Gambit

# Overview

Gambit is a superscalar processor with a 52-bit native operating mode. Native mode makes use of a 32-entry register file. Native mode instructions vary in length up to 52-bits. The processor manages branching using a compare instruction and compare result registers.

## Motivation

Gambit’s goal is to be a cost reduced processing core. Consuming 20% fewer resources than a 64-bit core would.

# Programming Model

## General Registers

Gambit contains a 32-entry by 52-bits wide general-purpose integer register file. One register designation is reserved for the stack pointer. There are four stack pointers, one for each operating level.

|  |  |  |  |
| --- | --- | --- | --- |
| Reg |  |  | Usage |
| R0 | z | This register is always zero |  |
| R1 | acc | Accumulator | First parameter / return value / loop count |
| R2 | x | ‘x’ index register | Second parameter |
| R3 | y | ‘y’ index register | Third parameter |
| R4 |  |  |  |
| R5 |  |  |  |
| R6 |  |  |  |
| R7 to 29 |  |  |  |
| R30 | fp | frame pointer |  |
| R31 | sp | stack pointer | user stack pointer |
| R31 | ssp |  | supervisor stack pointer |
| R31 | hsp |  | hypervisor stack pointer |
| R31 | msp |  | machine stack pointer |

## Compare Result Registers

Compare result registers are two-bit registers that can hold a value from -1 to 1. Compare results are set as the target to several instructions including compare, bit test, and set operations.

-1 means the first operand was less than the second one. 0 means both operands are equal, +1 means the first operand was greater than the second.

|  |  |
| --- | --- |
| Reg | Usage Convention |
| C0 | Integer Compares / set |
| C1 | Float compares / set |
| C2 to C7 |  |

## Link Registers

There are four link registers provided for subroutine calls. There is an additional link register to support exception processing routines.

|  |  |
| --- | --- |
| Reg | Usage |
| L0 | scrap (jumps) |
| L1 | subroutine calls |
| L2 |  |
| L3 |  |
|  |  |
| XL | exception link |

## Control and Status Registers

### Control Register Zero (CSR #000)

This register contains miscellaneous control bits including a bit to enable protected mode.

|  |  |  |
| --- | --- | --- |
| Bit |  | Description |
| 0 | Pe | Protected Mode Enable: 1 = enabled, 0 = disabled |
| 8 to 13 |  |  |
| 16 |  |  |
| 30 | DCE | data cache enable: 1=enabled, 0 = disabled |
| 32 | BPE | branch predictor enable: 1=enabled, 0=disabled |
| 34 | WBM | write buffer merging enable: 1 = enabled, 0 = disabled |
| 35 | SPLE | speculative load enable (1 = enable, 0 = disable) (0 default) |
| 36 |  |  |
| 51 | D | debug mode status. this bit is set during an interrupt routine if the processor was in debug mode when the interrupt occurred. |

This register supports bit set / clear CSR instructions.

DCE

Disabling the data cache is useful for some codes with large data sets to prevent cache loading of values that are used infrequently. Disabling the data cache may reduce security risks for some kinds of attacks. The instruction cache may not be disabled. Enabling / disabling the data cache is also available via the cache instruction.

BPE

Disabling branch prediction will significantly affect the cores performance but may be useful for debugging. Disabling branch prediction causes all branches to be predicted as not-taken. No entries will be updated in the branch history table if the branch predictor is disabled.

WBM bit

Merging of values stored to memory may be disabled by setting this bit. On reset write buffer merging is disabled because it is likely desirable to setup I/O devices. Many I/O devices require updates to individual bytes by separate store instructions. (Write buffer merging is not currently implemented).

SPLE

Enabling speculative loads give the processor better performance at an increased security risk to meltdown attacks.

### HARTID (0x001)

This register contains a number that is externally supplied on the hartid\_i input bus to represent the hardware thread id or the core number. No core should have the value zero as the hartid.

### TICK (0x002)

This register contains a tick count of the number of clock cycles that have passed since the last reset. Note that this register should not be used for precise timing as the processor’s clock frequency may vary for performance and power reasons. The TIME CSR may be used for wall-clock timing as it has its own timing source.

### CAUSE (0x006)

This register contains a code indicating the cause of an exception or interrupt. The break handler will examine this code in order to determine what to do. Only the low order 13 bits are implemented. The high order bits read as zero and are not updateable.

### BADADDR (CSR 0x007)

This register contains the effective address for a load / store operation that caused a memory management exception or a bus error. Note that the address of the instruction causing the exception is available in the XL register.

### BAD\_INSTR (CSR 0x00B)

This register contains a copy of the exceptioned instruction.

### SEMA (CSR 0x00C) Semaphores

This register is available for system semaphore or flag use. The least significant bit is tied to the reservation address status input (rb\_i). It will be set if a SWC instruction was successful. The least significant bit is also cleared automatically when an interrupt (BRK) or interrupt return (RTI) instruction is executed. Any one of the remaining bits may also be cleared by an RTI instruction. This could be a busy status bit for the interrupt routine. Bits in this CSR may be set or cleared with one of the CSRxx instructions. This register has individual bit set / clear capability.

|  |  |
| --- | --- |
| Semaphore | Usage Convention |
| 0 | LDDR / STDC status bit |
| 1 | system garbage collection protector |
| 2 | system |
| 3 | input / output focus list |
| 4 | keyboard |
| 5 | system busy |
| 6 | memory management |
| 7-127 | currently unassigned |

### DOI\_STACK (0x040)

This register contains the stacks for the data operating level, code operating level and interrupt mask. All three stacks are packed into a single register for convenience and performance if the stacks are required to be saved or restored as part of context. When an exception or interrupt occurs, a) the interrupt stack is shifted to the left by three and the low order bits are set to all ones causing all interrupts to be masked b) the code operating level stack is shifted to the left and the low order bits are set to zero causing a switch to the machine operating level for code 3) the data operating level stack is shifted to the left and the low order bits are set to zero causing the machine operating level to be used for data access.

When an RTI instruction is executed these registers are shifted to the right, restoring the previous settings. a) The last interrupt stack entry is set to seven masking all interrupts on stack underflow. The low order three bits represent the current interrupt mask level. b) The last code operating level stack entry is set to zero causing a switch to machine mode on stack underflow. c) The last data operating level stack entry is set to zero causing the machine operating level to be used for data access.

Only the low order 45 bits of the register are implemented, remaining bits read as zero.

Bits 0 to 2 represent the current interrupt mask setting.

Bits 15 to 17 represent the current code operating level setting.

Bits 30 to 32 represent the current data operating level setting.

### PL\_STACK (0x042,0x043)

This pair of CSR’s contains the privilege level stack. When an exception or interrupt occurs, this register is shifted to the left by 13, when an RTI instruction is executed this register is shifted to the right by 13. On RTI the last stack entry will be set to zero which will select privilege level zero on stack underflow. The low order thirteen bits of the register represent the current privilege level.

### STATUS (0x044)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 51 37 | 36 29 | 28 | 27 26 | 25 24 | 23 14 | 13 4 | 3 | 2 | 1 | 0 |
| ~ | ASID | MPRV | XS | FS | ~10 | ~10 | ~ | ~3 | | |

|  |  |  |
| --- | --- | --- |
| Bit |  | Meaning |
| 0 to 2 | ~3 | reserved |
| 4 to 13 | ~10 | reserved |
| 14 to 23 | ~10 | reserved |
| 24 to 25 | FS | floating point state |
| 26 to 27 | XS | additional core extension state |
| 28 | MPRV | memory data level swap |
| 29 to 36 | ASID | address space identifier |
| 37 to 51 |  | reserved |

### BM\_CTR (0xFC1)

This register contains a 40-bit counter of the number of branch misses since the last reset.

### IRQ\_CTR (0xFC3)

This register is reserved to contain a 40-bit count of the number of interrupt requests.

### BR\_CTR (0xFC4)

This register contains a 40-bit counter of the number of branches committed since the last reset.

### TIME (0xFE0, 0xFE1)

The TIME pair of registers corresponds to the wall clock real time. This register can be used to compute the current time based on a known reference point. The register value will typically be a fixed number of seconds offset from the real wall clock time. CSR 0xFE0 bits are driven by the tm\_clk\_i clock time base input which is independent of the cpu clock. The tm\_clk\_i input is a fixed frequency used for timing that cannot be less than 10MHz. The bits represent the fraction of one second. CSR 0xFE1 bits represent seconds passed. For example, if the tm\_clk\_i frequency is 100MHz the bits should count from 0 to 99,999,999 then cycle back to 0 again. When the bits cycle back to 0 again, the bits of the CSR 0xFE1 register are incremented.

Note that this register has a fixed time basis, unlike the TICK register whose frequency may vary with the cpu clock. The cpu clock input may vary in frequency to allow for performance and power adjustments.

Note the time register is not set until the seconds register is set. The fraction of seconds value won’t transfer to the time register until the seconds are set.

Each of these registers is only 40-bits in size.

### TIME (0xFE1)

This register contains a reference of the number of seconds offset from the real wall clock time. It is a carry over from CSR 0xFE0 which contains the fraction of seconds value. The bits of the register represent the number of seconds passed since an arbitrary point in the past.

### INSTRET (0xFE2)

This register contains a count of the number of instructions retired (successfully completed) by the core.

### INFO (0xFF0 to 0xFFF)

This set of registers contains general information about the core including the manufacturer name, cpu class and name, and model number.

## Memory Addressing

The cpu is word oriented with byte addressable memory. The smallest addressable unit of data is a 13-bit byte. Up to 252 Bytes of data are supported and 252 bytes of code. Code and data share the same address space.

Memory data may be accessed using load and store instructions. The load and store instructions support two basic addressing modes: register indirect with displacement and scaled indexed addressing.

Program code is byte aligned.

## Operating Levels

The core has six distinct operating levels. The highest operating level is operating level zero which is called the machine operating level. Operating level zero has complete access to the machine. Other operating levels may have more restricted access. When an interrupt occurs, the operating level is set to the machine level. When operating at level zero addresses are not subjected to translation and the virtual address and physical address are the same.

|  |  |  |
| --- | --- | --- |
| Operating Level |  | Privilege Level Available |
| 0 | machine | 0 to 8191 |
| 1 | hypervisor level | 0 to 8191 |
| 2 | supervisor level1 | 16 to 8191 |
| 3 | supervisor level2 | 128 to 8191 |
| 4 | supervisor level3 | 1024 to 8191 |
| 5 | user | 7168 to 8191 |

### Switching Operating Levels

The operating level is automatically switched to the machine level when an interrupt occurs. The BRK instruction may be used to switch operating levels. The REX instruction may also be used by an interrupt handler to switch the operating level to a lower level. The RTI instruction will switch the operating level back to what it was prior to the interrupt.

## Privilege Levels

The core supports an 8192-level privilege level system. Available privilege levels correspond to the operating level as shown in the table above.

# Exceptions

All processor exceptions vector using the same vector stored in memory location $FFFFFFFFE0000. The exceptions are differentiated by the value in the exception cause register.

|  |  |  |
| --- | --- | --- |
| Cause Code | Which Exception |  |
| 0 | no exception |  |
| 25h | Unimplemented instruction encountered |  |
| 28h | divide by zero |  |
| 150h to 157h | Interrupt occurred |  |
| 160h | Non-maskable interrupt occurred |  |
| 161h | Sequence number reset |  |
| 170h | Processor was reset |  |
| 140h to 14Fh | the BRK instruction was executed |  |

## Sequence Numbers

Sequence numbers are mentioned here because they are used by branch instructions to determine what to invalidate. There a few different ways to handle invalidating instructions in the queue that follow a branch miss. Perhaps the lowest cost and fastest means is to use branch tags. Two other means are using a set of branch shadow bitmasks and using instruction sequence numbers.

Sequence numbers are probably conceptually the simplest means to understand and are simple to implement. The drawback is that they may require more hardware than other means. The order of instructions in the queue is associated with a number. The core assigns a sequence number to each instruction as it enters the instruction queue. One purpose of the sequence number is to allow the core to determine which instructions should be invalidated because of a branch miss. All the instructions in the queue with a sequence number coming after the branch instruction’s sequence number will be invalidated. In the case of Gambit, the sequence number assigned is simply the tick count times two, plus the queue slot number. One challenge of such a simple approach is that sequence numbers might overflow. The size of the sequence number is set small enough that there’s no impact to core performance, and large enough that overflows are infrequent. The chosen size is 26 bits, allowing up to 64 million clock cycles between each sequence number overflow. If the sequence number were to overflow in an uncontrolled fashion the order of instructions in the queue would be upset. So, the sequence number overflow happens in a controlled fashion. Just before the number would overflow an interrupt is generated. During the interrupt routine the sequence number is reset to zero at a point of execution of instructions where sequence numbering won’t impact execution of instructions. This is done by executing enough NOP instructions to fill the processor queue then issuing a sequence number reset command. It’s important that no branches are executed during the sequence number reset. Note that the sequence number interrupt may be avoided by resetting the sequence number before it overflows. Many systems have a periodic interrupt which occurs much more frequently than a sequence number would need to be reset. Resetting the sequence number could be done during this routine. It does require a number of instructions to execute without branching; the number of such instructions depends on the processor queue size.

The sequence number mechanism allows the core to speculate across any number of branches that might be in the instruction queue. However, sequence numbers require more bits in the instruction queue than branch tags and even more hardware.

For debugging purposes, it can be handy to see the sequence number assigned to the instruction.

# Instruction Set Summary

The instruction set includes basic arithmetic, logic, and shift instructions including add, sub, and, or, eor, asl, rol, lsr, and ror. There are several additional instructions which are alternate mnemonics for other instructions. These include bit, and cmp.

The cycle counts are assuming no wait states are required for either instructions or data and both instructions and data can be found in the cache.

# ALU Operations

## ADD – Addition

**Description**:

Add two operand values and place the result in the target register. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## ADDIS – Add Immediate Shifted

**Description**:

Add an immediate value to bits 22 to 51 of register Ra5 and place the result in target register Rt5. Use this instruction when performing an addition with a full 52-bit constant or when building a value in a register.

**Formats Supported**: RIS

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## AND – Bitwise ‘And’

**Description**:

Bitwise ‘And’ two operand values and place the result in the target register. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## ANDIS – And Immediate Shifted

**Description**:

Bitwise ‘and’ an immediate value to bits 22 to 51 of register Ra5 and place the result in target register Rt5. Use this instruction when performing an addition with a full 52-bit constant or when building a value in a register. The constant used is one extended on the right hand side.

**Formats Supported**: RIS

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## ASL – Arithmetic Shift Left

**Description**:

Left shift one operand value by a second operand value and place the result in the target register. Zeros are shifted into the least significant bits. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## ASR – Arithmetic Shift Right

**Description**:

Right shift one operand value by a second operand value while preserving the sign bit and place the result in the target register. The sign bit is preserved as the shift takes place. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## BIT – Test Bits

**Description**:

Bitwise ‘And’ two operand values and place a result in a compare result register. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value. If the result of the ‘and’ operation is zero, a zero is stored in the compare result register. If the result of the ‘and’ operation is non-zero then either +1 or -1 is stored in the result register depending on the sign of the result.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## CMP – Comparison

**Description**:

Compare two operand values and store the relationship in the target compare result register. The operand values are treated as signed integers. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

If the first operand is less than the second, a minus one is stored in the target register. If the first operand equals the second a zero is stored to the target register, otherwise a positive one is stored.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## CMPU – Unsigned Comparison

**Description**:

Compare two operand values and store the relationship in the target compare result register. The operand values are treated as unsigned integers. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

If the first operand is less than the second, a minus one is stored in the target register. If the first operand equals the second a zero is stored to the target register, otherwise a positive one is stored.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## CSR – Control and Status Register Operation

**Description**:

The CSR instruction may be used to read, write or read and write a control or status register. The CSR to access is identified by a twelve-bit constant in the instruction. The current value of the CSR will be read into the register specified by the Rt field of the instruction. If Rt is zero the read data will be discarded. The CSR will be updated with the contents of the register specified by the Ra field of the instruction. If Ra is specified as register zero, then no update takes place. The CSR read / update operation is an atomic operation.

**Formats Supported**: CSR

|  |  |  |
| --- | --- | --- |
| Op3 |  | Operation |
| 0 | CSRRD | Only read the CSR, no update takes place, Rs1 should be R0. |
| 1 | CSRRW | Both read and write the CSR |
| 2 | CSRRS | Read CSR then set CSR bits |
| 3 | CSRRC | Read CSR then clear CSR bits |
| 4 |  | reserved |
| 5 | CSRRWI | Read and Write CSR with immediate |
| 6 | CSRRSI | Read and set using immediate |
| 7 | CSRRCI | Read and clear using immediate |

CSRRS and CSRRC operations are only valid on registers that support the capability.

The OL2 field is reserved to specify the operating level. Note that registers cannot be accessed by a lower operating level.

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

**Examples**:

CSRRD r1,r0,#CAUSECD ; read the cause code register into r1

## DIV – Division

**Description**:

Divide two operand values and place the result in the target register. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or a small immediate value. Both operands are treated as signed values.

**Formats Supported**: RR, RI8

**Execution Units**: ALU

**Clock Cycles**: 55

**Exceptions**: none

## EOR – Bitwise Exclusive ‘Or’

**Description**:

Bitwise exclusive ‘Or’ two operand values and place the result in the target register, updating status flags. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## LSR – Logical Shift Right

**Description**:

Right shift one operand value by a second operand value and place the result in the target register. Zeros are shifted into the most significant bits. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## MOV – Move Register to Register

**Description**:

This instruction moves from one general-purpose register to another general-purpose register. It is an alternate mnemonic for the OR instruction where Rb is assumed to be R0.

**Formats Supported**: RR

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## MVF – Move From Register

**Description**:

This instruction moves from a specific use register to a general-purpose register.

**Formats Supported**: RR

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

|  |  |  |
| --- | --- | --- |
| Ra5 | Register |  |
| 0 to 3 | L0 to L3 | link registers |
| 4 | XL | exception link register |
| 5,6 |  | reserved |
| 7 | CA | all compare registers combined |
| 8 to 15 | C0 to C7 | compare result registers |
| 16 to 23 |  | reserved (vector mask) |
| 24 | SSP | supervisor stack pointer |
| 25 | HSP | hypervisor stack pointer |
| 26 | MSP | machine stack pointer |
| 27 to 31 |  | reserved |

## MVT – Move To Register

**Description**:

This instruction moves a general-purpose register into one of the specific use registers.

**Formats Supported**: RR

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

|  |  |  |
| --- | --- | --- |
| Rt5 | Register |  |
| 0 to 3 | L0 to L3 | link registers |
| 4 | XL | exception link register |
| 5,6 |  | reserved |
| 7 | CA | all compare registers combined |
| 8 to 15 | C0 to C7 | compare result registers |
| 16 to 23 |  | reserved (vector mask) |
| 24 | SSP | supervisor stack pointer |
| 25 | HSP | hypervisor stack pointer |
| 26 | MSP | machine stack pointer |
| 27 to 31 |  | reserved |

## MUL – Multiplication

**Description**:

Multiply two operand values and place the result in the target register. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value. Both operands are treated as signed values.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## OR – Bitwise ‘Or’

**Description**:

Bitwise ‘Or’ two operand values and place the result in the target register, updating status flags. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## ORIS – Or Immediate Shifted

**Description**:

Bitwise ‘or’ an immediate value to bits 22 to 51 of register Ra5 and place the result in target register Rt5. Use this instruction when performing an addition with a full 52-bit constant or when building a value in a register. The constant used is zero extended on the right hand side.

**Formats Supported**: RIS

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## PERM – Permute Bytes

**Description**:

This instruction allows any combination of bytes in a source register to be copied to a target register. There are four two-bit fields which specify which source bytes to copy to the target. The source fields may be either a constant specified in the instruction, or the low order eight bits of register Rb. Field S0 indicates the source byte for target byte position 0. S1 indicates the source byte for target byte position 1. S2 and S3 work similarly for the last two target bytes. There are many interesting possibilities with this instruction. A single source byte could be copied to all target byte positions for instance. Or the order of bytes in a word could be reversed.

**Formats Supported**: RR, RI8

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## ROL – Rotate Left

**Description**:

Rotate left one operand value by a second operand value and place the result in the target register, updating status flags. The most significant bits are placed in the least significant bits. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## ROR – Rotate Right

**Description**:

Rotate right one operand value by a second operand value and place the result in the target register, updating status flags. The least significant bits are placed in the most significant bits. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value.

**Formats Supported**: RR, RI8

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

## SUB – Subtraction

**Description**:

Subtract two operand values and place the result in the target register, updating status flags. The first operand must be in a register specified by the Ra5 field of the instruction. The second operand may be either a register specified by the Rb5 field of the instruction, or an immediate value. The subtraction instruction has an alternate mnemonic CMP which updates the flags differently when the target register is R0.

**Formats Supported**: RR, RI8, RI22, RI35

**Execution Units**: ALU

**Clock Cycles**: 0.5

**Exceptions**: none

# Memory Operations

## LD – Load Data (52 bits)

**Description**:

Data is loaded from the memory address which is either the sum of Ra and an immediate value or the sum of Ra and Rb. Both register indirect with displacement and indexed addressing are supported.

**Formats Supported**: RR, RI8, RI22, RI35

**Operation:**

Rt = Memory52[d+Ra]

or

Rt = Memory52[Ra+Rb]

**Execution Units**: Mem

**Clock Cycles**: 4 if data is in the cache.

**Exceptions**: none

## LDB – Load Data Byte (13 bits)

**Description**:

Data is loaded from the memory address which is the sum of Ra and an immediate value. Only register indirect with displacement addressing is supported.

**Formats Supported**: RI36

**Flags Affected**: n z

**Operation:**

Rt = Memory13[d+Ra]

**Execution Units**: Mem

**Clock Cycles**: 4 if data is in the cache.

**Exceptions**: none

## ST – Store Data (52 bits)

**Description**:

Data is stored to the memory address which is either the sum of Ra and an immediate value or the sum of Ra and Rb. Both register indirect with displacement and indexed addressing are supported.

**Formats Supported**: RR, RI8, RI22, RI35

**Flags Affected**: none

**Operation:**

Memory52[d+Ra] = Rs

or

Memory52[Ra+Rb] = Rs

**Execution Units**: Mem

**Clock Cycles**: 4 if data is in the cache.

**Exceptions**: none

## STB – Store Data Byte (13 bits)

**Description**:

Data is stored to the memory address which is the sum of Ra and an immediate value. Only register indirect with displacement addressing is supported.

**Formats Supported**: RI35

**Flags Affected**: none

**Operation:**

Memory13[d+Ra] = Rs

or

Memory13[Ra+Rb] = Rs

**Execution Units**: Mem

**Clock Cycles**: 4 if data is in the cache.

**Exceptions**: none

# Flow Control (Branch Unit) Operations

There are only signed versions of branches since the branch is primarily due to the output of the compare instruction which outputs values of -1,0 or +1.

## BEQ – Branch if Equal

**Description**:

This instruction branches to the target address if the compare results register is zero. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction. This branch may be either statically or dynamically predicted.

**Formats Supported**: Bcc

|  |  |
| --- | --- |
| P2 | Meaning |
| 0 | no prediction (dynamic prediction) |
| 1 | reserved |
| 2 | predict not taken (static prediction) |
| 3 | predict taken (static prediction) |

**Operation:**

If (cr==0)

PC = NextPC + Displacement12

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## BGE – Branch if Greater Than or Equal

**Description**:

This instruction branches to the target address if the compare results register is greater than or equal to zero (0 or +1). The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

**Formats Supported**: Bcc

**Operation:**

If (cr >= 0)

PC = NextPC + Displacement12

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## BGT – Branch if Greater Than

**Description**:

This instruction branches to the target address if the compare results register is greater than zero (+1). The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

**Formats Supported**: Bcc

**Operation:**

If (cr > 0)

PC = NextPC + Displacement12

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## BLE – Branch if Less Than or Equal

**Description**:

This instruction branches to the target address if the compare results register is less than or equal to zero. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

**Formats Supported**: Bcc

**Operation:**

If (cr <= 0)

PC = NextPC + Displacement12

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## BLT – Branch if Less Than

**Description**:

This instruction branches to the target address if the compare results register is less than zero (-1 or -2). The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

**Formats Supported**: Bcc

**Operation:**

If (cr < 0)

PC = NextPC + Displacement12

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## BNE – Branch if Not Equal

**Description**:

This instruction branches to the target address if the compare results register is not zero. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction.

**Formats Supported**: Bcc

**Operation:**

If (cr <> 0)

PC = NextPC + Displacement12

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## BRA – Branch Always

**Description**:

This instruction branches unconditionally to the target address. The target address is the address of the next instruction plus a 12-bit displacement specified in the instruction. The prediction bits should be set to indicate a static prediction of taken so that the branch does not consume history table resources.

**Formats Supported**: Bcc

**Operation:**

PC = NextPC + Displacement12

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## BRK – Break

**Description**:

This instruction initiates the processor break routine. The cause code register is set to four. The program counter is reset to $FFFFFFFFE0000 and instructions begin executing.

**Formats Supported**: BRK

**Operation:**

CAUSE = 40h | Const4

OLS = OLS << 2

DLS = DLS << 2

XL = PC + 1

PC = $FFFFFFFFE0000

**Execution Units**: Branch

**Clock Cycles**:

**Exceptions**: none

**Notes**:

## JMP – Jump

**Description**:

This instruction is an alternate mnemonic for the JAL instruction where the link register is assumed to be L0. JMP transfers execution of instructions to the address specified by the instruction. The target address may be either a 43-bit absolute address or an address contained in a register. For absolute address mode only the low order 43 bits of the program counter are affected. The upper nine bits of the program counter remain the same.

**Formats Supported**: ABS43, R

**Flags Affected**: none

**Operation:**

PC = Address43

or

PC = Ra

**Execution Units**: Branch

**Clock Cycles**: 1

**Exceptions**: none

## JAL – Jump and Link

**Description**:

Store the return address in the specified link register then jump to the address specified as an absolute constant or the contents of register Ra4. The target address may be either a 43-bit absolute address or an address contained in a register. For absolute address mode only the low order 43 bits of the program counter are affected. The upper nine bits of the program counter remain the same. Note that only registers R0 to R15 may be specified as containing a jump target. This is due to limitations in the instruction encoding.

**Formats Supported**: ABS43, R

**Flags Affected**: none

**Operation:**

Lk = NextPC

PC = Address43

or

PC = Ra

**Execution Units**: Mem

**Clock Cycles**: 0.5

**Exceptions**: none

**Notes**:

The next PC is either the current PC plus four when absolute addressing is used, or the current PC plus one if register indirect addressing is used.

## MRK – Marker

**Description**:

This instruction is treated by the processor as a NOP operation. It is used to mark positions in a software emulator or simulator.

**Formats Supported**: MRK

**Flags Affected**: none

**Operation:** none

**Execution Units**: Branch

**Clock Cycles**: 0.5

**Exceptions**: none

## NMI – Non-Maskable Interrupt

**Description**:

This instruction initiates the processor exception handling routine. The cause code register is set to 60h. The program counter is reset to $FFFFFFFFE0000 and instructions begin executing.

**Formats Supported**: BRK

**Operation:**

CAUSE = 60h

OLS = OLS << 2

DLS = DLS << 2

IMS = (IMS << 3) | 7

XL = PC + 1

PC = $FFFFFFFFE0000

**Execution Units**: Branch

**Clock Cycles**:

**Exceptions**: none

**Notes**:

## NOP – No Operation

**Description**:

This instruction is treated by the processor as a NOP operation.

**Formats Supported**: MRK

**Flags Affected**: none

**Operation:** none

**Execution Units**: Branch

**Clock Cycles**: 0.5

**Exceptions**: none

## PFI – Poll for Interrupt

**Description**:

The PFI instruction tests for the presence of an interrupt and performs the interrupt routine if an interrupt is present. If no interrupt is present a NOP operation is performed, and the program continues with the next instruction. PFI does not check for a non-maskable (NMI) interrupt or a reset (RST). Processing for the interrupt routine begins at the universal exception handler address of $FFFFFFFFE0000.

PFI may scan three interrupt signalling lines, which lines to scan are specified by a bit-mask in the instruction.

**Formats Supported**: PFI

**Flags Affected**: none

**Operation:**

If (IRQ & scan mask)

Cause Code = 50h | IRQ Level

OLS = OLS << 3

DLS = DLS << 3

IMS = (IMS << 3) | 7

PLS = PLS << 13

XLR = PC + 1;

PC = $FFFFFFFFE0000

Else

NOP

**Execution Units**: Fetch stage

**Clock Cycles**:

**Exceptions**: none

**Notes**:

## REX – Redirect Exception

**Description**:

This instruction redirects an exception from an operating level to a lower operating level and privilege level. If the target operating level is hypervisor then the hypervisor privilege level (1) is set. If the target operating level is supervisor, then one of the supervisor privilege levels must be chosen. This instruction if successful jumps to the target exception handler and does not return. If this instruction fails execution will continue with the next instruction.

This instruction may fail if exceptions are not enabled at the target level.

When redirecting the target privilege level is set to the bitwise ‘or’ of an immediate constant specified in the instruction and register Ra. One of these two values should be zero. The result should be a value in the range 2 to 8191. The instruction will not allow setting the privilege level numerically less than the operating level.

The exception handler address is $FFFFFFFFE0000

The cause (cause) and bad address (badaddr) registers of the originating level are copied to the corresponding registers in the target level.

The REX instruction also specifies the interrupt mask level to set for further processing.

Attempting to redirect the operating level to the machine level (0) will be ignored. The instruction will be treated as a NOP with the exception of setting the interrupt mask register.

**Instruction Format**: REX

|  |  |  |
| --- | --- | --- |
| Tgt3 |  |  |
| 0 | not used | 0 to 8191 |
| 1 | redirect to hypervisor level | 0 to 8191 |
| 2 | redirect to supervisor level1 | 16 to 8191 |
| 3 | redirect to supervisor level2 | 128 to 8191 |
| 4 | redirect to supervisor level3 | 1024 to 8191 |
| 5 |  | 7168 to 8191 |

**Clock Cycles**: 4

**Execution Units: Branch**

Example:

|  |
| --- |
| REX 5,12,r0 ; redirect to supervisor handler, privilege level two  ; If the redirection failed, exceptions were likely disabled at the target level.  ; Continue processing so the target level may complete it’s operation.  RTI ; redirection failed (exceptions disabled ?) |

**Notes**:

Since all exceptions are initially handled at the machine level the machine level handler must check for disabled lower level exceptions.

## RST – Reset Processor

**Description**:

This instruction initiates the processor reset routine. The cause code register is set to 70h. The program counter is reset to $FFFFFFFFE0000 and instructions begin executing.

**Formats Supported**: RST

**Operation:**

**Execution Units**: Branch

**Clock Cycles**:

**Exceptions**: none

**Notes**:

## RTI – Return from Interrupt Subroutine

**Description**:

Restore the previous interrupt and operating level and transfer program execution back to the address in the exception link register. One of the first sixteen semaphore registers may also be cleared. Semaphore register zero is always cleared by this instruction.

**Formats Supported**: RTI

**Flags Affected**: none

**Operation:**

OLS = OLS >> 3

DLS = DLS >> 3

IMS = IMS >> 3

PLS = PLS >> 13

Semaphore[0] = 0

Semaphore[Sema4] = 0

PC = XL

**Execution Units**: Mem

**Clock Cycles**:

**Exceptions**: none

**Notes**:

## RTS – Return from Subroutine

**Description**:

Transfer program execution to an address stored in a link register. The link register will have been previously set by a subroutine call operation.

**Formats Supported**: RTS

**Flags Affected**: none

**Operation:**

PC = Lk

**Execution Units**: Mem

**Clock Cycles**: 0.5

**Exceptions**: none

**Notes**:

## SNR – Sequence Number Reset

**Description**:

This instruction initiates the processor exception handling routine. The cause code register is set to 61h for the sequence number reset interrupt which is non-maskable. The program counter is reset to $FFFFFFFFE0000 and instructions begin executing. A sequence number reset exception is generated internally by the core based on the tick count which is about to roll-over in the sequence number bits. (A portion of the tick count is used for sequence numbering).

**Formats Supported**: BRK

**Operation:**

CAUSE = 61h

OLS = OLS << 3

DLS = DLS << 3

IMS = (IMS << 3) | 7

PLS = PLS << 13

XL = PC + 1

PC = $FFFFFFFFE0000

**Execution Units**: Branch

**Clock Cycles**:

**Exceptions**: none

**Sample Code**:

|  |  |
| --- | --- |
| CSRRD r1,r0,#CAUSE  CMP c0,r1,#61h  BNE c0,.notSNR  NOP  … (61 more NOP’s)  NOP  CSRRSI r0,#1,#SNRREG  RTI  .notSNR: | ; read the cause  ; is it a sequence reset?  ; no go do other exception code  ; use 63 nops to flush the processor queue  ;  ;  ; pulse the sequence number reset  ; and it’s finished so return. |

**Notes**:

## STP – Stop

**Description**:

This instruction is used to stop the processor by stopping the clock internally. The stopped state may be exited by a reset or nmi.

**Formats Supported**: STP

**Flags Affected**: none

**Operation:** none

**Execution Units**: Branch

**Clock Cycles**: 0.5

**Exceptions**: none

## WAI – Wait for Interrupt

**Description**:

The WAI instruction waits for an interrupt to occur by holding the program counter steady. This instruction is similar to the PFI instruction except that it stops and waits for an interrupt whereas PFI doesn’t wait. WAI does not check for a non-maskable (NMI) interrupt or a reset (RST).

**Formats Supported**: WAI

**Flags Affected**: none

**Operation:**

If (IRQ)

Cause Code = 50h | IRQ Level

OLS = OLS << 3

DLS = DLS << 3

IMS = (IMS << 3) | 7

PLS = PLS << 13

XLR = PC + 1;

PC = $FFFFFFFFE0000

Else

PC = PC

**Execution Units**: Fetch stage

**Clock Cycles**:

**Exceptions**: none

**Notes**:

# Oddball

An assortment of instructions that are not executed on one of the regular functional units.

## CACHE – Cache Command

CACHE Cmd, [Rn]

**Description:**

This instruction commands the cache controller to perform an operation. Commands are summarized in the command table below. Commands may be issued to both the instruction and data cache at the same time.

**Instruction Formats**: CACHE

**Commands:**

|  |  |  |
| --- | --- | --- |
| IC2 | Mne. | Operation |
| 0 | NOP | no operation |
| 1 | invline | invalidate line associated with given address |
| 2 | invall | invalidate the entire cache (address is ignored) |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| DC3 | Mne. | Operation |
| 0 | NOP | no operation |
| 1 | enable | enable cache (instruction cache is always enabled) |
| 2 | disable | not valid for the instruction cache |
| 3 | invline | invalidate line associated with given address |
| 4 | invall | invalidate the entire cache (address is ignored) |
|  |  |  |

Notes:

# Instruction Formats

## Arithmetic / Logical

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ADD |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | Rt5 | 04h | ADD Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | Rt5 | 04h | ADD Rt,Ra,#imm8 | 2 |
|  | | Imm22 | | | | Ra5 | Rt5 | 14h | ADD Rt,Ra,#imm22 | 3 |
| Imm35 | | | | | | Ra5 | Rt5 | 24h | ADD Rt,Ra,#imm35 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| SUB |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | Rt5 | 05h | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | Rt5 | 05h | Rt,Ra,#imm8 | 2 |
|  | | Imm22 | | | | Ra5 | Rt5 | 15h | Rt,Ra,#imm22 | 3 |
| Imm35 | | | | | | Ra5 | Rt5 | 25h | Rt,Ra,#imm35 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CMP |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | 1 | Ct3 | 06h | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | 1 | Ct3 | 06h | Rt,Ra,#imm8 | 2 |
|  | | Imm22 | | | | Ra5 | 1 | Ct3 | 16h | Rt,Ra,#imm22 | 3 |
| Imm35 | | | | | | Ra5 | 1 | Ct3 | 26h | Rt,Ra,#imm35 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CMPU |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | 1 | Ct3 | 07h | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | 1 | Ct3 | 07h | Rt,Ra,#imm8 | 2 |
|  | | Imm22 | | | | Ra5 | 1 | Ct3 | 17h | Rt,Ra,#imm22 | 3 |
| Imm35 | | | | | | Ra5 | 1 | Ct3 | 27h | Rt,Ra,#imm35 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MUL |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | Rt5 | 0Eh | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | Rt5 | 0Eh | Rt,Ra,#imm8 | 2 |
|  | | Imm22 | | | | Ra5 | Rt5 | 1Eh | Rt,Ra,#imm22 | 3 |
| Imm35 | | | | | | Ra5 | Rt5 | 2Eh | Rt,Ra,#imm35 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| AND |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | Rt5 | 08h | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | Rt5 | 08h | Rt,Ra,#imm6 | 2 |
|  | | Imm22 | | | | Ra5 | Rt5 | 18h | Rt,Ra,#imm14 | 3 |
| Imm35 | | | | | | Ra5 | Rt5 | 28h | Rt,Ra,#imm30 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BIT |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | ~ | Ct3 | 55h | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | ~ | Ct3 | 55h | Rt,Ra,#imm6 | 2 |
|  | | Imm22 | | | | Ra5 | ~ | Ct3 | 65h | Rt,Ra,#imm14 | 3 |
| Imm35 | | | | | | Ra5 | ~ | Ct3 | 75h | Rt,Ra,#imm30 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OR |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | Op3 | Rb5 | Ra5 | Rt5 | 09h | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | Rt5 | 09h | Rt,Ra,#imm6 | 2 |
|  | | Imm22 | | | | Ra5 | Rt5 | 19h | Rt,Ra,#imm14 | 3 |
| Imm35 | | | | | | Ra5 | Rt5 | 29h | Rt,Ra,#imm30 | 4 |

|  |  |
| --- | --- |
| Op3 | Function |
| 0 | normal ‘Or’ operation |
| 1 | zero extend byte result |
| 2 | zero extend wyde result |
| 3 | reserved |
| 4 | reserved |
| 5 | sign extend byte (13 bits) |
| 6 | sign extend wyde (26 bits) |
| 7 | reserved |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| EOR |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | Rb5 | Ra5 | Rt5 | 0Ah | Rt,Ra,Rb | 2 |
|  | | | 1 | Imm8 | | Ra5 | Rt5 | 0Ah | Rt,Ra,#imm6 | 2 |
|  | | Imm22 | | | | Ra5 | Rt5 | 1Ah | Rt,Ra,#imm14 | 3 |
| Imm35 | | | | | | Ra5 | Rt5 | 2Ah | Rt,Ra,#imm30 | 4 |

## Shifted Immediate

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ADDIS | |  | | | | | | Opcode |  | Bytes |
| 51 39 | | | 38 26 | 25 22 | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
| ~5 | Imm30 | | | | | Ra5 | Rt5 | 23h | Rt,Ra,#imm35 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ANDIS | |  | | | | | | Opcode |  | Bytes |
| 51 39 | | | 38 26 | 25 22 | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
| ~5 | Imm30 | | | | | Ra5 | Rt5 | 22h | Rt,Ra,#imm35 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ORIS | |  | | | | | | Opcode |  | Bytes |
| 51 39 | | | 38 26 | 25 22 | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
| ~5 | Imm30 | | | | | Ra5 | Rt5 | 2Ch | Rt,Ra,#imm35 | 4 |

## Shift Operations

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ASL |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | | Rb5 | Ra5 | Rt5 | 0Ch | Rt,Ra,Rb | 2 |
|  | | | 1 | ~2 | Imm6 | | Ra5 | Rt5 | 0Ch | Rt,Ra,#imm6 | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ROL |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | | Rb5 | Ra5 | Rt5 | 1Ch | Rt,Ra,Rb | 2 |
|  | | | 1 | ~2 | Imm6 | | Ra5 | Rt5 | 1Ch | Rt,Ra,#imm6 | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LSR |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | | Rb5 | Ra5 | Rt5 | 0Dh | Rt,Ra,Rb | 2 |
|  | | | 1 | ~2 | Imm6 | | Ra5 | Rt5 | 0Dh | Rt,Ra,#imm6 | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ROR |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | | Rb5 | Ra5 | Rt5 | 1Dh | Rt,Ra,Rb | 2 |
|  | | | 1 | ~2 | Imm6 | | Ra5 | Rt5 | 1Dh | Rt,Ra,#imm6 | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ASR |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | | Rb5 | Ra5 | Rt5 | 2Dh | Rt,Ra,Rb | 2 |
|  | | | 1 | ~2 | Imm6 | | Ra5 | Rt5 | 2Dh | Rt,Ra,#imm6 | 2 |

## Load and Store Instructions

|  |  |
| --- | --- |
| S3 | Scale by |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |
| 4 to 7 | reserved |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LD |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 17 | | | 16 12 | 11 7 | 6 0 |  |  |
|  | | | 0 | S3 | Rb5 | Ra5 | Rt5 | 50h | Rt.[Ra+Rb] | 2 |
|  | | | 1 | Disp8 | | Ra5 | Rt5 | 50h | Rt,d8[Ra] | 2 |
|  | | Disp22 | | | | Ra5 | Rt5 | 60h | Rt,d22[Ra] | 3 |
| Addr35 | | | | | | Ra5 | Rt5 | 70h | Rt,d35[Ra] | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| LDB |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 17 | | | 16 12 | 11 7 | 6 0 |  |  |
|  | | | 0 | S3 | Rb5 | Ra5 | Rt5 | 51h | Rt.[Ra+Rb] | 2 |
|  | | | 1 | Disp8 | | Ra5 | Rt5 | 51h | Rt,d8[Ra] | 2 |
|  | | Disp22 | | | | Ra5 | Rt5 | 61h | Rt,d22[Ra] | 3 |
| Addr35 | | | | | | Ra5 | Rt5 | 71h | Rt,d35[Ra] | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ST |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 17 | | | 16 12 | 11 7 | 6 0 |  |  |
|  | | | 0 | S3 | Rb5 | Ra5 | Rs5 | 58h | Rs.[Ra+Rb] | 2 |
|  | | | 1 | Disp8 | | Ra5 | Rs5 | 58h | Rs,d8[Ra] | 2 |
|  | | Disp22 | | | | Ra5 | Rs5 | 68h | Rs,d22[Ra] | 3 |
| Addr35 | | | | | | Ra5 | Rs5 | 78h | Rs,d35[Ra] | 4 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| STB |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 17 | | | 16 12 | 11 7 | 6 0 |  |  |
|  | | | 0 | S3 | Rb5 | Ra5 | Rs5 | 59h | Rs.[Ra+Rb] | 2 |
|  | | | 1 | Disp8 | | Ra5 | Rs5 | 59h | Rs,d8[Ra] | 2 |
|  | | Disp22 | | | | Ra5 | Rs5 | 69h | Rs,d22[Ra] | 3 |
| Addr35 | | | | | | Ra5 | Rs5 | 79h | Rs,d35[Ra] | 4 |

## Flow Control

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| JAL | Flags: | | | | | | Bytes |
|  | | Address43 | | L2 | 42h | JAL abs43 | 4 |
|  | | | Ra4 | L2 | 48h | JAL [Ra] | 1 |

### {RTGRP}

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RTS |  | ~2 | L2 | 0 | 44h | RTS | 1 |
| RTI |  | Sema4 | | 1 | 44h | RTI | 1 |
| RTD |  | ~4 | | 2 | 44h | RTD | 1 |

### {WAIGRP}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| PFI |  | Sigmsk4 | 0 | 02h | PFI | 1 |
| IRQ |  | ~4 | 1 | 02h | IRQ | 1 |
| WAI |  | Sigmsk4 | 2 | 02h | WAI | 1 |
| IRQ |  | ~4 | 3 | 02h | IRQ | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| STP |  | ~4 | 0 | 43h | STP | 1 |
| NOP |  | ~4 | 1 | 43h | NOP | 1 |
| MRK |  | Const4 | 2 | 43h | MRK | 1 |
|  |  | ~4 | 3 | 43h |  |  |
| MEMSB |  | 0 | 3 | 43h |  |  |
| MEMDB |  | 1 | 3 | 43h |  |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| BEQ |  | Disp12 | Cr3 | P2 | 0 | 40h | BEQ disp | 2 |
| BNE |  | Disp12 | Cr3 | P2 | 1 | 40h | BNE disp | 2 |
| BGT |  | Disp12 | Cr3 | P2 | 2 | 40h | BGT disp | 2 |
| BLT |  | Disp12 | Cr3 | P2 | 3 | 40h | BLT disp | 2 |
| BGE |  | Disp12 | Cr3 | P2 | 0 | 41h | BGE disp | 2 |
| BLE |  | Disp12 | Cr3 | P2 | 1 | 41h | BLE disp | 2 |
| BRA |  | Disp12 | Cr3 | P2 | 2 | 41h | BRA disp | 2 |

### {BRKGRP}

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| RST |  | ~4 | | | 3 | 00h | RST | 1 |
| NMI |  | ~2 | | 0 | 2 | 00h | NMI | 1 |
| SNR |  | ~2 | | 1 | 2 | 00h | SNR | 1 |
| IRQ |  | ~ | i3 | | 1 | 00h | IRQ | 1 |
| BRK |  | Const4 | | | 0 | 00h | BRK | 1 |

## Shuffle

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PERM |  | | | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | | 21 17 | | | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | | Rb5 | | | Ra5 | Rt5 | 20h | Rt,Ra,Rb | 2 |
|  | | | 1 | S3 | S2 | | S1 | S0 | Ra5 | Rt5 | 20h | Rt,Ra,#imm8 | 2 |

## Control and Status Register Access

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CSR | 38 36 | 3533 | 32 29 | 28 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | OP3 | OL3 | ~4 | Regno12 | Ra5 | Rt5 | 01h | CSR | 3 |

## Register Move

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MOV |  | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | 05 | Ra5 | Rt5 | 09h | Rt,Ra,Rb | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MT |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | | 6 0 |  |  |
|  | |  | 0 | ~3 | ~5 | Ra5 | 03 | L2 | 4Ah | Lt,Ra | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MT |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | | 6 0 |  |  |
|  | |  | 0 | ~3 | ~5 | Ra5 | 12 | C3 | 4Ah | Ct,Ra | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MF |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | | 11 7 | 6 0 |  |  |
|  | |  | 0 | ~3 | ~5 | 0 | L2 | Rt5 | 5Ah | Rt,La | 2 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| MT |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 22 | | 21 17 | 16 12 | 11 7 | | 6 0 |  |  |
|  | |  | 0 | ~3 | ~5 | Ra5 | 13 | 32 | 4Ah | Rt,CA | 2 |

## Oddball

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CACHE |  | | | | | | | | Opcode |  | Bytes |
| 51 39 | | 38 26 | 25 17 | | | | 16 12 | 11 7 | 6 0 |  |  |
|  | | | 0 | ~3 | dc3 | ic2 | Ra5 | ~5 | 7Ah | <cmd>,[Ra] | 2 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| REX | |  | | | | | | Opcode |  | Bytes |
| 51 39 | 38 33 | | 32 29 | 28 26 | 25 13 | 12 | 11 7 | 6 0 |  |  |
|  | ~6 | | IM4 | Tgt3 | PL13 | ~ | Ra5 | 6Ah | Ra,#PL,#Tgt,#IM | 3 |

# Opcode Maps

## Root Level

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BRK | CSR | {WAIGRP} | DIV | ADD | SUB | CMP | CMPU | AND | OR | EOR |  | ASL | LSR | MUL |  |
| 1x | {string} | {string} | {VP} |  | ADD | SUB | CMP | CMPU | AND | OR | EOR |  | ROL | ROR | MUL |  |
| 2x | PERM |  | ANDIS | ADDIS | ADD | SUB | CMP | CMPU | AND | OR | EOR |  | ORIS | ASR | MUL |  |
| 3x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| 4x | {Branch} | {Branch} | JAL | STP | {RTGRP} |  |  |  | JAL [R] |  | MTx |  | SEQ | SLT | FSLT | FADD |
| 5x | LD | LDB |  |  |  | BIT |  |  | ST | STB | MFx |  | SNE | SLE | FSLE | FSUB |
| 6x | LD | LDB |  |  |  | BIT |  |  | ST | STB | REX |  | FSEQ | SLTU |  | FMUL |
| 7x | LD | LDB |  |  |  | BIT |  |  | ST | STB | CACHE |  | FSNE | SLEU |  | FDIV |

# Appendix

## Register Tags

As part of the internal workings of the core, it tracks registers using a seven-bit register tag associated with each register in the programming model.

|  |  |  |
| --- | --- | --- |
| Tag | Register |  |
| 0 to 31 | r0 to r31 general-purpose integer registers |  |
| 32 to 63 | f0 to f31 general-purpose floating-point registers |  |
| 64 to 95 | reserved for vector registers |  |
| 96 to 100 | link registers |  |
| 101 | machine status register |  |
| 102 | reserved for vector length |  |
| 103 | packed compare results |  |
| 104 to 111 | compare results registers |  |
| 112 to 119 | reserved for vector mask registers |  |
| 120 | supervisor SP |  |
| 121 | hypervisor SP |  |
| 122 | machine SP |  |
|  |  |  |