rfPhoenix

(c) 2022 Robert Finch

## Register File

There are 64 general purpose registers. R0 is special in that it may be written once with the value zero. After a zero is written to r0 it can no longer be updated.

Registers designated as vector mask registers are also special in that one of them may be specified as a mask register in the instruction.

Link registers are special in that they may be specified by call, jump, or unconditional branch instructions as the target register to store the next instruction pointer value in.

Alternate stack pointers are reference via register code 31 for different operating modes. For example in machine mode a stack pointer reference will translate to register 46.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Reg | Mnen. |  |  | Reg |  |  |
| 0 | Zero |  |  | 32 | VM0 | Vector Mask |
| 1 | A0 | Return Value |  | 33 | VM1 |  |
| 2 | A1 |  |  | 34 | VM2 |  |
| 3 | T0 | Temporaries |  | 35 | VM3 |  |
| 4 | T1 |  |  | 36 | VM4 |  |
| 5 | T2 |  |  | 37 | VM5 |  |
| 6 | T3 |  |  | 38 | VM6 |  |
| 7 | T4 |  |  | 39 | VM7 |  |
| 8 | T5 |  |  | 40 | LC | Loop Counter |
| 9 | T6 |  |  | 41 | LR1 | Link Registers |
| 10 | T7 |  |  | 42 | LR2 |  |
| 11 | S0 | Register Vars |  | 43 | R43 |  |
| 12 | S1 |  |  | 44 | SSP | Alternate stack pointers |
| 13 | S2 |  |  | 45 | HSP |  |
| 14 | S3 |  |  | 46 | MSP |  |
| 15 | S4 |  |  | 47 | ISP |  |
| 16 | S5 |  |  | 48 | EIP0 | Exception IPs |
| 17 | S6 |  |  | 49 | EIP1 |  |
| 18 | S7 |  |  | 50 | EIP2 |  |
| 19 | S8 |  |  | 51 | EIP3 |  |
| 20 | S9 |  |  | 52 | EIP4 |  |
| 21 | A2 | Arguments |  | 53 | EIP5 |  |
| 22 | A3 |  |  | 54 | EIP6 |  |
| 23 | A4 |  |  | 55 | EIP7 |  |
| 24 | A5 |  |  | 56 | R56 |  |
| 25 | A6 |  |  | 57 | R57 |  |
| 26 | A7 |  |  | 58 | R58 |  |
| 27 | GP3 |  |  | 59 | R59 |  |
| 28 | GP2 |  |  | 60 | R60 |  |
| 29 | GP1 |  |  | 61 | R61 |  |
| 30 | FP | Frame Pointer |  | 62 | R62 |  |
| 31 | SP | Stack Pointer |  | 63 | R63 |  |

## Vector Register File

The vector register file contains 64 general purpose vector registers. The vector registers are unified and may contain either integer or floating-point data.

## Instruction Formats

There are relatively few instruction formats. Instructions are 40-bits in size with two exceptions. The NOP instruction is a single byte to allow code to be aligned on any byte boundary. The constant postfix instruction is 24-bits in size as that is all that is needed.

Register Format, Indexed Load / Store Format

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | ~2 | ~ | Func6 | Tb | Rb6 | Mask3 | Ta | Ra6 | Tt | Rt6 | Opcode6 |

Float Register Format

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Rm2 | Tc | Rc6 | Tb | Rb6 | Mask3 | Ta | Ra6 | Tt | Rt6 | Opcode6 |

Immediate Format, Load / Store Format

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 38 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Immediate16 | Mask3 | Ta | Ra6 | Tt | Rt6 | Opcode6 |

Branch

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| Displacement17 | Cnd3 | 0 | Ra6 | 0 | Rb6 | Opcode6 |

Call, Jump

|  |  |  |
| --- | --- | --- |
| 39 8 | 7 6 | 5 0 |
| Target32 | Rt2 | Opcode6 |

Postfix

|  |  |  |
| --- | --- | --- |
| 23 8 | 7 6 | 5 0 |
| Immediate16 | ~2 | Opcode6 |

Major Opcode

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0x | 0  BRK | 1  PFX | 2  R2 | 3 | 4  ADDI | 5  SUBFI | 6  MULI | 7 |
| 1x | 8  ANDI | 9  ORI | 10  XORI | 11  NOP | 12 | 13 | 14  CMP EQI | 15  CMP NEI |
| 2x | 16  CMP LTI | 17  CMP GEI | 18  CMP LEI | 19  CMP GTI | 20  CMP LTUI | 21  CMP GEUI | 22  CMP LEUI | 23  CMP GTUI |
| 3x | 24  JSR abs | 25  BSR rel | 26  JMP abs | 27  BRA rel | 28  Bcc | 29  FBcc | 30  FCMP EQI | 31  FCMP NEI |
| 4x | 32 | 33 | 34 | 35 | 36  FCMP LTI | 37  FCMP GEI | 38  FCMP LEI | 39  FCMP GTI |
| 5x | 40 | 41 | 42 | 43 | 44  FMA | 45  FMS | 46  FNMA | 47  FNMS |
| 6x | 48  LDB | 49  LDBU | 50  LDW | 51  LDWU | 52  LDT | 53 | 54 | 55 |
| 7x | 56  STB | 57  STW | 58  STT | 59 | 60 | 61 | 62 | 63  CACHE |

Major Func

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0x | 0 | 1 | 2  VSHUF | 3  VEX | 4  ADD | 5  SUB | 6  MUL | 7 |
| 1x | 8  AND | 9  OR | 10  XOR | 11 | 12 | 13  VEINS | 14  CMP EQ | 15  CMP NE |
| 2x | 16  CMP LT | 17  CMP GE | 18  CMP LE | 19  CMP GT | 20  CMP LTU | 21  CMP GEU | 22  CMP LEU | 23  CMP GTU |
| 3x | 24  SLLI | 25  SRLI | 26  SRAI | 27  SLL | 28  SRL | 29  SRA | 30  FCMP EQ | 31  FCMP NE |
| 4x | 32  VSLLVI | 33  VSRLVI | 34  VSLLV | 35  VSRLV | 36  FCMP LT | 37  FCMP GE | 38  FCMP LE | 39  FCMP GT |
| 5x | 40  ITOF | 41  FTOI | 42 | 43 | 44 | 45 | 46 | 47 |
| 6x | 48  LDBX | 49  LDBUX | 50  LDWX | 51  LDWUX | 52  LDTX | 53 | 54 | 55 |
| 7x | 56  STBX | 57  STWX | 58  STTX | 59 | 60 | 61 | 62 | 63  CACHEX |

### Extended Constants

The upper 16 bits of a 32-bit constant may be specified using a postfix instruction which overrides sign extension of the constant.

### Vector Masking

If masking is enabled via the ‘m’ bit in the instruction then the target lanes corresponding to one bits in the mask register are updated. Other lanes of the target register are not affected.

### ADD - Register-Register

**Description:**

Add two registers and place the sum in the target register. All register values are treated as integers.

**Instruction Format:** R2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | ~2 | ~ | 04h6 | Tb | Rb6 | Mask3 | Ta | Ra6 | Tt | Rt6 | 02h6 |

**Operation:**

Rt = Ra + Rb

**Clock Cycles:** 1

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### ADDI - Register-Immediate

**Description:**

Add a register and an immediate value and place the sum in the target register. All values are treated as signed integers.

**Instruction Format:** R2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 38 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Immediate16 | Mask3 | Ta | Ra6 | Tt | Rt6 | 046 |

**Operation:**

Rt = Ra + Imm

**Clock Cycles:** 1

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### AND - Register-Register

**Description:**

Bitwise ‘and’ two registers and place the result in the target register. All register values are treated as integers.

**Instruction Format:** R2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | ~2 | ~ | 08h6 | Tb | Rb6 | Mask3 | Ta | Ra6 | Tt | Rt6 | 02h6 |

**Operation:**

Rt = Ra & Rb

**Clock Cycles:** 1

**Execution Units:** All Integer ALU’s

**Exceptions:** none

### ANDI - Register-Immediate

**Description:**

Bitwise ‘And’ a register and an immediate value and place the sum in the target register. All values are treated as signed integers.

**Instruction Format:** R2

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 38 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Immediate16 | Mask3 | Ta | Ra6 | Tt | Rt6 | 086 |

**Operation:**

Rt = Ra & Imm

**Clock Cycles:** 1

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### NOP – No Operation

**Description:**

NOP does not perform any operation. Unlike most instructions, NOP occupies only a single byte. This is to allow NOPs to be used to align code to byte addresses.

**Instruction Format:** NOP

|  |  |
| --- | --- |
| 7 6 | 5 0 |
| ~2 | 0Bh6 |

**Operation:**

< none >

**Clock Cycles:** 1

**Execution Units:** All Integer ALU’s

**Exceptions:** none

**Notes:**

### VEX / VMOVS – Vector Element Extract

**Synopsis**

Vector element extract.

**Description**

A vector register element from Va is transferred into a general-purpose register Rt. The element to extract is identified by Rb. Rb and Rt are scalar registers. This instruction may also be used to broadcast a vector element to all elements of a target vector register.

**Instruction Format:** R2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | ~2 | ~ | 036 | 0 | Rb6 | Mask3 | 1 | Ra6 | Tt | Rt6 | 026 |

**Operation**

Rt = Va[Rb]

**Clock Cycles:** 1

**Exceptions**: none

### VSHUF – Vector Shuffle

**Synopsis**

Shuffle vector elements.

**Description**

Vector register elements from Va are transferred into a vector register Rt. The elements to transfer are guided by vector register Rb.

**Instruction Format:** R2

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | ~2 | ~ | 026 | 1 | Rb6 | Mask3 | 1 | Ra6 | 1 | Rt6 | 026 |

**Operation**

Rt = Va[Rb]

**Clock Cycles:** 1

**Exceptions**: none

## Floating-Point Operations

### FMA – Floating Point Multiply Add

**Description:**

Multiply two floating point numbers in registers Ra and Rb add a third number from register Rc and place the result into target register Rt. The multiplication and addition are fused with no intermediate rounding.

**Instruction Format: F3**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Rm2 | Tc | Rc6 | Tb | Rb6 | Mask3 | Ta | Ra6 | Tt | Rt6 | 446 |

**Operation:**

Rt = Ra \* Rb + Rc

**Clock Cycles: 7**

**Execution Units:** All Floating Point

### FMS – Floating Point Multiply Subtract

**Description:**

Multiply two floating point numbers in registers Ra and Rb subtract a third number from register Rc and place the result into target register Rt. The multiplication and addition are fused with no intermediate rounding.

**Instruction Format: F3**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Rm2 | Tc | Rc6 | Tb | Rb6 | Mask3 | Ta | Ra6 | Tt | Rt6 | 456 |

**Operation:**

Rt = Ra \* Rb - Rc

**Clock Cycles: 7**

**Execution Units:** All Floating Point

## Memory Operations

### CACHE – Cache Command

**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. The address of the cache line to be invalidated is passed in Ra if needed.

**Instruction Format:**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 38 23 | 22 20 | 19 | 18 13 | 12 | 11 9 | 8 6 | 5 0 |
| m | Displacement16 | Mask3 | Ta | Ra6 | 0 | DC3 | IC3 | 636 |

**Commands:**

|  |  |  |
| --- | --- | --- |
| IC3 | Mne. | Operation |
| 0 | NOP | no operation |
| 1 | invline | invalidate line associated with given address |
| 2 | invall | invalidate the entire cache (address is ignored) |
| 3 to 7 |  | reserved |

|  |  |  |
| --- | --- | --- |
| 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) |
| 5 to 7 |  | reserved |

**Clock Cycles:** 20

**Execution Units:** All ALU’s / Memory

**Operation:**

**Exceptions:** DBG

### CACHEX – Cache Command

**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 Format:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 3837 | 36 | 25 30 | 29 | 28 23 | 22 20 | 19 | 18 13 | 12 | 11 9 | 8 6 | 5 0 |
| m | ~2 | ~ | 636 | Tb | Rb6 | Mask3 | Ta | Ra6 | Tt | DC3 | IC3 | 26 |

**Commands:**

|  |  |  |
| --- | --- | --- |
| IC3 | Mne. | Operation |
| 0 | NOP | no operation |
| 1 | invline | invalidate line associated with given address |
| 2 | invall | invalidate the entire cache (address is ignored) |
| 3 to 7 |  | reserved |

|  |  |  |
| --- | --- | --- |
| 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) |
| 5 to 7 |  | reserved |

**Clock Cycles:** 20

**Execution Units:** All ALU’s / Memory

**Operation:**

**Exceptions:** DBG

### LDB – Load Byte

**Description:**

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

This instruction may load data from the cache and cause a cache load operation if the data isn’t in the cache provided the current memory page is cacheable.

**Instruction Format: RI**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 38 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Displacement16 | Mask3 | Ta | Ra6 | Tt | Rt6 | 486 |

**Clock Cycles:** 20 (one memory access)

**Execution Units:** All ALU’s / Memory

**Operation:**

Rt = sign extend (memory8[Ra+displacement])

**Exceptions:** DBE, TLB, RDV

### LDT – Load Tetra

**Description:**

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

This instruction may load data from the cache and cause a cache load operation if the data isn’t in the cache provided the current memory page is cacheable.

Loading a vector register loads all 512-bits. A vector mask register may be used to determine which elements of the vector are updated.

**Instruction Format:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 39 | 38 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| m | Displacement16 | Mask3 | Ta | Ra6 | Tt | Rt6 | 526 |

**Clock Cycles:** 20 (one memory access)

**Execution Units:** All ALU’s / Memory

**Operation:**

Rt = sign extend (memory32[Ra+displacement])

**Exceptions:** DBE, TLB, RDV

## Branches

### Conditions

Conditional branches branch to the target address only if the condition is true. The condition is determined by the comparison of two general-purpose registers. Two sets of conditional branches are supported. One set for integers and a second set for floating-point values.

### Conditional Branch Format

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| Displacement17 | Cnd3 | 0 | Ra6 | 0 | Rb6 | Opcode6 |

### Branch Conditions

The branch opcode determines the condition under which the branch will execute.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ▼ |  |  |  |  | ▼ |
| 39 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| Displacement17 | Cnd3 | 0 | Ra6 | 0 | Rb6 | Opcode6 |

|  |  |  |
| --- | --- | --- |
| Cnd3 | Integer Comparison Test | Float |
| 0 | signed less than | less than |
| 1 | signed greater or equal | greater than or equal |
| 2 | unsigned less than | less than or equal |
| 3 | unsigned greater than or equal | greater than |
| 4 | reserved | reserved |
| 5 | reserved | reserved |
| 6 | equal | equal |
| 7 | not equal | not equal |

### BEQ – Branch if Equal

**Description**:

This instruction branches to the target address if the contents of the Ra equals the contents of Rb, otherwise program execution continues with the next instruction. The values are treated as signed integers. For a further description see [Branch Instructions](#_Branch_Instructions).

**Formats Supported**: Bcc

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| Displacement17 | 03 | 0 | Ra6 | 0 | Rb6 | 286 |

**Operation:**

If (Ra==Rb)

IP = IP + Constant

**Execution Units**: Branch

**Exceptions**: none

**Notes:**

### BNE – Branch if Not Equal

**Description**:

This instruction branches to the target address if the contents of the Ra does not equal the contents of Rb, otherwise program execution continues with the next instruction. The values are treated as signed integers. For a further description see [Branch Instructions](#_Branch_Instructions).

**Formats Supported**: Bcc

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| Displacement17 | 13 | 0 | Ra6 | 0 | Rb6 | 286 |

**Operation:**

If (Ra<>Rb)

IP = IP + Constant

**Execution Units**: Branch

**Exceptions**: none

**Notes:**

### FBEQ – Float Branch if Equal

**Description**:

This instruction branches to the target address if the contents of the Ra equals the contents of Rb, otherwise program execution continues with the next instruction. The values are treated as single precision floating-point numbers. For a further description see [Branch Instructions](#_Branch_Instructions).

**Formats Supported**: Bcc

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 39 23 | 22 20 | 19 | 18 13 | 12 | 11 6 | 5 0 |
| Displacement17 | 03 | 0 | Ra6 | 0 | Rb6 | 296 |

**Operation:**

If (Ra==Rb)

IP = IP + Constant

**Execution Units**: Branch

**Exceptions**: none

**Notes:**

For a floating-point comparison positive and negative zero are considered equal.

### JMP – Jump to Address

**Description**:

This instruction always jumps to the target address. The target address range is 4GB.

**Formats Supported**: JMP

|  |  |  |
| --- | --- | --- |
| 39 8 | 7 6 | 5 0 |
| Target32 | 02 | 246 |

**Operation:**

IP = Constant

**Execution Units**: Branch

**Exceptions**: none

**Notes:**

### JSR – Jump to Subroutine

**Description**:

This instruction always jumps to the target address. The address of the next instruction is stored in a link register. The target address range is 4GB.

**Formats Supported**: JMP

|  |  |  |
| --- | --- | --- |
| 39 8 | 7 6 | 5 0 |
| Target32 | Rt2 | 246 |

**Operation:**

Lk = next IP

IP = Constant

**Execution Units**: Branch

**Exceptions**: none

**Notes:**