DRAFT: Do not distribute

MIT Concurrent VLSI Architecture Memo 59

Massachusetts Institute of Technology
Artificial Intelligence Laboratory

The MAP Instruction Set Reference Manual v1.55[1]

William J. Dally, Stephen W. Keckler, Nick Carter,
Andrew Chang, Marco Fillo, Whay S. Lee

January 22, 1997

Abstract

This document describes the instruction set of the multi-ALU processor (MAP) of the MIT M-Machine. First the instruction format is detailed, followed by the integer, memory, and floating point unit operations. This document should be read in conjunction with *M-Machine Architecture v1.0* and *M-Machine Microarchitecture v1.0*.

[1]The research described in this paper was supported by the Advanced Research Projects Agency and monitored by the Air Force Electronic Systems Division under contract F19628-92-C-0045.

Contents

1 Overview **6**

2 Operation Fields **7**

3 Summary of Operations **10**

4 Bit Encodings **12**

5 Integer Unit Operations **16**

5.1 Arithmetic Operations 16

5.1.1 ADD 16

5.1.2 ADDU - add unsigned 16

5.1.3 SUB - subtract 16

5.1.4 SUBU - subtract unsigned 17

5.1.5 ASH - arithmetic shift 17

5.1.6 LSH - logical shift 18

5.1.7 ROT - rotate 18

5.1.8 AND - bitwise logical and 18

5.1.9 OR - bitwise logical or 19

5.1.10 XOR - bitwise logical exclusive or 19

5.1.11 NOT - bitwise logical negation 20

5.2 Byte Manipulation 20

5.2.1 EXTB - extract byte 20

5.2.2 EXTH - extract halfword 20

5.2.3 INSB - insert byte 21

5.2.4 INSH - insert halfword 21

5.3 Comparison Operations 22

5.3.1 ILT - integer less than 22

5.3.2 ILE - integer less than or equal 22

5.3.3 ULT - unsigned less than 23

5.3.4 ULE - unsigned less than or equal 23

5.3.5 INE - integer not equal 23

5.3.6 IEQ - integer equal 24

5.3.7 CCAND - condition code logical and . . . 24
5.3.8 CCOR - condition code logical or . . . 25
5.3.9 CCNAND - condition code logical nand . . . 25
5.4 Data Movement . . . 25
5.4.1 MOV - move register . . . 25
5.4.2 EMPTY - unset register presence bits . . . 26
5.4.3 CCEMPTY - unset condition register presence bits . . . 26
5.5 Control Flow Operations . . . 27
5.5.1 BR - branch . . . 27
5.5.2 JMP - jump . . . 27
5.5.3 ILL - illegal instruction . . . 28
5.6 Address Calculation . . . 28
5.6.1 LEA - load effective address . . . 28
5.6.2 LEAB - load effective address from base . . . 29
5.6.3 UNSETPTR - unsets pointer bit . . . 29
5.6.4 SETPTR - set pointer bit . . . 30
5.6.5 ISPTR - test pointer bit . . . 30
5.6.6 ISERR - test for *Errval* . . . 30
5.7 Immediate Operations . . . 31
5.7.1 SHORU - shift & or unsigned 16-bit immediate . . . 31
5.7.2 IMM - create 16-bit immediate . . . 31
5.8 Configuration Space Operations . . . 32
5.8.1 IGTWR - write GTLB . . . 32
5.8.2 IGTRD - read GTLB . . . 32
5.8.3 IGPRB - probe GTLB . . . 33
5.9 Communication Operations . . . 33
5.9.1 ISND0 - send priority 0 message, *user level* . . . 33
5.9.2 ISND0O - send priority 0 message, *user level* . . . 34
5.9.3 ISND0P - send priority 0 message physical . . . 34
5.9.4 ISND0PO - send priority 0 message physical, ordered . . . 34
5.9.5 ISND0PNT - send priority 0 message physical, no throttling . . . 35
5.9.6 ISND0PNTO - send priority 0 message physical, no throttling, ordered . . . 35
5.9.7 ISND1PNT - send priority 1 message physical, no throttling . . . 35

5.9.8 ISND1PNTO - send priority 1 message physical, no throttling ordered 36

6 Memory Unit Operations **37**

6.1 Standard Memory Access 37

6.1.1 LD - load register 37

6.1.2 ST - store integer register 37

6.1.3 FST - store floating point register 38

6.2 Synchronizing Operations 39

6.2.1 LDS - load register and synchronize 39

6.2.2 LDSU - load register and synchronize, unfaulting 39

6.2.3 LDSCND - load register and synchronize, conditional 40

6.2.4 STS - store integer register and synchronize 41

6.2.5 STSU - store integer register and synchronize, unfaulting 41

6.2.6 STSCND - store integer register and synchronize, conditional 42

6.2.7 FSTS - store fp register and synchronize 43

6.2.8 FSTSU - store fp register and synchronize, unfaulting 43

6.2.9 FSTSCND - store fp register and synchronize, conditional 44

6.3 Uncached Operations 44

6.3.1 LUC - load register uncached 44

6.4 Address Calculation 45

6.4.1 LEA - load effective address 45

6.4.2 LEAB - load effective address from base 46

6.5 Control Flow Operations 46

6.5.1 CBAR - cluster barrier 46

6.6 Special Memory Operations 47

6.6.1 SRS - store integer register, replace sync bit 47

6.6.2 FSRS - store fp register, replace sync bit 47

6.6.3 FLNE - flush cache line 48

6.6.4 GETCSTAT - read cache line status 48

6.6.5 PUTCSTAT - write cache line status 49

6.6.6 MBAR - memory barrier 49

6.7 Thread Management Operations 50

6.7.1 HFORK - fork H-Thread 50

6.7.2 HEXIT - terminate H-Thread . . . 50
6.8 Arithmetic Operations . . . 51
6.8.1 ADD - integer add, signed . . . 51
6.8.2 SUB - integer subtract, signed . . . 51
6.8.3 MOV - move register . . . 51
6.8.4 AND - bitwise logical and . . . 52
6.8.5 OR - bitwise logical or . . . 52
6.8.6 XOR - bitwise logical exclusive or . . . 53
6.8.7 NOT - bitwise logical negation . . . 53

7 Floating Point Unit Operations 54

7.1 Notes on FP operands . . . 54
7.2 FP Arithmetic Operations . . . 54
7.2.1 FADD - floating point add . . . 54
7.2.2 FSUB - floating point subtract . . . 55
7.2.3 FMUL - floating point multiply . . . 55
7.2.4 FDIV - floating point divide . . . 56
7.2.5 FMULA - floating point multiply and add . . . 56
7.2.6 FSQRT - floating point square root . . . 57
7.3 Integer Arithmetic Operations . . . 57
7.3.1 IMUL - integer multiply . . . 57
7.3.2 HMUL - multiply, return High bits . . . 57
7.3.3 IDIV - integer divide . . . 58
7.3.4 IDIVU - unsigned integer divide . . . 58
7.4 Data Movement . . . 59
7.4.1 MOV - move register . . . 59
7.4.2 FEMPTY - unset floating point register presence bits . . . 59
7.5 Data Conversion . . . 59
7.5.1 ITOF - integer to floating point conversion . . . 59
7.5.2 FTOI - floating point to signed integer conversion . . . 60
7.6 Comparison Operations . . . 60
7.6.1 FLT - floating point less than . . . 60
7.6.2 FLE - floating point less than or equal . . . 61

7.6.3 FEQ - floating point equal 61
7.6.4 FNE - floating point not equal 61
7.7 Immediate Operations 62
7.7.1 FSHORU - shift & or unsigned 16-bit immediate 62
7.7.2 FIMM - create 16-bit immediate 62
7.8 Communication Operations 63
7.8.1 FSND0 - send priority 0 message, *user level* 63
7.8.2 FSND0O - send priority 0 message, ordered, *user level* 63
7.8.3 FSND0P - send priority 0 message physical 63
7.8.4 FSND0PO - send priority 0 message physical, ordered 64
7.8.5 FSND0PNT - send priority 0 message physical, no throttling 64
7.8.6 FSND0PNTO - send priority 0 message physical, no throttling, ordered 64
7.8.7 FSND1PNT - send priority 1 message, physical, no throttling 65
7.8.8 FSND1PNTO - send priority 1 message physical, no throttling, ordered 65

1 Overview

This document details the M-Machine Multi-ALU Processor's (MAP) instruction set. It is meant to be a manual for the assembly language programmer and used as a specification for assemblers and compilers.

The first section discusses some of the operation fields common to most types of operations. Next, tables listing the operations for each execution unit are presented. Then, the actual bit encodings of the instructions are displayed in three figures. Finally, each operation is presented in detail, including the operation format and side effects. These operations are organized by function unit type: integer, memory, and floating-point.

Definitions: The following terms and definitions are employed throughout this *M-Machine Instruction Set Reference v1.55*.

V-Thread	A system or user process composed of one, two or three H-Threads. Separate V-Threads communicate and synchronize through memory locations.
H-Thread	A sequential stream of instructions composed of a maximum of three parallel operations (one integer, one memory, and one floating point). Separate H-Threads of the same V-Thread may communicate and synchronize through specialized mechanisms, registers and/or memory locations.
Operation	The task for a single execution unit in a cluster.
Instruction	A *long instruction word* (LIW) for a single cluster consisting of three operations - one per execution unit. Any of the three operations may be em NOP's.
M-Machine	The generic name for the processor architecture.
MAP Chip	The name for the multiALU processer chip, which is the first hardware implementation of the M-Machine architecture.
INTU (I)	The integer execution unit.
MEMU (M)	The memory execution unit.
FPU (F)	The floating point execution unit.
Cache Line	Eight 64-bit words.
Block	Same as cache line.
Registers	Each data register is 66 bits wide: 64-bits data, a "pointer" bit, a memory synchronization bit. A unique scoreboard bit exists for each data register. Each condition code register is 1 bit wide. A unique scoreboard bit exists for each condition code register.
Datatypes	All registers hold 64-bit data. All floating point operations are double precision. Condition code values are binary: either boolean "true" or false.
Pointer Bit	The pointer bit is a tag indicating whether the word is of pointer type.
Valid Bit	The valid bit is synonymous with the register presence bit, and indicates if the register contains a value that can be used as a source operand for a subsequent operation.
RSZ bit	The RSZ bit is synonymous with the register sync bit This bit allows the value of the synchronization bit in memory to be replicated in the register file.
Empty	The forcing of the validity bit for a register to be false.

Pack Bits	Meaning
00	Instruction contains only an integer operation
01	Instruction contains only a memory operation
10	Instruction contains only a floating point operation
11	Instruction contains more than one operation

Pack Bits	Meaning
00	Instruction contains I-M combination
01	Instruction contains I-F combination
10	Instruction contains I-M-F combination
11	Instruction contains M-F combination

2 Operation Fields

The opcodes and representations of operations described in this document are intended to be final. The current version of the simulator (v1.3) does not conform to these bit encodings. See version 1.0 of this document for those encodings. The *massl* syntax for assembly language programs is described in *Guide to the M-Machine Assembler and Massl Assembly Language v1.0*. All operations have the following three fields:

- pack: 2 bits used for tight packing of operations
- cond: 2 bits for conditional execution of operation
- cr: 4 bits to specify the condition register used for conditional execution

Operation Packing: While in memory, instructions are packed so that NOPs do not need to be represented. The *pack* bits encode how many and which operations reside in an instruction. The high two bits of each operation is reserved for this encoding. The instruction fetch logic of the MAP chip pipeline detects instruction boundaries. The meanings of the pack bits for the first operation of each instruction is explained in the table below.

If the instruction contains only one operation, then the rest of the instruction is automatically padded out with NOPs by the instruction fetch logic and the IP is incremented by 4 bytes. If the instruction contains more than one operation, the high 2 bits of the next operation must be examined. Those bits encode the following states:

For those instructions that contain only two non-NOP operations, the instruction is padded with an additional NOP before proceeding forward in the pipeline.

Conditional Execution: All operations may be executed based on a condition (*cond*) specified in a condition register (*cr*). These *cond* states are shown in Table 1.

Condition	Encoding	Function
CF	00	conditionally execute if FALSE
CT	01	conditionally execute if TRUE
UA	10	unconditionally execute always
UN	11	unconditionally execute never

Table 1: Condition codes.

src1, src2, src3: integer or fp registers (0–15)
dest: integer or fp register (0–15)
dh: destination H-Thread (0–2), relative addressed
dr: destination register file (INT=0, FP=1)
imm: immediate indicator (for second source: 0 = reg, 1 = imm)
ccdest: condition register destination (0–15)
bank: message composition bank (0,1)
length: message length in words (0–15)
ccsrc1, ccsrc2: condition register source (0–15)
pre: precondition for synchronization (0–3)
post: postcondition for synchronization (0,1)

Table 2: Other fields in operations.

Operation Packing: There is no unique opcode for NOP. Instead, NOP's are created my the compiler and assembler as *add i0 i0 i0*.

Other Fields: Table 2 summarizes the other fields that may reside in an operation.

Note that `[field]` in an operation specification indicates that `field` is optional.

Register Addressing and Encodings: The destination for most operations in the MAP ISA is either a data register or a condition code register.

Data registers are designated by the unique name specified in the "dest" field of an operation. Some operations allow can set the "dh" field indicating the relatively addressed cluster and the "dr" field indicating register file selected (either integer or floating point) in the indicated cluster. Therefore, it is possible for these operations to migrate data without an explict MOV operation.

CC Reg	Encoding			
Local cc0, cc1, cc2, cc3	1100	1101	1110	1111
Global-local cc0, cc1, cc2, cc3	0000	0001	0010	0011
Global-remote (h1.cc0, h1.cc1, h1.cc2, h1.cc3)	0100	0101	0110	0111
Global-remote (h2.cc0, h2.cc1, h2.cc2, h2.cc3)	1000	1001	1010	1011

Table 3: Condition Code Register Encodings

The M-Machine architecture defines three classes of condition code registers: local, global-local, global-remote. The encodings for these operations are detailed in Table 3.

Local condition code registers are private. They can be read, written, and emptied *only* by the parent H-Thread. Each H-Thread has four local condition code registers.

Global-local condition code registers may be read, written, and emptied by the parent H-Thread. Global-local condition code registers for one H-Thread appear as Global-remote condition code registers for the two other H-Threads within the same V-Thread. As such, the two other H-Threads may be read and empty these registers. Each H-Thread has four Global-local condition code registers. Thus each V-Thread possesses twelve (12) global condition code registers. All updates performed by an H-Thread on its Global-local registers are automatically propogated to the two associated H-Threads within the same V-Thread.

Global-remote condition code registers may be read, and emptied by an H-Thread. Each H-Thread has eight (8) Global-remote condition code registers.

All H-Threads use relative addressing to name the Global condition code registers.

Operation Formats: The operation format entry for the operations described below are an indicator as to the decoding required by the hardware. The assembler writer will require this information as well. However, the programmer (user and system) do not need to be concerned with the formats.

Please refer to the *M-Machine Architecture v1.0* for a detailed description of the user visible machine state.

3 Summary of Operations

The complete list of operations supported by the M-Machine architecture is listed in the three following tables. The operations are arranged by execution unit and type. All M-Machine operations are a fixed 32-bits in length. The type of an instruction is based on its exact 32-bit encoding. The M-Machine has several classes of instructions, these are organized into types based on the number/type of operands and the number of bits used to uniquely identify the operation (between 4 and 8). Table 4 lists the operations executed by the integer unit.

Type						
A	add	sub	addu	subu	lea	leab
	and	or	rot	xor	ash	lsh
	insb	insh	extb	exth		
G	empty	ccempty	br	jmp		
P	imm	shoru				
B	unsetptr, setptr	not	mov			
E	ccand	ccor	ccnand			
I	isptr	iserr				
C	ilt	ile	ult	ule	ine	ieq
D	isnd0	isnd0p	isnd0pnt	isnd1p	igtwr	igtrd
	isnd0o	isnd0po	isnd0pnto	isnd1po		igprb
H	ill					

Table 4: Integer Unit Operations (49)

Table 5 lists the operations executed by the memory unit.

Type						
A	ld	luc	st	fst	srs	fsrs
	getcstat	putcstat				
K	ldscnd	ldsu	lds	stscnd	stsu	sts
	fstscnd	fstsu	fsts			
B	not	mov				
Z	add	sub	lea	leab	and	or
	xor	flne				
C	hfork					
H	ill	cbar	mbar		hexit	

Table 5: Memory Unit Operations (32)

Table 6 lists the operations executed by the floating point unit.

Type						
A	fadd	fsub	fmul	fdiv	hmul	
	imul	idiv	idivu			
G	fmula	fsqrt	itof	ftoi	ftoiu	
	mov	fempty				
C	flt	fle	fne	feq		
P	fimm	fshoru				
D	fsnd0	fsnd0p	fsnd0pnt	fsnd1p	fgtwr	fgtrd
	fsnd0o	fsnd0po	fsnd0pnto	fsnd1po		fgprb
H	ill					

Table 6: Floating Point Unit Operations (33)

4 Bit Encodings

The final bit encodings used for the M-Machine instruction set are presented in Figure 1, Figure 2, and Figure 3.

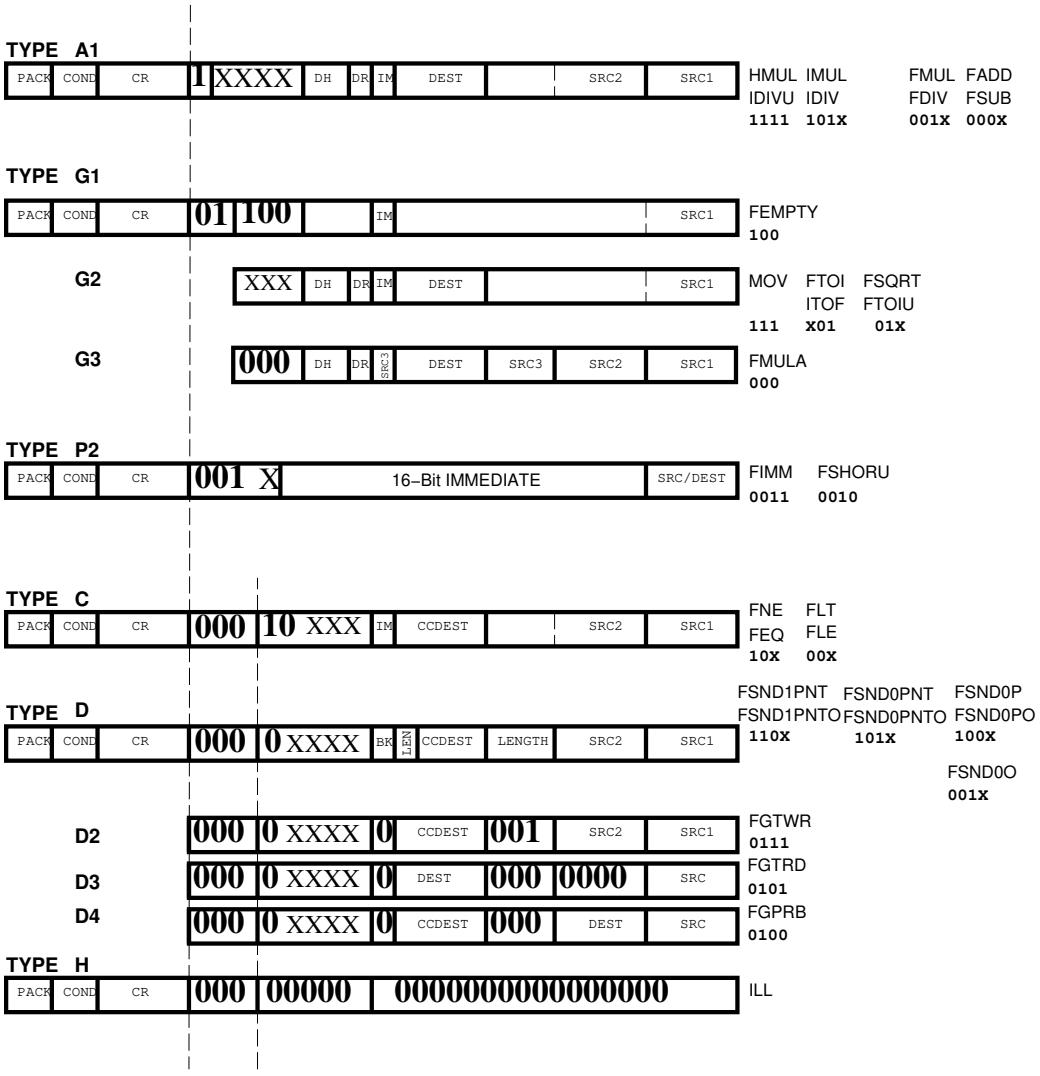

Figure 1: Bit Encodings: Floating Point Unit Operations

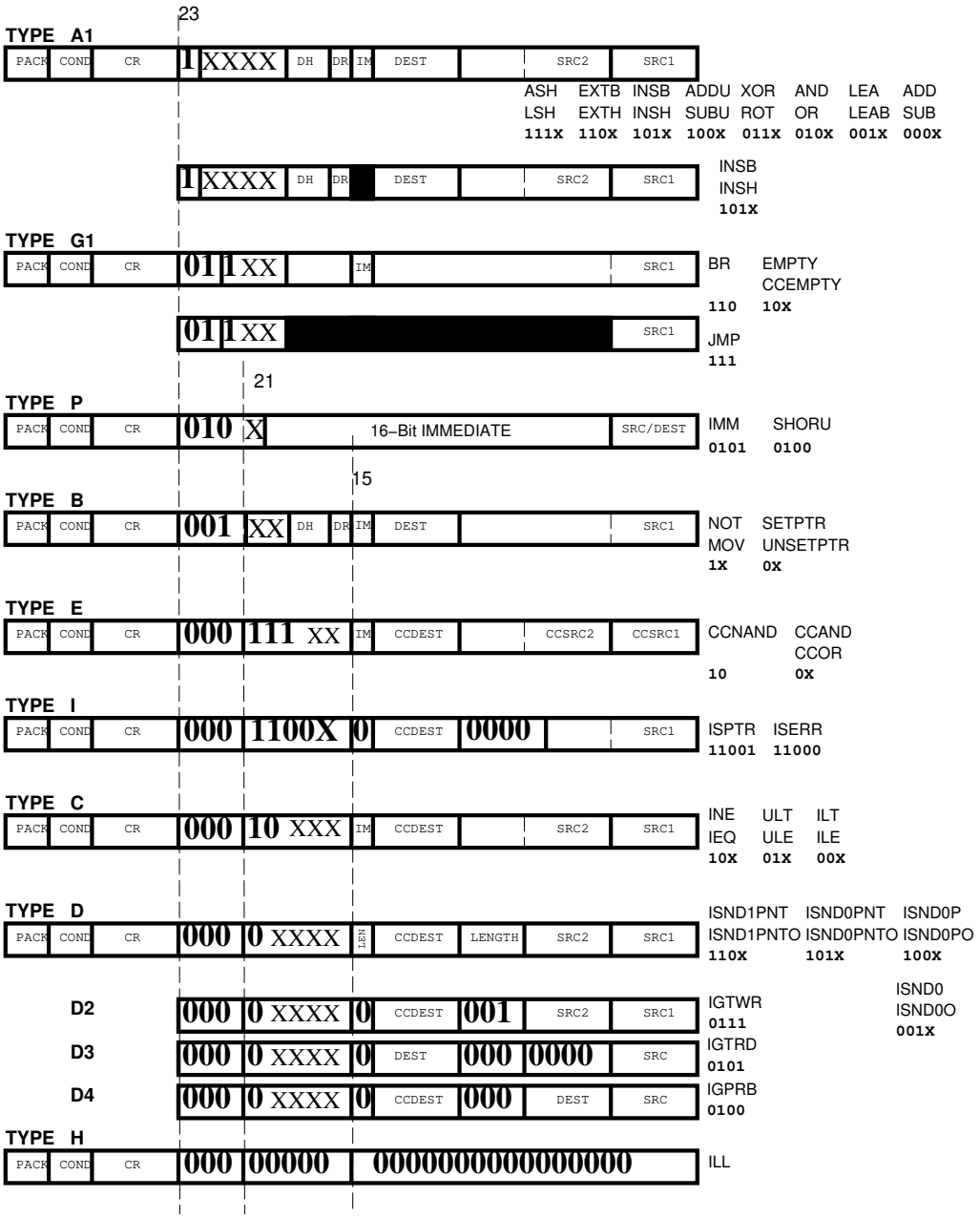

Figure 2: Bit Encodings: Integer Unit Operations

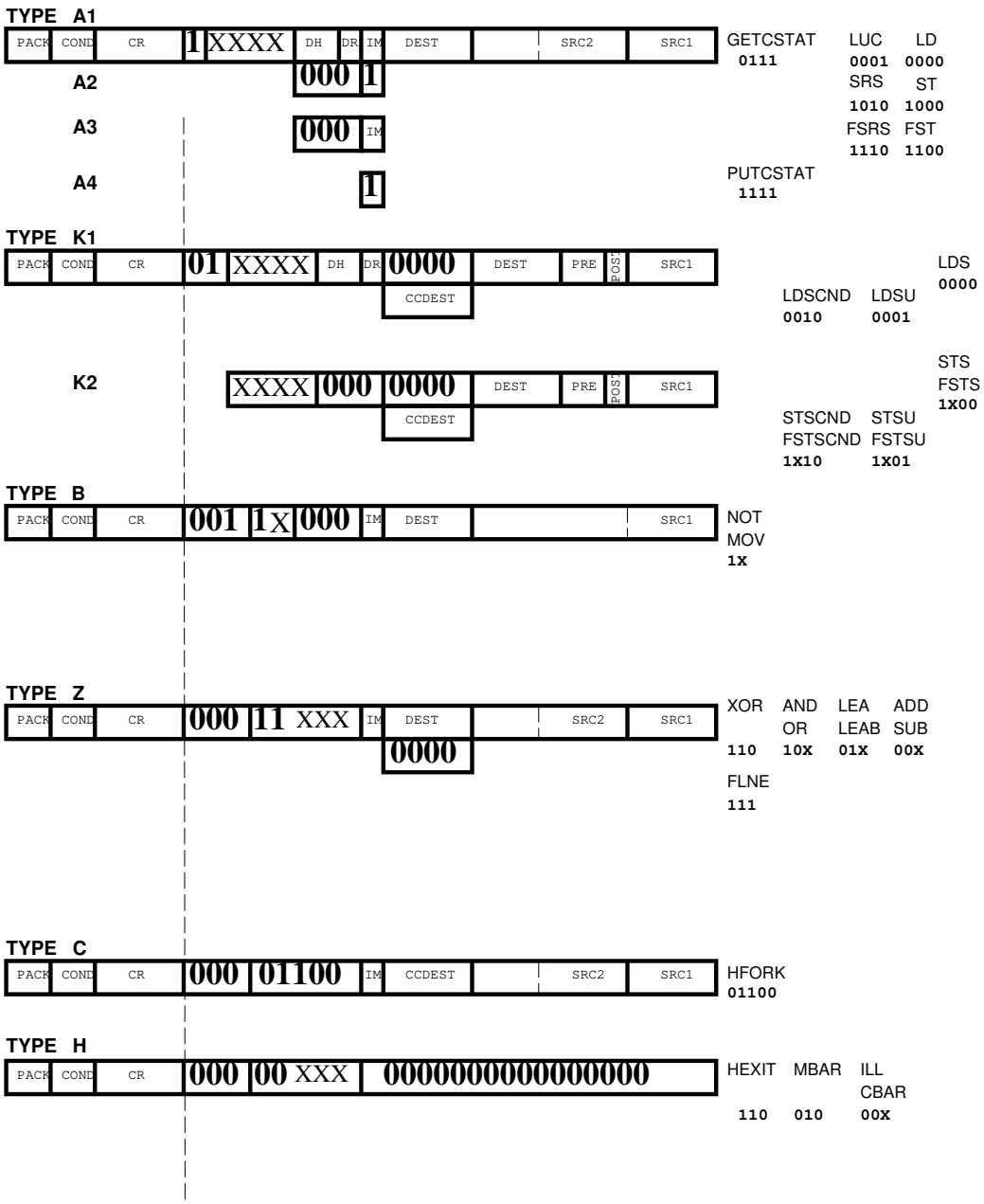

Figure 3: Bit Encodings: Memory Unit Operations

5 Integer Unit Operations

5.1 Arithmetic Operations

5.1.1 ADD

Format: `ialu` [cond cr] `add` src1, src2, dest

pack	cond	cr	1	0000	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Integer signed addition. The value in *src2* can either be a register name or a short immediate.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* + *src2*

5.1.2 ADDU - add unsigned

Format: `ialu` [cond cr] `addu` src1, src2, dest

pack	cond	cr	1	1000	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Integer unsigned addition. The value in *src2* can either be a register name or a short immediate. Returns a value that is congruent mod 2^n of the actual value on overflow.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* + *src2*

5.1.3 SUB - subtract

Format: `ialu` [cond cr] `sub` src1, src2, dest

pack	cond	cr	1	0001	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Integer signed subtraction. The value in *src2* can either be a register name or a short immediate.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* − *src2*

5.1.4 SUBU - subtract unsigned

Format: `ialu` [cond cr] `subu` src1, src2, dest

pack	cond	cr	1	1001	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Integer unsigned subtraction. The value in *src2* can either be a register name or a short immediate. Returns a value that is congruent mod 2^n of the actual value on overflow.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* − *src2*

5.1.5 ASH - arithmetic shift

Format: `ialu` [cond cr] `ash` src1, src2, dest

pack	cond	cr	1	1110	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Arithmetic shift: Direction is determined by the sign of the src2 (or immediate). A positive sign results in a left shift. A negative sign results in a right shift. During right shifts sign extension is performed on the high order bits.

The amount of the shift is determined by the low order 6 bits of src2 (or the immediate).

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← ashift(*src1*, *src2*)

5.1.6 LSH - logical shift

Format: `ialu` [cond cr] `lsh` src1, src2, dest

pack	cond	cr	1	1111	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Logical shift: Direction is determined by the sign of the src2 (or immediate). A positive sign results in a left shift. A negative sign results in a right shift. Zeroes are shifted into the MSB during right shifts and into the LSB during left shifts.

The amount of the shift is determined by the low order 6 bits of src2 (or the immediate).

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← lshift(*src1*, *src2*)

5.1.7 ROT - rotate

Format: `ialu` [cond cr] `rot` src1, src2, dest

pack	cond	cr	1	0111	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Rotate: Direction is determined by the sign of the src2 (or immediate). A positive sign results in a left shift. A negative sign results in a right shift.

The amount of the shift is determined by the low order 6 bits of src2 (or the immediate).

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← rotate(*src1*, *src2*)

5.1.8 AND - bitwise logical and

Format: `ialu` [cond cr] `and` src1, src2, dest

pack	cond	cr	1	0100	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Bitwise logical AND. If one/both of the inputs is/are *pointer* type the output is still the bitwise logical AND of the operands, but the pointer bit is cleared.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* & *src2*

5.1.9 OR - bitwise logical or

Format: `ialu` [cond cr] `or` src1, src2, dest

pack	cond	cr	1	0101	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Bitwise logical OR. If one/both of the inputs is/are *pointer* type the output is still the bitwise logical OR of the operands, but the pointer bit is cleared.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* | *src2*

5.1.10 XOR - bitwise logical exclusive or

Format: `ialu` [cond cr] `xor` src1, src2, dest

pack	cond	cr	1	0110	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Bitwise logical XOR. If one/both of the inputs is/are *pointer* type the output is still the bitwise logical XOR of the operands, but the pointer bit is cleared.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* ⊕ *src2*

5.1.11 NOT - bitwise logical negation

Format: `ialu` [cond cr] `not` src, dest

pack	cond	cr	001	10	dh	dr	imm	dest	src
2	2	4	3	2	2	1	1	4	11

Form Type: B

Description: Bitwise logical NOT. If the input is *pointer* type the output is still the bitwise logical NEGATION, but the pointer bit is cleared.

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← ¬*src*

5.2 Byte Manipulation

5.2.1 EXTB - extract byte

Format: `ialu` [cond cr] `extb` src1, src2, dest

pack	cond	cr	1	1100	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Extracts a byte from *src1* and places it in the low byte of *dest*. The position of the byte is specified by the low 3 bits of *src2*. The rest of the bits in *dest* are zeroed out.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
tmp← *src1* ≫ (*src2* ≪ 3)
dest← tmp & 0xff

5.2.2 EXTH - extract halfword

Format: `ialu` [cond cr] `exth` src1, src2, dest

pack	cond	cr	1	1101	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Extracts a halfword (4 bytes) from *src1* and places it in the low four bytes of *dest*. The position of the halfword is specified by the third least significant bit (bit 2) of *src2*. The rest of the bits in *dest* are zeroed out.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
tmp← *src1* ≫ ((*src2* & 0x4) ≪ 3)
dest← tmp & 0xffffffff

5.2.3 INSB - insert byte

Format: **ialu** [cond cr] **insb** src1, src2, dest

pack	cond	cr	1	1010	dh	dr	unused	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: ANI

Description: Inserts the byte located in the low bits of the register *src2* into a byte location of the word in register *src1*. The byte location is specified in bits 32–34 (a number from 0–7) in register *src2*. Note: *src2* can NOT be an immediate.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
tmp1← ((*src2* ≫ 32) & 0x7) ≪ 3
tmp2← *src1* & ¬(0xff ≪ tmp1)
dest← tmp2 | ((*src2* & 0xff) ≪ tmp1)

5.2.4 INSH - insert halfword

Format: **ialu** [cond cr] **insh** src1, src2, dest

pack	cond	cr	1	1011	dh	dr	unused	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: ANI

Description: Inserts the halfword located in the 32 bits of the register *src2* into a halfword location of the word in register *src1*. The halfword location is specified in bit 34 (0, 1) in register *src2*. Note: *src2* can NOT be an immediate.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
tmp1← ((*src2* ≫ 32) & 0x4) ≪ 3
tmp2← *src1* & ¬(0xffffffff ≪ tmp1)
dest← tmp2 | ((*src2* & 0xffffffff) ≪ tmp1)

5.3 Comparison Operations

5.3.1 ILT - integer less than

Format: `ialu` [cond cr] `ilt` src1, src2, ccdest

pack	cond	cr	000	10000	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the signed integer in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* $<$ *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest$\leftarrow$ *src1* $<$ *src2*

5.3.2 ILE - integer less than or equal

Format: `ialu` [cond cr] `ile` src1, src2, ccdest

pack	cond	cr	000	10001	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the signed integer in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* $\leq$ *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest$\leftarrow$ *src1* $\leq$ *src2*

5.3.3 ULT - unsigned less than

Format: `ialu` [cond cr] `ult` src1, src2, ccdest

pack	cond	cr	000	10010	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the unsigned integer in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* $<$ *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest$\leftarrow$ *src1* $<$ *src2*

5.3.4 ULE - unsigned less than or equal

Format: `ialu` [cond cr] `ule` src1, src2, ccdest

pack	cond	cr	000	10011	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the unsigned integer in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* $\leq$ *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest$\leftarrow$ *src1* $\leq$ *src2*

5.3.5 INE - integer not equal

Format: `ialu` [cond cr] `ieq` src1, src2, ccdest

pack	cond	cr	000	10100	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the signed integer in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* $\neq$ *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest$\leftarrow$ *src1* $\neq$ *src2*

5.3.6 IEQ - integer equal

Format: `ialu` [cond cr] `ieq` src1, src2, ccdest

pack	cond	cr	000	10101	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the signed integer in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* = *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest$\leftarrow$ *src1* == *src2*

5.3.7 CCAND - condition code logical and

Format: `ialu` [cond cr] `ccand` ccsrc1, ccsrc2, ccdest

pack	cond	cr	000	11100	imm	ccdest	ccsrc2	ccsrc1
2	2	4	3	5	1	4	7	4

Form Type: E

Description: Performs the logical "and" of the contents of condition registers *src1* and *src2*. The source operands may be any local or global condition registers. The destination may be a local or a writable global condition register.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
ccdest$\leftarrow$ *ccsrc1* && *ccsrc2*

5.3.8 CCOR - condition code logical or

Format: `ialu` [cond cr] `ccor` ccsrc1, ccsrc2, ccdest

pack	cond	cr	000	11101	imm	ccdest	ccsrc2	ccsrc1
2	2	4	3	5	1	4	7	4

Form Type: E

Description: Performs the logical "or" of the contents of condition registers *src1* and *src2*. The source operands may be any local or global condition registers. The destination may be a local or a writable global condition register.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
ccdest← *ccsrc1* || *ccsrc2*

5.3.9 CCNAND - condition code logical nand

Format: `ialu` [cond cr] `ccnand` ccsrc1, ccsrc2, ccdest

pack	cond	cr	000	11110	imm	ccdest	ccsrc2	ccsrc1
2	2	4	3	5	1	4	7	4

Form Type: E

Description: Performs the logical "nand" of the contents of condition registers *src1* and *src2*. The source operands may be any local or global condition registers. The destination may be a local or a writable global condition register.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
ccdest← ¬(*ccsrc1* && *ccsrc2*)

5.4 Data Movement

5.4.1 MOV - move register

Format: `ialu` [cond cr] `mov` src, dest

pack	cond	cr	001	11	dh	dr	imm	dest	src
2	2	4	3	2	2	1	1	4	11

Form Type: B

Description: Moves value from *src* to *dest*. *Preserves* the RSZ bit.

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src*

5.4.2 EMPTY - unset register presence bits

Format: `ialu` [cond cr] `empty` src

pack	cond	cr	01	100	val	imm	src
2	2	4	2	3	3	1	15

Form Type: G

Description: Sets the register presence bits to empty. The low 16 bits of *src* are a bitmask indicating which registers to empty. A "1" in position i means that register i is to be emptied. The eighteen bits of immediate are split between the *val* field and the *src* field.

Errval/Exceptions:

Generates *no* Errvals.

Demotes any pointer, except Errval, in *src* operand to an integer.

Causes an *EX_EMP* exception if source operand is Errval.

Operation:
```
if(test(cond, cr))
    tmp←src & 0xffff
    for(i=0; i<16; i++)
        if(tmp & 0x1 == 1)
            REG(i).status ← 0
        tmp←tmp ≫ 1
```

5.4.3 CCEMPTY - unset condition register presence bits

Format: `ialu` [cond cr] `ccempty` src

pack	cond	cr	01	101	val	imm	src
2	2	4	2	3	3	1	15

Form Type: G

Description: Sets the condition register presence bits to empty. The low 16 bits of *src* are a bitmask indicating which registers to empty. A "1" in position i means that register i is to be emptied. The local condition registers specified in the low 4 bits, while the writable global condition registers are specified in bits 8 and 9. The eighteen bits of immediate are split between the *val* field and the *src* field. The eighteen bits of immediate are split between the *val* field and the *src* field.

Errval/Exceptions:

Generates *no* Errvals.

Demotes any pointer, except Errval, in *src* operand to an integer.

Causes an *EX_EMP* exception if source operand is Errval.

Operation:
```
if(test(cond, cr))
    tmp←src & 0xffff
    for(i=0; i<4; i++)
        if(tmp & 0x1 == 1)
            CCREG(local, i).status ← 0
        tmp←tmp ≫ 1
    tmp←tmp ≫ 4
```

```
for(i=0; i<2; i++)
    if(tmp & 0x1 == 1)
        CCREG(global, i).status ← 0
    tmp←tmp ≫ 1
```

5.5 Control Flow Operations

5.5.1 BR - branch

Format: `ialu` [cond cr] `br` src

pack	cond	cr	01	110	val	imm	src
2	2	4	2	3	3	1	15

Form Type: G

Description: Control flow change. Branch target is IP relative and the number specified in *src* is the relative number of bytes. Note that it is the programmer's (or assembler's) responsibility to ensure that all branch targets are on halfword boundaries and are to the beginning of a valid instruction. The eighteen bits of immediate are split between the *val* field and the *src* field. The M-Machine architecture has three (3) branch delay slots. Branch or jump operations are not allowed in branch delay slots.

Errval/Exceptions:

Forwards an Errval to the IFU under the following conditions

1. *src* is an Errval
2. Result of increment of IP is out of segment (*EV_BSEG*)

Demotes any pointer, except Errval, in *src* operand to an integer.

Upon discovery of an Errval from a branch instruction, the IFU issues illegal instructions to the pipeline which will eventually cause an exception (after all of the branch delay slots have been issued). No exception is immediately triggered.

Operation:
```
if(test(cond, cr))
    IP← IP + src
```

5.5.2 JMP - jump

Format: `ialu` [cond cr] `jmp` src

pack	cond	cr	01	111	unused	src
2	2	4	2	3	15	4

Form Type: GNI

Description: Control flow change to computed jump target. The jump target is contained in *src*. Note: *src* can NOT be an immediate. The M-Machine architecture has three (3) delay slots after a jump. Branch or jump operations are not allowed in jump delay slots.

Errval/Exceptions:

Forwards an Errval to the IFU under the following conditions

1. *src* is an ErrVal
2. Permission violation (wrong pointer type) on jump target (*EV_JPRM*)
3. Source operand is an integer (*EV_JINT*).

Upon discovery of an Errval from a branch instruction, the IFU issues illegal instructions to the pipeline which will eventually cause an exception (after all of the branch delay slots have been issued). No exception is immediately triggered.

Operation: if(test(*cond*, *cr*))
IP← *src*

5.5.3 ILL - illegal instruction

Format: `ialu` [cond cr] `ill`

pack	cond	cr	000	00000	0
2	2	4	3	5	16

Form Type: H

Description: Causes an illegal instruction exception when executed.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_ILL* exception on excecution.

Operation: if(test(*cond*, *cr*))
Illegal instruction fault

5.6 Address Calculation

5.6.1 LEA - load effective address

Format: `ialu` [cond cr] `lea` src1, src2, dest

pack	cond	cr	1	0010	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Creates a new pointer by adding the integer contents of the register *src2* to the address of the pointer in *src1*, storing the result into register *dest*. A fault is signalled if the operation creates a pointer outside the segment of the original pointer.

Errval/Exceptions:

Propagates an Errval if *src1* or *src2* an Errval.

Generates an Errval if

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer *EV_PERM*.

Demotes any pointer, except Errval, in *src2* to an integer.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* + *src2*

5.6.2 LEAB - load effective address from base

Format: `ialu` [cond cr] `leab` src1, src2, dest

pack	cond	cr	1	0011	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Creates a new pointer by adding the integer contents of the register *src2* to the segment base of the pointer in *src1*, storing the result into register *dest*. A fault is signalled if the operation creates a pointer outside the segment of the original pointer.

Errval/Exceptions:

Propagates an Errval if *src1* or *src2* an Errval.

Generates an Errval if

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer *EV_PERM*.

Demotes any pointer, except Errval, in *src2* to an integer.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← Ptr_base(*src1*) + *src2*

5.6.3 UNSETPTR - unsets pointer bit

Format: `ialu` [cond cr] `setptr` src, dest

pack	cond	cr	001	01	dh	dr	imm	dest	src
2	2	4	3	2	2	1	1	4	11

Form Type: B

Description: Clears the pointer bit of the contents of register *src* and places the result in register *dest*. An UNSETPTR performed on an existing integer results in a pass through.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
tmp← *src1*
tmp.ptr← 0
dest← tmp1

5.6.4 SETPTR - set pointer bit

Format: `ialu` [cond cr] `setptr` src, dest

pack	cond	cr	001	00	dh	dr	imm	dest	src
2	2	4	3	2	2	1	1	4	11

Form Type: B

Description: Sets the pointer bit of the contents of register *src* and places the result in register *dest*. A SETPTR performed on an existing pointer results in a pass through. This operation is privileged and may only be executed in privileged mode.

Errval/Exceptions:

Propagates Errval in SRC.

Generates an Errval if integer bit pattern results in an Errval. Errval type may be unknown.

Causes a *EX_PRIV* exception if IP is not execute-system.

Operation:
if(test(*cond*, *cr*))
tmp← *src1*
tmp.ptr← 1
dest← tmp1

5.6.5 ISPTR - test pointer bit

Format: `ialu` [cond cr] `isptr` src, ccdest

pack	cond	cr	000	11001	0	ccdest	0	src
2	2	4	3	5	1	4	4	7

Form Type: I

Description: Checks the pointer bit of the value in register *src*, placing "1" in *ccdest* if *src* is a pointer, and "0" otherwise. Returns true when src is an Errval.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation:
if(test(*cond*, *cr*))
ccdest← *src*.ptr

5.6.6 ISERR - test for *Errval*

Format: `ialu` [cond cr] `iserr` src, ccdest

pack	cond	cr	000	11000	0	ccdest	0	src
2	2	4	3	5	1	4	4	7

Form Type: I

Description: Checks the pointer bit and permission fields of the value in register *src*, placing "1" in *ccdest* if *src* is an *Errval*, and "0" otherwise.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation:

```
if(test(cond, cr))
    if( (src.permission == Error Value) && (src.ptr) )
      ccdest← 1
    else
      ccdest← 0
```

5.7 Immediate Operations

5.7.1 SHORU - shift & or unsigned 16-bit immediate

Format: `ialu` [cond cr] `shoru` val, dest

pack	cond	cr	001	0	val	dest
2	2	4	3	1	16	4

Form Type: P

Description: Left shift (*Logical*) the value in *dest* 16 bits and "bitwise OR" the 16-bit immediate specificed in *val* into the 16 lowest bits of *dest*. No sign extension is performed on the result.

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation:

```
if(test(cond, cr))
    tmp← left lshift(dest)
    dest← tmp | val
```

5.7.2 IMM - create 16-bit immediate

Format: `ialu` [cond cr] `imm` val, dest

pack	cond	cr	001	1	val	dest
2	2	4	3	1	16	4

Form Type: P

Description: Create the 16-Bit immediate specified by *val* and place into *dest*. Sign extension *is* performed.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation:

```
if(test(cond, cr))
    dest← SEXT(val)
```

5.8 Configuration Space Operations

5.8.1 IGTWR - write GTLB

Format: `ialu` [cond cr] `igtwr`, src1, src2, ccdest

pack	cond	cr	000	00111	0	ccdest	001	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically perform a write of the GTLB. This operation implements a write access to the GTLB via the SEND mechanism employing a one word message. The *src1* operand encodes the index into the GTLB, *src2* encodes the virtual page ID and the base node, and the valid bit. The third required word for the GTLB write is provided in register *i4*. This word encodes the log pages per node, Z-extent, Y-extent, X-extent and page length mask. *This is a priviledged operation.*

The *ccdest* field specifies a cc register which is written with TRUE by the GTLB unit when the operation is complete

- *src1* (index into table):

don't care	GTLB entry index	don't care
	6 bits	13 bits

LSB

- *src2* (GTLB entry - word0):

don't care	valid	base node	virtual page id
	1 bit	16 bits	41 bits

LSB

- *word0* (GTLB entry - word1):

don't care	log pages per node	z extent	y extent1	x extent	page length mask
	6 bits	3 bits	3 bits	3 bits	41 bits

LSB

Errval/Exceptions:

Causes a *EX_PRIV* exception if IP is not execute-system.

5.8.2 IGTRD - read GTLB

Format: `ialu` [cond cr] `igtrd`, src, dest

pack	cond	cr	000	00101	0	dest	0	0	src
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically perform a read transaction on the GTLB. This operation implements a read access to the GTLB via the SEND mechanism employing a zero length message. The *src* operand encodes the virtual address index into the GTLB, *dest* encodes the the return register ID.

The address used is expected to be of the following format:

don't care	GTLB entry index	word 0/1	don't care
	6 bits	1 bit	12 bits

LSB

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

5.8.3 IGPRB - probe GTLB

Format: `ialu` [cond cr] `igprb`, src, dest, ccdest

pack	cond	cr	000	00100	0	ccdest	0	dest	src
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically perform a probe of the GTLB. This operation implements a probe access to the GTLB via the SEND mechanism employing a zero length message. The *src* operand encodes the *virtual address* to probe, *ccdest* encodes the return condition code register. A "1" (true) will be returned for successful probe. A "0" (false) will be returned otherwise. A *igprb* which returns false receives an unspecified value in the *dest* register. If the GTLB entry is found, the physical home node for the virtual address is returned in *dest*.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_GPRB* exception if src is an Errval.

Operation: if(test(*cond*, *cr*))

5.9 Communication Operations

5.9.1 ISND0 - send priority 0 message, *user level*

Format: `ialu` [cond cr] `isnd0`, length, src1, src2, ccdest

pack	cond	cr	000	00010	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically sends a priority 0 message from the message composition registers. The message destination address (a virtual address) is specified in *src1* and a pointer to the code at the remote end is specified in *src2*. The *src1* value is translated into a node identifier automatically by the GTLB. If *src2* is not in the *enter message* state, a fault will occur. The body of the message is specified by the *length* field. The length field is split between *lenmsb* and *len*. The integer register file contains the message body and the *length* (0–9 for user threads, 0–11 for the Event V-Thread) field specifies the number of words in the message body, beginning from register i4. When the message has entered the network, *ccdest* is written with the value TRUE.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_SNDVA* exception if *src1* (ie. Virtual Addr) is an ErrVal.

Causes *EX_SENDIP* exception if *src2* (ie. dest IP) is not *Enter Message* type.

5.9.2 ISND0O - send priority 0 message, *user level*

Format: `ialu` [cond cr] `isnd0`, length, src1, src2, ccdest

pack	cond	cr	000	00011	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: ISND0 with ordered delivery guaranteed.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_SNDVA* exception if *src1* (ie. Virtual Addr) is an ErrVal.

Causes *EX_SENDIP* exception if *src2* (ie. dest IP) is not *Enter Message* type.

5.9.3 ISND0P - send priority 0 message physical

Format: `ialu` [cond cr] `isnd0p`, length, src1, src2, ccdest

pack	cond	cr	000	01000	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically sends a priority 0 message from the message composition registers with no GTLB translation.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

5.9.4 ISND0PO - send priority 0 message physical, ordered

Format: `ialu` [cond cr] `isnd0p`, length, src1, src2, ccdest

pack	cond	cr	000	01001	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: ISND0P with ordered delivery guaranteed.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

5.9.5 ISND0PNT - send priority 0 message physical, no throttling

Format: `ialu` [cond cr] `isnd0pt`, length, src1, src2, ccdest

pack	cond	cr	000	01010	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: ISNDOP with no decrement of the outgoing message counter.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

5.9.6 ISND0PNTO - send priority 0 message physical, no throttling, ordered

Format: `ialu` [cond cr] `isnd0pt`, length, src1, src2, ccdest

pack	cond	cr	000	01011	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: ISND0PNT with ordered message delivery guranteed.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

5.9.7 ISND1PNT - send priority 1 message physical, no throttling

Format: `ialu` [cond cr] `isnd1p`, length, src1, src2, ccdest

pack	cond	cr	000	01100	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically sends a priority 1 message from the message composition registers, with no decrement of the outgoing message counter. and no GTLB translation (physical node id in *src1*).

Note: this is a system level send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

5.9.8 ISND1PNTO - send priority 1 message physical, no throttling ordered

Format: `ialu` [cond cr] `isnd1p`, length, src1, src2, ccdest

pack	cond	cr	000	01101	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: ISND1PNT with ordered message delivery guranteed.

Note: this is a system level send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

6 Memory Unit Operations

6.1 Standard Memory Access

6.1.1 LD - load register

Format: `memu` [cond cr] `ld` src1, [src2,] dest

pack	cond	cr	1	0000	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Load a word from memory into a register and post-increment the pointer. The memory address is a pointer contained in register *src1*, and the result is written to register *dest*. The *src2* field specifies an offset to add to the pointer in *src1* after the memory operation is issued. The post-incremented pointer is stored back in *src1*.

The synchronization bit located in the address indicated by *src1* is copied into the RSZ bit of the register indicated by *dest*.

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* or *src2* is an errval.

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Propagates an Errval into *dest* (via memory system) if *src1* (ie. target address) is an Errval.

Generates an Errval into *dest* (via memory system) if:

1. *src1* (ie. target address) is NOT a read, read/write, physical or config pointer (*EV_LDPERM*).

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← MEM(*src1*)
dest.RSZ← MEM(*src1*).sync
src1← *src1* + *src2*

6.1.2 ST - store integer register

Format: `memu` [cond cr] `st` src1, [src2,] dest

pack	cond	cr	1	1000	0	1	dest	src2	src1
2	2	4	1	4	3	1	4	7	4

Form Type: A

Description: Store a word from an integer register into memory and post-increment the pointer. The memory address is a pointer contained in integer register *dest*, and the source is an integer register specified in *src1*. The *src2* field specifies an immediate offset that is to be added to the pointer. The post-incremented pointer is stored back in *dest*.

The synchronization bit located in the address indicated by *dest* is unchanged.

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* is an errval (*src2* must be an immediate).

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical, or config.

Operation:
if(test(*cond*, *cr*))
MEM(*dest*)← REG(int, *src1*)
dest← *dest* + *src2*

6.1.3 FST - store floating point register

Format: **memu** [cond cr] **fst** src1, [src2,] dest

pack	cond	cr	1	1100	0	imm	dest	src2	src1
2	2	4	1	4	3	1	4	7	4

Form Type: A

Description: Store a word from a floating point register into memory and post-increment the pointer. The memory address is a pointer contained in integer register *dest*, and the source is a floating point register specified in *src1*. The *src2* field specifies an offset that is to be added to the pointer. The post-incremented pointer is stored back in *dest*.

The synchronization bit located in the address indicated by *dest* is unchanged.

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* or *src2* is an errval.

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical, or config.

Operation:
if(test(*cond*, *cr*))
MEM(*dest*)← REG(fp, *src1*)
dest← *dest* + *src2*

6.2 Synchronizing Operations

6.2.1 LDS - load register and synchronize

Format: `memu` [cond cr] `lds` pre, post, src, dest

pack	cond	cr	01	0000	dh	dr	0	dest	pre	post	src
2	2	4	2	4	2	1	4	4	2	1	4

Form Type: K

Description: Load a word from memory into a register and check the synchronization state. The memory address is a pointer contained in register *src*, and the result is written to register *dest*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the load completes and the synchronization bit of the location is set to the postcondition *post*. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. Otherwise, a synchronization fault occurs.

The synchronization bit located in the address indicated by *src1* is copied into the RSZ bit of the register indicated by *dest*.

Errval/Exceptions:

Propagates an Errval into *dest* (via memory system) if *src1* (ie. target address) is an Errval.

Generates an Errval into *dest* (via memory system) if:

1. *src1* (ie. target address) is NOT a read, read/write, physical or config pointer (*EV_LDPERM*).

Causes *no* exceptions.

Operation:

```
if(test(cond, cr))
    tmp ← MEM(src)
    tmp.RSZ ← MEM(src).sync
    if(tmp.sync ∈ SyncStates(pre))
        MEM(src).sync ← post
        dest← tmp
        dest.RSZ← tmp.RSZ
    else
        FAULT(Sync_Failure)
```

6.2.2 LDSU - load register and synchronize, unfaulting

Format: `memu` [cond cr] `ldsu` pre, post, src, dest, ccdest

pack	cond	cr	01	0001	dh	dr	ccdest	dest	pre	post	src
2	2	4	2	4	2	1	4	4	2	1	4

Form Type: K

Description: Load a word from memory into a register and check the synchronization state. The memory address is a pointer contained in register *src*, and the result is written to register *dest*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the load completes, the synchronization bit of the location is set to the postcondition *post*, and TRUE is returned in *ccdest*. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. If the synchronization bit does not match the precondition, or a block status miss occurs, the load is aborted, and FALSE is returned in *ccdest*, a local condition register. LTLB misses occur in the normal manner.

The synchronization bit located in the address indicated by *src1* is copied into the RSZ bit of the register indicated by *dest*.

Errval/Exceptions:

Propagates an Errval into *dest* (via memory system) if *src1* (ie. target address) is an Errval.

Generates an Errval into *dest* (via memory system) if:

1. *src1* (ie. target address) is NOT a read, read/write, physical or config pointer (*EV_LDPERM*).

Causes *no* exceptions.

Operation:

```
if(test(cond, cr))
    tmp ← MEM(src)
    tmp.RSZ ← MEM(src).sync
    if((tmp.sync ∈ SyncStates(pre)) and no fault)
        MEM(src).sync ← post
        dest← tmp
        dest.RSZ← tmp.RSZ
        ccdest← TRUE
    else
        ccdest← FALSE
```

6.2.3 LDSCND - load register and synchronize, conditional

Format: `memu` [cond cr] `ldscnd` pre, post, src, dest, ccdest

pack	cond	cr	01	0010	dh	dr	ccdest	dest	pre	post	src
2	2	4	2	4	2	1	4	4	2	1	4

Form Type: K

Description: Load a word from memory into a register and check the synchronization state. The memory address is a pointer contained in register *src*, and the result is written to register *dest*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the load completes, the synchronization bit of the location is set to the postcondition *post*, and TRUE is returned in *ccdest*. Otherwise, FALSE is returned in *ccdest*, a local condition register. LTLB and block status misses cause the usual faults to occur.

Errval/Exceptions:

Propagates an Errval into *dest* (via memory system) if *src1* (ie. target address) is an Errval.

Generates an Errval into *dest* (via memory system) if:

1. *src1* (ie. target address) is NOT a read, read/write, physical or config pointer (*EV_LDPERM*).

Causes *no* exceptions.

Operation:

```
if(test(cond, cr))
    tmp ← MEM(src)
    tmp.RSZ ← MEM(src).sync
    if(tmp.sync ∈ SyncStates(pre))
        MEM(src).sync ← post
        dest← tmp
        dest.RSZ← tmp.RSZ
        ccdest← TRUE
    else
        ccdest← FALSE
```

6.2.4 STS - store integer register and synchronize

Format: `memu` [cond cr] `sts` pre, post, src, dest

pack	cond	cr	01	1000	0	0	dest	pre	post	src
2	2	4	2	4	3	4	4	2	1	4

Form Type: K

Description: Store a word into memory and check the synchronization state. The memory address is a pointer contained in integer register *dest*, and the source is an integer register specified in *src*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the store completes and the synchronization bit of the location is set to the postcondition *post*. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. Otherwise, a synchronization fault occurs.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical or config.

Operation:

```
if(test(cond, cr))
    tmp ← MEM(src)
    if(tmp.sync ∈ SyncStates(pre))
        MEM(dest)← REG(int, src)
        MEM(dest).sync ← post
    else
        FAULT(Sync_Failure)
```

6.2.5 STSU - store integer register and synchronize, unfaulting

Format: `memu` [cond cr] `stsu` pre, post, src, dest, ccdest

pack	cond	cr	01	1001	0	ccdest	dest	pre	post	src
2	2	4	2	4	3	4	4	2	1	4

Form Type: K

Description: Store a word into memory and check the synchronization state. The memory address is a pointer contained in integer register *dest*, and the source is an integer register specified in *src*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the store completes, the synchronization bit of the location is set to the postcondition *post*, and TRUE is returned in *ccdest*, a local condition register. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. If the synchronization bit does not match the precondition, or a block status miss occurs, the store does not occur, and FALSE is returned in *ccdest*. LTLB miss faults are taken normally.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical or config.

Operation:
```
if(test(cond, cr))
    tmp ← MEM(src)
    if((tmp.sync ∈ SyncStates(pre)) and no fault)
        MEM(dest)← REG(int, src)
        MEM(dest).sync ← post
        ccdest← TRUE
    else
        ccdest ← FALSE
```

6.2.6 STSCND - store integer register and synchronize, conditional

Format: `memu` [cond cr] `stscnd` pre, post, src, dest, ccdest

pack	cond	cr	01	1010	0	ccdest	dest	pre	post	src
2	2	4	2	4	3	4	4	2	1	4

Form Type: K

Description: Store a word into memory and check the synchronization state. The memory address is a pointer contained in integer register *dest*, and the source is an integer register specified in *src*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the store completes, the synchronization bit of the location is set to the postcondition *post*, and TRUE is returned in *ccdest*, a local condition register. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. Otherwise, FALSE is returned in *ccdest*. LTLB and block status misses cause the usual faults to occur.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical or config.

Operation:
```
if(test(cond, cr))
    tmp ← MEM(src)
    if(tmp.sync ∈ SyncStates(pre))
        MEM(dest)← REG(int, src)
        MEM(dest).sync ← post
        ccdest← TRUE
    else
        ccdest← FALSE
```

6.2.7 FSTS - store fp register and synchronize

Format: `memu` [cond cr] `fsts` pre, post, src, dest

pack	cond	cr	01	1100	0	0	dest	pre	post	src
2	2	4	2	4	3	4	4	2	1	4

Form Type: K

Description: Store a word into memory and check the synchronization state. The memory address is a pointer contained in integer register *dest*, and the source is a floating point register specified in *src*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the store completes and the synchronization bit of the location is set to the postcondition *post*. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. Otherwise, a synchronization fault occurs.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical or config.

Operation:

```
if(test(cond, cr))
    tmp ← MEM(src)
    if(tmp.sync ∈ SyncStates(pre))
        MEM(dest)← REG(fp, src)
        MEM(dest).sync ← post
    else
        FAULT(Sync_Failure)
```

6.2.8 FSTSU - store fp register and synchronize, unfaulting

Format: `memu` [cond cr] `fstsu` pre, post, src, dest, ccdest

pack	cond	cr	01	1101	0	ccdest	dest	pre	post	src
2	2	4	2	4	3	4	4	2	1	4

Form Type: K

Description: Store a word into memory and check the synchronization state. The memory address is a pointer contained in integer register *dest*, and the source is a floating point register specified in *src*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the store completes, the synchronization bit of the location is set to the postcondition *post*, and TRUE is returned into *ccdest*. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. If the synchronization bit does not match the precondition, or a block status miss occurs, the store does not complete, and FALSE is returned in *ccdest*. LTLB miss faults occur as usual.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical or config.

Operation:

```
if(test(cond, cr))
    tmp ← MEM(src)
    if((tmp.sync ∈ SyncStates(pre)) and no fault)
        MEM(dest)← REG(fp, src)
        MEM(dest).sync ← post
        CCDEST ← TRUE
    else
        CCDEST ← FALSE
```

6.2.9 FSTSCND - store fp register and synchronize, conditional

Format: `memu` [cond cr] `fstscnd` pre, post, src, dest, ccdest

pack	cond	cr	01	1110	0	ccdest	dest	pre	post	src
2	2	4	2	4	3	4	4	2	1	4

Form Type: K

Description: Store a word into memory and check the synchronization state. The memory address is a pointer contained in integer register *dest*, and the source is a floating point register specified in *src*. The synchronization state is checked in the memory system. If the synchronization bit matches a state specified by the encoding of the precondition *pre*, then the store completes, the synchronization bit of the location is set to the postcondition *post*, and TRUE is returned it *ccdest*. The precondition is the same two bit code used for conditional execution: *CT, CF, UA, UN*. The postcondition is either 1 (true) or 0. Otherwise, the store does not complete, and FALSE is returned in *ccdest*. LTLB and block status misses cause the usual faults to occur.

Errval/Exceptions:

Generates *no* Errvals.

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical or config.

Operation:

```
if(test(cond, cr))
    tmp ← MEM(src)
    if(tmp.sync ∈ SyncStates(pre))
        MEM(dest)← REG(fp, src)
        MEM(dest).sync ← post
        ccdest ← TRUE
    else
        ccdest ← FALSE
```

6.3 Uncached Operations

6.3.1 LUC - load register uncached

Format: `memu` [cond cr] `luc` src1, [src2,] dest

pack	cond	cr	1	0001	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Load a word from memory into a register and post-increment the pointer, but do not load the line into the cache. The memory address is a pointer contained in register *src1*, and the result is written to register *dest*. The *src2* field specifies an offset to add to the pointer in *src1* after the memory operation is issued. The post-incremented pointer is stored back in *src1*.

The synchronization bit located in the address indicated by *src1* is copied into the RSZ bit of the register indicated by *dest*.

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* or *src2* is an errval.

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Propagates an Errval into *dest* (via memory system) if *src1* (ie. target address) is an Errval.

Generates an Errval into *dest* (via memory system) if:

1. *src1* (ie. target address) is NOT a read, read/write, physical or config pointer (*EV_LDPERM*).

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← MEM(*src1*)
dest.RSZ← MEM(*src1*).sync
src1← *src1* + *src2*

6.4 Address Calculation

6.4.1 LEA - load effective address

Format: `memu` [cond cr] `lea` src1, src2, dest

pack	cond	cr	0	11010	imm	dest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description: Creates a new pointer by adding the integer contents of the register *src2* to the address of the pointer in *src1*, storing the result into register *dest*. A fault is signalled if the operation creates a pointer outside the segment of the original pointer.

Errval/Exceptions:

Propagates an Errval if *src1* or *src2* an Errval.

Generates an Errval if

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer *EV_PERM*.

Demotes any pointer, except Errval, in *src2* to an integer.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* + *src2*

6.4.2 LEAB - load effective address from base

Format: `memu` [cond cr] `leab` src1, src2, dest

pack	cond	cr	0	11011	imm	dest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description: Creates a new pointer by adding the integer contents of the register *src2* to the segment base of the pointer in *src1*, storing the result into register *dest*. A fault is signalled if the operation creates a pointer outside the segment of the original pointer.

Errval/Exceptions:

Propagates an Errval if *src1* or *src2* an Errval.

Generates an Errval if

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer *EV_PERM*.

Demotes any pointer, except Errval, in *src2* to an integer.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← Ptr_base(*src1*) + *src2*

6.5 Control Flow Operations

6.5.1 CBAR - cluster barrier

Format: `memu` [cond cr] `cbar`

pack	cond	cr	000	00001	0
2	2	4	3	5	16

Form Type: H

Description: Initiates a cluster barrier. The CBAR instruction will not issue from the SZ stage until both of the other clusters reach their own CBARs.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
Cluster Barrier

6.6 Special Memory Operations

6.6.1 SRS - store integer register, replace sync bit

Format: **memu** [cond cr] **srs** src1, [src2,] dest

pack	cond	cr	1	1010	0	1	dest	src2	src1
2	2	4	1	4	3	1	4	7	4

Form Type: A

Description: Store a word into memory. The memory address is a pointer contained in integer register *dest*, and the source is an integer register specified in *src1*. The RSZ bit of *src1* is stored into memory and replaces the synchronization bit of the address contained in *dest*. The *src2* field specifies an offset that is to be added to the pointer. The post-incremented pointer is stored back in *dest*.

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* is an errval (*src2* must be an immediate).

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical, or config.

Operation:
if(test(*cond*, *cr*))
MEM(*dest*)← REG(int, *src1*)
MEM(*dest*).sync ← REG(int, *src1*).RSZ
dest← *dest* + *src2*

6.6.2 FSRS - store fp register, replace sync bit

Format: **memu** [cond cr] **fsrs** src1, [src2,] dest

pack	cond	cr	1	1110	0	imm	dest	src2	src1
2	2	4	1	4	3	1	4	7	4

Form Type: A

Description: Store a word into memory. The memory address is a pointer contained in the integer register *dest*, and the source is a floating point register specified in *src1*. The RSZ bit of *src1* is stored into memory and replaces the synchronization bit of the address contained in *dest*. The *src2* field specifies an offset that is to be added to the pointer. The post-incremented pointer is stored back in *dest*.

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* is an errval (*src2* must be an immediate).

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Causes an *EX_STPRM* exception if *src1* is NOT: read/write, physical, or config.

Operation:
if(test(*cond*, *cr*))
MEM(*dest*)← REG(fp, *src1*)
MEM(*dest*).sync ← REG(fp, *src1*).RSZ
dest← *dest* + *src2*

6.6.3 FLNE - flush cache line

Format: `memu` [cond cr] `flne` src1[, src2]

pack	cond	cr	000	11111	imm	0	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description: Flush cache of line containing the location specified in integer register *src1*, and increment *src1* by *src2*. If the line is not in the cache, no action is taken.

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* or *src2* is an errval.

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Perform *NOP* if *src1* is NOT read/write, physical, config, read, execute or enter.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
    Flush_Cache_Line(*src1*)
    src1 ← *src1* + *src2*

6.6.4 GETCSTAT - read cache line status

Format: `memu` [cond cr] `getcstat` src1, [src2,] dest

pack	cond	cr	1	0111	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Load cache status bits for the line containing the location specified in integer register *src1* into the low two bits of integer register *dest*. If the line is not in the cache, a cache miss must first be handled. The pointer in *src1* is incremented by *src2*.

Errval/Exceptions:

Errval/Exceptions:

Propagates an Errval into *src1* if *src1* or *src2* is an errval.

Generates an Errval into *src1* if:

1. Result of post-increment is out of segment (*EV_SEG*.
2. *src1* is a Key or Enter pointer (*EV_PERM*).

Propagates an Errval into *dest* (via memory system) if *src1* (ie. target address) is an Errval.

Generates an Errval into *dest* (via memory system) if:

1. *src1* (ie. target address) is a config pointer (*EV_LDPERM*).

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
    dest ← Cache_Status(*src1*)
    src1 ← *src1* + *src2*

6.6.5 PUTCSTAT - write cache line status

Format: `memu` [cond cr] `putcstat` src1, [src2,] dest

pack	cond	cr	1	1111	dh	dr	1	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Performs a FLNE of the cache line specificed by *src1* to synchronize status bits. Return status bits to register specified in *dest*. If specified line is not in cache, return the status bits from the TLB. A TLB miss is possible, this is handled before returning the previous status bit values or performing the update. Store low two bits of *src2* short immediate into the cache status bits for line containing the location specified in integer register *src1*.

This is a privileged operation and may only be executed in supervisor mode.

Errval/Exceptions:

Propagates an Errval into *dest* (via memory system) if *src1* (ie. target address) is an Errval.

Generates an Errval into *dest* (via memory system) if:

1. *src1* (ie. target address) is NOT a read, read/write, physical or config pointer (*EV_LDPERM*).

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
Cache_Status(*dest*) ← *src* & 0x3

6.6.6 MBAR - memory barrier

Format: `memu` [cond cr] `mbar`

pack	cond	cr	000	00010	0
2	2	4	3	5	16

Form Type: H

Description: Stalls until all outstanding load operations have returned.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation: while(MemCounter > 0)
stall

6.7 Thread Management Operations

6.7.1 HFORK - fork H-Thread

Format: **memu** [cond cr] **hfork** src1, src2, ccdest

pack	cond	cr	000	01100	1	ccdest	0	src2	0	src1
2	2	4	3	5	1	4	4	2	1	4

Form Type: C

Description: Forks a thread into a neighboring H-Thread slot in the current V-Thread; *src1* contains the IP for the new thread and *src2* indicates which H-Thread slot to use. Thread slot identifiers are relative to the current H-Thread: *src2* indicates the number of clusters to the right on which the new thread should execute. *src2* must be an immediate and the field width is 2 bits. If a thread is already running there, "0" is returned in *ccdest* and the hfork is aborted. Otherwise, "1" is returned in *ccdest*, which is a local condition register.

Errval/Exceptions:

Generates *NO* Errvals.

Causes an exception if

1. *src1* is NOT an execute, enter, or physical pointer *EX_HFIP*.

Operation:

```
if(test(cond, cr))
    newslot ← (current_H-Thread + src2) % 4
  if(VACANT(newslot) == 1)
    newslot.ip ← src1
    ccdest ← 1
    VACANT(newslot) ← 0
  else
    ccdest ← 0
```

6.7.2 HEXIT - terminate H-Thread

Format: **memu** [cond cr] **hexit**

pack	cond	cr	000	00110	0
2	2	4	3	5	16

Form Type: H

Description: Exits the H-Thread slot by setting the VACANT bit.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation:

```
if(test(cond, cr))
    VACANT(current_slot) ← 0
```

6.8 Arithmetic Operations

These arithmetic operation are meant to enhance the capability of the memory unit without adding any additional hardware. Thus no comparison operations are included and all destinations must be in the integer register file.

6.8.1 ADD - integer add, signed

Format: **memu** [cond cr] **add** src1, src2, dest

pack	cond	cr	0	11000	imm	dest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description: The value in *src2* can either be a register name or a short immediate. Returns a value that is congruent mod 2^n of the actual value on overflow.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* + *src2*

6.8.2 SUB - integer subtract, signed

Format: **memu** [cond cr] **sub** src1, src2, dest

pack	cond	cr	0	11001	imm	dest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description: The value in *src2* can either be a register name or a short immediate. Returns a value that is congruent mod 2^n of the actual value on overflow.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* − *src2*

6.8.3 MOV - move register

Format: **memu** [cond cr] **mov** src, dest

pack	cond	cr	001	11	0	imm	dest	src
2	2	4	3	2	3	1	4	11

Form Type: B

Description:

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src*

6.8.4 AND - bitwise logical and

Format: **memu** [cond cr] **and** src1, src2, dest

pack	cond	cr	0	11100	imm	dest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description:

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* & *src2*

6.8.5 OR - bitwise logical or

Format: **memu** [cond cr] **or** src1, src2, dest

pack	cond	cr	0	11101	imm	dest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description:

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* | *src2*

6.8.6 XOR - bitwise logical exclusive or

Format: `memu` [cond cr] `xor` src1, src2, dest

pack	cond	cr	0	11110	imm	dest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: Z

Description:

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* ⊕ *src2*

6.8.7 NOT - bitwise logical negation

Format: `memu` [cond cr] `not` src, dest

pack	cond	cr	001	10	0	imm	dest	src
2	2	4	3	2	3	1	4	11

Form Type: B

Description:

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← ¬*src*

7 Floating Point Unit Operations

7.1 Notes on FP operands

Symbols used in this section:

- NAN = IEEE Not-A-Number
- INF = IEEE infinity
- ZERO = IEEE zero
- Errval = error value
- NUM = any valid IEEE number (excluding ZERO)

For numerical operations, if an operand has the ptr bit set, and is not an Errval, the ptr bit is unset.

Where possible NANs are propagated.

7.2 FP Arithmetic Operations

7.2.1 FADD - floating point add

Format: `falu` [cond cr] `fadd` src1, src2, dest

pack	cond	cr	1	0000	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description:

$INF + INF = INF$

$INF - INF = NAN$

$INF \pm NUM = INF$

$INF \pm ZERO = INF$

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* + *src2*

7.2.2 FSUB - floating point subtract

Format: `falu` [cond cr] `fsub` src1, src2, dest

pack	cond	cr	1	0001	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description:

$INF + INF = INF$

$INF - INF = NAN$

$INF \pm NUM = INF$

$INF \pm ZERO = INF$

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* - *src2*

7.2.3 FMUL - floating point multiply

Format: `falu` [cond cr] `fmul` src1, src2, dest

pack	cond	cr	1	0010	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description:

$INF \times INF = INF$

$INF \times NUM = INF$

$INF \times ZERO = NAN$

$ZERO \pm NUM = ZERO$

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* × *src2*

7.2.4 FDIV - floating point divide

Format: `falu` [cond cr] `fdiv` src1, src2, dest

pack	cond	cr	1	0011	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description:

$INF \div INF = NAN$

$INF \div ZERO = INF$

$INF \div NUM = INF$

$ZERO \div INF = ZERO$

$ZERO \div ZERO = NAN$

$ZERO \div NUM = ZERO$

$NUM \div INF = ZERO$

$NUM \div ZERO = INF$

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* ÷ *src2*

7.2.5 FMULA - floating point multiply and add

Format: `falu` [cond cr] `fmula` src1, src2, src3, dest

pack	cond	cr	01	000	dh	dr	s3msb	dest	src3	src2	src1
2	2	4	2	3	2	1	1	4	3	4	4

Form Type: G

Description: Multiplies the two floating point numbers in *src1* and *src2* and adds the result to the value in *src3* before storing back into register *dest*. Intermediate rounding is performed between the add and the multiply. NAN, INF, and ZERO behavior is same as composition of FMUL and FADD.

The value of *src3* is split between *s3msb* and *src3*.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2, which has precedence over SRC3.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← (*src1* × *src2*) + *src3*

7.2.6 FSQRT - floating point square root

Format: `falu` [cond cr] `fsqrt` src, dest

pack	cond	cr	01	010	dh	dr	imm	dest	src
2	2	4	2	3	2	1	1	4	11

Form Type: G

Description:

$\sqrt{+INF} = INF$

$\sqrt{-INF} = NAN$

$\sqrt{ZERO} = ZERO$

$\sqrt{-NUM} = NAN$

Errval/Exceptions:

Propagates Errval from source operand.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← $\sqrt{src}$

7.3 Integer Arithmetic Operations

7.3.1 IMUL - integer multiply

Format: `falu` [cond cr] `imul` src1, src2, dest

pack	cond	cr	1	1010	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Calculates the 128 bit product of 2 64 bit integers and returns the low 64 bits.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← (*src1* × *src2*)[63:0]

7.3.2 HMUL - multiply, return High bits

Format: `falu` [cond cr] `hmul` src1, src2, dest

pack	cond	cr	1	1110	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Calculates the 128 bit product of 2 64 bit integers and returns the high 64 bits.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← (*src1* × *src2*)[127:64]

7.3.3 IDIV - integer divide

Format: **falu** [cond cr] **idiv** src1, src2, dest

pack	cond	cr	1	1011	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Performs signed integer divide with rounding.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Generates *EX_IDIV0* Errval if SRC2 = 0.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* ÷ *src2*

7.3.4 IDIVU - unsigned integer divide

Format: **falu** [cond cr] **idivu** src1, src2, dest

pack	cond	cr	1	1111	dh	dr	imm	dest	src2	src1
2	2	4	1	4	2	1	1	4	7	4

Form Type: A

Description: Performs unsigned integer divide with rounding.

Errval/Exceptions:

Propagates Errval from source operands. SRC1 errval has precedence over SRC2.

Generates *EX_IDIV0* Errval if SRC2 = 0.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← *src1* ÷ *src2*

7.4 Data Movement

7.4.1 MOV - move register

Format: `falu` [cond cr] `mov` src, dest

pack	cond	cr	01	111	dh	dr	imm	dest	src
2	2	4	2	3	2	1	1	4	11

Form Type: G

Description: Moves value from *src* to *dest*. *Preserves* the RSZ bit.

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation:
```
if(test(cond, cr))
    dest← src
```

7.4.2 FEMPTY - unset floating point register presence bits

Format: `falu` [cond cr] `fempty` src

pack	cond	cr	01	100	val	imm	src
2	2	4	2	3	3	1	15

Form Type: G

Description: Sets the floating point register presence bits to empty. The low 16 bits of *src* are a bitmask indicating which floating point registers to empty. A 1 in position i means that register i is to be emptied.

Errval/Exceptions:

Generates *no* Errvals.

Demotes any pointer, except Errval, in *src* operand to an integer.

Causes an *EX_EMP* exception if source operand is Errval.

Operation:
```
if(test(cond, cr))
    tmp←src & 0xffff
    for(i=0; i<16; i++)
        if(tmp & 0x1 == 1)
            REG(i).status ← 0
        tmp←tmp ≫ 1
```

7.5 Data Conversion

7.5.1 ITOF - integer to floating point conversion

Format: `falu` [cond cr] `itof` src, dest

pack	cond	cr	01	101	dh	dr	imm	dest	src
2	2	4	2	3	2	1	1	4	11

Form Type: G

Description: Converts the integer in *src* to IEEE double precision format, storing the result in register *dest*.

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← (double)*src*

7.5.2 FTOI - floating point to signed integer conversion

Format: `falu` [cond cr] `ftoi` src, dest

pack	cond	cr	01	001	dh	dr	imm	dest	src
2	2	4	2	3	2	1	1	4	11

Form Type: G

Description: Converts the floating point number in *src* to a two's complement integer stored in register *dest*.

Errval/Exceptions:

Propagates Errval from source operand.

NAN, INF, or out of range floating point op generates *EV_FTOI* Errval.

Maximum floating point number = 0x43dfffffffb00000

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← (int)*src*

7.6 Comparison Operations

7.6.1 FLT - floating point less than

Format: `falu` [cond cr] `flt` src1, src2, ccdest

pack	cond	cr	000	10000	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the floating point number in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* < *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest← *src1* < *src2*

7.6.2 FLE - floating point less than or equal

Format: `falu` [cond cr] `fle` src1, src2, ccdest

pack	cond	cr	000	10001	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the floating point number in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* <= *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest← *src1* <= *src2*

7.6.3 FEQ - floating point equal

Format: `falu` [cond cr] `feq` src1, src2, ccdest

pack	cond	cr	000	10100	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the floating point number in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* = *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest← *src1* == *src2*

7.6.4 FNE - floating point not equal

Format: `falu` [cond cr] `fne` src1, src2, ccdest

pack	cond	cr	000	10101	imm	ccdest	src2	src1
2	2	4	3	5	1	4	7	4

Form Type: C

Description: Compares the floating point number in register *src1* with that in *src2*, placing a "1" (TRUE) in the destination condition register if *src1* $\neq$ *src2* and placing a "0" (FALSE) there otherwise. The destination may be a local condition register or one of the writable global condition registers.

Errval/Exceptions:

Generates *no* Errvals.

Demotes pointers in either source operands into integers.

Causes an *EX_CMP* exception if either source operand is an Errval.

Operation: if(test(*cond*, *cr*))
ccdest← *src1* $\neq$ *src2*

7.7 Immediate Operations

7.7.1 FSHORU - shift & or unsigned 16-bit immediate

Format: `falu` [cond cr] `fshoru` val, dest

pack	cond	cr	001	0	val	dest
2	2	4	3	1	16	4

Form Type: P

Description: Left shift (*Logical*) the value in *dest* 16 bits and "bitwise OR" the 16-bit immediate specificed in *val* into the 16 lowest bits of *dest*. No sign extension is performed on the result.

Errval/Exceptions:

Propagates an Errval if the source operand is an Errval.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
tmp← left lshift(*dest*)
dest← *tmp* | *val*

7.7.2 FIMM - create 16-bit immediate

Format: `falu` [cond cr] `fimm` val, dest

pack	cond	cr	001	1	val	dest
2	2	4	3	1	16	4

Form Type: P

Description: Create the 16-Bit immediate specified by *val* and place into *dest*. Sign extension *is* performed.

Errval/Exceptions:

Generates *no* Errvals.

Causes *no* exceptions.

Operation: if(test(*cond*, *cr*))
dest← SEXT(*val*)

7.8 Communication Operations

7.8.1 FSND0 - send priority 0 message, *user level*

Format: `falu` [cond cr] `fsnd0`, length, src1, src2, ccdest

pack	cond	cr	000	00010	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically sends a priority 0 message from the message composition registers. The message destination address (a virtual address) is specified in *src1* and a pointer to the code at the remote end is specified in *src2*. The *src1* value is translated into a node identifier automatically by the GTLB. If *src2* is not in the *enter message* state, a fault will occur. The body of the message is specified by the *length* field. The length field is split between *lenmsb* and *len*. The integer register file contains the message body and the *length* (0–9 for user threads, 0–11 for the Event V-Thread) field specifies the number of words in the message body, beginning from register i4. When the message has entered the network, *ccdest* is written with the value TRUE.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_SNDVA* exception if *src1* (ie. Virtual Addr) is an ErrVal.

Causes *EX_SENDIP* exception if *src2* (ie. dest IP) is not *Enter Message* type.

7.8.2 FSND0O - send priority 0 message, ordered, *user level*

Format: `falu` [cond cr] `fsnd0`, length, src1, src2, ccdest

pack	cond	cr	000	00011	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: FSND0 with ordered delivery guaranteed.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_SNDVA* exception if *src1* (ie. Virtual Addr) is an ErrVal.

Causes *EX_SENDIP* exception if *src2* (ie. dest IP) is not *Enter Message* type.

7.8.3 FSND0P - send priority 0 message physical

Format: `falu` [cond cr] `fsnd0p`, length, src1, src2, ccdest

pack	cond	cr	000	01000	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically sends a priority 0 message from the message composition registers with no GTLB translation.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

7.8.4 FSND0PO - send priority 0 message physical, ordered

Format: `falu` [cond cr] `fsnd0p`, length, src1, src2, ccdest

pack	cond	cr	000	01001	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: FSND0P with ordered delivery guaranteed.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

7.8.5 FSND0PNT - send priority 0 message physical, no throttling

Format: `ialu` [cond cr] `fsnd0pt`, length, src1, src2, ccdest

pack	cond	cr	000	01010	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: FSNDOP with no decrement of the outgoing message counter.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

7.8.6 FSND0PNTO - send priority 0 message physical, no throttling, ordered

Format: `falu` [cond cr] `fsnd0pt`, length, src1, src2, ccdest

pack	cond	cr	000	01011	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: FSND0PNT with ordered message delivery guranteed.

Note: this is a privileged send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

7.8.7 FSND1PNT - send priority 1 message, physical, no throttling

Format: `falu` [cond cr] `fsnd1p`, length, src1, src2, ccdest

pack	cond	cr	000	01100	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: Atomically sends a priority 1 message from the message composition registers, with no decrement of the outgoing message counter. and no GTLB translation (physical node id in *src1*).

Note: this is a system level send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.

7.8.8 FSND1PNTO - send priority 1 message physical, no throttling, ordered

Format: `falu` [cond cr] `fsnd1p`, length, src1, src2, ccdest

pack	cond	cr	000	01101	lenmsb	ccdest	len	src2	src1
2	2	4	3	5	1	4	3	4	4

Form Type: D

Description: ISND1PNT with ordered message delivery guranteed.

Note: this is a system level send operation.

Errval/Exceptions:

Generates *no* Errvals.

Causes *EX_PRIV* exception if IP is not execute-system.