point multiplicand field.

source2 The field ID of the floating-point multiplier field. These values may be taken from a NEWS neighbor.

source2-value A floating-point immediate operand to be used as the multiplier.

source3 The field ID of the floating-point addend field. These values may be taken from a NEWS neighbor.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then

let $g = geometry(current\text{-}vp\text{-}set)$
$dest[k] \leftarrow source1[k] \times source2[news\text{-}neighbor(g, k, axis, direction)] + source3[k]$
if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

Two operands are multiplied as floating-point numbers; to the product is added a third operand. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

For CM:f-news-mult-add-4-1L, *source2* is taken from a NEWS neighbor.

For CM:f-news-mult-const-add-4-1L, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* or *source3-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-mult-add-4-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-multiply-3-1L *temp*, *source1*, *temp*, s, e
CM:f-add-3-1L *dest*, *temp*, *source3*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-MULT-SUB

From the product of two floating-point source values (one from a NEWS neighbor) is subtracted yet another floating-point source value; the result is placed in the destination field.

Formats

CM:f-news-mult-sub-4-1L	*dest, source1, source2, source3, axis, direction, s, e*
CM:f-news-mult-const-sub-4-1L	*dest, source1, source2-value, source3, axis, direction, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point multiplicand field.

source2 The field ID of the floating-point multiplier field.

source2-value A floating-point immediate operand to be used as the multiplier.

source3 The field ID of the floating-point subtrahend field.

source3-value A floating-point immediate operand to be used as the subtrahend.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    let *g* = *geometry*(*current-vp-set*)
    $dest[k] \leftarrow source1[k] \times source2[news\text{-}neighbor(g, k, axis, direction)] - source3[k]$
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

Two operands, *source1* and *source2*, are multiplied as floating-point numbers; from the product is subtracted a third operand, *source3*. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

For CM:f-news-mult-sub-4-1L, *source2* is taken from a NEWS neighbor.

For and CM:f-news-mult-const-sub-4-1L, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* or *source3-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

A call to CM:f-news-mult-sub-4-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-multiply-3-1L *temp*, *source1*, *temp*, *s*, *e*
CM:f-subtract-3-1L *dest*, *temp*, *source3*, *s*, *e*

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-SUB

The difference of two floating-point source values (one from a NEWS neighbor) is placed in the destination field.

Formats

CM:f-news-sub-2-1L	*dest*, *source*, *axis*, *direction*, *s*, *e*
CM:f-news-sub-always-2-1L	*dest*, *source*, *axis*, *direction*, *s*, *e*
CM:f-news-sub-3-1L	*dest*, *source1*, *source2*, *axis*, *direction*, *s*, *e*
CM:f-news-sub-always-3-1L	*dest*, *source1*, *source2*, *axis*, *direction*, *s*, *e*
CM:f-news-sub-const-3-1L	*dest*, *source1*, *source2-value*, *axis*, *direction*, *s*, *e*
CM:f-news-sub-const-a-3-1L	*dest*, *source1*, *source2-value*, *axis*, *direction*, *s*, *e*

Operands

dest The field ID of the floating-point destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the floating-point first source field) field. This is the minuend.

source2 The field ID of the floating-point second source field. This is the subtrahend.

source2-value A floating-point immediate operand to be used as the second source.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*[k] = 1 then
let g = *geometry*(*current-vp-set*)
dest[k] ← *source1*[k] − *source2*[*news-neighbor*(g, k, *axis*, *direction*)]
if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The operands are treated as as floating-point numbers and one is subtracted from another. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The instructions with two operands take *source* from a NEWS neighbor, subtract it from *dest*, and store the result stored back in *dest*.

For the instructions CM:f-news-sub-3-1L and CM:f-news-sub-always-3-1L, *source2* is obtained from a NEWS neighbor.

For the instructions CM:f-news-sub-const-3-1L and CM:f-news-sub-const-a-3-1L, *source2* is a constant and *source1* is obtained from a NEWS neighbor. Note that the *a* in CM:f-news-sub-const-a-3-1L stands for "always."

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

A call to CM:f-news-sub-3-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-subtract-3-1L *dest*, *source1*, *temp*, s, e

but is faster at high VP ratios and requires little temporary memory.

F-NEWS-SUB-MULT

Calculates the value $(a - x)b$, when one of the operands is taken from a NEWS neighbor, and places the result in the destination.

Formats

CM:f-news-sub-mult-4-1L	*dest*, *source1*, *source2*, *source3*, *axis*, *direction*, *s*, *e*
CM:f-news-sub-const-mult-4-1L	*dest*, *source1*, *source2-value*, *source3*, *axis*, *direction*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

source3 The field ID of the floating-point third source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Note that in the conditional cases the storing of data depends only on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    let $g =$ *geometry*(*current-vp-set*)
    dest$[k] \leftarrow$ (*source1* − *source2*[*news-neighbor*(g, k, *axis*, *direction*)]) $\times$ *source3*$[k]$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The difference of two operands is multiplied by the value of a third operand. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. Each instruction takes one source field from a NEWS neighbor; the default is *source2*.

The CM:f-news-sub-mult-4-1L instruction takes *source2* from a NEWS neighbor. For the CM:f-news-sub-const-mult-4-1L instruction, *source2* is a constant and *source3* is taken from a NEWS neighbor.

If *direction* is :upward then each processor retrieves data from the neighbor whose NEWS coordinate is one greater along *axis*, with the processor whose coordinate is greatest retrieving data from the processor whose coordinate is zero.

If *direction* is :downward then each processor retrieves data from the neighbor whose NEWS coordinate is one less along *axis*, with the processor whose coordinate is zero retrieving data from the processor whose coordinate is greatest.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

A call to CM:f-news-sub-mult-4-1L is equivalent to the sequence

CM:get-from-news-1L *temp*, *source2*, *axis*, *direction*, $(s + e + 1)$
CM:f-sub-mult-1L *dest*, *source1*, *temp source3*, *s*, *e*

but is faster at high VP ratios and requires little temporary memory.

NEXT-STACK-FIELD-ID

Determines the next stack field id that would be returned by a call to CM:allocate-stack-field.

Formats result ← CM:next-stack-field-id

Operands None.

Result An unsigned integer, the field ID that will be returned by the next invocation of CM:allocate-stack-field.

Context This operation is unconditional. It does not depend on the *context-flag*.

This function returns the next stack field id to be allocated.

FE-PACKED-ARRAY-FORMAT

This front-end instruction returns an array format descriptor for a packed front-end array format. A format descriptor may be used as the *format* argument to any array transfer instruction, although this is not required.

See also CM:fe-array-format and CM:fe-structure-array-format.

Formats result ← CM:fe-packed-array-format *cm-element-size*, *[array-element-size]*

Operands *cm-element-size* A signed integer immediate operand to be used as the number of bits each Connection Machine element occupies in the front-end array. This must be a power of two between 1 and 128.

array-element-size A signed integer immediate operand to be used as the number of bits in each front-end array element. This must be a power of two between 1 and 128.

In Lisp/Paris, this argument is optional. If not specified, it defaults to the actual front-end element size or, if the front-end array elements are general (i.e., of type t), *array-element-size* defaults to the value of *cm-element-size*.

Result The array format descriptor specified.

Context This is a front-end operation. It does not depend on the value of the *context-flag*.

The return value is a format descriptor for packed arrays; it can be passed to any array transfer instruction. In this format, multiple Connection Machine array elements are packed into each front-end array element during array transfers in either direction between the Connection Machine and the front-end computer.

By using this instruction, it is also possible to specify an extended-element front-end array format. In an extended-element format, each CM element is stored in multiple front-end array elements.

The value of *cm-element-size* defines the unit of measure for the *fe-offset-vector* argument to the CM:read-from-news-array and CM:write-to-news-array instructions.

The value of *array-element-size* defines the unit of measure for the argument *fe-dimension-vector* to the CM:read-from-news-array and CM:write-to-news-array instructions.

The number of Connection Machine elements packed into each front-end array element is the ratio of *array-element-size* to *cm-element-size*. If *array-element-size* is larger than *cm-element-size*, multiple Connection Machine elements are packed into each front-end array element. Alternatively, if *array-element-size* is smaller than *cm-element-size*, each CM element is stored in more than one front-end array element.

The ordering of the packing defaults to the standard ordering for the front end. For example, on a VAX the Connection Machine element with the smallest coordinates is put into the least significant bits of the front-end array element. On a Sun, the Connection Machine element with the largest coordinates is put into the least significant bits of the front-end array element.

F-C-PHASE

Calculates the phase of the complex source field and puts the result in the floating-point destination field.

Formats CM:f-c-phase-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of the *dest* field in this format is $s + e + 1$. The total length of the *source* field in this format is $2(s + e + 1)$.

Overlap The *dest* field must be either identical to *source*, identical to $(source+s+e+1)$, or disjoint from *source*.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow atan2(source[k].imag, source[k].real)$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The phase of a number is the angle part of its polar representation as a complex number.

PHYSICAL-VP-SET

Returns a VP set that has one virtual processor for each physical processor.

Formats result ← CM:physical-vp-set

Operands None.

Result A VP set ID, identifying the VP set whose VP ratio is 1.

Context This operation is unconditional. It does not depend on the *context-flag*.

C-C-POWER

Raises a complex number to a complex power.

Formats

CM:c-c-power-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-c-power-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-c-power-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-c-power-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex first source field.

source2 The field ID of the complex second source field.

source2-value A complex immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected. *test-flag* is set if zero is raised to a non-positive power; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    $\mathit{dest}[k] \leftarrow \mathit{source1}[k]^{\mathit{source2}[k]}$
    if $\mathit{source1}[k] = 0.0$ and $\mathit{source2}[k].\mathit{real} \leq 0.0$
    and $\mathit{source2}[k].\mathit{imag} = 0.0$ then
      $\mathit{test\text{-}flag}[k] \leftarrow 1$
    else $\mathit{test\text{-}flag}[k] \leftarrow 0$
    if ⟨overflow occurred in processor k⟩ then $\mathit{overflow\text{-}flag}[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent), using exp and ln operations.

The result is stored into the memory field *dest.* The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

C-F-POWER

Raises a complex number to a floating-point power.

Formats

CM:c-f-power-2-1L	*dest/source1, source2, s, e*
CM:c-f-power-3-1L	*dest, source1, source2, s, e*
CM:c-f-power-constant-2-1L	*dest/source1, source2-value, s, e*
CM:c-f-power-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex first source field.

source2 The field ID of the floating-point second source field.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest and source1* and *source2* fields. The total length of the *dest and source1* field in this format is $2(s + e + 1)$. The total length of the *source2* field in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected. *test-flag* is set if zero is raised to a non-positive power; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k] \leftarrow source1[k]^{source2[k]}$
    if $source1[k] = 0.0$ and $source2[k].real \leq 0.0$
    and $source2[k].imag = 0.0$ then
      $test\text{-}flag[k] \leftarrow 1$
    else $test\text{-}flag[k] \leftarrow 0$
    if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent), using exp and ln operations.

The result is stored into the memory field *dest.* The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

C-S-POWER

Raises a complex number to a signed integer power.

Formats

CM:c-s-power-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:c-s-power-2-2L	*dest*/*source1*, *source2*, *slen2*, *s*, *e*
CM:c-s-power-constant-2-1L	*dest*/*source1*, *source2-value*, *s*, *e*
CM:c-s-power-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex base field.

source2 The field ID of the signed integer exponent field.

source2-value A signed integer immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $2(s+e+1)$.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected. *test-flag* is set if zero is raised to a negative power; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] ← *source1*[k]$^{source2[k]}$
    if *source1*[k] = 0.0 and *source2*[k] < 0 then
      test-flag[k] ← 1
    else *test-flag*[k] ← 0
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The *source1* field (the base) is raised to the power *source2* (the exponent), using repeated multiplications.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

C-U-POWER

Raises a complex number to an unsigned integer power.

Formats

CM:c-u-power-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:c-u-power-2-2L	*dest*/*source1*, *source2*, *slen2*, *s*, *e*
CM:c-u-power-constant-2-1L	*dest*/*source1*, *source2-value*, *s*, *e*
CM:c-u-power-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source1 The field ID of the complex base field.

source2 The field ID of the unsigned integer exponent field.

source2-value An unsigned integer immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $2(s+e+1)$.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    desk$[k] \leftarrow$ *source1*$[k]^{source2[k]}$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent), using repeated multiplications.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

F-F-POWER

Raises a floating-point number to a floating-point power.

Formats

CM:f-f-power-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-f-power-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-f-power-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-f-power-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point base field.

source2 The field ID of the floating-point exponent field.

source2-value A floating-point immediate operand to be used as the exponent.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if the base is negative and the exponent is non-zero, or if the base is zero and the exponent is non-positive; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    if *source1*$[k] = 0$ then
      if *source2*$[k] \leq 0$ then
        dest$[k] \leftarrow 0$
        test-flag$[k] \leftarrow 1$
      else
        dest$[k] \leftarrow 0$
        test-flag$[k] \leftarrow 0$
    else if *source1*$[k] < 0$ then

```
        if source2[k] = 0 then
            dest[k] ← 1.0
            test[k] ← 0
        else
            dest[k] ← ⟨undefined⟩
            test-flag[k] ← 1
    else
        dest[k] ← exp(source2[k] × ln source1[k])
        test-flag[k] ← 0
        if ⟨overflow occurred in processor k⟩ then overflow-flag[k] ← 1
```

The *source1* field (the base) is raised to the power *source2* (the exponent).

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

F-S-POWER

Raises a floating-point number to a signed integer power.

Formats

CM:f-s-power-3-2L	*dest, source1, source2, slen2, s, e*
CM:f-s-power-2-2L	*dest/source1, source2, slen2, s, e*
CM:f-s-power-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-s-power-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point base field.

source2 The field ID of the signed integer exponent field.

source2-value A signed integer immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    if $source2[k] < 0$ then
      let $temp1_k = 1.0/source1[k]$
      let $temp2_k = -source2[k]$
    else
      let $temp1_k = source1[k]$
      let $temp2_k = source2[k]$
    if $temp2_k\langle 0\rangle = 0$ 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 field ID of the floating-point destination field.

source1 — The field ID of the floating-point base field.

source2 — The field ID of the unsigned integer exponent field.

source2-value — An unsigned integer immediate operand to be used as the second source.

s, *e* — The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 — The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $\textit{temp}_k = \textit{source1}[k]$
    if $(\textit{slen2} = 0) \vee (\textit{source2}[k]\langle 0\rangle = 0)$ then
      $\textit{dest}[k] \leftarrow 1.0$
    else
      $\textit{dest}[k] \leftarrow \textit{temp}_k$
    for j from 1 to $\textit{slen2} - 1$ do
      if $\textit{source2}[k]\langle j : \textit{slen2} - 1\rangle \neq 0$ then let $\textit{temp}_k = \textit{temp}_k \times \textit{temp}_k$
      if $\textit{source2}[k]\langle j\rangle$ then $\textit{dest}[k] \leftarrow \textit{dest}[k] \times \textit{temp}_k$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The *source1* field (the base) is raised to the power *source2* (the exponent).

The result is stored into the memory field *dest.* The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

S-S-POWER

Raises a signed integer to a signed integer power.

Formats

CM:s-s-power-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-s-power-2-1L	*dest/source1, source2, len*
CM:s-s-power-3-1L	*dest, source1, source2, len*
CM:s-s-power-constant-2-1L	*dest/source1, source2-value, len*
CM:s-s-power-constant-3-1L	*dest, source1, source2-value, len*
CM:s-s-power-constant-3-2L	*dest, source1, source2-value, dlen, slen*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer base field.

source2 The field ID of the signed integer exponent field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-s-power-3-3L and CM:s-s-power-constant-3-2L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen For CM:s-s-power-constant-3-2L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-s-power-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-s-power-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if zero is raised to a negative power; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source2}[k] < 0$ then
      if $\textit{source1}[k] = 1$ then $\textit{dest}[k] \leftarrow 1$
      else $\textit{dest}[k] \leftarrow 0$
    else if $\textit{source2}[k] = 0$ then
      $\textit{dest}[k] \leftarrow 1$
    else
    $\textit{dest}[k] \leftarrow (\textit{source1}[k])^{\textit{source2}[k]}$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$
    else $\textit{overflow-flag}[k] \leftarrow 0$

The *source1* field (the base) is raised to the power *source2* (the exponent). If the exponent is negative, the result is always 0; if the exponent is zero, the result is always 1.

The result is stored into the memory field *dest.* The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len.*

S-U-POWER

Raises a signed integer to a unsigned integer power.

Formats

CM:s-u-power-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-u-power-constant-2-1L	*dest/source1*, *source2-value*, *len*
CM:s-u-power-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:s-u-power-constant-3-2L	*dest*, *source1*, *source2-value*, *dlen*, *slen1*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer base field.

source2 The field ID of the unsigned integer exponent field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-u-power-3-3L and CM:s-u-power-constant-3-2L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-u-power-3-3L and CM:s-u-power-constant-3-2L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-u-power-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, *source1* must be either disjoint from or identical to the *dest* field while *source2* must be disjoint from the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do

if *context-flag*[k] = 1 then
  if *source2*[k] = 0 then
    dest[k] ← 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 zero, the result is always 1.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-S-POWER

Raises a unsigned integer to a signed integer power.

Formats

CM:u-s-power-3-3L *dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-s-power-constant-2-1L *dest*/*source1*, *source2-value*, *len*
CM:u-s-power-constant-3-1L *dest*, *source1*, *source2-value*, *len*
CM:u-s-power-constant-3-2L *dest*, *source1*, *source2-value*, *dlen*, *slen1*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer base field.

source2 The field ID of the signed integer exponent field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:u-s-power-3-3L and CM:u-s-power-constant-3-2L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-s-power-3-3L and CM:u-s-power-constant-3-2L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-s-power-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. However, *source1* must be either disjoint from or identical to the *dest* field while *source2* must be disjoint from the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags

overflow-flag is set if the result cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if zero is raised to a negative power; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    test-flag$[k] \leftarrow 0$
    if *source1*$[k] = 0$ then
      test-flag$[k] \leftarrow 1$
    if *source2*$[k] < 0$ then
      $dest[k] \leftarrow \left\lfloor 1 \div source1[k]^{|source2[k]|} \right\rfloor$
    else if *source2*$[k] = 0$ then
      $dest[k] \leftarrow 1$
    else
    $dest[k] \leftarrow (source1[k])^{source2[k]}$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The *source1* field (the base) is raised to the power *source2* (the exponent). If the exponent is negative, the result is the truncation of the reciprocal of *source1* raised to the absolute value of *source2*. If the exponent is zero, the result is always 1.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit. If, in any particular processor, an attempt is made to raise zero to a negative power, the test flag in that processor is set.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-U-POWER

Raises an unsigned integer to an unsigned integer power.

Formats

CM:u-u-power-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-u-power-2-1L	*dest/source1, source2, len*
CM:u-u-power-3-1L	*dest, source1, source2, len*
CM:u-u-power-constant-2-1L	*dest/source1, source2-value, len*
CM:u-u-power-constant-3-1L	*dest, source1, source2-value, len*
CM:u-u-power-constant-3-2L	*dest, source1, source2-value, dlen, slen1*

Operands

dest The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer base field.

source2 The field ID of the unsigned integer exponent field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-u-power-3-3L and CM:u-u-power-constant-3-2L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-u-power-3-3L and CM:u-u-power-constant-3-2L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-u-power-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
if *source2*$[k] = 0$ then
dest$[k] \leftarrow 1$
else
dest$[k] \leftarrow (source1[k])^{source2[k]}$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *overflow-flag*$[k] \leftarrow 0$

The *source1* field (the base) is raised to the power *source2* (the exponent). If the exponent is zero, the result is always 1.

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

POWER-UP

This operation resets the Nexus, causing all front-end computers to become logically detached from the Connection Machine system.

Formats CM:power-up

Context This operation is unconditional. It does not depend on the *context-flag*.

This function resets the state of the Nexus, causing all front-end computers to become logically detached from the Connection Machine system. When a Connection Machine system is first powered up or is to be completely reset for other reasons, this is the first operation to perform. Any of the front-end computers may be used to do it.

If users on other front-end computers are actively using the Connection Machine system, their computations will be disrupted. Normally all the front-end computers are connected not only through the Connection Machine Nexus but also through some sort of communications network; a front end that executes CM:power-up will attempt to send messages through this network to the other front-end computers on the same Nexus indicating that a CM:power-up operation is being performed.

F-RANDOM

Stores a pseudo-randomly generated floating-point number into the destination field.

Formats CM:f-random-1L *dest*, *s*, *e*

Operands	*dest*	The field ID of the floating-point destination field.
	s, *e*	The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.
Context		This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then

$$dest[k] \leftarrow \frac{\langle \text{pseudo-random choice of some } j,\ +0 \leq j < 2^{len} \rangle}{2^{len}}$$

where *len* is the length of the destination field.

Into the destination field of each selected processor is stored a floating-point number pseudo-randomly chosen from a uniform distribution between zero (inclusive) and one (exclusive).

The seed for the Paris random number generator is automaticaly initialized the first time the random number generator is called. A value derived from the system clock is used. It is nonetheless possible to explicitly initialize the random number generator by call CM:initialize-random-generator.

Note: Less simple but more flexible random number generation routines are provided as part of the CM Scientific Subroutines Library (CMSSL). For instance, the CMSSL random number generators may be checkpointed to guard against accidental interuptions.

U-RANDOM

Stores a pseudo-randomly generated unsigned integer into the destination field.

Formats CM:u-random-1L *dest*, *len*, *limit*

Operands	*dest*	The field ID of the unsigned integer destination field.
	len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
	limit	An unsigned integer immediate operand to be used as the exclusive upper bound on values to be generated.
Context		This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow \langle$pseudo-random choice of some j, $0 \leq j < limit\rangle$

The *dest* field in each selected processor receives a pseudo-randomly chosen from a uniform distribution ranging from zero (inclusive) to the specified limit (exclusive).

F-RANK

The destination field in every selected processor receives the rank of that processor's key among all keys in the scan set for that processor.

Formats CM:f-rank-2L *dest*, *source*, *axis*, *dlen*, *s*, *e*, *direction*, *smode*, *sbit*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the floating-point source field. This is the sort key.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
dlen	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be no larger than the value returned by CM:geometry-coordinate-length.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.
direction	Either :upward or :downward.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *source* and *sbit* fields must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $S_k = scan\text{-}set(g, k, axis, direction, smode, sbit)$
case *direction* of
:upward:
let $L_k = \{ m \mid m \in S_k \wedge ((source[m] < source[k]) \vee (source[m] = source[k] \wedge m <$
:downward:
let $L_k = \{ m \mid m \in S_k \wedge ((source[m] > source[k]) \vee (source[m] = source[k] \wedge m >$
$dest[k] \leftarrow |L_k|$

where *scan-set* is as defined on page 44.

See section 5.20 on page 42 for a general description of scan sets and the effect of the *axis*, *direction*, *smode*, and *sbit* operands.

This operation determines the ordering necessary to sort the *source* fields within each scan set. It does not not actually move the data so as to sort it, but merely indicates where the data should be moved so as to sort it. A stable ranking is guaranteed. That is, two identical keys will be ranked in the order in which they occur in the *source* field.

In more detail: The *dest* field in each selected processor receives, as an unsigned integer, the rank of that processor's key within the set of keys in the scan set for that processor. This rank may be used to calculate a send address a CM:send operation may then be used to put the data into sorted order. (An advantage of decoupling the rank determination from the reordering process is that the data to be moved may be much larger than the key that determines the ordering, and indeed it may be desirable to reorder the other data but not the key itself. In this way ranking and reordering each need operate only on the relevant data.)

The way in which the rank operation uses scan sets has one unusual twist: A rank that is partitioned into scan sets restarts the rank *ordering* within each scan set (or segment). However, the rank *indices* assigned are not restarted within each scan set.

Specifically, along the entire *axis* specified, only one processor receives a rank index of 0. Rank indices in the first scan set (segment) begin at 0 and run through $n - 1$, where n is the number of active processors in the scan set; ranks in the second segment begin at n; and so forth. Thus, the smallest key in the first scan set has rank 0, the next smallest has rank 1; the smallest key in the second scan set has rank n, the next smallest has rank $n + 1$, and so on. Within each scan set the ranking index assigned to any given processor determines the rank of that processor's key value relative to the keys of all other active processors within that scan set. The non-repeating indices produce correctly sorted values when used by a send operation either along the entire axis (the scan subclass) or within one or more segments (the scan sets).

This operation was originally documented to result in a set of indexes that restart at 0 for each segment. To obtain that effect use the following strategy:

1) Use the rank function.

2) Set the context bit on for processors with segment bits and then call CM:my-news-address.

3) Use a segmented copy-scan operation to copy the NEWS address within each segment.

4) Subtract the results of the segmented copy scan from the results of the rank ordering.

S-RANK

The destination field in every selected processor receives the rank of that processor's key among all keys in the scan set for that processor.

Formats CM:s-rank-2L *dest*, *source*, *axis*, *dlen*, *slen*, *direction*, *smode*, *sbit*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the signed integer source field. This is the sort key.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be no larger than the value returned by CM:geometry-coordinate-length.

slen The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *source* and *sbit* fields must not overlap the *dest* field.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-set}(g, k, \textit{axis}, \textit{direction}, \textit{smode}, \textit{sbit})$
    case *direction* of
      :upward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] < \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k] \wedge m <$
      :downward:
        let $L_k = \{\, m \mid m \in S_k \wedge ((\textit{source}[m] > \textit{source}[k]) \vee (\textit{source}[m] = \textit{source}[k] \wedge m >$
    $\textit{dest}[k] \leftarrow |L_k|$

where *scan-set* is as defined on page 44.

See section 5.20 on page 42 for a general description of scan sets and the effect of the *axis*, *direction*, *smode*, and *sbit* operands.

This operation determines the ordering necessary to sort the *source* fields within each scan set. It does not not actually move the data so as to sort it, but merely indicates where the data should be moved so as to sort it. A stable ranking is guaranteed. That is, two identical keys will be ranked in the order in which they occur in the *source* field.

In more detail: The *dest* field in each selected processor receives, as an unsigned integer, the rank of that processor's key within the set of keys in the scan set for that processor. This rank may be used to calculate a send address a CM:send operation may then be used to put the data into sorted order. (An advantage of decoupling the rank determination from the reordering process is that the data to be moved may be much larger than the key that determines the ordering, and indeed it may be desirable to reorder the other data but not the key itself. In this way ranking and reordering each need operate only on the relevant data.)

The way in which the rank operation uses scan sets has one unusual twist: A rank that is partitioned into scan sets restarts the rank *ordering* within each scan set (or segment). However, the rank *indices* assigned are not restarted within each scan set.

Specifically, along the entire *axis* specified, only one processor receives a rank index of 0. Rank indices in the first scan set (segment) begin at 0 and run through $n - 1$, where n is the number of active processors in the scan set; ranks in the second segment begin at n; and so forth. Thus, the smallest key in the first scan set has rank 0, the next smallest has rank 1; the smallest key in the second scan set has rank n, the next smallest has rank $n + 1$, and so on. Within each scan set the ranking index assigned to any given processor determines the rank of that processor's key value relative to the keys of all other active processors within that scan set. The non-repeating indices produce correctly sorted values when used by a send operation either along the entire axis (the scan subclass) or within one or more segments (the scan sets).

U-RANK

The destination field in every selected processor receives the rank of that processor's key among all keys in the scan set for that processor.

Formats CM:u-rank-2L *dest, source, axis, dlen, slen, direction, smode, sbit*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field. This is the sort key.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be no larger than the value returned by CM:geometry-coordinate-length.

slen The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $S_k = scan\text{-}set(g, k, axis, direction, smode, sbit)$
    case *direction* of
      :upward:
        let $L_k = \{ m \mid m \in S_k \wedge ((source[m] < source[k]) \vee (source[m] = source[k] \wedge m$
      :downward:

$$\text{let } L_k = \{\, m \mid m \in S_k \wedge ((source[m] > source[k]) \vee (source[m] = source[k] \wedge m > k)),$$
$$dest[k] \leftarrow |L_k|$$

where *scan-set* is as defined on page 44.

See section 5.20 on page 42 for a general description of scan sets and the effect of the *axis*, *direction*, *smode*, and *sbit* operands.

This operation determines the ordering necessary to sort the *source* fields within each scan set. It does not not actually move the data so as to sort it, but merely indicates where the data should be moved so as to sort it. A stable ranking is guaranteed. That is, two identical keys will be ranked in the order in which they occur in the *source* field.

In more detail: The *dest* field in each selected processor receives, as an unsigned integer, the rank of that processor's key within the set of keys in the scan set for that processor. This rank may be used to calculate a send address a CM:send operation may then be used to put the data into sorted order. (An advantage of decoupling the rank determination from the reordering process is that the data to be moved may be much larger than the key that determines the ordering, and indeed it may be desirable to reorder the other data but not the key itself. In this way ranking and reordering each need operate only on the relevant data.)

The way in which the rank operation uses scan sets has one unusual twist: A rank that is partitioned into scan sets restarts the rank *ordering* within each scan set (or segment). However, the rank *indices* assigned are not restarted within each scan set.

Specifically, along the entire *axis* specified, only one processor receives a rank index of 0. Rank indices in the first scan set (segment) begin at 0 and run through $n - 1$, where n is the number of active processors in the scan set; ranks in the second segment begin at n; and so forth. Thus, the smallest key in the first scan set has rank 0, the next smallest has rank 1; the smallest key in the second scan set has rank n, the next smallest has rank $n + 1$, and so on. Within each scan set the ranking index assigned to any given processor determines the rank of that processor's key value relative to the keys of all other active processors within that scan set. The non-repeating indices produce correctly sorted values when used by a send operation either along the entire axis (the scan subclass) or within one or more segments (the scan sets).

C-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as complex numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:c-read-from-news-array-1L *front-end-array*, *fe-offset-vector*, *cm-start-vector*, *cm-end-vector*, *cm-axis-vector*, *source*, *s*, *e*, *[fe-rank*, *fe-dimension-vector*, *format]*

Operands

front-end-array A front-end array (possibly multidimensional) of complex data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers specifying NEWS axes.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(s + e + 1)$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Complex number values are copied from the Connection Machine processors to the specified *front-end-array.*

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank.*

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if $cm\text{-}axis\text{-}vector[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, from C or Fortran, one of the following predefined complex *format* values may be used: CM_complex_float_single or CM_complex_float_double. For complex data types in C, two front-end elements are used for each Connection Machine element.

When calling Paris from Lisp, the *format* parameter is a keyword argument; for complex transfers, only arrays of type t may be used.

Definition For all i such that $0 \leq i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

let $s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$

let $k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$

$\textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}} \leftarrow source[k_i]$

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
     let $k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$
     $\textit{front-end-array}_{\textit{offset-vector}_0+s_0, \textit{offset-vector}_1+s_1, \ldots, \textit{offset-vector}_{rank-1}+s_{rank-1}}$
      $\leftarrow source[k_{s_0,s_1,\ldots,s_{rank-1}}]$

F-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as floating-point numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:f-read-from-news-array-1L *front-end-array*, *fe-offset-vector*, *cm-start-vector*, *cm-end-vector*, *cm-axis-vector*, *source*, *s*, *e*, *[fe-rank*, *fe-dimension-vector*, *format]*

Operands

front-end-array A front-end array (possibly multidimensional) of floating-point data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Floating-point number values are transferred from the Connection Machine processors to the specified *array.*

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank.*

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined floatingpoint *format* values may be used. These are CM_float_single or CM_float_double from C or Fortran, and :float-single or :float-double from Lisp.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type and size of the Connection Machine field.

Definition For all i such that $0 \leq i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$\textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}} \leftarrow source[k_i]$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
  for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
    for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}} \leftarrow source[k_{s_0, s_1, \ldots, s_{rank-1}}]$$

S-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as signed integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:s-read-from-news-array-1L *front-end-array*, *fe-offset-vector*, *cm-start-vector*, *cm-end-vector*, *cm-axis-vector*, *source*, *len*, *[fe-rank*, *fe-dimension-vector*, *format]*

Operands

front-end-array A front-end array (possibly multidimensional) of signed integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes.

source The field ID of the signed integer source field.

len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Signed integer values are transferred from the Connection Machine processors to the specified *array*.

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

When calling Paris from Lisp, the array may be either a general array (of type t) containing signed integers, or a specialized integer-element array (such as an array of type (unsigned-byte 8)).

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank*.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined signed *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined signed format keywords are :8-bit, :16-bit, :32-bit, :2-bit-packed, and :4-bit-packed.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type and size of the Connection Machine field.

Definition For all i such that $0 \leq i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$\textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}} \leftarrow source[k_i]$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0+s_0, offset_1+s_1, \ldots, offset_{rank-1}+s_{rank-1}} \leftarrow source[k_{s_0,s_1,\ldots,s_{rank-1}}]$$

U-READ-FROM-NEWS-ARRAY

Copies a field within a set of processors forming a subarray of the NEWS grid into a subarray (of the same shape) of an array in the memory of the front end. Both the source and destination values are treated as unsigned integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:u-read-from-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, source, len, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of unsigned integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes.

source The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of the NEWS grid into a similarly shaped subblock of an array in the front end. Unsigned integer values are transferred from the Connection Machine processors to the specified *array*.

The *source* parameter specifies the memory address within each processor of the field to be copied.

The *front-end-array* parameter specifies the front-end destination array into which one element from each processor specified by *source* is copied.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the source field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays of length *fe-rank*.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) The front-end array is filled in row major order. That is, the last dimension varies fastest. When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element to receive Connection Machine data. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to copy to the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to copy to the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if $cm\text{-}axis\text{-}vector[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined unsigned *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_1_bit_packed, CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined unsigned format keywords are :8-bit, :16-bit, :32-bit, :1-bit-packed, :2-bit-packed, and **:4-bit-packed**.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the CM field.

Definition For all i such that $0 \leq i < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$\textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}} \leftarrow source[k_i]$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
  for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
    for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$\textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}} \leftarrow source[k_{s_0, s_1, \ldots, s_{rank-1}}]$$

C-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as a complex number and returns it to the front end.

Formats result ← CM:c-read-from-processor-1L *send-address-value*, *source*, *len*

Operands

send-address-value An immediate operand, the send address of a single particular processor.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *source* field. The total length of an operand in this format is $2(s + e + 1)$.

Result A complex number, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as a floating-point number to the front end.

F-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as a floating-point number and returns it to the front end.

Formats result ← CM:f-read-from-processor-1L *send-address-value*, *source*, *s*, *e*

Operands

send-address-value	An immediate operand, the send address of a single particular processor.
source	The field ID of the floating-point source field.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Result A floating-point number, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as a floating-point number to the front end.

S-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as a signed integer and returns it to the front end.

Formats result ← CM:s-read-from-processor-1L *send-address-value, source, len*

Operands

send-address-value An immediate operand, the send address of a single particular processor.

source The field ID of the signed integer source field.

len The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Result A signed integer, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as a signed integer to the front end.

U-READ-FROM-PROCESSOR

Reads the source field of a single specified processor as an unsigned integer and returns it to the front end.

Formats result ← CM:u-read-from-processor-1L *send-address-value*, *source*, *len*

Operands

send-address-value An immediate operand, the send address of a single particular processor.

source The field ID of the unsigned integer source field.

len The length of the *source* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Result An unsigned integer, the contents of the *source* field in the specified virtual processor.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *source*[*send-address-value*] to front end

The *source* field of the processor whose send address is the immediate operand *send-address-value* is read and returned as an unsigned integer to the front end.

C-RECIPROCAL

Calculates the reciprocal of a complex number.

Formats CM:c-reciprocal-1-1L *dest/source*, *s*, *e*
CM:c-reciprocal-2-1L *dest*, *source*, *s*, *e*

Operands
dest The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating point overflow occurs; otherwise it is unaffected.

test-flag is set if divistion by zero occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
$dest[k] \leftarrow \frac{1}{source[k]}$

A reciprocal of the complex *source* field is place in the complex *dest* field.

REDUCE-WITH-C-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the complex source fields from all the selected processors in that scan class.

Formats CM:reduce-with-c-add-1L *dest*, *source*, *axis*, *s*, *e*, *to-coordinate*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s+e+1)$.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ 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

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \mathit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36 of the *Paris Reference Manual*.

See section 5.16 beginning on page 34 for a general description of reduce operations. The CM:reduce-with-c-add operation combines *source* fields by performing complex addition.

The operation CM:reduce-with-c-add-1L differs from CM:spread-with-c-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-F-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the floating-point source fields from all the selected processors in that scan class.

Formats CM:reduce-with-f-add-1L *dest*, *source*, *axis*, *s*, *e*, *to-coordinate*

Operands

dest — The field ID of the floating-point destination field.

source — The field ID of the floating-point source field.

axis — An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

to-coordinate — An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$
    if $extract\text{-}news\text{-}coordinate(g, axis, k) = to\text{-}coordinate$ then

$$dest[k] \leftarrow \left(\sum_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-f-add operation combines *source* fields by performing floating-point addition.

The operation CM:reduce-with-f-add-1L differs from CM:spread-with-f-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-S-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the signed integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-s-add-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-s-add operation combines *source* fields by performing signed integer addition.

The operation CM:reduce-with-s-add-1L differs from CM:spread-with-s-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-U-ADD

Within each scan class one particular processor (if it is selected) receives the sum of the unsigned integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-u-add-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$
    if $extract\text{-}news\text{-}coordinate(g, axis, k) = to\text{-}coordinate$ then

$$dest[k] \leftarrow \left(\sum_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-u-add operation combines *source* fields by performing unsigned integer addition.

The operation CM:reduce-with-u-add-1L differs from CM:spread-with-u-add-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-COPY

Within each scan class one particular processor (if it is selected) receives a copy of the source value from a particular value within its scan subclass.

Formats CM:reduce-with-copy-1L *dest*, *source*, *axis*, *len*, *to-coordinate*, *from-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

from-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class is to be read.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $c = \textit{deposit-news-coordinate}(g, k, \textit{axis}, \textit{from-coordinate})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then
      $\textit{dest}[k] \leftarrow \textit{source}[c]$

where *deposit-news-coordinate* is as defined on page 40.

See section 5.20 on page 42 for a general description of reduce operations.

REDUCE-WITH-LOGAND

Within each scan class one particular processor (if it is selected) receives the bitwise logical AND of the source fields from all the selected processors in that scan class.

Formats CM:reduce-with-logand-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\bigwedge_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-logand operation combines *source* fields by performing bitwise logical AND operations.

The operation CM:reduce-with-logand-1L differs from CM:spread-with-logand-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-LOGIOR

Within each scan class one particular processor (if it is selected) receives the bitwise logical inclusive OR of the source fields from all the selected processors in that scan class.

Formats CM:reduce-with-logior-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ 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

$$dest[k] \leftarrow \left(\bigvee_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-logior operation combines *source* fields by performing bitwise logical inclusive OR operations.

The operation CM:reduce-with-logior-1L differs from CM:spread-with-logior-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-LOGXOR

Within each scan class one particular processor (if it is selected) receives the bitwise logical exclusive OR of the source fields from all the selected processors in that scan class.

Formats CM:reduce-with-logxor-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

- *dest* — The field ID of the destination field.
- *source* — The field ID of the source field.
- *axis* — An unsigned integer immediate operand to be used as the number of a NEWS axis.
- *len* — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
- *to-coordinate* — An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ 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

$$dest[k] \leftarrow \left(\bigoplus_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations.

The operation CM:reduce-with-logxor-1L differs from CM:spread-with-logxor-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-F-MAX

Within each scan class one particular processor (if it is selected) receives the largest of the floating-point source fields from all the selected processors in that scan class.

Formats CM:reduce-with-f-max-1L *dest*, *source*, *axis*, *s*, *e*, *to-coordinate*

Operands
- *dest* — The field ID of the floating-point destination field.
- *source* — The field ID of the floating-point source field.
- *axis* — An unsigned integer immediate operand to be used as the number of a NEWS axis.
- *s*, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.
- *to-coordinate* — An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then
      $\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m]\right)$
where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-f-max operation combines *source* fields by performing an floating-point maximum operation.

The operation CM:reduce-with-f-max-1L differs from CM:spread-with-f-max-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-S-MAX

Within each scan class one particular processor (if it is selected) receives the largest of the signed integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-s-max-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-s-max operation combines *source* fields by performing a signed integer maximum operation.

The operation CM:reduce-with-s-max-1L differs from CM:spread-with-s-max-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-U-MAX

Within each scan class one particular processor (if it is selected) receives the largest of the unsigned integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-u-max-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands *dest* The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-u-max operation combines *source* fields by performing an unsigned integer maximum operation.

The operation CM:reduce-with-u-max-1L differs from CM:spread-with-u-max-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-F-MIN

Within each scan class one particular processor (if it is selected) receives the smallest of the floating-point source fields from all the selected processors in that scan class.

Formats CM:reduce-with-f-min-1L *dest*, *source*, *axis*, *s*, *e*, *to-coordinate*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $context\text{-}flag[k] = 1$ 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

$$dest[k] \leftarrow \left(\min_{m \in C_k} source[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-f-min operation combines *source* fields by performing an floating-point minimum operation.

The operation CM:reduce-with-f-min-1L differs from CM:spread-with-f-min-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-S-MIN

Within each scan class one particular processor (if it is selected) receives the smallest of the signed integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-s-min-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    if $\textit{extract-news-coordinate}(g, \textit{axis}, k) = \textit{to-coordinate}$ then

$$\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-s-min operation combines *source* fields by performing a signed integer minimum operation.

The operation CM:reduce-with-s-min-1L differs from CM:spread-with-s-min-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

REDUCE-WITH-U-MIN

Within each scan class one particular processor (if it is selected) receives the smallest of the unsigned integer source fields from all the selected processors in that scan class.

Formats CM:reduce-with-u-min-1L *dest*, *source*, *axis*, *len*, *to-coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

to-coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class, if any, is to receive the result.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $C_k = \mathit{scan\text{-}subclass}(g, k, \mathit{axis})$
    if $\mathit{extract\text{-}news\text{-}coordinate}(g, \mathit{axis}, k) = \mathit{to\text{-}coordinate}$ then
      $\mathit{dest}[k] \leftarrow \left(\min_{m \in C_k} \mathit{source}[m]\right)$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of reduce operations. The CM:reduce-with-u-min operation combines *source* fields by performing an unsigned integer minimum operation.

The operation CM:reduce-with-u-min-1L differs from CM:spread-with-u-min-1L only in that the result is stored in (at most) one processor of the scan class rather than in all selected processors of the scan class.

F-REM

The remainder from dividing one floating-point source value by another is placed in the destination field.

Formats

CM:f-rem-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:f-rem-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:f-rem-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:f-rem-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field. This is the quotient.

source1 The field ID of the floating-point first source field. This is the dividend.

source2 The field ID of the floating-point second source field. This is the divisor.

source2-value A floating-point immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *test-flag* is set if division by zero occurs; otherwise it is cleared.

overflow-flag is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *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 = ⌈v⌉
    else if even(⌊v⌋) then
      let n = ⌊v⌋
    else
      let n = ⌈v⌉
    dest[k] ← source1[k] − source2[k] × n
  else
    dest[k] ← ⟨unpredictable⟩
    test-flag[k] ← 1
  if ⟨overflow occurred in processor k⟩ then overflow-flag[k] ← 1
```

The remainder from the *source1* operand when divided by the *source2* operand is calculated treating both as floating-point numbers. The result is stored into memory. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-REM

The remainder from the truncating division of one signed integer by another is placed in the destination field. Overflow is also computed.

Formats

CM:s-rem-2-1L	*dest/source1, source2, len*
CM:s-rem-3-1L	*dest, source1, source2, len*
CM:s-rem-constant-2-1L	*dest/source1, source2-value, len*
CM:s-rem-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the signed integer remainder field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source2*[*k*] = 0 then
      dest[*k*] ← ⟨unpredictable⟩
    else
      $dest[k] \leftarrow sign(source1[k]) \times \left(|source1[k]| - |source2[k]| \times \left\lfloor \frac{|source1[k]|}{|source2[k]|} \right\rfloor \right)$
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1
    else *overflow-flag*[*k*] ← 0

The remainder resulting from the truncating division of the signed integer *source1* by the signed integer *source2* operand is stored into the *dest* field. The result always has the same sign as the *source1* operand. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The value of the destination is unpredictalbe if the divisor is zero.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-REM

The remainder from the truncating division of one unsigned integer by another is placed in the destination field. Overflow is also computed.

Formats

CM:u-rem-2-1L	*dest/source1, source2, len*
CM:u-rem-3-1L	*dest, source1, source2, len*
CM:u-rem-constant-2-1L	*dest/source1, source2-value, len*
CM:u-rem-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest The field ID of the unsigned integer remainder field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *test-flag* is set if divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if *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 remainder resulting from the truncating division of the unsigned integer *source1* by the unsigned integer *source2* operand is stored into the *dest* field. For unsigned integers this is of course the same as the mod operation.

The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The value of the destination is unpredictable if the divisor is zero.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

REMOVE-FIELD-ALIAS

Removes the specified alias field ID from the field to which it refers, leaving the field intact.

Formats CM:remove-field-alias *alias-id*

Operands *alias-id* An alias field ID. This must be an alias field ID returned by CM:make-field-alias.

Context This operation is unconditional. It does not depend on the *context-flag*.

Removing an alias field ID does not affect the memory field to which it refers.

F-F-ROUND

Rounds each source field value to the nearest integer value and stores the result as a floating-point number in the destination field.

Formats CM:f-f-round-1-1L *dest/source*, *s*, *e*
CM:f-f-round-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow sign(source) \times round(source[k])$

The *source* field, treated as a floating-point number, is rounded to the nearest integer and the result is stored in the *dest* field as a floating-point number.

If the *source* field value is exactly midway between two integers, then it is rounded to the even integer.

S-ROUND

The quotient of two signed integer source values, rounded to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:s-round-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:s-round-2-1L	*dest/source1, source2, len*
CM:s-round-3-1L	*dest, source1, source2, len*
CM:s-round-constant-2-1L	*dest/source1, source2-value, len*
CM:s-round-constant-3-1L	*dest, source1, source2-value, len*

Operands

dest — The field ID of the signed integer quotient field.

source1 — The field ID of the signed integer dividend field.

source2 — The field ID of the signed integer divisor field.

source2-value — A signed integer immediate operand to be used as the second source.

len — The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen — The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 — The length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 — The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
let $v = \frac{source1[k]}{source2[k]}$
if $v > \left\lfloor v + \frac{1}{2} \right\rfloor$ then
$dest[k] \leftarrow \lfloor v \rfloor$
else if $v < \left\lfloor v + \frac{1}{2} \right\rfloor$ then
$dest[k] \leftarrow \lceil v \rceil$
else if $even(\lfloor v \rfloor)$ then
$dest[k] \leftarrow \lfloor v \rfloor$
else
$dest[k] \leftarrow \lceil v \rceil$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The signed integer *source1* operand is divided by the signed integer *source2* operand. The mathematical quotient, rounded to the nearest integer (or to whichever of two equally near neighbors is even) is stored into the signed integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

S-F-ROUND

Converts floating-point source field values to signed integer values by rounding to the nearest integer.

Formats CM:s-f-round-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the floating-point source field.
len	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    let v = *source*[k]
    if $v > \left\lfloor v + \frac{1}{2} \right\rfloor$ then
      dest[k] $\leftarrow \lfloor v \rfloor$
    else if $v < \left\lfloor v + \frac{1}{2} \right\rfloor$ then
      dest[k] $\leftarrow \lceil v \rceil$
    else if *even*($\lfloor v \rfloor$) then
      dest[k] $\leftarrow \lfloor v \rfloor$
    else
      dest[k] $\leftarrow \lceil v \rceil$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] $\leftarrow 1$

The *source* field, treated as a floating-point number, is rounded to the nearest integer (to the nearest even integer if its value is equal to an integer plus $\frac{1}{2}$). The result is stored into the *dest* field as a signed integer.

U-ROUND

The quotient of two unsigned integer source values, rounded to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:u-round-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-round-2-1L	*dest*/*source1*, *source2*, *len*
CM:u-round-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-round-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:u-round-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 The length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if the divisor is zero; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then

$$\text{let } v = \frac{source1[k]}{source2[k]}$$
if $v > \left\lfloor v + \frac{1}{2} \right\rfloor$ then
  $dest[k] \leftarrow \lfloor v \rfloor$
else if $v < \left\lfloor v + \frac{1}{2} \right\rfloor$ then
  $dest[k] \leftarrow \lceil v \rceil$
else if $even(\lfloor v \rfloor)$ then
  $dest[k] \leftarrow \lfloor v \rfloor$
else
  $dest[k] \leftarrow \lceil v \rceil$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The unsigned integer *source1* operand is divided by the unsigned integer *source2* operand. The mathematical quotient, rounded to the nearest integer (or to whichever of two equally near neighbors is even) is stored into the unsigned integer memory field *dest*.

The various operand formats allow the second source operand to be either a memory field or a constant; in some cases the destination field initially contains one source operand.

The *overflow-flag* and *test-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-F-ROUND

Converts the floating-point source field values to unsigned integer values, which are stored in the destination field.

Formats CM:u-f-round-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the floating-point source field.
len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *dest* > $\lfloor source \rfloor$ then
      dest ← $\lfloor source \rfloor$
    else if *dest* < $\lfloor source \rfloor$ then
      dest ← $\lceil source \rceil$
    else if *even*($\lfloor source \rfloor$) then
      dest ← $\lfloor source \rfloor$
    else
      dest ← $\lceil source \rceil$
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

The *source* field, treated as a floating-point number, is rounded to the nearest integer (to the nearest even integer if its value is equal to an integer plus $\frac{1}{2}$), which is stored into the *dest* field as an unsigned integer.

F-S-SCALE

In each selected processor, multiplies a floating-point number by a specified power of two and stores the result in the destination.

Formats

CM:f-s-scale-2-2L	*dest/source1, source2, slen2, s, e*
CM:f-s-scale-3-2L	*dest, source1, source2, slen2, s, e*
CM:f-s-scale-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-s-scale-constant-3-1L	*dest, source1, source2-value, s, e*

Operands

dest — The field ID of the floating-point destination field.

source1 — The field ID of the floating-point first source field. This is the quantity to be scaled.

source2 — The field ID of the signed integer second source field. This is the base-2 logarithm of the scale factor.

source2-value — A signed integer immediate operand to be used as the second source.

s, e — The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 — The length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap — The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags — *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context — This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    $dest[k] \leftarrow \left\lfloor source1[k] \times 2^{source2[k]} \right\rfloor$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The operand *source1* is scaled by the power of two specified by *source2*. (This is faster than an equivalent multiplication by a power of two.)

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

F-U-SCALE

Multiplies a floating-point number by a specified power of two and stores the result into the destination.

Formats

CM:f-u-scale-2-2L	*dest*/*source1*, *source2*, *slen2*, *s*, *e*
CM:f-u-scale-3-2L	*dest*, *source1*, *source2*, *slen2*, *s*, *e*
CM:f-u-scale-constant-2-1L	*dest*/*source1*, *source2-value*, *s*, *e*
CM:f-u-scale-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source1 — The field ID of the floating-point first source field. This is the quantity to be scaled.

source2 — The field ID of the unsigned integer second source field. This is the base-2 logarithm of the scale factor.

source2-value — An unsigned integer immediate operand to be used as the second source.

s, *e* — The significand and exponent lengths for the *dest* and *source1* fields. The total length of an operand in this format is $s + e + 1$.

slen2 — The length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do

if *context-flag*[*k*] = 1 then

$dest[k] \leftarrow \lfloor source1[k] \times 2^{source2[k]} \rfloor$

if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

The operand *source1* is scaled by the power of two specified by *source2*. (This is faster than an equivalent multiplication by a power of two.)

The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand.

SCAN-WITH-C-ADD

The destination field in every selected processor receives the sum of the complex source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-c-add-1L *dest, source, axis, s, e, direction, inclusion, smode, sbit*

Operands

dest	The field ID of the complex destination field.
source	The field ID of the complex source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
s, e	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s+e+1)$.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $context\text{-}flag[k] = 1$ then
  let $g = geometry(current\text{-}vp\text{-}set)$
  let $S_k = scan\text{-}subset(g, k, axis, direction, inclusion, smode, sbit)$
  if $|S_k| = 0$ then
   $dest[k] \leftarrow 0$
  else
   $$dest[k] \leftarrow \left(\sum_{m \in S_k} source[m] \right)$$

where *scan-subset* is as defined on page 36 of the *Paris Reference Manual.*

See the section beginning on 34 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-c-add operation combines *source* fields by performing complex addition. If the scan subset for a selected processor is empty, then the complex value +0.0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-ADD

The destination field in every selected processor receives the sum of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-f-add-1L *dest*, *source*, *axis*, *s*, *e*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then

let $g = \textit{geometry}(\textit{current-vp-set})$

let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$

if $|S_k| = 0$ then

$\textit{dest}[k] \leftarrow 0$

else

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-add operation combines *source* fields by performing floating-point addition. If the scan subset for a selected processor is empty, then the floating-point value +0.0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-S-ADD

The destination field in every selected processor receives the sum of the signed integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-s-add-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $S_k = scan\text{-}subset(g, k, axis, direction, inclusion, smode, sbit)$
    if $|S_k| = 0$ then
      $dest[k] \leftarrow 0$
    else

$$dest[k] \leftarrow \left(\sum_{m \in S_k} source[m] \right)$$

where *scan-subset* is as defined on page 45.

SCAN-WITH-ADD

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-s-add operation combines *source* fields by performing signed integer addition. If the scan subset for a selected processor is empty, then the signed integer value 0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-U-ADD

The destination field in every selected processor receives the sum of the unsigned integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-u-add-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $S_k = scan\text{-}subset(g, k, axis, direction, inclusion, smode, sbit)$
    if $|S_k| = 0$ then
      $dest[k] \leftarrow 0$
    else

$$dest[k] \leftarrow \left(\sum_{m \in S_k} source[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-u-add operation combines *source* fields by performing unsigned integer addition. If the scan subset for a selected processor is empty, then the unsigned integer value 0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-COPY

The destination field in every selected processor receives the *first* source field from the processors below or above it in some ordering of the processors.

Formats CM:scan-with-copy-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the destination field.
source	The field ID of the source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if $\textit{context-flag}[k] = 1$ then
  let $g = \textit{geometry}(\textit{current-vp-set})$
  let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
  if $|S_k| = 0$ then
    $\textit{dest}[k] \leftarrow 000\ldots000$
  else
    case *direction* of
      :upward : let $m' = \min_{m \in S_k} m$
      :downward : let $m' = \max_{m \in S_k} m$
    $\textit{dest}[k] \leftarrow \textit{source}[m']$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-copy operation stores into each processor *k* the *source* field value from the first processor in the scan subset for processor *k* (where "first" means the processor with lowest address for an upward scan, or with highest address for a downward scan). Generally speaking, the net effect is to propagate a value from the first processor in a group to all the other processors in the group, although variations on this effect are provided by the various possibilities for the *inclusion* and *smode* arguments.

If the scan subset for a selected processor is empty, then the *dest* field for that processor is set to all zero bits. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-LOGAND

The destination field in every selected processor receives the bitwise logical AND of the source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-logand-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow$ 111...111
    else
      $$\textit{dest}[k] \leftarrow \left(\bigwedge_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

SCAN-WITH-LOGAND

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-logand operation combines *source* fields by performing bitwise logical AND operations. If the scan subset for a selected processor is empty, then the unsigned integer value $-2^{\text{len}} - 1$ (all ones) is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-LOGIOR

The destination field in every selected processor receives the bitwise logical inclusive OR of the source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-logior-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ 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

$$dest[k] \leftarrow \left(\bigvee_{m \in S_k} source[m] \right)$$

where *scan-subset* is as defined on page 45.

SCAN-WITH-LOGIOR

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-logior operation combines *source* fields by performing bitwise logical inclusive OR operations. If the scan subset for a selected processor is empty, then the unsigned integer value 0 (all zero bits) is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-LOGXOR

The destination field in every selected processor receives the bitwise logical exclusive OR of the source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-logxor-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 000\ldots000$
    else

$$\textit{dest}[k] \leftarrow \left(\bigoplus_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

SCAN-WITH-LOGXOR

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations. If the scan subset for a selected processor is empty, then the unsigned integer value 0 (all zero bits) is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-MAX

The destination field in every selected processor receives the largest of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-f-max-1L *dest*, *source*, *axis*, *s*, *e*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest — The field ID of the floating-point destination field.

source — The field ID of the floating-point source field.

axis — An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

direction — Either :upward or :downward.

inclusion — Either :exclusive or :inclusive.

smode — Either :none, :start-bit, or :segment-bit.

sbit — The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ 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
      $\mathit{dest}[k] \leftarrow -\infty$
    else
      $\mathit{dest}[k] \leftarrow \left(\max_{m \in S_k} \mathit{source}[m]\right)$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-max operation combines *source* fields by performing an floating-point maximum operation. If the scan subset for a selected processor is empty, then the floating-point value $-\infty$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-S-MAX

The destination field in every selected processor receives the largest of the signed integer source fields from processors below or above it in some ordering of the processors.

Formats

CM:scan-with-s-max-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest — The field ID of the signed integer destination field.

source — The field ID of the signed integer source field.

axis — An unsigned integer immediate operand to be used as the number of a NEWS axis.

len — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

direction — Either :upward or :downward.

inclusion — Either :exclusive or :inclusive.

smode — Either :none, :start-bit, or :segment-bit.

sbit — The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap

The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context

This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ 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
      $\textit{dest}[k] \leftarrow -2^{len-1}$
    else
      $\textit{dest}[k] \leftarrow \left(\max_{m \in S_k} \textit{source}[m]\right)$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-s-max operation combines *source* fields by performing a signed integer maximum operation. If the scan subset for a selected processor is empty, then the signed integer value -2^{len-1} is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-U-MAX

The destination field in every selected processor receives the largest of the unsigned integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-u-max-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 0$
    else
      $$\textit{dest}[k] \leftarrow \left(\max_{m \in S_k} \textit{source}[m]\right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-u-max operation combines *source* fields by performing an unsigned integer maximum operation. If the scan subset for a selected processor is empty, then the unsigned integer value 0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-MIN

The destination field in every selected processor receives the smallest of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-f-min-1L *dest*, *source*, *axis*, *s*, *e*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest	The field ID of the floating-point destination field.
source	The field ID of the floating-point source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.
direction	Either :upward or :downward.
inclusion	Either :exclusive or :inclusive.
smode	Either :none, :start-bit, or :segment-bit.
sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    let g = *geometry*(*current-vp-set*)
    let S_k = *scan-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 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-min operation combines *source* fields by performing an floating-point minimum operation. If the scan subset for a selected processor is empty, then the floating-point value $+\infty$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-S-MIN

The destination field in every selected processor receives the smallest of the signed integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-s-min-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

  if $\textit{context-flag}[k] = 1$ 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

      $\textit{dest}[k] \leftarrow 2^{len-1} - 1$

    else

      $\textit{dest}[k] \leftarrow \left(\min_{m \in S_k} \textit{source}[m]\right)$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-s-min operation combines *source* fields by performing a signed integer minimum operation. If the scan subset for a selected processor is empty, then the signed integer value $2^{len-1} - 1$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-U-MIN

The destination field in every selected processor receives the smallest of the unsigned integer source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-u-min-1L *dest*, *source*, *axis*, *len*, *direction*, *inclusion*, *smode*, *sbit*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

direction Either :upward or :downward.

inclusion Either :exclusive or :inclusive.

smode Either :none, :start-bit, or :segment-bit.

sbit The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  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
      $\textit{dest}[k] \leftarrow 2^{len} - 1$
    else
      $\textit{dest}[k] \leftarrow \left(\min_{m \in S_k} \textit{source}[m]\right)$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-u-min operation combines *source* fields by performing an unsigned integer minimum operation. If the scan subset for a selected processor is empty, then the unsigned integer value $2^{len} - 1$ is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SCAN-WITH-F-MULTIPLY

The destination field in every selected processor receives the product of the floating-point source fields from processors below or above it in some ordering of the processors.

Formats CM:scan-with-f-multiply-1L *dest, source, axis, s, e, direction, inclusion, smode, sbit*

Operands	*dest*	The field ID of the floating-point destination field.
	source	The field ID of the floating-point source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	s, e	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.
	direction	Either :upward or :downward.
	inclusion	Either :exclusive or :inclusive.
	smode	Either :none, :start-bit, or :segment-bit.
	sbit	The field ID of the segment bit or start bit (a one-bit field). If *smode* is :none then this may be CM:*no-field*.

Overlap The fields *source* and *sbit* may overlap in any manner. However, the *sbit* field must not overlap the *dest* field, and the field *source* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $S_k = \textit{scan-subset}(g, k, \textit{axis}, \textit{direction}, \textit{inclusion}, \textit{smode}, \textit{sbit})$
    if $|S_k| = 0$ then
      $\textit{dest}[k] \leftarrow 1$
    else

$$\textit{dest}[k] \leftarrow \left(\prod_{m \in S_k} \textit{source}[m] \right)$$

where *scan-subset* is as defined on page 45.

See section 5.20 on page 42 for a general description of scan operations and the effect of the *axis*, *direction*, *inclusion*, *smode*, and *sbit* operands.

The CM:scan-with-f-multiply operation combines *source* fields by performing floating-point multiplication. If the scan subset for a selected processor is empty, then the floating-point value 1.0 is stored in the *dest* field for that processor. Note that this can occur only when the *inclusion* argument is :exclusive.

SEND

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. If a processor receives more than one message, then the message data received by that processor will be unpredictable.

Formats CM:send-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is stored into the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else if $|S_k| = 1$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \textit{source}[\textit{choice}(S_k)]$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \langle\text{undefined}\rangle$

where the *choice* function arbitrarily but deterministically chooses an element from a set.

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d. Note that, although the *send-address* operand is a field in the current VP set, its value must specify a valid send address for *dest*, which may belong to a different VP set.

The CM:send operation combines multiple incoming messages in an unpredictable manner. This operation may be used when the programmer can guarantee that no processor will receive more than one message. Using this operation when it is appropriate may speed message delivery. The destination area need not be prepared.

SEND-ASET32-U-ADD

Sends a message from every selected processor to a specified destination processor and stores it there, as if by aset32, in an array. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected. All incoming messages are combined with the destination array element using unsigned integer addition.

Formats

CM:send-aset32-u-add-2L *array*, *send-address*, *source*, *index*, *slen*, *index-len*, *index-limit*

Operands

array The field ID of the destination array field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *slen*.

slen The length of the *source* field. This must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of the destination array.

Overlap

The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context

This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is combined with the field indicated by *array* regardless of the *context-flag* of the receiving processor.

Definition

For every virtual processor k in the *current-vp-set* do
  let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
  for every processor k' in S_k do
    if $\textit{index}[k'] < \textit{index-limit}$ then
      let $r = \textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))$

let $m = \lfloor \frac{k}{r} \rfloor \bmod 32$
let $i = index[k']$
for all j such that $0 \le j < dlen$ do
  let $temp_k\langle j \rangle = array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \lfloor \frac{j}{32} \rfloor)\rangle$
let $sum_k = temp_k + source[k']$
for all j such that $0 \le j < dlen$ do
  $array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \lfloor \frac{j}{32} \rfloor)\rangle \leftarrow sum_k\langle j \rangle$
else
  ⟨error⟩

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into an array element within processor p_d. Note that in each case the array element to be modified in processor p_d is determined by the value of *index* within p_s, not the value within p_d.

The CM:send-aset32-u-add operation combines incoming messages with unsigned integer addition. To receive the sum of only the messages, the destination *array* should first be cleared in all processors that might receive a message.

SEND-ASET32-LOGIOR

Sends a message from every selected processor to a specified destination processor and stores it there, as if by aset32, in an array. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected. All incoming messages are combined with the destination array element using bitwise logical inclusive OR.

Formats CM:send-aset32-logior-2L *array, send-address, source, index, slen, index-len, index-limit*

Operands

array The field ID of the destination array field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *slen*.

slen The length of the *source* field. This must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of the destination array.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is combined with the field indicated by *array* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do
  let $S_k = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \}$
  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 $m = \lfloor \frac{k}{r} \rfloor \bmod 32$
let $i = index[k']$
for all j such that $0 \leq j < dlen$ do
  let $q = k - m \times r + (j \bmod 32) \times r$
  let $b = 32 \times (i + \lfloor \frac{j}{32} \rfloor)$
  $array[q]\langle b \rangle \leftarrow array[q]\langle b \rangle \vee source[k']\langle j \rangle$
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-logior operation combines incoming messages with a bitwise logical inclusive OR operation. To receive the logical inclusive OR of only the messages, the destination *array* should first be cleared in all processors that might receive a message.

SEND-ASET32-OVERWRITE

Sends a message from every selected processor to a specified destination processor and stores it there, as if by aset32, in an array. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected. If a processor receives more than one message destinated for the same array element, then one is stored in that array element and the rest are discarded.

Formats CM:send-aset32-overwrite-2L *array, send-address, source, index, slen, index-len, index-limit*

Operands

array The field ID of the destination array field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

index The field ID of the unsigned integer index into the array field. This is used as a per-processor index into *array*. It specifies portions of the *array* memory area in increments of *slen*.

slen The length of the *source* field. This must be a multiple of 32.

index-len The length of the *index* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for the *index*. This is taken as the extent of the destination array.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is combined with the field indicated by *array* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \}$

let $k' = \textit{choice}(S_k)$

if $\textit{index}[k'] < \textit{index-limit}$ then

let $r = \textit{geometry-total-vp-ratio}(\textit{geometry}(\textit{current-vp-set}))$

let $m = \left\lfloor \frac{k}{r} \right\rfloor \bmod 32$
let $i = index[k']$
for all j such that $0 \le j < dlen$ do
$array[k - m \times r + (j \bmod 32) \times r]\langle 32 \times (i + \left\lfloor \frac{j}{32} \right\rfloor)\rangle \leftarrow source[k']\langle j \rangle$
else
⟨error⟩

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into an array element within processor p_d. Note that in each case the array element to be modified in processor p_d is determined by the value of *index* within p_s, not the value within p_d.

The CM:send-aset32-overwrite operation will store one of the messages sent to a particular array element, discarding all other messages as well as the original contents of that array element in the receiving processor.

SEND-TO-NEWS

Each processor sends a message to a neighboring processor along a specified NEWS axis.

Formats CM:send-to-news-1L *dest, source, axis, direction, len*
CM:send-to-news-always-1L *dest, source, axis, direction, len*

Operands *dest* The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

direction Either :upward or :downward.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional, but whether data is copied depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is stored into the field indicated by *dest* regardless of the *context-flag* of the receiving processor.

Note that in the conditional case the storing of data depends only on the *context-flag* of the processor sending the data, not on the *context-flag* of the processor receiving the data.

Definition For every virtual processor k in the *current-vp-set* do
if (always or *context-flag*[k] = 1) then
let g = *geometry*(*current-vp-set*)
dest[*news-neighbor*(g, k, *axis*, *direction*)] ← *source*[k]

The *source* field in each processor is stored into the *dest* field of that processor's neighbor along the NEWS axis specified by *axis* in the direction specified by *direction*.

If *direction* is :upward then each processor stores data into the neighbor whose NEWS coordinate is one greater, with the processor whose coordinate is greatest storing data into the processor whose coordinate is zero.

If *direction* is :downward then each processor stores data into the neighbor whose NEWS coordinate is one less, with the processor whose coordinate is zero storing data into the processor whose coordinate is greatest.

SEND-TO-QUEUE32

Sends a message from every selected processor to a specified destination processor and stores it there, as if by aset32, in a queue. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors.

Formats CM:send-to-queue32-1L *dest*, *send-address*, *source*, *slen*, *index-limit*

Operands

dest The field ID of the queue field. The length of this field must accommodate 32 bits for the *queue.count* subfield, plus *index – limit* × *slen* bits for the *queue.elements* subfield, where *index-limit* is the number of queue elements in each processor.

send address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

slen The length of the *source* field. This is also the length of each queue element. It is currently restricted to 32 bits.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for a zero-based index into *queue.elements*. The value of this argument must be at least 1 and should never exceed the number of elements that can be stored in the queue.

Overlap The fields *send-address* and *source* may overlap in any manner. No overlap with the *dest* field is allowed.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is queued in the field indicated by *dest* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
let T_k be a sub-set of S_k where $|T_k| = \min(|S_k| + \textit{queue.count}, \textit{index-limit})$
for i from *queue.count* to $\textit{queue.count} + |T_k| - 1$ do
  $\textit{queue.elements}[i] \leftarrow T_k[i]$
$\textit{queue.count} \leftarrow \textit{queue.count} + |S_k|$

Note that if $(|S_k| + \textit{queue.count} > \textit{index-limit})$ then there is some choice in picking the elements of T_k.

The destination field is treated as two subfields: *queue.count* and *queue.elements*. *Queue.count* is 32 bits long and records the number of enqueued messages. *Queue.elements* stores the enqueued messages; it is formatted as a slicewise array (accessed using aref32 and aset32), and starts at an offset of 32 bits from the start of the destination field. Its length is a multiple of the message length: at least *index-limit* × *slen* and possibly greater.

The *index-limit* argument specifies the maximum number of elements that any processor's *queue.elements* subfield may accumulate. If any processor receives more messages than this specified number, the queue overflows and messages are lost. If a *queue.elements* subfield overflows, the *queue.count* subfield for that processor nonetheless accurately reflects the number of messages received.

For any given communication pattern, both the order of message queueing and the selection of messages preserved or discarded in case of queue overflow are deterministic. That is, the order and selection of enqueued messages can be predictably reproduced from one invocation to the next.

This determinism is especially important for applications that use successive CM:send-to-queue32-1L calls to send large data structures by breaking up them up into chunks of length *slen*. By holding the *send-address* argument constant, such applications can send successive chunks of *slen* bits each to corresponding queues.

To prepare an empty queue for a CM:send-to-queue-1L instruction, the *queue.count* subfield should be set to zero. From Lisp/Paris, this is done by executing the following code in the destination context:

```
(let   ((zeros (allocate-stack-field 32))
  (context-hold (allocate-stack-field 1)))
  (cm:move-constant-always zeros 0 32)
  (cm:store-context context-hold)
  (cm:set-context)
  (cm:aset32-2L zeros queue zeros 32 32 1)
  (cm:load-context context-hold)
)
```

The CM:send-to-queue32-1L operation is conditional on the context of the source field; the set of queues that will *receive* messages is independent of the currently active set. To zero the *queue.count* subfield in only those queues that are to receive messages, execute the following code in the source context:

```
(let   ((zeros (allocate-stack-field 32)))
  (cm:move-constant-always zeros 0 32)
  (cm:send-aset32-overwrite-2L queue dest zeros zeros 32 32 1)
)
```

After the CM:send-to-queue32 operation, the local count can be retrieved by executing the following code in the destination context:

```
(let   ((zeros (allocate-stack-field 32)))
  (count-field (allocate-stack-field 32))
  )
  (cm:move-constant-always zeros 0 32)
  (cm:aref32-2L count-field queue zeros 32 32 1)
)
```

The i^{th}) message can be retrieved from *queue.elements* by executing the following code in the destination context:

```
(let   ((index (allocate-stack-field 32))
  (data-field (allocate-stack-field message-length))
  )
  (cm:move-constant-always index i 32)
  (cm:aref32-2L data-field (+ 32 queue) index len 32 queue-size)
)
```

Note that *queue.elements* is offset from the queue field by 32 bits.

An artificially small queue size may be used by passing CM:send-to-queue-1L an index-limit value that is less than the number of elements of length slen that could be stored in the *queue.elements* portion of the destination field. If this is done, the queues will be partially filled. However, the correct queue size should always be used as the index-limit argument to CM:aref32-2L when reading elements from the queue.

SEND-WITH-C-ADD

Sends a message from every selected processor to a destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using complex addition.

Formats CM:send-with-c-add-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest The field ID of the complex destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

notify The field ID of the notification bit (a one-bit field).

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition Let $P = \{ m \mid 0 \le m \le \text{CM:*user-send-address-limit*} \}$
For every virtual processor k in *vp-set*(*dest*) do
  let $S_k = \{ m \mid m \in P \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
  if $|S_k| = 0$ then
    if $\textit{notify}[k] \not\equiv \text{CM:*no-field*}$ then $\textit{notify}[k] \leftarrow 0$
  else
    if $\textit{notify}[k] \not\equiv \text{CM:*no-field*}$ then $\textit{notify}[k] \leftarrow 1$
    $$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

SEND-WITH-ADD

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose absolute send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-c-add operation adds incoming messages to the *dest* field, treating all quantities as complex numbers. To receive the sum of only the messages, the destination area should initially be set to zero in all processors that might receive a message.

SEND-WITH-F-ADD

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using floating-point addition.

Formats CM:send-with-f-add-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest The field ID of the floating-point destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
if $|S_k| = 0$ then
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-f-add operation adds incoming messages together with the *dest* field as floating-point numbers. To receive the sum of only the messages, the destination area should first be set to zero in all processors that might receive a message.

SEND-WITH-S-ADD

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using signed integer addition.

Formats CM:send-with-s-add-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the signed integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-s-add operation adds incoming messages into the *dest* field as signed integers. Carry-out and arithmetic overflow are not detected. To receive the sum of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-U-ADD

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using unsigned integer addition.

Formats CM:send-with-u-add-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the unsigned integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the unsigned integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] + \left(\sum_{m \in S_k} \textit{source}[m] \right)$$

SEND-WITH-ADD

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-u-add operation adds incoming messages into the *dest* field as unsigned integers. Carry-out and arithmetic overflow are not detected. To receive the sum of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-LOGAND

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using bitwise logical AND.

Formats CM:send-with-logand-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap

The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context

This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] \wedge \left(\bigwedge_{m \in S_k} \textit{source}[m] \right)$$

SEND-WITH-LOGAND

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-logand operation will combine all messages and the original contents of the destination field with a bitwise logical AND operation. To receive the logical AND of only the messages, the destination area should first be set to all-ones in all processors that might receive a message.

SEND-WITH-LOGIOR

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using bitwise logical inclusive OR.

Formats CM:send-with-logior-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] \vee \left(\bigvee_{m \in S_k} \textit{source}[m] \right)$$

SEND-WITH-LOGIOR

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-logior operation combines incoming messages with a bitwise logical inclusive OR operation. To receive the logical inclusive OR of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-LOGXOR

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the destination field using bitwise logical exclusive OR.

Formats CM:send-with-logxor-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap

The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context

This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \textit{dest}[k] \oplus \left(\bigoplus_{m \in S_k} \textit{source}[m] \right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-logxor operation is similar but combines incoming messages with a bitwise logical EXCLUSIVE OR operation. To receive the logical EXCLUSIVE OR of only the messages, the destination area should first be cleared in all processors that might receive a message.

SEND-WITH-F-MAX

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a floating-point maximum operation.

Formats CM:send-with-f-max-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest The field ID of the floating-point destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

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 field should first be set to the smallest possible value: $-\infty$.

SEND-WITH-S-MAX

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a signed integer maximum operation.

Formats CM:send-with-s-max-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the signed integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \max\left(\textit{dest}[k], \max_{m \in S_k} \textit{source}[m]\right)$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-s-max operation combines incoming messages with the *dest* field using signed integer maximum operations. The *test-flag* is not affected by the maximum operation.

To receive the maximum of only the messages, the destination field should first be set to athe smallest possible value: -2^{len-1}.

SEND-WITH-U-MAX

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using an unsigned integer maximum operation.

Formats CM:send-with-u-max-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands
- *dest* The field ID of the unsigned integer destination field.
- *send-address* The field ID of the send address field. For each processor, this indicates to which processor a message is sent.
- *source* The field ID of the unsigned integer source field.
- *len* The length of the *dest* and *source* fields.
- *notify* The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$
  $\textit{dest}[k] \leftarrow \max\left(\textit{dest}[k], \max_{m \in S_k} \textit{source}[m]\right)$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-u-max operation combines incoming messages with the *dest* field using unsigned integer maximum operations. The *test-flag* is not affected by the maximum operation.

To receive the maximum of only the messages, the destination field should first be set to the smallest possible value: zero.

SEND-WITH-F-MIN

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a floating-point minimum operation.

Formats CM:send-with-f-min-1L *dest*, *send-address*, *source*, *s*, *e*, *notify*

Operands

dest The field ID of the floating-point destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{\, m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \,\}$

if $|S_k| = 0$ then

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$

else

if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 1$

$$\textit{dest}[k] \leftarrow \min\left(\textit{dest}[k], \min_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-f-min operation combines incoming messages with the *dest* field using floating-point minimum operations. The *test-flag* is not affected by the minimum operation.

To receive the minimum of only the messages, the destination field should first be set to the largest value possible: $+\infty$.

SEND-WITH-S-MIN

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using a signed integer minimum operation.

Formats CM:send-with-s-min-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the signed integer destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the signed integer source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \}$
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 $\textit{notify}[k] \leftarrow 1$
  $$\textit{dest}[k] \leftarrow \min\left(\textit{dest}[k], \min_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-s-min operation combines incoming messages with the *dest* field using signed integer minimum operations. The *test-flag* is not affected by the minimum operation.

To receive the minimum of only the messages, the destination field should first be set to the largest possible value: $2^{len-1} - 1$.

SEND-WITH-U-MIN

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. All incoming messages are combined with the *dest* field using an unsigned integer minimum operation.

Formats CM:send-with-u-min-1L *dest, send-address, source, len, notify*

Operands
- *dest* The field ID of the unsigned integer destination field.
- *send-address* The field ID of the send address field. For each processor, this indicates to which processor a message is sent.
- *source* The field ID of the unsigned integer source field.
- *len* The length of the *dest* and *source* fields.
- *notify* The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is combined with the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \}$
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 $\textit{notify}[k] \leftarrow 1$
  $$\textit{dest}[k] \leftarrow \min\left(\textit{dest}[k], \min_{m \in S_k} \textit{source}[m]\right)$$

For every selected processor p_s, a message *length* bits long is sent from that processor to the processor p_d whose send address is stored at location *send-address* in the memory of

processor p_s. The message is taken from the *source* field within processor p_s and is stored into the *dest* field within processor p_d.

The CM:send-with-u-min operation combines incoming messages with the *dest* field using unsigned integer minimum operations. The *test-flag* is not affected by the minimum operation.

To receive the minimum of only the messages, the destination field should first be set to the largest possible value: $2^{len} - 1$.

SEND-WITH-OVERWRITE

Sends a message from every selected processor to a specified destination processor. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors. Messages are all delivered to the same address within each receiving processor. If a processor receives more than one message, then one is delivered and the rest are discarded.

Formats CM:send-with-overwrite-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length of the *dest* and *source* fields.

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *send-address* and *source* may overlap in any manner. Similarly, the *send-address* and *dest* may overlap in any manner. However, it is forbidden for the *source* and *dest* to overlap.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is stored into the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \}$
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] ← 1

The destination memory bit is set within each selected processor.

SET-CONTEXT

Unconditionally makes all processors active.

Formats CM:set-context

Context This operation is unconditional.

Definition For every virtual processor k in the *current-vp-set* do
$context\text{-}flag[k] \leftarrow 1$

Within each processor, the context bit for that processor is unconditionally set.

SET-FIELD-ALIAS-VP-SET

Sets the VP set of the specified alias fieldID to the specified VP set.

Formats CM:set-field-alias-vp-set *alias-id*, *vp-set*

Operands	*alias-id*	An alias field ID. This must be an alias fieldID returned by CM:make-field-alias. This alias id need not be in the current VP set.
	vp-set	A VP set ID. This need not be the current VP set.
Context	This operation is unconditional. It does not depend on the *context-flag*.	

This function sets the VP set of *alias-id* to *vp-set*.

An error is signaled if the physical length of the aliased field is not exactly divisible by the VP ratio of *vp-set*. (See the definitions of CM:make-field-alias for more information about the physical length of an aliased field.)

SET-SAFETY-MODE

Formats CM:set-safety-mode *safety-mode*

Operands *safety-mode* An unsigned integer, the safety level. Currently only the values 0 and 1 are meaningful.

Context This operation is unconditional. It does not depend on the *context-flag*.

The safety mode is set to the specified value. A non-zero value indicates that the Paris interface should perform various extra error checks and consistency checks that may be helpful in detecting bugs in user programs. Of course, the price of these error checks is reduced execution speed.

SET-SYSTEM-LEDS-MODE

Formats CM:set-system-leds-mode *leds-mode*

Operands *leds-mode* Either :leds-off, :leds-on, :leds-throb, :leds-diagnostics, :leds-perfmon, :leds-sync, or :leds-blink-sync.

Context This operation is unconditional. It does not depend on the *context-flag*.

The lights on the front and back of the Connection Machine system cabinet can be controlled in a variety of ways. The **cm:set-system-leds-mode** operation selects what information will be displayed in the lights. If the specified *leds-mode* is :leds-off, then all the lights are turned off, and thereafter the user operations **cm:latch-leds** and **cm:latch-leds-always** may be used to control the lights. Other values for *leds-mode* select one of the system-supplied display modes. (The operations **cm:latch-leds** and **cm:latch-leds-always** may still be used when in a system-supplied display mode, but the user-specified pattern is unlikely to persist as it may be immediately altered by the system, depending on the mode.)

The names of the possible modes shown above are for the C/Paris and Fortran/Paris interfaces. Through an accident of history, the names for the leds modes are different in the Lisp/Paris interface:

C and Fortran	Lisp
CM_leds_off	nil
CM_leds_on	t
CM_leds_throb	:throb
CM_leds_diagnostics	:diagnostics
CM_leds_perfmon	:performance-monitor
CM_leds_sync	:synch
CM_leds_blink_sync	:blink-and-synch

C'est la vie.

SET-VP-SET

Declares a specified VP set to be current.

Formats CM:set-vp-set *vp-set-id*

Operands *vp-set-id* A VP set ID.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *current-vp-set* ← *vp-set-id*

The VP set specified by the *vp-set-id* becomes the current VP set. Most Paris operations implicitly operate within the virtual processors of the current VP set.

SET-VP-SET-GEOMETRY

Alters the geometry of an existing VP set.

Formats CM:set-vp-set-geometry *vp-set-id*, *geometry-id*

Operands	*vp-set-id*	A VP set ID.
	geometry-id	A geometry ID.
Context	This operation is unconditional. It does not depend on the *context-flag*.	

The VP set specified by the *vp-set-id* is altered so that its geometry is that specified by the *geometry-id*. The new geometry must have the same total number of elements (product of axis lengths) as the old geometry.

SET-flag

Sets a specified flag bit.

Formats CM:set-test
CM:set-overflow

Context This operation is conditional.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ 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.

S-S-SHIFT

Shifts a signed integer by an amount specified by a signed integer.

Formats

CM:s-s-shift-2-2L	*dest/source1, source2, dlen, slen2*
CM:s-s-shift-constant-3-2L	*dest, source1, source2-value, dlen, slen1*

Operands

dest The field ID of the signed integer destination field.

source1 The field ID of the signed integer first source field. This is the quantity to be shifted.

source2 The field ID of the signed integer second source field. This is the shift distance (positive for a left shift, negative for a right shift).

source2-value A signed integer immediate operand to be used as the second source. The same shift distance is applied to each *source1* value.

dlen The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-s-shift-2-2L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-s-shift-constant-3-2L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do

if *context-flag*$[k] = 1$ then

$dest[k] \leftarrow \left\lfloor source1[k] \times 2^{source2[k]} \right\rfloor$

if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

else *overflow-flag*$[k] \leftarrow 0$

The operand *source1* is shifted by the number of bit positions specified by *source2*, where a positive shift distance indicates a left shift (that is, a shift toward more significant bit positions) and a negative shift distance indicates a right shift (that is, a shift toward less significant bit positions). A left shift introduces zero bits into the vacated (least significant) bit positions; a right shift introduces copies of the sign bit into the vacated (most significant) bit positions. This operation is sometimes called an *arithmetic shift*.

The result is stored into the memory field *dest*. The various operand formats allow the second source operand to be either a memory field or a constant. In the non-constant version the destination field initially contains one source operand.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *dlen*.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

U-S-SHIFT

Shifts an unsigned integer by an amount specified by a signed integer.

Formats CM:u-s-shift-2-2L *dest/source1, source2, dlen, slen2*
CM:u-s-shift-constant-3-2L *dest, source1, source2-value, dlen, slen1*

Operands *dest* The field ID of the unsigned integer destination field.

source1 The field ID of the unsigned integer first source field. This is the quantity to be shifted.

source2 The field ID of the signed integer second source field. This is the shift distance (positive for a left shift, negative for a right shift.)

source2-value A signed integer immediate operand to be used as the second source. The same shift distance is applied to each *source1* value.

dlen The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-s-shift-2-2L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:u-s-shift-constant-3-2L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. However, the *source2* field must not overlap the *dest* field, and the field *source1* must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Flags *overflow-flag* is set if the result cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then
$dest[k] \leftarrow \lfloor source1[k] \times 2^{source2[k]} \rfloor$
if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1
else *overflow-flag*[*k*] ← 0

The operand *source1* is shifted by the number of bit positions specified by *source2*, where a positive shift distance indicates a left shift (that is, a shift toward more significant bit positions) and a negative shift distance indicates a right shift (that is, a shift toward less significant bit positions). Zero-valued bits are introduced into the vacated bit positions (least significant for a left shift, most significant for a right shift). This operation is sometimes called a *logical shift*.

The result is stored into the memory field *dest*. The various operand formats allow the second source operand to be either a memory field or a constant. In the non-constant version, the destination field initially contains one source operand.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *dlen*.

The *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

C-C-SIGNUM

The signum of the complex source field is stored in the complex destination field.

Formats CM:c-c-signum-1-1L *dest/source*, *s*, *e*
CM:c-c-signum-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow signum(source[k])$

The signum of a complex number is a complex number of the same phase but with unit magnitude, unless the numer is a complex zero, in which case the result is a complex zero.

F-F-SIGNUM

Determines whether the floating-point source field is negative, minus zero, plus zero, or positive and places the value `-1.0`, `+0.0`, -0.0, or `1.0` in the destination field accordingly.

Formats CM:f-f-signum-1-1L *dest/source*, *s*, *e*
CM:f-f-signum-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    if *source*[*k*] < 0 then *dest*[*k*] ← −1.0
    else if *source*[*k*] > 0 then *dest*[*k*] ← 1.0
    else *dest*[*k*] ← *source*[*k*]

The signum function of the *source* operand is placed in the *dest* operand. The result is `-1.0`, `-0.0`, `+0.0`, or `1.0` thus indicating whether the source value is negative, minus zero, plus zero, or positive, respectively. If the *source* operand is a NaN, then it is copied unchanged.

S-F-SIGNUM

Determines whether the floating-point source field is negative, zero, or positive and places the value -1, 0, or 1 in the destination field accordingly.

Formats CM:s-f-signum-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands	*dest*	The field ID of the signed integer destination field.
	source	The field ID of the floating-point source field.
	dlen	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap: The fields *dest* and *source* must not overlap in any manner.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if *context-flag*[k] = 1 then
  if *source*[k] < 0 then *dest*[k] ← −1
  else if *source*[k] > 0 then *dest*[k] ← 1
  else *dest*[k] ← 0

The signum function of the *source* operand is placed in the *dest* operand. The result is -1, 0, or 1 according to whether the source value is negative (but non-zero), zero (+0 or −0), or positive (but non-zero), respectively.

S-S-SIGNUM

Determines whether the signed integer source field is negative, zero, or positive and places the value -1, 0, or 1 in the destination field accordingly.

Formats

CM:s-s-signum-1-1L	*dest/source*, *len*
CM:s-s-signum-2-1L	*dest*, *source*, *len*
CM:s-s-signum-2-2L	*dest*, *source*, *dlen*, *slen*

Operands

dest — The field ID of the signed integer destination field.

source — The field ID of the signed integer source field.

len — The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen — The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen — The length of the *source* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    if $\textit{source}[k] < 0$ then $\textit{dest}[k] \leftarrow -1$
    else if $\textit{source}[k] > 0$ then $\textit{dest}[k] \leftarrow 1$
    else $\textit{dest}[k] \leftarrow 0$

The signum function of the *source* operand is placed in the *dest* operand. The result is -1, 0, or 1 according to whether the source value is negative, zero, or positive, respectively.

C-SIN

The sine of the complex source field is placed in the complex destination field.

Formats CM:c-sin-1-1L *dest*/*source*, *s*, *e*
CM:c-sin-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*[k] = 1 then
dest[k] ← sin *source*[k]
if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-SIN

Calculates the floating-point sine of the source field values and stores the result in the floating-point destination field.

Formats CM:f-sin-1-1L *dest/source*, *s*, *e*
CM:f-sin-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then
dest[*k*] ← sin *source*[*k*]

The sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

C-SINH

The hyperbolic sine of the complex source field is placed in the complex destination field.

Formats CM:c-sinh-1-1L *dest/source*, *s*, *e*
CM:c-sinh-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \sinh$ *source*$[k]$

The hyperbolic sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

F-SINH

Calculates the floating-point hyperbolic sine of the source field values and stores the result in the floating-point destination field.

Formats CM:f-sinh-1-1L *dest/source, s, e*
CM:f-sinh-2-1L *dest, source, s, e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, e The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
dest$[k] \leftarrow \sinh$ *source*$[k]$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The hyperbolic sine of the value of the *source* field is stored into the *dest* field.

Length Restriction: This transcendental function is computed in either standard single- or standard double-precision and then the result is moved into the destination, regardless of the declared size of the destination. Therefore use standard lengths only, such that $s = 23$ and $e = 8$ or $s = 52$ and $e = 11$.

SPREAD-FROM-PROCESSOR

A single source processor is specified. A copy of its source field value is spread to every (selected) processor in the destination field. Neither the destination nor the source field needs to be in the current VP set.

Formats

CM:spread-from-processor-1L *dest*, *send-address-value*, *source*, *len*
CM:spread-from-processor-a-1L *dest*, *send-address-value*, *source*, *len*

Operands

dest The field ID of the destination field.

send-address-value An unsigned integer immediate operand to be used as the the send address of the processor whose *source* value is to be spread.

source The field ID of the source field.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context

The non-always operations are conditional.

The always operations are unconditional.
For this instruction, -a is used instead of the standard -always suffix to indicate unconditional operation.

Definition

For every virtual processor k in *vp-set*(*dest*) do
  if (always or *context-flag*[k] = 1) then
    dest[k] ← *source*[*send-address-value*]

The value of the source field in the processor specified by *send-address-value* is spread to all (selected) processors in the destination field. The source and destination fields may reside in different VP sets.

SPREAD-WITH-C-ADD

The destination field in every selected processor receives the sum of the complex source fields from processors below or above it in some ordering of the processors.

Formats CM:spread-with-c-add-1L *dest*, *source*, *axis*, *s*, *e*

Operands

dest	The field ID of the complex destination field.
source	The field ID of the complex source field.
axis	An unsigned integer immediate operand to be used as the the number of a NEWS axis.
s, *e*	The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s+e+1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\mathit{context\text{-}flag}[k] = 1$ then
  let $C_k = \mathit{scan\text{-}subclass}(k, \{\, \mathit{axis} \,\})$

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \mathit{source}[m] \right)$$

where *scan-subclass* is as defined on page 36 of the *Paris Reference Manual.*

See the section beginning on page 36 for a general description of spread operations. The CM:spread-with-c-add operation combines *source* fields by performing complex addition.

A call to CM:spread-with-c-add-1L is equivalent to the sequence

CM:scan-with-c-add-1L *dest*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *source*, *axis*, $2 \times (s+e+1)$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-F-ADD

The destination field in every selected processor receives the sum of the floating-point source fields from all processors in its scan subclass.

Formats CM:spread-with-f-add-1L *dest*, *source*, *axis*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    let $g = \mathit{geometry}(\mathit{current\text{-}vp\text{-}set})$
    let $C_k = \mathit{scan\text{-}subclass}(g, k, \mathit{axis})$

$$\mathit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \mathit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-f-add operation combines *source* fields by performing floating-point addition.

A call to CM:spread-with-f-add-1L is equivalent to the sequence

CM:scan-with-f-add-1L *temp*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, $s + e + 1$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-S-ADD

The destination field in every selected processor receives the sum of the signed integer source fields from all processors in its scan subclass.

Formats CM:spread-with-s-add-1L *dest*, *source*, *axis*, *len*

Operands	*dest*	The field ID of the signed integer destination field.
	source	The field ID of the signed integer source field.
	axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
	len	The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap: The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context: This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *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 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-s-add operation combines *source* fields by performing signed integer addition.

A call to CM:spread-with-s-add-1L is equivalent to the sequence

CM:scan-with-s-add-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-U-ADD

The destination field in every selected processor receives the sum of the unsigned integer source fields from all processors in its scan subclass.

Formats CM:spread-with-u-add-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\sum_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-u-add operation combines *source* fields by performing unsigned integer addition.

A call to CM:spread-with-u-add-1L is equivalent to the sequence

CM:scan-with-u-add-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-COPY

The destination field in every selected processor receives a copy of the source value from a particular value within its scan subclass.

Formats CM:spread-with-copy-1L *dest*, *source*, *axis*, *len*, *coordinate*

Operands

dest The field ID of the unsigned integer destination field.

source The field ID of the unsigned integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

coordinate An unsigned integer immediate operand to be used as the NEWS coordinate along *axis* indicating which element of the scan class is to be replicated.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    let g = *geometry*(*current-vp-set*)
    let c = *deposit-news-constant*(g, k, *axis*, *coordinate-value*)
    dest$[k] \leftarrow$ *source*$[c]$

where *deposit-news-constant* is defined in the dictionary entry for CM:deposit-news-coordinate.

See section 5.20 on page 42 for a general description of spread operations.

SPREAD-WITH-LOGAND

The destination field in every selected processor receives the bitwise logical AND of the source fields from all processors in its scan subclass.

Formats CM:spread-with-logand-1L *dest*, *source*, *axis*, *len*

Operands

dest The field ID of the destination field.

source The field ID of the source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
 if $\textit{context-flag}[k] = 1$ then
  let $g = \textit{geometry}(\textit{current-vp-set})$
  let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\bigwedge_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-logand operation combines *source* fields by performing bitwise logical AND operations.

A call to CM:spread-with-logand-1L is equivalent to the sequence

CM:scan-with-logand-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-LOGIOR

The destination field in every selected processor receives the bitwise logical inclusive OR of the source fields from all processors in its scan subclass.

Formats CM:spread-with-logior-1L *dest*, *source*, *axis*, *len*

Operands

- *dest* — The field ID of the destination field.
- *source* — The field ID of the source field.
- *axis* — An unsigned integer immediate operand to be used as the number of a NEWS axis.
- *len* — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ 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 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-logior operation combines *source* fields by performing bitwise logical inclusive OR operations.

A call to CM:spread-with-logior-1L is equivalent to the sequence

CM:scan-with-logior-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-LOGXOR

The destination field in every selected processor receives the bitwise logical exclusive OR of the source fields from all processors in its scan subclass.

Formats CM:spread-with-logxor-1L *dest*, *source*, *axis*, *len*

Operands

dest — The field ID of the destination field.

source — The field ID of the source field.

axis — An unsigned integer immediate operand to be used as the number of a NEWS axis.

len — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\bigoplus_{m \in C_k} \textit{source}[m] \right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-logxor operation combines *source* fields by performing bitwise logical exclusive OR operations.

A call to CM:spread-with-logxor-1L is equivalent to the sequence

CM:scan-with-logxor-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-F-MAX

The destination field in every selected processor receives the largest of the floating-point source fields from all processors in its scan subclass.

Formats CM:spread-with-f-max-1L *dest*, *source*, *axis*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$
    $$dest[k] \leftarrow \left(\max_{m \in C_k} source[m]\right)$$
where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-f-max operation combines *source* fields by performing an floating-point maximum operation.

A call to CM:spread-with-f-max-1L is equivalent to the sequence

CM:scan-with-f-max-1L *temp*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, $s + e + 1$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-S-MAX

The destination field in every selected processor receives the largest of the signed integer source fields from all processors in its scan subclass.

Formats CM:spread-with-s-max-1L *dest*, *source*, *axis*, *len*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    let $g = geometry(current\text{-}vp\text{-}set)$
    let $C_k = scan\text{-}subclass(g, k, axis)$
    $$dest[k] \leftarrow \left(\max_{m \in C_k} source[m]\right)$$
where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-s-max operation combines *source* fields by performing a signed integer maximum operation.

A call to CM:spread-with-s-max-1L is equivalent to the sequence

CM:scan-with-s-max-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-U-MAX

The destination field in every selected processor receives the largest of the unsigned integer source fields from all processors in its scan subclass.

Formats CM:spread-with-u-max-1L *dest*, *source*, *axis*, *len*

Operands

- *dest* — The field ID of the unsigned integer destination field.
- *source* — The field ID of the unsigned integer source field.
- *axis* — An unsigned integer immediate operand to be used as the number of a NEWS axis.
- *len* — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap — The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context — This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$

$$\textit{dest}[k] \leftarrow \left(\max_{m \in C_k} \textit{source}[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-u-max operation combines *source* fields by performing an unsigned integer maximum operation.

A call to CM:spread-with-u-max-1L is equivalent to the sequence

CM:scan-with-u-max-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-F-MIN

The destination field in every selected processor receives the smallest of the floating-point source fields from all processors in its scan subclass.

Formats CM:spread-with-f-min-1L *dest*, *source*, *axis*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source — The field ID of the floating-point source field.

axis — An unsigned integer immediate operand to be used as the number of a NEWS axis.

s, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    let $g = \textit{geometry}(\textit{current-vp-set})$
    let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
    $\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m]\right)$
where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-f-min operation combines *source* fields by performing an floating-point minimum operation.

A call to CM:spread-with-f-min-1L is equivalent to the sequence

CM:scan-with-f-min-1L *temp*, *source*, *axis*, *s*, *e*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, $s + e + 1$, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-S-MIN

The destination field in every selected processor receives the smallest of the signed integer source fields from all processors in its scan subclass.

Formats CM:spread-with-s-min-1L *dest*, *source*, *axis*, *len*

Operands

dest The field ID of the signed integer destination field.

source The field ID of the signed integer source field.

axis An unsigned integer immediate operand to be used as the number of a NEWS axis.

len The length of the *dest* and *source* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
let $g = \textit{geometry}(\textit{current-vp-set})$
let $C_k = \textit{scan-subclass}(g, k, \textit{axis})$
$$\textit{dest}[k] \leftarrow \left(\min_{m \in C_k} \textit{source}[m]\right)$$
where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-s-min operation combines *source* fields by performing a signed integer minimum operation.

A call to CM:spread-with-s-min-1L is equivalent to the sequence

CM:scan-with-s-min-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

SPREAD-WITH-U-MIN

The destination field in every selected processor receives the smallest of the unsigned integer source fields from all processors in its scan subclass.

Formats CM:spread-with-u-min-1L *dest*, *source*, *axis*, *len*

Operands

dest	The field ID of the unsigned integer destination field.
source	The field ID of the unsigned integer source field.
axis	An unsigned integer immediate operand to be used as the number of a NEWS axis.
len	The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
let $g = geometry(current\text{-}vp\text{-}set)$
let $C_k = scan\text{-}subclass(g, k, axis)$

$$dest[k] \leftarrow \left(\min_{m \in C_k} source[m]\right)$$

where *scan-subclass* is as defined on page 44.

See section 5.20 on page 42 for a general description of spread operations. The CM:spread-with-u-min operation combines *source* fields by performing an unsigned integer minimum operation.

A call to CM:spread-with-u-min-1L is equivalent to the sequence

CM:scan-with-u-min-1L *temp*, *source*, *axis*, *len*, :upward, :inclusive, :none, *dont-care*
CM:scan-with-copy-1L *dest*, *temp*, *axis*, *len*, :downward, :inclusive, :none, *dont-care*

but may be faster.

C-SQRT

Calculates the square root of the complex source field and places it in the complex destination field.

Formats CM:c-sqrt-1-1L *dest/source*, *s*, *e*
CM:c-sqrt-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then
$dest[k] \leftarrow \sqrt{source}$

In each selected processor, the square root of the *source* field value is placed in the *dest* field.

F-SQRT

Calculates the floating-point square root of the source field values and stores the result in the floating-point destination field.

Formats CM:f-sqrt-1-1L *dest*/*source*, *s*, *e*
CM:f-sqrt-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *test-flag* is set if the *source* is negative and non-zero; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\mathit{context\text{-}flag}[k] = 1$ then
    if $\mathit{source}[k] > 0$ then
      $\mathit{dest}[k] \leftarrow \sqrt{\mathit{source}[k]}$
    else if $\mathit{source}[k] = \pm 0$ then
      $\mathit{dest}[k] \leftarrow \mathit{source}[k]$
    else if : $\mathit{source} : [k] < 0$ then
      $\mathit{dest}[k] \leftarrow \langle\text{unpredictable}\rangle$
      $\mathit{test}[k] \leftarrow 1$

If the *source* value is non-negative, then the square root of that value is placed in the destination. The square root of -0 is defined to be -0.

If the *source* operand is a NaN, then it is copied to the *dest* field unchanged.

STORE-CONTEXT

Unconditionally stores the context bit into memory.

Formats CM:store-context *dest*

Operands *dest* The field ID of the destination bit (a one-bit field).

Context This operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
$dest[k] \leftarrow context\text{-}flag[k]$

Within each processor, the context bit for that processor is unconditionally stored into memory.

STORE-flag

Conditionally stores a flag bit into memory.

Formats

CM:store-test	*dest*
CM:store-test-always	*dest*
CM:store-overflow	*dest*
CM:store-overflow-always	*dest*

Operands *dest* The field ID of the destination bit (a one-bit field).

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] $\leftarrow$ *flag*[k]

where *flag* is *test-flag* or *overflow-flag*, as appropriate.

Within each processor, the indicated flag for that processor is stored into memory.

FE-STRUCTURE-ARRAY-FORMAT

This instruction returns an array format descriptor for a particular slot in an array of structures. A format descriptor may be passed to any array transfer instruction to specify a front-end array format, although this is not required. See also CM:fe-array-format and CM:fe-packed-array-format.

This instruction is not provided for the Lisp interface to Paris.

Formats result ← CM:fe-structure-array-format *cm-element-byte-size*, *structure-byte-size*

Operands *cm-element-byte-size* A signed integer immediate operand to be used as the number of bytes each Connection Machine element occupies in the front-end array. This must be a power of two between 1 and 16.

structure-byte-size A signed integer immediate operand to be used as the length of the front-end structure in bytes. This may be any positive integer.

Result The array format descriptor specified.

Context This is a front-end operation. It does not depend on the value of the *context-flag*.

The return value is a format descriptor for a front-end array of structures. Such a format descriptor can be passed to any of the CM array transfer instructions in order to allow transfers in either direction between CM fields and a front-end array of structures. If this is done, one CM element per selected processor is copied into, or receives data from, the specified slot across an array of structures on the front end.

Values for both *cm-element-byte-size* and *cm-structure-byte-size* may be obtained by calls to sizeof(...).

The value of *cm-element-byte-size* specifies the length of the structure slot in bytes. It also defines the unit of measure for the *fe-offset-vector* argument to the CM:read-from-news-array and CM:write-to-news-array instructions.

The value of *structure-byte-size* specifies the length of the entire stucture in bytes. It also defines the unit of measure for the argument *fe-dimension-vector* to the CM:read-from-news-array and CM:write-to-news-array instructions.

If a slot other than the first slot in the front-end structure is the destination of a CM:read-from-news-array or the source for a CM:write-to-news-array transfer instruction, then a pointer to that slot must be provided as the value of *front-end-array*. This is a bit tricky. The pointer must identify the location of the chosen slot in the structure that is the first element of the array of structures.

Here is an example in C.

```
#define n_foos 256

/* declare array of structure foo */
struct foo { int a; double b; char c; } fooarray[n_foos];

/* declare the format */
CM_array_format_t foo_format;

/* declare an offset for the 'b' slot of struct foo */
/* this is a pointer to a double - b is a double */
double *bslot_pointer;

/* lots of other declarations etc. in here */
...

/* create format descriptor for foo.b */
foo_format = CM_structure_array_format(sizeof(double), sizeof(struct foo));

/* create pointer offset to slot b of struct foo */
bslot_pointer = &fooarray[0].b;

/* store src-field values in slot b of each foo struct in foo_array */
/* all variables xxxx_vector should be self explanatory */

CM_f_read_from_news_array_1L(bslot_pointer, offset_vector,
                             start_vector, end_vector, axis_vector,
                             src_field, 23, 8, rank,
                             dimension_vector, foo_format);
```

Slot b of each foo structure in the array foo_array receives a copy of the value stored in the corresponding CM *src-field* processor.

The value of bslot_pointer is a pointer to the b slot of the first foo structure in foo_array. Given this starting place, foo_format indicates how many bytes must be skipped between b slots.

For further examples, refer to the manual entitled *Introduction to Programming in C/Paris.*

F-SUBF-CONST-MULT

Calculates a value $(b - a)x$ and places it in the destination.

Formats

CM:f-subf-const-mult-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-subf-const-mult-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-subf-const-mult-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-subf-const-mult-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source (subtrahend) field.

source2-value A floating-point immediate operand to be used as the second source (minuend).

source3 The field ID of the floating-point third source (multiplier) field.

source3-value A floating-point immediate operand to be used as the third source (multiplier).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    $dest[k] \leftarrow (source2\text{-}value[k] - source1[k]) \times source3[k]$
  if ⟨overflow occurred in processor k⟩ then $overflow\text{-}flag[k] \leftarrow 1$

The operand *source1* is subtracted from *source2-value*, treating them as floating-point numbers, and then the difference is multiplied by a third operand *source3*. The result is stored in the destination field. The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operands *source2-value* and *source3-value* should be double-precision front-end values (in Lisp, automatic coercion is performed if necessary). The constants are then converted, in effect, to the format specified by *s* and *e* before the operation is performed.

A call to CM:f-subf-const-mult-1L is equivalent to the sequence

CM:f-subfrom-constant-3-1L *dest*, *source1*, *source2-value*, *s*, *e*
CM:f-multiply-3-1L *dest*, *dest*, *source3*, *s*, *e*

but may be faster.

F-SUB-MULT

Calculates a value $(x - a)b$ and places it in the destination.

Formats

CM:f-sub-mult-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-sub-mult-always-1L	*dest*, *source1*, *source2*, *source3*, *s*, *e*
CM:f-sub-const-mult-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-sub-const-mult-always-1L	*dest*, *source1*, *source2-value*, *source3*, *s*, *e*
CM:f-sub-mult-const-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-sub-mult-const-always-1L	*dest*, *source1*, *source2*, *source3-value*, *s*, *e*
CM:f-sub-const-mult-const-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*
CM:f-sub-const-mult-const-a-1L	*dest*, *source1*, *source2-value*, *source3-value*, *s*, *e*

Operands

dest The field ID of the floating-point destination field.

source1 The field ID of the floating-point first source (minuend) field.

source2 The field ID of the floating-point second source (subtrahend) field.

source2-value A floating-point immediate operand to be used as the second source (subtrahend).

source3 The field ID of the floating-point third source (multiplier) field.

source3-value A floating-point immediate operand to be used as the third source (multiplier).

s, *e* The significand and exponent lengths for the *dest*, *source1*, *source2*, and *source3* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1*, *source2*, and *source3* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or *context-flag*$[k] = 1$) then
    dest$[k] \leftarrow$ (*source1*$[k]$ − *source2*$[k]$) $\times$ *source3*$[k]$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$

The operand *source2* is subtracted from *source1*, treating them as floating-point numbers, and then the difference is multiplied by a third operand *source3*. The result is stored in the destination field.

The various operand formats allow the second and third source operands to be either memory fields or constants.

The constant operand *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by *s* and *e*.

A call to CM:f-sub-mult-1L is equivalent to the sequence

CM:f-subtract-3-1L *temp*, *source1*, *source2*, *s*, *e*
CM:f-multiply-3-1L *dest*, *temp*, *source3*, *s*, *e*

but may be faster.

C-SUBTRACT

The difference of two complex source values is placed in the destination field.

Formats

CM:c-subtract-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-subtract-always-2-1L	*dest/source1*, *source2*, *s*, *e*
CM:c-subtract-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-subtract-always-3-1L	*dest*, *source1*, *source2*, *s*, *e*
CM:c-subtract-constant-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-subtract-const-always-2-1L	*dest/source1*, *source2-value*, *s*, *e*
CM:c-subtract-constant-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:c-subtract-const-always-3-1L	*dest*, *source1*, *source2-value*, *s*, *e*
CM:c-subfrom-2-1L	*dest/source2*, *source1*, *s*, *e*
CM:c-subfrom-always-2-1L	*dest/source2*, *source1*, *s*, *e*
CM:c-subfrom-constant-2-1L	*dest/source2*, *source1-value*, *s*, *e*
CM:c-subfrom-const-always-2-1L	*dest/source2*, *source1-value*, *s*, *e*
CM:c-subfrom-constant-3-1L	*dest*, *source2*, *source1-value*, *s*, *e*
CM:c-subfrom-const-always-3-1L	*dest*, *source2*, *source1-value*, *s*, *e*

Operands

dest The field ID of the complex destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the complex first source field. This is the minuend.

source2 The field ID of the complex second source field. This is the subtrahend.

source1-value A complex immediate operand to be used as the first source.

source2-value A complex immediate operand to be used as the second source.

s, *e* The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \textit{source1}[k] - \textit{source2}[k]$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$

The operand *source2* is subtracted from *source1*, treated as as complex numbers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The constant operand *source1-value* or *source2-value* should be a double-precision complex front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

F-SUBTRACT

The difference of two floating-point source values is placed in the destination field.

Formats

CM:f-subtract-2-1L	*dest/source1, source2, s, e*
CM:f-subtract-always-2-1L	*dest/source1, source2, s, e*
CM:f-subtract-3-1L	*dest, source1, source2, s, e*
CM:f-subtract-always-3-1L	*dest, source1, source2, s, e*
CM:f-subtract-constant-2-1L	*dest/source1, source2-value, s, e*
CM:f-subtract-const-always-2-1L	*dest/source1, source2-value, s, e*
CM:f-subtract-constant-3-1L	*dest, source1, source2-value, s, e*
CM:f-subtract-const-always-3-1L	*dest, source1, source2-value, s, e*
CM:f-subfrom-2-1L	*dest/source2, source1, s, e*
CM:f-subfrom-always-2-1L	*dest/source2, source1, s, e*
CM:f-subfrom-constant-2-1L	*dest/source2, source1-value, s, e*
CM:f-subfrom-const-always-2-1L	*dest/source2, source1-value, s, e*
CM:f-subfrom-constant-3-1L	*dest, source2, source1-value, s, e*
CM:f-subfrom-const-always-3-1L	*dest, source2, source1-value, s, e*

Operands

dest The field ID of the floating-point destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the floating-point first source field. This is the minuend.

source2 The field ID of the floating-point second source field. This is the subtrahend.

source1-value A floating-point immediate operand to be used as the first source.

source2-value A floating-point immediate operand to be used as the second source.

s, e The significand and exponent lengths for the *dest*, *source1*, and *source2* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format. It is permissible for all the fields to be identical.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context The non-always operations are conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination and flag may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
 if (always or *context-flag*[k] = 1) then
  dest[k] ← *source1*[k] − *source2*[k]
 if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1

The operand *source2* is subtracted from *source1*, treated as as floating-point numbers. The result is stored into the memory field *dest*. The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The constant operand *source1-value* or *source2-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary). Before the operation is performed, the constant is converted, in effect, to the format specified by s and e.

S-SUBTRACT

The difference of two signed integer source values is placed in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats

CM:s-subtract-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-subtract-2-1L	*dest*/*source1*, *source2*, *len*
CM:s-subtract-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-subtract-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:s-subtract-constant-3-1L	*dest*, *source1*, *source2-value*, *len*
CM:s-subfrom-2-1L	*dest*/*source2*, *source1*, *len*
CM:s-subfrom-constant-2-1L	*dest*/*source2*, *source1-value*, *len*
CM:s-subfrom-constant-3-1L	*dest*, *source2*, *source1-value*, *len*

Operands

dest The field ID of the signed integer destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the signed integer first source field. This is the minuend.

source2 The field ID of the signed integer second source field. This is the subtrahend.

source1-value A signed integer immediate operand to be used as the first source.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-subtract-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-subtract-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-subtract-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is equivalent to "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \textit{source1}[k] - \textit{source2}[k]$
    if ⟨no borrow needed in processor k⟩ then $\textit{carry-flag}[k] \leftarrow 1$
    else $\textit{carry-flag}[k] \leftarrow 0$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$
    else $\textit{overflow-flag}[k] \leftarrow 0$

The operand *source2* is subtracted from *source1*, treated as as signed integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The various operand formats allow the first and second source operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source1-value* or *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-SUBTRACT

The difference of two unsigned integer source values is placed in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats

CM:u-subtract-3-3L	*dest, source1, source2, dlen, slen1, slen2*
CM:u-subtract-2-1L	*dest/source1, source2, len*
CM:u-subfrom-2-1L	*dest/source2, source1, len*
CM:u-subtract-3-1L	*dest, source1, source2, len*
CM:u-subtract-constant-2-1L	*dest/source1, source2-value, len*
CM:u-subfrom-constant-2-1L	*dest/source2, source1-value, len*
CM:u-subtract-constant-3-1L	*dest, source1, source2-value, len*
CM:u-subfrom-constant-3-1L	*dest, source2, source1-value, len*

Operands

dest The field ID of the unsigned integer destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the unsigned integer first source field. This is the minuend.

source2 The field ID of the unsigned integer second source field. This is the subtrahend.

source1-value An unsigned integer immediate operand to be used as the first source.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-subtract-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-subtract-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-subtract-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is equivalent to "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*[k] = 1 then
    dest[k] ← *source1*[k] – *source2*[k]
    if ⟨no borrow needed in processor k⟩ then *carry-flag*[k] ← 1
    else *carry-flag*[k] ← 0
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*[k] ← 1
    else *overflow-flag*[k] ← 0

The operand *source2* is subtracted from *source1*, treated as as unsigned integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The various operand formats allow operands to be either memory fields or constants; in some cases the destination field initially contains one source operand. The "subfrom" operations allow for the destination to be subtracted from the other operand, or for a memory field to be subtracted from an immediate value.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source1-value* or *source2-value* should be an unsigned integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

S-SUBTRACT-BORROW

In each selected processor, computes the difference of two signed integer source values and places it in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats CM:s-subtract-borrow-3-1L *dest*, *source1*, *source2*, *len*

Operands

dest — The field ID of the signed integer destination field. This is the difference, the result of the subtraction operation.

source1 — The field ID of the signed integer first source field. This is the minuend.

source2 — The field ID of the signed integer second source field. This is the subtrahend.

len — The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there is no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is interpreted as "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if *context-flag*$[k] = 1$ then
    dest$[k] \leftarrow$ *source1*$[k]$ − *source2*$[k]$ + (*carry-flag*$[k] - 1$)
    if ⟨no borrow needed in processor k⟩ then *carry-flag*$[k] \leftarrow 1$
    else *carry-flag*$[k] \leftarrow 0$
    if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
    else *overflow-flag*$[k] \leftarrow 0$

The operand *source2* is subtracted from *source1*, treated as signed integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

U-SUBTRACT-BORROW

In each selected processor, computes the difference of two unsigned integer source values and places it in the destination field. "Borrow-in" and "borrow-out" are simulated by the *carry-flag*, and overflow is also computed.

Formats CM:u-subtract-borrow-3-1L *dest*, *source1*, *source2*, *len*

Operands

dest The field ID of the unsigned integer destination field. This is the difference, the result of the subtraction operation.

source1 The field ID of the unsigned integer first source field. This is the minuend.

source2 The field ID of the unsigned integer second source field. This is the subtrahend.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags *carry-flag* is set if there no borrow-in to the high-order bit position; otherwise it is cleared.

For subtraction, "carry" is equivalent to "not borrow." Thus, if *source1* is greater than or equal to *source2*, then the *carry-flag* is set – meaning there is no borrow. Conversely, if *source1* is less than *source2*, a borrow *is* required so the *carry-flag* is cleared.

overflow-flag is set if the difference cannot be represented in the destination field; otherwise it is cleared.

Context This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
  if $\textit{context-flag}[k] = 1$ then
    $\textit{dest}[k] \leftarrow \textit{source1}[k] - \textit{source2}[k] + (\textit{carry-flag}[k] - 1)$
    if ⟨no borrow needed in processor k⟩ then $\textit{carry-flag}[k] \leftarrow 1$
    else $\textit{carry-flag}[k] \leftarrow 0$
    if ⟨overflow occurred in processor k⟩ then $\textit{overflow-flag}[k] \leftarrow 1$
    else $\textit{overflow-flag}[k] \leftarrow 0$

The operand *source2* is subtracted from *source1*, treated as as unsigned integers. A borrow bit is simulated by inverting the *carry-flag*. The result is stored into the memory field *dest*.

The *carry-flag* and *overflow-flag* may be altered by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

SWAP

Swaps the contents of two bit fields.

Formats
CM:swap-2-1L *dest1/source1, dest2/source2, len*
CM:swap--always-2-1L *dest1/source1, dest2/source2, len*

Operands

dest1 The field ID of the first destination field.

source1 The field ID of the first source (same as first destination) field.

dest2 The field ID of the second destination field.

source2 The field ID of the second source (same as second destination) field.

len The length of the *dest1*, *source1*, *dest2*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap The fields *dest1* and *dest2* must not overlap in any manner.

Context The non-always operations are conditional. The destination may be altered only in processors whose *context-flag* is 1.

The always operations are unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
  if (always or $context\text{-}flag[k] = 1$) then
    let $temp1_k = source1[k]$
    let $temp2_k = source2[k]$
    let $dest1[k] \leftarrow temp2_k$
    let $dest2[k] \leftarrow temp1_k$

Each of the two provided fields is copied into the other so as to exchange their contents.

C-TAN

Calculates the complex tangent of the source field values and stores the result in the complex destination field.

Formats CM:c-tan-1-1L *dest/source*, *s*, *e*
CM:c-tan-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    dest[*k*] ← 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-TAN

Calculates the floating-point tangent of the source field values and stores the result in the floating-point destination field.

Formats CM:f-tan-1-1L *dest/source*, *s*, *e*
CM:f-tan-2-1L *dest*, *source*, *s*, *e*

Operands

dest — The field ID of the floating-point destination field.

source — The field ID of the floating-point source field.

s, *e* — The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then
dest[*k*] ← 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.

C-TANH

Calculates the complex hyperbolic tangent of the source field values and stores the result in the complex destination field.

Formats CM:c-tanh-1-1L *dest/source*, *s*, *e*
CM:c-tanh-2-1L *dest*, *source*, *s*, *e*

Operands

dest The field ID of the complex destination field.

source The field ID of the complex source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $2(s + e + 1)$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two complex fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
if *context-flag*[*k*] = 1 then
dest[*k*] ← tanh *source*

The hyperbolic tangent of the value of the *source* field is stored into the *dest* field.

F-TANH

Calculates the floating-point hyperbolic tangent of the source field values and stores the result in the floating-point destination field.

Formats CM:f-tanh-1-1L *dest*/*source*, *s*, *e*
CM:f-tanh-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Flags *overflow-flag* is set if floating-point overflow occurs; otherwise it is unaffected.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    dest[*k*] ← tanh *source*
if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

The hyperbolic tangent of the value of the *source* field is stored into the *dest* field.

TIME

Times other operations and reports both the total amount of time elapsed and the amount of time spent executing on the Connection Machine system.

This instruction is available only from the Lisp/Paris interface. For Fortran/Paris and C/Paris users, the equivalent functionality is provided by the CM:timer- series of functions – which may also be used from Lisp. The CM:timer- functions are documented in this dictionary and also in the *CM System User's Guide*.

Formats CM:time *form, [return-statistics-p]*

Operands

form — The a Lisp, Lisp/Paris, or *Lisp form to be timed. This must be a single Lisp expression. To time more than one expression, enclose them in a progn form.

return-statistics-p — The answer to the question, "Do you want timing statistics returned as the value of the macro?". This is an optional keyword argument and defaults to NIL. When specified, the invocation must include the keyword :return-statistics-p followed by T or NIL.

Context — This operation is unconditional. It does not depend on the *context-flag*.

The CM:time facility is a Lisp macro, not a function. It is used in the Lisp/Paris interface to time the execution of other operations on the Connection Machine system.

A call to the CM:time macro may contain a single Lisp expression; this is executed in the normal manner, but before the value is returned, timing information is printed out as for the Common Lisp **time** macro.

Specifying a NIL value to the :return-statistics-p (the default) causes the statistics to be displayed on standard output.

Specifying a T value to the :return-statistics-p causes the statistics to be returned as two floating-point values in a list that is the return value of the macro call.

The first number reported is elapsed time during execution on both the front-end computer and the Connection Machine system. In addition, timing information related to Connection Machine system performance is printed. The second number reported is the amount of that time that the Connection Machine system was actually executing instructions (not waiting for the front end). For optimal performance, the programmer strives to obtain the maximum percentage of Connection Machine utilization possible.

For further information about timing code from the Lisp/Paris interface, see the *CM System User's Guide* chaper entitled "In The Lisp Environment."

TIME

The timing facility is provided in the C/Paris and Fortran/Paris interfaces through a set of functions whose names all begin with CM:timer-.

TIMER

The timing facility. A set of instructions that together determine how much time any part of a program takes to execute on the Connection Machine.

Formats	CM:timer-clear	*timer*
	CM:timer-start	*timer*
	CM:timer-stop	*timer*
	CM:timer-print	*timer*
	CM:timer-read-starts	*timer*
	CM:timer-read-elapsed	*timer*
	CM:timer-read-cm-busy	*timer*
	CM:timer-read-cm-idle	*timer*
	CM:timer-read-run-state	*timer*
	CM:timer-set-starts	*timer*, *int*

Operands	*timer*	The integer used to identify the timer being used.. This must be an unsigned integer immediate operand between 0 (inclusive) and CM*max-number-of-timers* (exclusive).
	int	For CM:timer-set-starts, the start number to which the specified timer is to be reset.

Context This operation is unconditional. It does not depend on the *context-flag.*

To activate multiple timers, assign each an integer identifier. Nested calls to different timers is permitted. Each timer can record timings of up to 43 hours, with microsecond precision.

Four basic operations are required in order to use this timing facility. Use them in the following order:

CM:timer-clear

Sets the total elapsed time, total CM busy time, and number of starts for *timer* to zero.

CM:timer-start

Starts the clock running for *timer*. Elapsed time (also known as wall time) and CM busy time are accumulated. Number of starts is incremented.

CM:timer-stop

Stops the clock running for timer. The specified timer's state variables for CM elapsed time and CM busy time are updated. A subsequent call to CM:timer-start – without an intervening call to CM:timer-clear – restarts the timer and *adds* to the accumulated elapsed and busy values for this timer.

CM:timer-print

Prints information about *timer*, including, but not limited to: the number of starts, the total elapsed time, and the total time that the Connection Machine was busy while this timer was active.

To use a timer, first invoke CM:timer-clear to zero the timer values. Then, call CM:timer-start and CM:timer-stop any number of times. Finally call CM:timer-print.

For each timer, state variables for CM elapsed time and CM busy time are maintained. Elapsed time records how much time has elapsed between each pair of CM:timer-start and CM:timer-stop calls that have been made since CM:timer-clear was last called for *timer*. CM busy time records the total time the CM has spent being active between each pair of CM:timer-start and CM:timer-stop calls that have been made since CM:timer-clear was last called for *timer*.

The following five functions return state values for a specified timer:

CM:timer-read-starts

Returns an unsigned integer, the number of times CM:timer-start has been called for this timer.

CM:timer-read-elapsed

Returns the total elapsed time, in seconds, accumulated while *timer* was running.

CM:timer-read-cm-busy

Returns the total CM busy time, in seconds, accumulated while *timer* was running.

CM:timer-read-cm-idle

Returns the total CM idle time, in seconds, accumulated while *timer* was running. CM idle time is equal to total elapsed time minus the CM busy time.

CM:timer-read-run-state

Returns TRUE (or t or 1) if and only if *timer* is running. Otherwise, returns FALSE (or nil or 0).

One further operation is provided to reset the number of starts for the specified timer:

CM:timer-set-starts

Sets the number of starts for *timer* to the specified integer value. This is useful in code that stops a timer to query it and then restarts the same timer. CM:timer-set-starts can be used to set the number of starts to 1 less than the actual number of starts before restarting the timer. In this way, querying a timer does not change the number of starts ultimately recorded.

For a detailed guide to using the new timing facility, including information about conditions that affect timing accuracy, see the *CM System User's Guide*.

FE-TO-GRAY-CODE

Converts, on the front end, a nonnegative integer into a bit string representing a Gray-coded integer value.

Formats result ← CM:fe-to-gray-code *integer*

Operands *integer* An unsigned integer immediate operand to be used as the nonnegative integer.

Result An unsigned integer, the Gray code equivalent of *integer*.

Context This operation is performed on the front end. It does not depend on the CM *context-flag*.

Definition Return $integer \oplus \left\lfloor \frac{integer}{2} \right\rfloor$

This function calculates, entirely on the front end, a bit-string encoding in a particular reflected binary Gray code. The position of that value in the standard Gray code sequence is equal to the specified *integer*.

Note that the binary value 0 is always equivalent to a Gray code string that is all 0-bits.

U-TO-GRAY-CODE

Converts an unsigned binary integer to a bit string representing a Gray-coded integer value.

Formats

CM:u-to-gray-code-1-1L *dest/source*, *len*
CM:u-to-gray-code-2-1L *dest*, *source*, *len*

Operands

dest — The field ID of the destination field.

source — The field ID of the unsigned integer source field.

len — The length of the *dest* and *source* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The *source* field must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length.

Context

This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition

For every virtual processor k in the *current-vp-set* do
  if $context\text{-}flag[k] = 1$ then
    $dest[k]\langle len - 1\rangle \leftarrow source[k]\langle len - 1\rangle$
    for j from $len - 2$ to 0 do
      $dest[k]\langle j\rangle \leftarrow source[k]\langle j\rangle \oplus source[k]\langle j + 1\rangle$

The *source* operand is an unsigned binary integer, and is converted to a bit-string value in a particular reflected binary Gray code. The position of that value in the standard Gray code sequence is the *source*.

Note that the binary value 0 is always equivalent to a Gray code string that is all 0-bits.

TRANSPOSE32

Within each cluster of 32 physical processors, for every group of 32 virtual processors in such a cluster, copies one 32-bit field to another. During this copying operation, transposes the data as a 32-by-32 bit matrix. Thus, each virtual processor receives one bit from the source value of each virtual processor in its group of 32.

Formats CM:transpose32-1-1L *dest/source, len*
CM:transpose32-2-1L *dest, source, len*

Operands

source The field ID of the source field.

dest The field ID of the destination field.

len The length of the *source* and *dest* fields. This must be non-negative and no greater than CM:*maximum-integer-length*. This must be a multiple of 32.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length. The fields *dest* and *source* may overlap in any manner.

Context This operation is unconditional. The destination may be altered regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
for all j such that $0 \le j < dlen$ do
$\textit{dest}[k]\langle j\rangle \leftarrow$

$$\textit{source}\left[32r\left\lfloor\frac{k}{32r}\right\rfloor + (k \bmod r) + r(j \bmod 32)\right]\left\langle 32\left\lfloor\frac{j}{32}\right\rfloor + \frac{k \bmod 32}{r}\right\rangle$$

where r is the value of CM:*virtual-to-physical-processor-ratio* and j is the bit position in each field.

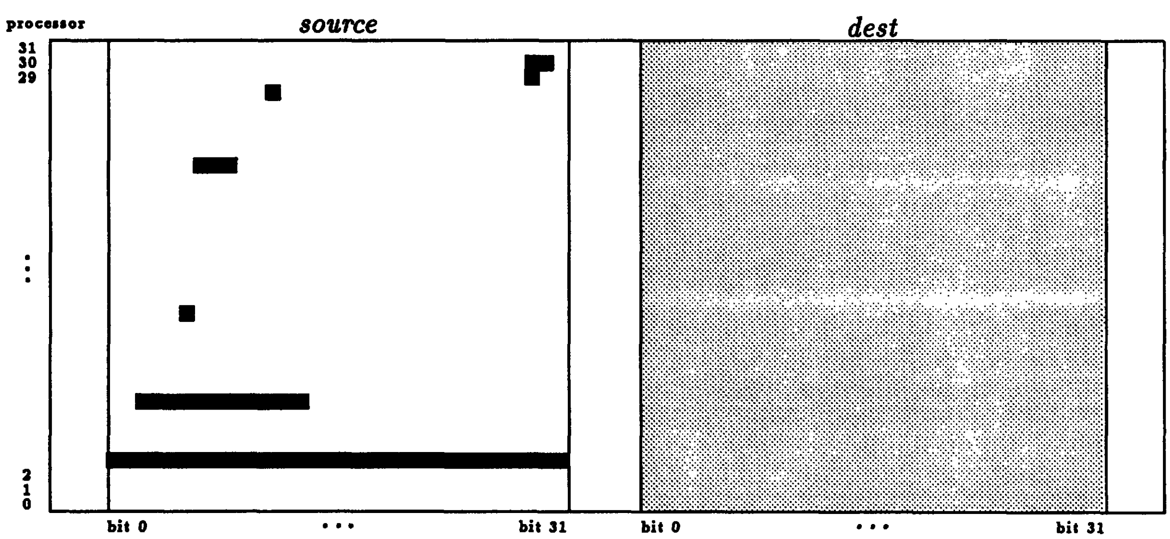
This instruction copies each 32-bit field to the corresponding 32-bit field within each virtual processor. In the course of copying the bits, it "transposes" them so that a 32-bit value lying entirely within the *source* field of one virtual processor is made to occupy a memory slice, that is, one bit in each of 32 virtual processors. The opposite is also true: the 32-bit value that ends up in the *dest* field of a virtual processor is made up of one bit from each of 32 virtual processors. Transposed data is said to be stored in a *slicewise* format.

For the purposes of this instruction, the physical processors are divided into clusters of 32. Two processors are in the same cluster if their physical processor numbers agree in all but the five least significant bits.

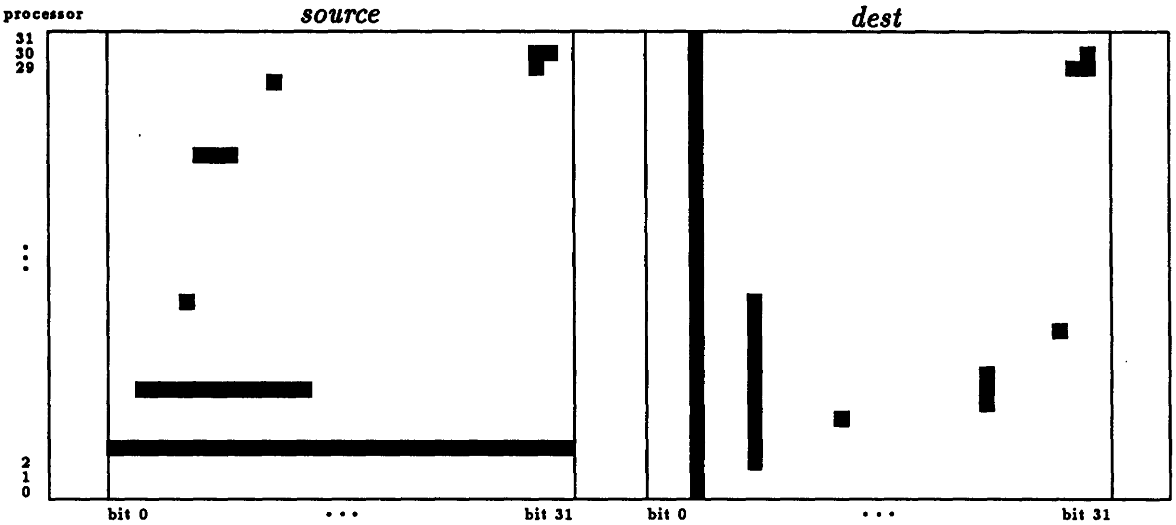
The virtual processors are similarly divided into groups of 32; a group of virtual processors consists of one virtual processor from each physical processor of a cluster, such that the virtual processors occupy the same physical memory locations within their respective physical processors. Thus, two virtual processors are in the same group if their virtual processor numbers agree in all but bit positions n through $n+4$, where n is the number of virtual processors bits in each physical processor.

The CM:transpose32 operation may then be understood as taking the 32 32-bit *source* values from a group of 32 virtual processors as the rows of a 32-by-32 bit matrix, and then storing the columns of this matrix into the *dest* fields of these same virtual processors.

The process may be understood pictorially. Suppose that before the operation the memory of a group of 32 virtual processors looks like this:



Then, after the CM:transpose32 operation, it will look like this:



Knowledge of the internal details of Connection Machine VP memory layout is required to use this instruction properly on *source* values represented in more than 32-bits.

This instruction reorients processor data into a slicewise format that permits rapid, indirect field addressing. A memory region containing transposed data may be viewed either as a single, *shared slicewise array* or as a set of *parallel slicewise arrays.* (See the CM:aref32 and CM:aref32-shared dictionary entries for a description of these data formats.) Viewed as a shared slicewise array, this is especially useful for quickly constructing lookup tables.

Transposition is reversed by applying the CM:transpose32 instruction to a field already stored in the slicewise format. To preserve the correlation between processors and data, this instruction should not be used on slicewise data that was orginally stored by providing CM:aset32 or CM:aset32-shared with an *index-limit* other than 32.

F-F-TRUNCATE

Rounds each source field value to the largest integral value not greater than that value and stores the result as a floating-point number in the destination field.

Formats CM:f-f-truncate-1-1L *dest/source*, *s*, *e*
CM:f-f-truncate-2-1L *dest*, *source*, *s*, *e*

Operands *dest* The field ID of the floating-point destination field.

source The field ID of the floating-point source field.

s, *e* The significand and exponent lengths for the *dest* and *source* fields. The total length of an operand in this format is $s + e + 1$.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two floating-point fields are identical if they have the same address and the same format.

Context This operation is conditional. The destination may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $context\text{-}flag[k] = 1$ then
$dest[k] \leftarrow sign(source) \times \lfloor |source[k]| \rfloor$

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of zero, which is stored into the *dest* field as a floating-point number.

S-F-TRUNCATE

Rounds each floating-point source field value to the largest integer not greater than that value and stores the result as a signed integer in the destination field.

Formats CM:s-f-truncate-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands

dest	The field ID of the signed integer destination field.
source	The field ID of the floating-point source field.
len	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.
s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap The fields *dest* and *source* must not overlap in any manner.

Flags *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    $dest[k] \leftarrow sign(source) \times \lfloor |source[k]| \rfloor$
  if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1 else *overflow-flag*[*k*] ← 0

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of zero, which is stored into the *dest* field as a signed integer.

S-TRUNCATE

The quotient of two signed integer source values, rounded toward zero to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:s-truncate-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:s-truncate-2-1L	*dest*/*source1*, *source2*, *len*
CM:s-truncate-3-1L	*dest*, *source1*, *source2*, *len*
CM:s-truncate-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:s-truncate-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the signed integer quotient field.

source1 The field ID of the signed integer dividend field.

source2 The field ID of the signed integer divisor field.

source2-value A signed integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

dlen For CM:s-truncate-3-3L, the length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen1 For CM:s-truncate-3-3L, the length of the *source1* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

slen2 For CM:s-truncate-3-3L, the length of the *source2* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if divisor is zero; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if *context-flag*$[k] = 1$ then
if *source2*$[k] = 0$ then
dest$[k] \leftarrow$ ⟨unpredictable⟩
else
$$dest[k] \leftarrow sign(source1[k]) \times sign(source2[k]) \times \left\lfloor \left| \frac{source1[k]}{source2[k]} \right| \right\rfloor$$
if ⟨overflow occurred in processor k⟩ then *overflow-flag*$[k] \leftarrow 1$
else *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. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-TRUNCATE

The quotient of two unsigned integer source values, rounded toward zero to the nearest integer, is placed in the destination field. Overflow is also computed.

Formats

CM:u-truncate-3-3L	*dest*, *source1*, *source2*, *dlen*, *slen1*, *slen2*
CM:u-truncate-2-1L	*dest*/*source1*, *source2*, *len*
CM:u-truncate-3-1L	*dest*, *source1*, *source2*, *len*
CM:u-truncate-constant-2-1L	*dest*/*source1*, *source2-value*, *len*
CM:u-truncate-constant-3-1L	*dest*, *source1*, *source2-value*, *len*

Operands

dest The field ID of the unsigned integer quotient field.

source1 The field ID of the unsigned integer dividend field.

source2 The field ID of the unsigned integer divisor field.

source2-value An unsigned integer immediate operand to be used as the second source.

len The length of the *dest*, *source1*, and *source2* fields. This must be non-negative and no greater than CM:*maximum-integer-length*.

dlen For CM:u-truncate-3-3L, the length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen1 For CM:u-truncate-3-3L, the length of the *source1* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

slen2 For CM:u-truncate-3-3L, the length of the *source2* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Overlap

The fields *source1* and *source2* may overlap in any manner. Each of them, however, must be either disjoint from or identical to the *dest* field. Two integer fields are identical if they have the same address and the same length. It is permissible for all the fields to be identical.

Flags

overflow-flag is set if the quotient cannot be represented in the destination field; otherwise it is cleared.

test-flag is set if divisor is zero; otherwise it is cleared.

Context

This operation is conditional. The destination and flags may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor k in the *current-vp-set* do
if $\textit{context-flag}[k] = 1$ then
if $\textit{source2}[k] = 0$ then
$\textit{dest}[k] \leftarrow \langle\text{unpredictable}\rangle$
else
$$\textit{dest}[k] \leftarrow \left\lfloor \frac{\textit{source1}[k]}{\textit{source2}[k]} \right\rfloor$$
if $\langle$overflow occurred in processor $k\rangle$ then $\textit{overflow-flag}[k] \leftarrow 1$
else $\textit{overflow-flag}[k] \leftarrow 0$

The unsigned integer *source1* operand is divided by the unsigned integer *source2* operand. The floor of the mathematical quotient is stored into the unsigned integer memory field *dest*. The various operand formats allow operands to be either memory fields are constants; in some cases the destination field initially contains one source operand.

The *overflow-flag* may be affected by these operations. If overflow occurs, then the destination field will contain as many of the low-order bits of the true result as will fit.

The constant operand *source2-value* should be a signed integer front-end value. Generally the constant has the same length as the field operand it replaces, although this is not strictly required. Regardless of the length of the constant, however, the operation is performed using exactly the number of bits specified by *len*.

U-F-TRUNCATE

Rounds each source field value to the largest integer not greater than that value and stores the result as an unsigned integer in the destination field.

Formats CM:u-f-truncate-2-2L *dest*, *source*, *dlen*, *s*, *e*

Operands	*dest*	The field ID of the unsigned integer destination field.
	source	The field ID of the floating-point source field.
	len	The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.
	s, *e*	The significand and exponent lengths for the *source* field. The total length of an operand in this format is $s + e + 1$.

Overlap: The fields *dest* and *source* must not overlap in any manner.

Flags: *overflow-flag* is set if the result cannot be represented in the *dest* field; otherwise it is cleared.

Context: This operation is conditional. The destination and flag may be altered only in processors whose *context-flag* is 1.

Definition For every virtual processor *k* in the *current-vp-set* do
  if *context-flag*[*k*] = 1 then
    $dest \leftarrow sign(source) \times \lfloor |source| \rfloor$
    if ⟨overflow occurred in processor *k*⟩ then *overflow-flag*[*k*] ← 1

The *source* field, treated as a floating-point number, is rounded to the nearest integer in the direction of zero, and the result is stored into the *dest* field as an unsigned integer.

F-VAX-TO-IEEE

Converts the floating-point source field values from VAX floating-point format to IEEE floating-point format and stores the result in the destination field.

Formats CM:f-vax-to-ieee-1L *ieee-dest*, *vax-source*, *len*

Operands
: *ieee-dest* The field ID of the floating-point destination field.
: *vax-source* The field ID of the floating-point source field.
: *len* The length of the *vax-source* and *ieee-dest* fields. The value of *len* must be either 32 or 64.

Overlap
: The fields *ieee-dest* and *vax-source* may overlap in any manner.

Flags
: *overflow-flag* is set if the vax-source cannot be represented in the destination field; otherwise it is cleared. If *vax-source* is the VAX "undefined variable", the IEEE destination is set to NaN(all 1's) and the *overflow-flag* is cleared. VAX double precision format uses three more mantissa bits than the IEEE double precision format uses. These bits are simply dropped during the conversion. The *overflow-flag* is always cleared for double-precision conversion.

Context
: This operation is conditional. The flag may be altered only in processors whose *context-flag* is 1.

The CM operates internally on floating point data in IEEE format whereas the VAX uses a VAX floating-point format. In each active processor, this function converts a floating-point field in VAX format to a field in standard IEEE format.

The value of *len* specifies the precision of *vax-source*. If *len* is specified as 32, then VAX 'F' format is used. If *len* is specified as 64, then VAX 'D' format is used.

VAX and IEEE floating-point formats are incompatible, so there are a number of potential inaccuracies in the translation. These are described in the flags description above.

This instruction is useful for rapidly converting floating-point data from VAX to IEEE format. For example, if data is transferred from a VAX to a file in the CM file system, CM:f-vax-to-ieee-1L should be called after reading the data file.

All Paris front end to CM data transfer functions automatically convert the data from the front-end format appropriately so it is not necessary to call CM:vax-to-ieee before calling, for instance, one of the write-to-news-array instructions.

To convert data back to VAX floating-point format, see the definition of CM:f-ieee-to-vax-1L.

VP-SET-GEOMETRY

Returns the geometry associated with a given VP set.

Formats result ← CM:vp-set-geometry *vp-set-id*

Operands *vp-set-id* A VP set ID.

Result A geometry ID, identifying the current geometry of the specified VP set.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition Return *geometry*(*vp-set-id*)

The geometry associated with the specified VP set is returned.

WARM-BOOT

This operation is used by the Lisp/Paris interface to reinitialize the Connection Machine system without disturbing user memory.

Formats CM:warm-boot

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation clears error status indicators for the attached Connection Machine hardware. It also clears the IFIFO and OFIFO in the bus interface and possibly loads fresh microcode into the attached microcontroller(s). The user memory areas in the Connection Machine system are not disturbed, but are checked for errors; any memory errors are reported. Certain system memory areas in the Connection Machine system are reinitialized, but the state of the pseudo-random number generator is not altered and the system lights-display mode is not altered. The intent is to recover from an error condition while preserving as much of the machine state as possible.

The facility for warm-booting Connection Machine hardware is provided in different ways in the Lisp/Paris interface (on the one hand) and the C/Paris and Fortran/Paris interfaces (on the other hand).

In the Lisp/Paris interface, CM:warm-boot is a function.

This operation takes no arguments and returns no values. It signals an error if the warm-boot process was not successful.

There are two sets of initializations, kept in the variables CM:*before-warm-boot-initializations* and CM:*after-warm-boot-initializations*, that are evaluated before and after anything else occurs.

In the C/Paris and Fortran/Paris interfaces, there is no CM:warm-boot operation. Instead, a related operation called CM:init is used.

C-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both source and destination values are treated as complex numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:c-write-to-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, dest, s, e, [fe-rank, fe-dimension-vector, format]*

Operands *front-end-array* A front-end array (possibly multidimensional) of complex data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must be of length *fe-rank*.

dest The field ID of the complex destination field. Must have length equal to the rank of the *dest* geometry.

s, e The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $2(s + e + 1)$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array in the front end into a similarly shaped subblock of the NEWS grid. Complex number values are transferred from the specified *front-end-array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which the data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine destination field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the destination field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, from C or Fortran, one of the following predefined complex *format* values may be used: CM_complex_float_single or CM_complex_float_double. For complex data types in C, two front-end elements are used for each Connection Machine element.

When calling Paris from Lisp, the *format* parameter is a keyword argument; for complex transfers only arrays of type t may be used

Definition For all i such that $0 \leq j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$dest[k_i] \leftarrow \textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
  for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
    for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$dest[k_{s_0, s_1, \ldots, s_{rank-1}}] \leftarrow$$
$$\textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}}$$

F-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both source and destination values are treated as floating-point numbers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:f-write-to-news-array-1L *front-end-array*, *fe-offset-vector*, *cm-start-vector*, *cm-end-vector*, *cm-axis-vector*, *dest*, *s*, *e*, *[fe-rank*, *fe-dimension-vector*, *format]*

Operands

front-end-array A front-end array (possibly multidimensional) of floating-point data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must have length equal to the rank of the *dest* geometry.

dest The field ID of the floating-point destination field.

s, *e* The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array in the front end into a similarly shaped subblock of the NEWS grid. Floating-point number values are transferred from the specified *array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which the data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine destination field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the destination field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined floating-point *format* values may be used. These are CM_float_single or CM_float_double from C or Fortran, and :float-single or :float-double from Lisp.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the Connection Machine field.

Definition For all i such that $0 \leq j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$dest[k_i] \leftarrow \textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do

for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do

for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$dest[k_{s_0, s_1, \ldots, s_{rank-1}}] \leftarrow$$

$$\textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}}$$

S-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both the source and destination values are treated as signed integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:s-write-to-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, dest, len, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of signed integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must have length equal to the rank of the *dest* geometry.

dest The field ID of the signed integer destination field.

len The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array from the front end into a similarly shaped subblock of the NEWS grid. Signed integer values are transferred from the specified *array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which the data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

When calling Paris from Lisp, the array may be either a general array (of type t) containing signed integers, or a specialized integer-element array (such as an array of type (unsigned-byte 8)).

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if $cm\text{-}axis\text{-}vector[A] = B$, then axis A of the Connection Machine destination field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the destination field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined signed *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_1_bit_packed, CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined signed format keywords are :8-bit, :16-bit, :32-bit, :1-bit-packed, :2-bit-packed, and :4-bit-packed.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the Connection Machine field.

Definition For all i such that $0 \leq j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$dest[k_i] \leftarrow \textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do

for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do

for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0, s_1, \ldots, s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$dest[k_{s_0, s_1, \ldots, s_{rank-1}}] \leftarrow \textit{front-end-array}_{offset_0 + s_0, offset_1 + s_1, \ldots, offset_{rank-1} + s_{rank-1}}$$

U-WRITE-TO-NEWS-ARRAY

Copies a subarray of an array in the memory of the front end into a field within a set of processors forming a subarray (of the same shape) of the NEWS grid. Both the source and destination values are treated as unsigned integers.

Note: The read-from-news-array and write-to-news-array operations do *not* require that the specified CM field be in the current VP set.

Formats CM:u-write-to-news-array-1L *front-end-array, fe-offset-vector, cm-start-vector, cm-end-vector, cm-axis-vector, dest, len, [fe-rank, fe-dimension-vector, format]*

Operands

front-end-array A front-end array (possibly multidimensional) of unsigned integer data.

fe-offset-vector A front-end vector of signed integer subscript offsets for the *front-end-array*. Must be of length *fe-rank*.

cm-start-vector A front-end vector of signed integer inclusive lower bounds for NEWS indices. Must be of length *fe-rank*.

cm-end-vector A front-end vector of signed integer exclusive upper bounds for NEWS indices. Must be of length *fe-rank*.

cm-axis-vector A front-end vector of signed integer numbers indicating NEWS axes. Must have length equal to the rank of the *dest* geometry.

dest The field ID of the unsigned integer dest field.

len The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

fe-rank A signed integer, the rank (number of dimensions) of the *front-end-array*. This argument is not provided when calling Paris from Lisp.

fe-dimension-vector A front-end vector of signed integer dimensions of the *front-end-array*. This argument is not provided when calling Paris from Lisp. Must be of length *fe-rank*.

format The array descriptor for *front-end-array*. This is a keyword argument when calling Paris from Lisp.

Context This operation is unconditional. It does not depend on the *context-flag*.

This operation copies a rectangular subblock of an array from the front end into a similarly shaped subblock of the NEWS grid. Unsigned integer values are transferred from the specified *array* to the Connection Machine processors.

The *dest* parameter specifies the memory address within each processor of the field into which data is stored.

The *front-end-array* parameter specifies the front-end source array from which one element is copied to each processor specified by *dest*.

The *fe-rank* parameter specifies the rank of the front-end array and is normally equal to the rank of the destination field geometry. When calling Paris from Lisp, this value can be deduced from the value of *front-end-array* and must not be specified.

The vector arguments are one-dimensional front-end arrays.

The *fe-dimension-vector* parameter specifies the dimensions of the front-end array. These dimensions are measured in units of *array-element-size*, which is implicitly specified by *format*. (See the description of *format* below.) When calling Paris from Lisp, the front-end array dimensions can be deduced from the value of *front-end-array* and must not be specified.

The *fe-offset-vector* parameter contains the coordinate of the first front-end array element transferred to the Connection Machine. The length of this argument is measured in units of *cm-element-size*, except during an extended array transfer – when it is measured in units of (*stride* × *array-element-size*). Notice that *cm-element-size*, *array-element-size*, and *stride* are parameters to the operations that return the *format* array descriptor. (See the description of *format* below.)

The *cm-start-vector* parameter specifies the coordinate of the first CM element to receive data from the front end. The *cm-end-vector* parameter specifies the coordinate of the last CM element to receive data from the front end. Both of these are permuted by by the values in *cm-axis-vector*.

The *cm-axis-vector* parameter specifies how Connection Machine axes are mapped to front-end array axes. For example, if *cm-axis-vector*$[A] = B$, then axis A of the Connection Machine source field geometry is mapped to axis B of the front-end array. The length of this vector must be equal to the rank of the source field geometry.

The *format* parameter is an array descriptor that specifies the format of the front-end array. An appropriate descriptor may be obtained by a call to CM:array-format, CM:packed-array-format, or CM:structure-array-format. Alternatively, one of the predefined unsigned *format* values may be used.

From C or Fortran a value of CM_8_bit, CM_16_bit, or CM_32_bit specifies an unpacked front-end array while CM_1_bit_packed, CM_2_bit_packed, or CM_4_bit_packed specifies a front-end array in which several CM elements are packed into each array element. From Lisp, the predefined unsigned format keywords are :8-bit, :16-bit, :32-bit, :1-bit-packed, :2-bit-packed, and :4-bit-packed.

When calling Paris from Lisp, the *format* parameter is a keyword argument. If not specified, it defaults based on the element type of the front-end array or, if the array is of type t, based on the type of the Connection Machine field.

Definition For all i such that $0 \leq j < \prod_{j=0}^{rank-1} (end_j - start_j)$ do

for all m such that $0 \leq m < rank$ do

$$\text{let } s_{\langle i,m \rangle} = \left\lfloor \frac{i}{\prod_{j=m+1}^{rank-1} (end_j - start_j)} \right\rfloor \bmod (end_m - start_m)$$

$$\text{let } k_i = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_{i,j})$$

$$dest[k_i] \leftarrow \textit{front-end-array}_{s_{\langle i,0 \rangle}, s_{\langle i,1 \rangle}, \ldots, s_{\langle i,rank-1 \rangle}}$$

Another formulation:

For all s_0 such that $0 \leq s_0 < (end_0 - start_0)$ do
  for all s_1 such that $0 \leq s_1 < (end_1 - start_1)$ do
    for all s_2 such that $0 \leq s_2 < (end_2 - start_2)$ do

$\ddots$

for all s_{rank-1} such that $0 \leq s_{rank-1} < (end_{rank-1} - start_{rank-1})$ do

$$\text{let } k_{s_0,s_1,\ldots,s_{rank-1}} = \bigvee_{j=0}^{rank-1} \textit{make-news-coordinate}(axis_j, start_j + s_j)$$

$$dest[k_{s_0,s_1,\ldots,s_{rank-1}}] \leftarrow \textit{front-end-array}_{offset_0+s_0,offset_1+s_1,\ldots,offset_{rank-1}+s_{rank-1}}$$

C-WRITE-TO-PROCESSOR

Stores an immediate complex number operand value into the destination field of a single specified processor.

Formats CM:c-write-to-processor-1L *send-address-value*, *dest*, *source-value*, *len*

Operands *send-address-value* An immediate operand, the send address of a single particular processor.

dest The field ID of the complex destination field.

source-value A complex immediate operand to be used as the source.

s, *e* The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $2(s + e + 1)$.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition $dest[send\text{-}address\text{-}value] \leftarrow source\text{-}value$

The specified *source-value*, a complex number, is stored into the *dest* field of the processor whose send address is the immediate operand *send-address-value*.

The constant operand *source-value* should be a double-precision front-end value (in Lisp, automatic coercion is performed if necessary).

F-WRITE-TO-PROCESSOR

Stores an immediate floating-point number operand value into the destination field of a single specified processor.

Formats CM:f-write-to-processor-1L *send-address-value*, *dest*, *source-value*, *s*, *e*

Operands *send-address-value* An immediate operand, the send address of a single particular processor.

dest The field ID of the floating-point destination field.

source-value A floating-point immediate operand to be used as the source.

s, *e* The significand and exponent lengths for the *dest* field. The total length of an operand in this format is $s + e + 1$.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *dest*[*send-address-value*] ← *source-value*

The specified *source-value*, a floating-point number, is stored into the *dest* field of the processor whose send address is the immediate operand *send-address-value*.

S-WRITE-TO-PROCESSOR

Stores an immediate signed integer operand value into the destination field of a single specified processor.

Formats CM:s-write-to-processor-1L *send-address-value*, *dest*, *source-value*, *len*

Operands

send-address-value	An immediate operand, the send address of a single particular processor.
dest	The field ID of the signed integer destination field.
source-value	A signed integer immediate operand to be used as the source.
len	The length of the *dest* field. This must be no smaller than 2 but no greater than CM:*maximum-integer-length*.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *dest*[*send-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 field ID of the unsigned integer destination field.

source-value An unsigned integer immediate operand to be used as the source.

len The length of the *dest* field. This must be non-negative and no greater than CM:*maximum-integer-length*.

Context This operation is unconditional. It does not depend on the *context-flag*.

Definition *dest*[*send-address-value*] ← *source-value*

The specified *source-value*, an unsigned integer, is stored into the *dest* field of the processor whose send address is the immediate operand *send-address-value*.

The
Connection Machine
System

Paris Release Notes

Version 6.1
January 1992

Thinking Machines Corporation
Cambridge, Massachusetts

First printing, January 1992

Thinking Machines Corporation
245 First Street
Cambridge, Massachusetts 02142–1264
(617) 234–1000/876–1111

Contents

List of Tables v
About Version 6.1 Paris Release Notes vii
Customer Support xi

1 About Paris Version 6.1 .... **1**

1.1 Microcode Version 1
1.2 On-Line Documentation 1
1.3 Layered Software Compatibility 2
Front-End Operating Systems 2
High-Level CM Languages and Libraries 2
1.4 CM-2 Hardware Options Supported 2
Memory Chips 2
Floating-Point Accelerators 3
Double-Precision FPA Performance 3
Accuracy of Floating-Point Operations 4
1.5 Back Compatibility 5
Back-Compatibility Mode 5

2 New and Enhanced Instructions **5**

3 Enhanced Performance **6**

4 Implementation Restrictions **6**

4.1 Field Allocation 6
4.2 Operand Lengths 7
4.3 Instructions Use Stack Memory 8
4.4 Incomplete Support for IEEE Floating-Point 9
4.5 Integer Flags 10

5 Implementation and Documentation Errors **10**

5.1 Known Errors Corrected 10

5.2 Known Errors Outstanding 10

cm:power-up-ignores-nexus-clockspeed 11

cross-vp-move-F77-constants-missing 11

cross-vp-move-breaks-CMF/Paris-array-section-transfers 12

field-length-doc-misleading 13

no-dest-overlap-for-sends-or-gets 13

6 Debugging Hint **14**

paris-safety-hint 14

Appendix A CM-2 Performance Notes **15**

A.1 General Router Communication 15

Send Speed 16

Get Speed 17

A.2 Arithmetic Timing Tables 18

Appendix B Paris Version 6.1 Change Pages **51**

B.1 What to Do with These Change Pages 51

List of Figures

Table 1. Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's **19**

Table 2. Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's **35**

About Version 6.1 Paris Release Notes

Objectives

These release notes describe how the Paris language fits into the existing suite of Connection Machine programming languages. In addition, language features new with this release are enumerated and change pages are provided. New timing tables for Paris arithmetic operations running on a CM-2 are also included.

Intended Audience

The Paris language and its documentation are intended for experienced developers of Connection Machine Models CM-2 and CM-200 system software and applications. Read this document if you are using the Paris library contained in CM-2 System Software Version 6.1.

Revision Information

These Version 6.1 Paris release notes supersede all previous Paris release notes and they supplement the *Paris Reference Manual*, Version 6.0. A current *Programming in Paris* binder, should contain the following documents:

- *Paris Reference Manual* Version 6.0, February 1991, as updated with the Version 6.1 change pages contained in the release notes
- *Paris Release Notes* Version 6.1, January 1992

Organization of This Manual

Seven sections make up these release notes, as described below.

1. **About Paris Version 6.1**

   Paris is introduced, the microcode version number is stated, instructions for using the on-line manual pages are given, and compatibility between Paris and specific layered software product versions is detailed. The CM-2 chip options supported by Paris are described along with a permanent restriction on 32-bit FPAs. Back-compatibility is discussed.

2. **New and Enhanced Instructions**

   The Paris language features that are new or enhanced as of Version 6.1 are briefly described.

3. **Enhanced Performance**

   The major performance enhancements offered with Version 6.1 are discussed.

4. **Implementation Restrictions**

   Restrictions imposed by the CM-2 Paris implementation are detailed. These include restrictions on field allocation size, operand lengths, IEEE floating-point instructions and flags, and integer flags.

5. **Implementation and Documentation Errors**

   Known errors corrected with this release are listed; known errors that remain outstanding are explained.

6. **Debugging Hint**

   The importance of using Paris safety checking while developing CM-2 application code is emphasized.

Appendix A CM-2 Performance Notes

Helpful information about CM-2 performance is presented. General router communication is discussed, followed by a set of timing tables for Paris arithmetic instructions.

Appendix B Paris Version 6.1 Change Pages

This is a set of dictionary pages for Paris instructions that are either new or changed with Version 6.1. Follow the instructions to update your *Paris Reference Manual* with these change pages.

Related Documents

The following related documents are helpful to Paris programmers.

- *CM User's Guide* Version 6.1, printed October 1991
- *Introduction to Programming in C/Paris* Version 5.0, printed June 1989

Notation Conventions

The table below displays the notation conventions observed in Paris documentation.

Convention	Meaning
bold typewriter	UNIX and CM System Software commands, command options, and filenames, when they appear embedded in text.
bold sans serif	Language elements, such as keywords, operators, and function names, when they appear embedded in text.
italics	Argument names and placeholders in function and command formats.
`typewriter`	Code examples and code fragments.

All Paris Version 6.*x* documentation follows the conventions for alphabetizing, syntax, and pseudocode established at the beginning of Chapter 9 of the *Paris Reference Manual* Version 6.0. One further convention is observed.

In the Formats portion of dictionary entries, brackets, [and], enclose arguments that—for the Lisp/Paris interface—are either optional, not provided, or are keywords. Wherever this notation is used, the Operands list explicitly states whether the brackets enclose unprovided, optional, or keyword arguments. For example, in the format line

Formats result ← **CM:intern–geometry** *dimension–array, [rank]*

the *rank* operand is not provided when calling Paris from Lisp.

1 About Paris Version 6.1

The Paris language is a relatively low-level instruction set designed for programming the Connection Machine models CM-2 and CM-200. It provides a large number of operations similar to the machine-level instruction set of a serial computer. Paris implements the virtual processing paradigm, whereby each of up to 65,536 physical processors can simulate multiple processors. Intended primarily as the basis for higher-level Connection Machine languages, Paris may nonetheless be called directly from standard Lisp, C, or Fortran as well as from the Connection Machine languages CM Fortran, C*, and *Lisp.

Paris Version 6.1 provides an expanded instruction set, significant performance improvements for communication operations, and corrections to a number of implementation errors.

1.1 Microcode Version

Paris Version 6.1 is shipped as part of the CM base system software (CMSS) microcode version 6104. This microcode version number is used as part of certain commands to load or link CM languages or libraries. For more information check the documentation provided with the CM languages and libraries you are using.

1.2 On-Line Documentation

As of Version 6.1, on-line documentation of all Paris instructions is available. Use the **cmman** command to display the manual pages for an instruction. Thus, at the UNIX prompt,

```
% cmman CM_<rootname>
```

displays the man page for the Paris instruction **CM_*<rootname>***, where ***<rootname>*** is the name that appears at the top of a dictionary page in the *Paris Reference Manual*. You can omit the **_1L**, **_2L**, etc suffixes, as well as the **_always**, **_constant**, **_const** qualifiers. If you don't want to use **cmman**, you may use the UNIX **man** command or the **xman** facilities. See Chapter 3 of the *CM User's Guide* for further information. Note that the conceptual material at the beginning of the *Paris Reference Manual* is not yet available on line.

1.3 Layered Software Compatibility

Front-End Operating Systems

- Sun-4 front ends require SunOS Version 4.1 or 4.1.1 in order to run CM System Software (CMSS) Version 6.1. We recommend using SunOS 4.1.1 if possible.
- VAX front ends requires ULTRIX Version 4.2 in order to run CMSS Version 6.1.

High-Level CM Languages and Libraries

The following versions of CM languages and libraries run with CMSS Version 6.1 and therefore may be used with Paris 6.1:

- CM Fortran Version 1.1
- C* Version 6.0.2
- *Lisp Version 6.1
- Visualization Version 2.0
- CMSSL Version 2.2.1

1.4 CM-2 Hardware Options Supported

Memory Chips

- 256K chips, sometimes called *small memories*, which provide 64K bits/processor
- 1M chips, sometimes called *large memories*, which provide 256K bits/processor

- 4M chips, sometimes called *jumbo memories*, which provide 1M bits/processor

Floating-Point Accelerators

CM-2 model Connection Machine systems may be configured either without floating-point accelerator units or with one floating-point accelerator unit (FPA) for every 32 CM physical processors. Two kinds of FPA chip are supported by the current system software: 32-bit and 64-bit floating-point accelerators, known as single-precision and double-precision FPAs.

CM-200 model Connection Machine systems are always configured with one 64-bit FPA for every 32 CM physical processors.

For the majority of Paris instructions, the performance differences between systems without FPAs, systems with single-precision FPAs, and systems with double-precision FPAs are in the 5% to 10% range. Some instructions, however, exhibit substantial performance variations across machine configurations. These cases are described below.

Double-Precision FPA Performance

On a CM-2 configured with 64-bit FPAs and running CM System Software Version 6.*x*, double-precision floating-point instructions are approximately 20 times faster than on machines that either have no FPAs or have 32-bit FPAs.

On a CM-2 configured with 64-bit FPAs and running CM System Software Version 6.*x*, single-precision floating-point operations are approximately twice as fast as double-precision floating-point operations. The only exceptions are the single-precision tangent, arctangent, arcsine, and arc cosine instructions, which are approximately three times faster than their double-precision counterparts on 64-bit FPAs.

In addition, 64-bit performance differs from 32-bit performance in the following ways:

- Basic arithmetic without constants is 5–8% slower with 64-bit FPAs than with 32-bit FPAs
- Basic arithmetic with constants is 8–25% faster with 64-bit FPAs than with 32-bit FPAs
- Transcendental functions are about as fast with 64-bit FPAs as with 32-bit FPAs, with a 5% variance in either direction

Accuracy of Floating-Point Operations

On machines configured either with 64-bit FPAs or with no FPAs, all single- and double-precision floating-point operations are accurate and—where IEEE standards exist— are IEEE compliant. For most operations, this is also true of machines configured with 32-bit FPAs. The exceptions are described below.

This is a permanent restriction of the 32-bit FPA chip: On Model CM-2 Connection Machines configured with 32-bit FPAs, single-precision floating-point divide and square root operations are not IEEE compliant. Similarly, single-precision floating-point sine and cosine operations are not as accurate as on machines configured otherwise (there are no IEEE standards for the accuracy of transcendental functions). In all these cases the accuracy is off by one or two bits only and thus, for most applications, presents no problem.

To get around this restriction, you may want to sacrifice speed for accuracy and change your code in one of the two following ways:

1. Use only double-precision divide, square root, sine, and cosine operations.

2. Turn off the FPA before calling a single-precision divide, square root, sine, or cosine operation, and turn it back on afterwards. Thus, to call **CM:f-cos-1-1L**, for instance,

   from Lisp/Paris, wrap the call in a let form like so:

```
(let ((cmi::*wtl3132-p* nil)
      (cmi::*sprint-chip-p* :sprint))
            (CM:f-cos-1-1L field 23 8 ))
```

   from C/Paris (or similarly from CM Fortran/Paris or Fortran/Paris) switch the variable **_CMI_wtl3131_p** off and then on again:

```
extern int _CMI_wtl3132_p;
{
   int old_wtl;

   old_wtl = _CMI_wtl3132_p;

   _CMI_wtl3132_p = 0;
   CM_f_cos_1_1L(field 23 8);

   _CMI_wtl3132_p = old_wtl;
}
```

1.5 Back Compatibility

Version 6.1 supports all documented instructions provided in Versions 4.*x*, 5.*x*, and 6.*x* to date.

Back-Compatibility Mode

Any existing programs that call Paris 4.*x* instructions must be recompiled and relinked with the new Paris object library and then run in back-compatibility mode. Back-compatibility mode implements the 4.*x* stack discipline by allocating the stack in field zero and making stack address offsets into this field. See Appendix A in the *CM User's Guide* for information on executing programs in back-compatibility mode.

Be forewarned: There will be no support for back-compatibility mode after this release.

2 New and Enhanced Instructions

Paris Version 6.1 introduces the following new instructions:

- **CM:permuted-send-1L** and **CM:permuted-get-1L**

  These instructions are alternatives to **CM:send-1L** and **CM:get-1L**. Use them in cases where congested communication patterns cause the original send and get routines to perform poorly.

- **CM:send-to-shared-queue32-1L**

  This instruction sends a message from every selected processor to a specified destination sprint node and stores it there in a queue.

In addition, a related instruction is enhanced in this version:

- **CM:send-to-queue32-1L**

  This instruction sends a message from every selected processor to a specified destination processor and stores it there in a queue. It now supports messages of lengths 32, 64, 96, or 128 bits. (Only 32-bit messages were supported in Version 6.0.)

These four instructions are documented in the change pages included at the end of these release notes. Please insert the change pages into your *Paris Reference Manual.*

3 Enhanced Performance

Paris Version 6.1 includes several enhancements, which together make Paris code easier to write and faster to execute.

- **Better Communication Performance.** In the area of communications, Paris Version 6.1 offers significant performance improvements. In particular:
  - Version 6.1 includes completely rewritten router microcode.
  - **CM:cross-vp-move-1L** gains substantially improved performance due to its use of **CM:permuted-send-1L**.
  - Use of a "divided get" strategy drastically improves the performance of the whole family of **get** instructions under limited memory conditions.

    A **get** instruction that in previous versions would have run out of memory now succeeds. If the system detects insufficient memory, it will divide the data into two or more chunks, and transfer it in that manner. The performance penalty is slight.
- **C/Paris Error Handler Improved.** The C/Paris Error handler has been enhanced to assist in debugging several obscure hardware problems.
- **Support for Shared Libraries.** On Sun front ends, a dynamically linked, shared version of the Paris library is available, greatly reducing application size and link time. (This will be available with patch release Version 6.1.1, expected in late January, 1992.)

4 Implementation Restrictions

4.1 Field Allocation

In Version 6.1, Paris field allocations are supported up to, but not including, 64K bits. Thus the largest Paris field that may be allocated is 65535 bits long. On a CM with memory chips that provide either 256K or 1M bits/processor, therefore, it may not be possible to allocate a field with a length equal to the value returned by the **CM:available-memory** function.

4.2 Operand Lengths

Paris instructions do not take arbitrarily long fields as operands. Also, almost all operand fields must have lengths greater than zero.

As noted in the *Paris Reference Manual*, Section 3.7 "Configuration Variables," the only field lengths guaranteed to work for any operand to any Paris instruction from one release to the next are those less than or equal to **CM:*maximum–integer–length*** (128) for integer fields and less than or equal to **CM:*maximum-significand-length*** + **CM:*maximum-exponent-length*** (96 + 32) for floating-point fields. Some floating-point operations, such as the transcendental and trigonometric functions, are further limited to work only for standard floating-point lengths of 32 and 64 bits (as noted in the appropriate Paris Reference Manual dictionary entries).

In Versions 6.*x*, certain Paris instructions will work for fields longer than the guaranteed maximums. The limits to which an instruction is subject can generally be determined by considering what kind of instruction it is:

- Arithmetic operations that require the implementation of complicated algorithms that use internal "scratch" memory are affected by a fuzzy limit between 255 and 1,500 bits. Examples include multiplication and division, which must handle carry and remainder bits. These are limited by the size of scratch memory and by the way they use it. In general, such instructions are limited to lengths up to 255 bits.

- The basic mathematical instructions (addition, subtraction) and the bitwise logical operations are limited by the size of the length argument they can receive. In Versions 6.*x*, most Paris length arguments are limited to 12 bits.[1] If a longer length argument is provided to Paris, only the 12 low-order bits are passed to microcode functions. Since the maximum value that can be represented in 12 bits is 4095, the maximum operand length for these Paris instructions is now 4095.

- The **move** and **swap** instructions, as well as the **read-from-processor** and **write-to-processor** instructions, take 16-bit lengths (with one exception[2]). Thus, these instructions can address the maximum field length (65535 bits). Please note that on large memory machines (either 256K or 1M bits/processor), a VP ratio of 2 or more is required to physically move more than 65535 bits at a time per physical processor with the Paris move instructions.

---

1. This restriction has been in place since the release of Version 5.2 and was first reported in *In Parallel* of March 1990.
2. The **CM:f–move–2L** operation is limited to 12-bit lengths and can therefore only work with fields up to 4095 bits long.

- The **send** and **get** instructions are generally constrained by the constant **CM:*maximum–message–length***, which has been defined as 128. This constant is an upper bound on the number of bits transferred between processors by certain router instructions. The **CM:*maximum–message–length*** restriction applies to the following Version 6.x router instructions:

  CM:send–with–f–max–1L
  CM:send–with–f–min–1L
  CM:send–with–f–add–1L
  CM:send–aset32–overwrite–1L
  CM:send–aset32–u–add–1L
  CM:send–aset32–logior–1L
  CM:get–aref32

- The following Version 6.x router instructions have *no* message length upper bound; their message size is limited only by available memory:

  CM:get–1L
  CM:permuted–send–1L
  CM:send–1L
  CM:send–with–overwrite–1L
  CM:send–with–logxor–1L
  CM:send–with–logior–1L
  CM:send–with–logand–1L
  CM:send–with–u–min–1L
  CM:send–with–u–max–1L
  CM:send–with–s–min–1L
  CM:send–with–s–max–1L
  CM:send–with–u–add–1L
  CM:send–with–s–add–1L

In general, Paris fields are assumed to have lengths greater than zero. (See the *Paris Reference Manual*, Section 2.4.) The only Paris operations that are guaranteed to work with operand fields of zero length are the unsigned move instructions (**CM:u–move–1L, 2L** et al) and **CM:allocate–stack–field.**

4.3 Instructions Use Stack Memory

Most Paris instructions use some temporary memory space allocated on the stack. Stack memory use falls into three categories: constant, proportional to VP ratio, and unbounded. It is possible for a program to run out of stack space while executing an instruction that falls into any of these categories. If this happens, the program will fail with a message indicating that there is insufficient temporary memory. Instructions that fall within the last category are most likely to exhaust memory. These include gets, scans, ranks, and some sends. Solutions include attaching to a bigger portion of the CM, upgrading to larger memory chips, and changing data layouts to reduce VP ratios or restructure communication patterns.

4.4 Incomplete Support for IEEE Floating-Point

Support for IEEE floating-point instructions and flags is incomplete in Paris. In particular:

- the five IEEE floating-point flags are not supported
- denormalized numbers are not supported
- **Infinity** and **NaN** values are only partially supported

Also, all Version 6.*x* floating-point instructions:

- set the integer *test–flag* and the integer *overflow–flag* if division by zero occurs, and otherwise leave them unaffected
- set the integer *overflow–flag* if floating-point overflow occurs, and otherwise leave the *overflow–flag* unaffected
- produce a zero result on underflow, with no other indication

When floating-point overflow occurs, the value stored in the destination field varies depending on the floating-point hardware present.

A floating-point overflow on a machine equipped with double-precision floating-point accelerators (FPAs) produces the IEEE overflow "biased" result (see IEEE spec Std 754–1985). On machines not equipped with the double-precision FPAs, the result will be either 0.0, or a quiet **NaN** (plus or minus infinity).

For this reason, we recommend that you avoid writing code that depends on the resultant values in overflow conditions.

4.5 Integer Flags

All 6.*x* integer operations:

- set the *overflow flag* if an integer overflow occurs and otherwise clear it

  On overflows, bits up to the destination length are correctly set. The few exceptions to this rule are noted in the appropriate *Paris Reference Manual* dictionary entries.

- set the *test flag* if an integer divide by zero occurs and otherwise clear it

- produce a zero result on underflow, with no other indication

5 Implementation and Documentation Errors

5.1 Known Errors Corrected

The following implementation errors, reported in the *Revised Paris Release Notes*, February 1991, are fixed in Paris Version 6.1:

f–move–constant–0.0–slow
long–sends–with–notify–fail
no–segment–bits–for–rank

5.2 Known Errors Outstanding

All known bugs that remain unrepaired in Paris Version 6.1 are detailed below.

ID **cm:power-up-ignores-nexus-clockspeed**

DDTs ID: TMCaa00621 (cmos)

Environment

Any CM-200 configuration; any front end; Lisp/Paris Version 6.1.

Description

(cm:power-up) ignores the nexus clock speed parameter in the CM configuration file (`configuration.lisp`). It always chooses crystal 0 when setting the CM nexus speed.

Workaround

Explicitly specify which speed you want on powerup.

Status

Fixed in the upcoming patch release 6.1.1. Meanwhile, a patch is available from Thinking Machines Corporation Customer Support.

ID **cross-vp-move-F77-constants-missing**

DDTs ID: TMCaa00232 (paris)

Environment

Any CM-2 configuration; any front end; F77/Paris Version 6.1.

Description

From F77/Paris, the **CM:cross-vp-move-1L** instruction does not work with its documented named constants.

The named constants **CM_cvpm_indexed** and **CM_cvpm_mapped** do not appear in the include file `paris-configuration-fort.h`. They are however defined in the **paris.h** include file. These named constants are documented in the *Paris Reference Manual*, Version 6.0 dictionary entry for **CM:cross-vp-move-1L**.

Workaround

For either constant, substitute **CM_no_axis**, which also represents the null value. Alternatively, explicitly define the constants for your application by copying the **paris.h** definitions.

Status

Open

ID cross-vp-move-breaks-CMF/Paris-array-section-transfers

DDTs ID: TMCaa00617 (paris)

Environment

Any CM-2 configuration; any front end; fieldwise CMF/Paris Version 6.1.

Description

A bug in **cross-vp-move** causes CMF/Paris array section transfers to fail.

Workaround

Set the internal variable `_CMI_cvpm_mode = 1` (meaning `:cvpm-go-slow`)

Status

Fixed in the upcoming patch release 6.1.1 and in Version 6.2. Meanwhile, a patch is available from Thinking Machines Corporation Customer Support.

ID field–length–doc–misleading

Description

1. All Paris field operands must have lengths greater than zero (with two exceptions, described below). All *Paris Reference Manual* dictionary entries with field length definitions (*len*, *dlen*, *slen*) that read "This must be non-negative" should read "This must be greater than zero."

2. One exception to the positive-length-field rule is that unsigned **move–** operations may take operands of length zero. The **U–MOVE** dictionary entry definitions for *len*, *dlen*, and *slen* each read "This must be no smaller than 2," whereas they should read, "This must be no smaller than zero."

3. A second exception to the positive-length-field rule is that each of the **allocate–** instructions (**allocate–heap–field**, **allocate–stack–field**, etc.) may take a *len* argument of zero. This permits the trick of using the ID of a zero-length field as a stack/heap pointer.

Status

1 and 2 are fixed in the Version 6.1 UNIX man pages for Paris and in the next edition of the *Paris Reference Manual.*

The documentation for 3 was never in error but is clarified in the next edition of the *Paris Reference Manual.*

ID no–dest–overlap–for–sends–or–gets

DDTs ID: TMCaa00662 (paris)

Description

For all Paris **send** and **get** instructions, the "Overlap" descriptions in the *Paris Reference Manual* Version 6.0 are in error. (However, the V6.1 change pages for the new **permuted–send** and **permuted–get** operations are correct.) When invoking any **send** or **get**, only the *source* and *send–address* fields may overlap. No overlap between the *dest* field and either the *source* or *send–address* is permitted.

Status

Fixed in next edition of the *Paris Reference Manual* (V6.2)

---

6 Debugging Hint

Here is a hint for effective C/Paris debugging.

ID paris–safety–hint

Environment

Paris, Version 5.*x* or 6.*x*; UNIX front end, using the C-shell; any CM configuration.

Description

Paris safety checking can be turned on by default. When the C/Paris library is linked with C code, Paris safety checking is turned off by default. To speed the debugging process, turn safety checking on.

To turn Paris safety checking on by default, add the following line to your **.cshrc** file:

```
if ($?CMDEVICE) cmsetsafety on
```

This line turns Paris safety on each time you use a **cmattach** subshell.

Appendix A

CM-2 Performance Notes

Here we offer information to help you predict CM-2 performance for two classes of Paris instructions. First, we discuss general router communication, broadly explaining the factors that determine execution speed for this class of instructions. Next, a table of test timings for Paris Version 6.1 arithmetic instructions is presented.

A.1 General Router Communication

The Paris **send** and **get** instructions are among the most powerful operations available on the Connection Machine system. The **send** and **get** instructions without **–news** in their names are collectively referred to as *general router communications*. They allow any processor to send or receive data from any other processor. (See the "General Communication" section of the *Paris Reference Manual*, Version 6.0, Chapter 5.)

While powerful, general router communications are among the longest-running Paris operations provided. CM-2 programmers are therefore encouraged to use general router communications judiciously. Wherever appropriate, NEWS communication (accomplished with instructions whose names include the term **–news**) should be used.

The time required to execute a general router communication instruction depends primarily on the degree of router "traffic congestion" induced by a particular instruction invocation. Router congestion is caused by complex communication patterns and by high VP ratios. The *permuted* **send** and **get** instructions should be used when router congestion is predictably high. (**CM:permuted–send–1L** and **CM:permuted–get–1L** are new with Version 6.1; see the provided change pages.)

Guidelines helpful in predicting the performance of general router communication instructions are provided below.

Send Speed

To a first approximation, the time required to do a Paris **send** operation is controlled by the following factors:

- Communication pattern complexity
- VP ratio
- Message length
- Specific instruction implementation

Communication pattern complexity. The relative locations of the source and destination processors determine the degree of congestion. If, at a particular time during the send, many messages must travel over the same path, then communication is slower than if, at a particular time during the send, message paths are evenly distributed across the machine.

The congestion induced within the CM-2 router by a particular communication pattern is quantified by the number of internal router cycles (termed *petit cycles*) required to complete a send. In general, most patterns are low congestion patterns and take some small number of petit cycles—less than a random pattern takes.

An example of an extremely low congestion pattern is one that emulates NEWS—that is, a pattern in which each virtual processor sends a message to one of its neighbors. Low congestion patterns involve many-to-many communication.

A high congestion pattern involves many-to-few communication at some point in the send. That is, a pattern in which all or many processors send messages to virtual processors on the same physical chip takes many petit cycles to complete. For instance, while matrix transposition appears to require many-to-many communication, at a VP ratio greater than one, it tends to create high congestion. Why? If the matrix rows are stored across a set of physical processors and the columns are stored in virtual processor banks within these processors, then sending a whole row to a single column is many-to-few communication. Use a permuted send to increase the execution speed of this type of communication.

It is interesting to note that while most regular patterns are low congestion patterns, most high congestion patterns are regular patterns.

VP ratio. The higher the VP ratio, the more messages are likely to be sent across the same router paths. The number of petit cycles required to perform any given send instruction increases in roughly linear proportion with the VP ratio.

Message length. The duration of each petit cycle is a fixed overhead plus a certain amount per bit of data sent, so doubling a message length less than doubles the router time required to send the message. The minimum message length handled is approximately 25 bits; fewer bits may be sent, but this is no faster than sending 25. Messages over approximately 128 bits long are transferred as multiple messages, which can substantially slow and complicate a send operation.

Generally, for message lengths within the range of approximately 25 to 128 bits, it takes less time to send one long message than to send several short ones.

Specific instruction implementation. The exact operations performed by a specific send instruction affect execution time.

For example, the instruction **CM:send–with–f–add–1L** takes longer than its integer counterparts. Before the floating-point data is transmitted by the router, it is denormalized to a fixed-point format. This denormalization takes time and also increases the message length. (This is not the case for the **CM:send–with–f–{min,max}–1L** instructions, which are implemented in a manner that avoids denormalization.)

As another example, router time generally decreases with increased enroute combining. Instructions whose names contain the term **–with** perform combining. While executing a combining instruction, the router attempts to combine any two messages headed for the same destination processor. Enroute combining reduces congestion and speeds up router execution because, as messages are combined, their number is reduced.

An exceptional case is the instruction **CM:send–with–logxor**, for which there is no hardware support. In contrast to other combining instructions, enroute combining is not done for a **CM:send–with–logxor** operation. Therefore, the time required to accomplish a **CM:send–with–logxor** operation is bounded below by a constant times the maximum number of items sent to any one destination.

Get Speed

A Paris **get** operation is accomplished by a process known as *backwards routing*. First, a **send** is done by the processors that are requesting data, and routing state information is saved. Then, the **send** is reversed, using the saved routing state information. Although this second phase is slightly faster than the first, one may assume that, for any given communication pattern, a **get** takes twice as long as a **send**.

For a highly congested communication pattern at a high VP ratio, a **get** operation could use a substantial amount of CM temporary memory if it attempted to move all the data at once. To avoid this problem, a "divided get" is automatically used; when insufficient memory is detected, the input data is divided into chunks that are moved separately.

A.2 Arithmetic Timing Tables

The following pages contain timing tables for the Paris Version 6.1 arithmetic instructions running on CM-2 systems with both 64- and 32-bit floating-point accelerators (FPAs). Reported times include only Connection Machine execution time. That is, they do not include front-end execution time. Each instruction was tested at a variety of VP ratios within a 1-dimensional geometry. Reported times are in units of microseconds.

Table 1 reports times with 64-bit FPAs. Table 2 reports times with 32-bit FPAs. Each table has five columns. In the first column, labeled "**Size**", the values 32 and 64 distinguish two rows of times for each instruction: one using 32-bit operands and one using 64-bit operands. The second column, labeled "**Name**", contains the name of the timed instruction. The third and forth columns, labeled "**VPR 1 and Sdev**" and "**VPR 16 and Sdev**", give timings at VP ratios of 1 and 16, along with the standard deviation in each case. The last column, labeled "**Ave(1,4,16,128) and Sdev**", gives the average (mean) time at VP ratios of 1, 4, 16, 32, and 128 and the standard deviation for the average. Timings taken using different input data specifications will vary.

These timings were done in two batches, running on two separate hardware configurations: The 32-bit FPA batch was run from a Sun 4/370 front end connected to a CM-2 with 8K processors, 32-bit floating-point accelerator chips, and 256K bits of memory per processor. The 64-bit FPA batch was run from a Sun-4/330 front end connected to a CM-2 with 512 processors, 64-bit floating-point accelerator chips and 256K bits of memory per processor. Timings taken using different hardware configurations may vary.

The timing numbers reported here were empirically derived; they are reliable within a 10% margin of accuracy. Use these numbers to compare the relative performance of different Paris instructions.

A speedup of approximately 40% over these CM-2 timings can be expected on a CM-200 model Connection Machine system.

Table 1. Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_c_acos_1_1L	2564.85	0.3717	32040.86	0.6195	2128.40	0.6228
64	CM_c_acos_1_1L	4603.19	0.5384	62056.99	0.87185	4090.97	0.7233
32	CM_c_acosh_1_1L	3159.46	0.5198	39949.67	0.6789	2642.44	0.6478
64	CM_c_acosh_1_1L	5702.83	0.5979	77524.20	0.8493	5097.81	0.6911
32	CM_c_asin_1_1L	2558.84	0.3313	32010.37	0.8119	2125.44	0.6663
64	CM_c_asin_1_1L	4596.81	0.6911	62022.80	0.5782	4087.43	0.6145
32	CM_c_asinh_1_1L	2474.00	0.4279	30862.74	0.8886	2050.81	0.7070
64	CM_c_asinh_1_1L	4450.37	0.5950	59886.13	0.6406	3950.24	0.6603
32	CM_c_atan_1_1L	3038.54	0.3978	38394.76	0.7985	2540.67	0.7118
64	CM_c_atan_1_1L	5497.38	0.5318	74555.03	0.7006	4905.50	0.6782
32	CM_c_atanh_1_1L	2162.92	0.5345	27695.48	0.6412	1826.23	0.6321
64	CM_c_atanh_1_1L	3990.53	0.4857	54350.70	0.7994	3571.54	0.6716
32	CM_c_c_signum_1_1L	495.92	0.2367	6291.49	0.6678	415.76	0.4624
64	CM_c_c_signum_1_1L	870.48	0.4200	11945.37	0.3916	782.65	0.4112
32	CM_c_conjugate_1_1L	8.97	0.1779	44.92	0.1118	4.20	0.1408
64	CM_c_conjugate_1_1L	8.98	0.1690	44.94	0.1116	4.63	0.1464
32	CM_c_cos_1_1L	2740.21	0.2887	34372.98	0.6820	2277.07	0.5867
64	CM_c_cos_1_1L	4390.34	0.3531	58887.52	0.5389	3885.79	0.5337
32	CM_c_cosh_1_1L	1688.55	0.3227	23653.83	0.4725	1524.38	0.5308
64	CM_c_cosh_1_1L	2905.94	0.3607	41472.15	0.7512	2682.96	0.4920
32	CM_c_exp_1_1L	1165.70	0.2787	16824.31	0.3531	1074.08	0.3543
64	CM_c_exp_1_1L	1956.72	0.3402	28364.09	0.4669	1825.43	0.4078
32	CM_c_ln_1_1L	1443.49	0.3190	18610.93	0.5336	1221.09	0.5326
64	CM_c_ln_1_1L	2663.95	0.4391	36733.39	0.6368	2404.42	0.5517
32	CM_c_negate_1_1L	14.96	0.1604	87.15	0.1325	7.55	0.1363
64	CM_c_negate_1_1L	14.95	0.1536	87.14	0.1268	8.20	0.1320
32	CM_c_reciprocal_1_1L	279.04	0.2065	3557.75	0.4113	234.83	0.3528
64	CM_c_reciprocal_1_1L	510.47	0.2680	6989.27	0.3491	458.33	0.3301
32	CM_c_sin_1_1L	2732.98	0.3266	34328.07	0.6423	2273.25	0.5978
64	CM_c_sin_1_1L	4383.12	0.4739	58842.97	0.5455	3881.70	0.5184
32	CM_c_sinh_1_1L	1688.48	0.3075	23653.81	0.4968	1524.35	0.5220
64	CM_c_sinh_1_1L	2905.97	0.3491	41472.27	0.6839	2682.98	0.4720
32	CM_c_sqrt_1_1L	704.52	0.3562	8439.29	0.4348	566.47	0.4955
64	CM_c_sqrt_1_1L	1186.49	0.4350	15592.12	0.6344	1036.27	0.5660

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_c_tan_1_1L	5940.31	0.3267	74754.94	0.7834	4951.03	0.6323
64	CM_c_tan_1_1L	9648.49	0.3929	129822.66	0.8484	8563.38	0.5818
32	CM_c_tanh_1_1L	1837.30	0.4192	25383.55	0.5013	1641.52	0.5876
64	CM_c_tanh_1_1L	3177.45	0.3078	45083.26	0.6752	2923.00	0.5951
32	CM_f_abs_1_1L	9.62	0.1594	28.29	0.1213	3.52	0.1422
64	CM_f_abs_1_1L	9.64	0.1752	28.29	0.1148	4.07	0.1474
32	CM_f_acos_1_1L	1052.19	0.5089	12818.90	0.5098	855.09	0.5617
64	CM_f_acos_1_1L	1927.83	0.3882	25898.65	0.5383	1707.31	0.4642
32	CM_f_acosh_1_1L	879.51	0.4148	11547.98	0.3237	752.25	0.4729
64	CM_f_acosh_1_1L	1349.08	0.3429	18382.36	0.7514	1206.83	0.4741
32	CM_f_asin_1_1L	1076.10	0.5695	12942.20	0.5818	867.29	0.6603
64	CM_f_asin_1_1L	1953.38	0.5010	25962.33	0.4842	1718.39	0.5469
32	CM_f_asinh_1_1L	833.03	0.4078	11183.33	0.4670	724.22	0.5034
64	CM_f_asinh_1_1L	1194.53	0.3354	16746.81	0.3436	1089.33	0.4282
32	CM_f_atan_1_1L	721.50	0.4360	8701.37	0.5089	582.88	0.5294
64	CM_f_atan_1_1L	1535.84	0.5759	20620.25	0.4682	1361.69	0.4762
32	CM_f_atanh_1_1L	1066.64	0.4854	12817.88	0.4622	856.42	0.5822
64	CM_f_atanh_1_1L	1499.70	0.3488	19271.39	0.3112	1290.93	0.3812
32	CM_f_cos_1_1L	343.09	0.1289	5205.17	0.1556	328.22	0.1744
64	CM_f_cos_1_1L	522.41	0.1896	7863.67	0.2028	498.08	0.2124
32	CM_f_cosh_1_1L	929.92	0.6060	10795.50	0.4448	730.85	0.5520
64	CM_f_cosh_1_1L	1486.52	0.4388	19062.70	0.3732	1277.64	0.4379
32	CM_f_exp_1_1L	380.45	0.1697	5348.68	0.2159	344.48	0.2402
64	CM_f_exp_1_1L	671.62	0.1859	9727.23	0.1863	624.05	0.2837
32	CM_f_exp2_1_1L	379.13	0.1655	5347.34	0.2008	344.12	0.2395
64	CM_f_exp2_1_1L	669.15	0.2348	9724.80	0.2078	623.23	0.2693
32	CM_f_f_ceiling_1_1L	502.05	0.3397	6482.90	0.4277	418.14	0.5047
64	CM_f_f_ceiling_1_1L	731.57	0.5352	11613.46	0.5051	722.36	0.5481
32	CM_f_f_floor_1_1L	458.43	0.4416	6095.65	0.4841	397.09	0.5219
64	CM_f_f_floor_1_1L	734.48	0.3017	11059.67	0.6440	703.20	0.5224
32	CM_f_f_round_1_1L	586.00	0.2655	9229.32	0.4294	576.13	0.4006
64	CM_f_f_round_1_1L	1209.63	0.2852	19490.30	0.4150	1208.76	0.3315
32	CM_f_f_signum_1_1L	27.90	0.2012	335.12	0.1788	22.47	0.1868
64	CM_f_f_signum_1_1L	41.32	0.1741	562.42	0.1958	36.95	0.1715
32	CM_f_ln_1_1L	346.91	0.2110	5193.87	0.2554	328.93	0.3109
64	CM_f_ln_1_1L	511.58	0.1561	7673.71	0.3478	486.40	0.3131

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_f_log10_1_1L	346.90	0.1881	5193.88	0.2563	328.93	0.3038
64	CM_f_log10_1_1L	512.20	0.1850	7673.71	0.3332	486.56	0.3131
32	CM_f_log2_1_1L	337.35	0.2217	5059.07	0.2246	320.26	0.2489
64	CM_f_log2_1_1L	501.12	0.1473	7538.18	0.3560	477.35	0.2394
32	CM_f_negate_1_1L	9.62	0.1657	28.31	0.1330	3.53	0.1600
64	CM_f_negate_1_1L	9.63	0.1706	28.30	0.1345	4.07	0.1565
32	CM_f_sin_1_1L	333.85	0.1378	5128.32	0.1702	321.81	0.2059
64	CM_f_sin_1_1L	578.97	0.2892	8612.53	0.2301	546.27	0.2284
32	CM_f_sinh_1_1L	1004.79	0.5122	11475.27	0.7198	780.49	0.6376
64	CM_f_sinh_1_1L	1574.01	0.3868	20023.71	0.4142	1345.66	0.4523
32	CM_f_sqrt_1_1L	119.00	0.1248	1710.64	0.1817	109.58	0.2086
64	CM_f_sqrt_1_1L	225.04	0.1308	3325.69	0.3184	212.84	0.2465
32	CM_f_tan_1_1L	358.32	0.1335	5520.02	0.2108	346.29	0.1957
64	CM_f_tan_1_1L	1319.66	0.2914	19017.70	0.4624	1226.32	0.3670
32	CM_f_tanh_1_1L	789.57	0.4278	10143.15	0.3980	668.13	0.5063
64	CM_f_tanh_1_1L	1358.05	0.4610	18580.40	0.3386	1217.50	0.3752
32	CM_lognot_1_1L	20.80	0.1334	241.22	0.1206	16.33	0.1330
64	CM_lognot_1_1L	33.16	0.1921	449.02	0.1230	29.55	0.1464
32	CM_lognot_always_1_1L	20.80	0.1850	241.23	0.1201	16.33	0.1464
64	CM_lognot_always_1_1L	33.11	0.1252	449.03	0.1292	29.54	0.1277
32	CM_s_abs_1_1L	49.98	0.1676	709.92	0.2254	45.62	0.2106
64	CM_s_abs_1_1L	85.60	0.1581	1282.54	0.8852	81.76	0.5679
32	CM_s_isqrt_1_1L	673.07	0.3497	10668.83	0.3302	668.19	0.3634
64	CM_s_isqrt_1_1L	2200.40	0.5064	35103.24	0.8657	2195.83	0.7074
32	CM_s_negate_1_1L	27.53	0.1991	344.71	0.1857	22.87	0.1836
64	CM_s_negate_1_1L	44.96	0.1790	629.85	0.1399	40.99	0.1597
32	CM_u_isqrt_1_1L	690.01	0.2947	10946.78	0.3333	685.46	0.3711
64	CM_u_isqrt_1_1L	2225.97	0.4459	35526.72	0.7926	2222.05	0.6441
32	CM_u_negate_1_1L	27.52	0.2149	346.40	0.1731	22.93	0.1858
64	CM_u_negate_1_1L	44.97	0.1542	630.92	0.1788	41.05	0.1744
32	CM_c_acos_2_1L	2564.88	0.4433	32041.20	0.6994	2126.38	0.6693
64	CM_c_acos_2_1L	4633.98	0.7176	62057.88	0.9735	4098.73	0.8499
32	CM_c_acosh_2_1L	3160.05	0.4802	39950.06	0.7257	2646.10	0.8035
64	CM_c_acosh_2_1L	5776.58	0.5842	77524.45	1.0702	5116.25	0.8045
32	CM_c_add_2_1L	85.96	0.3197	1018.75	0.2950	68.56	0.3532
64	CM_c_add_2_1L	159.28	0.2366	2039.96	0.3235	136.77	0.3308

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_c_add_always_2_1L	83.25	0.2354	814.72	0.2722	58.02	0.2391
64	CM_c_add_always_2_1L	134.68	0.1982	1808.03	0.2357	119.34	0.2606
32	CM_c_asin_2_1L	2564.64	0.4225	32270.08	0.8260	2135.26	0.6501
64	CM_c_asin_2_1L	4637.05	0.7600	62540.70	0.7651	4121.83	0.7355
32	CM_c_asinh_2_1L	2474.13	0.4608	30862.65	0.7916	2048.81	0.7207
64	CM_c_asinh_2_1L	4481.22	0.5876	59886.89	0.6457	3958.01	0.6586
32	CM_c_atan_2_1L	3038.56	0.3579	38394.83	0.8636	2543.20	0.6865
64	CM_c_atan_2_1L	5550.83	0.5122	74555.53	0.7622	4918.90	0.6505
32	CM_c_atanh_2_1L	2163.30	0.4864	27695.51	0.6586	1828.53	0.6907
64	CM_c_atanh_2_1L	4042.13	0.5133	54350.75	0.7285	3584.44	0.6617
32	CM_c_c_signum_2_1L	539.06	0.2898	7269.60	0.4495	469.02	0.4011
64	CM_c_c_signum_2_1L	949.20	0.3785	13859.50	0.2734	892.25	0.3241
32	CM_c_conjugate_2_1L	50.46	0.1340	835.83	0.1916	49.80	0.1854
64	CM_c_conjugate_2_1L	86.12	0.1778	1588.94	0.1599	96.40	0.1703
32	CM_c_cos_2_1L	2711.01	0.2411	33954.30	0.5728	2251.96	0.5538
64	CM_c_cos_2_1L	4343.62	0.4672	58090.16	0.5319	3841.44	0.5552
32	CM_c_cosh_2_1L	1688.48	0.3125	23653.37	0.4265	1523.17	0.4587
64	CM_c_cosh_2_1L	2909.69	0.3305	41471.65	0.6416	2683.89	0.4606
32	CM_c_divide_2_1L	498.85	0.2273	6473.43	0.4217	425.67	0.4564
64	CM_c_divide_2_1L	919.35	0.4255	12888.47	0.3520	839.72	0.3267
32	CM_c_divide_always_2_1L	562.74	0.2599	6424.46	0.3117	437.51	0.3344
64	CM_c_divide_always_2_1L	947.07	0.1481	13721.57	0.2257	884.71	0.2917
32	CM_c_divinto_2_1L	498.83	0.2367	6473.40	0.4257	425.62	0.4631
64	CM_c_divinto_2_1L	919.35	0.4684	12888.47	0.3561	839.72	0.3355
32	CM_c_divinto_always_2_1L	562.78	0.2756	6424.41	0.3516	437.52	0.3398
64	CM_c_divinto_always_2_1L	947.07	0.1438	13721.56	0.2188	884.73	0.2713
32	CM_c_exp_2_1L	1165.53	0.2245	16898.42	0.2087	1079.94	0.3417
64	CM_c_exp_2_1L	1956.60	0.3561	28512.62	0.3488	1832.32	0.3718
32	CM_c_f_cis_2_1L	673.51	0.1796	10404.18	0.1756	654.91	0.2158
64	CM_c_f_cis_2_1L	1102.60	0.2082	16625.94	0.2995	1059.27	0.2655
32	CM_c_ln_2_1L	1443.53	0.2689	18611.21	0.4999	1226.69	0.5290
64	CM_c_ln_2_1L	2671.80	0.4340	36734.44	0.6320	2406.48	0.5323
32	CM_c_multiply_2_1L	234.77	0.2053	3109.25	0.3239	203.53	0.3443
64	CM_c_multiply_2_1L	434.31	0.2277	6245.91	0.2732	404.16	0.3128
32	CM_c_multiply_always_2_1L	263.68	0.2339	2749.54	0.2561	192.19	0.3340
64	CM_c_multiply_always_2_1L	430.83	0.1550	6042.11	0.3575	393.66	0.2637

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_c_negate_2_1L	51.09	0.1482	834.54	0.2012	49.87	0.1812
64	CM_c_negate_2_1L	86.76	0.1335	1587.44	0.1489	96.53	0.2048
32	CM_c_reciprocal_2_1L	280.21	0.2177	3559.75	0.4085	235.25	0.3532
64	CM_c_reciprocal_2_1L	511.70	0.2871	6994.39	0.3594	458.93	0.3474
32	CM_c_sin_2_1L	2703.74	0.3146	33909.42	0.5413	2248.20	0.5459
64	CM_c_sin_2_1L	4336.12	0.4203	58045.99	0.4337	3837.28	0.5601
32	CM_c_sinh_2_1L	1688.50	0.3107	23653.36	0.4348	1523.17	0.4587
64	CM_c_sinh_2_1L	2909.72	0.3286	41471.65	0.6758	2683.90	0.4681
32	CM_c_sqrt_2_1L	704.81	0.3151	8442.04	0.5251	566.78	0.5033
64	CM_c_sqrt_2_1L	1187.20	0.4460	15596.41	0.4560	1036.64	0.4446
32	CM_c_subfrom_2_1L	99.35	0.2758	1100.13	0.2119	75.45	0.2347
64	CM_c_subfrom_2_1L	171.73	0.2308	2121.37	0.2833	144.00	0.3169
32	CM_c_subfrom_always_2_1L	94.02	0.2846	864.03	0.1904	62.77	0.2261
64	CM_c_subfrom_always_2_1L	144.32	0.2354	1857.83	0.2106	124.33	0.2352
32	CM_c_subtract_2_1L	85.90	0.1983	1018.80	0.2196	68.54	0.2534
64	CM_c_subtract_2_1L	159.25	0.2061	2039.94	0.2806	136.76	0.2968
32	CM_c_subtract_always_2_1L	83.21	0.2054	814.70	0.2398	58.01	0.2258
64	CM_c_subtract_always_2_1L	134.65	0.1995	1808.00	0.2455	119.33	0.2354
32	CM_c_tan_2_1L	5905.98	0.5810	74333.42	0.5884	4923.42	0.6593
64	CM_c_tan_2_1L	9651.77	0.4258	129018.93	0.6569	8530.99	0.6612
32	CM_c_tanh_2_1L	1837.25	0.3576	25383.60	0.4883	1639.67	0.5975
64	CM_c_tanh_2_1L	3181.97	0.2665	45082.88	0.5956	2924.13	0.5564
32	CM_f_abs_2_1L	28.00	0.1549	418.79	0.1800	25.54	0.1822
64	CM_f_abs_2_1L	45.00	0.1396	795.36	0.1920	45.94	0.1568
32	CM_f_acos_2_1L	1052.23	0.5060	12818.93	0.5779	854.97	0.5482
64	CM_f_acos_2_1L	1927.83	0.3727	25898.62	0.5379	1710.37	0.4716
32	CM_f_acosh_2_1L	879.64	0.3983	11548.21	0.3294	753.48	0.4714
64	CM_f_acosh_2_1L	1349.14	0.4053	18382.39	0.7202	1208.07	0.4944
32	CM_f_add_2_1L	44.68	0.2361	510.70	0.2055	34.72	0.2331
64	CM_f_add_2_1L	81.03	0.1910	1020.65	0.2967	68.76	0.2795
32	CM_f_add_always_2_1L	43.51	0.2371	409.04	0.2353	29.50	0.2260
64	CM_f_add_always_2_1L	68.86	0.2263	905.16	0.2283	60.10	0.2186
32	CM_f_asin_2_1L	1075.93	0.5248	13033.47	0.4328	870.44	0.5298
64	CM_f_asin_2_1L	1953.79	0.4948	26144.49	0.5400	1725.99	0.5398
32	CM_f_asinh_2_1L	832.81	0.4468	11274.84	0.3701	728.88	0.4667
64	CM_f_asinh_2_1L	1195.21	0.3131	16929.46	0.5372	1096.41	0.4758

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_f_atan_2_1L	720.65	0.4117	8793.28	0.3022	586.25	0.4387
64	CM_f_atan_2_1L	1539.11	0.5222	20803.58	0.5383	1366.94	0.4863
32	CM_f_atanh_2_1L	1066.60	0.5293	12909.31	0.3853	861.18	0.5177
64	CM_f_atanh_2_1L	1499.74	0.4106	19454.51	0.4385	1296.57	0.4193
32	CM_f_c_abs_2_1L	225.68	0.2022	2906.83	0.3057	191.35	0.2813
64	CM_f_c_abs_2_1L	413.60	0.2295	5714.29	0.4091	373.42	0.3441
32	CM_f_cos_2_1L	343.08	0.1705	5205.18	0.1309	328.22	0.1677
64	CM_f_cos_2_1L	522.37	0.1947	7863.65	0.1941	500.97	0.1953
32	CM_f_cosh_2_1L	930.14	0.5448	10795.51	0.4864	730.90	0.5582
64	CM_f_cosh_2_1L	1486.53	0.4620	19062.71	0.3681	1275.15	0.4173
32	CM_f_divide_2_1L	86.83	0.1674	1186.36	0.3062	76.94	0.3015
64	CM_f_divide_2_1L	161.54	0.1188	2305.03	0.2000	149.10	0.1836
32	CM_f_divide_always_2_1L	85.48	0.2376	1084.48	0.2735	71.67	0.2848
64	CM_f_divide_always_2_1L	149.23	0.1667	2189.40	0.2401	140.40	0.2268
32	CM_f_divinto_2_1L	89.78	0.2129	1187.29	0.2259	77.63	0.2443
64	CM_f_divinto_2_1L	162.46	0.1708	2303.95	0.2658	149.32	0.2385
32	CM_f_divinto_always_2_1L	86.45	0.1626	1082.61	0.2247	71.80	0.2783
64	CM_f_divinto_always_2_1L	150.22	0.1524	2188.80	0.2377	140.65	0.2433
32	CM_f_exp_2_1L	380.88	0.1683	5348.70	0.2167	344.57	0.2662
64	CM_f_exp_2_1L	671.50	0.1703	9727.25	0.1677	626.93	0.2236
32	CM_f_exp2_2_1L	379.55	0.2078	5347.37	0.2083	344.21	0.2606
64	CM_f_exp2_2_1L	669.09	0.1977	9724.77	0.1815	626.12	0.2155
32	CM_f_f_ceiling_2_1L	528.49	0.4148	6673.00	0.4547	434.55	0.4895
64	CM_f_f_ceiling_2_1L	776.21	0.2968	11432.54	0.5390	744.83	0.4977
32	CM_f_f_floor_2_1L	532.47	0.3411	6609.78	0.4708	434.43	0.5276
64	CM_f_f_floor_2_1L	778.36	0.2735	11364.15	0.8004	745.23	0.5635
32	CM_f_f_round_2_1L	613.04	0.2568	9650.50	0.4056	601.61	0.4378
64	CM_f_f_round_2_1L	1254.33	0.3280	20288.64	0.2748	1263.03	0.3453
32	CM_f_f_signum_2_1L	27.92	0.1666	342.08	0.1393	22.73	0.1510
64	CM_f_f_signum_2_1L	41.27	0.1249	569.40	0.1447	37.16	0.1275
32	CM_f_ln_2_1L	346.91	0.1889	5381.73	0.2354	335.97	0.2486
64	CM_f_ln_2_1L	511.60	0.1757	7673.70	0.3107	489.32	0.2936
32	CM_f_log10_2_1L	346.91	0.2052	5381.73	0.2535	335.97	0.2524
64	CM_f_log10_2_1L	512.16	0.1478	7673.71	0.3360	489.46	0.3013
32	CM_f_log2_2_1L	337.36	0.1872	5245.94	0.1991	327.26	0.2639
64	CM_f_log2_2_1L	501.10	0.1537	7538.08	0.4007	480.26	0.2485

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_f_max_2_1L	53.42	0.1644	655.24	0.3033	43.69	0.3363
64	CM_f_max_2_1L	94.53	0.1360	1231.22	0.2291	82.04	0.2059
32	CM_f_min_2_1L	53.41	0.1601	655.20	0.3102	43.68	0.3390
64	CM_f_min_2_1L	94.53	0.1556	1231.19	0.2085	82.04	0.1950
32	CM_f_mod_2_1L	664.07	0.4402	8883.59	0.3963	578.56	0.4982
64	CM_f_mod_2_1L	1184.85	0.3983	16408.05	0.5334	1072.13	0.5504
32	CM_f_multiply_2_1L	44.77	0.2698	510.65	0.2158	34.73	0.2380
64	CM_f_multiply_2_1L	81.05	0.2034	1020.65	0.3035	68.77	0.2704
32	CM_f_multiply_always_2_1L	43.52	0.2456	409.06	0.2141	29.50	0.2251
64	CM_f_multiply_always_2_1L	68.88	0.2127	905.19	0.2287	60.11	0.2163
32	CM_f_negate_2_1L	27.99	0.1504	418.81	0.1847	25.54	0.1736
64	CM_f_negate_2_1L	45.01	0.1425	795.36	0.1740	45.95	0.1546
32	CM_f_rem_2_1L	501.93	0.3609	6722.58	0.3959	437.58	0.5058
64	CM_f_rem_2_1L	945.04	0.3243	13270.54	0.4155	861.70	0.4230
32	CM_f_sin_2_1L	333.87	0.1307	5202.47	0.2193	324.59	0.1719
64	CM_f_sin_2_1L	579.49	0.2182	8760.80	0.1820	555.68	0.1924
32	CM_f_sinh_2_1L	1004.82	0.5230	11568.61	0.4738	783.99	0.4870
64	CM_f_sinh_2_1L	1574.00	0.4477	20207.03	0.3562	1349.00	0.4451
32	CM_f_sqrt_2_1L	119.83	0.2000	1709.71	0.2181	109.72	0.2562
64	CM_f_sqrt_2_1L	225.74	0.1725	3326.42	0.3702	213.06	0.2822
32	CM_f_subfrom_2_1L	46.49	0.1981	509.77	0.1987	35.06	0.2151
64	CM_f_subfrom_2_1L	82.08	0.2019	1019.92	0.3296	69.03	0.2671
32	CM_f_subfrom_always_2_1L	45.09	0.2466	407.09	0.1871	29.75	0.2023
64	CM_f_subfrom_always_2_1L	69.65	0.1985	904.25	0.1832	60.29	0.1906
32	CM_f_subtract_2_1L	44.64	0.2319	510.69	0.2523	34.71	0.2357
64	CM_f_subtract_2_1L	81.05	0.2077	1020.64	0.2975	68.76	0.2670
32	CM_f_subtract_always_2_1L	43.50	0.2458	409.02	0.2406	29.50	0.2271
64	CM_f_subtract_always_2_1L	68.85	0.1931	905.20	0.2220	60.10	0.2070
32	CM_f_tan_2_1L	358.30	0.1702	5594.05	0.1600	349.06	0.1571
64	CM_f_tan_2_1L	1320.19	0.3426	19200.80	0.3197	1233.65	0.3211
32	CM_f_tanh_2_1L	789.52	0.4889	10234.03	0.3222	671.70	0.3638
64	CM_f_tanh_2_1L	1358.45	0.4556	18763.40	0.5415	1224.84	0.4133
32	CM_logand_2_1L	25.81	0.1692	412.49	0.1386	24.76	0.1445
64	CM_logand_2_1L	43.29	0.1295	788.96	0.1401	45.20	0.1563
32	CM_logand_always_2_1L	24.62	0.1377	396.36	0.1432	23.73	0.1440
64	CM_logand_always_2_1L	42.16	0.1733	773.05	0.1586	44.17	0.1504

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_logand_const_always_2_1L	46.64	0.1829	417.77	0.1690	29.63	0.2055
64	CM_logand_const_always_2_1L	81.78	0.2279	812.87	0.2018	60.33	0.2921
32	CM_logand_constant_2_1L	47.75	0.2142	433.83	0.2757	30.65	0.2593
64	CM_logand_constant_2_1L	82.94	0.1640	828.97	0.4176	61.36	0.2776
32	CM_logandc1_2_1L	25.76	0.1335	412.46	0.1260	24.75	0.1323
64	CM_logandc1_2_1L	43.28	0.1344	788.96	0.1240	45.19	0.1513
32	CM_logandc1_always_2_1L	24.63	0.1555	396.34	0.1151	23.73	0.1441
64	CM_logandc1_always_2_1L	42.10	0.1235	773.08	0.1847	44.15	0.1460
32	CM_logandc1_const_always_2_1L	46.68	0.2191	417.84	0.2682	29.64	0.2660
64	CM_logandc1_const_always_2_1L	83.55	0.4402	812.89	0.2643	60.77	0.3305
32	CM_logandc1_constant_2_1L	47.79	0.1726	433.75	0.1635	30.65	0.2098
64	CM_logandc1_constant_2_1L	82.90	0.2945	828.76	0.2286	61.35	0.3150
32	CM_logandc2_2_1L	25.86	0.2049	412.50	0.1364	24.77	0.1476
64	CM_logandc2_2_1L	43.29	0.1230	788.97	0.1459	45.20	0.1565
32	CM_logandc2_always_2_1L	24.63	0.1344	396.35	0.1008	23.73	0.1304
64	CM_logandc2_always_2_1L	42.10	0.1140	773.09	0.1728	44.15	0.1436
32	CM_logandc2_const_always_2_1L	46.64	0.1892	417.78	0.1751	29.63	0.2308
64	CM_logandc2_const_always_2_1L	81.76	0.1695	812.88	0.2071	60.32	0.2055
32	CM_logandc2_constant_2_1L	47.78	0.2064	433.74	0.1559	30.65	0.1703
64	CM_logandc2_constant_2_1L	82.97	0.2050	828.77	0.2515	61.37	0.2248
32	CM_logeqv_2_1L	25.77	0.1400	412.48	0.1289	24.75	0.1261
64	CM_logeqv_2_1L	43.28	0.1090	788.95	0.1351	45.19	0.1465
32	CM_logeqv_always_2_1L	24.63	0.1184	396.34	0.1050	23.73	0.1317
64	CM_logeqv_always_2_1L	42.12	0.1223	773.10	0.1891	44.16	0.1541
32	CM_logeqv_const_always_2_1L	46.67	0.2109	417.92	0.2833	29.64	0.2711
64	CM_logeqv_const_always_2_1L	81.76	0.1904	812.97	0.3451	60.32	0.2444
32	CM_logeqv_constant_2_1L	47.77	0.1930	433.73	0.1460	30.65	0.1642
64	CM_logeqv_constant_2_1L	82.93	0.1565	828.75	0.1958	61.35	0.1915
32	CM_logior_2_1L	25.78	0.1284	412.48	0.1172	24.75	0.1214
64	CM_logior_2_1L	43.26	0.1168	788.96	0.1451	45.19	0.1474
32	CM_logior_always_2_1L	24.62	0.1180	396.35	0.1095	23.73	0.1282
64	CM_logior_always_2_1L	42.11	0.1118	773.10	0.1728	44.16	0.1403
32	CM_logior_const_always_2_1L	46.64	0.1975	417.80	0.1963	29.63	0.2052
64	CM_logior_const_always_2_1L	81.77	0.1879	812.85	0.2061	60.33	0.2782
32	CM_logior_constant_2_1L	47.78	0.2076	433.75	0.1815	30.65	0.1902
64	CM_logior_constant_2_1L	82.95	0.1786	828.75	0.2331	61.36	0.2062

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_lognand_2_1L	25.79	0.1432	412.50	0.1363	24.76	0.1486
64	CM_lognand_2_1L	43.29	0.1219	789.01	0.1635	45.20	0.1628
32	CM_lognand_always_2_1L	24.62	0.1214	396.35	0.1071	23.73	0.1269
64	CM_lognand_always_2_1L	42.12	0.1129	773.10	0.1777	44.16	0.1437
32	CM_lognand_const_always_2_1L	46.64	0.1988	417.76	0.1604	29.63	0.1860
64	CM_lognand_const_always_2_1L	81.76	0.1630	812.89	0.2600	60.32	0.2812
32	CM_lognand_constant_2_1L	47.84	0.2415	433.92	0.3065	30.67	0.3011
64	CM_lognand_constant_2_1L	82.97	0.2653	828.75	0.2229	61.37	0.3025
32	CM_lognor_2_1L	25.78	0.1437	412.48	0.1287	24.75	0.1331
64	CM_lognor_2_1L	43.28	0.1144	788.94	0.1350	45.19	0.1472
32	CM_lognor_always_2_1L	24.62	0.1179	396.35	0.1025	23.73	0.1228
64	CM_lognor_always_2_1L	42.11	0.1127	773.09	0.1753	44.16	0.1426
32	CM_lognor_const_always_2_1L	46.63	0.1877	417.75	0.1480	29.63	0.1713
64	CM_lognor_const_always_2_1L	81.79	0.1796	812.87	0.2365	60.32	0.1941
32	CM_lognor_constant_2_1L	47.77	0.1847	433.72	0.1301	30.65	0.1486
64	CM_lognor_constant_2_1L	82.94	0.1656	828.74	0.1973	61.36	0.2193
32	CM_lognot_2_1L	25.78	0.1359	412.48	0.1215	24.75	0.1197
64	CM_lognot_2_1L	43.28	0.1090	788.96	0.1416	45.19	0.1502
32	CM_lognot_always_2_1L	24.61	0.1300	396.34	0.1190	23.72	0.1398
64	CM_lognot_always_2_1L	42.11	0.1052	773.03	0.1523	44.15	0.1321
32	CM_logorc1_2_1L	25.78	0.1326	412.47	0.1142	24.75	0.1199
64	CM_logorc1_2_1L	43.29	0.1137	788.95	0.1347	45.19	0.1469
32	CM_logorc1_always_2_1L	24.62	0.1219	396.35	0.1021	23.73	0.1256
64	CM_logorc1_always_2_1L	42.11	0.1130	773.08	0.1723	44.16	0.1429
32	CM_logorc1_const_always_2_1L	46.62	0.1843	417.75	0.1532	29.62	0.1748
64	CM_logorc1_const_always_2_1L	81.76	0.1775	812.88	0.2155	60.32	0.2127
32	CM_logorc1_constant_2_1L	47.77	0.1883	433.74	0.1537	30.65	0.1648
64	CM_logorc1_constant_2_1L	82.96	0.1912	828.79	0.2540	61.36	0.2214
32	CM_logorc2_2_1L	25.77	0.1262	412.48	0.1190	24.75	0.1162
64	CM_logorc2_2_1L	43.28	0.1106	788.95	0.1333	45.19	0.1444
32	CM_logorc2_always_2_1L	24.62	0.1200	396.35	0.1104	23.73	0.1298
64	CM_logorc2_always_2_1L	42.11	0.1095	773.09	0.1846	44.16	0.1460
32	CM_logorc2_const_always_2_1L	46.63	0.1932	417.78	0.1720	29.63	0.1818
64	CM_logorc2_const_always_2_1L	81.75	0.1576	812.86	0.2189	60.31	0.1984
32	CM_logorc2_constant_2_1L	47.75	0.1797	433.73	0.1398	30.65	0.1606
64	CM_logorc2_constant_2_1L	82.96	0.1981	828.78	0.2417	61.36	0.2179

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_logxor_2_1L	25.78	0.1328	412.47	0.1262	24.75	0.1252
64	CM_logxor_2_1L	43.28	0.1074	788.95	0.1371	45.19	0.1425
32	CM_logxor_always_2_1L	24.63	0.1173	396.35	0.1040	23.73	0.1256
64	CM_logxor_always_2_1L	42.12	0.1186	773.08	0.1653	44.16	0.1447
32	CM_logxor_const_always_2_1L	46.62	0.1862	417.76	0.1450	29.62	0.1800
64	CM_logxor_const_always_2_1L	81.76	0.1633	812.88	0.2242	60.32	0.2043
32	CM_logxor_constant_2_1L	47.78	0.1831	433.72	0.1233	30.65	0.1529
64	CM_logxor_constant_2_1L	82.94	0.1727	828.82	0.3581	61.36	0.2818
32	CM_s_abs_2_1L	49.98	0.1399	901.97	0.1458	52.82	0.1652
64	CM_s_abs_2_1L	85.58	0.1332	1654.57	0.1227	93.35	0.1717
32	CM_s_add_2_1L	28.63	0.1425	460.50	0.1297	27.65	0.1544
64	CM_s_add_2_1L	46.13	0.1274	837.09	0.2002	48.03	0.1612
32	CM_s_add_carry_2_1L	29.17	0.1641	469.65	0.1374	28.21	0.1644
64	CM_s_add_carry_2_1L	46.66	0.1282	845.76	0.2015	48.57	0.1618
32	CM_s_add_constant_2_1L	34.93	0.1702	452.88	0.1873	29.77	0.1931
64	CM_s_add_constant_2_1L	64.72	0.1545	854.74	0.2182	56.72	0.1953
32	CM_s_add_flags_2_1L	28.63	0.1473	460.50	0.1224	27.65	0.1515
64	CM_s_add_flags_2_1L	46.13	0.1188	837.09	0.2096	48.03	0.1638
32	CM_s_ceiling_2_1L	1191.24	0.3669	19299.86	0.3261	1199.62	0.5356
64	CM_s_ceiling_2_1L	3880.14	0.6104	62596.87	9.2663	3904.38	7.8702
32	CM_s_ceiling_constant_2_1L	1196.50	0.3469	19107.98	0.8953	1192.42	0.8169
64	CM_s_ceiling_constant_2_1L	3867.51	0.6244	62033.16	0.8206	3875.28	0.7460
32	CM_s_floor_2_1L	1196.02	0.3847	19365.33	0.7001	1204.00	0.8585
64	CM_s_floor_2_1L	3889.80	0.5817	62744.54	0.5304	3913.77	0.5502
32	CM_s_floor_constant_2_1L	1201.33	0.3536	19185.26	0.6973	1197.25	0.7556
64	CM_s_floor_constant_2_1L	3877.39	0.6532	62177.77	0.7823	3884.57	0.7543
32	CM_s_isqrt_2_1L	673.12	0.3569	10668.83	0.3441	668.21	0.3778
32	CM_c_add_3_1L	88.43	0.2521	1020.90	0.2361	69.22	0.2570
64	CM_c_add_3_1L	160.70	0.1965	2045.01	0.3083	137.43	0.2777
32	CM_c_add_always_3_1L	85.59	0.2490	811.56	0.1944	58.44	0.2439
64	CM_c_add_always_3_1L	135.91	0.1685	1809.46	0.2364	119.77	0.1982
32	CM_c_divide_3_1L	467.09	0.2935	5848.90	0.5270	387.99	0.4955
64	CM_c_divide_3_1L	862.19	0.3306	11662.54	0.4435	767.84	0.4195
32	CM_c_divide_always_3_1L	531.73	0.2111	5800.39	0.3441	400.05	0.3579
64	CM_c_divide_always_3_1L	890.96	0.1711	12495.17	0.3062	813.11	0.3172

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_c_multiply_3_1L	193.67	0.2191	2315.61	0.3578	155.54	0.3205
64	CM_c_multiply_3_1L	357.33	0.2500	4699.07	0.4317	312.28	0.3668
32	CM_c_multiply_always_3_1L	259.31	0.2597	2444.27	0.3309	176.42	0.3451
64	CM_c_multiply_always_3_1L	413.57	0.2317	5434.06	0.4686	361.25	0.3281
32	CM_c_subtract_3_1L	88.41	0.2799	1020.95	0.2085	69.21	0.2710
64	CM_c_subtract_3_1L	160.69	0.1867	2045.00	0.2835	137.43	0.2828
32	CM_c_subtract_always_3_1L	85.71	0.2471	811.60	0.1872	58.47	0.2229
64	CM_c_subtract_always_3_1L	135.93	0.1536	1809.45	0.2081	119.78	0.1860
32	CM_f_add_3_1L	46.30	0.3419	512.19	0.1876	35.11	0.2764
64	CM_f_add_3_1L	82.02	0.1533	1024.02	0.2110	69.21	0.1882
32	CM_f_add_always_3_1L	45.07	0.2150	407.54	0.2072	29.76	0.2103
64	CM_f_add_always_3_1L	69.67	0.1734	906.44	0.1760	60.42	0.1772
32	CM_f_atan2_3_1L	840.79	0.3896	10121.97	0.3970	677.02	0.5295
64	CM_f_atan2_3_1L	1714.09	0.4401	23132.67	0.3468	1522.20	0.4280
32	CM_f_divide_3_1L	89.73	0.2520	1189.41	0.3381	77.71	0.3137
64	CM_f_divide_3_1L	162.47	0.1893	2308.64	0.2767	149.54	0.2371
32	CM_f_divide_always_3_1L	86.94	0.1617	1083.54	0.2510	71.95	0.2017
64	CM_f_divide_always_3_1L	150.20	0.1490	2190.77	0.1563	140.77	0.1677
32	CM_f_max_3_1L	55.04	0.2675	654.83	0.4902	44.01	0.4198
64	CM_f_max_3_1L	95.87	0.1419	1243.87	0.2809	82.98	0.2043
32	CM_f_min_3_1L	54.56	0.1917	654.30	0.4671	43.91	0.4071
64	CM_f_min_3_1L	95.87	0.1388	1243.89	0.3286	82.98	0.2639
32	CM_f_mod_3_1L	664.78	0.3100	8605.87	0.5889	566.74	0.5129
64	CM_f_mod_3_1L	1087.99	0.3190	16216.53	0.5370	1028.28	0.4263
32	CM_f_multiply_3_1L	46.38	0.3655	512.20	0.2372	35.13	0.3864
64	CM_f_multiply_3_1L	82.02	0.1521	1024.00	0.1989	69.21	0.1953
32	CM_f_multiply_always_3_1L	45.06	0.2075	407.52	0.1786	29.76	0.2729
64	CM_f_multiply_always_3_1L	69.68	0.2026	906.44	0.1636	60.44	0.2406
32	CM_f_rem_3_1L	548.86	0.3099	6821.13	0.6631	453.27	0.5092
64	CM_f_rem_3_1L	848.10	0.2974	13175.83	0.5408	820.46	0.4707
32	CM_f_subtract_3_1L	46.30	0.3442	512.18	0.1920	35.11	0.2665
64	CM_f_subtract_3_1L	82.02	0.1567	1024.02	0.2131	69.21	0.1972
32	CM_f_subtract_always_3_1L	45.05	0.2315	407.53	0.1585	29.75	0.2151
64	CM_f_subtract_always_3_1L	69.66	0.1669	906.44	0.1639	60.42	0.1739
32	CM_logand_3_1L	32.19	0.2552	623.37	0.1481	35.01	0.1885
64	CM_logand_3_1L	53.99	0.1274	1207.36	0.3152	68.06	0.2482

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_logand_always_3_1L	26.25	0.1689	477.30	0.1248	27.15	0.1538
64	CM_logand_always_3_1L	43.27	0.1545	928.99	0.1361	54.04	0.1460
32	CM_logand_const_always_3_1L	47.66	0.1928	498.38	0.2491	32.92	0.2609
64	CM_logand_const_always_3_1L	84.09	0.2130	968.51	0.1562	67.65	0.1910
32	CM_logand_constant_3_1L	53.25	0.1795	644.54	0.2198	40.71	0.2058
64	CM_logand_constant_3_1L	93.60	0.1589	1246.95	0.2240	81.38	0.2333
32	CM_logandc1_3_1L	32.11	0.2652	623.37	0.1517	34.99	0.1895
64	CM_logandc1_3_1L	54.00	0.1304	1207.35	0.3151	68.07	0.2433
32	CM_logandc1_always_3_1L	26.25	0.1705	477.30	0.1204	27.15	0.1538
64	CM_logandc1_always_3_1L	43.26	0.1466	928.98	0.1388	54.04	0.1412
32	CM_logandc1_const_always_3_1L	47.64	0.1594	498.29	0.1530	32.91	0.1757
64	CM_logandc1_const_always_3_1L	84.07	0.2006	968.55	0.2044	67.64	0.2047
32	CM_logandc1_constant_3_1L	53.23	0.1785	644.53	0.2078	40.70	0.1881
64	CM_logandc1_constant_3_1L	93.62	0.1553	1246.95	0.2265	81.39	0.2376
32	CM_logandc2_3_1L	32.14	0.2689	623.38	0.1533	35.00	0.1921
64	CM_logandc2_3_1L	53.98	0.1382	1207.34	0.3056	68.06	0.2450
32	CM_logandc2_always_3_1L	26.24	0.1679	477.31	0.1350	27.15	0.1553
64	CM_logandc2_always_3_1L	43.26	0.1496	928.98	0.1318	54.04	0.1399
32	CM_logandc2_const_always_3_1L	47.65	0.1854	498.30	0.1499	32.91	0.1859
64	CM_logandc2_const_always_3_1L	84.10	0.2194	968.52	0.1612	67.65	0.2009
32	CM_logandc2_constant_3_1L	53.26	0.2498	644.55	0.2357	40.71	0.2221
64	CM_logandc2_constant_3_1L	93.65	0.1939	1246.98	0.2794	81.40	0.3183
32	CM_logeqv_3_1L	32.11	0.2453	623.37	0.1567	34.99	0.1931
64	CM_logeqv_3_1L	53.99	0.1347	1207.37	0.3111	68.06	0.2471
32	CM_logeqv_always_3_1L	26.27	0.1879	477.31	0.1282	27.16	0.1586
64	CM_logeqv_always_3_1L	43.28	0.1579	928.98	0.1312	54.04	0.1466
32	CM_logeqv_const_always_3_1L	47.66	0.1845	498.34	0.1968	32.92	0.2071
64	CM_logeqv_const_always_3_1L	84.13	0.2208	968.55	0.1877	67.65	0.2243
32	CM_logeqv_constant_3_1L	53.21	0.1762	644.53	0.2225	40.70	0.2326
64	CM_logeqv_constant_3_1L	93.85	0.5497	1246.94	0.2270	81.45	0.3323
32	CM_logior_3_1L	32.13	0.2548	623.39	0.1491	35.00	0.1900
64	CM_logior_3_1L	54.01	0.1635	1207.39	0.3115	68.07	0.2510
32	CM_logior_always_3_1L	26.24	0.1664	477.31	0.1383	27.15	0.1561
64	CM_logior_always_3_1L	43.28	0.1654	928.98	0.1261	54.05	0.1465
32	CM_logior_const_always_3_1L	47.65	0.1590	498.31	0.1591	32.91	0.2351
64	CM_logior_const_always_3_1L	84.12	0.2319	968.54	0.1915	67.66	0.2729

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_logior_constant_3_1L	53.23	0.2120	644.62	0.3393	40.71	0.2648
64	CM_logior_constant_3_1L	93.63	0.1579	1246.97	0.2074	81.40	0.2542
32	CM_lognand_3_1L	32.19	0.2646	623.38	0.1556	35.01	0.1902
64	CM_lognand_3_1L	53.98	0.1198	1207.38	0.3335	68.06	0.2449
32	CM_lognand_always_3_1L	26.24	0.1653	477.31	0.1332	27.15	0.1591
64	CM_lognand_always_3_1L	43.28	0.1580	928.99	0.1382	54.04	0.1411
32	CM_lognand_const_always_3_1L	47.66	0.1748	498.31	0.1867	32.92	0.1807
64	CM_lognand_const_always_3_1L	84.08	0.2015	968.55	0.1895	67.64	0.1946
32	CM_lognand_constant_3_1L	53.23	0.1690	644.55	0.2206	40.70	0.1982
64	CM_lognand_constant_3_1L	93.61	0.1317	1247.11	0.3901	81.39	0.2999
32	CM_lognor_3_1L	32.13	0.2598	623.38	0.1579	35.00	0.1888
64	CM_lognor_3_1L	53.99	0.1384	1207.35	0.3239	68.06	0.2501
32	CM_lognor_always_3_1L	26.26	0.1741	477.32	0.1319	27.16	0.1520
64	CM_lognor_always_3_1L	43.32	0.2127	928.99	0.1439	54.05	0.1618
32	CM_lognor_const_always_3_1L	47.66	0.1936	498.34	0.1876	32.92	0.2205
64	CM_lognor_const_always_3_1L	84.09	0.2097	968.55	0.1899	67.65	0.1930
32	CM_lognor_constant_3_1L	53.23	0.1737	644.56	0.2426	40.70	0.2030
64	CM_lognor_constant_3_1L	93.65	0.2101	1247.26	0.5439	81.40	0.3361
32	CM_logorc1_3_1L	32.14	0.2706	623.37	0.1531	35.00	0.1973
64	CM_logorc1_3_1L	53.99	0.1410	1207.38	0.3080	68.07	0.2461
32	CM_logorc1_always_3_1L	26.25	0.1656	477.31	0.1368	27.15	0.1574
64	CM_logorc1_always_3_1L	43.28	0.1565	928.98	0.1321	54.05	0.1435
32	CM_logorc1_const_always_3_1L	47.63	0.1643	498.30	0.1385	32.91	0.2332
64	CM_logorc1_const_always_3_1L	84.11	0.2544	968.72	0.4763	67.65	0.3009
32	CM_logorc1_constant_3_1L	53.23	0.1745	644.57	0.2540	40.71	0.2979
64	CM_logorc1_constant_3_1L	93.66	0.2483	1246.96	0.2429	81.40	0.2772
32	CM_logorc2_3_1L	32.10	0.2432	623.37	0.1528	34.99	0.1865
64	CM_logorc2_3_1L	53.97	0.1304	1207.38	0.3026	68.06	0.2389
32	CM_logorc2_always_3_1L	26.27	0.1672	477.31	0.1213	27.16	0.1604
64	CM_logorc2_always_3_1L	43.28	0.1524	929.01	0.1347	54.05	0.1384
32	CM_logorc2_const_always_3_1L	47.65	0.1887	498.32	0.1704	32.92	0.1884
64	CM_logorc2_const_always_3_1L	84.45	0.3060	968.55	0.1800	67.74	0.2292
32	CM_logorc2_constant_3_1L	53.22	0.1700	644.53	0.2251	40.70	0.1996
64	CM_logorc2_constant_3_1L	93.63	0.1484	1246.96	0.2198	81.39	0.2376
32	CM_logxor_3_1L	32.14	0.2608	623.38	0.1526	35.00	0.1899
64	CM_logxor_3_1L	53.99	0.1319	1207.36	0.3117	68.06	0.2490

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_logxor_always_3_1L	26.24	0.1685	477.30	0.1300	27.15	0.1539
64	CM_logxor_always_3_1L	43.27	0.1529	928.98	0.1306	54.04	0.1422
32	CM_logxor_const_always_3_1L	47.64	0.1761	498.29	0.1431	32.91	0.1751
64	CM_logxor_const_always_3_1L	84.07	0.1970	968.53	0.1727	67.64	0.1917
32	CM_logxor_constant_3_1L	53.22	0.1722	644.53	0.2162	40.70	0.1900
64	CM_logxor_constant_3_1L	93.63	0.1835	1246.95	0.2208	81.39	0.2351
32	CM_s_add_3_1L	35.07	0.1877	669.44	0.1414	37.83	0.1785
64	CM_s_add_3_1L	56.97	0.1530	1253.47	0.1649	71.01	0.1527
32	CM_s_add_carry_3_1L	35.62	0.1774	678.48	0.1752	38.39	0.1842
64	CM_s_add_carry_3_1L	57.46	0.1348	1262.41	0.1356	71.47	0.1635
32	CM_s_add_constant_3_1L	56.24	0.1521	690.69	0.1760	43.56	0.1744
64	CM_s_add_constant_3_1L	96.69	0.1591	1293.26	0.1829	84.37	0.1985
32	CM_s_ceiling_3_1L	1191.19	0.4202	19300.21	0.5169	1199.61	0.4759
64	CM_s_ceiling_3_1L	3880.09	0.6085	62595.29	9.7806	3901.36	8.1275
32	CM_s_ceiling_constant_3_1L	1197.30	0.3457	19110.81	0.5435	1192.73	0.5923
64	CM_s_ceiling_constant_3_1L	3868.17	0.5422	62034.30	0.7361	3875.59	0.7438
32	CM_s_floor_3_1L	1195.96	0.3484	19365.06	0.6890	1203.96	0.6127
64	CM_s_floor_3_1L	3889.71	0.5830	62744.22	0.5579	3910.89	0.5739
32	CM_s_floor_constant_3_1L	1202.15	0.3520	19194.40	0.7794	1197.82	0.6820
64	CM_s_floor_constant_3_1L	3878.19	0.5312	62193.10	0.8088	3885.47	0.7686
32	CM_s_max_3_1L	51.28	0.1756	1030.54	0.1740	58.03	0.1768
64	CM_s_max_3_1L	91.80	0.1458	1990.97	0.1637	114.26	0.1675
32	CM_s_max_constant_3_1L	72.95	0.1701	1051.85	0.2077	63.87	0.1995
64	CM_s_max_constant_3_1L	131.58	0.1636	2030.74	0.2800	127.64	0.2817
32	CM_s_min_3_1L	51.30	0.1899	1032.99	0.1830	58.13	0.1949
64	CM_s_min_3_1L	91.77	0.1682	1993.38	0.1735	114.34	0.1873
32	CM_s_min_constant_3_1L	72.98	0.1625	1054.16	0.1695	63.96	0.1954
64	CM_s_min_constant_3_1L	131.58	0.1557	2033.28	0.3148	127.72	0.2715
32	CM_s_mod_3_1L	1079.46	0.2980	17603.07	0.3831	1091.37	0.3461
64	CM_s_mod_3_1L	3680.74	0.4437	59577.70	0.5889	3710.26	0.4874
32	CM_s_mod_constant_3_1L	1096.37	0.2708	17531.98	0.5073	1092.62	0.4231
64	CM_s_mod_constant_3_1L	3716.82	0.4698	59438.89	0.5813	3716.89	0.6272
32	CM_s_multiply_3_1L	51.43	0.3576	562.79	0.2520	38.68	0.2922
64	CM_s_multiply_3_1L	3669.47	0.1550	58595.19	0.3902	3664.33	0.3302
32	CM_s_multiply_constant_3_1L	45.47	0.3092	470.29	0.2393	32.93	0.2632
64	CM_s_multiply_constant_3_1L	240.78	0.1663	3544.80	0.7800	221.34	0.4682

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_s_rem_3_1L	1067.56	0.3353	17373.44	0.8031	1077.63	0.8397
64	CM_s_rem_3_1L	3655.85	0.3575	59143.32	0.4565	3680.83	0.5246
32	CM_s_rem_constant_3_1L	1088.74	0.2878	17296.18	0.4710	1079.49	0.5177
64	CM_s_rem_constant_3_1L	3695.22	0.4275	58999.77	0.7935	3688.38	0.7099
32	CM_s_round_3_1L	1234.60	0.4487	20004.27	0.3769	1243.26	0.5845
64	CM_s_round_3_1L	3963.88	0.5488	63950.68	0.5220	3985.79	0.5630
32	CM_s_round_constant_3_1L	1240.81	0.2854	19804.59	0.5392	1236.17	0.5167
64	CM_s_round_constant_3_1L	3952.61	0.4895	63376.02	0.8632	3959.46	0.7508
32	CM_s_subfrom_constant_3_1L	57.18	0.1859	691.16	0.1881	43.78	0.2133
64	CM_s_subfrom_constant_3_1L	97.23	0.1507	1293.73	0.2199	84.49	0.1962
32	CM_s_subtract_3_1L	35.06	0.1794	669.44	0.1330	37.83	0.1716
64	CM_s_subtract_3_1L	56.95	0.1587	1253.46	0.1626	71.00	0.1572
32	CM_s_subtract_borrow_3_1L	36.49	0.1765	678.92	0.1848	38.60	0.2135
64	CM_s_subtract_borrow_3_1L	57.91	0.1734	1262.88	0.1331	71.62	0.1700
32	CM_s_subtract_constant_3_1L	57.18	0.1766	691.14	0.1684	43.78	0.2059
64	CM_s_subtract_constant_3_1L	97.22	0.1458	1293.71	0.1967	84.54	0.1879
32	CM_u_add_3_1L	34.62	0.1946	657.57	0.1322	37.21	0.1510
64	CM_u_add_3_1L	56.49	0.1510	1241.69	0.1552	70.30	0.1619
32	CM_u_add_carry_3_1L	35.18	0.1954	666.68	0.1548	37.77	0.1789
64	CM_u_add_carry_3_1L	57.00	0.1416	1250.68	0.1499	70.87	0.1675
32	CM_u_add_constant_3_1L	55.80	0.1456	678.84	0.1332	42.94	0.1587
64	CM_u_add_constant_3_1L	96.29	0.1750	1281.52	0.1727	83.66	0.1868
32	CM_u_ceiling_3_1L	1052.59	0.2070	17045.68	0.5652	1059.56	0.4324
64	CM_u_ceiling_3_1L	3627.98	0.4153	58561.42	0.8057	3649.09	0.6279
32	CM_u_ceiling_constant_3_1L	1061.46	0.3183	16914.31	0.6023	1056.07	0.5561
64	CM_u_ceiling_constant_3_1L	3618.99	0.5305	58018.00	0.8521	3625.00	0.8067
32	CM_u_floor_3_1L	1022.72	0.3386	16584.19	0.3993	1030.35	0.3867
64	CM_u_floor_3_1L	3572.24	0.4219	57645.08	0.4844	3592.49	0.6023
32	CM_u_floor_constant_3_1L	1031.19	0.2058	16442.17	0.5436	1026.36	0.5636
64	CM_u_floor_constant_3_1L	3563.41	0.5678	57137.35	0.8451	3569.79	0.7701
32	CM_u_max_3_1L	51.12	0.1784	1028.17	0.1460	57.89	0.1669
64	CM_u_max_3_1L	91.62	0.1553	1988.70	0.1387	114.14	0.1513
32	CM_u_max_constant_3_1L	72.79	0.1607	1049.42	0.1587	63.72	0.1620
64	CM_u_max_constant_3_1L	131.48	0.1661	2028.52	0.2353	127.54	0.2128
32	CM_u_min_3_1L	51.10	0.1760	1030.49	0.1547	57.98	0.1641
64	CM_u_min_3_1L	91.62	0.1708	1990.91	0.1633	114.21	0.1801

Table 1 (cont'd.) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 64-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_u_min_constant_3_1L	72.81	0.1722	1051.76	0.1715	63.82	0.1833
64	CM_u_min_constant_3_1L	131.46	0.1559	2030.63	0.2419	127.60	0.2309
32	CM_u_mod_3_1L	985.42	0.2353	16002.43	0.5336	993.74	0.4095
64	CM_u_mod_3_1L	3503.61	0.3277	56563.34	0.3471	3524.53	0.3676
32	CM_u_mod_constant_3_1L	1002.41	0.2812	15924.07	0.4967	994.74	0.4211
64	CM_u_mod_constant_3_1L	3539.69	0.5316	56414.62	0.5908	3530.93	0.6087
32	CM_u_multiply_3_1L	947.02	0.1387	15035.38	0.2793	941.34	0.2927
64	CM_u_multiply_3_1L	3383.71	0.2126	54023.11	0.2518	3378.56	0.2342
32	CM_u_multiply_constant_3_1L	125.97	0.2652	1350.35	0.3342	92.00	0.3210
64	CM_u_multiply_constant_3_1L	181.06	0.1706	2578.46	0.2367	161.30	0.2129
32	CM_u_rem_3_1L	1008.56	0.2612	16418.53	0.5173	1018.12	0.4283
64	CM_u_rem_3_1L	3545.10	0.3141	57357.14	0.5803	3569.36	0.4725
32	CM_u_rem_constant_3_1L	1029.60	0.3154	16344.65	0.5927	1020.22	0.4602
64	CM_u_rem_constant_3_1L	3584.43	0.3321	57212.00	0.5441	3576.85	0.5699
32	CM_u_round_3_1L	1080.51	0.2692	17507.76	0.5303	1088.09	0.4336
64	CM_u_round_3_1L	3678.70	0.3515	59360.63	0.7659	3699.46	0.5967
32	CM_u_round_constant_3_1L	1089.01	0.3296	17365.59	0.4953	1084.08	0.5295
64	CM_u_round_constant_3_1L	3669.91	0.4206	58843.26	0.7684	3676.38	0.6962
32	CM_u_subfrom_constant_3_1L	55.83	0.1739	678.86	0.1576	42.95	0.1811
64	CM_u_subfrom_constant_3_1L	96.29	0.2186	1281.51	0.1837	83.66	0.1989
32	CM_u_subtract_3_1L	34.62	0.1354	657.58	0.1216	37.21	0.1335
64	CM_u_subtract_3_1L	56.48	0.1221	1241.70	0.1262	70.30	0.1481
32	CM_u_subtract_borrow_3_1L	35.16	0.1548	666.65	0.1287	37.77	0.1577
64	CM_u_subtract_borrow_3_1L	56.97	0.1263	1250.67	0.1175	70.86	0.1591
32	CM_u_subtract_constant_3_1L	55.83	0.1640	678.87	0.1545	42.95	0.1724
64	CM_u_subtract_constant_3_1L	96.27	0.1579	1281.50	0.1710	83.66	0.1825

Table 2. Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_c_acos_1_1L	2768.83	0.3406	35453.80	0.5419	2340.54	0.4997
64	CM_c_acos_1_1L	161019.49	1.1107	2563761.50	0.8360	160473.67	0.9203
32	CM_c_acosh_1_1L	3415.85	0.3472	43857.03	0.4864	2893.00	0.6351
64	CM_c_acosh_1_1L	190312.62	0.7103	3029783.04	0.8110	189643.53	0.8122
32	CM_c_asin_1_1L	2765.18	0.5821	35420.24	0.4371	2337.88	0.5703
64	CM_c_asin_1_1L	160971.26	0.8022	2563090.67	0.7721	160430.10	0.8000
32	CM_c_asinh_1_1L	2680.49	0.3426	34271.76	0.4843	2263.27	0.5461
64	CM_c_asinh_1_1L	160823.54	0.8307	2560953.00	0.8485	160292.64	0.8489
32	CM_c_atan_1_1L	3287.81	0.3542	42221.16	0.7197	2784.77	0.6139
64	CM_c_atan_1_1L	188234.76	0.8294	2996351.84	0.7917	187559.31	0.8203
32	CM_c_atanh_1_1L	2169.44	0.2949	27769.79	0.4291	1833.34	0.5223
64	CM_c_atanh_1_1L	149003.56	0.8326	2371410.28	0.8813	148450.05	0.9123
32	CM_c_c_signum_1_1L	585.30	0.3007	8043.03	0.2835	521.12	0.4149
64	CM_c_c_signum_1_1L	23179.17	0.5464	370351.91	0.9375	23158.27	0.6929
32	CM_c_conjugate_1_1L	8.86	0.1392	44.94	0.1048	4.17	0.1234
64	CM_c_conjugate_1_1L	8.87	0.1442	44.95	0.1113	4.59	0.1328
32	CM_c_cos_1_1L	2124.21	0.2604	26647.00	0.5315	1768.56	0.4761
64	CM_c_cos_1_1L	240566.09	0.9536	3829925.70	0.8987	239743.58	0.8905
32	CM_c_cosh_1_1L	1621.27	0.4020	22388.35	0.6056	1449.72	0.5966
64	CM_c_cosh_1_1L	197372.12	0.8373	3148424.21	0.8156	196964.10	1.3403
32	CM_c_exp_1_1L	869.55	0.2607	12053.99	0.4120	779.67	0.3211
64	CM_c_exp_1_1L	170197.41	0.9188	2710575.52	0.8415	169642.44	1.1029
32	CM_c_ln_1_1L	1405.86	0.2818	18198.23	0.6009	1198.14	0.5575
64	CM_c_ln_1_1L	113498.54	0.7802	1807944.36	1.0022	113151.76	1.0847
32	CM_c_negate_1_1L	14.98	0.1667	87.11	0.0978	7.56	0.1230
64	CM_c_negate_1_1L	14.97	0.1636	87.10	0.0879	8.20	0.1274
32	CM_c_reciprocal_1_1L	294.80	0.2031	4002.85	0.3044	260.20	0.2784
64	CM_c_reciprocal_1_1L	15655.23	0.5966	250693.34	0.6040	15666.61	0.5796
32	CM_c_sin_1_1L	2117.05	0.3166	26602.70	0.5980	1764.80	0.5564
64	CM_c_sin_1_1L	240554.08	0.8281	3829863.26	0.8802	239737.67	1.0967
32	CM_c_sinh_1_1L	1621.16	0.3930	22388.33	0.6032	1449.69	0.6195
64	CM_c_sinh_1_1L	197377.15	0.8876	3148503.55	0.8012	196969.05	1.2760
32	CM_c_sqrt_1_1L	933.93	0.3587	12163.54	0.4489	799.06	0.4244
64	CM_c_sqrt_1_1L	32125.36	0.6845	510883.27	0.8234	31987.44	0.7269
32	CM_c_tan_1_1L	4723.73	0.3334	59732.57	0.7948	3955.87	0.6730
64	CM_c_tan_1_1L	509591.88	0.8881	8105743.17	0.8871	507500.29	1.5760

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_c_tanh_1_1L	1561.03	0.3227	21094.78	0.5074	1373.41	0.4696
64	CM_c_tanh_1_1L	205250.77	0.7277	3272700.17	0.7728	204765.15	0.7675
32	CM_f_asinh_1_1L	870.77	0.4397	11890.77	0.4710	770.32	0.6004
64	CM_f_asinh_1_1L	52596.21	0.7560	837031.42	0.7624	52399.03	0.8019
32	CM_f_atan_1_1L	654.38	0.4627	7684.14	0.6120	519.10	0.6124
64	CM_f_atan_1_1L	60087.59	0.8162	957729.06	0.6795	59936.24	0.9608
32	CM_f_atanh_1_1L	918.37	0.4315	11552.87	0.5989	764.78	0.5972
64	CM_f_atanh_1_1L	48073.81	0.5563	763753.07	0.8495	47839.13	0.7801
32	CM_f_cos_1_1L	245.31	0.1461	3615.21	0.1630	229.20	0.1633
64	CM_f_cos_1_1L	53138.19	0.6204	850185.79	0.8707	53147.72	0.7282
32	CM_f_cosh_1_1L	718.84	0.5185	8512.74	0.6244	573.53	0.4990
64	CM_f_cosh_1_1L	63815.84	0.4961	1014551.38	0.8194	63538.70	0.7800
32	CM_f_exp_1_1L	277.29	0.3253	3728.90	0.1924	242.92	0.2796
64	CM_f_exp_1_1L	57098.01	0.6971	912913.29	0.5866	57082.05	0.7636
32	CM_f_exp2_1_1L	275.90	0.2037	3727.48	0.1998	242.54	0.2343
64	CM_f_exp2_1_1L	54550.32	0.6173	872269.39	0.5925	54539.32	0.7050
32	CM_f_f_ceiling_1_1L	459.86	0.3872	6307.59	0.5386	408.32	0.4190
64	CM_f_f_ceiling_1_1L	1751.24	0.4211	25802.81	0.5070	1657.68	0.5324
32	CM_f_f_floor_1_1L	462.57	0.4544	6105.71	0.3201	401.31	0.4488
64	CM_f_f_floor_1_1L	1753.54	0.4516	25541.40	0.6001	1640.57	0.5419
32	CM_f_f_round_1_1L	586.13	0.3392	9230.84	0.3183	576.36	0.6056
64	CM_f_f_round_1_1L	1209.87	0.3670	19523.22	0.7997	1218.11	0.5306
32	CM_f_f_signum_1_1L	27.83	0.2110	335.07	0.1732	22.45	0.1742
64	CM_f_f_signum_1_1L	41.28	0.1322	562.41	0.1789	36.94	0.1547
32	CM_f_ln_1_1L	305.56	0.1828	4550.47	0.2492	288.15	0.2906
64	CM_f_ln_1_1L	40533.99	0.8760	645663.45	0.8589	40409.82	1.1006
32	CM_f_log10_1_1L	306.55	0.2313	4550.50	0.2176	288.35	0.2893
64	CM_f_log10_1_1L	40533.52	0.7751	645663.41	0.8269	40409.64	1.0454
32	CM_f_log2_1_1L	297.18	0.1682	4434.85	0.2224	280.67	0.2240
64	CM_f_log2_1_1L	37979.73	0.5974	605013.25	0.7785	37865.10	1.0138
32	CM_f_negate_1_1L	9.65	0.1632	28.31	0.1542	3.53	0.1464
64	CM_f_negate_1_1L	9.64	0.1754	28.31	0.1389	4.08	0.1552
32	CM_f_sin_1_1L	242.48	0.1331	3644.19	0.1433	229.36	0.1454
64	CM_f_sin_1_1L	54212.47	0.5415	866918.05	0.6241	54197.28	0.7293
32	CM_f_sinh_1_1L	789.09	0.5545	9176.18	0.3773	621.37	0.5432
64	CM_f_sinh_1_1L	63905.66	0.6647	1015544.22	0.7283	63608.15	0.7782

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev	VPR 16 and Sdev	Ave(1,4,16,32,128) and Sdev
32	CM_f_sqrt_1_1L	193.67 0.2420	3015.25 0.2834	189.61 0.2638
64	CM_f_sqrt_1_1L	7382.55 0.4741	118004.78 0.8237	7377.31 0.7338
32	CM_f_tan_1_1L	286.67 0.1376	4350.81 0.3704	273.54 0.2793
64	CM_f_tan_1_1L	49194.44 0.6339	784391.50 0.7361	49081.59 0.7600
32	CM_f_tanh_1_1L	708.65 0.3714	9037.02 0.3858	596.81 0.5554
64	CM_f_tanh_1_1L	69117.64 0.9515	1099212.75 0.8116	68829.25 0.7967
32	CM_lognot_1_1L	20.82 0.1391	241.23 0.0879	16.33 0.1285
64	CM_lognot_1_1L	33.12 0.1207	449.01 0.1382	29.54 0.1302
32	CM_lognot_always_1_1L	20.82 0.1534	241.22 0.1117	16.33 0.1215
64	CM_lognot_always_1_1L	33.12 0.1109	449.00 0.1085	29.54 0.1151
32	CM_s_abs_1_1L	49.99 0.1265	709.91 0.2054	45.71 0.1680
64	CM_s_abs_1_1L	85.60 0.1250	1282.79 0.7009	81.76 0.4453
32	CM_s_isqrt_1_1L	673.05 0.3370	10668.68 0.3129	668.16 0.5039
64	CM_s_isqrt_1_1L	2200.19 0.4102	35106.04 0.5520	2195.89 0.5306
32	CM_s_negate_1_1L	27.48 0.1676	344.69 0.1373	22.90 0.1643
64	CM_s_negate_1_1L	44.89 0.1222	629.85 0.1422	40.98 0.1573
32	CM_u_isqrt_1_1L	690.00 0.2652	10946.88 0.4025	685.46 0.3826
64	CM_u_isqrt_1_1L	2226.01 0.4459	35526.89 0.8580	2222.09 0.7178
32	CM_u_negate_1_1L	27.57 0.1703	346.40 0.1533	22.98 0.1856
64	CM_u_negate_1_1L	44.99 0.1096	630.89 0.1686	41.05 0.1562
32	CM_c_acos_2_1L	2798.23 0.3734	35453.75 0.5378	2346.42 0.4970
64	CM_c_acos_2_1L	161040.90 0.8746	2563761.52 0.7879	160479.03 0.8498
32	CM_c_acosh_2_1L	3461.48 0.5907	43857.03 0.5391	2902.11 0.6781
64	CM_c_acosh_2_1L	190267.95 0.6560	3029783.17 0.8265	189632.37 0.8106
32	CM_c_add_2_1L	91.08 0.1387	1079.30 0.1761	72.75 0.1853
64	CM_c_add_2_1L	1901.23 0.5162	30730.15 0.4495	1916.18 0.4726
32	CM_c_add_always_2_1L	89.86 0.1508	913.13 0.2687	64.36 0.2573
64	CM_c_add_always_2_1L	1900.85 0.4044	30732.62 0.5984	1916.28 0.5314
32	CM_c_asin_2_1L	2800.25 0.3460	35679.78 0.4564	2357.92 0.5805
64	CM_c_asin_2_1L	161003.19 0.8417	2563604.85 0.8310	160462.36 0.8310
32	CM_c_asinh_2_1L	2709.96 0.3095	34271.36 0.5317	2269.15 0.5483
64	CM_c_asinh_2_1L	160844.76 0.7608	2560953.02 0.8082	160297.96 0.8047
32	CM_c_atan_2_1L	3323.67 0.5401	42221.21 0.6703	2791.95 0.6517
64	CM_c_atan_2_1L	188184.49 0.8654	2996351.85 0.8275	187546.75 0.8377
32	CM_c_atanh_2_1L	2203.73 0.3394	27769.84 0.4590	1840.18 0.5492
64	CM_c_atanh_2_1L	148982.99 0.9636	2371410.18 0.8474	148444.90 0.9386

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev	VPR 16 and Sdev	Ave(1,4,16,32,128) and Sdev
32	CM_c_c_signum_2_1L	628.10 0.2528	9022.13 0.4114	578.71 0.3807
64	CM_c_c_signum_2_1L	23288.13 0.6188	372263.91 0.8246	23275.45 0.7407
32	CM_c_conjugate_2_1L	50.46 0.1606	835.84 0.1785	52.08 0.1788
64	CM_c_conjugate_2_1L	86.10 0.1941	1588.94 0.1500	96.40 0.1687
32	CM_c_cos_2_1L	2093.34 0.2869	26219.83 0.5524	1740.73 0.4897
64	CM_c_cos_2_1L	240774.56 0.7141	3828157.48 0.7647	239706.53 1.2025
32	CM_c_cosh_2_1L	1623.18 0.3330	22388.08 0.5651	1450.10 0.5600
64	CM_c_cosh_2_1L	197726.15 0.8185	3148424.37 0.8196	197052.61 1.3392
32	CM_c_divide_2_1L	513.22 0.2315	6908.64 0.3460	450.34 0.3446
64	CM_c_divide_2_1L	28011.99 0.6157	447732.51 0.7687	27993.62 0.7046
32	CM_c_divide_always_2_1L	584.90 0.1767	7102.84 0.3974	475.45 0.3613
64	CM_c_divide_always_2_1L	27887.96 0.5138	445981.82 0.9210	27879.38 0.9445
32	CM_c_divinto_2_1L	513.21 0.2177	6908.61 0.3703	450.34 0.3474
64	CM_c_divinto_2_1L	27983.65 0.3764	447732.59 0.8249	27986.54 0.6681
32	CM_c_divinto_always_2_1L	584.90 0.1739	7102.71 0.4392	475.45 0.3668
64	CM_c_divinto_always_2_1L	27858.78 0.6209	445981.92 0.8924	27872.08 0.9361
32	CM_c_exp_2_1L	869.85 0.2560	12128.32 0.3943	783.42 0.3559
64	CM_c_exp_2_1L	169983.43 0.7296	2710765.49 0.8028	169598.17 0.8382
32	CM_c_f_cis_2_1L	484.38 0.1531	7329.98 0.2198	464.36 0.2340
64	CM_c_f_cis_2_1L	107669.46 0.6866	1717217.06 0.9521	107426.53 0.9992
32	CM_c_ln_2_1L	1411.87 0.2651	18198.31 0.5722	1199.37 0.5433
64	CM_c_ln_2_1L	113563.83 0.6799	1807944.54 0.9733	113168.10 1.0317
32	CM_c_multiply_2_1L	233.71 0.1757	3099.05 0.2119	202.93 0.2626
64	CM_c_multiply_2_1L	12350.28 0.4590	197383.57 0.6855	12341.88 0.7129
32	CM_c_multiply_always_2_1L	265.72 0.1859	2778.76 0.4918	194.20 0.3191
64	CM_c_multiply_always_2_1L	12269.87 0.3900	195824.32 0.7596	12248.20 0.7032
32	CM_c_negate_2_1L	51.07 0.1378	834.53 0.1986	52.15 0.1739
64	CM_c_negate_2_1L	86.75 0.1402	1587.45 0.1293	96.54 0.2200
32	CM_c_reciprocal_2_1L	393.68 0.2113	5672.57 0.3640	363.78 0.2969
64	CM_c_reciprocal_2_1L	15778.53 0.4320	252584.84 0.5178	15785.75 0.5426
32	CM_c_sin_2_1L	2086.03 0.3328	26175.61 0.4305	1736.93 0.5061
64	CM_c_sin_2_1L	240765.70 0.7583	3828102.51 1.0132	239703.21 1.3954
32	CM_c_sinh_2_1L	1623.19 0.3735	22388.22 0.6872	1450.11 0.6110
64	CM_c_sinh_2_1L	197728.84 0.8767	3148503.31 0.7976	197056.95 1.3646
32	CM_c_sqrt_2_1L	933.52 0.3544	12163.71 0.4948	798.98 0.4369
64	CM_c_sqrt_2_1L	32103.35 0.6212	511069.77 0.8473	31990.54 0.7136

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev	VPR 16 and Sdev	Ave(1,4,16,32,128) and Sdev
32	CM_c_subfrom_2_1L	103.63 0.1775	1169.89 0.1561	79.93 0.2636
64	CM_c_subfrom_2_1L	1916.44 0.3425	30858.20 0.6124	1926.31 0.5829
32	CM_c_subfrom_always_2_1L	99.45 0.1472	967.66 0.2494	69.11 0.1961
64	CM_c_subfrom_always_2_1L	1914.69 0.2593	30833.13 0.7408	1924.71 0.5829
32	CM_c_subtract_2_1L	90.98 0.1629	1079.21 0.1277	72.72 0.1577
64	CM_c_subtract_2_1L	1905.85 0.2583	30809.66 0.5221	1921.04 0.4599
32	CM_c_subtract_always_2_1L	89.87 0.1504	913.08 0.2657	64.36 0.2533
64	CM_c_subtract_always_2_1L	1905.42 0.6441	30808.26 0.3448	1920.86 0.3964
32	CM_c_tan_2_1L	4700.06 0.3280	59299.64 0.5258	3929.11 0.5570
64	CM_c_tan_2_1L	509984.65 0.8049	8103992.71 0.9323	507511.56 1.6119
32	CM_c_tanh_2_1L	1563.13 0.3745	21094.67 0.5248	1373.83 0.4720
64	CM_c_tanh_2_1L	205514.35 0.7925	3272700.22 0.8612	204831.05 0.8121
32	CM_f_exp2_2_1L	275.67 0.2426	3727.49 0.2063	242.53 0.2629
64	CM_f_exp2_2_1L	54753.58 0.6007	872269.54 0.5831	54590.14 0.6906
32	CM_f_f_ceiling_2_1L	532.73 0.3150	6590.33 0.3170	436.61 0.5348
64	CM_f_f_ceiling_2_1L	1806.09 0.3926	26118.53 0.5381	1676.85 0.5592
32	CM_f_f_floor_2_1L	536.71 0.3843	6665.75 0.3979	437.95 0.5026
64	CM_f_f_floor_2_1L	1710.77 0.5368	26149.54 0.4992	1652.94 0.5701
32	CM_f_f_round_2_1L	613.10 0.4479	9652.40 0.5444	604.97 0.6039
64	CM_f_f_round_2_1L	1254.32 0.3568	20321.89 0.7593	1266.94 0.5382
32	CM_f_f_signum_2_1L	27.92 0.1682	342.09 0.2202	22.73 0.1821
64	CM_f_f_signum_2_1L	41.30 0.1824	569.42 0.1456	37.27 0.1733
32	CM_f_ln_2_1L	305.58 0.1841	4738.74 0.2248	296.37 0.2498
64	CM_f_ln_2_1L	40575.91 0.8174	645663.43 0.8052	40420.25 1.0471
32	CM_f_log10_2_1L	306.55 0.2444	4738.70 0.2691	296.56 0.2682
64	CM_f_log10_2_1L	40575.55 0.6492	645663.43 0.8032	40420.15 1.0380
32	CM_f_log2_2_1L	297.26 0.2751	4622.26 0.3228	288.89 0.2854
64	CM_f_log2_2_1L	38023.70 0.8010	605013.42 0.8103	37876.10 1.0461
32	CM_f_max_2_1L	51.21 0.0887	710.33 0.1348	44.87 0.1551
64	CM_f_max_2_1L	163.20 0.1611	3240.50 0.1671	192.96 0.1762
32	CM_f_min_2_1L	50.74 0.1047	703.14 0.1603	44.42 0.1842
64	CM_f_min_2_1L	163.21 0.1699	3238.36 0.2437	192.86 0.2626
32	CM_f_mod_2_1L	1835.99 0.2964	26677.89 0.4573	1701.99 0.5232
64	CM_f_mod_2_1L	10002.99 0.5261	158144.61 0.8218	9919.44 0.6732
32	CM_f_multiply_2_1L	42.59 0.0944	472.29 0.1198	32.44 0.1373
64	CM_f_multiply_2_1L	2515.25 0.1567	40192.65 0.4191	2513.09 0.2946

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev	VPR 16 and Sdev	Ave(1,4,16,32,128) and Sdev
32	CM_f_multiply_always_2_1L	41.87 0.0869	386.89 0.1142	28.09 0.1258
64	CM_f_multiply_always_2_1L	2515.37 0.1589	40193.30 0.2705	2513.16 0.2460
32	CM_f_negate_2_1L	27.98 0.1423	418.83 0.1998	25.54 0.1714
64	CM_f_negate_2_1L	44.98 0.1272	795.36 0.1358	48.80 0.1702
32	CM_f_rem_2_1L	1623.41 0.2541	24471.17 0.8090	1550.31 0.5654
64	CM_f_rem_2_1L	8812.59 0.3581	140237.65 0.5520	8785.88 0.4415
32	CM_f_sin_2_1L	242.48 0.1844	3718.08 0.2402	233.99 0.1910
64	CM_f_sin_2_1L	54279.40 0.5972	867101.83 0.6492	54222.68 0.6843
32	CM_f_sinh_2_1L	788.86 0.4799	9267.66 0.5720	624.91 0.4390
64	CM_f_sinh_2_1L	63955.63 0.9940	1015731.09 0.8508	63629.36 0.9212
32	CM_f_sqrt_2_1L	193.69 0.2135	3015.18 0.2573	189.62 0.2217
64	CM_f_sqrt_2_1L	7382.58 0.4583	118004.95 0.8173	7377.29 0.7384
32	CM_f_subfrom_2_1L	48.06 0.1721	546.33 0.2062	37.23 0.2032
64	CM_f_subfrom_2_1L	991.53 0.3013	15987.27 0.3528	997.58 0.4039
32	CM_f_subfrom_always_2_1L	47.08 0.1920	459.26 0.1925	32.76 0.2314
64	CM_f_subfrom_always_2_1L	991.51 0.3476	15989.43 0.3519	997.66 0.4291
32	CM_f_subtract_2_1L	46.87 0.0989	540.96 0.1108	36.73 0.1243
64	CM_f_subtract_2_1L	954.51 0.2401	15406.38 0.4624	961.05 0.3901
32	CM_f_subtract_always_2_1L	46.33 0.0862	458.03 0.2334	32.54 0.1657
64	CM_f_subtract_always_2_1L	954.44 0.2865	15405.80 0.2920	961.00 0.2969
32	CM_f_tan_2_1L	286.64 0.1912	4425.43 0.2416	278.19 0.2558
64	CM_f_tan_2_1L	49236.78 0.8199	784573.98 0.8519	49100.90 0.7954
32	CM_f_tanh_2_1L	708.28 0.3461	9127.93 0.3837	600.22 0.3716
32	CM_s_isqrt_2_1L	673.04 0.2993	10668.70 0.3614	668.16 0.4999
64	CM_s_isqrt_2_1L	2200.20 0.5057	35105.97 0.5954	2195.89 0.5622
32	CM_s_max_2_1L	45.27 0.1782	821.97 0.1174	47.96 0.1635
64	CM_s_max_2_1L	81.06 0.1285	1574.87 0.2276	94.31 0.1632
32	CM_s_max_constant_2_1L	66.96 0.2084	843.41 0.1952	56.12 0.2281
64	CM_s_max_constant_2_1L	120.70 0.2151	1614.84 0.3527	107.65 0.3259
32	CM_s_min_2_1L	45.35 0.1835	819.67 0.1948	47.90 0.1635
64	CM_s_min_2_1L	80.98 0.1492	1572.71 0.1429	94.19 0.1325
32	CM_s_min_constant_2_1L	67.01 0.2420	841.13 0.2216	56.01 0.2236
64	CM_s_min_constant_2_1L	120.60 0.2077	1612.65 0.3887	107.52 0.3344
32	CM_s_mod_2_1L	1079.45 0.2800	17603.03 0.3599	1096.10 0.3487
64	CM_s_mod_2_1L	3698.54 0.3992	59577.67 0.6300	3717.50 0.5453

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_s_mod_constant_2_1L	1096.43	0.2754	17531.03	0.4077	1096.23	0.5514
64	CM_s_mod_constant_2_1L	4499.92	0.5043	59438.41	0.6861	3912.64	0.6032
32	CM_s_multiply_2_1L	1079.36	0.1515	17151.85	0.6038	1073.60	0.4673
64	CM_s_multiply_2_1L	3631.22	0.2108	57983.57	0.3302	3626.09	0.2651
32	CM_s_multiply_constant_2_1L	204.20	0.2694	2704.25	0.5425	177.27	0.3669
64	CM_s_multiply_constant_2_1L	280.42	0.2581	4313.05	0.7148	274.36	0.5540
32	CM_s_negate_2_1L	27.37	0.2182	437.10	0.0919	26.33	0.1815
64	CM_s_negate_2_1L	44.91	0.1727	813.43	0.1106	49.58	0.1525
32	CM_s_rem_2_1L	1044.46	0.2960	16959.80	0.8681	1056.91	0.7615
64	CM_s_rem_2_1L	3632.29	0.5318	58350.02	0.5077	3643.35	0.5252
32	CM_s_rem_constant_2_1L	1061.47	0.3174	16874.25	0.7137	1056.42	0.5891
64	CM_s_rem_constant_2_1L	4422.38	0.6323	58200.08	0.7086	3835.30	0.7107
32	CM_s_round_2_1L	1234.65	0.4651	20004.28	0.3175	1247.23	0.4480
64	CM_s_round_2_1L	4795.66	0.3324	63950.56	0.4782	4196.75	0.4658
32	CM_s_round_constant_2_1L	1240.13	0.4417	19796.60	1.4553	1238.29	1.0953
64	CM_s_round_constant_2_1L	4779.10	0.5892	63375.07	0.9727	4166.06	0.8602
32	CM_s_s_signum_2_1L	40.49	0.1201	557.37	0.1868	36.00	0.1571
64	CM_s_s_signum_2_1L	71.36	0.1798	1047.07	0.3057	67.19	0.2508
32	CM_s_subfrom_2_1L	34.43	0.2055	596.88	0.1555	35.00	0.1674
64	CM_s_subfrom_2_1L	56.92	0.1398	1105.95	0.1406	66.34	0.1399
32	CM_s_subfrom_constant_2_1L	57.19	0.2244	618.66	0.2596	42.98	0.2672
64	CM_s_subfrom_constant_2_1L	96.92	0.2045	1146.05	0.4041	79.78	0.3512
32	CM_s_subtract_2_1L	28.64	0.2191	460.50	0.1471	27.68	0.2105
64	CM_s_subtract_2_1L	46.12	0.1351	837.07	0.1912	51.00	0.1672
32	CM_s_subtract_borrow_2_1L	29.18	0.2028	469.66	0.1659	28.24	0.1757
64	CM_s_subtract_borrow_2_1L	46.67	0.1178	845.77	0.2040	51.55	0.1740
32	CM_s_subtract_constant_2_1L	34.97	0.2198	452.93	0.1869	29.82	0.2249
64	CM_s_subtract_constant_2_1L	64.76	0.2214	854.81	0.2701	56.73	0.2237
32	CM_swap_2_1L	39.24	0.1504	685.91	0.1499	40.38	0.1556
64	CM_swap_2_1L	70.20	0.1405	1326.72	0.1747	80.00	0.1723
32	CM_swap_always_2_1L	33.57	0.1855	534.24	0.1665	32.37	0.1608
64	CM_swap_always_2_1L	59.46	0.1376	1046.26	0.1587	64.15	0.1488
32	CM_transpose32_2_1L	22.27	0.1541	255.29	0.1246	17.30	0.1637
64	CM_transpose32_2_1L	35.80	0.1238	487.40	0.2731	32.04	0.2235
32	CM_u_add_2_1L	28.03	0.2291	446.67	0.1167	26.92	0.1551
64	CM_u_add_2_1L	45.53	0.1357	823.18	0.1098	50.20	0.1805

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_u_add_carry_2_1L	28.60	0.2110	455.61	0.1508	27.49	0.1793
64	CM_u_add_carry_2_1L	46.07	0.1307	832.13	0.1211	50.75	0.1274
32	CM_u_add_constant_2_1L	34.29	0.2427	441.73	0.2371	29.12	0.2419
64	CM_u_add_constant_2_1L	64.01	0.1562	843.60	0.1694	56.02	0.2172
32	CM_u_add_flags_2_1L	28.03	0.2353	446.67	0.1188	26.92	0.1554
64	CM_u_add_flags_2_1L	45.52	0.1512	823.18	0.1059	50.20	0.1790
32	CM_u_ceiling_2_1L	1052.60	0.2070	17045.47	0.5057	1063.10	0.4118
64	CM_u_ceiling_2_1L	4398.67	0.2387	58560.97	0.8254	3844.74	0.6789
32	CM_u_ceiling_constant_2_1L	1060.60	0.3120	16908.04	0.5815	1057.98	0.5192
64	CM_u_ceiling_constant_2_1L	4384.30	0.4880	58015.50	0.6433	3816.14	0.6412
32	CM_u_floor_2_1L	1022.80	0.2894	16583.87	0.3090	1033.92	0.3143
64	CM_u_floor_2_1L	4343.09	0.3136	57644.81	0.5310	3788.06	0.4673
·32	CM_u_floor_constant_2_1L	1030.62	0.3455	16436.36	0.6548	1028.37	0.5598
64	CM_u_floor_constant_2_1L	4328.34	0.5057	57134.85	0.7667	3760.86	0.6660
32	CM_u_from_gray_code_2_1L	26.04	0.2116	417.18	0.1516	25.04	0.1775
64	CM_u_from_gray_code_2_1L	43.50	0.1832	793.82	0.2456	48.31	0.1912
32	CM_u_isqrt_2_1L	689.97	0.2774	10946.85	0.3907	685.45	0.3552
64	CM_u_isqrt_2_1L	2226.08	0.4778	35526.62	0.7415	2222.06	0.6795
32	CM_u_max_2_1L	45.12	0.1865	819.52	0.2065	47.83	0.1596
64	CM_u_max_2_1L	80.95	0.1385	1572.65	0.1527	94.18	0.1384
32	CM_u_max_constant_2_1L	66.93	0.2097	841.01	0.2304	55.99	0.2216
64	CM_u_max_constant_2_1L	120.56	0.2135	1612.56	0.4114	107.50	0.3339
f 32	CM_u_min_2_1L	45.13	0.1970	817.18	0.1364	47.74	0.1541
64	CM_u_min_2_1L	80.95	0.1439	1570.29	0.1489	94.07	0.1548
32	CM_u_min_constant_2_1L	66.95	0.2058	838.71	0.1775	55.88	0.2024
64	CM_u_min_constant_2_1L	120.57	0.2078	1610.22	0.4256	107.40	0.3744
32	CM_u_mod_2_1L	985.42	0.2463	16002.47	0.4983	997.26	0.4450
64	CM_u_mod_2_1L	3521.60	0.7624	56563.36	0.3338	3531.94	0.4455
32	CM_u_mod_constant_2_1L	1002.46	0.3421	15923.86	0.4942	997.10	0.4971
64	CM_u_mod_constant_2_1L	4310.91	0.4997	56413.29	0.8817	3723.67	0.7354
32	CM_u_multiply_2_1L	947.05	0.1474	15035.44	0.2908	941.37	0.2622
64	CM_u_multiply_2_1L	3383.70	0.2269	54023.13	0.2600	3378.55	0.2462
32	CM_u_multiply_constant_2_1L	147.02	0.2968	1775.40	0.3947	119.35	0.3774
64	CM_u_multiply_constant_2_1L	222.29	0.2247	3379.97	0.3606	216.09	0.3170
32	CM_u_negate_2_1L	27.52	0.2539	437.80	0.1390	26.40	0.1907
64	CM_u_negate_2_1L	44.99	0.1802	814.11	0.1048	49.64	0.1400

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_u_rem_2_1L	985.43	0.2504	16002.56	0.5282	997.27	0.4318
64	CM_u_rem_2_1L	3521.63	0.7108	56563.34	0.3449	3531.94	0.4272
32	CM_u_rem_constant_2_1L	1002.40	0.3377	15923.84	0.4868	997.08	0.4806
64	CM_u_rem_constant_2_1L	4310.93	0.4853	56413.21	0.8583	3723.68	0.7758
32	CM_u_round_2_1L	1080.59	0.2967	17507.49	0.4346	1091.56	0.4247
64	CM_u_round_2_1L	4472.68	0.2929	59360.52	0.9056	3900.72	0.7305
32	CM_u_round_constant_2_1L	1088.52	0.3912	17360.49	0.5286	1086.06	0.5586
64	CM_u_round_constant_2_1L	4111.13	0.6560	58830.46	1.9684	3786.11	1.9257
32	CM_u_subfrom_2_1L	34.22	0.1425	584.68	0.1367	34.42	0.1487
64	CM_u_subfrom_2_1L	56.51	0.1275	1094.69	0.2271	65.71	0.2108
32	CM_u_subfrom_constant_2_1L	55.84	0.2345	606.02	0.2095	42.07	0.2250
64	CM_u_subfrom_constant_2_1L	96.11	0.2009	1134.52	0.3449	79.02	0.3076
32	CM_u_subtract_2_1L	28.02	0.1382	446.66	0.1316	26.92	0.1601
64	CM_u_subtract_2_1L	45.53	0.1070	823.18	0.1245	50.20	0.1164
32	CM_u_subtract_borrow_2_1L	28.60	0.2107	455.62	0.1538	27.48	0.1760
64	CM_u_subtract_borrow_2_1L	46.07	0.1168	832.13	0.1254	50.75	0.1212
32	CM_u_subtract_constant_2_1L	34.32	0.2325	441.76	0.2259	29.12	0.2358
64	CM_u_subtract_constant_2_1L	64.01	0.1639	843.59	0.1569	56.02	0.2026
32	CM_u_to_gray_code_2_1L	52.07	0.0892	865.28	0.4050	51.58	0.2270
64	CM_u_to_gray_code_2_1L	92.50	0.1257	1699.08	0.2299	103.16	0.1743
32	CM_c_add_3_1L	92.91	0.1694	1090.21	0.2244	73.68	0.2465
64	CM_c_add_3_1L	2007.24	0.2761	32321.77	0.9263	2017.11	0.6889
32	CM_c_add_always_3_1L	90.94	0.1270	915.45	0.2062	64.69	0.2317
64	CM_c_add_always_3_1L	2007.51	0.3553	32322.70	0.8366	2017.38	0.6207
32	CM_c_divide_3_1L	481.91	0.1443	6283.31	0.2860	412.74	0.2600
64	CM_c_divide_3_1L	27903.69	0.3501	446503.71	0.7929	27908.83	0.6392
32	CM_c_divide_always_3_1L	554.46	0.1913	6478.40	0.3136	438.08	0.2972
64	CM_c_divide_always_3_1L	27779.73	0.5337	444752.99	1.1549	27794.63	1.1103
32	CM_c_multiply_3_1L	192.61	0.1834	2305.51	0.2712	154.97	0.2184
64	CM_c_multiply_3_1L	12251.02	0.4784	195837.50	0.7768	12244.44	0.7253
32	CM_c_multiply_always_3_1L	261.06	0.1735	2466.64	0.2821	177.87	0.2466
64	CM_c_multiply_always_3_1L	12177.44	0.3475	194668.06	1.2191	12171.10	1.1092
32	CM_c_subtract_3_1L	92.92	0.1698	1090.18	0.2427	73.68	0.2563
64	CM_c_subtract_3_1L	2002.80	0.3003	32331.70	0.5215	2016.71	0.4652
32	CM_c_subtract_always_3_1L	90.94	0.1446	915.47	0.2035	64.69	0.2345
64	CM_c_subtract_always_3_1L	2002.74	0.4277	32332.35	0.6326	2016.75	0.5532

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev	VPR 16 and Sdev	Ave(1,4,16,32,128) and Sdev
32	CM_f_add_3_1L	48.89 0.1676	546.71 0.1680	37.41 0.1779
64	CM_f_add_3_1L	990.59 0.2402	16161.29 0.8143	1005.42 0.6124
32	CM_f_add_always_3_1L	47.90 0.1459	459.26 0.2019	32.94 0.2273
64	CM_f_add_always_3_1L	990.87 0.2759	16163.06 0.7129	1005.61 0.5849
32	CM_f_atan2_3_1L	769.41 0.5260	9049.29 0.4131	610.38 0.5406
64	CM_f_atan2_3_1L	59420.89 0.7650	946444.34 0.8650	59241.05 0.7575
32	CM_f_divide_3_1L	141.82 0.2476	2243.19 0.2201	140.81 0.2637
64	CM_f_divide_3_1L	4821.75 0.1726	77327.01 0.3265	4830.26 0.2615
32	CM_f_divide_always_3_1L	95.32 0.1631	1403.35 0.2129	89.43 0.2096
64	CM_f_divide_always_3_1L	4821.59 0.1995	77328.79 0.2838	4830.30 0.2543
32	CM_f_max_3_1L	74.90 0.1431	1127.77 0.1627	70.58 0.1909
64	CM_f_max_3_1L	234.55 0.1700	4658.44 0.1722	277.49 0.1791
32	CM_f_min_3_1L	74.43 0.1825	1120.62 0.1628	70.13 0.2274
64	CM_f_min_3_1L	234.54 0.2054	4656.13 0.2057	277.38 0.2657
32	CM_f_mod_3_1L	1787.12 0.2649	26725.88 0.4389	1699.05 0.5066
64	CM_f_mod_3_1L	10034.32 0.5713	159097.28 0.7982	9969.64 0.6442
32	CM_f_multiply_3_1L	44.60 0.1663	478.10 0.1234	33.13 0.1469
64	CM_f_multiply_3_1L	2553.69 0.1923	40968.89 0.5249	2559.19 0.4299
32	CM_f_multiply_always_3_1L	43.45 0.1695	388.10 0.1475	28.48 0.1645
64	CM_f_multiply_always_3_1L	2554.09 0.1841	40977.74 1.4530	2559.66 1.2762
32	CM_f_rem_3_1L	1620.48 0.2508	24518.97 0.5508	1552.94 0.5701
64	CM_f_rem_3_1L	8851.69 0.3131	140897.19 0.7319	8822.84 0.6169
32	CM_f_subtract_3_1L	48.91 0.1691	546.75 0.1914	37.42 0.1896
64	CM_f_subtract_3_1L	991.54 0.2329	16167.38 0.3241	1006.01 0.2905
32	CM_f_subtract_always_3_1L	47.89 0.1696	459.27 0.2147	32.94 0.2454
64	CM_f_subtract_always_3_1L	991.45 0.2518	16167.53 0.4237	1006.01 0.3714
32	CM_logand_3_1L	32.22 0.2379	623.37 0.1445	36.82 0.1886
64	CM_logand_3_1L	53.99 0.1170	1207.50 0.3085	70.35 0.2203
32	CM_logand_always_3_1L	26.27 0.1441	477.32 0.1251	29.02 0.1462
64	CM_logand_always_3_1L	43.26 0.1207	929.01 0.1394	54.61 0.1326
32	CM_logand_const_always_3_1L	47.69 0.2149	498.36 0.1814	34.79 0.2106
64	CM_logand_const_always_3_1L	84.04 0.2824	968.36 0.3023	68.18 0.2904
32	CM_logand_constant_3_1L	53.29 0.2330	644.57 0.2243	42.53 0.2400
64	CM_logand_constant_3_1L	93.36 0.2403	1246.84 0.3398	83.58 0.2976
32	CM_logandc1_3_1L	32.18 0.1908	623.37 0.1464	36.82 0.1759
64	CM_logandc1_3_1L	53.98 0.1208	1207.44 0.2626	70.35 0.2082

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_logandc1_always_3_1L	26.26	0.1484	477.31	0.1324	29.02	0.1532
64	CM_logandc1_always_3_1L	43.27	0.1312	928.99	0.1371	54.61	0.1321
32	CM_logandc1_const_always_3_1L	47.69	0.1994	498.35	0.1601	34.79	0.1945
64	CM_logandc1_const_always_3_1L	84.05	0.2775	968.38	0.3173	68.19	0.2924
32	CM_logandc1_constant_3_1L	53.26	0.1872	644.55	0.2093	42.52	0.2062
64	CM_logandc1_constant_3_1L	93.31	0.1819	1246.89	0.3774	83.57	0.2945
32	CM_logandc2_3_1L	32.20	0.2026	623.37	0.1434	36.82	0.1753
64	CM_logandc2_3_1L	53.99	0.1326	1207.41	0.2492	70.35	0.2076
32	CM_logandc2_always_3_1L	26.26	0.1418	477.32	0.1338	29.02	0.1509
64	CM_logandc2_always_3_1L	43.27	0.1263	929.00	0.1363	54.61	0.1285
32	CM_logandc2_const_always_3_1L	47.69	0.2029	498.34	0.1640	34.79	0.1945
64	CM_logandc2_const_always_3_1L	84.01	0.2742	968.36	0.3084	68.18	0.2845
32	CM_logandc2_constant_3_1L	53.25	0.1963	644.58	0.2207	42.52	0.2100
64	CM_logandc2_constant_3_1L	93.31	0.1821	1246.84	0.3361	83.57	0.2864
32	CM_logeqv_3_1L	32.20	0.1820	623.37	0.1434	36.82	0.1753
64	CM_logeqv_3_1L	53.99	0.1275	1207.37	0.2537	70.35	0.2072
32	CM_logeqv_always_3_1L	26.25	0.1477	477.33	0.1300	29.02	0.1523
64	CM_logeqv_always_3_1L	43.27	0.1338	929.00	0.1343	54.61	0.1323
32	CM_logeqv_const_always_3_1L	47.69	0.1980	498.36	0.1877	34.78	0.1978
64	CM_logeqv_const_always_3_1L	84.03	0.2563	968.38	0.3218	68.18	0.2848
32	CM_logeqv_constant_3_1L	53.26	0.1878	644.59	0.2105	42.52	0.2037
64	CM_logeqv_constant_3_1L	93.29	0.1636	1246.86	0.3630	83.56	0.2866
32	CM_logior_3_1L	32.19	0.1997	623.37	0.1396	36.82	0.1773
64	CM_logior_3_1L	54.01	0.1493	1207.41	0.2576	70.35	0.2129
32	CM_logior_always_3_1L	26.25	0.1480	477.32	0.1284	29.02	0.1504
64	CM_logior_always_3_1L	43.27	0.1277	928.99	0.1358	54.61	0.1313
32	CM_logior_const_always_3_1L	47.68	0.1900	498.35	0.1594	34.78	0.1869
64	CM_logior_const_always_3_1L	84.05	0.2663	968.37	0.3099	68.19	0.2840
32	CM_logior_constant_3_1L	53.28	0.1869	644.58	0.2275	42.52	0.2203
64	CM_logior_constant_3_1L	93.29	0.1532	1246.87	0.3574	83.56	0.2796
32	CM_lognand_3_1L	32.19	0.1844	623.36	0.1441	36.82	0.1737
64	CM_lognand_3_1L	53.99	0.1313	1207.40	0.2840	70.35	0.2164
32	CM_lognand_always_3_1L	26.27	0.1561	477.34	0.1084	29.02	0.1376
64	CM_lognand_always_3_1L	43.26	0.1283	929.01	0.1079	54.61	0.1174
32	CM_lognand_const_always_3_1L	47.69	0.1856	498.35	0.1802	34.79	0.1972
64	CM_lognand_const_always_3_1L	83.79	0.2473	968.37	0.3053	68.12	0.2867

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_lognand_constant_3_1L	53.28	0.1989	644.57	0.2145	42.52	0.2137
64	CM_lognand_constant_3_1L	93.30	0.1688	1246.86	0.3603	83.57	0.2876
32	CM_lognor_3_1L	32.19	0.1903	623.36	0.1435	36.82	0.1725
64	CM_lognor_3_1L	53.99	0.1288	1207.40	0.2699	70.35	0.2130
32	CM_lognor_always_3_1L	26.25	0.1436	477.32	0.1275	29.02	0.1486
64	CM_lognor_always_3_1L	43.27	0.1286	929.00	0.1417	54.61	0.1356
32	CM_lognor_const_always_3_1L	47.71	0.2258	498.37	0.1833	34.79	0.2098
64	CM_lognor_const_always_3_1L	84.06	0.2840	968.37	0.2818	68.19	0.2853
32	CM_lognor_constant_3_1L	53.24	0.1884	644.56	0.2015	42.52	0.2063
64	CM_lognor_constant_3_1L	93.30	0.1629	1246.87	0.3832	83.56	0.2974
32	CM_logorc1_3_1L	32.19	0.1940	623.37	0.1444	36.82	0.1718
64	CM_logorc1_3_1L	54.00	0.1346	1207.38	0.2857	70.35	0.2187
32	CM_logorc1_always_3_1L	26.26	0.1529	477.32	0.1267	29.02	0.1573
64	CM_logorc1_always_3_1L	43.26	0.1213	929.01	0.1302	54.61	0.1275
32	CM_logorc1_const_always_3_1L	47.69	0.1984	498.34	0.1614	34.78	0.1966
64	CM_logorc1_const_always_3_1L	84.01	0.2715	968.38	0.3094	68.18	0.2930
32	CM_logorc1_constant_3_1L	53.26	0.1740	644.56	0.2115	42.52	0.2091
64	CM_logorc1_constant_3_1L	93.31	0.1920	1246.85	0.3494	83.57	0.2931
32	CM_logorc2_3_1L	32.19	0.1858	623.37	0.1467	36.82	0.1702
64	CM_logorc2_3_1L	53.99	0.1278	1207.38	0.2705	70.35	0.2166
32	CM_logorc2_always_3_1L	26.25	0.1363	477.33	0.1315	29.02	0.1420
64	CM_logorc2_always_3_1L	43.29	0.1383	929.01	0.1216	54.62	0.1312
32	CM_logorc2_const_always_3_1L	47.69	0.1790	498.35	0.1642	34.79	0.1924
64	CM_logorc2_const_always_3_1L	84.07	0.3206	968.37	0.2930	68.19	0.3020
32	CM_logorc2_constant_3_1L	53.28	0.2142	644.56	0.2133	42.52	0.2153
64	CM_logorc2_constant_3_1L	93.34	0.1917	1246.88	0.3344	83.57	0.2894
32	CM_logxor_3_1L	32.19	0.1918	623.36	0.1343	36.82	0.1711
64	CM_logxor_3_1L	53.99	0.1340	1207.39	0.2688	70.35	0.2117
32	CM_logxor_always_3_1L	26.26	0.1517	477.32	0.1302	29.02	0.1536
64	CM_logxor_always_3_1L	43.27	0.1243	929.00	0.1304	54.61	0.1300
32	CM_logxor_const_always_3_1L	47.69	0.1952	498.34	0.1570	34.79	0.1945
64	CM_logxor_const_always_3_1L	84.00	0.2455	968.37	0.2977	68.17	0.2819
32	CM_logxor_constant_3_1L	53.27	0.1939	644.58	0.2162	42.52	0.2152
64	CM_logxor_constant_3_1L	93.31	0.1844	1246.84	0.3370	83.57	0.2806
32	CM_s_add_3_1L	35.06	0.1398	669.45	0.1387	39.75	0.1504
64	CM_s_add_3_1L	56.94	0.1384	1253.46	0.1348	73.26	0.1337

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_s_add_carry_3_1L	35.63	0.1406	678.49	0.1549	40.30	0.1680
64	CM_s_add_carry_3_1L	57.46	0.1344	1262.44	0.1314	73.81	0.1300
32	CM_s_add_constant_3_1L	56.29	0.1845	690.76	0.2096	45.51	0.2104
64	CM_s_add_constant_3_1L	96.47	0.1886	1293.15	0.3508	86.54	0.2834
32	CM_s_ceiling_3_1L	1191.28	0.3279	19299.86	0.3209	1202.27	0.4018
64	CM_s_ceiling_3_1L	4682.71	0.4991	62594.40	10.6714	4104.91	8.5385
32	CM_s_ceiling_constant_3_1L	1197.42	0.3915	19110.95	0.5366	1195.44	2.2916
64	CM_s_ceiling_constant_3_1L	4268.66	0.5840	62034.01	0.7704	3975.71	0.7507
32	CM_s_floor_3_1L	1196.02	0.4007	19365.06	0.6471	1206.73	3.7578
64	CM_s_floor_3_1L	4692.68	0.4579	62744.06	0.5426	4114.45	0.5181
32	CM_s_floor_constant_3_1L	1202.18	0.3989	19193.70	0.7156	1200.51	0.5615
64	CM_s_floor_constant_3_1L	4278.12	0.5790	62192.27	0.7474	3985.44	0.7525
32	CM_s_max_3_1L	51.28	0.1217	1030.54	0.1407	61.00	0.1662
64	CM_s_max_3_1L	91.78	0.1370	1990.93	0.1548	116.53	0.1493
32	CM_s_max_constant_3_1L	73.00	0.1800	1051.87	0.2120	66.85	0.2185
64	CM_s_max_constant_3_1L	131.33	0.2032	2030.73	0.3819	129.83	0.3376
32	CM_s_min_3_1L	51.31	0.1716	1033.01	0.1720	61.14	0.1984
64	CM_s_min_3_1L	91.74	0.1526	1993.39	0.1655	116.64	0.1546
32	CM_s_min_constant_3_1L	72.97	0.1597	1054.25	0.1997	66.97	0.2258
64	CM_s_min_constant_3_1L	131.35	0.2073	2033.28	0.3563	129.95	0.3301
32	CM_s_mod_3_1L	1079.47	0.2777	17603.06	0.3178	1094.99	0.3268
64	CM_s_mod_3_1L	4464.04	0.4231	59577.61	0.5841	3908.88	0.5183
32	CM_s_mod_constant_3_1L	1096.50	0.2779	17531.00	0.3429	1096.25	0.5345
64	CM_s_mod_constant_3_1L	3762.54	0.4401	59438.12	0.5368	3728.29	0.5064
32	CM_s_multiply_3_1L	1099.93	0.2284	17481.08	0.2928	1094.16	0.2756
64	CM_s_multiply_3_1L	3669.57	0.2114	58595.31	0.4016	3664.36	0.3526
32	CM_s_multiply_constant_3_1L	183.44	0.2177	2312.10	0.8556	151.60	0.5419
64	CM_s_multiply_constant_3_1L	240.81	0.2054	3546.28	0.4246	228.33	0.4315
32	CM_s_rem_3_1L	1067.61	0.3709	17373.42	0.7680	1080.14	0.6578
64	CM_s_rem_3_1L	4427.79	0.4446	59143.33	0.4885	3879.55	0.5578
32	CM_s_rem_constant_3_1L	1088.78	0.2996	17296.42	0.4938	1081.98	0.4507
64	CM_s_rem_constant_3_1L	3730.77	0.4959	58997.11	0.8096	3700.06	0.6966
32	CM_s_round_3_1L	1234.61	0.4456	20004.20	0.3574	1245.88	0.4839
64	CM_s_round_3_1L	4801.79	0.4568	63950.40	0.4830	4198.26	0.4798
32	CM_s_round_constant_3_1L	1240.90	0.3511	19804.68	0.5529	1238.75	0.5669
64	CM_s_round_constant_3_1L	4376.68	0.4945	63375.74	0.8085	4065.51	0.7247

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_s_subfrom_constant_3_1L	57.16	0.1672	691.15	0.1663	45.69	0.1851
64	CM_s_subfrom_constant_3_1L	96.93	0.1732	1293.52	0.2814	86.69	0.2482
32	CM_s_subtract_3_1L	35.05	0.1459	669.44	0.1361	39.75	0.1579
64	CM_s_subtract_3_1L	56.93	0.1408	1253.47	0.1403	73.26	0.1367
32	CM_s_subtract_borrow_3_1L	36.51	0.1583	678.94	0.1945	40.51	0.1962
64	CM_s_subtract_borrow_3_1L	57.91	0.1882	1262.89	0.1387	73.96	0.1507
32	CM_s_subtract_constant_3_1L	57.17	0.1788	691.16	0.1516	45.71	0.1852
64	CM_s_subtract_constant_3_1L	96.92	0.1581	1293.52	0.2971	86.69	0.2473
32	CM_u_add_3_1L	34.63	0.1584	657.58	0.1591	39.05	0.1630
64	CM_u_add_3_1L	56.49	0.1303	1241.71	0.1257	72.60	0.1313
32	CM_u_add_carry_3_1L	35.17	0.1492	666.67	0.1372	39.63	0.1711
64	CM_u_add_carry_3_1L	56.99	0.1405	1250.69	0.1165	73.14	0.1364
32	CM_u_add_constant_3_1L	55.85	0.1905	678.93	0.1770	44.79	0.2094
64	CM_u_add_constant_3_1L	96.05	0.2627	1281.46	0.3253	85.89	0.2883
32	CM_u_ceiling_3_1L	1052.63	0.2002	17045.43	0.5518	1062.05	0.4114
64	CM_u_ceiling_3_1L	4411.26	0.2655	58561.07	0.9166	3847.91	0.7481
32	CM_u_ceiling_constant_3_1L	1061.57	0.3498	16914.12	0.6281	1058.51	0.5472
64	CM_u_ceiling_constant_3_1L	4034.78	0.4613	58017.40	0.9804	3728.94	0.7919
32	CM_u_floor_3_1L	1022.93	0.2752	16583.86	0.3336	1032.75	0.3482
64	CM_u_floor_3_1L	4354.71	0.2957	57644.78	0.5102	3790.96	0.4470
32	CM_u_floor_constant_3_1L	1031.40	0.3654	16442.29	0.5726	1028.75	0.5405
64	CM_u_floor_constant_3_1L	3978.83	0.4851	57136.90	0.7995	3673.66	0.7189
32	CM_u_max_3_1L	51.12	0.1459	1028.20	0.1519	60.84	0.1653
64	CM_u_max_3_1L	91.60	0.1402	1988.70	0.1481	116.38	0.1413
32	CM_u_max_constant_3_1L	72.82	0.1820	1049.49	0.1913	66.68	0.2275
64	CM_u_max_constant_3_1L	131.15	0.2114	2028.55	0.3549	129.68	0.3213
32	CM_u_min_3_1L	51.11	0.1410	1030.52	0.1404	60.97	0.1657
64	CM_u_min_3_1L	91.61	0.1423	1990.90	0.1865	116.49	0.1692
32	CM_u_min_constant_3_1L	72.82	0.1903	1051.83	0.1983	66.81	0.2183
64	CM_u_min_constant_3_1L	131.16	0.2021	2030.63	0.3851	129.78	0.3337
32	CM_u_mod_3_1L	985.43	0.2194	16002.50	0.4767	996.09	0.4091
64	CM_u_mod_3_1L	4274.75	0.3882	56563.34	0.3401	3720.23	0.3582
32	CM_u_mod_constant_3_1L	1002.72	0.4715	15923.88	0.4987	997.15	0.5076
64	CM_u_mod_constant_3_1L	3574.21	0.4949	56413.02	0.8867	3539.48	0.7556
32	CM_u_multiply_3_1L	947.05	0.1638	15035.44	0.3674	941.37	0.2873
64	CM_u_multiply_3_1L	3383.65	0.2164	54023.15	0.2742	3378.54	0.2421

Table 2 (cont'd) Timings of Paris V6.1 Arithmetic Instructions on CM-2 with 32-bit FPA's

Size	Name	VPR 1 and Sdev		VPR 16 and Sdev		Ave(1,4,16,32,128) and Sdev	
32	CM_u_multiply_constant_3_1L	126.02	0.2638	1350.45	0.3243	92.11	0.2972
64	CM_u_multiply_constant_3_1L	181.09	0.1992	2578.51	0.2457	168.12	0.2825
32	CM_u_rem_3_1L	1008.61	0.2577	16418.56	0.5575	1020.45	0.4876
64	CM_u_rem_3_1L	4316.05	0.2663	57357.11	0.5464	3767.86	0.4228
32	CM_u_rem_constant_3_1L	1029.58	0.3317	16344.81	0.6207	1022.56	0.5018
64	CM_u_rem_constant_3_1L	3618.84	0.5166	57211.39	0.5493	3588.33	0.5715
32	CM_u_round_3_1L	1080.61	0.2279	17507.43	0.4323	1090.40	0.3690
64	CM_u_round_3_1L	3719.86	0.3624	59360.56	0.9150	3712.53	0.7610
32	CM_u_round_constant_3_1L	1089.24	0.3716	17365.73	0.4868	1086.52	1.7908
64	CM_u_round_constant_3_1L	3701.34	0.6297	58843.01	0.7061	3684.24	0.6923
32	CM_u_subfrom_constant_3_1L	55.83	0.2108	678.92	0.1689	44.80	0.2007
64	CM_u_subfrom_constant_3_1L	96.06	0.1789	1281.45	0.3199	85.89	0.2619
32	CM_u_subtract_3_1L	34.64	0.1509	657.58	0.1337	39.05	0.1467
64	CM_u_subtract_3_1L	56.49	0.1269	1241.71	0.1272	72.59	0.1355
32	CM_u_subtract_borrow_3_1L	35.18	0.1544	666.67	0.1606	39.63	0.1856
64	CM_u_subtract_borrow_3_1L	57.02	0.1539	1250.69	0.1646	73.15	0.1492
32	CM_u_subtract_constant_3_1L	55.83	0.2042	678.91	0.1774	44.78	0.2099
64	CM_u_subtract_constant_3_1L	96.04	0.1676	1281.45	0.3222	85.89	0.2751

Appendix B

Paris Version 6.1 Change Pages

B.1 What to Do with These Change Pages

Change pages correct and update a manual. The change pages in this packet provide dictionary entries for Paris instructions that are either new or updated with Version 6.1.

By page number, insert the change pages into your copy of the *Paris Reference Manual*, Version 6.0.

Placement of Change Pages

Change Page Sequence	Add after page	Replace pages
412a – 412d	412	
533 – 536d		533 – 536

PERMUTED-GET

Each selected processor gets a message from a specified source processor, possibly itself. A source processor may supply messages even if it is not selected. Messages are all retrieved from the same memory address within each source processor, and all the source processors may be in a VP set different from the VP set of the destination processors.

Use this operation for congested communication patterns; otherwise use **CM:get-1L**.

Formats CM:**permuted-get-1L** *dest*, *send-address*, *source*, *len*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates from which processor a message is retrieved.

source The field ID of the source field.

len The length, in bits, of the *dest* and *source* fields. This must be greater than zero and no greater than .

Overlap The *dest* and *send-address* may overlap in any manner. However, it is forbidden for the *source* to overlap with either the *send-address* or 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
    $\textit{dest}[k] \leftarrow \textit{source}[\textit{send-address}[k]]$

This operation is functionally equivalent to CM:**get-1L**: 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.

CM:**send-1L** and CM:**get-1L** behave poorly on a small class of communication patterns known as congested patterns. In contrast, CM:**permuted-send-1L** and CM:**permuted-get-1L** do a bit of extra work before the main communication step in an attempt to decongest the communication patterns. For congested patterns, the permuted routing functions are usually considerably faster than their simpler counterparts. Conversely, for patterns that are not congested, the permuted routing functions are slower. In addition, the permuted routing functions require more memory than their simpler counterparts.

PERMUTED-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.

Use this operation for congested communication patterns; otherwise use **CM:send-1L**.

Formats CM:permuted-send-1L *dest*, *send-address*, *source*, *len*, *notify*

Operands

dest The field ID of the destination field.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

len The length, in bits, of the *dest* and *source* fields. This must be greater than zero and no greater than .

notify The field ID of the notification bit (a one-bit field). This argument may be CM:*no-field* if no notification of message receipt is desired.

Overlap The *source* and *send-address* may overlap in any manner. However, it is forbidden for the *dest* to overlap with either the *send-address* or the *source* field.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the message, once transmitted to the receiving processor, is stored into the *dest* field regardless of the *context-flag* of the receiving processor. The *notify* bit may be altered in any processor regardless of the value of the *context-flag*.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \land \textit{context-flag}[m] = 1 \land \textit{send-address}[m] = k \}$
if $|S_k| = 0$ then
  if $\textit{notify}[k] \not\equiv$ CM:*no-field* then $\textit{notify}[k] \leftarrow 0$
else if $|S_k| = 1$ 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 *notify*[*k*] $\not\equiv$ CM:*no-field* then *notify*[*k*] $\leftarrow$ 1
dest[*k*] $\leftarrow$ ⟨undefined⟩

where the *choice* function arbitrarily but deterministically chooses an element from a set.

This operation is functionally equivalent to CM:send-1L: 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:**permuted-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.

CM:**send-1L** and CM:**get-1L** behave poorly on a small class of communication patterns known as congested patterns. In contrast, CM:**permuted-send-1L** and CM:**permuted-get-1L** do a bit of extra work before the main communication step in an attempt to decongest the communication patterns. For congested patterns, the permuted routing functions are usually considerably faster than their simpler counterparts. Conversely, for patterns that are not congested, the permuted routing functions are slower. In addition, the permuted routing functions require more memory than their simpler counterparts.

SEND-TO-NEWS

Each processor sends a message to a neighboring processor along a specified NEWS axis.

Formats	CM:send-to-news-1L	*dest, source, axis, direction, len*
	CM:send-to-news-always-1L	*dest, source, axis, direction, len*

Operands

- *dest* — The field ID of the destination field.
- *source* — The field ID of the source field.
- *axis* — An unsigned integer immediate operand to be used as the number of a NEWS axis.
- *direction* — Either :upward or :downward.
- *len* — The length, in bits, of the *dest* and *source* fields. This must be greater than zero and no greater than CM:*maximum-integer-length*.

Overlap The *source* field must be either disjoint from or identical to the *dest* field. Two bit fields are identical if they have the same address and the same length.

Context This operation is conditional, but whether data is copied depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is stored into the field indicated by *dest* regardless of the *context-flag* of the receiving processor.

Note that in the conditional case the storing of data depends only on the *context-flag* of the processor sending the data, not on the *context-flag* of the processor receiving the data.

Definition For every virtual processor *k* in the *current-vp-set* do
  if (always or *context-flag*[*k*] = 1) then
    let *g* = *geometry*(*current-vp-set*)
    dest[*news-neighbor*(*g*, *k*, *axis*, *direction*)] ← *source*[*k*]

The *source* field in each processor is stored into the *dest* field of that processor's neighbor along the NEWS axis specified by *axis* in the direction specified by *direction*.

If *direction* is :upward then each processor stores data into the neighbor whose NEWS coordinate is one greater, with the processor whose coordinate is greatest storing data into the processor whose coordinate is zero.

If *direction* is :downward then each processor stores data into the neighbor whose NEWS coordinate is one less, with the processor whose coordinate is zero storing data into the processor whose coordinate is greatest.

SEND-TO-QUEUE32

Sends a message from every selected processor to a specified destination processor and stores it there, as if by **aset32**, in a queue. Each selected processor may specify any processor as the destination, including itself. A destination processor may receive messages even if it is not selected, and all the destination processors may be in a VP set different from the VP set of the source processors.

Formats CM:send-to-queue32-1L *dest*, *send-address*, *source*, *slen*, *index-limit*

Operands *dest* The field ID of the queue field. The length of this field must accommodate 32 bits for the *queue.count* subfield, plus *index-limit* × *slen* bits for the *queue.elements* subfield, where *index-limit* is the number of queue elements in each processor.

send-address The field ID of the send address field. For each processor, this indicates to which processor a message is sent.

source The field ID of the source field.

slen The length, in bits, of the *source* field. This must be greater than zero and no greater than . This is also the length in bits of each queue element. Values may be either 32, 64, 96, or 128.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for a zero-based index into *queue.elements*. The value of this argument must be at least 1 and should never exceed the number of elements that can be stored in the queue.

Overlap The fields *send-address* and *source* may overlap in any manner. No overlap with the *dest* field is allowed.

Context This operation is conditional, but whether a message is sent depends only on the *context-flag* of the originating processor; the data, once transmitted to the receiving processor, is queued in the field indicated by *dest* regardless of the *context-flag* of the receiving processor.

Definition For every virtual processor k in the *current-vp-set* do

let $S_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \textit{send-address}[m] = k \}$
let T_k be a sub-set of S_k where $|T_k| = \min(|S_k| + \textit{queue.count}, \textit{index-limit})$
for i from *queue.count* to $\textit{queue.count} + |T_k| - 1$ do
  $\textit{queue.elements}[i] \leftarrow T_k[i]$
$\textit{queue.count} \leftarrow \textit{queue.count} + |S_k|$

Note that if $(|S_k| + \textit{queue.count} \geq \textit{index-limit})$ then there is some choice in picking the elements of T_k.

The destination field is treated as two subfields: *queue.count* and *queue.elements*. *Queue.count* is 32 bits long and records the number of enqueued messages. *Queue.elements* stores the enqueued messages; it is formatted as a slicewise array (accessed using aref32 and aset32), and starts at an offset of 32 bits from the start of the destination field. Its length is a multiple of the message length: at least *index-limit* × *slen* and possibly greater.

The *index-limit* argument specifies the maximum number of elements that any processor's *queue.elements* subfield may accumulate. If any processor receives more messages than this specified number, the queue overflows and messages are lost. If a *queue.elements* subfield overflows, the *queue.count* subfield for that processor nonetheless accurately reflects the number of messages received.

For any given communication pattern, both the order of message queueing and the selection of messages preserved or discarded in case of queue overflow are deterministic. That is, the order and selection of enqueued messages can be predictably reproduced from one invocation to the next.

This determinism is especially important for applications that use successive CM:send-to-queue32-1L calls to send large data structures by breaking up them up into chunks of length *slen*. By holding the *send-address* argument constant, such applications can send successive chunks of *slen* bits each to corresponding queues.

To prepare an empty queue for a CM:send-to-queue-1L instruction, the *queue.count* subfield should be set to zero. From Lisp/Paris, this is done by executing the following code in the destination context:

```
(let    ((zeros (allocate-stack-field 32))
  (context-hold (allocate-stack-field 1)))
  (cm:move-constant-always zeros 0 32)
  (cm:store-context context-hold)
  (cm:set-context)
  (cm:aset32-2L zeros queue zeros 32 32 1)
  (cm:load-context context-hold)
)
```

The CM:send-to-queue32-1L operation is conditional on the context of the source field; the set of queues that will *receive* messages is independent of the currently active set. To zero the *queue.count* subfield in only those queues that are to receive messages, execute the following code in the source context:

```
(let    ((zeros (allocate-stack-field 32)))
  (cm:move-constant-always zeros 0 32)
  (cm:send-aset32-overwrite-2L queue dest zeros zeros 32 32 1)
)
```

After the CM:send-to-queue32 operation, the local count can be retrieved by executing the following code in the destination context:

```
(let ((zeros (allocate-stack-field 32))
  (count-field (allocate-stack-field 32))
  )
  (cm:move-constant-always zeros 0 32)
  (cm:aref32-2L count-field queue zeros 32 32 1)
)
```

The i^{th} message can be retrieved from *queue.elements* by executing the following code in the destination context:

```
(let ((index (allocate-stack-field 32))
  (data-field (allocate-stack-field message-length))
  )
  (cm:move-constant-always index i 32)
  (cm:aref32-2L data-field (+ 32 queue) index len 32 queue-size)
)
```

Note that *queue.elements* is offset from the queue field by 32 bits.

An artificially small queue size may be used by passing CM:send-to-queue-1L an *index-limit* value that is less than the number of elements of length *slen* that could be stored in the *queue.elements* portion of the destination field. If this is done, the queues will be partially filled. However, the correct queue size should always be used as the *index-limit* argument to CM:aref32-2L when reading elements from the queue.

SEND-TO-QUEUE32-SHARED

Sends a message from every selected processor to a specified destination "sprint node" and stores it there, as if by **CM:aset32-shared-1L**, in a queue. (A sprint node consists of 32 physical processors.) Each selected processor may specify any node as the destination, including its own. A destination node may receive messages even if none of its processors are selected. Messages of length 32, 64, 96, or 128 bits are supported.

Formats **CM:send-to-queue32-shared-1L** *dest*, *node-address*, *source*, *slen*, *index-limit*

Operands

dest The field ID of the queue field. Must be allocated as a contiguous block of memory of length $1 + slen/32 \times index\text{-}limit$ bits. This is a compound field consisting of two adjacent slicewise shared arrays: *queue.count* and *queue.elements*. The *queue.count* array consists of one 32-bit slice (one bit per physical processor) used to record the number of messages received by the sprint node. The *queue.elements* array stores the messages received. The rest of this dictionary entry assumes the field is allocated in a VP set with vp-ratio 1; however, this constraint is not mandatory.

node-address The field ID of the node address field. For each processor, this indicates the address of a sprint node. For each source processor this specifies the node that receives that processor's message. A node-address consists solely of the off-chip bits of a send address (no VP bits and no on-chip bits). The length of this field will vary with machine size.

source The field ID of the source field. Must be of length slen.

slen An unsigned integer immediate operand to be used as the length of the source field and the length of each message in the *queue.elements* array.. Value must be either 32, 64, 96, or 128.

index-limit An unsigned integer immediate operand to be used as the exclusive upper bound for a zero-based index into *queue.elements*. The value of this argument must be at least 1 and should never exceed the number of elements that can be stored in the queue.

Overlap The fields *node-address* and *source* may overlap in any manner. No overlap with the *dest* field is allowed.

Context This operation is conditional, but whether a message is sent depends only on the context flag of the source processor. The message, once transmitted to the destination node, is queued regardless of any context flag setting in the processors of the destination node.

Definition For every sprint node k

let $T_k = \{ m \mid m \in \textit{current-vp-set} \wedge \textit{context-flag}[m] = 1 \wedge \lfloor \textit{node} - \textit{address}[m]/2 \rfloor = k \}$
let Q_k be a subset of T_k where $|Q_k| = \min(|T_k|, \textit{index-limit})$
queue.count $\leftarrow |T_k|$
for i from 0 to $|Q_k| - 1$ do
  queue.elements$[i] \leftarrow Q_k[i]$

Note that if $(|T_k| > \textit{index-limit})$ then there is some choice in picking the elements of T_k.

This instruction enqueues messages in a slicewise shared array and records the number of messages received by each node.

A slicewise shared array is considered *shared* because each parallel instance of the array is stored by *node* rather than by processor. A node consists of 32 physical processors and all of their associated memory. Consequently, for this instruction, the basic data unit in a queue is a *slice*, which includes one physical bit for each of 32 physical processors. Slicewise shared arrays are accessed by CM:aref32-shared-1L and CM:aset32-shared-1L.

Conceptually, the destination field is treated as a compound field containing two adjacent slicewise shared arrays: *queue.count* and *queue.elements*. The *queue.count* subfield is a slicewise shared array of length 1; *queue.elements* is a slicewise shared array of length *slen*/32 × *index-limit*. These lengths are in units of slices (one bit per physical processor).

The *queue.count* array records the number of enqueued messages that have been received by a particular node. The *queue.elements* array stores the enqueued messages, starting at an offset of one slice (one bit per physical processor) from the start of the destination field. The length of the *queue.elements* array (in slices) is (*max-number-of-messages* × *slen*/32), where *max-number-of-messages* is the maximum number of messages that each queue must be able to store.

The source field provides messages, which must all be of the same length: *slen*. The only allowed message lengths are 32, 64, 96, and 128 bits (1, 2, 3, or 4 slices).

The *index-limit* argument specifies the maximum number of messages that any node's queue may accumulate. If more than *index-limit* messages are sent to any queue, they are discarded – but the *queue.count* array is updated to accurately reflect the number of messages received, including those discarded.

A *node-address* consists of the off-chip bits of a send address. One way to create a node address is to generate a send address and extract the off-chip bits by skipping any VP bits and on-chip bits. (See the code sample below.)

Each node contains two chips; the lowest bit in the sequence of off-chip bits distinguishes between the two chips. To send a message to the queue of a particular node, the address of either chip may be used. The choice may effect performance. To maximize performance, the two chips should be chosen with roughly equal frequency. If the frequencies cannot be predicted, the programmer may wish to set the low bit of the chip address randomly.

Programming Help

To prepare an empty queue for a CM:send-to-queue32-shared-1L instruction, the *queue.count* array should be set to zero by executing the following Lisp/Paris code (or its equivalent in C/ or Fortan/Paris) in the destination context (which is the physical-vp-set).

```
(let  ((zeros (CM:allocate-stack-field 32))
   (context-hold (CM:allocate-stack-field 1))
   )
  (cm:store-context context-hold)
  (cm:set-context)
  (cm:move-constant-always zeros 0 32)
  (cm:aset32-shared-2L zeros queue zeros 32 32 1)
  (cm:load-context context-hold)
)
```

To derive the node addresses from the send addresses so that each message is sent to the queue on its processor's local node, the following Lisp/Paris code (or its equivalent in C/ or Fortan/Paris) should be executed in the source context.

```
(with-stack-fields
    ((self-address 32)
     (node-address 12) ; size for a 64K CM2

;; Zero out the self-address and node-address fields.
  (cm:move-constant-always self-address 32)
  (cm:move-constant-always node-address 12)

  (cm:my-send-address self-address)
  (let* (
     (send-address-length (cm:geometry-send-address-length source-geometry))
     (physical-length cm:*physical-processors-length*)
     (virtual-length (- send-address-length physical-length))
     (on-chip-length 4)
     (off-chip-offset (+ virtual-length on-chip-length))
     (off-chip-length (- physical-length on-chip-length))
  )
  (cm:move-always node-address
     (cm:add-offset-to-field-id self-address off-chip-offset)
     off-chip-length
```

After a CM:send-to-queue32-shared-1L operation, the queue count for each node can be retrieved by all virtual processors associated with the node. Execute the following Lisp/Paris code (or its equivalent in C/ or Fortan/Paris) in the destination context:

```
(let ((zeros (CM:allocate-stack-field 32))
      (count-field (CM:allocate-stack-field 32))
     )
   (cm:move-constant-always zeros 0 32)
   (cm:aref32-shared-2L count-field queue zeros 32 32 1)
 )
```

The i^{th} message can be retrieved from the *queue.elements* array by executing the following Lisp/Paris code (or its equivalent in C/ or Fortan/Paris) in the destination context:

```
(let ((index (CM:allocate-stack-field 32))
      (data-field (CM:allocate-stack-field message-length))
      (queue.elements (cm:add-offset-to-field-id queue 1))
     )
   (cm:move-constant-always index i 32)
   (cm:aref32-shared-2L data-field queue.elements index len 32 index-limit)
 )
```

Note that the *queue.elements* array is offset from the queue field by one.

The order of message queueing, including which messages are discarded, is deterministic. As they arrive at the node, messages are enqueued in successive slots of the *queue.elements* array until *queue.elements* is full. Subsequent messages are discarded.