**Dark Star \* Dragon Nine**

# Programming Model

## General Purpose Register Array

DSD9 has an array of 64, 80 bit general purpose integer registers. The registers may hold either integer or floating point data.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Usage |  |  |  |
| r0 | always zero |  | r32 |  |
| r1 | return value / exception code |  | r33 | return value |
| r2 | return value / exception type |  | r34 |  |
| r3 | temporary register |  | r35 | temporary register |
| r4 | temporary register |  | r36 | temporary register |
| r5 | temporary register |  | r37 | temporary register |
| r6 | temporary register |  | r38 | temporary register |
| r7 | temporary register |  | r39 | temporary register |
| r8 | temporary register |  | r40 | temporary register |
| r9 | temporary register |  | r41 | temporary register |
| r10 | temporary register |  | r42 | temporary register |
| r11 | register var |  | r43 |  |
| r12 | register var |  | r44 | thread pointer |
| r13 | register var |  | r45 | global pointer |
| r14 | register var |  | r46 | exception link register |
| r15 | register var |  | r47 | frame pointer |
| r16 | register var |  | r48 | r48 operating level 0 |
| r17 | register var |  | r49 | r48 operating level 1 |
| r18 | register var |  | r50 | r48 operating level 2 |
| r19 |  |  | r51 | r48 operating level 3 |
| r20 |  |  | r52 | r52 operating level 0 |
| r21 |  |  | r53 | r52 operating level 1 |
| r22 |  |  | r54 | r52 operating level 2 |
| r23 |  |  | r55 | r52 operating level 3 |
| r24 |  |  | r56 | r56 operating level 0 |
| r25 |  |  | r57 | r56 operating level 1 |
| r26 |  |  | r58 | r56 operating level 2 |
| r27 |  |  | r59 | r56 operating level 3 |
| r28 |  |  | r60 | stack pointer – level 0 |
| r29 |  |  | r61 | stack pointer – level 1 |
| r30 |  |  | r62 | stack pointer – level 2 |
| r31 |  |  | r63 | stack pointer – level 3 |

## Register Banking

For purposes of high-speed exception processing a portion of the register file banked and some registers are not available depending on the operating level. The last sixteen registers are banked according to the core’s operating level. The operating level is added to the register reference to determine which physical register to use. For the user operating level only r48, r52, r56, and r60 are usable register codes. An attempt to read or write any of the other registers above r48 will result in a privilege violation. Internally when the user references r48, r51 is actually read or written. At the supervisor level when r48 is referenced, r50 is actually the physical register updated or read. The supervisor has access to the user level register by referencing r51. At the machine level all registers are accessible and logical register numbers match physical register numbers.

## Register Usage

Several of the general purpose registers have specific uses.

r0 – always reads as zero. This register may be used where the constant zero is required.

r1 – assigned usage is as a return value register. This register also contains the exception code for a locally handled exception. The core loads r1 with the cause code during exception processing.

r2 – has an assigned usage as a return value register. This register is also set to the exception type (24) during exception processing.

r46 – is assigned to be used as the catch handler address register (or exception linkage register). When an exception is routed to local handlers, the core jumps to the address contained in this register.

r47 – has a dedicated use as the stack frame pointer. When referenced in a memory operation causes the core to check the effective address against stack bounds.

r60 – is dedicated to being used as the stack pointer. When referenced in a memory operation causes the core to check the effective address against stack bounds. This register is also automatically updated by stack operations such as push / pop / call / ret.

# Instruction Cache

The instruction cache line size is 160 bits or eight 20 bit instructions. Instructions may span cache lines. There is no wasted memory space.

# Special Purpose Registers

Most special purpose registers are accessible only in kernel mode. A privilege violation will result if attempting to access a special purpose register in user mode that is not available to that mode.

There are no results forwarding on the update of a special purpose register. If the value of the register is required immediately in the following few instructions, then some provision must be made to allow the special purpose register to update. This can be done by following a move to the spr with a couple of NOP instructions. Alternately a branch to a delay subroutine could be performed.

## Program Counter

The program counter identifies which instruction to execute. Normally the program counter increments by the size of the instruction, but the increment may be overridden using one of the flow control instructions. The program counter addresses 20 bit instruction parcels rather than bytes in order to accommodate the 20 and 40 bit instructions. The minimum instruction size is 20 bits. The byte address is 2.5 times the instruction address.

|  |
| --- |
| 63 0 |
|  |

## Control Register Zero (CSR #000)

This register contains a bit to enable protected mode.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 63 | 62 32 30 9 6 1 | | | | | | | | 0 |
| D | ~ |  |  | bpe | ce |  | ~ |  | Pe |

D: debug mode status. this bit is set during an interrupt routine if the processor was in debug mode when the interrupt occurred.

PE: Protected Mode enable: 1 = enabled, 0 = disabled.

CE: cache enable: 1=enabled, 0 = disabled

bpe: branch predictor enable: 1=enabled, 0=disabled

## Tick Count Register (SPR 04 or TICK)

This register contains a count of the number of clock cycles that have passed since the last time the processor was reset. Tick may be used for high-resolution timing or performance measurement.

### Clock Register (SPR 06)

The clock register controls clock gating to the processor to allow lower power consumption. Gating is controlled with a bit pattern which is fed to a clock enable gate. The pattern is 50 bits long, allowed clock control (or power control) in 2% increments. For example loading the register with h2AAAAAAAAAAAA will cause every other clock to be gated off, reducing the effective operating frequency of the core in half. Loading the register with a zero will stop the clock completely. However, a non-maskable interrupt or reset will reload the clock register with all ones, causing the processor to operate at maximum frequency.

|  |  |
| --- | --- |
| 63 50 | 49 0 |
| ~14 | clock gating pattern49..0 |

## DBPC (SPR07)

This register stores the return address for a debug interrupt processing routine. This register is automatically loaded when a debug interrupt occurs. The program counter is loaded from this register automatically as part of the RTD instruction processing.

|  |
| --- |
| 63 0 |
| Value 63..0 |

## IPC (SPR08)

This register stores the return address for a hardware interrupt (NMI / IRQ) processing routine. This register is automatically loaded when a hardware interrupt occurs. The program counter is loaded from this register automatically as part of the RTI instruction processing.

|  |
| --- |
| 63 0 |
| Value 63..0 |

## EPC (SPR09)

This register stores the return address for a software exception processing routine (OVERFLOW / privilege violation). This register is automatically loaded when a software exception occurs. The program counter is loaded from this register automatically as part of the RTE instruction processing.

|  |
| --- |
| 63 0 |
| Value 63..0 |

## Interrupt Vector Table Base Address (SPR 10 or VBR)

This register contains the physical base address of the interrupt vector table in memory. The Table is 4kB aligned.

|  |  |
| --- | --- |
| 63 12 | 11 0 |
| Address63..12 | 00012 |

Interrupt vector table entries are 64 bits in size.

|  |
| --- |
| 63 0 |
| Address 63..0 |

## MULH (SPR14)

This register contains the high order bits of the multiplier product. It is available to both kernel and user modes.

|  |
| --- |
| 63 0 |
| Value 63..0 |

## EA (SPR40)

This register holds the effective address associated with a memory tag. The tag number is contained in bits 16 to 26. The tag associated with this address will be accessible in the TAGS special purpose register. Note that this register and following tag access should be executed with interrupts disabled to prevent the effective address from changing before the tag is updated or read. Also no memory operation should occur between setting this register and updating or reading the tag. This register also reflects the latest effective address calculated by the processor and will be automatically updated when a memory operation occurs.

|  |  |  |
| --- | --- | --- |
| 63 0 | | |
| ~ | tag number11 | Offset16 |

## TAGS (SPR41)

This register makes the tag value accessible for update or read-back. It is used in association with the EA special purpose register. Writing this register will update the tag identified in the EA register.

|  |  |
| --- | --- |
| 63 0 | |
| ~ | Tag16 |

## LOTGRP (SPR 42)

This register contains a list of memory groups that the process belongs to. The owning group associated with a memory tag is compared to this list during a memory access. If the group is in the list then the memory access is allowed, otherwise a memory fault exception occurs. This comparison takes place only in user mode; in kernel mode the kernel owns all of memory so the memory access is always allowed.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 63 60 | 59 50 | 49 40 | 39 30 | 29 20 | 19 10 | 9 0 |
| ~ | Group5 | Group4 | Group3 | Group2 | Group1 | Group0 |

## Compare and Swap (SPR44 or CAS)

This register is to support the compare and swap (CAS) instruction. If the value in the addressed memory location identified by the CAS instruction is equal to the value in the CAS register, then the source register is written to the memory location, and the source register is loaded with the value 1. Otherwise if the value in the addressed memory location doesn’t match the value in this register, then value at the memory location is loaded into the CAS register, and the source register is set to zero. No write to memory occurs if the match fails.

|  |
| --- |
| 63 0 |
| Value 63..0 |

## MYST (SPR45)

This register is to supports the MYST instruction. During execution of the MYST instruction the function code of the operation to be performed is loaded from this register. The MYST register is available to both user and kernel modes.

|  |  |
| --- | --- |
| 63 7 | 6 0 |
| ~57 | Funct7 |

## Debug Address Register (SPR50 to SPR53 or DBAD0 to DBAD3)

These registers contain addresses of instruction or data breakpoints.

|  |
| --- |
| 63 0 |
| Address 63..0 |

## Debug Control Register (SPR54)

These registers contains bits controlling the circumstances under which a debug interrupt will occur.

|  |  |  |  |
| --- | --- | --- | --- |
| bits |  |  |  |
| 3 to 0 | Enables a specific debug address register to do address matching. If the corresponding bit in this register is set and the address (instruction or data) matches the address in the debug address register then a debug interrupt will be taken. |  |  |
| 17, 16 | This pair of bits determine what should match the debug address register zero in order for a debug interrupt to occur.   |  |  |  | | --- | --- | --- | | 17:16 |  |  | | 00 | match the instruction address |  | | 01 | match a data store address |  | | 10 | reserved |  | | 11 | match a data load or store address |  | |  |  |
| 19, 18 | This pair of bits determine how many of the address bits need to match in order to be considered a match to the debug address register. These bits are ignored when matching instruction addresses, which are always half-word aligned.   |  |  |  | | --- | --- | --- | | 19:18 |  | Size | | 00 | all bits must match | byte | | 01 | all but the least significant bit should match | char | | 10 | all but the two LSB’s should match | half | | 11 | all but the three LSB’s should match | word | |  |  |
| 23 to 20 | Same as 16 to 19 except for debug address register one. |  |  |
| 27 to 24 | Same as 16 to 19 except for debug address register two. |  |  |
| 31 to 28 | Same as 16 to 19 except for debug address register three. |  |  |
| 62 | This bit is a history bit for single stepping mode. The debug interrupt records bit 63 into bit 62 when a debug interrupt occurs. Then turns off SSM by writing a zero to bit 63. On return from debug routine (RTD) this bit is restored into bit 63 re-enabling SSM. |  |  |
| 63 | This bit enables SSM (single stepping mode) |  |  |

## Debug Status Register (SPR55)

This register contains bits indicating which addresses matched. These bits are set when an address match occurs, and must be reset by software.

|  |  |
| --- | --- |
| bit |  |
| 0 | matched address register zero |
| 1 | matched address register one |
| 2 | matched address register two |
| 3 | matched address register three |
| 63 to 4 | not used, reserved |

## Floating Point Status and Control Register –SPR20

The floating point status and control register may be read using the MFSPR instruction.

|  |  |  |  |
| --- | --- | --- | --- |
| Bit |  | Symbol | Description |
| 31:29 | **RM** | rm | rounding mode (unimplemented) |
| 28 | **E5** | inexe | - inexact exception enable |
| 27 | **E4** | dbzxe | - divide by zero exception enable |
| 26 | **E3** | underxe | - underflow exception enable |
| 25 | **E2** | overxe | - overflow exception enable |
| 24 | **E1** | invopxe | - invalid operation exception enable |
| 23 | **NS** | ns | - non standard floating point indicator |
| **Result Status** | | | |
| 22 |  | fractie | - the last instruction (arithmetic or conversion) rounded intermediate result (or caused a disabled overflow exception) |
| 21 | **RA** | rawayz | rounded away from zero (fraction incremented) |
| 20 | **SC** | C | denormalized, negative zero, or quiet NaN |
| 19 | **SL** | neg < | the result is negative (and not zero) |
| 18 | **SG** | pos > | the result is positive (and not zero) |
| 17 | **SE** | zero = | the result is zero (negative or positive) |
| 16 | **SI** | inf ? | the result is infinite or quiet NaN |
| **Exception Occurrence** | | | |
| 15 | **X6** | swt | {reserved} - set this bit using software to trigger an invalid operation |
| 14 | **X5** | inerx | - inexact result exception occurred (sticky) |
| 13 | **X4** | dbzx | - divide by zero exception occurred |
| 12 | **X3** | underx | - underflow exception occurred |
| 11 | **X2** | overx | - overflow exception occurred |
| 10 | **X1** | giopx | - global invalid operation exception – set if any invalid operation exception has occurred |
| 9 | **GX** | gx | - global exception indicator – set if any enabled exception has happened |
| 8 | **SX** | sumx | - summary exception – set if any exception could occur if it was enabled  - can only be cleared by software |
| **Exception Type Resolution** | | | |
| 7 | **X1T** | cvt | - attempt to convert NaN or too large to integer |
| 6 | **X1T** | sqrtx | - square root of non-zero negative |
| 5 | **X1T** | NaNCmp | - comparison of NaN not using unordered comparison instructions |
| 4 | **X1T** | infzero | - multiply infinity by zero |
| 3 | **X1T** | zerozero | - division of zero by zero |
| 2 | **X1T** | infdiv | - division of infinities |
| 1 | **X1T** | subinfx | - subtraction of infinities |
| 0 | **X1T** | snanx | - signaling NaN |

## Floating Point Number Format

The floating point number format used by DSD9 is the extended double precision format:

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 1 | 14 | 64 + 1 hidden bit |
| Sm | Se | EEEEEEEEEE | .MMMMMMM…..MMMMMMMM |

Sm = sign of mantissa

Se = sign of exponent

The exponent and mantissa are both represented as two’s complement numbers, however the sign bit of the exponent is inverted.

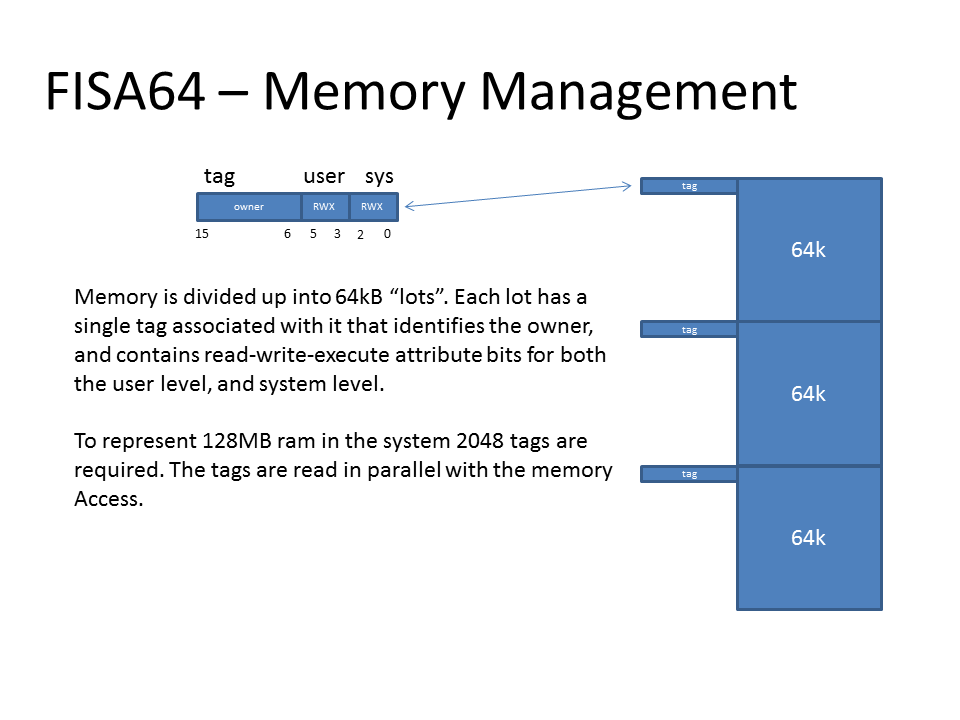
|  |  |
| --- | --- |
| SeEEEEEEEEEE |  |
| 11111111111 | Maximum exponent |
| …. |  |
| 01111111111 | exponent of zero |
| …. |  |
| 00000000000 | Minimum exponent |

The exponent ranges from -16382 to +16382

# Memory Protection System

A key feature required to increase system reliability and robustness is memory protection. Memory should be protected against inadvertent access by the process that doesn’t own a particular piece of memory. The system used here provides memory protection, but not address virtualization.

Memory is organized into lots which are 64kB in size. Memory is protected using a system of tags associated with each lot of memory. The tag associated with a memory lot contains the lot owner’s group, and read / write / execute indicators.



The lot owner field in the memory tag represents a group of processes which may access the memory lot. Each process in the system may be associated with up to six memory groups. Which memory groups the process is a part of is stored in the LOTGRP special purpose register.

### Interrupts

FISA64 uses a vectored interrupt system with support for 512 interrupt vectors.

### Interrupt Vector Table Usage

The following table outlines which vector is used for a given purpose. These vectors are specific to FISA64. Under the HW column an ‘x’ indicates that the interrupt is internally generated by the processor; the vector is hard-wired to that use. An ‘e’ indicates an externally generated interrupt, the usage may vary depending on the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Vecno |  | HW | Description |  |
| 0 |  |  |  |  |
| 1 |  |  |  |  |
| 2 |  |  | FMTK Scheduler |  |
| 3 |  |  | debug interrupt |  |
| 4 |  |  | OS API call |  |
| 449 | KRST | e | Keyboard reset interrupt |  |
| 450 | MSI | e | Millisecond Interrupt |  |
| 451 | TICK | e | FMTK Tick Interrupt |  |
| 463 | KBD | e | Keyboard interrupt |  |
| 487 | BND | x | Bounds check exception |  |
| 488 | DBZ | x | divide by zero |  |
| 489 | OFL | x | overflow |  |
| 493 | FLT | x | floating point exception |  |
| 494 | TAP | x | debug tap interrupt |  |
| 495 | SSM | x | single-step interrupt |  |
| 496 | BPT | x | breakpoint |  |
| 497 | EXF | x | Executable fault |  |
| 498 | DWF | x | Data write fault |  |
| 499 | DRF | x | data read fault |  |
|  |  |  |  |  |
| 501 | PRIV | x | privilege level violation |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 508 | DBE | x | data bus error |  |
| 509 | IBE | x | instruction bus error |  |
| 510 | NMI | x | Non-maskable interrupt |  |
|  |  |  |  |  |

# Instruction Set Description

A description of the instruction set follows.

## Instruction Size

Instructions are 40 bits in size.

|  |
| --- |
| 39 0 |
| Inst |

## Constant Extensions

Constants may be extended up to 80 bits using constant post-words.

Shown below are the instruction formats for the ‘ADD’ immediate instruction with 18, 54, and 80 bits.

18 bit constant

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 39 22 | 21 | 19 14 | 13 8 | 7 0 |  |
| Immediate18 | S2 | Rt6 | Ra6 | 048 | ADD Rt,Ra,#imm |

54 bit constant

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Immediate17..0 | S2 | Rt6 | Ra6 | | 048 | | ADD Rt, #imm |
| Immediate53..22 | | | | Ch4 | | I4 |

80 bit constant

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Immediate18..0 | | S2 | Rt6 | Ra6 | | 048 | | ADD Rt, #imm |
| Immediate53..22 | | | | | Ch4 | | I4 |
| ~10 | Immediate79..58 | | | | Ch4 | | I4 |

If a constant is encountered in the instruction stream it will be treated as a NOP operation. Normally the processor advances past constants as they are encountered.

## Branch Targets

Although instructions are 40 bits in size, each instruction is given a nybble address. This allows relative branch displacements to be used. The instruction address will always be a multiple of five nybbles as the minimum parcel size is 20 bits. A normal subtract may be used to calculate the branch displacement.

Since the parcel size is five nybbles and instruction addresses are always a multiple of five nybbles there is a limited number of possible valid addresses within a cache line (which is 160 bits). Eight 20 bit parcels fit within 160 bits.

## ADD - Addition

ADD.T Rt, Ra, #i19

ADD.H Rt, Ra, Rb

**Instruction Formats**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 04h8 | ~4 | Rt6 | S2 | Rb6 | Ra6 | 028 | ADD Rt,Ra,Rb |
| Immediate18 | | | S2 | Rt6 | Ra6 | 048 | ADD Rt,Ra,#imm |

Operation:

Register Immediate Form

Rt = Ra + immediate20

Register-Register Form

Rt = Ra + Rb

Notes***:***

The immediate constant may be extended up to 64 bits.

|  |  |
| --- | --- |
| S2 | Operation |
| 0 | Wyde parallel |
| 1 | Tetra parallel |
| 2 | Octa parallel |
| 3 | (reserved) Hexi |

## AND – Bitwise logical ‘and’

AND Rt, Ra, #i19

AND Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 32 | 31 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 08h8 | ~4 | Rt6 | S2 | Rb6 | Ra6 | 028 | AND Rt,Ra,Rb |
| Immediate18 | | | S2 | Rt6 | Ra6 | 088 | AND Rt,Ra,#imm |

Operation:

**Register Immediate Form**

Rt = Ra & immediate

**Register-Register Form**

Rt = Ra & Rb

Notes***:***

The immediate constant may be extended up to 64 bits with immediate postfix instructions.

## ASL – Arithmetic Shift Left Register

Description:

Shift register to the left by an amount specified in either a second register or a constant in the instruction and place the result in the target register. A Zero is shifted into bit 0.

**Instruction Format**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 32 | 31 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 32h8 | ~4 | Rt6 | S2 | Rb6 | Ra6 | 028 | ASL Rt,Ra,Rb |
| 42h8 | ~4 | Rt6 | S2 | Immed6 | Ra6 | 028 | ASL Rt,Ra,#i6 |

**Operation:**

Rt = Ra << Rb

Clock Cycles: 1

Exceptions: overflow if the sign bit changes

Notes:

This instruction performs the same operation as the SHL instruction except that it may cause an overflow exception.

## ASR – Arithmetic Shift Right

ASR Rt, Ra, #i6

ASR Rt, Ra, Rb

Description:

Shift register to the right preserving the sign bit, by an amount specified in either a second register or a constant in the instruction and place the result in the target register. The value of the most significant bit is unchanged.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 33h8 | ~5 | Rt6 | S2 | Rb6 | Ra6 | 028 | ASR Rt,Ra,Rb |
| 43h8 | ~5 | Rt6 | S2 | Immed6 | Ra6 | 028 | ASR Rt,Ra,#i6 |

Operation:

**Register Immediate Form**

Rt = Ra >> immediate6

**Register-Register Form**

Rt = Ra >> Rb

Notes:

Performs an arithmetic shift right, preserving the sign bit of the value.

## BFCHG – Bitfield Change

**Description:**

Inverts the bitfield in Ra located between the mask begin (mb) and mask end (me) bits and stores the result in the target register.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 40 36 | 3532 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 25 | ~4 | me6 | mb6 | Rt6 | Ra6 | 13h8 |

## BFCLR – Bitfield Clear

**Description:**

Sets the bits to zero of the bitfield in Ra located between the mask begin (mb) and mask end (me) bits and stores the result in the target register.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 40 36 | 3532 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 15 | ~4 | me6 | mb6 | Rt6 | Ra6 | 13h8 |

## BFEXT – Bitfield Extract

**Description:**

Extracts a bitfield from register Ra located between the mask begin (mb) and mask end (me) bits and places the sign extended result into the target register. This instruction may be used to sign extend a value beginning at any bit.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 40 36 | 3532 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 55 | ~4 | me6 | mb6 | Rt6 | Ra6 | 13h8 |

## BFEXTU – Bitfield Extract Unsigned

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 63 | me6 | mb6 | Rt5 | Ra5 | 03h7 |

**Description:**

Extracts a bitfield from register Ra located between the mask begin (mb) and mask end (me) bits and places the zero extended result into the target register. This instruction may be used to zero extend a value beginning at any bit.

## BFINS – Bitfield Insert

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 33 | me6 | mb6 | Rt5 | Ra5 | 03h7 |
| 43 | me6 | mb6 | Rt5 | Imm5 | 03h7 |

**Description:**

Inserts a bitfield into the target register located between the mask begin (mb) and mask end (me) bits from the low order bits of Ra or an immediate value.

## BFSET – Bitfield Set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 03 | me6 | mb6 | Rt5 | Ra5 | 03h7 |

**Description:**

Sets the bits to one of the bitfield in Ra located between the mask begin (mb) and mask end (me) bits and stores the result in the target register.

## BBC – Branch on Bit Clear

Description:

Branch if a bit in a register is false. The branch range is approximately +/- 64k bytes.

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target17..0 | ~1 | Bitno7 | Ra6 | 54h8 |

Operation:

if (~Ra[bitno])

pc <= pc + sign extended Target;

Clock Cycles: 1 if prediction is true, 3 otherwise

## BBS – Branch on Bit Set

Description:

Branch if a bit in a register is true. The branch range is approximately +/- 64k bytes.

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target17..0 | ~1 | Bitno7 | Ra6 | 55h8 |

Operation:

if (~Ra[bitno])

pc <= pc + sign extended Target;

Clock Cycles: 1 if prediction is true, 3 otherwise

## Bcc – Branch on Compare to Register

Description:

Branch if a comparison condition between a register and another register value is true. The branch range is approximately +/- 64k bytes.

These instructions allow a static branch prediction to be specified. If the branch prediction is statically specified then the branch does not consume room in the branch history table. This may improve the accuracy of predicted branches.

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target17..0 | P2 | Rb6 | Ra6 | Opcode8 |

Operation:

if (Ra cond Rb)

pc <= pc + sign extended Target;

Clock Cycles: 1 if prediction is true, 3 otherwise

Conditions:

|  |  |  |
| --- | --- | --- |
| Opcode8 | Mne. | Description |
| 46 | BEQ | branch if equal |
| 47 | BNE | not equal |
| 48 | BLT | signed less than |
| 49 | BGE | signed greater than or equal |
| 4A | BLE | signed less than or equal |
| 4B | BGT | signed greater than |
| 4C | BLTU | unsigned less than |
| 4D | BGEU | unsigned greater than or equal |
| 4E | BLEU | unsigned less than or equal |
| 4F | BGTU | unsigned greater than |

|  |  |
| --- | --- |
| P2 | Meaning |
| 0 | don’t predict |
| 1 | reserved |
| 2 | predict not taken |
| 3 | predict taken |

## BccI – Branch on Compare to Immediate

Description:

Branch if a comparison condition between a register and an eight bit immediate value is true. The branch range is approximately +/- 64k bytes. The immediate value is sign extended before use. Constant postfix words may be used to extend the immediate to 64 bits.

Instruction Formats:

|  |  |  |  |
| --- | --- | --- | --- |
| Target17..0 | Imm7..0 | Ra6 | Opcode8 |

Operation:

if (Ra cond #imm)

pc <= pc + displacement

Clock Cycles: 1 if prediction is true, 3 otherwise

Conditions:

|  |  |  |
| --- | --- | --- |
| Opcode8 | Mne. | Description |
| 56 | BEQ # | branch if equal |
| 57 | BNE # | not equal |
| 58 | BLT # | signed less than |
| 59 | BGE # | signed greater than or equal |
| 5A | BLE # | signed less than or equal |
| 5B | BGT # | signed greater than |
| 5C | BLTU # | unsigned less than |
| 5D | BGEU # | unsigned greater than or equal |
| 5E | BLEU # | unsigned less than or equal |
| 5F | BGTU # | unsigned greater than |

## BRA – Branch Unconditionally

BRA target\_address

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target17..0 | 32 | 06 | 06 | 46h8 |

Operation:

Notes:

## BRK – Breakpoint Exception

BRK address

Instruction Formats:

|  |  |  |  |
| --- | --- | --- | --- |
| 40 18 | 17 | 16 8 | 7 0 |
| ~ | N | Cause9 | 008 |

Operation:

if (h = 0)

epc = pc

else if (h = 1)

dbpc = pc

else if (h = 2)

ipc = pc

PC = memory[vbr + vector \* 8]

Notes:

Perform an interrupt, exception or debug handler. The handler type is indicated by the ‘H’ field of the instruction. The BRK instruction is used by hardware interrupts to call a hardware interrupt processing routine. The appropriate return instruction (RTE, RTD, or RTI) should be used to return from the BRK handler. The BRK instruction causes the processor to switch to kernel mode.

## CALL – Call Subroutine

CALL (abs,Rn)

CALL d(Rn)

Description:

This instruction performs a subroutine call operation. First the return address is pushed onto the stack (the address of the next instruction). Next the program counter is loaded from the specified operand.

If register 63 is specified for Ra, then the address of the instruction is substituted for use in calculating the target address. This allows program counter relative addressing to be used.

Instruction Formats:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Immed19 | ~2 | 06 | Ra6 | 50h8 | CALL |
| Immed19 | S2 | 06 | Ra6 | 51h8 | CALLI |

Operation:

Memory Indexed Indirect Form

SP = SP – 8

memory[SP] = return address

PC = memory[address + Rn]

Register Indirect with Displacement Form

SP = SP – 8

memory[SP] = return address

PC = displacement + Rn

Notes:

The address constant may be extended up to 64 bits with immediate postfix instructions.

## CAS – Compare and Swap

CAS R1,R2,d[R4]

**Instruction Format:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Disp15 | Rst5 | Ra5 | 6C7h | CAS |

**Operation:**

if memory[Ra+displacement] = casreg

memory[Ra + displacement] = Rst

Rst = 1

else

casreg = memory [Ra + displacement]

Rst = 0

**Description:**

If the contents of the addressed memory cell is equal to the contents of CAS special purpose register then a sixty-four bit value is stored to memory from the source register Rst and Rst is set equal to one. Otherwise Rst is set to zero and the contents of the memory cell is loaded into CAS. The memory address is the sum of the sign extended displacement and register Ra. The compare and swap operation is an atomic operation; the bus is locked during the load and potential store operation. This operation assumes that the addressed memory location is part of the volatile region of memory and bypasses the data cache. Note that the memory system must support bus locks in order for this instruction to work as expected.

This instruction is typically used to implement semaphores. The LWAR and SWCR may also be used to perform a similar function where the memory system does not support bus locks, but support address reservations instead.

**Assembler:**

CAS Rt,Rt,displacement[Ra]

## CHK – Check and Exception

CHK Rt, Ra, Bn

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 0Bh8 | ~5 | Rc6 | S2 | Rb6 | Ra6 | 028 | CHK Ra,Rb,Rc |
| Immediate19 | | | S2 | Rb6 | Ra6 | 0B8 | CHK Ra,Rb,#imm |

Operation:

if not (Ra >= Rb and Ra < Rc or immediate)

take bounds exception

Notes:

This instruction may be used to validate a pointer or array index.

A register is checked against the upper and lower bounds contained in a register identified by the instruction. If the register is not between the upper and lower bounds then a bounds check exception is taken.

The immediate constant may be extended up to 64 bits with immediate prefix instructions.

This exception may be routed to either a local or global exception handler depending on the setting in the EXROUT CSR.

## CMP - Comparison

CMP Rt, Ra, #i16

CMP Rt, Ra, Rb

**Description**

CMP performs a signed comparison of operands and sets the target register to -1, 0, or +1 if the first operand is less than, equal to, or greater than the second respectively.

The immediate constant may be extended up to 64 bits with immediate prefix instructions.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 06h8 | ~5 | Rt6 | S2 | Rb6 | Ra6 | 028 | CMP Rt,Ra,Rb |
| Immediate19 | | | S2 | Rb6 | Ra6 | 068 | CMP Rt,Ra,#imm |

Operation:

**Register Immediate Form**

if (Ra < immediate)

Rt = -1

else if (Ra = immediate)

Rt = 0

else

Rt = 1

**Register-Register Form**

if (Ra < Rb)

Rt = -1

else if (Ra = Rb)

Rt = 0

else

Rt = 1

## CMPU – Unsigned Comparison

CMPU Rt, Ra, #i19

CMPU Rt, Ra, Rb

**Description**

CMPU performs a unsigned comparison of operands and sets the target register to -1, 0, or +1 if the first operand is less than, equal to, or greater than the second respectively.

The immediate constant may be extended up to 64 bits with immediate prefix instructions.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 06h8 | ~5 | Rt6 | S2 | Rb6 | Ra6 | 028 | CMP Rt,Ra,Rb |
| Immediate19 | | | S2 | Rb6 | Ra6 | 068 | CMP Rt,Ra,#imm |

Operation:

**Register Immediate Form**

if (Ra < immediate)

Rt = -1

else if (Ra = immediate)

Rt = 0

else

Rt = 1

**Register-Register Form**

if (Ra < Rb)

Rt = -1

else if (Ra = Rb)

Rt = 0

else

Rt = 1

## COM – Bitwise Ones Complement

COM Rt, Ra

**Description**

All the bits in Ra are inverted and placed into the target register Rt. This is an alternate mnemonic for the XOR instruction.

**Instruction Formats**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 7FFFFh19 | | | ~2 | Rt6 | Ra6 | 0A8 | XOR Rt,Ra,#imm |

Operation:

Register Immediate Form

Rt = Ra ^ -1

Notes***:***

## CPUID – Processor Identification

CPUID Rt, Ra, #n

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 367 |  | ~ | I4 | Rt5 | Ra5 | 027 | CPUID |

Operation:

#### Register-Register Form

Rt = Processor Info Table[Ra|#n]

Notes:

The CPUID instruction returns information about the processor. The contents of register Ra and a four bit immediate value are OR’d together to form an index into the information table. One or the other of register Ra or the immediate value should be zero.

|  |  |  |
| --- | --- | --- |
| Index | bits | Information Returned |
| 0 | 15 to 0 | The processor core number. This field is determined from an external input. It would be hard wired to the number of the core in a multi-core system. |
|  | 23 to 16 | Processor chip number. On a motherboard with multiple chips this identifies the chip the core is located in. It is typically hardwired to zero. |
|  | 31 to 24 | Board number. The number of the processor board in a system with more than one board. |
|  | 39 to 32 | Box number, which box on a rack contains the processor. |
| 2 | 63 to 0 | Manufacturer name first eight chars |
| 3 | 63 to 0 | Manufacturer name |
| 4 | 63 to 0 | CPU class |
| 5 | 63 to 0 | CPU class |
| 6 | 63 to 0 | CPU Name |
| 7 | 63 to 0 | CPU Name |
| 8 | 63 to 0 | Model Number |
| 9 | 63 to 0 | Serial Number |
| 10 | 63 to 0 | Features bitmap |

## CSNZ – Conditional Set if Non-Zero

CSNZ Rt, Ra, #i19

CSNZ Rt, Ra, Rb

**Instruction Formats**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 618 | ~5 | Rt6 | S2 | Rb6 | Ra6 | 028 | CSZ Rt,Ra,Rb |
| Immediate19 | | | S2 | Rt6 | Ra6 | 618 | CSZ Rt,Ra,#imm |

Operation:

Register Immediate Form

If Ra<>0

Rt = immediate

Register-Register Form

If Ra<>0

Rt = Rb

Notes***:***

The immediate constant may be extended up to 64 bits.

|  |  |
| --- | --- |
| S2 | Operation |
| 0 | Wyde parallel |
| 1 | Tetra parallel |
| 2 | Octa parallel |
| 3 | Hexi |

## CSR – Control and Status Register Update

Description:

This instruction atomically reads the CSR into a target register then sets it to either an immediate value supplied by the instruction or a value supplied by a register. Individual bits in the CSR may be set or cleared by the CSRRSI and CSRRCI instructions. Which CSR register to access may be specified by an immediate constant in the instruction, or by the contents of a register.

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4039 | 3836 | 35 22 | 21 14 | 13 8 | 7 0 |  |
| Op2 | 03 | CSR14 | Imm7..0 | Rt6 | 0Fh8 | CSRI8 |

Immediate to CSR #

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Op2 | 13 | ~8 | Rt6 | Imm7..0 | Ra6 | 0Fh6 |  |

Immediate to CSR Ra

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Op2 | 23 | ~8 | Rt6 | ~2 | Rb6 | Ra6 | 0Fh6 |  |

Register to CSR Ra

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Op2 | 33 | CSR14 | ~2 | Rt6 | Ra6 | 0Fh6 |  |

Register to CSR #

|  |  |  |
| --- | --- | --- |
| Op2 | Mne. | Description |
| 0 | CSRRWI | Write the entire value of immediate to the CSR |
| 1 | CSRRSI | Set the bits in the CSR according to the bits set in the immediate |
| 2 | CSRRCI | Clear the bits in the CSR according to the bits set in the immediate |
| 3 |  | not used |

Note that not all CSR’s support the CSRRS and CSRRC instructions.

CSR’s are determined by the lower 12 bits of the CSR field in the instruction. The upper two bits of the CSR field are reserved, and may be used in the future to resolve the core’s operating level.

## CSZ – Conditional Set if Zero

CSZ Rt, Ra, #i19

CSZ Rt, Ra, Rb

**Instruction Formats**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 608 | ~5 | Rt6 | S2 | Rb6 | Ra6 | 028 | CSZ Rt,Ra,Rb |
| Immediate19 | | | S2 | Rt6 | Ra6 | 608 | CSZ Rt,Ra,#imm |

Operation:

Register Immediate Form

If Ra=0

Rt = immediate

Register-Register Form

If Ra=0

Rt = Rb

Notes***:***

The immediate constant may be extended up to 64 bits.

|  |  |
| --- | --- |
| S2 | Operation |
| 0 | Wyde parallel |
| 1 | Tetra parallel |
| 2 | Octa parallel |
| 3 | Hexi |

## DIV – Signed Division

DIV.T Rt, Ra, #i19

DIV.H Rt, Ra, Rb

Description:

**Instruction Formats**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 18h8 | ~5 | Rt6 | S2 | Rb6 | Ra6 | 028 | DIV Rt,Ra,Rb |
| Immediate19 | | | S2 | Rt6 | Ra6 | 188 | DIV Rt,Ra,#imm |

Operation:

Register Immediate Form

Rt = Ra / immediate

Register-Register Form

Rt = Ra / Rb

Notes***:***

The immediate constant may be extended up to 64 bits.

The signed registered form of the instruction may generate a divide by zero exception if the divisor is zero. All other forms of the instruction including signed division by a constant never generate any exceptions.

|  |  |
| --- | --- |
| S2 | Operation |
| 0 | Wyde parallel |
| 1 | Tetra parallel |
| 2 | Octa parallel |
| 3 | Hexi |

## FBcc – Float Branch on Compare to Register

Description:

Branch if the floating point comparison condition between a register and another register value is true. The branch range is approximately +/- 256k bytes.

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Target18..0 | P2 | Rb6 | Ra6 | Opcode8 |

Operation:

if (Ra cond #imm)

pc <= pc + displacement

Clock Cycles: 3 if branch is taken, otherwise 1

Conditions:

|  |  |  |
| --- | --- | --- |
| Opcode8 | Mne. | Description |
| 36 | FBEQ | branch if equal |
| 37 | FBNE | not equal |
| 38 | FBLT | less than |
| 39 | FBGE | greater than or equal |
| 3A | FBLE | less than or equal |
| 3B | FBGT | greater than |
| 3C | FBOR | ordered |
| 3D | FBUN | unordered |

|  |  |
| --- | --- |
| P2 | Precision |
| 0 | single |
| 1 | double |
| 2 | reserved |
| 3 | quad |

## IMM – Immediate Postfix

IMM #i37

Instruction Formats:

|  |  |  |  |
| --- | --- | --- | --- |
| Constant36..4 | Ch4 | C4 | IMM |

Operation:

Notes:

The IMM prefix appends 37 bits onto the 19 bit constant field of the previous instruction then sign extends the resulting 56 bit constant out to 64 bits. Two immediate prefix instructions may be used in succession in order to append up to 64 bits onto the constant field of the following instruction. Thus a full 64 bit constant may be used by most instructions.

The immediate postfix may not be used to extend the range of a branch instruction. If there is an immediate postfix applied to an instruction that doesn’t use a constant, then the postfix will be ignored.

## INC – Increment memory word

INC d(Rn),#n

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Displacement15 | Imm5 | Ra5 | 647h | INC d15(Rn),#n |

Operation:

**Register Indirect with Displacement Form**

memory[displacement + Ra] = memory[displacement + Ra] + n

Notes:

Increments the memory word by a signed five bit immediate constant. The displacement constant may be extended up to 64 bits with immediate prefix instructions.

## JAL – Jump and Link

JAL Rd,(abs,Rn)

JAL Rd,d(Rn)

Instruction Formats:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Immed19 | ~2 | Rt6 | Ra6 | 40h8 | JAL |
| Immed19 | S2 | Rt6 | Ra6 | 41h8 | JALI |

Operation:

Memory Indexed Indirect Form

PC = memory[address + Rn]

Register Indirect with Displacement Form

PC = displacement + Rn

Notes:

The address constant may be extended up to 64 bits with immediate postfix instructions.

For the indirect form the S2 bits identify the size of the address value stored in memory. After loading the value from memory it is zero extended and combined with the upper bits of the program counter to form the final address.

|  |  |
| --- | --- |
| S2 | Meaning |
| 0 | wyde |
| 1 | tetra |
| 2 | octa |
| 3 | hexi |

## JMP – Jump

JMP (abs,Rn)

JMP d(Rn)

Instruction Formats:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Immed19 | ~2 | 06 | Ra6 | 40h8 | JMP |
| Immed19 | S2 | 06 | Ra6 | 41h8 | JMPI |

Operation:

Memory Indexed Indirect Form

PC = memory[address + Rn]

Register Indirect with Displacement Form

PC = displacement + Rn

Notes:

The address constant may be extended up to 64 bits with immediate postfix instructions.

This instruction is an alternate mnemonic for the JAL / JALI instruction, where the target register is specified as zero.

If Ra is specified as 63 then the current instruction address is used in forming the target address. This allows program counter relative addresses to be formed.

## LDB – Load Byte with Sign Extend

## LDBX – Load Byte with Sign Extend

LDB Rt, d(Rn)

LDB Rt, d(Ra + Rb \* scale)

**Description:**

This instruction loads an eight bit quantity into a register then sign extends the value to the width of the machine.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement18 | | | ~2 | Rt6 | Ra6 | 80h8 | LDB Rt,d18(Rn) |
| Displacement8 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A0h8 | LDB Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDBU – Load Byte with Zero Extend

## LDBUX – Load Byte with Zero Extend

LDBU Rt, d(Rn)

LDBU Rt, d(Ra + Rb \* scale)

**Description:**

This instruction loads an eight bit quantity into a register then zero extends the value to the width of the machine.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 81h8 | LDBU Rt,d20(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A1h8 | LDBU Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDH – Load Hexibyte

## LDHX – Load Hexibyte

LDH Rt, d(Rn)

LDH Rt, d(Ra + Rb \* scale)

**Description:**

This instruction is reserved for a 128 bit version of the machine. This instruction loads a 128 quantity into a register.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 88h8 | LDH Rt,d19(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A8h8 | LDH Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

This instruction loads a thirty-two bit value from memory and sign extends it to sixty-four bits.

The displacement constant may be extended up to 64 bits with immediate prefix instructions. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDO – Load Octabyte with Sign Extend

## LDOX – Load Octabyte with Sign Extend

LDO Rt, d(Rn)

LDO Rt, d(Ra + Rb \* scale)

Description:

This instruction loads a 64 bit value into a register and sign extends it to the register width.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 86h8 | LDT Rt,d20(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A6h8 | LDT Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64 bits with immediate prefix instructions. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDOU – Load Octabyte with Zero Extend

## LDOUX – Load Octabyte with Zero Extend

LDOU Rt, d(Rn)

LDOU Rt, d(Ra + Rb \* scale)

Description:

This instruction loads a 64 bit value into a register and zero extends it to the register width. Note that this instruction has the same effect as LDO unless executed on a 128 machine.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 87h8 | LDOU Rt,d20(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A7h8 | LDOU Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64 bits with immediate prefix instructions. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDT – Load Tetrabyte with Sign Extend

## LDTX – Load Tetrabyte with Sign Extend

LDT Rt, d(Rn)

LDT Rt, d(Ra + Rb \* scale)

Description:

This instruction loads a 32 bit value into a register and sign extends it to the register width.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 84h8 | LDT Rt,d20(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A4h8 | LDT Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64 bits with immediate prefix instructions. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDTU – Load Tetrabyte with Zero Extend

## LDTUX – Load Tetrabyte with Zero Extend

LDTU Rt, d(Rn)

LDTU Rt, d(Ra + Rb \* scale)

Description:

This instruction loads a 32 bit value into a register and zero extends it to the register width.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 85h8 | LDTU Rt,d19(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A5h8 | LDTU Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64 bits with immediate prefix instructions. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDW – Load Wyde with Sign Extend

## LDWX – Load Wyde with Sign Extend

LDW Rt, d(Rn)

LDW Rt, d(Ra + Rb \* scale)

**Description:**

This instruction loads a sixteen bit quantity into a register then sign extends it to the width of the register.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 82h8 | LDW Rt,d20(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A2h8 | LDW Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LDWU – Load Wyde with Zero Extend

## LDWUX – Load Wyde with Zero Extend

LDWU Rt, d(Rn)

LDWU Rt, d(Ra + Rb \* scale)

**Description:**

This instruction loads a sixteen bit quantity into a register then zero extends it to the width of the register.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 83h8 | LDWU Rt,d20(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | A3h8 | LDWU Rt,d(Ra+Rb\*sc) |

Operation:

Register Indirect with Displacement Form

Rt = sign extend(memory[displacement + Ra])

Register-Register Form

Rt = sign extend(memory[offset + Ra + Rb \* scale])

Notes:

The displacement constant may be extended up to 64. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LEA – Load Effective Address

LEA Rt,d(Ra)

LEA Rt, d(Ra + Rb \* scale)

**Description**

This instruction loads the effective address of a memory operand into a register.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rt6 | Ra6 | 26h8 | LEA Rt,d19(Rn) |
| Displacement9 | Sc3 | Rt7 | ~2 | Rb6 | Ra6 | 27h8 | LEA Rt,d(Ra+Rb\*sc) |

Operation:

**Indexed Form**

Rt = address of (memory32[offset + Ra + Rb \* scale])

**Notes**:

This instruction loads the target register with the address of the memory determined by the indexing operation.

The displacement constant may be extended up to 64 bits with immediate prefix instructions.

|  |  |
| --- | --- |
| Sc3 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## LWAR – Load Word and Reserve

LWAR Rt, d(Rn)

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Displacement15 | Rt5 | Ra5 | 5C7h | LWAR Rt,d15(Rn) |

Operation:

#### Register Indirect with Displacement Form

Rt = memory[displacement + Ra]

Notes:

This instruction performs the same operation as a load word (LW) instruction except that it sets the sr\_o output signal during the load. The sr\_o output signal can be used to set a memory reservation. LWAR is useful for implementing semaphores.

There is no indexed form of this instruction.

The displacement constant may be extended up to 64 bits with immediate prefix instructions. The offset constant for indexed mode may not be extended.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

## MOV – Move Register to Register

Description:

The contents of the source register Ra are moved to the target register Rt.

Instruction Format:

|  |  |  |
| --- | --- | --- |
| Rt6 | Ra6 | D0h8 |

Clock Cycles: 1

Exceptions: none

## MUL - Multiplication

## MULU - Multiplication

MUL Rt, Ra, #i15

MUL Rt, Ra, Rb

MULU Rt, Ra, #i15

MULU Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 077h | ~3 | Rb5 | Rt5 | Ra5 | 027 | MUL Rt,Ra,Rb |
| Immediate15 | | | Rt5 | Ra5 | 077 | MUL Rt,Ra,#imm |
| 177h | ~3 | Rb5 | Rt5 | Ra5 | 027 | MULU Rt,Ra,Rb |
| Immediate15 | | | Rt5 | Ra5 | 177 | MULU Rt,Ra,#imm |

Operation:

#### Register Immediate Form

Rt = Ra \* immediate15

#### Register-Register Form

Rt = Ra \* Rb

Notes:

The immediate constant may be extended up to 64 bits with immediate prefix instructions.

## POP – Pop Registers from Stack

Instruction Format:

|  |  |  |
| --- | --- | --- |
| 19 14 | 13 8 | 7 0 |
| Rb6 | Ra6 | 9Eh8 |

Registers are popped in the order Ra to Rb.

Example:

POP r1/r2

## PUSH – Push Registers on Stack

Instruction Format:

|  |  |  |
| --- | --- | --- |
| 19 14 | 13 8 | 7 0 |
| Rb6 | Ra6 | 9Dh8 |

Registers are pushed in the order Ra to Rb.

Example:

PUSH r1/r2

## ROL – Rotate Left Register

Description:

Rotates the register to the left by an amount specified in either a second register or a constant in the instruction and place the result in the target register. Bit 63 is shifted into bit 0.

**Instruction Format**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 32 | 31 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 34h8 | ~4 | Rt6 | S2 | Rb6 | Ra6 | 028 | ROL Rt,Ra,Rb |
| 44h8 | ~4 | Rt6 | S2 | Immed6 | Ra6 | 028 | ROL Rt,Ra,#i6 |

**Operation:**

Rt = Ra << Rb

Clock Cycles: 1

Exceptions: none

Notes:

## ROR – Rotate Right

ROR Rt, Ra, #i6

ROR Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 337 |  | | Rb5 | Rt5 | Ra5 | 027 | ROR |
| 3B7 |  | Imm6 | | Rt5 | Ra5 | 027 | ROR # |

Operation:

#### Register Immediate Form

Rt = Ra >> immediate6

#### Register-Register Form

Rt = Ra >> Rb

Notes:

Least significant bits are rotated into the most significant bits.

## RTD – Return from Debug

This instruction returns the processor from debug mode into the mode prior. The program counter is loaded with the value in the DBPC register.

## RTI – Return from Interrupt

This instruction returns the processor from kernel mode into the mode prior. The program counter is loaded with the value in the IPC register.

## RET – Return From Subroutine

RET #i19

**Description**

This instruction performs a return from subroutine operation. The program counter is loaded from the stack, then the stack pointer incremented by eight.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | 636 | 636 | 8Fh8 | RET #i19 |
| 8Fh8 | Sc5 | Rb6 | ~2 | 636 | 636 | 028 | RET Rb |

Operation:

PC = memory[SP]

SP = SP + Immediate

**Notes**:

The stack pointer may be adjusted in order to remove parameters from the stack. In assembler code if an immediate value is specified it must include eight bytes for popping the return address. By default the immediate value is set to eight.

The stack pointer register always has the low order two bits set to zero.

## STB – Store Byte

## STBX – Store Byte

STB Rt, d(Rn)

STB Rt, d(Ra + Rb \* scale)

**Description**

This instruction stores a byte from a register to memory using either register indirect with displacement or scaled indexed addressing.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rs6 | Ra6 | 908h | STB Rs,d15(Rn) |
| Displacement10 | Sc3 | Rb6 | ~2 | Rs6 | Ra6 | B08h | STB Rs,d(Ra+Rb\*sc) |

Operation:

**Register Indirect with Displacement Form**

memory[displacement + Ra] = Rs

**Register-Register Form**

memory[offset + Ra + Rb \* scale] = Rs

**Notes:**

The displacement constant may be extended up to 64 bits with immediate prefix instructions.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## STO – Store Octa

## STOX – Store Octa

STO Rt, d(Rn)

STO Rt, d(Ra + Rb \* scale)

**Description**

This instruction stores an octa (64 bits) from a register to memory using either register indirect with displacement or scaled indexed addressing.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement18 | | | ~2 | Rs6 | Ra6 | 938h | STO Rs,d15(Rn) |
| Displacement9 | Sc3 | Rb6 | ~2 | Rs6 | Ra6 | B38h | STO Rs,d(Ra+Rb\*sc) |

Operation:

**Register Indirect with Displacement Form**

memory[displacement + Ra] = Rs

**Register-Register Form**

memory[offset + Ra + Rb \* scale] = Rs

**Notes:**

The displacement constant may be extended up to 64 bits with immediate prefix instructions.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## STT – Store Tetra

## STTX – Store Tetra

STT Rt, d(Rn)

STT Rt, d(Ra + Rb \* scale)

**Description**

This instruction stores a tetra (32 bits) from a register to memory using either register indirect with displacement or scaled indexed addressing.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rs6 | Ra6 | 928h | STT Rs,d15(Rn) |
| Displacement10 | Sc3 | Rb6 | ~2 | Rs6 | Ra6 | B28h | STT Rs,d(Ra+Rb\*sc) |

Operation:

**Register Indirect with Displacement Form**

memory[displacement + Ra] = Rs

**Register-Register Form**

memory[offset + Ra + Rb \* scale] = Rs

**Notes:**

The displacement constant may be extended up to 64 bits with immediate prefix instructions.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## STW – Store Wyde

## STWX – Store Wyde

STW Rt, d(Rn)

STW Rt, d(Ra + Rb \* scale)

**Description**

This instruction stores a wyde from a register to memory using either register indirect with displacement or scaled indexed addressing.

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Displacement19 | | | ~2 | Rs6 | Ra6 | 918h | STW Rs,d15(Rn) |
| Displacement10 | Sc3 | Rb6 | ~2 | Rs6 | Ra6 | B18h | STW Rs,d(Ra+Rb\*sc) |

Operation:

**Register Indirect with Displacement Form**

memory[displacement + Ra] = Rs

**Register-Register Form**

memory[offset + Ra + Rb \* scale] = Rs

**Notes:**

The displacement constant may be extended up to 64 bits with immediate prefix instructions.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 | 16 |

## SHL – Shift Left Register

Description:

Shift register to the left by an amount specified in either a second register or a constant in the instruction and place the result in the target register. A Zero is shifted into bit 0.

Instruction Format:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 30h8 | ~5 | Rt6 | S2 | Rb6 | Ra6 | 028 | SHL Rt,Ra,Rb |
| 40h8 | ~5 | Rt6 | S2 | Immed6 | Ra6 | 028 | SHL Rt,Ra,#i6 |

Operation:

Rt = Ra << Rb

Clock Cycles: 1

Exceptions: none

## STP – Stop Processor

STP

This instruction stops the processor placing it in low power mode by stopping the processor clock. The clock rate register is loaded with zero. The processor may begin processing again once a non-maskable interrupt occurs or a reset occurs. The processor may be slowed down without stopping the clock by adjusting the value in the clock rate register.

## SWCR – Store Word and Clear Reservation

SWCR Rt, d(Rn)

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Displacement15 | Rs5 | Ra5 | 6E8h | SWCR Rs,d15(Rn) |

Operation:

#### Register Indirect with Displacement Form

if (address reserved)

memory[displacement + Ra] = Rs

cr0[36] = 1

else

cr0[36] = 0

Notes:

Conditionally store a word to memory if an address reservation is present. If successful bit 36 of cr0 will be set, otherwise bit 36 of cr0 will be cleared. This instruction sets the cr\_o signal during execution. The memory system must be capable of aborting the store if there is no reservation present.

The memory access does not need to be aligned, but unaligned accesses will take longer to complete.

The displacement constant may be extended up to 64 bits with immediate prefix instructions.

|  |  |
| --- | --- |
| Sc2 Code | Multiply By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

## SXB – Sign Extend Byte

SXB Rt, Ra

**Description**

This instruction sign extends a byte in a register to the width of the register. This instruction is an alternate mnemonic for the BFEXT instruction.

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 40 36 | 3532 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 55 | ~4 | 76 | 06 | Rt6 | Ra6 | 13h8 |

Operation:

**Register Form**

Rt = sign extend (Ra)

Notes:

The most significant bits (8 to 63) are loaded with the sign extension of bit 7.

## SXW – Sign Extend Wyde

SXW Rt, Ra

**Description**

This instruction sign extends a wyde in a register to the width of the register. This instruction is an alternate mnemonic for the BFEXT instruction.

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 40 36 | 3532 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 55 | ~4 | 156 | 06 | Rt6 | Ra6 | 13h8 |

Operation:

**Register Form**

Rt = sign extend (Ra)

Notes:

The most significant bits (16 to 63) are loaded with the sign extension of bit 15.

## SXT – Sign Extend Tetra

SXT Rt, Ra

**Description**

This instruction sign extends a tetra in a register to the width of the register. This instruction is an alternate mnemonic for the BFEXT instruction.

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 40 36 | 3532 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 55 | ~4 | 316 | 06 | Rt6 | Ra6 | 13h8 |

Operation:

**Register Form**

Rt = sign extend (Ra)

Notes:

The most significant bits (16 to 63) are loaded with the sign extension of bit 15.

## WAIT – Wait For Interrupt

WAIT

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 377 |  | 035 | 005 | ~5 | 027 | WAI |

Operation:

if (no interrupt)

PC = PC

else

PC = PC + 4

Notes:

This instruction waits for an interrupt to occur before proceeding..

## XOR – Bitwise Exclusive ‘Or’

XOR Rt, Ra, #i19

XOR Rt, Ra, Rb

**Instruction Formats**:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 33 | 32 28 | 27 22 | 2120 | 19 14 | 13 8 | 7 0 |  |
| 0Ah8 | ~5 | Rt6 | ~2 | Rb6 | Ra6 | 028 | XOR Rt,Ra,Rb |
| Immediate19 | | | ~2 | Rt6 | Ra6 | 0A8 | XOR Rt,Ra,#imm |

Operation:

Register Immediate Form

Rt = Ra ^ immediate

Register-Register Form

Rt = Ra ^ Rb

Notes***:***

The immediate constant may be extended up to 64 bits.

## Floating Point Instruction Set

### FABS – Absolute Value

**Description:**

This instruction takes the absolute value of a double precision floating point number contained in a general purpose register. The sign bit of the number is cleared. The precision of the number is not affected and the number is not rounded.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 38 | 37 35 | 34 32 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~3 | Prec2 | Rm3 | FRt6 | ~6 | 056 | FRa6 | F1h8 |

**Clock Cycles:** 3

**Execution Units:** All Floating Point

**Operation:**

Rt = Ra

### FADD – Floating point addition

**Description:**

Add two floating point numbers in registers FRa and FRb and place the result into target register FRt.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 38 | 37 35 | 34 32 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~3 | Prec3 | Rm3 | FRt6 | 046 | FRb6 | FRa6 | F2h8 |

**Clock Cycles: 10**

**Execution Units:** All Floating Point

### FCMP - Float Compare

**Description:**

The register compare instruction compares two registers as floating point doubles and sets the flags in the target integer register as a result.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 40 38 | 37 35 | 34 32 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~3 | Prec3 | Rm3 | FRt6 | 016 | FRb6 | FRa6 | F2h8 |

**Clock Cycles:** 1

**Execution Units:** All ALU’s

**Operation:**

if Ra < Rb

Rt[1]= true

else

Rt[1] = false

if mag Ra < mag Rb

Rt[2] = true

else

Rt[2] = false

if Ra = Rb

Rt[0] = true

else

Rt[0] = false

if Ra <= Rb

Rt[3] = true

else

Rt[3] = false

if unordered

Rt[4] = true

else

Rt[4] = false

# Sample Code

## Register – Register Format Instructions

FISA64 includes a standard set of arithmetic and logical instructions including add / subtract / multiply/ divide / modulus / logical and / or / and exclusive or. Also present are shift instructions for both signed and unsigned operations.

The CMP instruction performs a signed comparison of two registers, or a register and immediate value and stores a -1, 0, or +1 in the target register if the first operand is less than, equal to or greater than the second operand respectively. The comparison result may be used by a following branch instruction. The CMPU instruction works the same way as CMP except that it performs an unsigned comparison. CMPU performs an unsigned comparison but produces a signed result.

Executing an RTI instruction enables interrupts. Interrupts may also be enabled and disabled with the CLI and SEI instructions. The RTI instruction also restored the processor mode (user or kernel) that was present before the interrupt. The processor does not support nested interrupts. However an interrupt may be processed during a software exception handler.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Func7 | ~3 | | Rb5 | | | | Rt5 | Ra5 | 027 | {RR} |
| 007 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | NAND |
| 017 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | NOR |
| 027 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | ENOR |
| 047 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | ADD |
| 057 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | SUB |
| 067 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | CMP |
| 077 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | MUL |
| 087 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | DIV |
| 097 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | MOD |
| 0A7 |  | | ~5 | | | | Rt5 | Ra5 | 027 | NOT |
| 0C7 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | AND |
| 0D7 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | OR |
| 0E7 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | EOR |
| 107 |  | | ~ | | | | Rt6 | Ra6 | 027 | SXB |
| 117 |  | | ~ | | | | Rt6 | Ra6 | 027 | SXC |
| 127 |  | | ~ | | | | Rt6 | Ra6 | 027 | SXH |
| 147 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | ADDU |
| 157 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | SUBU |
| 167 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | CMPU |
| 177 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | MULU |
| 187 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | DIVU |
| 197 |  | | Rb5 | | | | Rt5 | Ra5 | 027 | MODU |
| 1A7 |  | | Rb6 | | | | Rc5 | Ra5 | 027 | CHK |
| 1E7 | Spr8 | | | | | | Rc5 | Ra5 | 027 | MTSPR |
| 1F7 | Spr8 | | | | | | Rt5 | Ra5 | 027 | MFSPR |
| 307 |  | | | Rb5 | | | Rt5 | Ra5 | 027 | SLL |
| 317 |  | | | Rb5 | | | Rt5 | Ra5 | 027 | SRL |
| 327 |  | | | Rb5 | | | Rt5 | Ra5 | 027 | ROL |
| 337 |  | | | Rb5 | | | Rt5 | Ra5 | 027 | ROR |
| 347 |  | | | Rb5 | | | Rt5 | Ra5 | 027 | SRA |
| 367 |  | | | ~ | | I4 | Rt5 | Ra5 | 027 | CPUID |
| 377 |  | | | 005 | | | 005 | ~5 | 027 | CLI |
| 377 |  | | | 015 | | | 005 | ~5 | 027 | SEI |
| 377 |  | | | 025 | | | 005 | ~5 | 027 | STP |
| 377 |  | | | 035 | | | 005 | ~5 | 027 | WAI |
| 377 |  | | | 1D5 | | | 1E5 | ~5 | 027 | RTD |
| 377 |  | | | 1E5 | | | 1E5 | ~5 | 027 | RTE |
| 377 |  | | | 1F5 | | | 1E5 | ~5 | 027 | RTI |
| Func7 |  | Imm6 | | | | | Rt5 | Ra5 | 027 | Shifts # |
| 387 |  | Imm6 | | | | | Rt5 | Ra5 | 027 | SLL # |
| 397 |  | Imm6 | | | | | Rt5 | Ra5 | 027 | SRL # |
| 3A7 |  | Imm6 | | | | | Rt5 | Ra5 | 027 | ROL # |
| 3B7 |  | Imm6 | | | | | Rt5 | Ra5 | 027 | ROR # |
| 3C7 |  | Imm6 | | | | | Rt5 | Ra5 | 027 | SRA # |
| 407 | Pred4 | | | | Succ4 | | 05 | ~5 | 027 | FENCE |

Floating Point

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ~5 | Sz2 | rm3 | FPb5 | | FPt5 | FPa5 | 287 | FADD |
| ~5 | Sz2 | rm3 | FPb5 | | FPt5 | FPa5 | 297 | FSUB |
| ~5 | Sz2 | rm3 | FPb5 | | **Rt5** | FPa5 | 2A7 | FCMP |
| ~5 | Sz2 | rm3 | FPb5 | | FPt5 | FPa5 | 2B7 | FMUL |
| ~5 | Sz2 | rm3 | FPb5 | | FPt5 | FPa5 | 2C7 | FDIV |
| 607 | | ~3 | Sz2 | rm3 | FPt5 | FPa5 | 027 | FIX2FLT |
| 617 | | ~3 | Sz2 | rm3 | FPt5 | FPa5 | 027 | FLT2FIX |
| 627 | | ~3 | Sz2 | ~3 | FPt5 | FPa5 | 027 | FMOV |
| 637 | | ~3 | Sz2 | ~3 | FPt5 | FPa5 | 027 | FNEG |
| 647 | | ~3 | Sz2 | ~3 | FPt5 | FPa5 | 027 | FABS |
| 657 | | ~3 | ~2 | ~3 | **Rt5** | FPa5 | 027 | MFFP |
| 667 | | ~3 | Sz2 | ~3 | **Rt5** | FPa5 | 027 | MV2FIX |
| 1C7 | | ~3 | ~2 | ~3 | FPt5 | **Ra5** | 027 | MTFP |
| 1D7 | | ~3 | Sz2 | rm3 | FPt5 | **Ra5** | 027 | MV2FLT |

## Register – Immediate Format Instructions

There are signed and unsigned versions of instructions. The mnemonics of the unsigned instructions are post-fixed with a ‘U’.

ADD / SUB may generate an overflow exception when overflow occurs. ADDU / SUBU do not generate any exceptions.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed15 | Rt5 | Ra5 | 047 | ADD # |
| Immed15 | Rt5 | Ra5 | 057 | SUB # |
| Immed15 | Rt5 | Ra5 | 067 | CMP # |
| Immed15 | Rt5 | Ra5 | 077 | MUL # |
| Immed15 | Rt5 | Ra5 | 087 | DIV # |
| Immed15 | Rt5 | Ra5 | 097 | MOD # |
| Immed15 | Rt5 | ~5 | 0A7 | LDI # |
| Immed15 | Rc5 | Ra5 | 0B7 | CHK # |
| Immed15 | Rt5 | Ra5 | 0C7 | AND # |
| Immed15 | Rt5 | Ra5 | 0D7 | OR # |
| Immed15 | Rt5 | Ra5 | 0E7 | EOR # |
| Immed15 | Rt5 | Ra5 | 147 | ADDU # |
| Immed15 | Rt5 | Ra5 | 157 | SUBU # |
| Immed15 | Rt5 | Ra5 | 167 | CMPU # |
| Immed15 | Rt5 | Ra5 | 177 | MULU # |
| Immed15 | Rt5 | Ra5 | 187 | DIVU # |
| Immed15 | Rt5 | Ra5 | 197 | MODU # |

### Bitfield Instruction Formats

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Bitfield | | | | | | |
| Op3 | me6 | mb6 | Rt5 | Ra5 | 037 | BtFld |
| 03 | me6 | mb6 | Rt5 | Ra5 | 037 | BFSET |
| 13 | me6 | mb6 | Rt5 | Ra5 | 037 | BFCLR |
| 23 | me6 | mb6 | Rt5 | Ra5 | 037 | BFCHG |
| 33 | me6 | mb6 | Rt5 | Ra5 | 037 | BFINS |
| 43 | me6 | mb6 | Rt5 | Imm5 | 037 | BFINSI |
| 53 | me6 | mb6 | Rt5 | Ra5 | 037 | BFEXT |
| 63 | me6 | mb6 | Rt5 | Ra5 | 037 | BFEXTU |
| 73 | me6 | mb6 | Rt5 | Ra5 | 037 |  |

## Flow Control Instructions

There are six relational branches which branch based on the result of a signed comparison of a register to zero. In order to branch based on an unsigned comparison, the CMPU instruction must be used prior to the branch. Since branches inherently compare a register to zero it is often possible to omit a preceding compare (CMP) operation. Branches branch relative to the program counter using a 16 bit signed displacement. This allows branching within +/- 32kB of the current program address.

Two conditional branch instructions (BEQ, BNE) have 16 bit instruction forms.

The subroutine call instruction (BSR) stores the return address in the default link register – R31. The target address is specified as a 26 bit displacement from the current program counter.

In order to jump to a routine whose target address is computed in a register at run time, the JAL instruction is provided.

The BRA instruction works the same way as the BSR instruction, but doesn’t store the return address.

The RTS instruction is used to return from a subroutine and de-allocate a stack frame at the same time. The RTS instruction has a 16 bit instruction format.

The BRK instruction is used to transfer control to a kernel mode BRK handler. This is the means to communicate with the operating system. Hardware interrupts force an appropriate BRK instruction into the instruction stream.

The NOP instruction doesn’t perform any operation.

### Flow Control Instruction Formats

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Flow Control | | | | | | | | |
| Disp15..1 | | | ~2 | | 03 | Ra5 | 3D7 | BEQ |
| Disp15..1 | | | ~2 | | 13 | Ra5 | 3D7 | BNE |
| Disp15..1 | | | ~2 | | 23 | Ra5 | 3D7 | BGT |
| Disp15..1 | | | ~2 | | 33 | Ra5 | 3D7 | BGE / BPL |
| Disp15..1 | | | ~2 | | 43 | Ra5 | 3D7 | BLT / BMI |
| Disp15..1 | | | ~2 | | 53 | Ra5 | 3D7 | BLE |
|  | | | | Dsp5..1 | | Ra5 | 327 | BEQ |
|  | | | | Dsp5..1 | | Ra5 | 337 | BNE |
| T2 | ~4 | Vector9 | ~5 | | | ~5 | 387 | BRK |
|  | | | | Vector9 | | | 357 | SYS |
|  | | | | 39 | | | 367 | INT #3 |
| Offset25..1 | | | | | | | 397 | BSR |
| Offset25..1 | | | | | | | 3A7 | BRA |
| Disp9..1 | | | 237 | BRA |
| Immed15 | | | 1F5 | | | ~5 | 277 | RTL |
|  | | | | Imm9 | | | 377 | RTL |
| Immed15 | | | 1F5 | | | ~5 | 3B7 | RTS |
|  | | | | Imm9 | | | 307 | RTS |
| Immed15 | | | Rt5 | | | Ra5 | 3C7 | JAL |
| Immed15 | | | Rt5 | | | Ra5 | 3E7 | JALI |
| ~25 | | | | | | | 3F7 | NOP |
| 09 | | | 367 | NOP |

Block Move Instruction Formats

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | |  |  |  |  |
| 0 | ~8 | D | Sz | Rs 5 | Ra5 | 777 | SBI |
| 2 | ~8 | D | Sz | Rb5 | Ra5 | 777 | MBI |
|  | | | |  |  |  |  |
|  | | | |  |  |  |  |

## Memory Operate Instructions

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Memory | | | | | | | | 64 bit | |
| Disp15 | | | | Rt5 | | Ra5 | 407 | | LB |
| Disp15 | | | | Rt5 | | Ra5 | 417 | | LBU |
| Disp15 | | | | Rt5 | | Ra5 | 427 | | LC |
| Disp15 | | | | Rt5 | | Ra5 | 437 | | LCU |
| Disp15 | | | | Rt5 | | Ra5 | 447 | | LH |
| Disp15 | | | | Rt5 | | Ra5 | 457 | | LHU |
| Disp15 | | | | Rt5 | | Ra5 | 467 | | LW |
| Disp15 | | | | Rt5 | | Ra5 | 477 | | LEA |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 487 | | LBX |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 497 | | LBUX |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 4A7 | | LCX |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 4B7 | | LCUX |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 4C7 | | LHX |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 4D7 | | LHUX |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 4E7 | | LWX |
| Offs8 | Sc2 | | Rb5 | Rt5 | | Ra5 | 4F7 | | LEAX |
| Disp15 | | | | Rt5 | | Ra5 | 5C7 | | LWAR |
| Disp4 | Rt5 | 5F7 | | LW |
| Disp15 | | | | Rs5 | | Ra5 | 607 | | SB |
| Disp15 | | | | Rs5 | | Ra5 | 617 | | SC |
| Disp15 | | | | Rs5 | | Ra5 | 627 | | SH |
| Disp15 | | | | Rs5 | | Ra5 | 637 | | SW |
| Disp15 | | | | Imm5 | | Ra5 | 647 | | INC |
| Disp15 | | | | ~5 | | Ra5 | 657 | | PEA |
| Disp15 | | | | ~5 | | Ra5 | 667 | | PUSH m |
| Offs8 | Sc2 | | Rb5 | Rs5 | | Ra5 | 687 | | SBX |
| Offs8 | Sc2 | | Rb5 | Rs5 | | Ra5 | 697 | | SCX |
| Offs8 | Sc2 | | Rb5 | Rs5 | | Ra5 | 6A7 | | SHX |
| Offs8 | Sc2 | | Rb5 | Rs5 | | Ra5 | 6B7 | | SWX |
| Disp15 | | | | Rst5 | | Ra5 | 6C7 | | CAS |
| Disp15 | | | | Rs5 | | Ra5 | 6E7 | | SWCR |
| Offs8 | Sc2 | Rb5 | | Imm5 | | Ra5 | 6F7 | | INCX |

FISA64 is a load / store / push / pop architecture.

There are two different instruction formats for memory operating instructions. These are register indirect with displacement format and scaled indexed addressing format. Operand sizes of byte (8 bit), character (16 bit), half-word (32 bit) and word (64 bits) are supported. Sign and zero extension on load is available.

Loads and stores do not have to be aligned, however unaligned access will require additional clock cycles to complete.

## Caveats

### Branches

Branch instructions can’t make proper use of an immediate prefix because they don’t detect an immediate prefix at the If stage in order to keep the hardware simpler. (There is no requirement for conditional branching more than 15 bits). However a branch instruction just uses the same immediate value that is calculated for other instructions in the EX stage. This could lead to branches branching to two different locations if an immediate prefix is used for a branch.

For example if a prefix is used with a branch, BEQ \*+$100010 for instance. Then the branch will branch to \*+$10 if it is predicted taken, but to \*+100010 if it’s predicted not taken, then taken later in the EX stage.

If the branch is predicted taken, it’ll branch using the 15 displacement field from the instruction. If the branch is predicted not taken, but is taken later in the EX stage, it’ll branch using the full immediate value, which with prefixes could be up to 64 bits. The solution is that the assembler never outputs branches with prefixes. There is no hardware protection against using an immediate prefix with a branch.

In the IF stage ,rather than look at the previous instructions for an immediate prefix, the processor simply ignores the fact a prefix is present, and sign extends the branch displacement in the instruction without taking into account a prefix.

IF stage:

if (iopcode==`Bcc && predict\_taken) begin

pc <= pc + {{47{insn[31]}},insn[31:17],2'b00}; // Ignores potential immediate prefix

dbranch\_taken <= TRUE;

end

However, the EX stage uses a full immediate including any prefix, also to simplify hardware.

EX stage:

`Bcc: if (takb & !xbranch\_taken)

update\_pc(xpc + {imm,2'b00}); // This uses a “full” immediate value

else if (!takb & xbranch\_taken)

update\_pc(xpc + 64'd4);

### Software Exceptions

For software type exceptions (divide by zero, overflow) the address stored in the EPC register is the address of the next instruction, not the current instruction address. The issue is that if a system call is being performed one wants to return the next instruction. Since system calls and other software exceptions share the same exception logic, for the usual usage the next instruction address is stored off. It is difficult to determine what the previous address might be as there could be a prefix instruction present.

### Other Limitations

The task register can be read in user mode. This allows an application program to identify where in memory task control information is located. Ideally a user mode application should not be able to find out where operating system data is located. The task register is disabled from being updated by a user mode application so that the task isn’t inadvertently incorrectly switched.

## Major Opcodes

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BRK |  | {R2} |  | ADD # | SUB # | CMP # | CMPU # | AND # | OR # | EOR # |  | CSRI |  | {SYS} | CSR |
| 1x | MUL # | MULU # | MULSU # | MULH # | MULUH # | MULSUH # | CHK # | CHKX # | DIV # | DIVU # | DIVSU # | REM # | REMU # | REMSU # |  | CSR # |
| 2x | \_2ADD # | \_4ADD # | \_8ADD # | \_16ADD # | JMP |  | BEQ | BNE | BLT | BGE | BLE | BGT | BLTU | BGEU | BLEU | BGTU |
| 3x | {F1} | FCMP |  | {F3} |  | NOP | BEQ # | BNE # | BLT # | BGE # | BLE # | BGT # | BLTU # | BGEU # | BLEU # | BGTU # |
| 4x | LDB | LDBU | LDW | LDWU | LDT | LDTU | LDO | LDOU | LDH | LDVB | LDVW | LDVT | LDVO | LDVH | LDVHAR | RET |
| 5x | STB | STW | STT | STO | STH | STHCR | PUSH | PEA | STI | INC | POP |  |  |  | CACHE | CALL |
| 6x | LDBX | LDBUX | LDWX | LDWUX | LDTX | LDTUX | LDOX | LDOUX | LDHX | LV | LVWS | LVX |  |  |  | LCL |
| 7x | SDBX | SDWX | STTX | STOX | STHX | SV | SVWS | SVX | BFSET | BFCLR | BFCHG | BFINS | BFINSI | BFEXT | BFEXTU | MFLT |

# Opcode Map

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  |  | {rr} | {bitfld} | ADD# | SUB# | CMP# | MUL# | DIV# | MOD# | LD# | CHK# | AND# | OR# | EOR# |  |
| 1x | IMM10 16 | |  | IMM41 48 | ADDU# | SUBU# | CMPU # | MULU# | DIVU# | MODU# | LDF# |  |  |  |  |  |
| 2x | MOV 16 | | ADDQ 16 | BRA 16 | MYST | SUB SP 16 |  | RTL | FADD | FSUB | FCMP | FMUL | FDIV |  |  |  |
| 3x | RTS 16 | PUSH/POP | BEQ 16 | BNE 16 | LDIQ 16 | SYS 16 | {PCtrl} 16 | RTL 16 | BRK | BSR | BRA | RTS | JAL | Bcc | JALI | NOP |
| 4x | LB | LBU | LC | LCU | LH | LHU | LW | LEA | LBX | LBUX | LCX | LCUX | LHX | LHUX | LWX | LEAX |
| 5x | LFS | LFD | LFQ |  |  |  |  |  | LFSX | LFDX | LFQX |  | LWAR |  |  | LW BP 16 |
| 6x | SB | SC | SH | SW | INC | PEA | PUSH m | SW BP 16 | SBX | SCX | SHX | SWX | CAS | PEAX | SWCR | INCX |
| 7x | SFS | SFD | SFQ |  | SFSX | SFDX | SFQX | {BLK} | IMM | | | | | | | |

02 Group Func

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | NAND | NOR | ENOR |  | ADD | SUB | CMP | MUL | DIV | MOD | NOT |  | AND | OR | EOR |  |
| 1x | SXB | SXC | SXH |  | ADDU | SUBU | CMPU | MULU | DIVU | MODU | CHK | CHKX | MTFP | MV2FLT | MTSPR | MFSPR |
| 2x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 3x | SLL | SRL | ROL | ROR | ASR |  | CPUID | {ctrl} | SLLI | SRLI | ROLI | RORI | ASRI |  |  |  |
| 4x | FENCE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6x | FIX2FLT | FLT2FIX | FMOV | FNEG | FABS | MFFP | MV2FIX |  |  |  |  |  |  |  |  |  |
| 7x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  | | |  |
|  |  | 64 | Op3 | | |  |
| Immediate9 | | 64 | 03 | | | RTS |
| Immediate9 | | 64 | 73 | | | RTL |
| 04 | Ra5 | 64 | 13 | | | PUSH |
| 14 | FPa5 | 64 | 13 | | | PUSHF |
| 24 | Rt5 | 64 | 13 | | | POP |
| 34 | FPt5 | 64 | 13 | | | POPF |
| Disp4 | Ra5 | 64 | 23 | | | BEQ |
| Disp4 | Ra5 | 64 | 33 | | | BNE |
| Imm4 | Rt5 | 64 | 43 | | | LDI |
| Vector9 | | 64 | 53 | | | SYS |
| 09 | | 64 | 63 | | | NOP |
| 19 | | 64 | 63 | | | STP |
| 29 | | 64 | 63 | | | WAI |
| 39 | | 64 | 63 | | | INT #3 |
| 49 | | 64 | 63 | | | CLI |
| 59 | | 64 | 63 | | | SEI |
| 69 | | 64 | 63 | | | RTI |
| 79 | | 64 | 63 | | | RTE |
| 89 | | 64 | 63 | | | RTD |
| 99 | | 64 | 63 | | |  |
| Rt4 | Ra5 | 44 | 02 | | R | MOV |
| Imm4 | Rt5 | 44 | 13 | | | ADDQ |
| Immediate9 | | 24 | 02 | I | | IMM10 |

## Major Opcodes IR[7:0]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  | {R1} | {R2} | {R3} | ADD # | SUB # | CMP # |  | AND # | OR # | XOR # | CHK # |  | {SYS} | \_16ADD # | CSR |
| 1x | MUL # | MULU # | MULSU # | {BITFIELD} | ADDU # | SUBU # | CMPU # |  | DIV # | DIVU # | DIVSU # | REM # | REMU # | REMSU # | \_8ADD # |  |
| 2x | MULH # | MULUH # | MULSUH # |  | SADD # | SSUB # | LEA | LEAX | DIVR # |  |  |  |  |  | \_4ADD # |  |
| 3x |  |  |  |  |  |  | FBEQ | FBNE | FBLT | FBGE | FBLE | FBGT | FBOR | FBUN | \_2ADD # |  |
| 4x | JMP | JMP () |  |  |  |  | BEQ | BNE | BLT | BGE | BLE | BGT | BLTU | BGEU | BLEU | BGTU |
| 5x | CALL | CALL () |  |  | BBC # | BBS # | BEQ # | BNE # | BLT # | BGE # | BLE # | BGT # | BLTU # | BGEU # | BLEU # | BGTU # |
| 6x | CSZ # | CSNZ # | CSN # | CSNN # | CSP # | CSNP # | CSOD # | CSEV # | ZSZ # | ZSNZ # | ZSN # | ZSNN # | ZNP # | ZSNP # | ZSOD # | ZSEV # |
| 7x |  |  |  |  |  |  | SEQ # | SNE # | SLT # | SGE # | SLE # | SGT # | SLTU # | SGEU # | SLEU # | SGTU # |
| 8x | LDB | LDBU | LDW | LDWU | LDT | LDTU | LDD | LDOU | LDH | LDVB | LDVW | LDVT | LDVD | LDVH | LDVDAR |  |
| 9x | STB | STW | STT | STD | STH | STDCR | INC |  |  |  |  |  | PEA |  |  |  |
| Ax | LDBX | LDBUX | LDWX | LDWUX | LDTX | LDTUX | LDDX | LDOUX | LDHX | LV | LVWS | LVX | LCL | CAS | LDVDARX |  |
| Bx | STBX | STWX | STTX | STDX | STHX | STDCRX | INCX |  |  | SV | SVWS | SVX |  |  | CACHE |  |
| Cx | IMMEDIATE EXTENSION | | | | | | | | | | | | | | | |
| Dx |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ex | MOV | BRK |  |  |  |  |  |  |  |  | NOP | FPUSH | FPOP | PUSH | POP | RET |
| Fx |  | {F1} | {F2} |  |  |  |  |  |  |  |  |  |  |  |  | BRK |

## Major Functs IR[40:33] IR[7:0]=02

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  |  |  |  | ADD | SUB | CMP |  | AND | OR | XOR | CHK |  | {SYS} | \_16ADD |  |
| 1x | MUL | MULU | MULSU |  | ADDU | SUBU | CMPU |  | DIV | DIVU | DIVSU | REM | REMU | REMSU | \_8ADD |  |
| 2x | MULH | MULUH | MULSUH | LEAX | SADD | SSUB |  |  | DIVR |  |  |  |  |  | \_4ADD |  |
| 3x | SHL | SHRU | ASL | ASR | ROL | ROR |  |  |  |  |  |  |  |  | \_2ADD | ABS |
| 4x | SHL # | SHRU # | ASL # | ASR # | ROL # | ROR # |  |  | NAND | NOR | XNOR | ANDN | ORN |  | MIN | MAX |
| 5x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6x | CSZ | CSNZ | CSN | CSNN | CSP | CSNP | CSOD | CSEV | ZSZ | ZSNZ | ZSN | ZSNN | ZNP | ZSNP | ZSOD | ZSEV |
| 7x |  |  |  |  |  |  | SEQ | SNE | SLT | SGE | SLE | SGT | SLTU | SGEU | SLEU | SGTU |
| 8x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | RET |
| 9x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ax |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Bx |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Cx |  | | | | | | | | | | | | | | | |
| Dx |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Ex |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Fx |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |