Thor Guide

[Type the document subtitle]

[Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.]

2013

robfinch@Finitron.ca

Finitron

1/1/2013

Table of Contents

[Programming Model 5](#_Toc371985308)

[General Registers 5](#_Toc371985309)

[Predicates 6](#_Toc371985310)

[Predicate Conditions 6](#_Toc371985311)

[Segmentation 7](#_Toc371985312)

[Software Support 7](#_Toc371985313)

[Address Formation: 7](#_Toc371985314)

[Selecting a segment register 7](#_Toc371985315)

[Non-Segmented Code Area 7](#_Toc371985316)

[Changing the Code Segment 7](#_Toc371985317)

[Segment Usage Conventions 8](#_Toc371985318)

[Power-up State 8](#_Toc371985319)

[Affect on I/O Access 8](#_Toc371985320)

[Vectors 9](#_Toc371985321)

[Vector table: 9](#_Toc371985322)

[Instruction Formats 10](#_Toc371985323)

[RR - Register-Register 10](#_Toc371985324)

[RI - Register-Immediate 10](#_Toc371985325)

[CMP Register-Register Compare 10](#_Toc371985326)

[CMPI Register-Immediate Compare 10](#_Toc371985327)

[TST - Register Test Compare 10](#_Toc371985328)

[CTRL- Control 10](#_Toc371985329)

[BR - Relative Branch 10](#_Toc371985330)

[BRK/NOP 10](#_Toc371985331)

[JSR - Jump To Subroutine 10](#_Toc371985332)

[Instruction Set 12](#_Toc371985333)

[2ADDU - Register-Register 12](#_Toc371985334)

[2ADDUI - Register-Immediate 12](#_Toc371985335)

[4ADDU - Register-Register 13](#_Toc371985336)

[4ADDUI - Register-Immediate 13](#_Toc371985337)

[8ADDU - Register-Register 14](#_Toc371985338)

[8ADDUI - Register-Register 14](#_Toc371985339)

[16ADDU - Register-Register 15](#_Toc371985340)

[16ADDUI - Register-Register 15](#_Toc371985341)

[ADD - Register-Register 16](#_Toc371985342)

[ADDI - Register-Immediate 16](#_Toc371985343)

[ADDU - Register-Register 16](#_Toc371985344)

[ADDUI - Register-Immediate 16](#_Toc371985345)

[AND - Register-Register 17](#_Toc371985346)

[ANDI - Register-Immediate 17](#_Toc371985347)

[BR - Relative Branch 18](#_Toc371985348)

[BRK –Break 19](#_Toc371985349)

[BSR - Branch To Subroutine 20](#_Toc371985350)

[CLI – Clear Interrupt Mask 21](#_Toc371985351)

[CMP Register-Register Compare 22](#_Toc371985352)

[CMPI Register-Immediate Compare 22](#_Toc371985353)

[EOR - Register-Register 23](#_Toc371985354)

[EORI - Register-Immediate 23](#_Toc371985355)

[IMM64,IMM56,IMM48,IMM40,IMM32,IMM24,IMM16 24](#_Toc371985356)

[Immediate Extensions 24](#_Toc371985357)

[INT –Interrupt 25](#_Toc371985358)

[JMP - Jump To Address 26](#_Toc371985359)

[JSR - Jump To Subroutine Instruction 26](#_Toc371985360)

[LB – Load Byte 27](#_Toc371985361)

[LBX – Load Byte Indexed 27](#_Toc371985362)

[LBU – Load Byte Unsigned 28](#_Toc371985363)

[LBUX – Load Byte Unsigned Indexed 28](#_Toc371985364)

[LC – Load Character 29](#_Toc371985365)

[LCX – Load Character Indexed 29](#_Toc371985366)

[LCU – Load Character Unsigned 30](#_Toc371985367)

[LCUX – Load Character Indexed 30](#_Toc371985368)

[LDI - Load-Immediate 31](#_Toc371985369)

[LH – Load Half-Word 32](#_Toc371985370)

[LHX – Load Walf-word Indexed 32](#_Toc371985371)

[LHU – Load Half-word Unsigned 33](#_Toc371985372)

[LHUX – Load Half-word Unsigned Indexed 33](#_Toc371985373)

[LOOP – Loop Branch 34](#_Toc371985374)

[LVB – Load Volatile Byte 35](#_Toc371985375)

[LVC – Load Volatile Character 35](#_Toc371985376)

[LVH – Load Volatile Half-word 35](#_Toc371985377)

[LVW – Load Volatile Word 36](#_Toc371985378)

[LW – Load Word 36](#_Toc371985379)

[LWX – Load Word Indexed 36](#_Toc371985380)

[MFSPR – Special Register-Register 37](#_Toc371985381)

[MTSPR –Register-Special Register 37](#_Toc371985382)

[NEG - Negate Register 38](#_Toc371985383)

[NOP – No Operation 39](#_Toc371985384)

[OR - Register-Register 40](#_Toc371985385)

[ORI - Register-Immediate 40](#_Toc371985386)

[RTE – Return From Exception Routine 41](#_Toc371985387)

[*Description* 41](#_Toc371985388)

[RTI – Return From Interrupt Routine 41](#_Toc371985389)

[*Description* 41](#_Toc371985390)

[RTS – Return From Subroutine 42](#_Toc371985391)

[*Description* 42](#_Toc371985392)

[*Description* 42](#_Toc371985393)

[SB – Store Byte 43](#_Toc371985394)

[SBX – Store Byte Indexed 43](#_Toc371985395)

[SC – Store Character 44](#_Toc371985396)

[SCX – Store Character Indexed 44](#_Toc371985397)

[SEI – Set Interrupt Mask 45](#_Toc371985398)

[SH – Store Half-word 46](#_Toc371985399)

[SHX – Store Half-word Indexed 46](#_Toc371985400)

[STSB – Store String Byte 47](#_Toc371985401)

[STSW – Store String Word 47](#_Toc371985402)

[SW – Store Word 48](#_Toc371985403)

[SWX – Store Word Indexed 48](#_Toc371985404)

[SUB - Register-Register 49](#_Toc371985405)

[SUBI - Register-Immediate 49](#_Toc371985406)

[SUBU - Register-Register 50](#_Toc371985407)

[SUBUI - Register-Immediate 50](#_Toc371985408)

[SYNC – Synchronize Memory 51](#_Toc371985409)

[SYS –Call system routine 52](#_Toc371985410)

[TST - Register Test Compare 53](#_Toc371985411)

# Programming Model

## General Registers

There are 256 general purpose registers. General purpose registers are 64 bits wide.

Register #0 is always zero.

**Branch Registers**

The processor contains 16 branch registers (BR0-BR15). Several of branch registers are reserved for predefined purposes.

|  |  |  |
| --- | --- | --- |
| Reg # |  | Usage |
| 0 | Always Zero | Absolute address formation |
| 1 |  | Subroutine return address |
| 2 |  | This register is available for general use. |
| 3 |  | This register is available for general use. |
| 4 |  | This register is available for general use. |
| 5 |  | This register is available for general use. |
| 6 |  | This register is available for general use. |
| 7 |  | This register is available for general use. |
| 8 |  | This register is available for general use. |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 | Exception Table Pointer | This register points to the exception table in memory. |
| 13 | Exceptioned PC | This register is set when an exception occurs |
| 14 | Interrupted PC | This register is automatically set during a hardware interrupt |
| 15 | Program Counter |  |
|  |  |  |

## Predicates

The processor features predicated execution of all instructions. Whether or not an instruction is executed depends on the contents of a predicate register and the predicate condition specified in the predicate byte. There are 16 predicate registers each of which hold three flags. These flags are set as the result of a compare operation. The flags represent equality (eq) signed less than (lt) and unsigned less than (ltu). It should be noted that the compare instruction can’t overflow.

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | 2 | 1 | 0 |
| ~ | ltu | lt | eq |

All instructions are executed conditionally determined by the value of a predicate register. The special predicate 00 executes the break vector.

### Predicate Conditions

|  |  |  |  |
| --- | --- | --- | --- |
| Cond. |  | Test |  |
| 0 | PF | 0 | Always false – Instructions predicated with condition zero never execute regardless of the predicate register contents. This is used for extended immediate values as well. |
| 1 | PT | 1 | Always True – The instruction predicated with an always true condition always executes regardless of the predicate register contents. |
| 2 | PEQ | eq | Equal – instruction executes if the predicate register equal flag is set |
| 3 | PNE | !eq | Not Equal – instruction executes if the predicate register equal flag is clear |
| 4 | PLE | lt|eq | Less or Equal – predicate less or equal flag is set |
| 5 | PGT | !(lt|eq) | greater than |
| 6 | PGE | !lt | greater or equal |
| 7 | PLT | lt | less than |
| 8 | PLEU | ltu|eq | unsigned less or equal |
| 9 | PGTU | !(ltu|eq) | unsigned greater than |
| 10 | PGEU | !ltu | unsigned greater or equal |
| 11 | PLTU | ltu | unsigned less than |
|  |  |  |  |

# 

# Segmentation

The processor contains sixteen segment registers. The upper nibble of an address (bits 60 to 63) identifies which segment register to use during address formation for data addresses. For code addresses segment register #15 is always used.

* If segmentation is not desired then segmentation can effectively be ignored by setting all the segment registers to zero. The processor can also be built without segmentation by commenting out the ‘SEGMENTATION’ definition.

## Software Support

Segment registers may only be transferred to or from one of the general purpose registers. The mtspr and mfspr instructions can be used to perform the move.

## Address Formation:

Non-segmented address bits 0 to 11 pass through the segmentation module unchanged. Address bits 59 to 12 are added to the contents of the segment register to form the final segmented address. Address bits 60 to 63 identify the segment register to use. Note that there is no shift associated with the segment addition. Future implementations of the processor may include additional low order address bits in the segment register in order to allow a finer grain for memory page / paragraph size.

|  |  |
| --- | --- |
| Address[59:12] | Address[11:0] |
| + | + |
| Segment register value[63:12] | 00012 |
| = | |
| Segmented address[63:0] | |

## Selecting a segment register

The upper nibble of an address (bits 60 to 63) identifies which segment register to use. This selection applies to data addresses only. Code addresses always use segment register #15 – the code segment.

## Non-Segmented Code Area

The address range defined as 64’hFxxxxxxxxxxxxxxx (the top nibble is ‘F’) is a non-segmented code area. This area allows the operating system to work without paying attention to the code segment. Interrupt and exception vectors should vector into the non-segmented code area. The only way to change the code segment is by transferring to the operating system via a sys call instruction.

## Changing the Code Segment

The only way to change the code segment is by transferring to the operating system via a sys call instruction. The operating system, while operating in the non-segmented code area, can alter the code segment without causing a transfer of control. The operating system establishes the code segment for a task while running in the non-segmented code area.

## Segment Usage Conventions

Segment register #15 is the code segment (CS) register. All program counter addresses are formed with the code segment register unless the upper nibble of the address is ‘F’ in which case the code segment is ignored.

Segment register #14 is the stack segment (SS) register by convention.

## Power-up State

On reset the value in the segment registers are undefined. Note that the processor begins executing instructions out of the non-segmented code area as the reset address is 64’hFFFFFFFFFFFFFFF0. One of the first tasks of the boot program would be to initialize the segment registers to known values. The segment register must be setup to perform data accesses properly.

## Affect on I/O Access

I/O addresses are not subject to segmentation. I/O addresses are always non-segmented and unmapped.

## TLB

The processor uses a 64 entry TLB (translation look-aside buffer) in order to support virtual memory. The TLB supports variable page sizes from 8kB to 4GB. The TLB is organized as an eight-way eight-set cache.

The TLB is updated by first placing values into the TLB holding registers using the MTSPR instruction, then issuing a TLB write command using the TLB command instruction.

Address translations will not take place until the TLB is enabled. An enable TLB command must be issued using the TLB command instruction.

TLB Entries:

C2..0

G

D

V

ASID7...0

Virtual Address63...13

Physical Address63...13

Physical Address63...13

G = Global

ASID = address space identifier

C = cachability bits

D = dirty bit

V = valid bit

If the G bit is set in the TLB entry, then the ASID field is ignored during the address comparison.

## TLB Registers

### TLBWired (SPR#40h)

This register limits random updates to the TLB to a subset of the available number of ways. TLB ways below the value specified in the Wired register will not be updated randomly.

### TLBIndex (SPR#41h)

This register contains the entry number of the TLB entry to be read from or written to.

### TLBRandom (SPR #42h)

This register contains a random three bit value used to update a random TLB entry during a TLB write operation.

### TLBPageMask (SPR #43h)

The TLBPageMask register is a holding register that contains a mask that controls which address bits are significant during a TLB lookup. If a mask bit is ‘1’ then that address bit is not significant for comparison purposes. This allows the size of memory management pages to be varied.

|  |  |  |
| --- | --- | --- |
| 63 32 | 31 13 | 12 0 |
| ---- | Page Mask | ---- |

|  |  |  |
| --- | --- | --- |
|  | Mask | Page Size |
|  | 0000000000000000000 | 8kB |
|  | 0000000000000000011 | 16KB |
|  | 000000000000001111 | 64kB |
|  | 000000000000111111 | 256kB |
|  | 000000000011111111 | 1MB |
|  | 0000000001111111111 | 4MB |
|  | 000000011111111111 | 16MB |
|  | 000000111111111111 | 64MB |
|  | 000011111111111111 | 256MB |
|  | 001111111111111111 | 1GB |
|  | 111111111111111111 | 4GB |

### TLBPhysPage0 (SPR#45h)

### TLBPhysPage1 (SPR#46h)

The TLBPhysPage register is a holding register that contains the physical address for an associated virtual address. This register is transferred to or from the TLB by TLB instructions.

|  |  |
| --- | --- |
| 63 13 | 12 0 |
| Physical Page Address | ---- |

### TLBVirtPage (SPR #44h)

The TLBVirtPage register is a holding register that contains the virtual address for an associated physical address. This register is transferred to or from the TLB by TLB instructions.

|  |  |
| --- | --- |
| 63 13 | 12 0 |
| Virtual Page Address | ---- |

### TLBASID (SPR #47h)

The TLBASID register is a holding register that contains the address space identifier (ASID) , valid, dirty, global, and cachability bits associated with a TLB entry. This register is transferred to or from the TLB by TLB instructions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 63 16 | 15 8 | 6 4 | 2 | 1 | 0 |
| ----- | ASID | C | G | D | V |

# Vectors

The processor vectors to $FFFFFFFFFFFFFFF0 on a reset. All other vectoring is done through a vector table. The vector table allows for 256 entries. The vector table base address is established by branch address register BR12. During an external IRQ the processor looks at a vector number bus to determine the vector to use for the IRQ. This vector number may be hard-coded in which case all IRQ’s will be vectored to the same location. The address vectored to is the sum of BR12 and an offset supplied in the instruction multiplied by sixteen. The contents of BR12 are undefined at reset; this register must be loaded before interrupts can be processed.

### Vector table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vector Number | Usage / Description | |  |  |
| 0 | BREAK instruction vector | |  |  |
| 1 | SLEEP vector (branch to self) | |  |  |
| 2 | Task reschedule interrupt | |  |  |
| … |  | |  |  |
| 192 | Spurious interrupt | |  |  |
| 193 | IRQ level 1 | 1000 Hz interrupt |  |  |
| 194 | IRQ level 2 | 100 Hz interrupt |  |  |
| … | Other IRQ levels |  |  |  |
| 207 | IRQ level 15 | keyboard interrupt |  |  |
| … |  | |  |  |
| 248 | DTLBMiss | |  |  |
| 249 | ITLB Miss | |  |  |
| 250 | Unimplemented instruction | |  |  |
| 251 | Bus error – data load / store | |  |  |
| 252 | Bus error – instruction fetch | |  |  |
| 253 | reserved | |  |  |
| 254 | NMI interrupt vector | |  |  |
| 255 | * reserved | |  |  |

## Instruction Formats

Instructions vary in length from one to eight bytes. There are only a couple of single byte instructions consisting of only a predicate.

All instruction sequences begin with a predicate byte that determines the conditions under which the instruction executes. With the exception of special predicate values, the next field in the instruction is always the opcode byte. All opcodes may be preceded by an extended constant value.

### RR - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | Opcode8 | Pn4 | Pc4 |

### RI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | Opcode8 | Pn4 | Pc4 |

### CMP Register-Register Compare

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 12 | 11 8 | 7 0 | |
| Rb | Ra | Opc | Pt | Predicate | |
| Rb8 | Ra8 | 14 | Pt4 | Pn4 | Pc4 |

### CMPI Register-Immediate Compare

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 31 24 | | 23 16 | | 15 12 | | 11 8 | | 7 0 | |
| Immed | | Ra | | Opc | | Pt | | Predicate | |
| Immed8 | | Ra8 | | 24 | | Pt4 | | Pn4 | Pc4 |
| TST - Register Test Compare | | | | | | | |  | |  | CTRL- Control | | |
| 23 16 | 15 12 | | 11 8 | | 7 0 | | |  | |  | 15 8 | 7 0 | |
| Ra | Opc | | Pt | | Predicate | | |  | |  | Opcode | Predicate | |
| Ra8 | 04 | | Pt4 | | Pn4 | | Pc4 |  | |  | Opcode8 | Pn4 | Pc4 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BR - Relative Branch | | | | |  |  | | BRK/NOP | |
| 23 16 | 15 8 | | 7 0 | |  |  | | 7 0 | |
| Disp. | Opc | Disp. | Predicate | |  |  | | Predicate | |
| Disp7..0 | 34 | D11..8 | Pn4 | Pc4 |  | |  | 0/14 | 04 |

### JSR - Jump To Subroutine

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Disp. | Br | Brt | Opcode | Predicate | |
| Offset7..0 | Br4 | Brt4 | Opcode8 | Pn4 | Pc4 |

# Instruction Set

### 2ADDU - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 56h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 2 + Rb

#### Description:

Multiply Ra by two and add Rb and place the sum in the target register. This instruction will never cause an overflow exception.

### 2ADDUI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immed | Rb | Ra | Opcode | Predicate | |
| Immed8 | Rb8 | Ra8 | 6Bh8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 2 + immediate

#### Description:

Multiply Ra by two and add immediate and place the sum in the target register. This instruction will never cause an overflow exception.

### 4ADDU - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 57h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 4 + Rb

#### Description:

Multiply Ra by four and add Rb and place the sum in the target register. This instruction will never cause an exception.

### 4ADDUI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immed | Rb | Ra | Opcode | Predicate | |
| Immed8 | Rb8 | Ra8 | 6Ch8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 4 + immediate

#### Description:

Multiply Ra by four and add immediate and place the sum in the target register. This instruction will never cause an exception.

### 8ADDU - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 64h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 8 + Rb

#### Description:

Multiply Ra by eight and add Rb and place the sum in the target register. This instruction will never cause an exception.

### 8ADDUI - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immed | Rb | Ra | Opcode | Predicate | |
| Immed8 | Rb8 | Ra8 | 6Dh8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 8 + immediate

#### Description:

Multiply Ra by eight and add immediate and place the sum in the target register. This instruction will never cause an exception.

### 16ADDU - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 65h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 16 + Rb

#### Description:

Multiply Ra by sixteen and add Rb and place the sum in the target register. This instruction will never cause an exception.

### 16ADDUI - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immed | Rb | Ra | Opcode | Predicate | |
| Immed8 | Rb8 | Ra8 | 6Eh8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra \* 16 + immediate

#### Description:

Multiply Ra by sixteen and add immediate and place the sum in the target register. This instruction will never cause an exception.

### ADD - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 40h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra + Rb

#### Description:

Add two registers and place the sum in the target register. This instruction may cause an overflow exception.

### ADDI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | 48h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra + immediate

#### Description:

Add a register and immediate value and place the sum in the target register. This instruction may cause an overflow exception.

### ADDU - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 44h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra + Rb

#### Description:

This instruction will never cause any exceptions.

### ADDUI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | 4Ch8 | Pn4 | Pc4 |

### AND - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 50h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra & Rb

#### Description:

Locgially and’s two registers and places the result in a target register.

### ANDI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | 53h8 | Pn4 | Pc4 |

### BR - Relative Branch

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23 16 | 15 8 | | 7 0 | |
| Disp. | Op | Disp | Predicate | |
| Disp7..0 | 3h4 | D11..8 | Pn4 | Pc4 |

#### Operation:

pc <= pc + displacement

#### Description:

A branch is made relative to the address of the branch instruction.

* The displacement field cannot be extended with an immediate constant prefix. Branches are executed immediately in the ifetch stage of the processor before it is known if there is a prefix present.

### BRK –Break

|  |  |
| --- | --- |
| 7 0 | |
| Predicate | |
| 04 | 04 |

#### Operation:

<none>

#### Description:

This instruction contains only a predicate byte. The Break exception is executed.

### BSR - Branch To Subroutine

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Disp. | Br | Brt | Opcode | Predicate | |
| Offset7..0 | 154 | Brt4 | A2h8 | Pn4 | Pc4 |

#### Description:

This is an alternate mnemonic for the JSR instruction. Normally the BSR instruction is used with an immediate predicate constant in order to extend the address range of the jump. A jump is made to the sum of the sign extended offset supplied in the offset field of the instruction and the specified branch register Br.

The subroutine return address is stored in a branch register specified in the Brt field of the instruction.

### CLI – Clear Interrupt Mask

|  |  |  |
| --- | --- | --- |
| 15 8 | 7 0 | |
| Opcode | Predicate | |
| FAh8 | Pn4 | Pc4 |

#### Operation:

im = 0

#### Description:

This instruction is used to enable interrupts.

### CMP Register-Register Compare

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 12 | 11 8 | 7 0 | |
| Rb | Ra | Opc | Pt | Predicate | |
| Rb8 | Ra8 | 14 | Pt4 | Pn4 | Pc4 |

### CMPI Register-Immediate Compare

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 12 | 11 8 | 7 0 | |
| Immed | Ra | Opc | Pt | Predicate | |
| Immed8 | Ra8 | 24 | Pt4 | Pn4 | Pc4 |

#### Operation:

if signed Ra < signed immediate

P.lt = true

else

P.lt = false

if unsigned Ra < unsigned immediate

P.ltu = true

else

P.ltu = false

if Ra = immediate

P.eq = true

else

P.eq = false

#### Description:

The register immediate compare instruction compares a register to an immediate value and sets the flags in the target predict register as a result.

### EOR - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 52h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra ^ Rb

#### Description:

Logically exclusive or register with register and place result in target register.

### EORI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | 55h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra ^ immediate

### IMM64,IMM56,IMM48,IMM40,IMM32,IMM24,IMM16

### Immediate Extensions

The immediate extension predicates are used to extend the immediate constant of the following instruction. The extensions may add from one to seven bytes more to the constant. Most, but not all instructions can accept a predicated immediate.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Immediate | | | | | | | Predicate | |
| Immediate63..8 | | | | | | | 84 | 04 |
| Immediate55..8 | | | | | | 74 | 04 |
| Immediate47..8 | | | | | 64 | 04 |
| Immediate39..8 | | | | 54 | 04 |
| Immediate31..8 | | | 44 | 04 |
| Immediate23..8 | | 34 | 04 |
| Immediate15..8 | 24 | 04 |

### INT –Interrupt

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Disp. | Br | Brt | Opcode | Predicate | |
| Offset7..0 | Ch4 | Eh4 | A6h8 | Pn4 | Pc4 |

#### Description:

This instruction calls a system function located as the sum of the zero extended offset times 16 plus branch register 12. The return address is stored in the IPC register (Branch register #14).

The offset field of this instruction cannot be extended.

Note that this instruction is automatically invoked for hardware interrupt processing. This instruction would not normally be used by software and is not supported by the assembler. The return address stored is the address of the interrupt instruction, not the address of the next instruction. To call system routines use the SYS instruction.

### JMP - Jump To Address

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Disp. | Br | Brt | Opcode | Predicate | |
| Offset7..0 | Br4 | 04 | A2h8 | Pn4 | Pc4 |

#### Operation:

pc = Br[n] + offset

#### Description:

This is an alternate mnemonic for the JSR instruction.

Normally the JMP instruction is used with an immediate predicate constant in order to extend the address range of the jump. A jump is made to the sum of the sign extended offset supplied in the offset field of the instruction and the specified branch register Br.

### JSR - Jump To Subroutine Instruction

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Disp. | Br | Brt | Opcode | Predicate | |
| Offset7..0 | Br4 | Brt4 | A2h8 | Pn4 | Pc4 |

#### Operation:

Br[t] = pc

pc = Br[n] + offset

#### Description:

Normally the JSR instruction is used with an immediate predicate constant in order to extend the address range of the jump. A jump is made to the sum of the sign extended offset supplied in the offset field of the instruction and the specified branch register Br.

The subroutine return address is stored in a branch register specified in the Brt field of the instruction. Typically branch register #1 is used.

### LB – Load Byte

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 80h8 | Pn4 | Pc4 |

#### Operation:

Rt = sign extend (mem[Ra+offset])

#### Description:

An eight bit value is loaded from memory and sign extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra.

### LBX – Load Byte Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Rb8 | Ra8 | B0h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+Rb+offset]

#### Description:

An eight bit value is loaded from memory and placed in the target register. The memory address is the sum of the sign extended offset, register Ra and register Rb.

### LBU – Load Byte Unsigned

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 81h8 | Pn4 | Pc4 |

#### Operation:

Rt = zero extend (mem[Ra+offset])

#### Description:

An eight bit value is loaded from memory and zero extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra.

### LBUX – Load Byte Unsigned Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Rb8 | Ra8 | B1h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+Rb+offset]

#### Description:

An eight bit value is loaded from memory zero extended and placed in the target register. The memory address is the sum of the sign extended offset, register Ra and register Rb.

### LC – Load Character

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 82h8 | Pn4 | Pc4 |

#### Operation:

Rt = sign extend (mem[Ra+offset])

#### Description:

A sixteen bit value is loaded from memory and sign extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. The memory address must be character aligned.

### LCX – Load Character Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Rb8 | Ra8 | B2h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+Rb\*2+offset]

#### Description:

A sixteen bit value is loaded from memory, sign extended and placed in the target register. The memory address is the sum of the sign extended offset, register Ra and register Rb multiplied by 2.

### LCU – Load Character Unsigned

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 83h8 | Pn4 | Pc4 |

#### Operation:

Rt = zero extend (mem[Ra+offset])

#### Description:

A sixteen bit value is loaded from memory and zero extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. The memory address must be character aligned.

### LCUX – Load Character Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Rb8 | Ra8 | B3h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+Rb\*2+offset]

#### Description:

A sixteen bit value is loaded from memory, zero extended and placed in the target register. The memory address is the sum of the sign extended offset, register Ra and register Rb multiplied by 2.

### LDI - Load-Immediate

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Opcode | Predicate | |
| Immediate7..0 | Rt8 | 6Fh8 | Pn4 | Pc4 |

#### Operation:

Rt = immediate

#### Description:

This instruction loads a sign extended immediate constant into a register. The immediate constant may be extended by using an immediate prefix instruction.

### LH – Load Half-Word

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 84h8 | Pn4 | Pc4 |

#### Operation:

Rt = sign extend (mem[Ra+offset])

#### Description:

A thirty-two bit value is loaded from memory and sign extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. The memory address must be half-word aligned.

### LHX – Load Walf-word Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Rb8 | Ra8 | B4h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+Rb\*4+offset]

#### Description:

A thirty-two bit value is loaded from memory, sign extended and placed in the target register. The memory address is the sum of the sign extended offset, register Ra and register Rb multiplied by 4.

### LHU – Load Half-word Unsigned

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 85h8 | Pn4 | Pc4 |

#### Operation:

Rt = zero extend (mem[Ra+offset])

#### Description:

A thirty-two bit value is loaded from memory and zero extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. The memory address must be half-word aligned.

### LHUX – Load Half-word Unsigned Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Rb8 | Ra8 | B5h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+Rb\*4+offset]

#### Description:

A thirty-two bit value is loaded from memory, zero extended and placed in the target register. The memory address is the sum of the sign extended offset, register Ra and register Rb multiplied by 4.

### LOOP – Loop Branch

|  |  |  |  |
| --- | --- | --- | --- |
| 23 16 | 15 8 | 7 0 | |
| Disp. | Opcode | Predicate | |
| Disp7..0 | A4h8 | Pn4 | Pc4 |

#### Operation:

If lc <> 0

pc <= pc + displacement

lc = lc - 1

#### Description:

A branch is made relative to the current value of the program counter if the loop count register is non-zero. The loop count register is decremented by this instruction. The predicate condition must also be met.

### LVB – Load Volatile Byte

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | D0h8 | Pn4 | Pc4 |

#### Operation:

Rt = sign extend (mem[Ra+offset])

#### Description:

An eight bit value is loaded from memory and sign extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. This instruction bypasses the data cache. Use this instruction to load data from volatile memory regions such as I/O devices.

### LVC – Load Volatile Character

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | D1h8 | Pn4 | Pc4 |

#### Operation:

Rt = sign extend (mem[Ra+offset])

#### Description:

A sixteen bit value is loaded from memory and sign extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. This instruction bypasses the data cache. Use this instruction to load data from volatile memory regions such as I/O devices.

### LVH – Load Volatile Half-word

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | D2h8 | Pn4 | Pc4 |

#### Operation:

Rt = sign extend (mem[Ra+offset])

#### Description:

A thirty-two bit value is loaded from memory and sign extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. This instruction bypasses the data cache. Use this instruction to load data from volatile memory regions such as I/O devices.

### LVW – Load Volatile Word

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | D3h8 | Pn4 | Pc4 |

#### Operation:

Rt = sign extend (mem[Ra+offset])

#### Description:

A sixty-four bit value is loaded from memory and sign extended, then placed in the target register. The memory address is the sum of the sign extended offset and register Ra. This instruction bypasses the data cache. Use this instruction to load data from volatile memory regions such as I/O devices.

### LW – Load Word

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 86h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+offset]

#### Description:

A sixty-four bit value is loaded from memory and placed in the target register. The memory address is the sum of the sign extended offset and register Ra. The memory address must be word aligned.

### LWX – Load Word Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Rb8 | Ra8 | B6h8 | Pn4 | Pc4 |

#### Operation:

Rt = mem[Ra+Rb\*8+offset]

#### Description:

A sixty-four bit value is loaded from memory and placed in the target register. The memory address is the sum of the sign extended offset, register Ra and register Rb multiplied by eight. The memory address must be word aligned.

### MFSPR – Special Register-Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Spr | Opcode | Predicate | |
| Rt8 | Spr8 | A8h8 | Pn4 | Pc4 |

#### Operation:

Rt = Spr[n]

#### Special Purpose Registers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reg # | R/W |  |  |  |
| 0 | R | MID | Machine ID |  |
| 1 | R | FEAT | Features |  |
| 2 | R | TICK | Tick count |  |
| 3 | RW | LC | Loop Counter |  |
| 4 | RW | PREGS | Predicate register array |  |
|  |  |  |  |  |
| 16-31 | RW | BREGS | Branch register array (BR0 to BR15) |  |
| 32-47 | RW | SREGS | Segment register array (SEG0-SEG15) |  |

### MTSPR –Register-Special Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 8 | 7 0 | |
| Spr | Ra | Opcode | Predicate | |
| Spr8 | Ra8 | A9h8 | Pn4 | Pc4 |

#### Operation:

Spr[n] = Ra

### MUX – Multiplex

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rc | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rc8 | Rb8 | Ra8 | 72h8 | Pn4 | Pc4 |

#### Operation:

For n = 0 to 63

If Ra[n] is set then

Rt[n] = Rb[n]

else

Rt[n] = Rc[n]

#### Description:

If a bit in Ra is set then the bit of the target register is set to the corresponding bit in Rb, otherwise the bit in the target register is set to the corresponding bit in Rc.

### NEG - Negate Register

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Ra | Opcode | Predicate | |
| Rt8 | Ra8 | 70h8 | Pn4 | Pc4 |

#### Operation:

Rt = - Ra

#### Description:

This instruction negates a register and places the result in a target register.

### NOP – No Operation

|  |  |
| --- | --- |
| 7 0 | |
| Predicate | |
| 14 | 04 |

#### Operation:

<none>

#### Description:

This instruction contains only a predicate byte.

### OR - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 51h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra | Rb

#### Description:

Logically inclusively or two registers and place the result in the target register.

### ORI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | 54h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra | imm

### RTE – Return From Exception Routine

|  |  |  |
| --- | --- | --- |
| Opcode | Predicate | |
| F3h8 | Pn4 | Pc4 |

#### Operation**:**

pc = Br[13]

Flags = FlagsBackup

#### **Description**:

The program counter is loaded with the value contained in branch register #13 which is the EPC register.

### RTI – Return From Interrupt Routine

|  |  |  |
| --- | --- | --- |
| Opcode | Predicate | |
| F4h8 | Pn4 | Pc4 |

#### Operation:

pc = Br[14]

Flags = FlagsBackup

Flags.im = 0

#### **Description**:

The program counter is loaded with the value contained in branch register #14 which is the IPC register.

### RTS – Return From Subroutine

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Br | Im | Opcode | Predicate | |
| Br4 | Im4 | A3h8 | Pn4 | Pc4 |

#### Operation:

pc = Br[N]

#### **Description**:

The program counter is loaded with the value contained in the specified branch register plus a zero extended four bit immediate constant. The constant may not be extended. This allows the return instruction to return a few bytes past the usual return address. This is used to allow static parameters to be passed to the subroutine in inline code.

#### Short Form:

|  |  |
| --- | --- |
| Predicate | |
| 14 | 14 |

#### Operation:

pc = Br[1]

#### **Description**:

The program counter is loaded with the value contained in branch register #1.

### SB – Store Byte

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 90h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+offset] = Rb[7..0]

#### Description:

An eight bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset and register Ra.

### SBX – Store Byte Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rc | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rc8 | Rb8 | Ra8 | C0h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+Rb + offset] = Rb

#### Description:

An eight bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset, register Ra and Rb.

### SC – Store Character

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rt | Ra | Opcode | Predicate | |
| Offset7..0 | Rt8 | Ra8 | 91h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+offset] = Rb[15..0]

#### Description:

A sixteen bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset and register Ra. The memory address must be character aligned.

### SCX – Store Character Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rc | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rc8 | Rb8 | Ra8 | C1h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+Rb \* 2 + offset] = Rb

#### Description:

A sixteen bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset, register Ra and Rb multiplied by two. The memory address must be character aligned.

### SEI – Set Interrupt Mask

|  |  |  |
| --- | --- | --- |
| 15 8 | 7 0 | |
| Opcode | Predicate | |
| FBh8 | Pn4 | Pc4 |

#### Operation:

im = 1

#### Description:

The interrupt mask is set, disabling maskable interrupts.

### SH – Store Half-word

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rb8 | Ra8 | 92h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+offset] = Rb[31..0]

#### Description:

A thirty-two bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset and register Ra. The memory address must be half-word aligned.

### SHX – Store Half-word Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rc | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rc8 | Rb8 | Ra8 | C2h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+Rb \* 4 + offset] = Rb

#### Description:

A thirty-two bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset, register Ra and Rb multiplied by four. The memory address must be half-word aligned.

### STSB – Store String Byte

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 8 | 7 0 | |
| Rb | Ra | Opcode | Predicate | |
| Rb8 | Ra8 | 98h8 | Pn4 | Pc4 |

#### Operation:

temp = 0

while LC <> 0

mem[Ra+temp] = Rb[7:0]

temp =temp + 1

LC = LC – 1

#### Description:

This instruction stores a byte to consecutive memory locations beginning at the address in Ra until the loop counter reaches zero. This instruction is interruptible.

### STSW – Store String Word

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 31 24 | 23 16 | 15 8 | 7 0 | |
| Rb | Ra | Opcode | Predicate | |
| Rb8 | Ra8 | 9Ah8 | Pn4 | Pc4 |

#### Operation:

temp = 0

while LC <> 0

mem[Ra+temp] = Rb

temp =temp + 8

LC = LC – 1

#### Description:

This instruction stores a word to consecutive memory locations beginning at the address in Ra until the loop counter reaches zero. This instruction is interruptible.

### SW – Store Word

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rb8 | Ra8 | 93h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+offset] = Rb

#### Description:

A sixty-four bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset and register Ra. The memory address must be word aligned.

### SWX – Store Word Indexed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 47 40 | 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Offset | Rc | Rb | Ra | Opcode | Predicate | |
| Offset7..0 | Rc8 | Rb8 | Ra8 | C3h8 | Pn4 | Pc4 |

#### Operation:

memory[Ra+Rb \* 8 + offset] = Rb

#### Description:

A sixty-four bit value is stored to memory from the source register Rb. The memory address is the sum of the sign extended offset, register Ra and Rb multiplied by eight. The memory address must be word aligned.

### SUB - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 418 | Pn4 | Pc4 |

#### Operation:

Rt = Ra - Rb

#### Description:

This instruction subtracts one register from another and places the result into a third register. This instruction may cause an overflow exception.

### SUBI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | 49h8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra – Imm

#### Description:

This instruction subtracts an immediate value from a register and places the result into a register. This instruction may cause an overflow exception.

### SUBU - Register-Register

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Rt | Rb | Ra | Opcode | Predicate | |
| Rt8 | Rb8 | Ra8 | 458 | Pn4 | Pc4 |

#### Operation:

Rt = Ra - Rb

#### Description:

This instruction subtracts one register from another and places the result into a third register.

### SUBUI - Register-Immediate

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 32 | 31 24 | 23 16 | 15 8 | 7 0 | |
| Immediate | Rt | Ra | Opcode | Predicate | |
| Immediate7..0 | Rt8 | Ra8 | 4Dh8 | Pn4 | Pc4 |

#### Operation:

Rt = Ra – Imm

#### Description:

This instruction subtracts an immediate value from a register and places the result into a register.

### SYNC – Synchronize Memory

|  |  |  |
| --- | --- | --- |
| 15 8 | 7 0 | |
| Opcode | Predicate | |
| F8h8 | Pn4 | Pc4 |

#### Operation:

im = 1

#### Description:

All memory accesses before the SYNC command are completed before execution continues.

### SYS –Call system routine

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Disp. | Br | Brt | Opcode | Predicate | |
| Offset7..0 | Ch4 | Dh4 | A5h8 | Pn4 | Pc4 |

#### Description:

This instruction calls a system function located as the sum of the offset times 16 plus branch register 12. The return address is stored in the EPC register (Branch register #13).

### TLB – TLB Command

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 31 24 | 23 16 | | 15 8 | 7 0 | |
| Rb/Rt | Tn | Cmd | Opcode | Predicate | |
| Rb/Rt8 | Tn4 | Cmd4 | F0h8 | Pn4 | Pc4 |

#### Description:

The command is executed on the TLB unit. The command results are placed in internal TLB registers which can be read or written using TLB command instruction. If the operation is a read register operation then the register value is placed into Rt. If the operation is a write register operation, then the value for the register comes from Rb. Otherwise the Rb/Rt field in the instruction is ignored.

Tn4 – This field identifies which TLB register is being read or written.

|  |  |  |
| --- | --- | --- |
| Reg no. |  |  |
| 0 | Wired |  |
| 1 | Index |  |
| 2 | Random |  |
| 3 | Page Mask |  |
| 4 | Virtual page |  |
| 5 | Physical page 0 |  |
| 6 | Physical page 1 |  |
| 7 | ASID |  |
| 8 | Data miss address |  |
| 9 | Instruction miss address |  |

#### TLB Commands

|  |  |  |
| --- | --- | --- |
| Cmd | Description |  |
| 0 | No operation |  |
| 1 | Probe TLB entry |  |
| 2 | Read TLB entry |  |
| 3 | Write TLB entry corresponding to random register |  |
| 4 | Write TLB entry corresponding to index register |  |
| 5 | Enable TLB |  |
| 6 | Disable TLB |  |
| 7 | Read register |  |
| 8 | Write register |  |

Probe TLB – The TLB will be tested to see if an address translation is present.

Read TLB – The TLB entry specified in the index register will be copied to TLB holding registers.

Write Random TLB – A random TLB entry will be written into from the TLB holding registers.

Write Indexed TLB – The TLB entry specified by the index register will be written from the TLB holding registers.

Disable TLB – TLB address translation is disabled so that the physical address will match the supplied virtual address.

Enable TLB – TLB address translation is enabled. Virtual address will be translated to physical addresses using the TLB lookup tables.

The TLB will automatically update the miss address registers when a TLB miss occurs only if the registers are zero to begin with. System software must reset the registers to zero after a miss is processed. This mechanism ensures the first miss that occurs is the one that is recorded by the TLB.

### TST - Register Test Compare

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 23 16 | 15 12 | 11 8 | 7 0 | |
| Ra | Opc | Pt | Predicate | |
| Ra8 | 04 | Pt4 | Pn4 | Pc4 |

#### Operation:

if Ra < 0

Pt.lt = 1

else

Pt.lt = 0

if Ra = 0

Pt.eq = 1

else

Pt.eq = 0

Pt.ltu = 0

#### Description:

The register test compare compares a register against the value zero and sets the predicate flags appropriately.