# FT64v3 ISA

## Overview

The FT64v3 ISA is in part organized around the idea of a simple compiler at the expense of some hardware cost. The ISA makes use of a unified register file for integer and floating-point registers. This makes it a bit easier to manage register usage by the compiler. Having a large number of registers directly available means the compiler doesn’t have to be as sophisticated in it’s allocation of registers and good performance is still possible. However, increasing the number of bits required to represent registers leads to an increase in the size of instructions.

This instruction set uses a fixed 36-bit format instruction as its base because 32 bits isn’t quite enough. 36 bits may seem like an odd number, but the number of bits in an instruction is somewhat irrelevant as long as all instructions can fit on a cache line. Instruction caches are loaded in terms of cache lines. Sixteen 36-bit instructions will fit into a 576-bit cache line with no wasted space. 576 bits is 72 bytes, in range with typical cache line sizes. It is also a multiple of 64 bits, the width of the data bus.

## Design Considerations

### Compression

A number of contemporary processors make use of compressed instructions sets to improve code density as a way to compete with other designs using byte codes. Narrower instructions make better use of memory and caches. They have benefits in the form of performance and power consumption. They are so important to contemporary designs that RiSC-V for instance reserves three quarters of the opcode space for compressed instructions. Other processors have mode switching instructions to enable compressed instruction sets. This design directly supports compressed instructions; one half of the instruction space is allocated for compressed instructions. Half of the compressed instruction space is reserved for memory operations, the other half for other operations.

Typically, a compressed version of the instruction set relates to the uncompressed version in terms of registers used and field specifications. The register and field specifications map more or less directly to fields in the expanded instruction. The mapping is designed to take a minimal amount of logic to convert into expanded instructions so that the decompression doesn’t significantly impact the cycle time of the processor. A subset of registers may be supported along with a subset of operations that the processor provides.

Significant analysis has gone into which instructions should be available in a compressed form. It is often the same subset of instructions that are best compressed. These instructions typically include an add of a small amount to a register, and frame pointer or stack pointer relative loading / store instructions.

For this design, rather than develop a format for and completely specify fields for compressed instructions, compressed instructions are simply expanded from look-up tables. An application profiler may be used to select the set of instructions to compress. The compressed instruction set in use is definable at run-time, similar to a micro-code update capability. Using lookup tables may reduce the number of compressed instructions available, but the best selection for a given app(s) can be chosen. Decompressing instructions becomes a simple indexed table lookup rather than logic.

A parcel size one half of the chosen base instruction size or 18-bit parcels offer good compression. Compressed instructions use a 8k entry lookup table to decompress the instructions. It is envisioned that decompressing these instructions may require an extra clock cycle for access to the lookup table. Often the instruction will be able to queue in the pipeline without knowing the exact details of the instruction. Looking up the instruction is similar to having to fetch values from the register file. The exact opcode can be filled in a cycle later after lookup takes place.

A consequence of the delay in decompressing an instruction is that a compressed branch type instruction may have an additional clock cycle of delay causing a reduction in performance.

# Programming Model

### Register Usage Convention

The register usage convention probably has more to do with software than hardware. Excepting a couple of special cases, the registers are general purpose in nature.

R0 always has the value zero in all register sets.

|  |  |  |  |
| --- | --- | --- | --- |
| Register |  | Description / Suggested Usage | Saver |
| r0 | r0 | always reads as zero |  |
| r1-r4 | v0-v3 | return values / exception | caller |
| r5-r20 | t0-t15 | temporaries | caller |
| r21-r34 |  | register variables | callee |
| r35-r47 | a0-a12 | function arguments | caller |
| r52-r55 | c0-c3 | assembler usage |  |
| r56 |  | type number / function argument | caller |
| r57 | cp | class pointer / function argument | caller |
| r58 | tp | thread pointer | callee |
| r59 | gp | global pointer |  |
| r60 | xh | exception link register | caller |
| r61 | ra | return address / link register | caller |
| r62 | fp | base / frame pointer | callee |
| r63 | sp | stack pointer | callee |

### Program Counter

The program counter identifies which instruction to execute. The program counter increments by the size of the instruction which varies depending on the instruction. The increment may be overridden using one of the flow control instructions. Instructions are addressed using a two-bit fractional address. Instructions may be aligned on any bit pair.

|  |  |
| --- | --- |
| 61 0 | -1 -2 |
| Address[61..0] | Bp2 |

Notes:

## Exception Cause Codes

The following table outlines the cause code for a given purpose. These codes are specific to FT64. Under the HW column an ‘x’ indicates that the exception is internally generated by the processor; the cause code is hard-wired to that use. An ‘e’ indicates an externally generated interrupt, the usage may vary depending on the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Cause Code |  | HW | Description |  |
|  |  |  |  |  |
| 64 to 127 |  |  | Compressed Instructions Cause Code |  |
|  |  |  | FMTK Scheduler |  |
| 128 |  | e |  |  |
| 129 | KRST | e | Keyboard reset interrupt |  |
| 130 | MSI | e | Millisecond Interrupt |  |
| 131 | TICK | e |  |  |
| 158 | KBD | e | Keyboard interrupt |  |
| 159 | TSI | e | FMTK Time Slice Interrupt |  |
| 3 |  |  | Control-C pressed |  |
| 20 |  |  | Control-T pressed |  |
| 26 |  |  | Control-Z pressed |  |
| 32 | SSM | x | single step |  |
| 33 | DBG | x | debug exception |  |
| 34 | TGT | x | call target exception |  |
| 35 | MEM | x | memory fault |  |
| 36 | IADR | x | bad instruction address |  |
| 37 | UNIMP | x | unimplemented instruction |  |
| 38 | FLT | x | floating point exception |  |
| 39 | CHK | x | bounds check exception |  |
| 40 | DBZ | x | divide by zero |  |
| 41 | OFL | x | overflow |  |
|  | FLT | x | floating point exception |  |
| 49 | EXF | x | Executable fault |  |
| 50 | DWF | x | Data write fault |  |
| 51 | DRF | x | data read fault |  |
| 52 | SGB | x | segment bounds violation |  |
| 53 | PRIV | x | privilege level violation |  |
|  |  |  |  |  |
|  |  |  |  |  |
| 56 | STK | x | stack fault |  |
| 57 | CPF | x | code page fault |  |
| 58 | DPF | x | data page fault |  |
| 60 | DBE | x | data bus error |  |
| 61 | IBE | x | instruction bus error |  |
| 62 | NMI | x | Non-maskable interrupt |  |
|  |  |  |  |  |

# Instruction Set Description

## Overview

This instruction set uses a fixed 36 bit format instruction as its base. 36 bits may seem like an odd number, but the number of bits in an instruction is somewhat irrelevant as long as all instructions can fit on a cache line. Instruction caches are loaded in terms of cache lines. Sixteen 36-bit instructions will fit into a 576-bit (72 byte) cache line with no wasted space and it’s a multiple of 64 bits. Part of the reason for a 36-bit base instruction is that 32 bits isn’t quite enough. This is brought about because of the desire for a base of 64 available general-purpose registers in the instruction set and the desire to support a full instruction set. 64 registers were chosen because it makes the compiler simpler to implement. There is no distinguishing between integer and floating-point register sets.

## Instruction Addresses

Instructions are addressed in 36-bit parcels as if they were composed of two eighteen-bit parcels. An 18-bit parcel size is to allow for a compressed instruction set. In order to keep instruction addresses and data addresses the same, instruction addresses are represented with a two bit fractional portion.

## Formats

Instructions have a fixed 36 bit format. There are only a handful of different instruction formats. The opcode, register read Ra, Rb, and Rc fields always occur in the same place in an instruction to simplify decoding and keep the register read address which is needed prior to enqueue at a fixed decoding location. The Rt field is allowed to float around to make the instruction encoding easier. In a pipelined processor there is usually at least one clock cycle before Rt is used meaning it has time to be shifted around before it’s use.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | Fields | | | | | | | | | | | | | | |  |
|  | | | | Immed13 | | | | | | | | | | | Sz3 | Rt6 | Ra6 | Opcode8 | RI |
|  | | | | Funct6 | | | | | | | ~ | Sz3 | Rt6 | | | Rb6 | Ra6 | Opcode8 | R2 |
|  | | | | 016 | | | | | | | ~ | Sz3 | Funct6 | | | Rt6 | Ra6 | Opcode8 | R1 |
|  | | | | Funct6 | | | | | | | I | Sz3 | Rt6 | | | Rb6 | Ra6 | Opcode8 | SR |
|  | | | | Funct6 | | | | | | | I | Sz3 | Immed6 | | | Rt6 | Ra6 | Opcode8 | SI |
|  | | | | Funct4 | | | | | Me6 | | | | Mb6 | | | Rt6 | Ra6 | Opcode8 | BF |
|  | | Disp17 | | | | | | | | | | | | Cn3 | | Rb6 | Ra6 | Opcode8 | BD |
|  | | Disp17 | | | | | | | | | | | | Cn3 | | Bitno6 | Ra6 | Opcode8 | BB |
|  | | Disp17 | | | | | | | | | | | | Immed9 | | | Ra6 | Opcode8 | BI |
|  | | | | | | | | ~ | | | Cnd3 | | Rc6 | | | Rb6 | Ra6 | Opcode8 | BR |
|  | | | | Funct5 | | | | | | ar2 | | Sc3 | Rt6/Rc6 | | | Rb6 | Ra6 | Opcode8 | MX |
|  | | | | Op2 | | OL3 | | | | Regno11 | | | | | | Rt6 | Ra6 | Opcode8 | CS |
|  | | | | Address28 | | | | | | | | | | | | | | Opcode8 | JC |
|  | | Funct6 | | | ~ | | P | Prc3 | | | | Rm3 | Rt6 | | | Rb6 | Ra6 | Opcode8 | FP |
|  | Funct6 | | ~2 | Vm3 | | | P | Prc3 | | | | Rm3 | Rt6 | | | Rb6 | Ra6 | Opcode8 | VC |

There are a handful of additional formats primarily for control type instructions. See the particular instruction for the exact format used and additional information.

|  |  |
| --- | --- |
| Format | Instruction Group |
| RI | register-immediate and load / store with displacement |
| RR | register-register, two source registers |
| R1 | single source register |
| SR | shift register-register |
| SI | shift register-immediate |
| BF | bitfield |
| BD | branch with displacement |
| BB | branch on bit set / clear, decrement and branch |
| BI | branch equal immediate |
| BR | branch to register |
| MX | memory indexed |
| CS | control and status register access |
| JC | jump and call |
| FP | floating-point |
| VC | vector |

There are quite a few instructions operating on memory. Once volatile and non-volatile loads, stores, read-modify-write, and different sizes of operations are taken into consideration there are about 50 memory ops.

A single bit (bit 6 of the opcode) in the instruction determines if the instruction is a memory instruction or some other type of instruction. Memory instructions are further broken down into three groups – loads, stores, and read-modify-write instructions. Also, easily discernible by looking at the next two bits (bits 5 and 4) of the opcode.

## Compressed Register Specification

4 bit to 6 bit map

|  |  |  |  |
| --- | --- | --- | --- |
| Ra/Rt4 | Ra/Rt6 | Ra/Rt4 | Ra/Rt6 |
| 0 | 1 | 8 | 14 |
| 1 | 2 | 9 | 18 |
| 2 | 3 | 10 | 19 |
| 3 | 4 | 11 | 20 |
| 4 | 5 | 12 | 23 |
| 5 | 11 | 13 | 60 |
| 6 | 12 | 14 | 61 |
| 7 | 13 | 15 | 62 |

## ABS – Absolute Value

**Description:**

This instruction takes the absolute value of a register and places the result in a target register.

**Instruction Format:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 016 | ~ | Sz3 | 46 | Rt6 | Ra6 | 02h8 |

**Clock Cycles:** 1

**Execution Units:** ALU #0 only

**Operation:**

If Ra < 0

Rt = -Ra

else

Rt = Ra

Exceptions: none

Notes:

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word |
| 4 | Byte Parallel |
| 5 | Char Parallel |
| 6 | Half Parallel |
| 7 | Word |

## ADD - Addition

Description:

Add two values. The first operand must be in a register. The second operand may be in a register or may be an immediate value specified in the instruction.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 04h8 |

|  |  |
| --- | --- |
| ISz3 | Immediate Size |
| 0 | 13 |
| 1 | 31 |
| 2 | 49 |
| 3 | 67 |
| 4 to 7 | reserved |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 046 | Ov | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

|  |  |
| --- | --- |
| Ov |  |
| 0 | no overflow |
| 1 | overflow exception if overflow occurred and enabled in AEC |

Overflow works properly only on 64 bit values.

Compressed Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 0h4 | Immed6 | 2h2 | Ra/Rt61 |

1 Ra/Rt6 is not 0 (NOP) or 63 (ADDISP)

|  |  |  |  |
| --- | --- | --- | --- |
| 1h4 | Rb6 | 3h2 | Rt6 |

Rt = Rt + Rb

**Clock Cycles:** 0.5

**Execution Units:** All ALU’s

Exceptions:

The immediate form of the instruction will not cause an exception. The registered form of the instruction may cause an overflow exception if enabled in the AEC register.

Notes:

For sub-word forms the part of the register updated corresponds to the size selected. For instance, if a byte operation is specified then only the low order eight bits of the target register is updated, the remaining bits hold their current value. For parallel operation forms the registers are treated as if they were a group of registers corresponding to the size selected. And the same operation is performed on each part of the register. For parallel forms the entire register is updated.

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word |
| 4 | Byte Parallel |
| 5 | Char Parallel |
| 6 | Half Parallel |
| 7 | Word |

## AND – Bitwise And

Description:

Perform a bitwise ‘and’ operation between operands.

Instruction Format:

The immediate value is sign extended on the left before use.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 08h8 |

Rt = Ra & Rb

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 086 | ~ | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word1 |
| 4 | Byte Parallel1 |
| 5 | Char Parallel1 |
| 6 | Half Parallel1 |
| 7 | Word1 |

1 These codes are redundant with each other. They all have the same effect.

Compressed Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 2h4 | Immed6 | 2h2 | Ra/Rt61 |

1Ra/Rt is non-zero.

The immediate value is sign extended on the left before use.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4h4 | 12 | Rb4 | 22 | 32 | Ra/Rt4 |

Rt = Rt & Rb

Clock Cycles: 0.5

**Execution Units: All** ALUs

**Exceptions:** none

## ASL – Arithmetic Shift Left

Description:

Shift left with arithmetic overflow exception. Bits from the source register Ra are shifted left by the amount in register Rb or an immediate value. A zero is shifted into bit zero. The difference between this instruction and a SHL instruction is that ASL may cause an arithmetic overflow exception. SHL will never cause an exception. In most cases the SHL instruction is preferred.

Instruction Format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11h6 | 0 | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 11h6 | 1 | Sz3 | Imm6 | Rb6 | Ra6 | 02h8 |

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word1 |
| 4 | Byte Parallel1 |
| 5 | Char Parallel1 |
| 6 | Half Parallel1 |
| 7 | Word1 |

Compressed Instruction Format:

There is no compressed instruction format for this instruction. See SHL for a compressed version of the left shift instruction.

Clock Cycles: 1

**Execution Units:** ALU #0 Only

Exceptions:

An overflow exception may result if the bits shifted out from the MSB are not the same as the resulting sign bit and the exception is enabled in the AEC register. Exceptions are only caused by a word size operation.

## ASR – Arithmetic Shift Right

Description:

Bits from the source register Ra are shifted right by the amount in register Rb or an immediate value. The sign bit is shifted into the most significant bits.

Instruction Format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 13h6 | 0 | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 13h6 | 1 | Sz3 | Imm6 | Rb6 | Ra6 | 02h8 |

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word1 |
| 4 | Byte Parallel1 |
| 5 | Char Parallel1 |
| 6 | Half Parallel1 |
| 7 | Word1 |

Compressed Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4h4 | Immed6 | 22 | 12 | Ra/Rt4 |

Clock Cycles: 1

**Execution Units:** ALU #0 Only

Exceptions: none

## BBC –Branch if Bit Clear

Description:

If the specified bit in a register is clear, then a seventeen-bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 23 | 2220 | 19 14 | 13 8 | 7 0 |
| Displacement17 | 13 | Bitno6 | Ra6 | 3Ch8 |

Operation:

if (Ra[bitno]=0)

pc = pc + displacement

Clock Cycles: 1

**Execution Units:** FCU Only

Exceptions: none

## BBS –Branch if Bit Set

Description:

If the specified bit in a register is set, then a seventeen-bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 23 | 2220 | 19 14 | 13 8 | 7 0 |
| Displacement17 | 03 | Bitno6 | Ra6 | 3Ch8 |

Operation:

if (Ra[bitno]=1)

pc = pc + displacement

Clock Cycles: 1

**Execution Units:** FCU Only

Exceptions: none

## Bcc – Conditional Branch

Description:

If the branch condition is true, a seventeen-bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 14 | 13 8 | 7 0 |
| Displacement17 | Cond3 | Rb6 | Ra6 | 38h8 |

A branch to a value computed in a register may be performed using the instruction format shown below. Rc contains the target address which is an absolute address.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~ | Cond3 | Rc6 | Rb6 | Ra6 | 39h8 |

|  |  |  |
| --- | --- | --- |
| Cond3 | Mne. |  |
| 0 | BEQ | Ra = Rb signed |
| 1 | BNE | Ra <> Rb |
| 2 | BLT | Ra < Rb |
| 3 | BGE | Ra >= Rb |
| 4 | BLTU | Ra < Rb (unsigned) |
| 5 | BGEU | Ra >= Rb (unsigned) |
| 6 |  | reserved |
| 7 | BOR | Ra || Rb (either Ra or Rb is true) |

Clock Cycles: 1

**Execution Units:** FCU Only

## BEQ –Branch If Equal

Description:

If registers Ra and Rb contain the same value, a seventeen-bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 14 | 13 8 | 7 0 |
| Displacement17 | 03 | Rb6 | Ra6 | 38h8 |

A branch to a value computed in a register may be performed using the instruction format shown below. Rc contains the target address which is an absolute address.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~ | 03 | Rc6 | Rb6 | Ra6 | 39h8 |

Clock Cycles: 1

**Execution Units:** FCU Only

## BEQI –Branch if Equal Immediate

Description:

If a register is equal to a nine-bit sign extended value then a seventeen bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch. This instruction is useful for implementing case statements based on small values.

Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 39 23 | 22 14 | 13 8 | 7 0 |
| Displacement17 | Immed9 | Ra6 | 3Dh8 |

Operation:

if (Ra = Immediate)

pc = pc + displacement

Clock Cycles: 1

**Execution Units:** FCU Only

## BFCHG – Bitfield Change

Description:

A bitfield is inverted in the target register.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 35 32 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 24 | Me6 | Mb6 | Rt6 | Ra6 | 05h8 |

Clock Cycles: 1

**Execution Units:** ALU #0 Only

Exceptions: none

## BFCLR – Bitfield Clear

Description:

A btifield is cleared in the target register. This is an alternate mnemonic for the bitfield insert instruction.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 34 | Me6 | Mb6 | Rt6 | 06 | 05h8 |

Clock Cycles: 1

**Execution Units:** ALU #0 Only

Exceptions: none

## BGE –Branch If Greater or Equal

Description:

If the value in register Ra is greater than or equal to the value in register Rb, a seventeen-bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 14 | 13 8 | 7 0 |
| Displacement17 | 33 | Rb6 | Ra6 | 38h8 |

A branch to a value computed in a register may be performed using the instruction format shown below. Rc contains the target address which is an absolute address.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~ | 33 | Rc6 | Rb6 | Ra6 | 39h8 |

Clock Cycles: 1

**Execution Units:** FCU Only

## BRK – Hardware / Software Breakpoint

**Description:**

Invoke the break handler routine. The break handler routine handles all the hardware and software exceptions in the core. A cause code is loaded into the CAUSE CSR register. The break handler should read the CAUSE code to determine what to do. The break handler is located by TVEC[0]. This address should contain a jump to the break handler. Note the reset address is $FFFC0100.0. An exception will automatically switch the processor to the machine level operating mode. The break handler routine may redirect the exception to a lower level using the [REX](#_REX_–_Redirect) instruction.

The core maintains an internal eight level interrupt stack for each of the following:

|  |  |  |
| --- | --- | --- |
| Item Stacked | CSR reg |  |
| program counter | pc\_stack |  |
| operating level | ol\_stack | available as a single CSR |
| privilege level | pl\_stack | available as a single CSR |
| interrupt mask | im\_stack | available as a single CSR |
| register set | rs\_stack | available as a single CSR |

If further nesting of interrupts is required the stacks may be copied to memory as they are available from CSR’s.

On stack underflow a break exception is triggered.

**Instruction Format:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 35 27 | 26 22 | 21 18 | 1716 | 15 8 | 7 0 |
| User9 | WS5 | L4 | ~ | Cause Code8 | 00h8 |

WS = word skip 1 = software interrupt – return address is next instruction

WS = 0 = hardware interrupt – return address is current instruction

L4 = the priority level of the hardware interrupt, the priority level at time of interrupt is recorded in the instruction, the interrupt mask will be set to this level when the instruction commits. This field is not used for software interrupts and should be zero.

Cause Code = numeric code associated with the cause of the interrupt.

The User9 field may be used to pass constant data to the break handler.

**Compressed Instruction Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 17 14 | 13 8 | 76 | 5 0 |
| 14 | Cause Code6 | 22 | 06 |

WS = 1

L4 = 0

Cause Code6 translates to the range 64 to 127.

## CALL – Call Subroutine

Description:

Call a subroutine. This instruction is a longer address form than the JAL instruction and has the link register as an implied target for the return address. This is the preferred method to call a subroutine. If a larger address range is required then the JAL instruction must be used.

Instruction Format:

The address of the following instruction is stored in the link register. The high order PC bits are not affected. This allows accessing a subroutine within a 64MB region of memory. Note that with the use of an mmu this address range is often sufficient.

|  |  |
| --- | --- |
| Address[27..0] | 31h8 |

**Execution Units:** FCU

**Clock Cycles:** 1

**Exceptions:** none

Notes:

## CMP – Signed Comparison

Description:

The compare instruction places a 1, 0 or -1 in the target register based on the relationship between the two source operands. If they are equal a zero is placed in the target register, if register Ra is less than the second operand then a -1 is placed in the target register, otherwise a 1 is placed in the target register. The values are treated as signed operands. The immediate constant is sign extended to the width of the machine.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 06h8 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 06h6 | ~ | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

Clock Cycles: 0.5

## CMPU – Unsigned Comparison

Description:

The compare instruction places a 1, 0 or -1 in the target register based on the relationship between the two source operands. If they are equal a zero is placed in the target register, if register Ra is less than the second operand then a -1 is placed in the target register, otherwise a 1 is placed in the target register. The values are treated as unsigned operands. The immediate constant is sign extended to the width of the machine.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 07h8 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 07h6 | ~ | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

Clock Cycles: 0.5

## CSR – Control and Status Access

Description:

The CSR instruction group provides access to control and status registers in the core. For the read-write operation the current value of the CSR is placed in the target register Rt then the CSR is updated from register Ra. The CSR read / update operation is an atomic operation.

Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Op2 | OL3 | Regno11 | Rt6 | Ra6 | 0Eh8 |

|  |  |  |
| --- | --- | --- |
| Op2 |  | Operation |
| 0 | CSRRD | Only read the CSR, no update takes place, Ra 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 |

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

The OL3 field is reserved to specify the operating level. Note that registers cannot be accessed by a lower operating level.

|  |  |  |  |
| --- | --- | --- | --- |
| Regno12 |  | Access | Description |
| 001 | HARTID | R | hardware thread identifier (core number) |
| 002 | TICK | R | tick count, counts every cycle from reset |
| 030-037 | TVEC | RW | trap vector handler address |
| 040 | EPC | RW | exceptioned pc, pc value at point of exception |
| 044 | STATUSL | RWSC | status register, contains interrupt mask, operating level |
| 045 | STATUSH | RW | status register bits 64 to 127 |
| 080-0BF | CODE | RW | code buffers |
| 7F0 | INFO | R | Manufacturer name |
| 7F1 | “ | R | “ |
| 7F2 | “ | R | cpu class |
| 7F3 | “ | R | “ |
| 7F4 | “ | R | cpu name |
| 7F5 | “ | R | “ |
| 7F6 | “ | R | model number |
| 7F7 | “ | R | serial number |
| 7F8 | “ | R | cache sizes instruction (bits 32 to 63), data (bits 0 to 31) |

Clock Cycles: 0.5

## DBNZ –Decrement, Branch if Not Zero

Description:

If the specified register is non-zero then a seventeen bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch. The register is also decremented by one.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 14 | 13 8 | 7 0 |
| Displacement17 | 73 | 06 | Ra6 | 3Ch8 |

Operation:

if (Ra<>0)

pc = pc + displacement

Ra = Ra - 1

Clock Cycles: 1

**Execution Units:** FCU Only

Exceptions: none

## JAL – Jump-And-Link

Description:

Instruction Format:

This instruction loads the program counter with the sum of a register and a constant value specified in the instruction. In addition the address of the instruction following the JAL is stored in the specified target register. This instruction may be used to implement subroutine calls and returns.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 33h8 |

|  |  |
| --- | --- |
| ISz3 | Bits |
| 0 | 13 |
| 1 | 31 |
| 2 | 49 |
| 3 | 67 |
| 4 to 7 | reserved |

**Execution Units:** FCU

Clock Cycles: 1

## LB – Load Byte

Description:

This instruction loads a byte (8 bit) value from memory. The value is sign extended to 64 bits when placed in the target register.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 40h8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 00h6 | A | R | Sc2 | Rt6 | Rb6 | Ra6 | 4Dh8 |

Acquire and release bits determine the ordering of memory operations.

A = acquire – 1 = no following memory operations can take place before this one

R = release – 1 = this memory operation cannot take place before prior ones.

|  |  |
| --- | --- |
| Sc2 | Scale Rb By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

Clock Cycles: 4 minimum depending on memory access time

## LBO – Load Byte Only

Description:

This instruction loads a byte (8 bit) value from memory. Only the lower eight bits of the target register are updated, the upper bits of the register are not affected. This instruction may be used to perform unaligned memory loads when combined with a shift instruction.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 42h8 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 02h6 | A | R | Sc2 | Rt6 | Rb6 | Ra6 | 4Dh8 |

Acquire and release bits determine the ordering of memory operations.

A = acquire – 1 = no following memory operations can take place before this one

R = release – 1 = this memory operation cannot take place before prior ones.

|  |  |
| --- | --- |
| Sc2 | Scale Rb By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

Clock Cycles: 4 minimum depending on memory access time

## LDI – Load Immediate

Description:

This instruction loads an immediate value into a register.

Instruction Format:

This format is an alternate mnemonic for the OR instruction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | 06 | 09h8 |

|  |  |
| --- | --- |
| ISz3 | Bits |
| 0 | 13 |
| 1 | 31 |
| 2 | 49 |
| 3 | 67 |
| 4 to 7 | reserved |

Compressed Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 1h4 | Immed6 | 2h2 | Rt61 |

1 Ra is not 0 (SYS)

**Execution Units:** All ALU’s

**Exceptions:** none

Clock Cycles: 0.5

## OR – Bitwise Or

Description:

Perform a bitwise or operation between operands.

Instruction Format:

The immediate value is sign extended to the left before use.

Rt = Ra | immed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 09h8 |

Rt = Ra | Rb

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 096 | ~ | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word1 |
| 4 | Byte Parallel1 |
| 5 | Char Parallel1 |
| 6 | Half Parallel1 |
| 7 | Word1 |

1 These codes are redundant with each other. They all have the same effect.

Compressed Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4h4 | Immed6 | 2h2 | 2h2 | Ra/Rt4 |

The immediate value is sign extended on the left before use.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4h4 | 22 | Rb4 | 22 | 32 | Ra/Rt4 |

Rt = Rt | Rb

|  |  |  |  |
| --- | --- | --- | --- |
| Ra/Rt4 | Ra/Rt6 | Ra/Rt4 | Ra/Rt6 |
| 0 | 1 | 8 | 14 |
| 1 | 2 | 9 | 18 |
| 2 | 3 | 10 | 19 |
| 3 | 4 | 11 | 20 |
| 4 | 5 | 12 | 23 |
| 5 | 11 | 13 | 60 |
| 6 | 12 | 14 | 61 |
| 7 | 13 | 15 | 62 |

Clock Cycles: 0.5

**Execution Units: All** ALUs

Exceptions: none

## RET – Return from Subroutine

Description:

This instruction performs a subroutine return by loading the program counter with the contents of the return address register. Additionally, the stack pointer is adjusted by a constant supplied in the instruction. The immediate constant should be a multiple of eight to keep the stack word aligned.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | 3Dh6 | 3Fh6 | 32h8 |

PC = RA

SP = SP + Immediate

Compressed Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 2h4 | Immed6 | 2h2 | 06 |

For this format, the immediate constant is shifted left three times and zero extended before use.

Clock Cycles: 1

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 (2 to 6). 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 255. The instruction will not allow setting the privilege level numerically less than the operating level.

The location of the target exception handler is found in the trap vector register for that operating level (tvec[xx]).

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:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 35 31 | 30 28 | 27 20 | 1917 | 16 14 | 13 8 | 7 0 |
| ~5 | IM3 | PL8 | ~3 | Tgt3 | Ra6 | 35h6 |

|  |  |
| --- | --- |
| Tgt3 |  |
| 0 | not used |
| 1 | redirect to hypervisor level |
| 2 | redirect to supervisor level |
| 3 | redirect to supervisor level |
| 4 | redirect to supervisor level |
| 5 | redirect to supervisor level |
| 6 | redirect to supervisor level |
| 7 | not used |

Clock Cycles: 3

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.

## SB – Store Byte

Description:

This instruction stores a byte (8 bit) value to memory.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rs6 | Ra6 | 60h8 |

Operation:

Memory8[Ra + immediate] = Rs

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 00h6 | A | R | Sc2 | Rs6 | Rb6 | Ra6 | 67h8 |

Operation:

Memory8[Ra + Rb \* Scale] = Rs

Acquire and release bits determine the ordering of memory operations.

A = acquire – 1 = no following memory operations can take place before this one

R = release – 1 = this memory operation cannot take place before prior ones.

|  |  |
| --- | --- |
| Sc2 | Scale Rb By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

Clock Cycles: 4 minimum depending on memory access time

Notes:

Stores always write through to memory and therefore take a significant number of clock cycles before they are ready to be committed. Exceptions are checked for during the execution of a store operation. The store is only committed to memory once it can be guaranteed that no prior instruction will exception.

## SC – Store Character

Description:

This instruction stores a character (16 bit) value to memory.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rs6 | Ra6 | 61h8 |

Operation:

Memory16[Ra + immediate] = Rs

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 01h6 | A | R | Sc2 | Rs6 | Rb6 | Ra6 | 67h8 |

Operation:

Memory16[Ra + Rb \* Scale] = Rs

Acquire and release bits determine the ordering of memory operations.

A = acquire – 1 = no following memory operations can take place before this one

R = release – 1 = this memory operation cannot take place before prior ones.

|  |  |
| --- | --- |
| Sc2 | Scale Rb By |
| 0 | 1 |
| 1 | 2 |
| 2 | 4 |
| 3 | 8 |

Clock Cycles: 4 minimum depending on memory access time

Notes:

Stores always write through to memory and therefore take a significant number of clock cycles before they are ready to be committed. Exceptions are checked for during the execution of a store operation. The store is only committed to memory once it can be guaranteed that no prior instruction will exception.

## SHL – Shift Left

Description:

Bits from the source register Ra are shifted left by the amount in register Rb or an immediate value. A zero is shifted into bit zero. The difference between this instruction and an ASL instruction is that ASL may cause an arithmetic overflow exception. SHL will never cause an exception. SHL also has a compressed instruction form which ASL does not.

Instruction Format:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10h6 | 0 | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 10h6 | 1 | Sz3 | Imm6 | Rb6 | Ra6 | 02h8 |

Compressed Instruction Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 3h4 | Immed6 | 2h2 | Ra/Rt6 |

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word1 |
| 4 | Byte Parallel1 |
| 5 | Char Parallel1 |
| 6 | Half Parallel1 |
| 7 | Word1 |

Clock Cycles: 1

**Execution Units:** ALU #0 Only

Exceptions: none

## XOR – Bitwise Exclusive Or

Description:

Perform a bitwise exclusive or operation between operands.

Instruction Format:

The immediate value is sign extended to the left before use.

Rt = Ra ^ immed

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Immed13 | ISz3 | Rt6 | Ra6 | 0Ah8 |

Rt = Ra ^ Rb

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0A6 | ~ | Sz3 | Rt6 | Rb6 | Ra6 | 02h8 |

|  |  |
| --- | --- |
| Sz3 |  |
| 0 | Byte |
| 1 | Char |
| 2 | Half |
| 3 | Word1 |
| 4 | Byte Parallel1 |
| 5 | Char Parallel1 |
| 6 | Half Parallel1 |
| 7 | Word1 |

1 These codes are redundant with each other. They all have the same effect.

Compressed Instruction Format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4h4 | 32 | Rb4 | 22 | 32 | Ra/Rt4 |

Rt = Rt ^ Rb

|  |  |  |  |
| --- | --- | --- | --- |
| Ra/Rt4 | Ra/Rt6 | Ra/Rt4 | Ra/Rt6 |
| 0 | 1 | 8 | 14 |
| 1 | 2 | 9 | 18 |
| 2 | 3 | 10 | 19 |
| 3 | 4 | 11 | 20 |
| 4 | 5 | 12 | 23 |
| 5 | 11 | 13 | 60 |
| 6 | 12 | 14 | 61 |
| 7 | 13 | 15 | 62 |

Clock Cycles: 0.5

**Execution Units: All** ALUs

Exceptions: none

# Floating Point

|  |  |  |
| --- | --- | --- |
| Prec3 |  |  |
| 0 | 16 | Half |
| 1 | 32 | Single |
| 2 | 64 | Double |
| 3 | 96 | Triple |
| 4 | 128 | Quad |
| 5 |  | reserved |
| 6 |  | reserved |
| 7 |  | reserved |

|  |  |
| --- | --- |
| P |  |
| 0 | Normal operation |
| 1 | SIMD operation, perform same operation on all lanes |

**Representation**

The floating-point format is an IEEE-754 representation for double precision. Briefly,

**Double Precision Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 63 | 62 | 61 52 | 51 0 |
| SM | SE | Exponent | Mantissa |

SM – sign of mantissa

SE – sign of exponent

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

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

The exponent ranges from -1024 to +1023 for double precision numbers

Triple precision. Briefly,

**Triple Precision Format:**

|  |  |  |  |
| --- | --- | --- | --- |
| 95 | 94 | 93 72 | 71 0 |
| SM | SE | Exponent22 | Mantissa72 |

## FADD – Floating point addition

**Description:**

Add two floating point numbers in registers Ra and Rb and place the result into target register Rt.

**Instruction Format:**

**Register – Register Add**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 | 32 | 31 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 046 | ~ | P | Prec3 | Rm3 | Rt6 | Rb6 | Ra6 | 0Fh8 |

P: SIMD indicator, 1= SIMD

Register – Immediate Add

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 | 32 | 31 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 046 | ~ | P | Prec3 | Rm3 | Rt6 | 636 | Ra6 | 0Fh8 |

Additional bits as needed for precision.

Add from constant ROM

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 | 32 | 31 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 046 | ~ | P | Prec3 | Rm3 | Rt6 | Index6 | Ra6 | 1Fh8 |

**Clock Cycles: 10**

**Execution Units:** Floating Point

## FBEQ –Branch if Equal

Description:

If the values of two registers are equal a seventeen-bit sign extended value is added to the program counter. The branch is relative to the address of the instruction directly following the branch. Note that positive and negative zero are treated as equal.

Instruction Format:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 14 | 13 8 | 7 0 |
| Displacement17 | 03 | Rb6 | Ra6 | 3Eh5 |

Operation:

if (Ra = Rb)

pc = pc + displacement

Instruction Format:

A branch to a value computed in a register may be performed using the instruction format shown below. Rc contains the target address which is an absolute address.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~ | 04 | Rc6 | Rb6 | Ra6 | 3Fh6 |

Operation:

if (Ra = Rb)

pc = Rc

## FMA – Floating point multiply and add

**Description:**

Multiply two floating point numbers in registers Ra and Rb then add result to register Rc and place the result into target register Rt.

**Instruction Format:**

**Register – Register Add**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 38 | 37 35 | 34 32 | 31 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| ~ | P | Prec3 | Rm3 | Rt6 | Rc6 | Rb6 | Ra6 | 0Dh8 |

P: SIMD indicator, 1= SIMD

**Clock Cycles:**

**Execution Units:** Floating Point

## FSQRT – Floating point square root

**Description:**

Take the square root of the floating-point number in register Ra and place the result into target register Rt. The sign bit (bit 63) of the register is set to zero. If the value in Ra is negative, FSQRT returns a NaN indicating an attempt to take the square root of a negative number.

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 34 | 33 | 32 | 31 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 1Dh6 | ~ | P | Prec3 | Rm3 | Rt6 | 06 | Ra6 | 0Fh8 |

**Clock Cycles: 110**

**Execution Units:** Floating Point

|  |  |  |
| --- | --- | --- |
| Prec3 |  |  |
| 0 | 16 | Half |
| 1 | 32 | Single |
| 2 | 64 | Double |
| 3 | 96 | Triple |
| 4 | 128 | Quad |
| 5 | 16 | Integer |
| 6 | 32 | Integer |
| 7 | 64 | Integer |

## VADD - Add

Synopsis

Vector register add. Vt = Va + Vb

**Description**

Two vector registers (Va and Vb) are added together and placed in the target vector register Vt.

**Instruction Format**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 43 38 | 3736 | 35 33 | 32 | 31 29 | 28 26 | 25 20 | 19 14 | 13 8 | 7 0 |
| 046 | ~2 | Vm3 | P | Prec3 | Rm3 | Vt6 | Vb6 | Va6 | 01h8 |

**Operation**

for x = 0 to VL - 1

if (Vm[x]) Vt[x] = Va[x] + Vb[x]

## Compressed Instruction Formats

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  | |  |  | |  |
| 0000 | | 006 | | 10 | 06 | | NOP |
| 0000 | | Amt[8..3] | | 10 | 636 | | ADDISP |
| 0000 | | Amt6 | | 10 | Ra/Rt6 | | ADDI |
| 0001 | | Amt6 | | 10 | Rt6 | | LDI / SYS (SYS if Rt = 0) |
| 0010 | | Amt6 | | 10 | Ra/Rt6 | | RET / ANDI (RET if Ra=0) |
| 0011 | | Amt6 | | 10 | Rt6 | | SHLI |
| 0100 | | Amt6 | | 10 | 00 | Ra/Rt’4 | SHRI |
| 0100 | | Amt6 | | 10 | 01 | Ra/Rt’4 | ASRI |
| 0100 | | Amt6 | | 10 | 10 | Ra/Rt’4 | ORI |
| 0100 | | 00 | Rb’4 | 10 | 11 | Ra/Rt’4 | SUB |
| 0100 | | 01 | Rb’4 | 10 | 11 | Ra/Rt’4 | AND |
| 0100 | | 10 | Rb’4 | 10 | 11 | Ra/Rt’4 | OR |
| 0100 | | 11 | Rb’4 | 10 | 11 | Ra/Rt’4 | XOR |
| 0101 | | Address11..6 | | 10 | Address5..0 | | CALL |
| 0110 | |  | | 10 |  | | reserved |
| 0111 | | Disp11..6 | | 10 | Disp5..0 | | BRA |
| 10 | Disp8 | | | 10 | Ra6 | | BEQZ |
| 11 | Disp8 | | | 10 | Ra6 | | BNEZ |
| 0000 | | Rt6 | | 11 | Ra6 | | MOV |
| 0001 | | Rb6 | | 11 | Ra/Rt6 | | ADD |
| 0010 | | Rt6 | | 11 | Ra6 | | JALR |
| 0011 | | ?????? | | 11 | Ra6 | | reserved |
| The following two instructions have SP as an implied register read | | | | | | | |
| 0100 | | Disp7..2 | | 11 | Rt6 | | LH Rt,d[SP} |
| 0101 | | Disp8..3 | | 11 | Rt6 | | LW Rt,d[SP] |
| The following two instructions have FP as an implied register read | | | | | | | |
| 0110 | | Disp7..2 | | 11 | Rt6 | | LH Rt,d[FP] |
| 0111 | | Disp8..3 | | 11 | Rt6 | | LW Rt,d[FP] |
| The following two instructions have SP as an implied register read | | | | | | | |
| 1000 | | Disp7..2 | | 11 | Rb6 | | SH Rb,d[SP] |
| 1001 | | Disp8..3 | | 11 | Rb6 | | SW Rb,d[SP] |
| The following two instructions have FP as an implied register read | | | | | | | |
| 1010 | | Disp7..2 | | 11 | Rb6 | | SH Rb,d[FP] |
| 1011 | | Disp8..3 | | 11 | Rb6 | | SW Rb,d[FP] |
| 1100 | | d5..4 | Rt’4 | 11 | d3..2 | Ra’4 | LH Rt,d[Ra] |
| 1101 | | d6..5 | Rt’4 | 11 | d4..3 | Ra’4 | LW Rt,d[Ra] |
| 111? | | d5..4 | Rb’4 | 11 | d3..2 | Ra’4 | SH Rb,d[Ra] |
| 1111 | | d6..5 | Rb’4 | 11 | d4..3 | Ra’4 | SW Rb,d[Ra] |

Several instructions may not be compressed due to format or other hardware limitations. In particular the vector instructions and branch on bit set/clear, and branch equal to immediate instructions may not be compressed.

Compressed instructions must follow the formats in this table. ‘?’ indicates where the bit(s) use may be defined by the application. Bits 8 to 11 (the register spec field) may also be defined by the application. It may be desirable in some cases to use these bits for immediate constants rather than a register spec. Bits 4 to 6 and 8 to 17 are used as a 13 bit index into a lookup table for the decompressed instruction.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Fmt | # Insn | 17 14 | | | 13 8 | | 7 6 | 5 0 | |  | Sample Insn |
| CA | 3 | 00?? | | | ?? | ?/Rb’4 | 10 | Ra/Rt6 | |  | nop, addi, shli |
| CB | 2 | 0001 | | | ?? | ?/Rb’4 | 10 | Rt6 | |  | ldi |
| CC | 3 | 0100 | | | ?? | ?/Rb’4 | 10 | ?? | Ra/Rt’4 |  | shri, asri andi |
| CD | 4 | 0100 | | | ?? | ?/Rb’4 | 10 | 11 | Ra/Rt’4 |  | sub, and, or, xor |
| CE |  | 0101 | | |  |  | 10 |  |  | reserved |  |
| CF |  | 0110 | | |  |  | 10 |  |  | reserved |  |
| CG | 1 | 0111 | | | ?? | ?/Rb’4 | 10 | ?? | Ra’4 | ? = Displacement bit | bra |
| CH | 2 | 1 | ????? | | | ?/Rb’4 | 10 | Ra6 | | ? = Displacement bit | beqz bnez |
| CI | 4 | 00?? | | | Rt6 | | 11 | Ra6 | | 1 Full Read Port, 1 Full Write Port | mov add jalr |
| CJ | 2+ | 010 | | ??? | | ?/Rb’4 | 11 | Rt6 | | SP implied | lw Rt, d[SP] |
| CK | 2+ | 011 | | ??? | | ?/Rb’4 | 11 | Rt6 | | FP implied | lw Rt, d[FP] |
| CL | 2+ | 100 | | ??? | | ???? | 11 | Rb6 | | SP implied, 2nd Read port | sw Rb, d[SP] |
| CM | 2+ | 101 | | ??? | | ???? | 11 | Rb6 | | FP implied, 2nd read port | sw Rb, d[FP] |
| CN | 2+ | 110 | | ??? | | ?/Rt’4 | 11 | ?? | Ra’4 | 1 read port | lw Rt, d[Ra] |
| CO | 2++ | 111 | | ??? | | ?/Rb’4 | 11 | ?? | Ra’4 | 2 read ports | sw Rb, d[Ra] |

It is assumed that registers r32 to r63 will be used primarily for floating point and registers r0 to r31 for integer values.

To conserve space not all the bits of the decompressed instruction are stored. Only the following bits are needed in the decompression table: opcode bits 0 to 6 and instruction bits 14 to 35. Bit 7 is always a 1 for decompressed instructions so doesn’t need to be stored in the table. Bits 8 to 13 represent the register file address bits which are not required in the table as they are inserted from the decompressed instruction.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 35 20 | 19 14 | 13 8 | 7 | 6 0 |
| Other16 | Rt/Rb6 | Ra6 | 1 | Opcode7 |

## Major Opcode (inst. bits 0 to 6)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | BRK | {VECTOR} | {R2} |  | ADDI | {Bitfield} | CMPI | CMPUI | ANDI | ORI | XORI | QOPI | EXEC | FMA | CSR | {FLOAT} |
| 1x |  |  |  |  |  |  |  |  | MULI | MULUI | MULSUI | FNMS | DIVI | DIVUI | DIVSUI | {FLOATC} |
| 2x | SEQI | SNEI | SLTI | SGEI | SLEI | SGTI | SLTUI | SGEUI | SLEUI | SGTUI | CACHE | FNMA | MODI | MODUI | MODSUI | FMS |
| 3x | JMP | CALL | RET | JAL | SYS | REX |  |  | Bcc | BccR |  |  | BBc | BEQ# | FBcc | FBccR |
| 4x | LB | LBU | LBO | LC | LCU | LCO | LH | LHU | LHO | LW | LWU | LWO | LQ | {Indexed Load} | LV |  |
| 5x | LVB | LVBU | LVBO | LVC | LVCU | LVCO | LVH | LVHU | LVHO | LVW | LVWU | LVWO | LVQ | LVWR | LVV |  |
| 6x | SB | SC | SH | SW | SQ | SWC | SV | {Indexed} |  |  |  |  |  |  |  |  |
| 7x | ASWAP | AADD | AAND | AOR | AXOR | AMIN | AMAX | AMINU | AMAXU | ASHL | ASHR | INC |  |  |  |  |

## {R2} Major Func (inst. bits 30 to 35)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | {BCD} | {R1} |  |  | ADD | SUB | CMP | CMPU | AND | OR | XOR |  | NAND | NOR | XNOR |  |
| 1x | SHL | ASL | SHR | ASR | ROL | ROR |  |  | MUL | MULU | MULSU | MOV | DIV | DIVU | DIVSU |  |
| 2x | SEQ | SNE | SLT | SGE | SLE | SGT | SLTU | SGEU | SLEU | SGTU | CACHEX |  | MOD | MODU | MODSU |  |
| 3x |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

## {Indexed Load} Major Func (inst. bits 31 to 35)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | LBX | LBUX | LBOX | LCX | LCUX | LCOX | LHX | LHUX | LHOX | LWX | LWUX | LWOX | LQX |  | LVX |  |
| 1x | LVBX | LVBUX | LVBOX | LVCX | LVCUX | LVCOX | LVHX | LVHUX | LVHOX | LVWX | LVWUX | LVWOX | LVQX | LVWRX | LVVX |  |

## {Indexed} Major Func (inst. bits 31 to 35)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | s8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | SBX | SCX | SHX | SWX | SQX | SWCX | SVX |  |  |  |  |  |  |  |  |  |
| 1x | ASWAPX | AADDX | AANDX | AORX | AXORX | AMINX | AMAXX | AMINUX | AMAXUX | ASHLX | ASHRX | INCX |  |  |  |  |