FISA64

# Programming Model

## General Purpose Register Array

There is an array of 32, 64 bit general purpose integer registers.

|  |  |
| --- | --- |
|  | Usage |
| r0 | always zero |
| r1 | return value |
| r2 |  |
| r3 | temporary register |
| r4 | temporary register |
| r5 | temporary register |
| r6 | temporary register |
| r7 | temporary register |
| r8 | temporary register |
| r9 | temporary register |
| r10 | temporary register |
| r11 | register var |
| r12 | register var |
| r13 | register var |
| r14 | register var |
| r15 | register var |
| r16 | register var |
| r17 | register var |
| r18 | register var |
| r19 |  |
| r20 |  |
| r21 |  |
| r22 |  |
| r23 |  |
| r24 | task register (TR)1 |
| r25 | thread pointer |
| r26 | global pointer |
| r27 | frame pointer (BP) |
| r28 | catch link address (XLR) |
| r29 |  |
| r30 | stack pointer (SP) |
| r31 | return address (LR) |

1 Not updateable in user mode.

# 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.

## Control Register Zero (SPR 00 or CR0)

This register contains a bit to enable protected mode.

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

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)

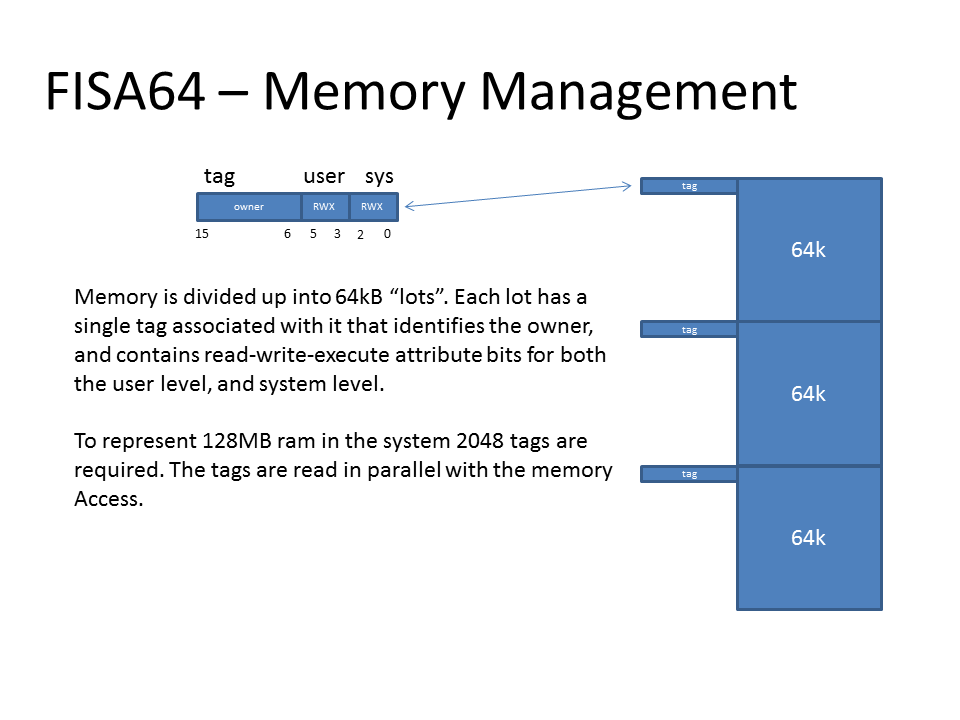
This register contains bits indicating which addresses matched. These bits are set when an address match occurs, and must be reset by software.

|  |  |
| --- | --- |
| 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 |

# 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 |  |
| 488 | DBZ | x | divide by zero |  |
| 489 | OFL | x | overflow |  |
| 493 | FLT | x | floating point exception |  |
| 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.

## ADD - addition

ADD Rt, Ra, #i15

ADD Rt, Ra, Rb

ADDU Rt, Ra, #i15

ADDU Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 047h | ~3 | Rb5 | Rt5 | Ra5 | 027 | ADD Rt,Ra,Rb |
| Immediate15 | | | Rt5 | Ra5 | 047 | ADD Rt,Ra,#imm |
| 147h | ~3 | Rb5 | Rt5 | Ra5 | 027 | ADDU Rt,Ra,Rb |
| Immediate15 | | | Rt5 | Ra5 | 147 | ADDU 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.

Currently the ADD and ADDU instruction both operate the same way. The distinction between the ADD and ADDU instructions is that the ADD instruction may cause an overflow exception, while the ADDU instruction never will.

## AND – bitwise logical ‘and’

AND Rt, Ra, #i15

AND Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0C7 | ~3 | Rb5 | Rt5 | Ra5 | 027 | AND Rt,Ra,Rb |
| Immediate15 | | | Rt5 | Ra5 | 0C7 | AND 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.

## ASR – Arithmetic Shift Right

ASR Rt, Ra, #i6

ASR Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 347 | ~3 | | Rb5 | Rt5 | Ra5 | 27 | ASR Rt, Ra, Rb |
| 3C7 | ~2 | Imm6 | | Rt5 | Ra5 | 27 | 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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 23 | me6 | mb6 | Rt5 | Ra5 | 03h7 |

#### 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.

## BFCLR – Bitfield Clear

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 13 | me6 | mb6 | Rt5 | Ra5 | 03h7 |

#### 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.

## BFEXT – Bitfield Extract

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 53 | 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 sign extended result into the target register. This instruction may be used to sign extend a value beginning at any bit.

## 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.

## Bcc – Branches

Bcc Ra,target\_address

Instruction Formats:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Disp15 | ~2 | Op3 | Ra5 | 3D7h | Bcc address |

Operation:

If (condition) PC= PC +{displacement,2’b00}

Notes:

Branches are relative to the current program counter. A branch is taken to the target address if the condition is true. Branches may branch forwards or backwards up to 64kB in range. The unused bits in the instruction should be set to zero.

The branch condition tests a register against the value zero.

|  |  |  |
| --- | --- | --- |
| Op3 | Mne. |  |
| 0 | BEQ | branch if equal to zero |
| 1 | BNE | branch if not equal |
| 2 | BGT | branch if greater than |
| 3 | BGE | branch if greater or equal |
| 4 | BLT | branch if less than |
| 5 | BLE | branch if less or equal |
| 6 |  | reserved |
| 7 |  | reserved |

## BRA – Branch Unconditionally

BRA target\_address

Instruction Formats:

|  |  |  |
| --- | --- | --- |
| Disp25 | 3A7h | Bcc address |

Operation:

PC= PC +{displacement,2’b00}

Notes:

Branches relative to the current program counter. A branch is taken to the target address if the condition is true. Branches may branch forwards or backwards up to 64MB in range.

## BRK – Breakpoint

BRK address

Instruction Formats:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| H2 | ~4 | Vector9 | ~10 | 387 | BRK |

Operation:

if (h = 0)

epc = pc

else if (h = 1)

dbpc = pc

else if (h = 2)

ipc = pc

PC = vbr + vector \* 16

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.

## BSR – Branch to Subroutine

BSR target

Instruction Formats:

|  |  |  |
| --- | --- | --- |
| Disp25 | 397h | BSR address |

Operation:

#### Relative Address Form

PC = PC+sign extend({Displacement,2’b00})

Branch to a subroutine using program counter relative addressing. The displacement field of the instruction is shifted left twice before being used. The subroutine must be within +/-64MB of the current program counter.

## 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.

### Assembler:

CAS Rt,Rt,displacement[Ra]

## CMP - Comparison

CMP Rt, Ra, #i16

CMP Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 067 | ~3 | Rb5 | Rt5 | Ra5 | 027 | CMP Rt,Ra,Rb |
| Immediate15 | | | Rt5 | Ra5 | 067 | 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

Notes:

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.

## CMPU – Unsigned Comparison

CMPU Rt, Ra, #i16

CMPU Rt, Ra, Rb

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 167 | ~3 | Rb5 | Rt5 | Ra5 | 027 | CMP Rt,Ra,Rb |
| Immediate15 | | | Rt5 | Ra5 | 167 | 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

Notes:

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.

## COM – bitwise ones complement

COM Rt, Ra

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| -115 | Rt5 | Ra5 | 0E7 | COM Rt, Ra |

Operation:

#### Register-Register Form

Rt = ~Ra

Notes:

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

## 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 |

## IMM – Immediate Prefix

IMM #i25

IMM #i25

Instruction Formats:

|  |  |  |
| --- | --- | --- |
| Constant25 | 7C7h | IMM |

Operation:

IMM1: constant buffer = sign extend (immediate32)

IMM2: constant buffer[63:32] = immediate32

Notes:

The IMM prefix appends 25 bits onto the 15 bit constant field of the following instruction then sign extends the resulting 40 bit constant out to 64 bits. Two immediate prefix instructions may be used in succession in order to append up to 49 bits onto the constant field of the following instruction. Thus a full 64 bit constant may be used by most instructions.

When debugging in single-step mode the immediate prefix is not treated as an independent instruction, rather it is an extension of the following instruction, so both the prefix and following instruction get executed in a single step.

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

## JMP – Jump

JMP (abs,Rn)

JMP d(Rn)

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed15 | 05 | Ra5 | 3C7 | JMP |
| Immed15 | 05 | Ra5 | 3E7 | 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 prefix instructions.

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

## LB – Load Byte with Sign Extend

LB Rt, d(Rn)

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

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Displacement15 | | | Rt5 | Ra5 | 407h | LB Rt,d15(Rn) |
| Offset8 | Sc2 | Rb5 | Rt5 | Ra5 | 487h | LB 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 |

## LBU – Load Byte with Zero Extend

LBU Rt, d(Rn)

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

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Displacement15 | | | Rt5 | Ra5 | 417h | LBU Rt,d15(Rn) |
| Offset8 | Sc2 | Rb5 | Rt5 | Ra5 | 497h | LBU Rt,d(Ra+Rb\*sc) |

Operation:

#### Register Indirect with Displacement Form

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

#### Register-Register Form

Rt = zero 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 |

## LC – Load Character with Sign Extend

LC Rt, d(Rn)

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

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Displacement15 | | | Rt5 | Ra5 | 417h | LC Rt,d15(Rn) |
| Offset8 | Sc2 | Rb5 | Rt5 | Ra5 | 497h | LC 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 sixteen 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 |

## LCU – Load Character with Zero Extend

LCU Rt, d(Rn)

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

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Displacement15 | | | Rt5 | Ra5 | 437h | LCU Rt,d15(Rn) |
| Offset8 | Sc2 | Rb5 | Rt5 | Ra5 | 4B7h | LCU Rt,d(Ra+Rb\*sc) |

Operation:

#### Register Indirect with Displacement Form

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

#### Register-Register Form

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

Notes:

A sixteen bit value is loaded from memory, zero extended and placed in the target register.

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 |

## LEA – Load Effective Address

LEA Rt,d(Ra)

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

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Disp15 | Rt5 | Ra5 | 477 | LEA |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Offs8 | Sc2 | Rb5 | Rt5 | Ra5 | 4F7 | LEAX |

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.

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

## LW – Load Word

LW Rt, d(Rn)

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

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Displacement15 | | | Rt5 | Ra5 | 467h | LW Rt,d15(Rn) |
| Offset8 | Sc2 | Rb5 | Rt5 | Ra5 | 4E7h | LW Rt,d(Ra+Rb\*sc) |

Operation:

#### Register Indirect with Displacement Form

Rt = memory[displacement + Ra]

#### Register-Register Form

Rt = 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 |

## 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 |

## 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.

## RTL – Return From Leaf Subroutine

RTL #i15

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed15 | 1F5 | 1E5 | 377 | RTL |

Operation:

PC = LR

SP = SP + Immediate

Notes:

This instruction is used to return from a leaf subroutine (A leaf subroutine does not call another routine). The link register is loaded into the program counter, then the stack pointer updated. The stack pointer may be adjusted in order to remove parameters from the stack. This instruction differs from the RTS instruction in that it doesn’t pop the link register from the stack. As a result the immediate constant specified to adjust the stack pointer is eight less than would be used for the RTS instruction.

## RTS – Return From Subroutine

RTS #i15

Instruction Formats:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed15 | 1F5 | 1E5 | 3B7 | RTS |

Operation:

LR = memory63..2[SP]

PC = LR

SP = SP + Immediate

Notes:

This instruction is used to return from a subroutine. The link register is popped from the stack and loaded into the program counter, then the stack pointer updated. 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 link register. By default the immediate value is set to eight.

## 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.

## SW – Store Word

SW Rt, d(Rn)

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

Instruction Formats:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Displacement15 | | | Rs5 | Ra5 | 638h | SW Rs,d15(Rn) |
| Offset8 | Sc2 | Rb5 | Rs5 | Ra5 | 6B8h | SW 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:

Store a word to memory. 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 |

## 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

Instruction Formats:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 107h |  | Rt5 | Ra5 | 027h | SXB |

Operation:

#### Register Form

Rt = sign extend (Ra)

Notes:

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

## WAI – Wait For Interrupt

WAI

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..

# 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 |
| 1E7 | Spr8 | | | | | ~ | Ra5 | 027 | MTSPR |
| 1F7 | Spr8 | | | | | Rt5 | ~ | 027 | MFSPR |
| 2x7 |  | | Rb5 | | | Rt5 | Ra5 | 027 | Scc |
| 207 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SEQ |
| 217 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SNE |
| 287 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SGT |
| 297 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SLE |
| 2A7 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SGE |
| 2B7 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SLT |
| 2C7 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SHI |
| 2D7 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SLS |
| 2E7 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SHS |
| 2F7 |  | | Rb5 | | | Rt5 | Ra5 | 027 | SLO |
| 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 |

## 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 | 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 # |
| Immed15 | Rt5 | Ra5 | 2x7 | Scc # |
| Immed15 | Rt5 | Ra5 | 207 | SEQ # |
| Immed15 | Rt5 | Ra5 | 217 | SNE # |
| Immed15 | Rt5 | Ra5 | 287 | SGT # |
| Immed15 | Rt5 | Ra5 | 297 | SLE # |
| Immed15 | Rt5 | Ra5 | 2A7 | SGE # |
| Immed15 | Rt5 | Ra5 | 2B7 | SLT # |
| Immed15 | Rt5 | Ra5 | 2C7 | SHI # |
| Immed15 | Rt5 | Ra5 | 2D7 | SLS # |
| Immed15 | Rt5 | Ra5 | 2E7 | SHS # |
| Immed15 | Rt5 | Ra5 | 2F7 | SLO # |

Bitfield Instructions

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 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 17 bit signed displacement. This allows branching within +/- 64kB of the current program address.

The subroutine call instruction (BSR) stores the return address in the default link register – R31. The target address is specified as a 27 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 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 | | | | | | | |
| Disp16..2 | | | ~2 | 03 | Ra5 | 3D7 | BEQ |
| Disp16..2 | | | ~2 | 13 | Ra5 | 3D7 | BNE |
| Disp16..2 | | | ~2 | 23 | Ra5 | 3D7 | BGT |
| Disp16..2 | | | ~2 | 33 | Ra5 | 3D7 | BGE / BPL |
| Disp16..2 | | | ~2 | 43 | Ra5 | 3D7 | BLT / BMI |
| Disp16..2 | | | ~2 | 53 | Ra5 | 3D7 | BLE |
| H | ~4 | Vector9 | ~5 | | 1E5 | 387 | BRK |
| Offset26..2 | | | | | | 397 | BSR |
| Offset26..2 | | | | | | 3A7 | BRA |
| Immed15 | | | 1F5 | | 1E5 | 377 | RTL |
| Immed15 | | | 1F5 | | 1E5 | 3B7 | RTS |
| Immed15 | | | Rt5 | | Ra5 | 3C7 | JAL |
| Immed15 | | | Rt5 | | Ra5 | 3E7 | JALI |
| ~25 | | | | | | 3F7 | NOP |

## Memory Operate Instructions

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.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |
| 815 | | | Rt5 | 1E 5 | 577 | | POP |
| Disp15 | | | Rt5 | Ra5 | 5C7 | | LWAR |
| 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 | | | 1E5 | Ra5 | 657 | | PEA |
| ~15 | | | 1E5 | Ra5 | 677 | | PUSH |
| Disp15 | | | 1E5 | 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 |

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.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 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# |  | AND# | OR# | EOR# |  |
| 1x |  |  |  |  | ADDU# | SUBU# | CMPU # | MULU# | DIVU# | MODU# |  |  |  |  |  |  |
| 2x | SEQ# | SNE# |  |  | MYST |  |  |  | SGT# | SLE# | SGE# | SLT# | SHI# | SLS# | SHS# | SLO# |
| 3x |  |  |  |  |  |  |  | RTL | 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 |  |  |  |  | POP | LFSX | LFDX | LFQX |  | LWAR |  |  |  |
| 6x | SB | SC | SH | SW | INC | PEA | PUSH m | PUSH r | SBX | SCX | SHX | SWX | CAS | PEAX | SWCR |  |
| 7x | SFS | SFD | SFQ |  |  |  |  |  | SFSX | SFDX | SFQX |  | 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 |  |  |  |  | MTSPR | MFSPR |
| 2x | SEQ | SNE |  |  |  |  |  |  | SGT | SLE | SGE | SLT | SHI | SLS | SHS | SLO |
| 3x | SLL | SRL | ROL | ROR | ASR |  | CPUID | {ctrl} | SLLI | SRLI | ROLI | RORI | ASRI |  |  |  |
| 4x | FENCE |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 5x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 6x | FADD | FSUB | FMUL | FDIV |  |  |  |  |  |  |  |  |  |  |  |  |
| 7x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |