Antares Rev. C

Instruction Set Reference Manual

1/15/1987

Copy Assigned to _____

Table of Contents

List of Tables.....iii

Introduction

Programming Model.....1
Special Registers.....3
Traps and Interrupts.....9
Condition Codes.....14
Notation.....16
Quick-Reference.....18

Instructions

Add with Carry Partial (AdCP).....24
Add (Add).....25
Add Immediate (Add).....26
Add with Carry (AddC).....27
Add Partial (AddP).....28
Add Program Counter (AdPC).....29
And (And).....30
And Complement (AndC).....31
Branch on Condition (Bcc).....32
Create Data Cache Line (CDC).....34
Clear Field (ClrF).....35
Clear Mode (ClrM).....36
Count Leading Zeroes (CLZ).....38
Compare (Cmp).....39
Compare Immediate (Cmp).....40
Compare Partial (CmpP).....41
Deposit (Dep).....42
Divide (Div).....44
Double Shift (Dsh).....46
Extract Signed (ExtS).....48

Extract Unsigned (ExtU).....50
Flush Data Cache Line (FDC).....52
Invalidate Data Cache Line (IDC).....53
Invalidate Instruction Cache Line (IIC).....54
Invalidate Instruction Cache (IICa).....55
Insert (Ins).....56
Interrupt (Int).....58
Invalidate Translation Lookaside Buffer (ITLB).....59
Jump Relative (Jmp).....60
Jump Absolute (Jmp).....61
Jump and Link (JmpL).....62
Load Condition (Lcc).....63
Load Immediate (Ld).....65
Load Byte (LdB).....66
Load Carry Partial (LdCP).....67
Load Multiple (LdM).....68
Load Word (LdW).....69
Load Word Direct (LdW).....70
Load Word (LdW).....72
Load Word Extended (LdWE).....73
Move (Mov).....74
Move to Special (Mov).....75
Move from Special (Mov).....77
Mask Generate (Msk).....79
Mask Generate (Msk).....80
Multiply (Mul).....81
Multiply Partial (MulP).....82
Negate (Neg).....83
Not (Not).....84
Or (Or).....85
Prefetch Data Cache Line (PDC).....86
Prefetch Instruction Cache Line (PIC).....87
Prefetch Instruction Cache Line (PIC).....88
Read Data Tag by Index (RDTX).....89
Restart (Res).....90
Resume (Rsm).....91
Return from Interrupt (RtI).....92
Subtract with CArry Partial (SbCP).....93
Send (Send).....94
Set Field (SetF).....95
Set Mode (SetM).....96

Shift Left (ShL).....98
Shift Right (ShR).....99
Skip on Condition (Skcc).....100
Store Byte (StB).....102
Store Multiple (StM).....103
Start (Strt).....104
Store Word (StW).....105
Store Word Direct (StW).....106
Store Word (StW).....107
Store Word Extended (StWE).....108
Subtract (Sub).....109
Subtract Immediate (Sub).....110
Subtract with Carry (SubC).....111
Subtract Partial (SubP).....112
Trap (Trap).....113
Test Field (TstF).....114
Test Mode (TstM).....115
Update Data Cache Line (UDC).....117
Validate Data Cache Line (VDC).....118
Wait (Wait).....119
Exclusive Or (XOr).....120

List of Tables

Table 1. Trap and Interrupt Enables.....11
Table 2. Traps and Interrupts.....12
Table 3. Operands.....16
Table 4. Operators and Symbols.....17
Table 5. Alphabetic Summary of Instructions.....18
Table 6. Functional Summary of Instructions.....21
Table 7. CC Field Encodings (Bcc).....32
Table 8. CC Field Encodings (Lcc).....63
Table 9. CC Field Encodings (Skcc).....100

Introduction

Antares is a new 32-bit CPU being developed by the Advanced Computer Development group. The CPU features a unique design, containing four individual processing units, or PUs, and on-board instruction and data caches. The Antares chip uses a compact instruction set: operations performed the most frequently can be executed with a single instruction, while less frequently used operations are synthesized by instruction sequences.

This manual describes the Antares instruction set. This introduction describes the Antares programming model and defines the special registers used by the chip. This section also covers the trap/interrupt mechanism and lists the traps and interrupts, as well as describing the notation used in the manual. Quick-reference tables list the instructions functionally and alphabetically.

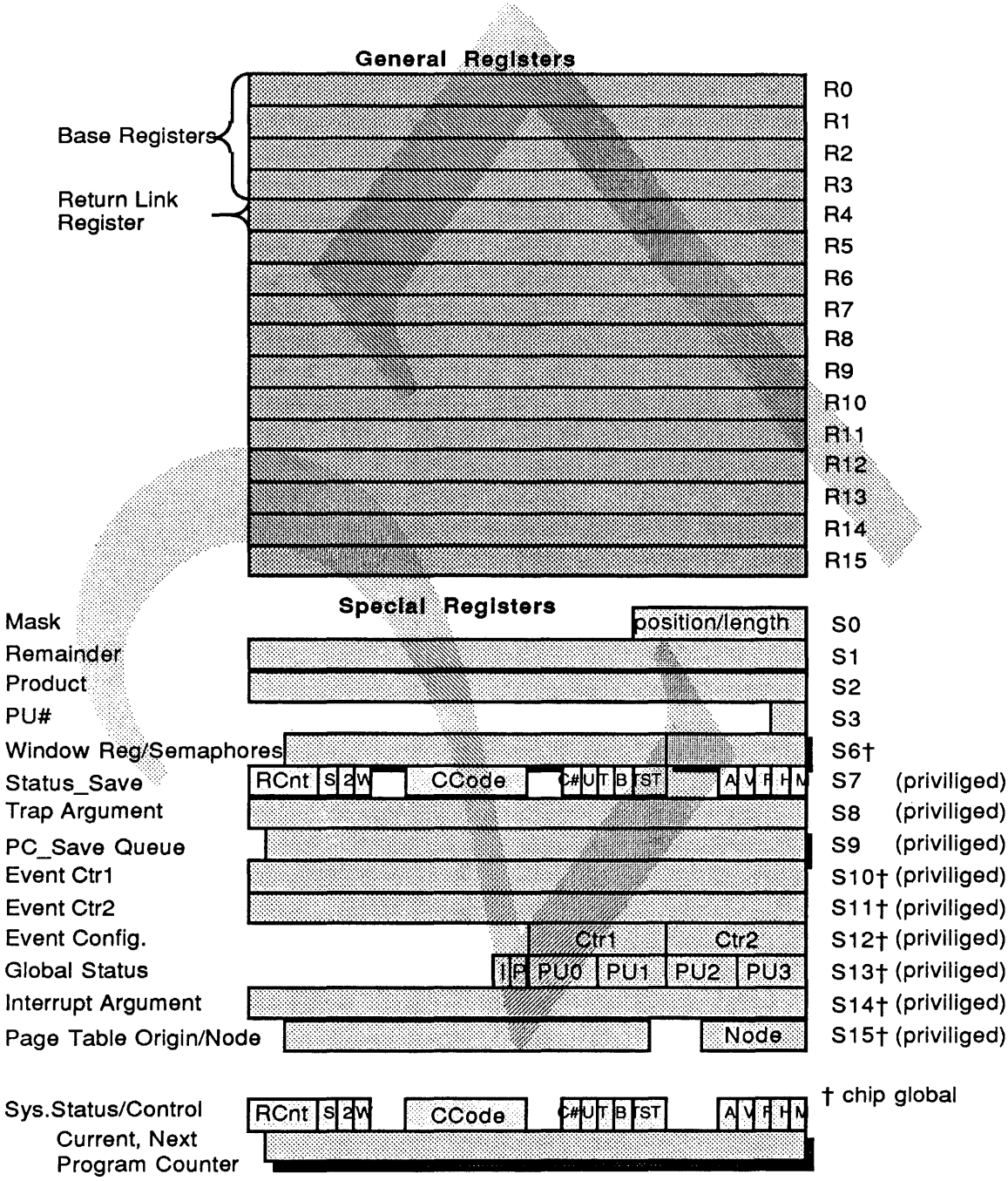
Following the introduction, the manual presents each instruction separately. Each description includes an operation statement, a description of the instruction's operation, a list of affected condition codes, exceptions, restrictions, timing, and format. Where appropriate examples are included.

A companion volume, *The Antares CPU: An Overview*, contains information about Antares architecture.

Programming Model

The drawing on the following page provides a programming model for the Antares CPU. It shows the general and special registers used by the chip. Of the 16 general registers, the first four (R0 through R3) are used as base registers for most instructions; however, all 16 general registers can be used for the extended instructions (LdWE and StWE). The fifth register (R4) serves as the return link register (used by the JmpL instruction).

Programming Model



Special Registers

The Antares CPU uses a number of special registers to execute its instructions. Registers S0, S1, S2, S3, S7, S8, and S9 apply to a specific PU, while S4, S6, S10, S11, S12, S13, S14, and S15 are chip global registers used by all four PUs. Registers S4 and S7 through S15 are privileged. Each special register has a specific purpose, as defined below.

Mask (S0)

S0 holds the definition of a field defined by the Mask Generate (Msk) instructions. The Mask register contains 15 bits in this implementation. The Mask and Move instructions set the Mask register.

Remainder (S1)

S1, used by the Divide (Div) instruction, fulfills two purposes: it allows formation of 64-bit dividends and stores the remainder from a division operation. When the MDfull bit in the System Status/Control register is set, the Divide (Div) instruction uses the contents of the Remainder register as the upper 32 bits of the dividend. The Remainder register contains 32 bits and is filled as a result of a divide operation or by a Move to Special (Mov) instruction.

Product (S2)

Used by the Multiply (Mul) and Multiply Paritial (MulP) instructions, the Product register allows formation of 64-bit products. When two 32-bit numbers are multiplied, the upper 32 bits of the product get stored in the Product register. In addition, if the MDfull bit in the System Status/Control register is set, a Mul or MulP instruction adds the contents of the Product register to the product. The Product register contains 32 bits and is filled as a result of a multiplication or it can be filled by a Move to Special (Mov) instruction.

PU# (S3)

S3 holds the PU# for the PU with which the register is associated. This 2-bit register is a read-only register.

Test (S4)

S4 is used for system diagnostics. It is a global, privileged register.

Window/Semaphores (S6)

The two S6 registers contain two parts: a window register and semaphores. Which of the two S6 registers used depends on the setting of the Cluster# bit in the System Status/Control register.

The 20-bit window registers are used by the Load Word (LdW) and Store Word (StW) Direct instructions to determine the address of a Memory word. One Window register is a read-only register that holds the constant 0x3FF00000. The other Window register can be modified using the Move to Special (Mov) instruction when the Cluster# bit in the System Status/Control register equals 1.

The Semaphore registers facilitate inter-PU communications. They are set by executing a Store Word (StW) Direct to addresses 0 through 7. Load Word (LdW) Direct from addresses 0 through 7 clears the Semaphore register. They are read and written along with the Window registers.

Status_Save (S7)

When traps are disabled, the PU's status is saved from the System Status Control register to the Status_Save register on each clock cycle. Enabling traps invalidates the contents of the Status_Save register. S7 is a privileged register.

Trap Argument (S8)

S8 holds the data address that caused the trap. Its contents can be read using the Mov from Special (Mov) instruction and are not valid unless traps are disabled. S8 is a privileged register.

PC_Save Queue (S9)

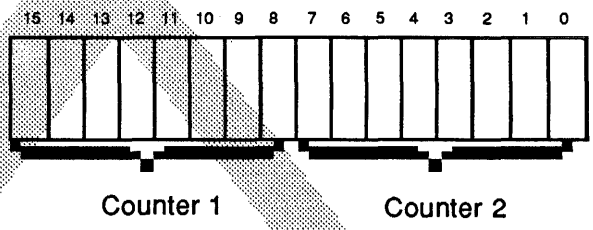
Saves the PC and next PC when the processor executes a Trap instruction. Reading S9 returns PC; writing to S9 moves Next PC to PC and writes into Next PC. The privileged register's contents are valid only when traps are disabled. S9 contains 30 bits.

Event Counter 1 (S10)
Event Counter 2 (S11)

S10 and S11 count the events specified by the Event Counter Configuration register (S12). The Event Counter registers aid performance analysis by counting specific events. If enabled, an interrupt occurs when the counter overflows. The two 32-bit privileged registers can be read using a Move from Special (Mov) instruction.

Event Configuration (S12)

Specifies which events will be counted into the S10 and S11 registers. S12 contains 16 bits, eight of which are allocated to each Event Counter register, as shown below.



15	7	Interrupt enable. If set, it allows an interrupt when the Event Counter overflows.
14	6	If set, indicates that the Event Counter has overflowed. Cannot be written with a 1; always resets if written.
13	5	If set, counts system mode events.
12	4	If set, counts user mode events.
11	3	If set, indicates that the next three bits (0..2 and 8..10) encode a Memory Mangement Unit (MMU) event to be counted:

10	2	Data out. If set, counts moveouts from the data cache.
9	1	Data in. If set, counts moveins to the data cache.
8	0	Instructions in. If set, counts moveins to the instruction cache. If bits 8..10 and/or 0..2 are all clear, counts TLB misses.

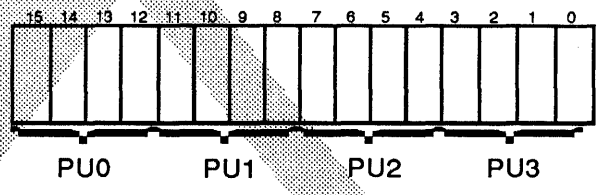
11	3	If clear, indicates that the next three bits encode a Processing Unit (PU) event to be counted:

10	2	If set, counts each cycle that the selected PU is not halted. If clear, counts each instruction that the selected PU finishes.
8,9	0,1	Selects PU (2 bits).

If the System and User bits are both clear, the Event Counter counts all cycles if the MMU/PU bit is set. If the MMU/PU bit is clear, the Event Counter does not count any cycles.

Global Status (S13)

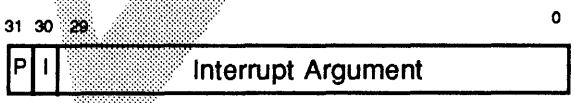
S13 facilitates deadlock detection. The Global Status register contains 16 bits, four for each PU. The four bits for each PU enable the PUs to determine the state of other PUs according to how the bits are set. S13 is a read-only, privileged register. The diagram below shows the bits used.



0, 4, 8, 12	User. If this bit is set for a particular PU, it indicates that the PU is in user mode.
1, 5, 9, 13	Trap enabled. If this bit is set, the PU is enabled for traps.
2, 6, 10, 14	Semaphore wait. If this bit is set, the PU is waiting for a semaphore to be modified.
3, 7, 11, 15	Halted. If this bit is set, the PU is halted because it has executed a Wait instruction.

Interrupt Argument (S14)

S14 contains the address that caused the IPB interrupt as well as the external interrupt pending and external interrupt enable bits. S14 is a chip global, privileged register that can be read using the Move from Special (Mov) instruction. Only the external interrupt pending and enable bits can be written with the Move to Special (Mov) instruction. The external interrupt enable bit can be set or cleared; the interrupt pending bit can be cleared only. The contents of bits 0 through 29 are valid only when interrupts are disabled.



30	External interrupt pending. If this bit is set, it indicates that an external interrupt is waiting for processing.
31	External interrupt enable. If this bit is set, it indicates that the CPU can be interrupted.

Page Table Origin/Node (S15) Bits 8 through 30 contain the page address of the base of the page table, which the TLB miss logic uses. Bits 0 through 5 contain the PU node number used by the IPB logic. The powerup logic normally sets the node number.

Current, Next Program Counters

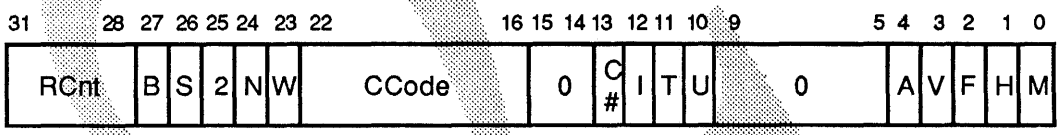
The Current and Next Program Counters hold the addresses of the currently executing instruction and the next instruction.

System Status/Control

The System Status/Control register holds the status and control information that the processor uses. The register contains the 10 mode bits and the condition codes, as well as other status information.

The 10 mode bits are set and cleared by the Set Mode (SetM) and Clear Mode (ClrM) instructions. Bits 0 through 7 can be set in user or system mode, bits 8 through 15 are restricted to system mode only. Both interrupts and traps clear bits 0 through 15.

The drawing below shows the contents of the System Status/Control register.



Bits	Field	Meaning
0	M	MD control
1	H	Halfword/byte
2	F	MD full
3	V	Overflow trap enable
4	A	Registers available
5-7		Reserved
8-9		Reserved
10	B	Branch taken trap enable
11	T	Trap enable
12	U	User/system
13	C#	Cluster number
14-15		Reserved

Bits	Field	Meaning
16-22	CCode	Condition codes
16	Z	Zero
17	N	Negative
18	V	Overflow
19	C0	Carry
20	C1	Carry
21	C2	Carry
22	C3	Carry
23-23		Reserved
25	W	Halted
26	2	Second half
27	S	Skipped
28-31	RCnt	Register count

MD control (M) will be defined later.

Halfword/byte (H) indicates whether the partial word instructions (AddP, AdCP, SubP, SbCP, CmpP, and MulP) execute in halfword or byte mode.

MD full (F) indicates whether or not the MD special register contains data to be used by a Multiply or Divide instruction. The MD full bit is cleared by Multiply and Divide instructions.

Overflow trap enable (V) allows a trap to occur when the V condition code bit is set by an Add, Add Immediate, Add with Carry, Negate, Subtract, Subtract Immediate, or Subtract with Carry instruction. (Divide--maybe.)

Registers available (A) is used by the interrupt/trap handler to indicate which PU was idle when trapped or interrupted. Executing a Resume or Start instruction clears the A bit for the target PU.

Branch taken trap enable (B) can be modified in system mode only. If set, it enables a trap when a branch is taken; clearing the bit disables traps on a taken branch. If set when a branch is taken, the processor branches and then traps before the shadow of the branch.

Trap Enable/Disable (T) can be modified in system mode only to enable traps. It is cleared to disable traps.

User/System (U) can be modified in system mode only. It is set to indicate user mode and cleared to indicate system mode.

Cluster number (C#) can be modified in system mode only to indicate which window register should be used to determine addresses for the Load and Store Word Direct instructions. If the bit is clear, the instruction uses the read-only window register containing the constant 0x3FF00000; if the bit is set, the instruction uses the modifiable window register.

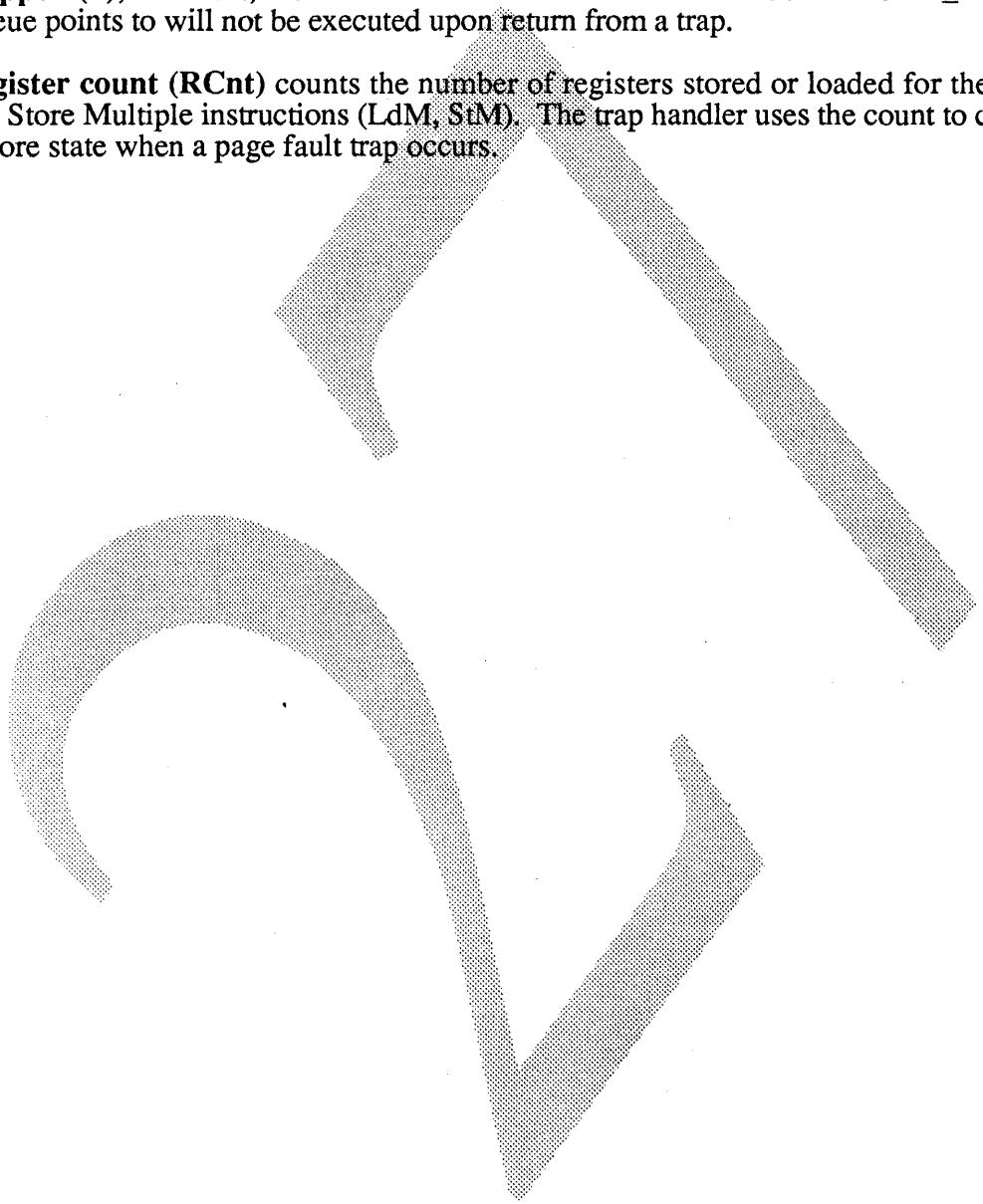
The **Condition Codes (CCode)** are modified by the integer instructions or by the Load Condition (Lcc) instruction, which writes directly to the System Status/Control register. They can be tested by the Branch and Skip on Condition (Bcc, Skcc) instructions.

Halted (W) indicates that the PU is halted. Other PUs use this status bit while executing Resume, Start, and Send instructions. Interrupt selection logic also uses the Halted bit.

Second half (2) indicates an instruction page fault on the second half of an extended operation when set. The page fault handler uses the Second half bit to restore state or to determine which part of the instruction faulted.

Skipped (S), when set, indicates that the instruction that the head of the PC_Save Queue points to will not be executed upon return from a trap.

Register count (RCnt) counts the number of registers stored or loaded for the Load and Store Multiple instructions (LdM, StM). The trap handler uses the count to correctly restore state when a page fault trap occurs.



Traps and Interrupts

The Antares CPU uses several different types of traps and interrupts which have specific characteristics. These characteristics include whether the exception is a *trap* or *interrupt*, whether it is a *directed* or *nondirected interrupt*, and whether it is *nonmaskable, maskable,* or *individually enabled.*

The interrupt handler distinguishes traps from interrupts by the source of the interrupt or trap. *Traps* are synchronous (i.e. caused by a particular instruction), while most *interrupts* are asynchronous (originate externally) and global in origin, except for the Inter-PU interrupt.

The externally originating interrupts are usually *nondirected*, which means that any PU can process the interrupt. The one type of interrupt that is *directed* is the Inter-PU interrupt (1B), which originates on the chip and is directed at particular PUs. The interrupt handler directs all traps to the PU that caused the trap.

When a nondirected interrupt occurs, the interrupt handler uses a priority system to select the PU to process the interrupt. It first looks for a PU with the *Available* bit set in the Status_Save register. If no PUs are "available," the interrupt handler looks for a "halted" PU. Running PUs have the lowest priority. If none of the PUs can accept an interrupt (i.e., trap enable bit in the Status_Save register is cleared), the handler holds the interrupt until a PU can accept it. Interrupts that can be held have a *pending* status bit in the Global Status register that indicates that the interrupt is waiting for processing.

If none of the PUs to which an Inter-PU interrupt is directed to can accept the interrupt, the instruction is held until the interrupt can be processed.

Several bits in the special registers enable traps and interrupts. Each PU has a master trap enable bit in its local Status_Save register (bit 11, T). The Status_Save register also contains two individual trap enables: the overflow trap enable (bit 3, V) and the taken branch trap enable (bit 10, B). The master CPU interrupt enable resides in the Global Status register (bit 18, I), and the event counter interrupt enables occupy bits 7 and 15 of the Event Configuration register.

Nonmaskable interrupts can override the state of the local trap enables. Interrupts and traps that have individual enabling bits are called *individually enabled*, and the rest fall into the category termed *maskable*.

The table below shows the possible states of the PU master trap enable bit and the individual enables. The individual enables include the overflow and taken branch trap enables and the master CPU and event counter interrupt enables.

Table 1. Trap and Interrupt Enables

Master PU Enable	Individual Enable	Meaning
1	1	PU can be interrupted or trap
1	0	PU can trap, individual trap/interrupt is disabled
0	0	PU has trapped or been interrupted, no others will be accepted
0	1	PU has trapped or been interrupted, another PU can be interrupted

The following events occur when a trap or interrupt occurs. (A → B → C means B is stored in C, then A is stored in B)

0	→	System Status/Control	→	Status_Save register
0	→	individual enable (if any)		
Trap # * 8	→	PC-Queue	→	PC_Save Queue
Address	→	Trap Argument/Interrupt Argument		

When a PU traps or interrupts, the System Status/Control register resets, including the master PU trap enable bit. If the trap or interrupt is indivudually enabled, the individual enable resets.

If a maskable or individually enabled trap or interrupt occurs on a PU when traps or interrupts are disabled for that PU, but enabled for the individual trap, a machine check (1E) interrupt occurs.

If multiple interrupts occur simultaneously, only the one with the highest number is reported; all other interrupts are held until interrupts are re-enabled.

When the trap enable bit in the System Status/Control register is set, each cycle moves the contents of System Status/Control into the Status_Save register and the contents of the Current and Next Program Counter registers into the PC_Save Queue. The transfer does not occur when the trap enable bit is cleared.

The following table lists the interrupt/trap causes. The table shows the interrupt and trap names, numbers, and other information, and lists some implications of the existence of the interrupt or trap.

Table 2. Traps and Interrupts

Name	Codes	Trap No.	Other
Reset	I, N, B	1F	Occurs on powerup reset (ND), reset pushbutton (ND), or Res instruction (D).
Machine Check	I, N, B	1E	Occurs on bus error, internal detectable hardware failure (might be used with a redundant 'shadow' mode, where it indicates a mismatch between what was put on the bus, and what should have been on the bus), and when trap occurs when traps are disabled.
Restart	I, N, B	1D	Caused by hardware deadlock detection (ND) or results from one PU interrupting another with the Restart (Res) instruction.
PowerFail/OverTemp	I, N, ND	1C	
Inter-PU	I, M, D	1B	Results from one PU interrupting another with the Interrupt (Int) instruction.
Inter-chip	I, M, ND	1A*	Results from access of an address mapped in another node with the page table 'message' bit set .

Codes: I = Interrupt, T = Trap, N = Nonmaskable, M = Maskable, E = Individually enabled, D = Directed, ND = Nondirected, B = Both

*Will set an address into the Interrupt Argument (S14)register.

Table 2. Interrupts (continued)

Name	Codes	Trap No.	Other
Event Counter	I, E, ND	19	Individual enables in Event Counter Configuration (S12) register. Results from overflow of one of the event counters (S10 and S11). Check bit in configuration register (S12) to determine which counter overflowed.
External	I, E, ND	18	I/O interrupt
Overflow/Divide by 0	T, E, D	17	Individual enable. Check opcode to distinguish an overflow from a divide by 0.
Illegal or Privileged Op	T, M, D	16	Check opcode to distinguish an illegal operation from a privileged operation or a taken branch.
Taken Branch	T, E, D	16	Check opcode to distinguish an illegal operation from a privileged operation or a taken branch.
Data access fault	T, M, D	15†	Results from user attempt to access system pageor from user or system attempt to access data on execute-only page or to write to a read-only page.
Instruction access fault	T, M, D	14	
Data Page Fault	T, M, D	13†	
Inst Page Fault	T, M, D	12	
Node Busy	T, M, D	11†	Results from failed access to an address mapped in another node with the page table 'message' bit set
Software Trap	T, M, D	F..0	Opcodes F..0

Codes: I = Interrupt, T = Trap, N = Nonmaskable, M = Maskable, E = Individually enabled, D = Directed, ND = Nondirected, B = Both

†Will set an address into the Trap Argument (S8) register.

Condition Codes

The System Status and Control register for the Antares CPU contains seven condition code bits. The processor sets them as described below for all instructions except partial word instructions.

N Negative condition code bit. This bit gets set if the true result is less than zero. The N condition code bit may be different from the most significant bit of the result if an arithmetic overflow occurs.

Z Zero condition code bit. This bit gets set if the result equals zero.

V Overflow condition code bit. This bit gets set if an arithmetic overflow occurs.

C Carry condition code bits. The highest of these four bits is set if a carry is generated out of the most significant bit of the operand for all instructions except the partial word instructions. The lower three bits are cleared. The indication *not* carry means that all four Carry bits are cleared.

The Carry bit gets set for the Add instructions under the following conditions:

$$S \cdot D + [(S + D) \cdot \neg R]$$

If the source and destination registers both have their sign bits set, then the Carry bit is set. If either the source or destination register has its sign bit set and the result does not have its sign bit set, the Carry bit is set.

The Carry bit gets set for the Subtract and Compare instructions under the following conditions:

$$(\neg S) \cdot D + [((\neg S) + D) \cdot R]$$

If the source register does not have its sign bit set and the destination register does have its sign bit set, the Carry bit is set. If the source register does not have its sign bit set or the destination register does have its sign bit set and the result does not have its sign bit set, the Carry bit is set.

Partial Word Condition Codes

The condition codes for partial word instructions (Add with Carry Partial, Add Partial, Compare Partial, Load Partial Carry, Subtract with Carry Partial, and Subtract Partial) function differently.

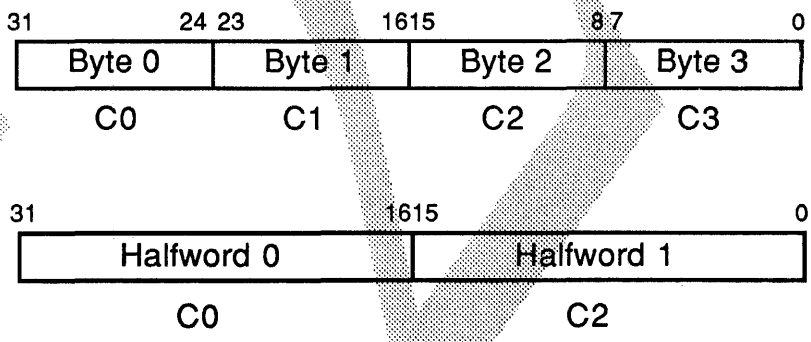
N Negative condition code bit. This bit gets set if any halfword/byte result is negative.

Z Zero condition code bit. This bit gets set if any halfword/byte result equals zero.

V Overflow condition code bit. This bit is always cleared; no overflow can occur in partial word operations.

C Carry condition code bits. A Carry bit gets set if a carry is generated out of the most significant bit of the halfword or byte corresponding to that specific Carry bit. The four Carry bits (C0 through C3) are set individually if the halfword/byte bit in the System Status/Control register is cleared. If the halfword/byte bit is set, Carry bits C0 and C2 are set and C1 and C3 are cleared.

The individual Carry bits correspond to the following bytes and halfwords:



Notation

The following tables show the notation used in this manual.

Table 3. Operands

Operand	Meaning
RegA	Source register A Instruction <3:0>
RegB	Source register B Instruction <7:4>
Base	Base register Instruction <1:0> or <3:0> if extended
Mask	Mask register
PC	Program counter
Next PC	Halfword address of next instruction to be executed
Imm, Pos, Len, Amt	Immediate operands: Immediate, Position, Length, Amount Constants encoded in the instruction
Displacement	Displacement/direct address
Sp	Special register
@R	Contents of memory location addressed by register
R0, R1...R15	General-purpose registers
R0...R3	Base registers for normal instructions
R-0...R16	Base registers for extended instructions
R4	Return link register
R<b>	Bit *b* of R
Reg<m:n>	bits m through n, inclusive, of specified register

Table 4. Operators and Symbols

Operator	Meaning
→	Move left operand to right operand
+	Add operands
-	Subtract right operand from left operand
*	Multiply operands
/	Divide left operand by right operand
,	Concatenate
~	Ones complement the operand
&	Bitwise logically AND the operands
^	Bitwise exclusive OR the operands
|	Bitwise logically OR the operands
<<	Shift the left operand left by the amount specified by the right operand
>>	Shift the left operand right by the amount specified by the right operand
@	Contents of memory location

Quick Reference

The tables in this section provide quick reference to Antares instructions. Table 5 contains an alphabetic summary, while Table 6 groups the instructions functionally.

Table 5. Alphabetic Summary of Instructions

Mnemonic	Instruction	Operation	Page
AdCP	Add with Carry Partial	AdCP RegB(H/B) + RegA → RegB	24
Add	Add	Add RegB + RegA → RegB	25
Add	Add Immediate	Add RegB + Imm → RegB	26
AddC	Add with Carry	AddC RegB + RegA → RegB	27
AddP	Add Partial	AddP RegB(H/B) + RegA → RegB	28
AdPC	Add Program Counter	AdPC PC + 1 + RegA → RegA	29
And	And	And RegB & RegA → RegB	30
AndC	And Complement	AndC RegB & ~RegA → RegB	31
Bcc	Branch on Condition	Bcc PC + Displacement	32
CDC	Create Data Cache Line	CDC @RegA	34
ClrF	Clear Field	ClrF 0 → RegA <Mask>	35
ClrM	Clear Mode	ClrM BitNumber	36
CLZ	Count Leading Zeros	CLZ RegA → RegB	38
Cmp	Compare	Cmp RegB - RegA	39
Cmp	Compare Immediate	Cmp RegB - Imm	40
CmpP	Compare Partial	CmpP RegB(H/B) - RegA	41
Dep	Deposit	Dep RegB → RegA <Mask>	42
Div	Divide	Div (MDfull & (Remainder<<32) , RegB) / RegA → {RegB, Remainder}	44
DSh	Double Shift	DSh (RegB, RegA) >> Mask → RegA	46
ExtS	Extract Signed	ExtS RegA <Mask> → RegB	48
ExtU	Extract Unsigned	ExtU RegA <Mask> → RegB	50
FDC	Flush Data Cache Line	FDC @RegA	52
IDC	Invalidate Data Cache Line	IDC @RegA	53
IIC	Invalidate Instruction Cache Line	IIC @RegA	54
IICa	Invalidate Instruction Cache	IICa	55
Ins	Insert	Ins RegB → RegA <Mask>	56
Int	Interrupt	Int PuMask	58
ITLB	Invalidate Translation Buffer	ITLB	59
Jmp	Jump Relative	Jmp PC + Displacement	60
Jmp	Jump Absolute	Jmp @RegA	61

Table 5. Alphabetic Summary of Instructions (continued)

Mnemonic	Instruction	Operation	Page
JmpL	Jump and Link	JmpL next PC + 1 → R4, Jmp @RegA	62
Lcc	Load Condition	Lcc → RegB	63
Ld	Load Immediate	Ld Imm → RegB	65
LdB	Load Byte	LdB @Base → RegA	66
LdCP	Load with Carry Partial	LdCP Extend ~(C0..C3) → RegA	67
LdM	Load Multiple	LdM @RegA → RegB..Reg1	68
LdW	Load Word	LdW @Base + Displacement → RegB	69
LdW	Load Word Direct	LdW @Displacement → RegB	70
LdW	Load Word	LdW @RegA → RegB	72
LdWE	Load Word Extended	LdWE @RegA + Displacement → RegB	73
Mov	Move	Mov RegA → RegB	74
Mov	Move to Special	Mov RegB → Sp	75
Mov	Move from Special	Mov Sp → RegB	77
Msk	Mask Generate	Msk Pos for Len	79
Msk	Mask Generate	Msk RegA for Len	80
Mul	Multiply	Mul RegB * RegA + {MDfull & Product} → {Product, RegB}	81
MulP	Multiply Partial	MulP RegB(H/B) * RegA → RegB	82
Neg	Negate	Neg - RegA → RegB	83
Not	Not	Not ~RegA → RegB	84
Or	Or	Or RegA | RegB → RegB	85
PDC	Prefetch Data Cache Line	PDC @RegA	86
PIC	Prefetch Instruction Cache Line	PIC @RegA	87
PIC	Prefetch Instruction Cache Line	PIC PC + Displacement	88
RDTX	Read Data Tag by Index	RDTX @RegA → RegB	89
Res	Restart	Res PuMask	90
Rsm	Resume	Rsm PuMask	91
Rtl	Return from Interrupt	Rtl	92
SbCP	Subtract with Carry Partial	SbCP RegB(H/B) - RegA → RegB	93
Send	Send	Send RegB → Destination Register<PuMask>	94
SetF	Set Field	SetF 1 → RegA<Mask>	95

Table 5. Alphabetic Summary of Instructions (continued)

Mnemonic	Instruction	Operation	Page
SetM	Set Mode	SetM BitNumber	96
ShL	Shift Left	ShL RegB << Amt → RegB	98
ShR	Shift Right	ShR RegA >> Amt → RegA	99
Skcc	Skip on Condition	Skcc	100
StB	Store Byte	StB RegB → @RegB	102
StM	Store Multiple	StM RegB..R1 → @RegA	103
Strt	Start	Strt RegB, PuMask	104
StW	Store Word	StW RegB → @Base + Displacement	105
StW	Store Word Direct	StW RegB → @Displacement	106
StW	Store Word	StW RegB → @Base	107
StWE	Store Word Extended	StWE RegB → @Base + Displacement	108
Sub	Subtract	Sub RegB - RegA → RegB	109
Sub	Subtract Immediate	Sub RegB - Imm → RegB	110
SubC	Subtract with Carry	SubC RegB - RegA + (Carry -1) → RegB	111
SubP	Subtract Partial	SubP RegB(H/B) - RegA → RegB	112
Trap	Trap	Trap Trapnum	113
TstF	Test Field	TstF RegA <Mask>	114
TstM	Test Mode	TstM BitNumber	115
UDC	Update Data Cache Line	UDC @RegA	117
VDC	Validate Data Cache Line	VDC @RegA	118
Wait	Wait	Wait PuMask	119
XOr	Exclusive Or	XOr RegA ^ RegB → RegB	120

Table 6. Functional Summary of Instructions

Mnemonic	Instruction	Operation	Page
	Register Load, Store, and Move Instructions		
Lcc	Load Condition	Lcc RegB	63
Ld	Load Immediate	Ld Imm → RegB	65
LdB	Load Byte	LdB @Base → RegB	66
LdCP	Load with Carry Partial	LdCP Extend ~(C0..C3) → RegA	67
LdM	Load Multiple	LdM @RegA → RegB..R1	68
LdW	Load Word	LdW @Base + Displacement → RegB	69
LdW	Load Word Direct	LdW @Displacement → RegB	70
LdW	Load Word	LdW @RegA → RegB	72
LdWE	Load Word Extended	LdWE @RegA + Displacement → RegB	73
Mov	Move	Mov RegA → RegB	74
Mov	Move to Special	Mov RegB → Sp	75
Mov	Move from Special	Mov Sp → RegB	77
StB	Store Byte	StB RegB → @Base	102
StM	Store Multiple	StM RegB → @RegB	103
StW	Store Word	StW RegB → @Base + Displacement	105
StW	Store Word Direct	StW RegB → @Displacement	106
StW	Store Word	StW RegB → @Base	107
StWE	Store Word Extended	StWE RegB → @Base + Displacement	108
	Arithmetic Instructions		
AdCP	Add with Carry Partial	AdCP RegB(H/B) + RegA → RegB	24
Add	Add	Add RegB + RegA → RegB	25
Add	Add Immediate	Add RegB + Imm → RegA	26
AddC	Add with Carry	AddC RegB + RegA → RegB	27
AddP	Add Partial	AddP RegB(H/B) + RegA → RegB	28
CLZ	Count Leading Zeros	CLZ RegA → RegB	38
Div	Divide	Div (MDfull & (Remainder<<32) + RegB) / RegA → { RegB, Remainder}	44
Mul	Multiply	RegB * RegA + (MDfull & Product) → {Product, RegB}	81
MulP	Multiply Partial	MulP RegB(H/B) * RegA → RegB	82
Neg	Negate	Neg - RegA → RegB	83
SbCP	Subtract with Carry Partial	SbCP RegB(H/B) - RegA → RegB	93
Sub	Subtract	Sub RegB - RegA → RegB	109
Sub	Subtract Immediate	Sub RegB - Imm → RegB	110
SubC	Subtract with Carry	SubC RegB - RegA + (Carry -1) → RegB	111
SubP	Subtract Partial	SubP RegB(H/B) - RegA → RegB	112

Table 6. Functional Summary of Instructions (continued)

Mnemonic	Instruction	Operation	Page
	Compare, Test, Branch, Jump, and Skip Instructions		
AdPC	Add Program Counter	AdPC PC + 1 + RegA → RegA	29
Bcc	Branch on Condition	Bcc PC + Displacement	32
Cmp	Compare	Cmp RegB - RegA	39
Cmp	Compare Immediate	Cmp RegB - Imm	40
CmpP	Compare Partial	RegB (H/B) - RegA	41
Jmp	Jump Relative	Jmp PC + Displacement	60
Jmp	Jump Absolute	Jmp @RegA	61
JmpL	Jump and Link	JmpL next PC + 1 → R4, Jmp @RegA	62
Skcc	Skip on Condition	Skcc	100
TstF	Test Field	TstF RegA	114
TstM	Test Mode	TstM BitNumber	115
	Shift, Logical, and Field Manipulation Instructions		
And	And	And RegB & RegA → RegB	30
AndC	And Complement	AndC RegB & ~RegA → RegB	31
ClrF	Clear Field	ClrF RegA	35
Dep	Deposit	Dep RegB → RegA <Mask>	42
DSh	Double Shift	DSh (RegB, RegA) >> Mask → RegA	46
ExtS	Extract Signed	ExtS RegA <Mask> → RegB	48
ExtU	Extract Unsigned	ExtU RegA <Mask> → RegB	50
Ins	Insert	Ins RegB → RegA <Mask>	56
Msk	Mask Generate	Msk Pos for Len	79
Msk	Mask Generate	Msk RegA for Len	80
Not	Not	Not ~RegA → RegB	84
Or	Or	Or RegA | RegB → RegB	85
SetF	Set Field	SetF1 → RegA <Mask>	95
ShL	Shift Left	ShL RegB << Amt → RegB	98
ShR	Shift Right	ShR RegA >> Amt → RegA	99
XOr	Exclusive Or	XOr RegA ^ RegB → RegB	120

Table 6. Functional Summary of Instructions (continued)

Mnemonic	Instruction	Operation	Page
	Cache Control Instructions		
CDC	Create Data Cache Line	CDC @RegA	34
FDC	Flush Data Cache Line	FDC @RegA	52
IDC	Invalidate Data Cache Line	IDC @RegA	53
IIC	Invalidate Instruction Cache Line	IIC @RegA	54
IICa	Invalidate Instruction Cache	IICa	55
PDC	Prefetch Data Cache Line	PDC @RegA	86
PIC	Prefetch Instruction Cache Line	PIC @RegA	87
PIC	Prefetch Instruction Cache Line	PIC PC + Displacement	88
UDC	Update Data Cache Line	UDC @RegA	117
VDC	Validate Data Cache Line	VDC @RegA	118
	Control Instructions		
ClrM	Clear Mode	ClrM BitNumber	36
Int	Interrupt	Int PuMask	58
ITLB	Invalidate Translation Lookaside Buffer	ITLB	59
RDTX	Read Data Tag by Index	RDTX @RegA → RegB	89
Res	Restart	Res PuMask	90
Rsm	Resume	Rsm PuMask	91
Rtl	Return from Interrupt	Rtl	92
Send	Send	Send RegB → Destination Register<PuMask>	94
SetM	Set Mode	SetM BitNumber	96
Strt	Start	Strt RegB, PuMask	104
Trap	Trap	Trap Trapnum	113
Wait	Wait	Wait PuMask	119

AdCP — Add with Carry Partial — AdCP

Operation: **AdCP RegB(H/B) + RegA → RegB**

Description: If **H/B** is clear, add **RegB** to **RegA** using carry bits C0, C1, C2, and C3 as carries into bytes. Set the C0, C, C2, and C3 carry bits from the carryout of the corresponding bytes.

If **H/B** is set, add **RegB** to **RegA** using carry bits C0 and C2 as carries into halfwords. Set the C0 and C2 carry bits from the carryout of the corresponding halfwords. Clear bits C1 and C3.

Condition codes: **H/B = 0**

N -- Set if any byte is negative, cleared otherwise.
Z -- Set if any byte is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any byte carries, cleared otherwise.

H/B = 1

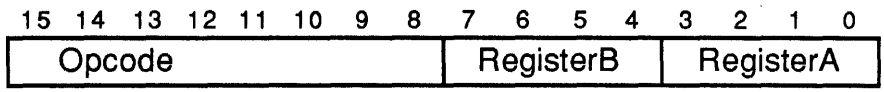
N -- Set if any halfword is negative, cleared otherwise.
Z -- Set if any halfword is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any halfword carries, cleared otherwise.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Add

Add

Add

Operation: **Add RegB + RegA → RegB**

Description: Add the contents of **RegA** to the contents of **RegB**. Store the result in **RegB**. Clear carry bits C1, C2, and C3.

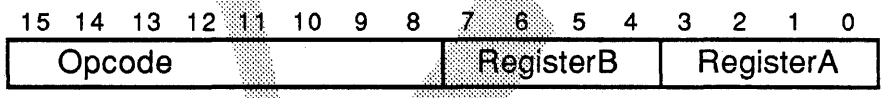
Condition codes:
N -- Set if result is negative, cleared otherwise.
Z -- Set if result is equal to zero, cleared otherwise.
V -- Set if an overflow is generated, cleared otherwise.
C -- Set if a carry is generated, cleared otherwise.

Exceptions: A trap occurs if the operation sets the V condition code bit and Trap_on_Overflow is enabled.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Add — Add Immediate — Add

Operation: **Add RegB + Imm → RegB**

Description: Add the contents of **RegB** to the immediate value **Imm**. Store the result in **RegB** and clear carry bits C1, C2, and C3. The range for the value **Imm** is 1 through 256; the encoding is (**Imm** -1).

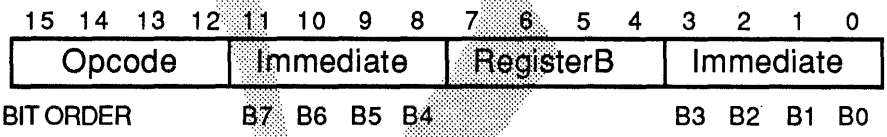
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- Set if an overflow is generated, cleared otherwise.
C -- Set if a carry is generated, cleared otherwise.

Exceptions: A trap occurs if the operation sets the V condition code bit and Trap_on_Overflow is enabled.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

AddC — Add with Carry — AddC

Operation: **AddC RegB + RegA → RegB**

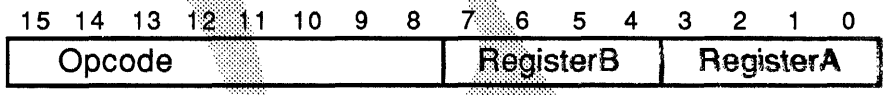
Description: Add the contents of the C0 carry status bit and the contents of **RegA** to the contents of **RegB**. Store the result in **RegB** and clear carry bits C1, C2, and C3.

Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- Set if an overflow is generated, cleared otherwise.
C -- Set if a carry is generated, cleared otherwise.

Exceptions: A trap occurs if the operation sets the V condition code bit and Trap_on_Overflow is enabled.

Timing: This instruction takes one cycle.

Format:



Example:

AddP — Add Partial — AddP

Operation: **AddP RegB(H/B) + RegA → RegB**

Description: If **H/B** is clear, add **RegB** to **RegA**, inhibiting carries between bytes. Set the C0, C1, C2, and C3 carry bits from the carryout of the corresponding bytes.

If **H/B** is set, add **RegB** to **RegA**, inhibiting carries between halfwords. Set the C0 and C2 carry bits from the carryout of the corresponding halfwords. Clear the C1 and C3 carry bits.

Condition codes: **H/B = 0**

N -- Set if any byte is negative, cleared otherwise.
Z -- Set if any byte is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any byte carries, cleared otherwise.

H/B = 1

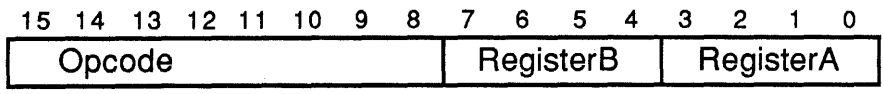
N -- Set if any halfword is negative, cleared otherwise.
Z -- Set if any halfword is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any halfword carries, cleared otherwise.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

AdPC — Add Program Counter — AdPC

Operation: **AdPC PC + 1 RegA → RegA**

Description: Add the halfword address of the **Current PC+ 1** to the contents of **RegA** and store the result in **RegA**.

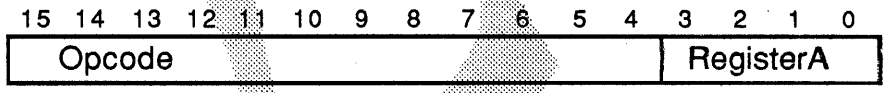
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

And

And

And

Operation: **And RegB & RegA → RegB**

Description: Logically AND the contents of **RegB** with the contents of **RegA**. Store the result in **RegB**.

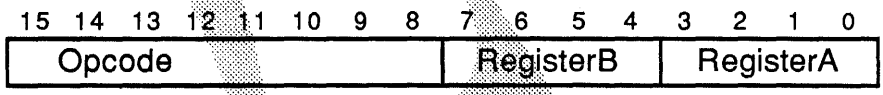
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

AndC — And Complement — AndC

Operation: **AndC RegB & ~RegA → RegB**

Description: Logically AND the contents of **RegB** with the ones complement of the contents of **RegA**. Store the result in **RegB**.

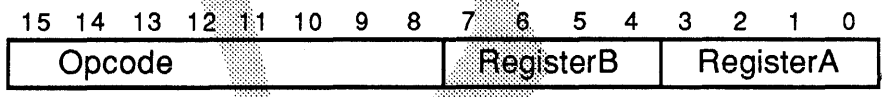
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Bcc — Branch on Condition — Bcc

Operation: **Bcc PC + Displacement**

Description: If the condition cc specified by the instruction is met, continue execution at the instruction whose halfword address is **PC + Displacement**. Otherwise, continue execution. In all cases, execute the instruction whose address is in **Next PC**. **Displacement** is a 9-bit quantity (8 bits + sign bit).

The encodings and meanings of the **cc** field are shown in the table below.

Table 7. CC Field Encodings

Encoding	cc	Condition	Meaning
0		reserved	
1	EQ	Z=1	equal
2	LT	N=1	less than
3	LE	N=1 or Z=1	less or equal
4	LO	C=0	lower
5	LS	C=0 or Z=1	less or same
6	OV	V=1	overflow
7		reserved	
8		reserved	
9	NE	Z=0	not equal
10	GE	N=0	greater or equal
11	GT	N=0 and Z=0	greater than
12	HS	C=1	higher or same
13	HI	C=1 and Z=0	higher
14	NV	V=0	no overflow
15		reserved	

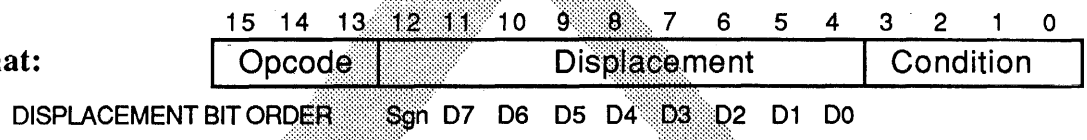
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

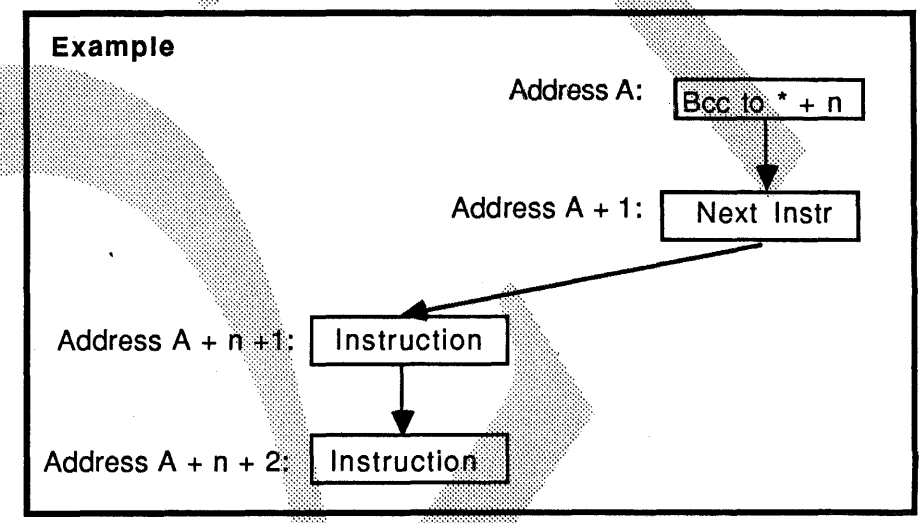
Timing: This instruction takes one cycle.

Format:



DISPLACEMENT BIT ORDER Sgn D7 D6 D5 D4 D3 D2 D1 D0

Example: The example below illustrates the order in which the processor executes instructions following the Bcc instruction.



CDC — Create Data Cache Line — CDC

Operation: CDC @RegA

Description: If the LRU cache line in the set addressed by **RegA** is modified, write the data back to Main Memory. Set the tag for the LRU line to valid and unmodified, with the address in **RegA**.

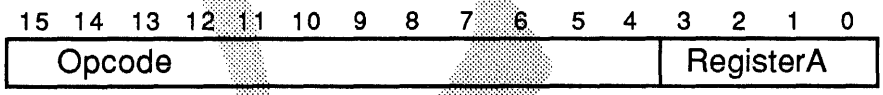
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

ClrF — Clear Field — ClrF

Operation: ClrF 0 → RegA<Mask>

Description: Clear the bits of the field defined by the **Mask** register in **RegA** to zeroes. Store the result in **RegA**. This instruction does not affect bits outside the field.

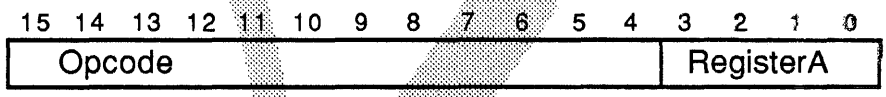
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special (Mov) instruction.

Timing: This instruction takes one cycle.

Format:



Example:

ClrM — Clear Mode — ClrM

Operation: **ClrM BitNumber**

Description: Clear the specified **Mode** bit in the **System Status/Control** register. The mode bits are:

0	MD control
1	Halfword/byte
2	MD full
3	Overflow trap enable
4	Register available
5	reserved
6	reserved
7	reserved
8 -9	reserved
10	Branch taken trap enable†
11	PU trap enable/disable†
12	User/system†
13	Cluster number†
14	reserved
15	reserved

†Indicates privileged.

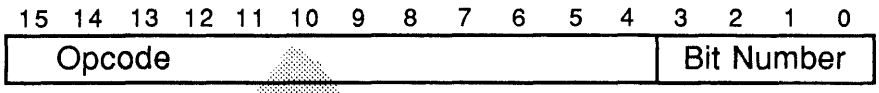
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if the instruction is executed when in user mode and any one of mode bits 8 through 15 is specified. An illegal operation trap occurs if mode bit 5, 6, 7, 8, 9, 14, or 15 is specified in the instruction.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

CLZ — Count Leading Zeros — CLZ

Operation: **CLZ RegA → RegB**

Description: Count the number of leading zeros in **RegA**. Store the number in **RegB**.

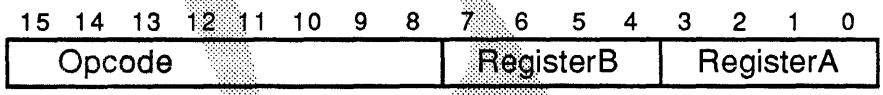
Condition codes:
N -- Set if **RegA** is less than zero, cleared otherwise.
Z -- Set if **RegA** equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Cmp — Compare — Cmp

Operation: **Cmp RegB - RegA**

Description: Subtract the contents of **RegA** from the contents of **RegB** and set the condition codes.

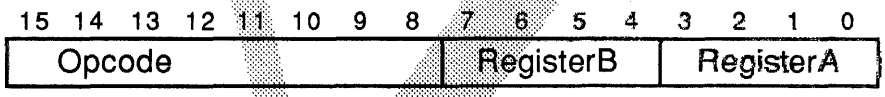
Condition codes:
N -- Set if **RegB** is arithmetically less than **RegA**, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if unsigned **RegB** is greater than or equal to **RegA**. (C = *not* borrow), cleared otherwise

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Cmp — Compare Immediate — Cmp

Operation: **Cmp RegB - Imm**

Description: Subtract the immediate value **Imm** from the contents of **RegB** and set the condition codes. The value **Imm** must be in the range 1 through 256. Encoding for **Imm** is (**Imm** - 1).

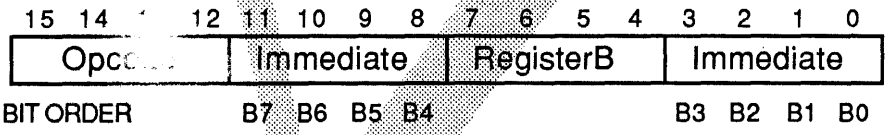
Condition codes:
N -- Set if **RegB** is arithmetically less than **Imm**, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if unsigned **RegB** is greater than or equal to **Imm**,. cleared otherwise (C = *not* borrow).

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



DISPLACEMENT BIT ORDER B7 B6 B5 B4 B3 B2 B1 B0

Example:

CmpP — Compare Partial — CmpP

Operation: **CmpP RegB(H/B) - RegA**

Description: Subtract **RegA** from **RegB**, forcing the carryin to 1 in each byte or halfword and set the condition codes.

Condition codes: **H/B = 0**

N -- Set if any byte is negative, cleared otherwise.
Z -- Set if any byte is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Load C0, C1, C2, C3.

H/B = 1

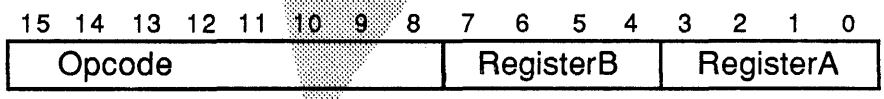
N -- Set if any halfword is negative, cleared otherwise.
Z -- Set if any halfword is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Load C0 and C2; clear C1 and C3.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Dep — Deposit — Dep

Operation: **Dep RegB → RegA<Mask>**

Description: Left-shift the contents of **RegB** until the least significant bit (LSB) is aligned with the LSB of the field defined by the **Mask** register. Clear all bits of the result that are not in the field defined by the **Mask** register and store that result into **RegA**.

If Length equals 32, this instruction is equivalent to Shift Left by **Pos**.

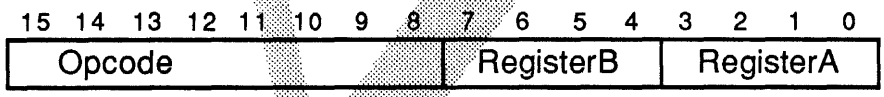
Condition codes: N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None

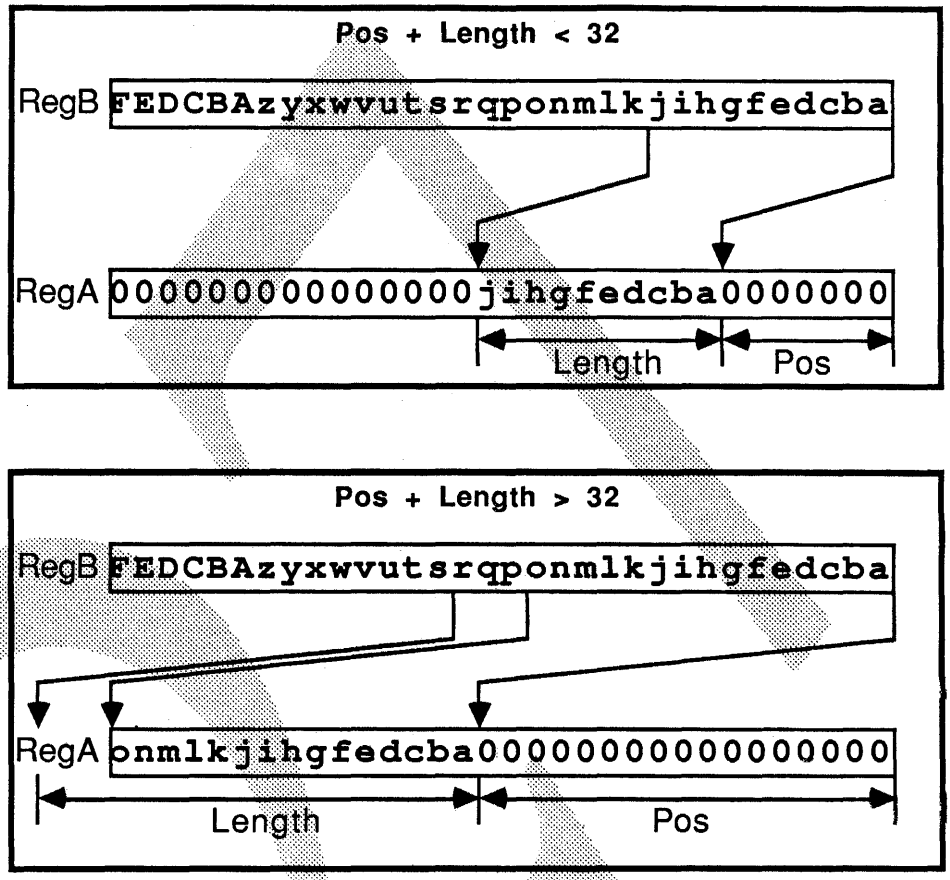
Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special (Mov) instruction.

Timing: This instruction takes one cycle.

Format:



Examples:


Pos + Length < 32
RegB FEDCBAzyxwvutsrqponmlkjihgfedcba
RegA 00000000000000jihgfedcba0000000
Length
Pos
Pos + Length > 32
RegB FEDCBAzyxwvutsrqponmlkjihgfedcba
RegA onmlkjihgfedcba00000000000000000
Length
Pos

Div — Divide — Div

Operation: **Div ((MDfull & Remainder) , RegB) / RegA → RegB , Remainder**

Description: If the **MDfull** bit is clear, divide the value of **RegB** by the value of **RegA**. If the **MDfull** bit is set, divide the value **Remainder** concatenated with **RegB** by **RegA**. Store the quotient into **RegB** and the remainder into special register **Remainder**. Both operands are treated as unsigned numbers.

The **Remainder** special register allows division of 64-bit numbers. The upper 32 bits of the dividend reside in the **Remainder** register. A special bit in the **System Status/Control** register, the **MDfull** bit, indicates whether or not the **Remainder** register contains data that should be used by the instruction. If the **MDfull** bit is set, the instruction assumes a 64-bit number; if the **MDfull** bit is cleared, it assumes a 32-bit number and does not use the **Remainder** register.

The **MDfull** bit is cleared on the first cycle of each Divide (Div) instruction. The bit gets set when something is moved into the **Remainder** register, using a Move to Special (Mov) instruction.

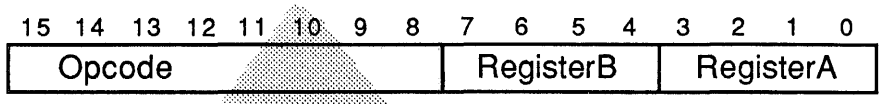
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected??
C -- Not affected.

Exceptions: An Overflow trap occurs if **Remainder** is greater than or equal to **RegA** and **MDfull** equals 1 and the Overflow trap enable bit is set.

Restrictions: None.

Timing: The result is not valid for 17(?) 33(?) cycles.

Format:



Example:

DSh — Double Shift — DSh

Operation: **DSh (RegB , RegA) >> Mask → RegA**

Description: Concatenate the contents of **RegB** with the contents of **RegA** and right shift the doubleword until the rightmost bit defined by the **Mask** register is right-aligned. Store the least significant word of the result in register **RegA**.

If **RegA** equals **RegB**, the result is equivalent to a Rotate Right.

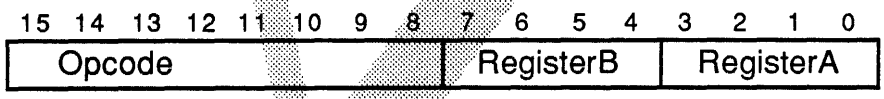
Condition codes:
N -- Set if result is negative, cleared otherwise.
Z -- Set if result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None

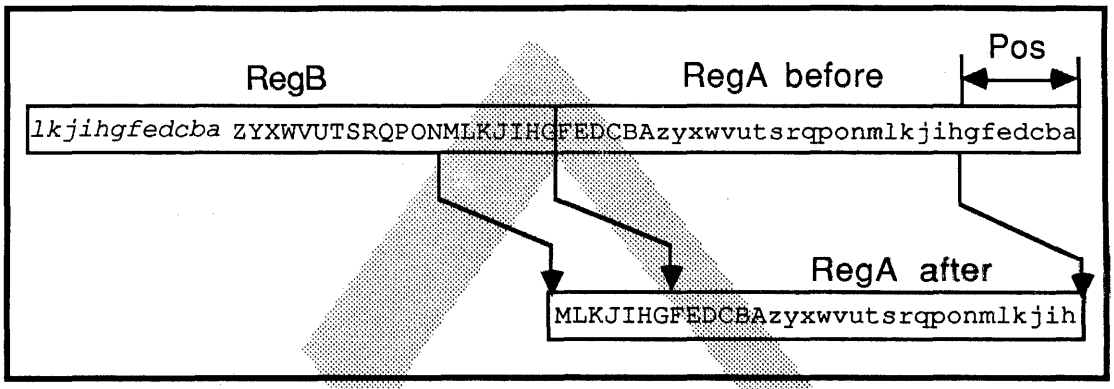
Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special (Mov) instruction.

Timing: This instruction takes one cycle.

Format:



Example:


RegB
RegA before
Pos
lkjihgfedcba ZYXWVUTSRQPONMLKJIHGFEDCBAzyxwvutsrqponmlkjihgfedcba
RegA after
MLKJIHGFEDCBAzyxwvutsrqponmlkjih

ExtS — Extract Signed — ExtS

Operation: **ExtS RegA<Mask> → RegB**

Description: Sign extend a field contained in **RegA** using the bit position of the leftmost bit of the field defined by the **Mask** register as the position of the sign bit, and extending from that bit position to the most significant bit of the word. Arithmetically shift the result right until the rightmost bit of the field defined by the **Mask** register is right-aligned. Store the result in **RegB**.

If Length equals 32, this instruction is equivalent to an arithmetic Shift Right.

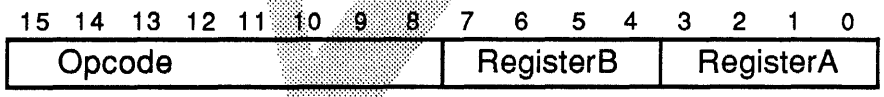
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None

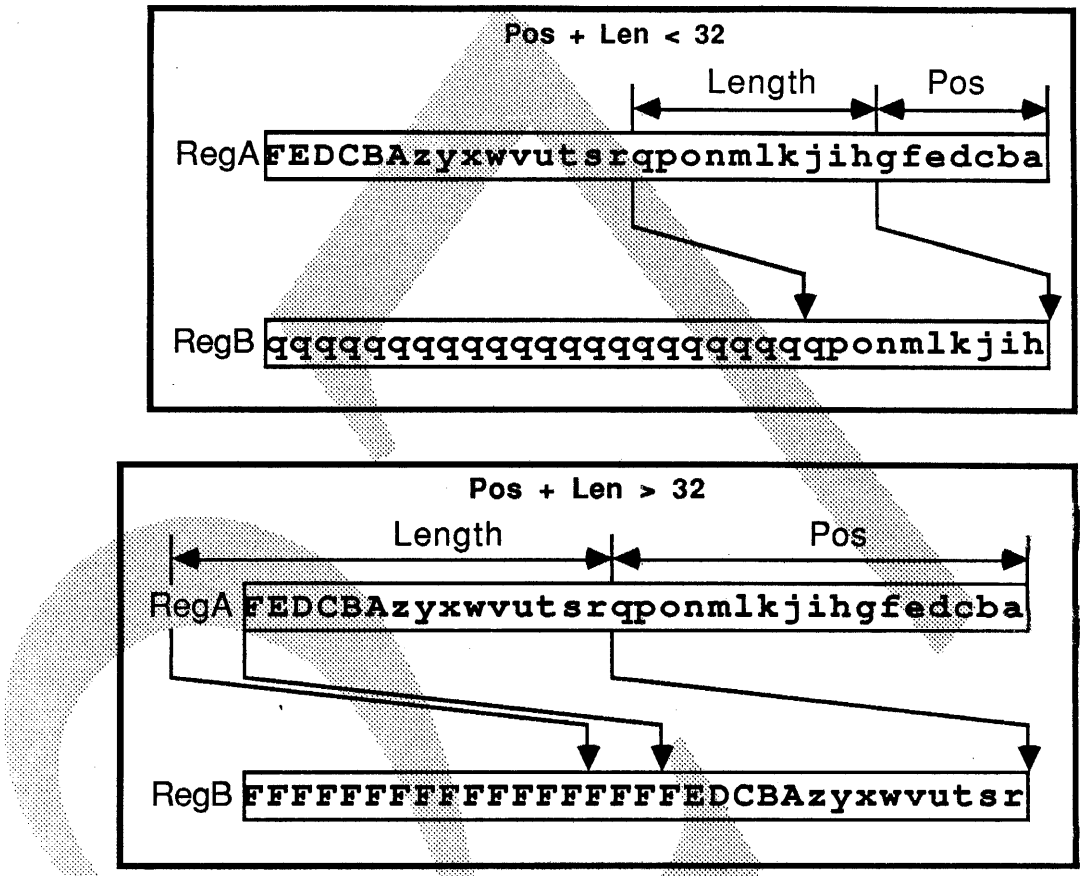
Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special instruction.

Timing: This instruction takes one cycle.

Format:



Examples:


Pos + Len < 32
Length
Pos
RegA FEDCBAzyxwvutsrqponmlkjihgfedcba
RegB qqqqqqqqqqqqqqqqqqqqqqponmlkjih
Pos + Len > 32
Length
Pos
RegA FEDCBAzyxwvutsrqponmlkjihgfedcba
RegB FFFFFFFFFFFFFFFFFEDCBAzyxwvutsr

ExtU — Extract Unsigned — ExtU

Operation: **ExtU RegA<Mask> → RegB**

Description: Clear the bits in **RegA** that are not in the field defined by the **Mask** register. Logically shift the result right until the rightmost bit of the field defined by the **Mask** register is right-aligned. Store the result in **RegB**.

If Length equals 32, this instruction is equivalent to a logical Right Shift by **Pos**.

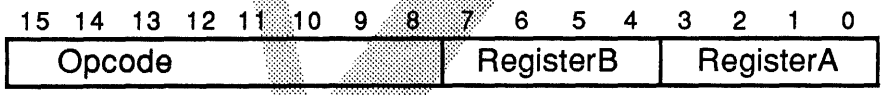
Condition codes: N -- Set if the result is negative.
Z -- Set if the result equals zero.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

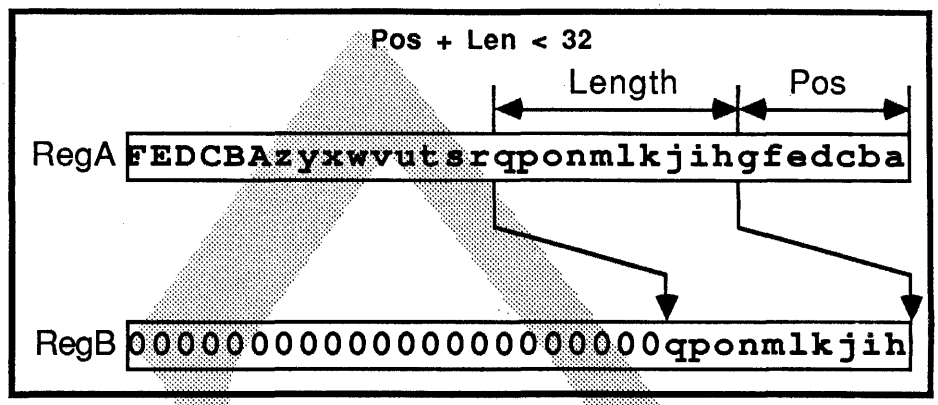
Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special (Mov) instruction.

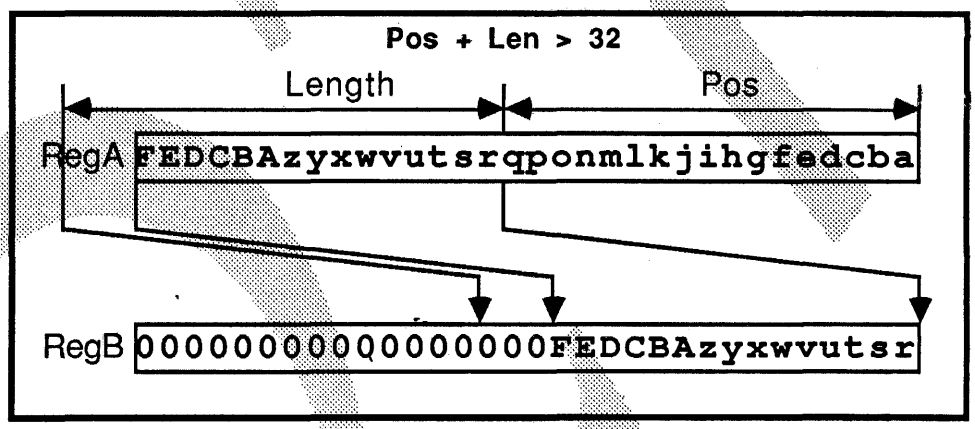
Timing: This instruction takes one cycle.

Format:



Examples:


Pos + Len < 32
Length
Pos
RegA FEDCBAzyxwvutsrqponmlkjihgfedcba
RegB 00000000000000000000qponmlkjih


Pos + Len > 32
Length
Pos
RegA FEDCBAzyxwvutsrqponmlkjihgfedcba
RegB 0000000000000000FEDCBAzyxwvutsr

FDC Flush Data Cache Line FDC

Operation: **FDC @RegA**

Description: If the cache line whose address is in **RegA** is in the data cache and has been modified, write the line back to Main Memory. Then mark the line LRU, invalid, and unmodified. If the line is not in the cache, this instruction has no effect.

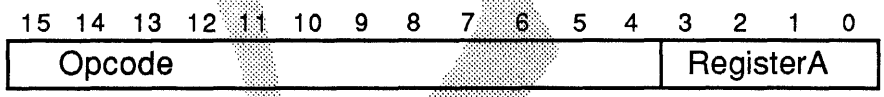
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

IDC Invalidate Data Cache Line IDC

Operation: **IDC @RegA**

Description: If the cache line addressed by **RegA** is in the data cache, mark it LRU, invalid, and unmodified. If the line is not in the cache, this instruction has no effect.

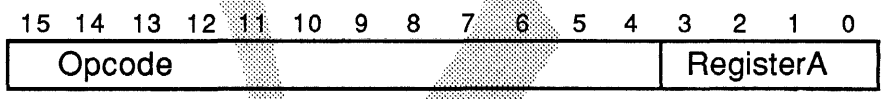
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

IIC — Invalidate Instruction Cache Line — IIC

Operation: IIC @RegA

Description: If the cache line addressed by **RegA** is in the instruction cache, mark it LRU and invalid. If the line is not in the cache, this instruction has no effect.

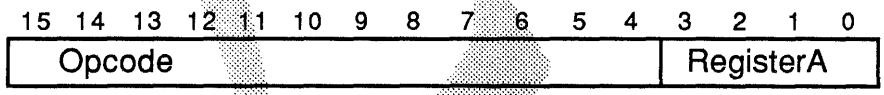
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

IICa Invalidate Instruction Cache IICa

Operation: IICa

Description: Invalidates the entire instruction cache.

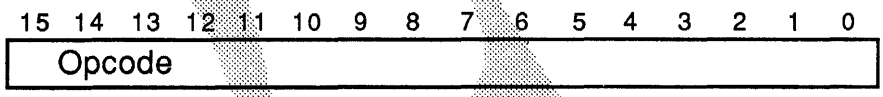
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Ins — Insert — Ins

Operation: **Ins RegB → RegA<Mask>**

Description: Left-shift the contents of **RegB** until the least significant bit (LSB) is aligned with the LSB of the field defined by the **Mask** register. Replace each bit in **RegA** with the corresponding bit of the result if the bit is included in the field defined by the **Mask** register.

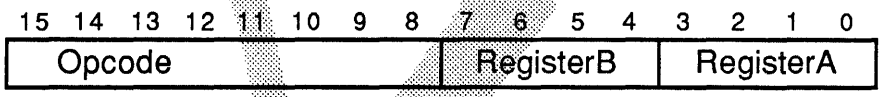
Condition codes:
N -- Set if the result is negative.
Z -- Set if the result equals zero.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

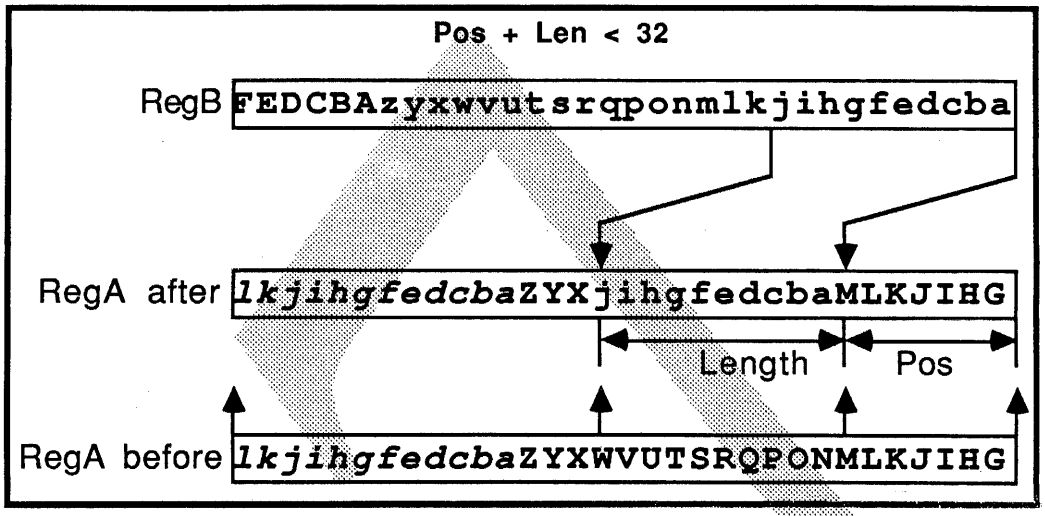
Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special (Mov) instruction.

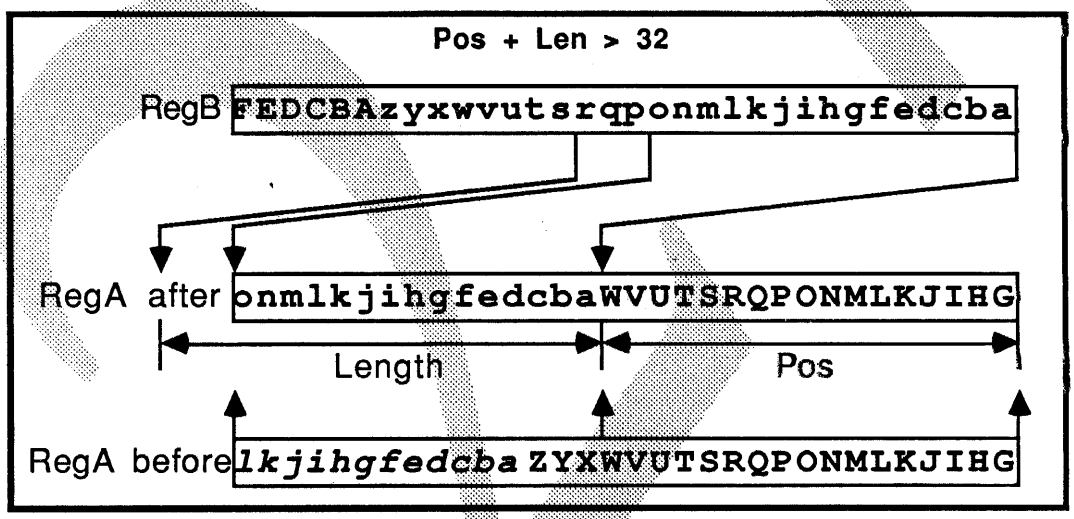
Timing: This instruction takes one cycle.

Format:



Example:

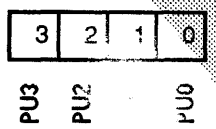

Pos + Len < 32
RegB FEDCBAzyxwvutsrqponmlkjihgfedcba
RegA after lkjihgfedcbaZYXjihgfedcbaMLKJIHG
Length
Pos
RegA before lkjihgfedcbaZYXWVUTSRQPONMLKJIHG


Pos + Len > 32
RegB FEDCBAzyxwvutsrqponmlkjihgfedcba
RegA after onmlkjihgfedcbaWVUTSRQPONMLKJIHG
Length
Pos
RegA before lkjihgfedcba ZYXWVUTSRQPONMLKJIHG

Int — Interrupt — Int

Operation: **Int PuMask**

Description: Cause each PU whose PuMask bit is set to trap to the Inter-PU interrupt vector 0x1C. The PuMask bits correspond to the PUs as shown below.



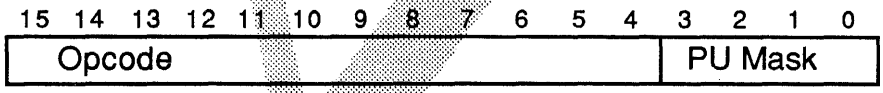
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if executed in user mode.

Restrictions: None.

Timing: The issuing PU blocks if any target PU is not trap-enabled.

Format:



Example:

ITLB Invalidate Translation Lookaside Buffer ITLB

Operation: ITLB

Description: Invalidate the translation lookaside buffer.

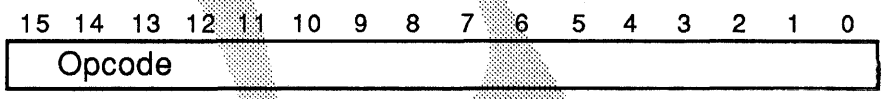
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Jmp — Jump Relative — Jmp

Operation: **Jmp PC + Displacement**

Description: Continue execution at the instruction whose halfword address is **PC + Displacement** after executing the next sequential instruction. **Displacement** is a 12-bit signed quantity (11 bits + sign bit).

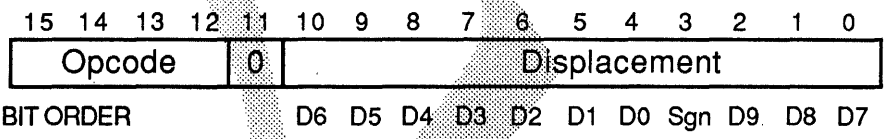
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



DISPLACEMENT BIT ORDER D6 D5 D4 D3 D2 D1 D0 Sgn D9 D8 D7

Example:

Jmp

Jump Absolute

Jmp

Operation: **Jmp @RegA**

Description: Continue execution at the instruction whose halfword address is in **RegA** after executing the next sequential instruction.

This instruction can be used to return from a subroutine.

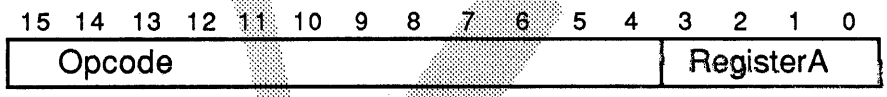
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

JmpL — Jump and Link — JmpL

Operation: **JmpL next PC + 1 → R4, Jmp @RegA**

Description: Store the halfword address of **next PC + 1** in **R4**, then continue execution at the instruction whose address is in **RegA** after executing the next sequential instruction.

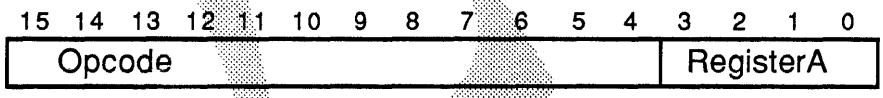
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Lcc Load Condition Lcc

Operation: **Lcc → RegB**

Description: If the condition codes match **cc**, then store 1 into **RegB**. If the condition codes do not match **cc**, store 0 into **RegB**.

Table 8. CC Field Encodings

Encoding	cc	Condition	Meaning
0		reserved	
1	EQ	Z=1	equal
2	LT	N=1	less than
3	LE	N=1 or Z=1	less or equal
4	LO	C=0	lower
5	LS	C=0 or Z=1	less or same
6	OV	V=1	overflow
7		reserved	
8		reserved	
9	NE	Z=0	not equal
10	GE	N=0	greater or equal
11	GT	N=0 and Z =0	greater than
12	HS	C=1	higher or same
13	HI	C=1 and Z=0	higher
14	NV	V=0	no overflow
15		reserved	

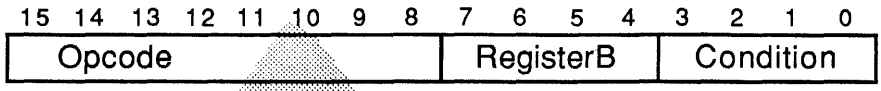
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Ld — Load Immediate — Ld

Operation: **Ld Imm → RegB**

Description: Load the immediate value **Imm** into **RegB**, clearing the upper 23 bits of **RegB**. **Imm** must be in the range 1 through 256. **Imm** is encoded (**Imm** - 1).

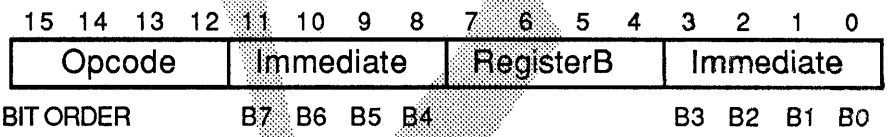
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



DISPLACEMENT BIT ORDER B7 B6 B5 B4 B3 B2 B1 B0

Example:

LdB — Load Byte — LdB

Operation: **LdB @Base → RegA**

Description: Load the value contained in the Memory byte addressed by **Base** into **RegA**. Right-justify the value, clearing the upper 24 bits. Then increment the byte address by one and store it back into **Base**. The byte number is contained in bits 31 and 30 of **Base.**

If a page fault interrupt occurs, then **RegA** must be restored to the value it had at the start of instruction execution to correctly re-execute the instruction upon return from the page fault handler.

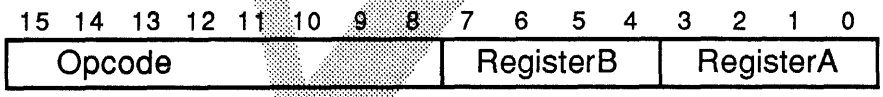
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes an additional cycle if **RegA** is used in the immediately following instruction and another additional cycle if used immediately following any Store instruction.

Format:



Example:

LdCP Load Carry Partial LdCP

Operation: **LdCP Extend ~(C0..C3) → RegA**

Description: Ones complement and sign extend the carry status bits and store in **RegA.**

In byte mode, sign extend each byte's carry bit to 8 bits and store them in **RegA.**

In halfword mode, sign extend each halfword's carry bit to 16 bits and store them in **RegA.**

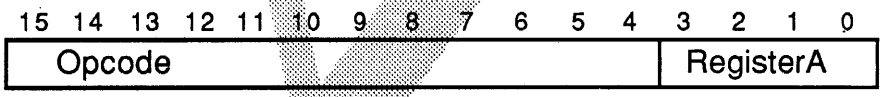
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

LdM — Load Multiple — LdM

Operation: **LdM @RegA → RegB..R1**

Description: Load registers, starting at **RegB** and ending at register **R1**, into the Main Memory location whose address is contained in **RegA**. Increment **RegA** by **RegB** after the last load. The encoding for **RegA** is (**RegA** - 1).

LdM is not interruptible except by page or data access fault traps. If the trap occurs on the load of **R1**, then **RegA** must be restored to the original value it had when the instruction started to correctly re-execute the instruction upon return from the page fault handler.

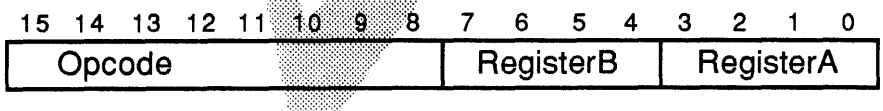
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: The result is undefined if **RegA** is in the range 1 through **RegB**.

Timing: This instruction takes one cycle per register loaded and one more cycle if used in the following instruction, plus an additional cycle if used following any Store instruction.

Format:



Example:

LdW — Load Word — LdW

Operation: **LdW @Base + Displacement → RegB**

Description: Load the Memory word whose address is register **Base** + **Displacement** into **RegB**. Register **Base** must be in the range 0 through 3. **Displacement** must be in the range 1 through 64.

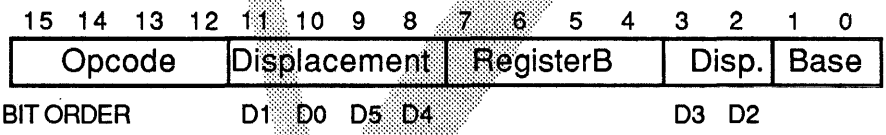
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes an additional cycle if **RegB** is used in the immediately following instruction and another additional cycle if used immediately following any Store instruction.

Format:



DISPLACEMENT BIT ORDER D1 D0 D5 D4 D3 D2

Example:

LdW — Load Word Direct — LdW

Operation: **LdW @Displacement → RegB**

Description: Load the Memory word whose word address is (**Window (Cluster#) + Displacement**) into **RegB**. **Displacement** must be in the range 0 through 255.

The **Cluster#** is a bit in the **System Status/Control** register that selects a window register prefixed to **Displacement**. If **Cluster#** is clear, the instruction uses the fixed, read-only window register that contains the constant 0x3FF00000.

If **Cluster#** is set, the instruction uses the other window register, which can be modified using a Move to Special (Mov) instruction.

Loads from addresses 0 through 7 freeze the processor if the associated semaphore bit is clear. The processor unfreezes when the semaphore bit is set. The semaphore bit is left cleared.

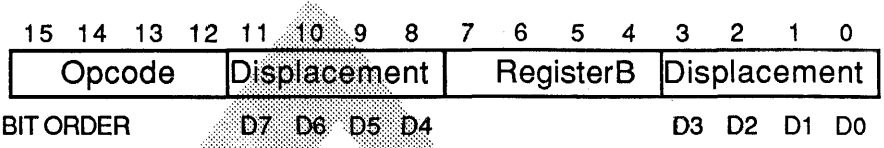
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes an additional cycle if **RegB** is used in the immediately following instruction and another additional cycle if used immediately following any Store instruction.

Format:



Example:

LdW — Load Word — LdW

Operation: **LdW @RegA → RegB**

Description: Load the Memory word whose address is in **RegA** into **RegB**.

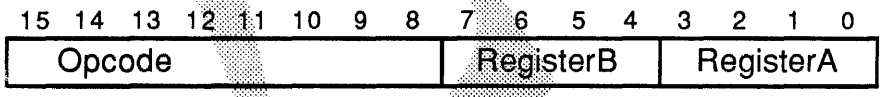
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes an additional cycle if **RegB** is used in the immediately following instruction and another additional cycle if used immediately following any Store instruction.

Format:



Example:

LdWE — Load Word Extended — LdWE

Operation: **LdWE @RegA + Displacement → RegB**

Description: Load the Memory word whose address is in **RegA** + **Displacement** into **RegB**. **RegA** can be any register. **Displacement** must be in the range 1 through 65536 and is encoded (**Displacement** -1).

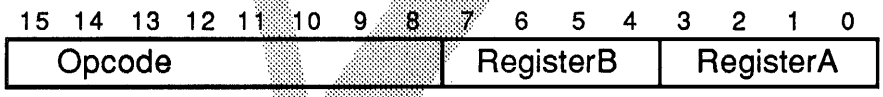
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

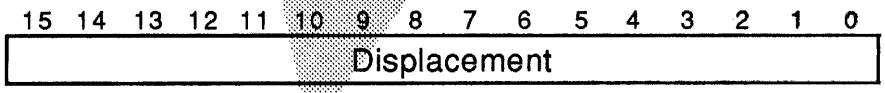
Restrictions: This instruction cannot be executed following an instruction that branches, including Bcc, Jmp, JmpL, and Skcc.

Timing: This instruction takes one or two cycles, plus an additional cycle if **RegB** is used in the immediately following instruction and another additional cycle if used immediately following any Store instruction.

Format:



(next halfword)



Example:

Mov — Move — Mov

Operation: **Mov RegA → RegB**

Description: Move the contents of **RegA** into **RegB**. **RegA** remains unchanged. Moving the contents of a register to the same register produces no effect.

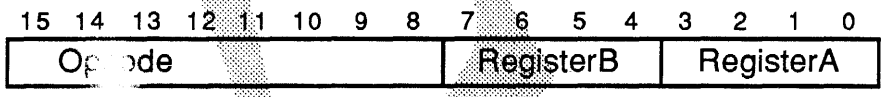
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Mov — Move to Special — Mov

Operation: **Mov RegB → Sp**

Description: Move the contents of **RegB** into special register **Sp**. The special registers and their encodings are:

0	Mask
1	Remainder
2	Product
3	reserved
4	Test †*
5	reserved
6	Window Register/Semaphores†
7	Status_Save (includes condition codes)*
8	Trap Argument*
9	PC_Save Queue*
10	Event Counter 1†
11	Event Counter 2†
12	Event Configuration†
13	reserved
14	Interrupt Argument (2 bits only)†*
15	Page Table Origin Node†*

†Indicates chip global
*Indicates privileged

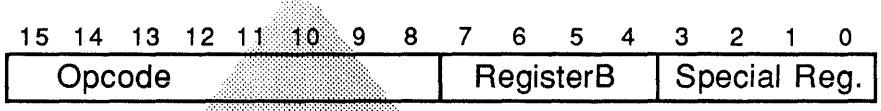
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if the **User/System** bit is set and the destination special register is privileged or if the destination special register is reserved.

Restrictions: Traps must be disabled when reading from or writing to the PC_Save Queue, Status_Save, or Trap Argument registers, and the external interrupt must be disabled when reading from or writing to the Interrupt Argument register.

Timing: This instruction takes one cycle.

Format:



Example:

Mov — Move from Special — Mov

Operation: **Mov Sp → RegB**

Description: Move the contents of special register **Sp** into **RegB**. The special registers and their encodings are:

0	Mask
1	Remainder
2	Product
3	PU#
4	Test †*
5	reserved
6	Window Register/Semaphores†
7	Status_Save (includes condition codes)*
8	Trap Argument*
9	PC_Save Queue*
10	Event Counter 1†
11	Event Counter 2†
12	Event Configuration†
13	Global Status
14	Interrupt Argument†*
15	Page Table Origin Node†*

†Indicates chip global
*Indicates privileged

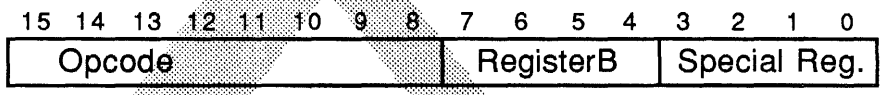
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A trap occurs if the **User Status** bit is set and the source special register is read privileged or if the source special register is reserved.

Restrictions: Traps must be disabled when reading from or writing to the PC_Save Queue, Status_Save, or Trap Argument registers. External interrupts must be disabled when reading from the Interrupt Argument register.

Timing: This instruction takes one cycle plus an additional cycle if the special register is a global register and **RegB** is used by the next instruction. Mov also takes two cycles if the special register is a global register and it is used following a Store instruction.

Format:



Example:

Msk — Mask Generate — Msk

Operation: **Msk** **Pos** for **Len**

Description: Generate a field definition with the right end of the field at bit position **Pos**, and with the left end at bit position min(31,**Pos**+**Len**-1). Store the definition in the **Mask** register. **Len** must be in the range 1 through 32 and is encoded (**Len** -1).

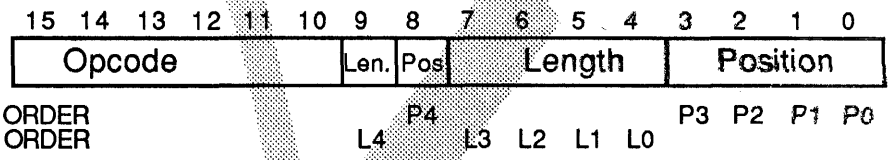
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Msk — Mask Generate — Msk

Operation: **Msk** **RegA** for **Len**

Description: Generate a field definition with the right end of the field at bit position **RegA**(modulo 32), and with the left end of the field at bit position min(31, **RegA**(modulo 32)+**Len**-1). Store the definition in the **Mask** register. **Len** must be in the range 1 through 32 and is encoded (**Len** -1).

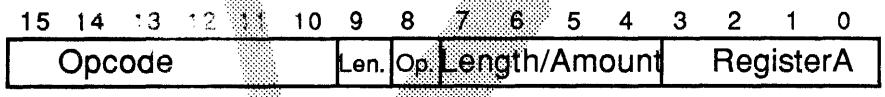
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



LENGTH/AMOUNT BIT ORDER L4 L3 L2 L1 L0

Example:

Mul — Multiply — Mul

Operation: **Mul RegB * RegA + (MDfull & Product) → {Product , RegB}**

Description: Multiply the contents of **RegB** times the contents of **RegA** and store the 64-bit result in registers **Product** and **RegA**. If **MDfull** is set, then add the contents of **Product** to the result of **RegB * RegA**. The upper 32 bits are returned in special register **Product**, and the lower 32 bits are returned in **RegB**. The operands are treated as signed(?) unsigned(?) numbers.

The **MDfull** bit is cleared after the first cycle of each Multiply (Mul) or Multiply Partial (MulP) instruction. The bit gets set by moving something into the **MD** register with a Move to Special (Mov) instruction, or by the Set Mode (SetM) instruction.

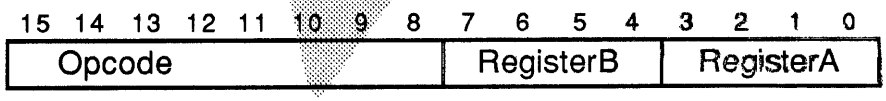
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: The result is not valid for five cycles.

Format:



Example:

Mul — Multiply — Mul

Operation: **Mul RegB * RegA → {Product , RegB}**

Pyxis Syntax:
```
mul >2 $1
$2 mul $1
$2 *= $1
```

Description: Multiply the contents of **RegB** times the contents of **RegA** and store the 64-bit result in registers **ProdR** and **RegA**. The upper 32 bits are returned in special register **ProdR**, and the lower 32 bits are returned in **RegB.**

The **Multiply Control** (**MC**) mode bit controls whether or not the mutiplicand is a signed quantity. If the **MC** bit is set, the Mul instruction uses the C0 carry bit to determine the sign of the multiplicand. If C0 contains a zero, the quantity is negative; if C0 contains a one, the quantity is positive. This affects only the upper half of the result.

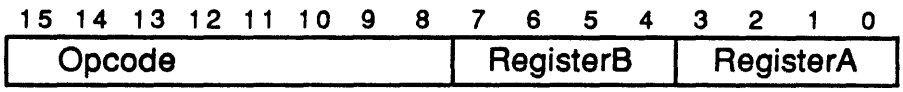
Condition codes:
N — Not affected.
Z — Not affected.
V — Not affected.
C — Not affected.

Exceptions: None.

Restrictions: None.

Timing: The result is not valid for five cycles.

Format:



MulP — Multiply Partial — MulP

Operation: **RegB * RegA → RegB**

Description: If the **H/B** bit in the **System Status/Control** register is clear, multiply **RegB** times **RegA**, inhibiting carries between bytes. If the **H/B** bit in the **System Status/Control** register is set, multiply **RegB** times **RegA**, inhibiting carries between halfwords.

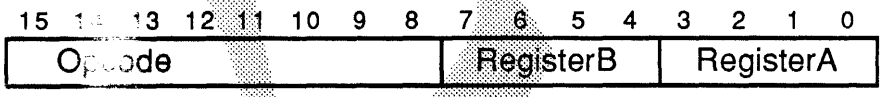
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes two cycles.

Format:



Example:

MulP — Multiply Partial — MulP

Operation: **RegB * RegA → ProdR**

Pyxis Syntax:

```
mulp >2 $1
$2 mulp $1
$2 *=| $1
```

Description: If the **H/B** bit in the ~~System~~ **Status/Control** register is clear, multiply **RegB** times **RegA**, inhibiting carries between bytes. If the **H/B** bit in the ~~System~~ **Status/Control** register is set, multiply **RegB** times **RegA**, inhibiting carries between halfwords. Store the result in the special register **ProdR**. The multiplier (**RegA**) is always a 16-bit value—only the lower half of the register is used.

The **Multiply Control** (**MC**) mode bit controls whether or not the mutiplicand is a signed quantity. If the **MC** bit is set, the MulP instruction uses the corresponding carry bit to determine the sign of the multiplicand. In byte mode, carry bits C0, C1, C2, and C3 are used; in halfword mode, C0 and C2 are used. If the corresponding carry bit contains a zero, the quantity is negative; if the carry bit contains a one, the quantity is positive.

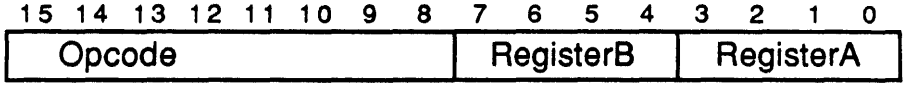
Condition codes:
N — Not affected.
Z — Not affected.
V — Not affected.
C — Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes three cycles.

Format:



Neg — Negate — Neg

Operation: **Neg - RegA → RegB**

Description: Twos complement the contents of **RegA** and store the result in **RegB**.

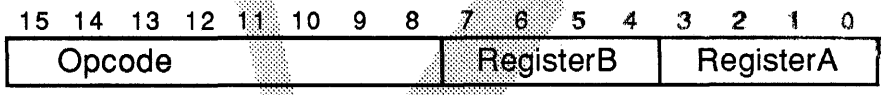
Condition codes:
N -- Set if **RegA** is positive, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- Set only if the result is -2^{31}.
C -- Set if result equals zero, cleared otherwise.
(C = *not* borrow)

Exceptions: A trap occurs if the operation sets the V condition code bit and Trap_on_Overflow is enabled.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Not

Not

Not

Operation: **Not ~RegA → RegB**

Description: Ones complement the contents of **RegA** and store the result in **RegB**.

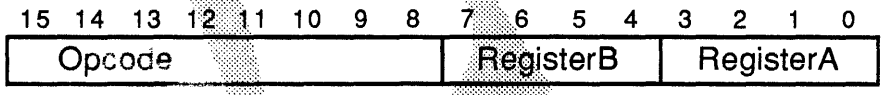
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Or

Or

Or

Operation: **Or RegA | RegB → RegB**

Description: Logically OR the contents of **RegB** with the contents of **RegA**. Store the result in **RegB**.

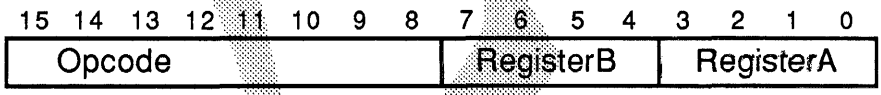
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

PDC Prefetch Data Cache Line PDC

Operation: **PDC @RegA**

Description: Transfer a line containing the address register **RegA** from Main Memory to the data cache. The transfer occurs only if there are no other transfers in progress and if the effective address (**RegA**) is not currently in the data cache.

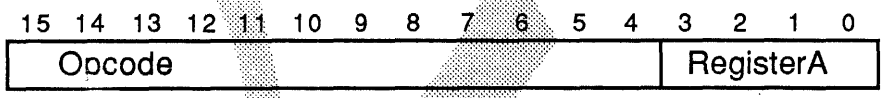
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

PIC Prefetch Instruction Cache Line PIC

Operation: **PIC @RegA**

Description: Transfer a line containing the address register **RegA** from Main Memory to the instruction cache. The transfer occurs only if there are no other transfers in progress and if the effective address (**RegA**) is not currently in the instruction cache.

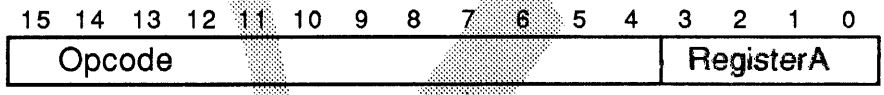
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

PIC Prefetch Instruction Cache Line PIC

Operation: **PIC PC + Displacement**

Description: Transfer a line containing **PC + Displacement**. The transfer occurs only if there are no other transfers in progress and if the effective address (**RegA**) is not currently in the instruction cache. **Displacement** is a 9-bit (8 bits plus sign bit) quantity in the range -256 through +255.

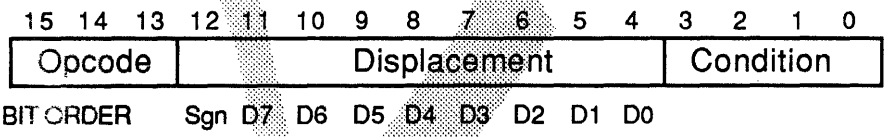
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



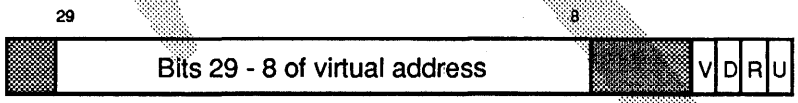
DISPLACEMENT BIT ORDER Sgn D7 D6 D5 D4 D3 D2 D1 D0

Example:

RDTX Read Data Tag by Index RDTX

Operation: **RDTX @RegA → RegB**

Description: Read the tag entry corresponding to the address in **RegA** and store it into **RegB**. The RDTX instruction is implementation-dependent. This implementation uses **RegA**<7:4> to select one of 16 sets and **RegA**<0:1> to select one of four associations. The figure below shows the format of the data returned.



V Valid bit.
R Read-only bit. Derived from page table entry.
D Dirty bit. Set on a write to the line.
U User/System bit. Derived from page table entry and compared to PU state.

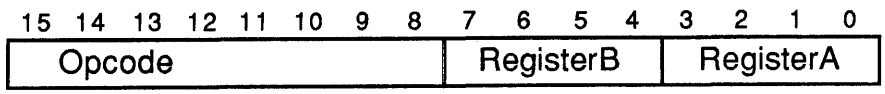
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if executed in user mode.

Restrictions: None.

Timing: This instruction takes two cycles, plus an additional cycle if **RegB** is used in the next instruction.

Format:

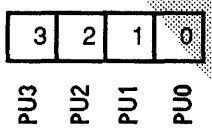


Example:

Res Restart Res

Operation: **Res PuMask**

Description: Cause each PU whose PuMask bit is set to trap to the Restart interrupt vector 0x1D. The PuMask bits correspond to the PUs as shown below.



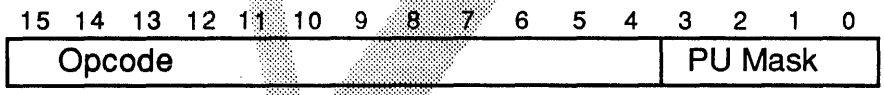
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:

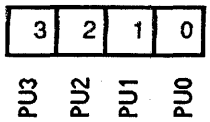


Example:

Rsm — Resume — Rsm

Operation: **Rsm PuMask**

Description: Cause each PU whose associated PuMask bit is set to clear the Register Available bit and continue execution at the next instruction. Resume causes processing to resume following a Wait instruction. The PuMask bits correspond to the PUs as shown below.



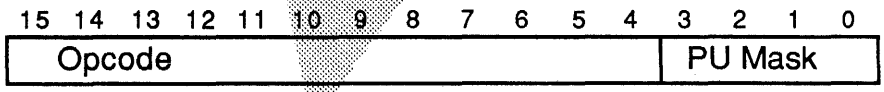
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: The issuing PU blocks if any target PU, except itself, is running, or if any target PU is in system state when the issuing PU is in user state.

Timing: This instruction takes one cycle.

Format:



Example:

RtI — Return from Interrupt — RtI

Operation: RtI

Description: Restore the **System Status/Control** register from the **Status_Save** register, advance the **PC_Save Queue**, and continue execution at the address which was at the head of the **PC_Save Queue**, after executing the next sequential instruction.

Interrupts are disabled while RtI is processed.

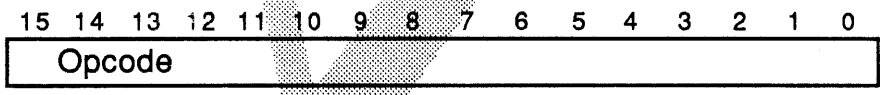
Condition codes:
N -- Restored from the Status_Save register.
Z -- Restored from the Status_Save register.
V -- Restored from the Status_Save register.
C -- Restored from the Status_Save register.

Exceptions: A privileged operation trap occurs if executed in user mode.

Restrictions: Two consecutive RtI instructions must be executed to return from an interrupt. The result of RtI is undefined if executed following an instruction that alters the flow of control, including Bcc, Jmp, JmpL, and Skcc. Traps must be disabled when executing RtI.

Timing: This instruction takes one cycle.

Format:



Example:

SbCP — Subtract with Carry Partial — SbCP

Operation: **SbCP RegB (H/B) - RegA → RegB**

Description: Subtract **RegA** from **RegB**, forcing carryins to be associated with the appropriate byte and halfword carry bits.

Condition codes: **H/B = 0**

N -- Set if any byte is negative, cleared otherwise.
Z -- Set if any byte is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any byte carries, cleared otherwise.

H/B = 1

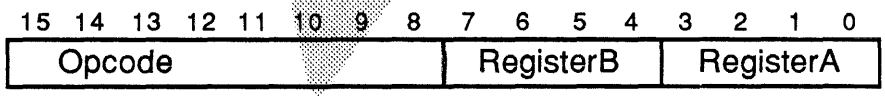
N -- Set if any halfword is negative, cleared otherwise.
Z -- Set if any halfword is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any halfword carries, cleared otherwise.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

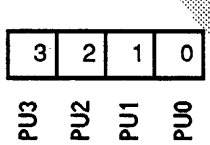
Send

Send

Send

Operation: **Send RegB → Destination Register<PuMask>**

Description: Move the contents of **RegB** into the destination register of each of the target PUs.The PuMask bits correspond to the PUs as shown below.



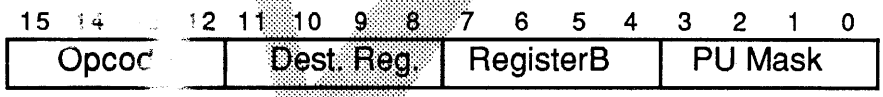
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: The PU blocks if any target PU except itself is running or if any target PU is in system state when the issuing PU is in user state.

Timing: This instruction takes one cycle.

Format:



Example:

SetF — Set Field — SetF

Operation: **SetF 1 → RegA<Mask>**

Description: Set the bits of the field in **RegA** defined by the **Mask** register to ones and store the result in **RegA**. Bits outside the field are unaffected.

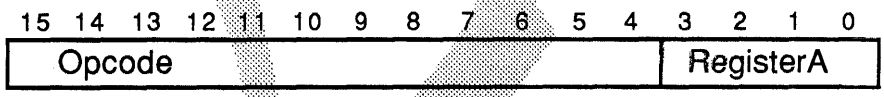
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special (Mov) instruction.

Timing: This instruction takes one cycle.

Format:



Example:

SetM — Set Mode — SetM

Operation: **SetM BitNumber**

Description: Set or clear the specified mode bit in the **System Status/Control** Register. The mode bits are:

0	MD control
1	Halfword/Byte
2	MD full
3	Overflow trap enable
4	Register available
5	reserved
6	reserved
7	reserved
8 -9	reserved
10	Branch taken trap enable†
11	PU trap enable/disable†
12	User/system†
13	Cluster number†
14	reserved
15	reserved

†Indicates privileged

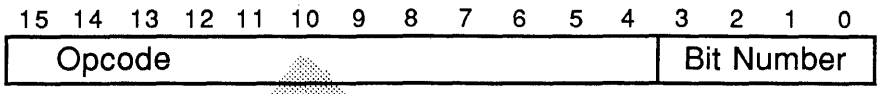
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if the system is in user mode and an attempt is made to set bits 8 through 15. An illegal operation trap occurs if an attempt is made to set bits 5, 6, 7, 8,9,14, or 15.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

ShL — Shift Left — ShL

Operation: **ShL RegB << Amt → RegB**

Description: Logically shift the contents of **RegB** left **Amt** places, shifting in zeroes. Store the result in **RegB**.

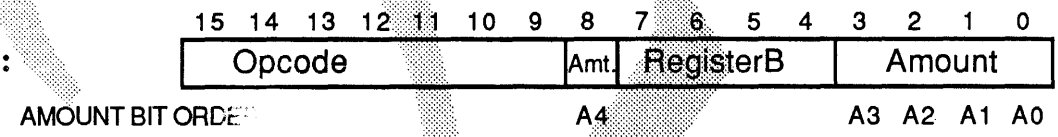
Condition codes:
N -- Set if the result is negative.
Z -- Set if the result equals zero.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: The result is undefined if **Amt** is zero. The encoding for **Amt** is (32-shift amount).

Timing: This instruction takes one cycle.

Format:



Example:

ShR — Shift Right — ShR

Operation: **ShR RegA >> Amt → RegA**

Description: Logically shift the contents of **RegA** right **Amt** places, shifting in zeroes. Store the result in **RegA**.

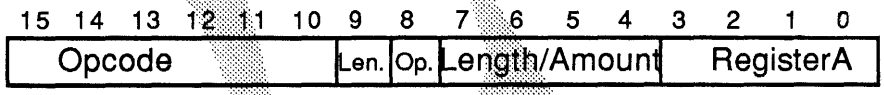
Condition codes:
N -- The N bit is cleared.
Z -- Set if the result equals zero, cleared otherwise.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: The result is undefined if **Amt** is zero.

Timing: This instruction takes one cycle.

Format:



LENGTH/AMOUNT BIT ORDER L4 L3 L2 L1 L0

Example:

Skcc — Skip on Condition — Skcc

Operation: Skcc

Description: If the condition **cc** specified by the instruction is met, skip (do not execute) the next sequential instruction. If the condition is not met, execute the next sequential instruction.

The encodings and meanings of the **cc** field are shown in the table below.

Table 9. CC Field Encodings

Encoding	cc	Condition	Meaning
0		reserved	
1	EQ	Z=1	equal
2	LT	N=1	less than
3	LE	N=1 or Z=1	less or equal
4	LO	C=0	lower
5	LS	C=0 or Z=1	less or same
6	OV	V=1	overflow
7		reserved	
8		reserved	
9	NE	Z=0	not equal
10	GE	N=0	greater or equal
11	GT	N=0 and Z =0	greater than
12	HS	C=1	higher or same
13	HI	C=1 and Z=0	higher
14	NV	V=0	no overflow
15		reserved	

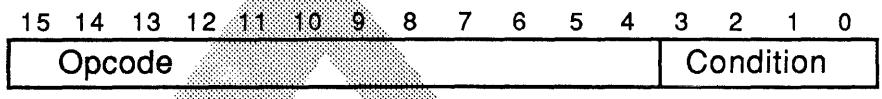
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes two cycles if the condition contained in **cc** is met.

Format:



Example:

StB — Store Byte — StB

Operation: **StB RegB → @RegA**

Description: Store the low byte of **RegB** into the Memory byte whose address is in **RegA**, then increment the byte address by one and store it back in **RegA**. The other bytes of the memory word containing the destination remain unchanged. The byte number is contained in bits 31 and 30 of **RegA.**

If a data access or page fault trap occurs, then **RegB** must be restored to the value it had at the start of instruction execution to correctly re-execute the instruction upon return from the page fault handler.

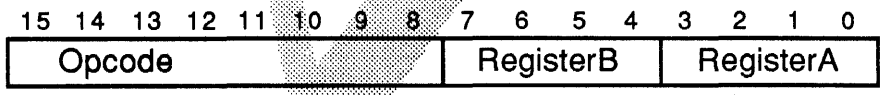
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A a data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

StM — Store Multiple — StM

Operation: StM RegB..R1 → @RegA

Description: Store registers, starting at **RegB** and ending at register **R1** into the Main Memory location whose address is in **RegA**. Decrement **RegA** by 1 before each store.

StM is not interruptible except by data access fault and page fault traps. If a fault occurs, then **RegB** must be restored to the original value it had at the start of instruction execution to correctly re-execute the instruction upon return from the page fault handler.

The **RCnt** field of the **Status_Save** register holds the number of the last register stored.

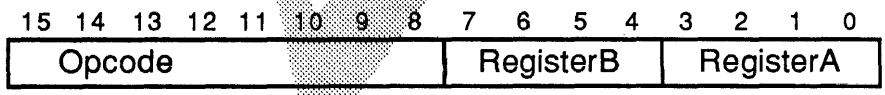
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: The result is undefined if **RegA** is in the range 1 through **RegB.**

Timing: This instruction takes one cycle per register stored.

Format:

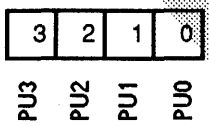


Example:

Strt — Start — Strt

Operation: **Strt RegB, PuMask**

Description: Cause each PU whose associated PuMask bit is set to clear its REgister Available bit and resume execution at the halfword address contained in **RegB**. The PuMask bits correspond to the PUs as shown below.



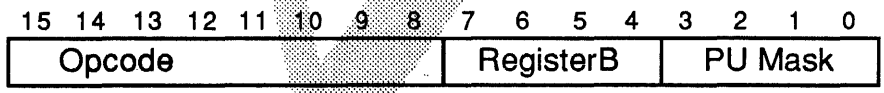
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: The issuing PU blocks if any target PU except itself is running or if any target PU is in system state when the issuing PU is in user state.

Timing: This instruction takes one cycle.

Format:



Example:

StW Store Word StW

Operation: **StW RegB → @Base + Displacement**

Description: Store the contents of **RegB** into the Memory word whose address is in register **Base + Displacement**. Register **Base** must be in the range 0 through 3. **Displacement** must be in the range 1 through 64 and is encoded (**Displacement** -1).

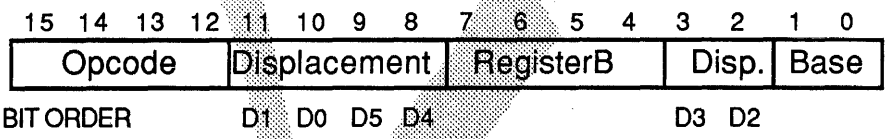
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

StW — Store Word Direct — StW

Operation: **StW RegB → @Displacement**

Description: Store the contents of **RegB** into the Memory word whose address is in **(Window (Cluster#) + Displacement)**.

The **Cluster#** is a bit in the **System Status/Control** register that selects a window register to be prefixed to **Displacement**. If **Cluster#** is clear, the instruction uses the fixed, read-only window register that contains the constant 0x3FF00000. If **Cluster#** is set, the instruction uses the other window register, which can be modified using a Move to Special (Mov) instruction.

Stores into addresses 0 through 7 freeze the processor if the associated semaphore bit is set. The processor unfreezes when the semaphore bit is cleared. The semaphore bit is left set.

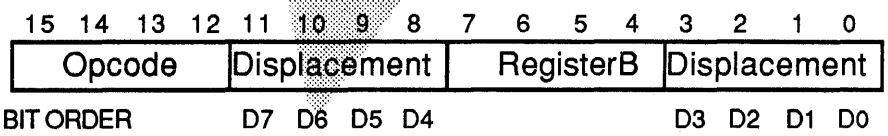
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

StW — Store Word — StW

Operation: **StW RegB → @Base**

Description: Store the contents of **RegB** in the Memory word whose address is in register **Base**.

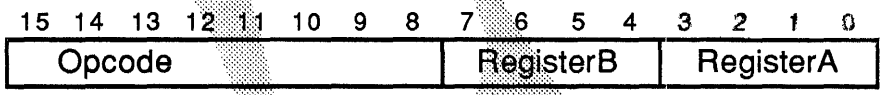
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

StWE Store Word Extended StWE

Operation: **StWE RegB → @Base + Displacement**

Description: Store the contents of **RegB** into the Memory word whose address is in register **Base + Displacement**. **Displacement** must be in the range 1 through 65536 and is encoded (**Displacement** - 1).

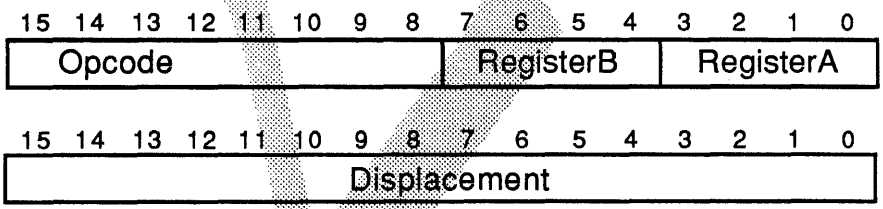
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access or page fault may occur if executed in user mode.

Restrictions: This instruction cannot be executed following an instruction that branches, including Bcc, Jmp, JmpL, and Skcc.

Timing: This instruction takes one or two cycles.

Format:



Example:

Sub — Subtract — Sub

Operation: **Sub RegB - RegA → RegB**

Description: Subtract the contents of **RegA** from the contents of **RegB**. Store the result in **RegB**.

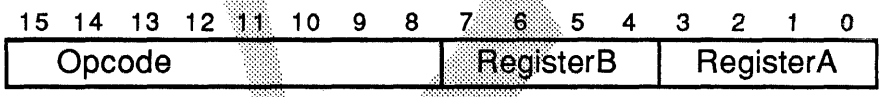
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
V -- Set if an overflow is generated, cleared otherwise.
C -- Set if a carry is generated, cleared otherwise.
(C = *not* borrow.)

Exceptions: A trap occurs if the V condition code bit is set and Trap_on_Overflow is enabled.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Sub — Subtract Immediate — Sub

Operation: **Sub RegB - Imm → RegB**

Description: Subtract the immediate value **Imm** from the contents of **RegB** and store the result in **RegB**. The range for **Imm** is 1 through 16, and **Imm** is encoded (**Imm** -1).

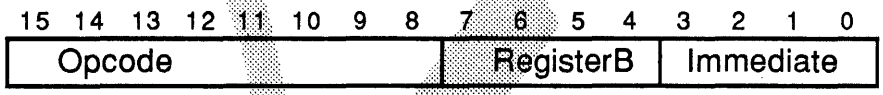
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
C -- Set if a carry is generated, cleared otherwise.
V -- Set if an overflow is generated, cleared otherwise.
(C = borrow.)

Exceptions: A trap occurs if the V condition code bit is set and Trap_on_Overflow is enabled.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



IMMEDIATE BIT ORDER B3 B2 B1 B0

Example:

SubC — Subtract with Carry — SubC

Operation: **SubC RegB - RegA + (Carry - 1) → RegB**

Description: Add the contents of the C0 carry status bit and the ones complement of the contents of **RegA** to the contents of **RegB**. Store the result in **RegB**.

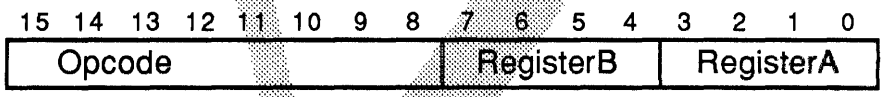
Condition codes:
N -- Set if the result is negative, cleared otherwise.
Z -- Set if the result equals zero, cleared otherwise.
C -- Set if a carry is generated, cleared otherwise.
V -- Set if an overflow is generated, cleared otherwise.
(C = *not* borrow.)

Exceptions: A trap occurs if the V condition code bit is set and Trap_on_Overflow is enabled.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

SubP — Subtract Partial — SubP

Operation: **SubP RegB (H/B) - RegA → RegB**

Description: If the **H/B** bit in the **System Status/Control** register is clear, subtract **RegA** from **RegB**, forcing all byte carryins to 1. If the **H/B** bit in the **System Status/Control** register is set, subtract **RegA** from **RegB**, forcing all halfword carryins to 1. Store the result in **RegB**.

Condition codes:

H/B = 0

N -- Set if any byte is negative, cleared otherwise.
Z -- Set if any byte is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any byte carries, cleared otherwise.

H/B = 1

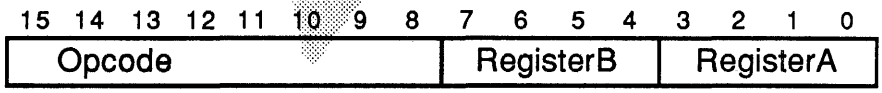
N -- Set if any halfword is negative, cleared otherwise.
Z -- Set if any halfword is equal to zero, cleared otherwise.
V -- The V bit is cleared.
C -- Set if any halfword carries, cleared otherwise.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

Trap

Trap

Trap

Operation: **Trap TrapNum**

Description: Move the contents of the **System Status/Control** register into the **Status_Save** register, move **next PC** and **next PC + 1** into the **PC_Save** queue, clear the **System Status/Control** register, and continue execution at address **(8 * TrapNum)**.

The Introduction lists the traps used by the Antares CPU.

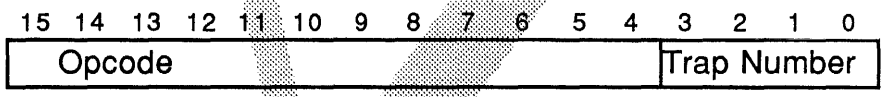
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None:

Timing: The Trap instruction takes two cycles.

Format:



Example:

TstF — Test Field — TstF

Operation: **TstF RegA <Mask>**

Description: Test the bits of the field in **RegA**, as defined by the **Mask** register, for zero and negative. Set the condition codes accordingly.

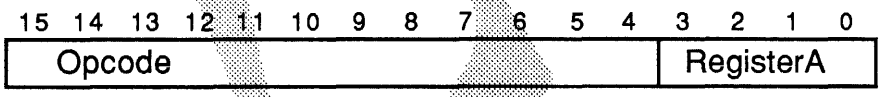
Condition codes:
N -- Set if the field bits are negative, cleared otherwise.
Z -- Set if the field bits are zero, cleared otherwise.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

Restrictions: The result is undefined if an illegal field definition has been moved into the **Mask** register with a Move to Special (Mov) instruction.

Timing: This instruction takes one cycle.

Format:



Example:

TstM Test Mode TstM

Operation: **TstM BitNumber**

Description: Test the specified bit in the **Status_Save** register and set the Carry condition code bit to the same value. The bits that can be tested include:

0	MD control
1	Halfword/Byte
2	MD full
3	Overflow trap enable
4	Register available
5	reserved
6	reserved
7	reserved
8 -9	reserved
10	Branch taken trap enable†
11	PU trap enable/disable†
12	User/system†
13	Cluster number†
14	reserved
15	reserved

†Indicates privileged.

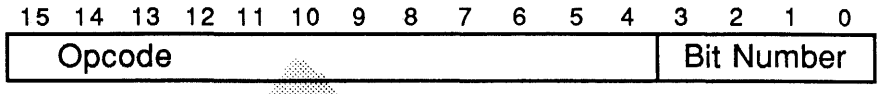
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A privileged operation trap occurs if the system is in user mode and any one of mode bits 8 through 15 is specified. An illegal operation trap occurs if mode bit 5, 6, 7, 8, 9, 14, or 15 is specified.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

UDC Update Data Cache Line UDC

Operation: **UDC @RegA**

Description: If the cache line addressed by **RegA** is in the data cache and has been modified, write it back to Main Memory and set it unmodified.

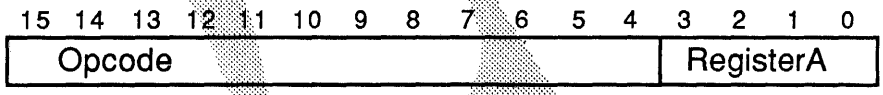
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

VDC Validate Data Cache Line VDC

Operation: VDC @RegA

Description: If the line addressed by **RegA** is in the data cache and has been modified, set it to unmodified.

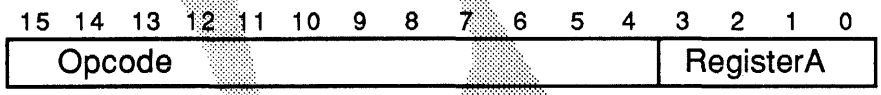
Condition codes:
N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: A data access fault may occur if executed in user mode.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

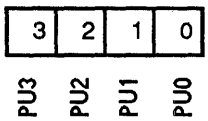
Wait

Wait

Wait

Operation: **Wait PuMask**

Description: Wait until each PU whose associated PuMask is bit set is waiting, then continue execution. A Resume (Rsm) or Start (Strt) instruction will also restart a PU. If no PuMask bits are set, halt the PU. The PuMask bits correspond to the PUs as shown below.



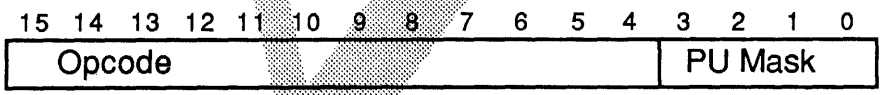
Condition codes: N -- Not affected.
Z -- Not affected.
V -- Not affected.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example:

XOr — Exclusive OR — XOr

Operation: **XOr RegA ^ RegB → RegB**

Description: Logically EXCLUSIVE OR the contents of **RegB** with the contents of **RegA**. Store the result in **RegB**.

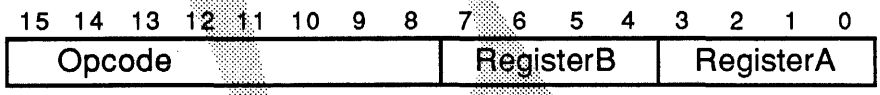
Condition codes:
N -- Set if the result is negative.
Z -- Set if the result is zero.
V -- The V bit is cleared.
C -- Not affected.

Exceptions: None.

Restrictions: None.

Timing: This instruction takes one cycle.

Format:



Example: