rtfItanium

©2019 Robert Finch

“If you build it, he will come” – Field of Dreams

## Features:

* 80-bit data path, double-extended floating-point
* 64 entry general purpose register file with unified floating-point and integer registers
* 3-way out-of-order (ooo) superscalar execution
* 40-bit instructions, three per 128-bit bundle
* Instruction L1, L2 and data L1, L2 caches
* Dual memory channels

### 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. r61 is the link register used implicitly by the call instruction.

|  |  |  |
| --- | --- | --- |
| Register | Description / Suggested Usage | Saver |
| r0 | always reads as zero |  |
| r1-r4 | return values / exception | caller |
| r5-r22 | temporaries | caller |
| r23-r37 | register variables | callee |
| r38-r47 | function arguments | caller |
| r48 | reserved for system |  |
| r49 | reserved for system |  |
| r50 | reserved for system |  |
| r51 | reserved for system |  |
| r52 | garbage collector |  |
| r53 | garbage collector |  |
| r54 | assembler usage |  |
| r55 | type number / function argument | caller |
| r56 | class pointer / function argument | caller |
| r57 | thread pointer | callee |
| r58 | global pointer |  |
| r59 | exception SP offset |  |
| r60 | exception link register | callee |
| r61 | return address / link register | callee |
| r62 | base / frame pointer | callee |
| r63 | stack pointer (hardware) | callee |

Bundle Format:

|  |  |  |  |
| --- | --- | --- | --- |
| 127 120 | 119 80 | 79 40 | 39 0 |
| Template8 | Slot2 | Slot1 | Slot0 |

# Instruction Formats:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Immediate7 | | | | | A | R | Immed15 | | | | | | Rs16 | Opcode4 | Rd6 | | | | ML |
| Immediate7 | | | | | A | R | Immed9 | | | Rs26 | | | Rs16 | Opcode4 | Immed6 | | | | MS |
| Immediate7 | | | | | A | R | Immed9 | | | DC3 | | IC3 | Rs16 | Eh4 | Immed6 | | | | CACHE |
| ~7 | | | | | A | R | ~3 | | ~6 | Rs26 | | | 636 | Dh4 | 636 | | | | PUSH |
| Immediate7 | | | | | A | R | Immed15 | | | | | | 636 | Bh4 | 636 | | | | PUSHC |
| Funct5 | | | | Immed2 | A | R | Sc3 | | Rs36 | Immed6 | | | Rs16 | Opcode4 | Rd6 | | | | MLX |
| Funct5 | | | | Immed2 | A | R | Sc3 | | Rs36 | Rs26 | | | Rs16 | Opcode4 | Immed6 | | | | MSX |
| 145 | | | | Immed2 | A | R | Sc3 | | Rs36 | DC3 | | IC3 | Rs16 | Fh4 | Immed6 | | | | CACHEX |
| 255 | | | | ~2 | ~ | ~ | ~3 | | ~6 | ~6 | | | ~6 | Opcode4 | ~6 | | | | MEMDB |
| 245 | | | | ~2 | ~ | ~ | ~3 | | ~6 | ~6 | | | ~6 | Opcode4 | ~6 | | | | MEMSB |
| Immediate7 | | | | | Op2 | | Immed15 | | | | | | Rs16 | Opcode4 | Rd6 | | | | RI |
| Funct5 | | | | 02 | 02 | | ~3 | | ~6 | ~6 | | | Rs16 | 14 | Rd6 | | | | R1 |
| Funct5 | | | | Funct22 | 02 | | Sz3 | | Rs36 | Rs26 | | | Rs16 | Opcode4 | Rd6 | | | | R3 |
| Funct5 | | | | Rg2 | o | w | ~3 | | Rs36 | Bw6 | | | Bo6 | Bh4 | Rd6 | | | | BF |
| Funct5 | | | | Rg2 | o | Bw4 | | | Rs36 | Rs26 | | | Bo6 | Bh4 | Rd6 | | | | BFI |
| Td22..13 | | | | | | To12..5 | | | | Rs26 | | | Rs16 | Opcode4 | To4..2 | | Cond3 | | [Bcc](#_Bcc_–_Conditional) |
| Td22..13 | | | | | | To12..5 | | | | Bitno6..1 | | | Rs16 | 5h4 | To4..2 | | B0 | Cn2 | BBS |
| Td22..13 | | | | | | To12..5 | | | | Immed6 | | | Rs16 | Opcode4 | To4..2 | | Imm3 | | BEQI |
| ~9 | | | | | | | ~3 | | Rs36 | Rs26 | | | Rs16 | 2h4 | ~2 | Cond4 | | | BRG |
| ~9 | | | | | | | ~3 | | ~6 | ~6 | | | ~6 | 3h4 | ~6 | | | | NOP |
| Immediate7 | | | | | ~2 | | Immed15 | | | | | | Rs16 | 8h6 | 616 | | | | JAL |
| Address30 | | | | | | | | | | | | | | Opcode4 | Address6 | | | | CALL |
| Immediate18 | | | | | | | | | | 616 | | | 636 | Bh4 | 636 | | | | RET |
| H | 04 | | | ~5 | | | | Cause8 | | ~2 | Imask4 | | Rs16 | Fh4 | ~6 | | | | BRK |
| 0 | 14 | | | ~5 | | | | 2558 | | ~2 | 04 | | 06 | Fh4 | ~6 | | | | PFI |
| 05 | | | | ~7 | | | | | ~6 | ~6 | | | Rs16 | Eh4 | Sema6 | | | | RTI |
| 15 | | | | ~5 | | | | PrivLvl8 | | T2 | Imask4 | | Rs16 | Eh4 | ~6 | | | | REX |
| 25 | | | | ~7 | | | | | ~6 | ~6 | | | ~6 | Eh4 | ~6 | | | | SYNC |
| 35 | | | | ~7 | | | | | ~6 | ~2 | Imask4 | | Rs16 | Eh4 | Rd6 | | | | SEI |
| 45 | | | | ~7 | | | | | Immed6 | Rs26 | | | Rs16 | Eh4 | ~6 | | | | WAIT |
| Immediate7 | | | | | ~ | ~ | Immed9 | | | Rs26 | | | Rs16 | Ch4 | Immed6 | | | | CHKI |
| ~12 | | | | | | | | | Rs36 | Rs26 | | | Rs16 | Dh4 | ~6 | | | | CHK |
| Funct5 | | | | Prec4 | | | Rm3 | | Rs36 | Rs26 | | | Rs16 | Opcode4 | Rd6 | | | | FLT3 |
| ~5 | | | | Prec4 | | | Rm3 | | Op6 | Rs26 | | | Rs16 | 14 | Rd6 | | | | FLT2 |
| Immediate7 | | | | | Op2 | | Immediate15 | | | | | | Rs16 | Opcode4 | Rd6 | | | | FLTLDI |
| Op2 | | OL2 | ~3 | | 02 | | ~3 | | Regno12 | | | | Rs16 | 54 | Rd6 | | | | CSR |
| Funct5 | | | | Cmd4 | | | ~11 | | | | Tn4 | | Rs16 | Opcode4 | Rd6 | | | | TLB |

Template bits 0 to 4 combined with the instruction slot number determine which functional unit the instructions is for. Template bits 5 to 7 are reserved and should be set to zero.

## Memory Loads

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode4 | xx00 | xx01 | xx10 | xx11 |
| 00xx | LDB | LDC | LDP | LDD |
| 01xx | LDBU | LDCU | LDPU | LDDR |
| 10xx | LDT | LDO |  |  |
| 11xx | LDTU | LDOU | LEA | {MLX} |

## Indexed Memory Loads

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Funct5 | xx000 | xx001 | xx010 | xx011 | xx100 | xx101 | xx110 | xx111 |
| 00xx | LDBX | LDCX | LDPX | LDDX | LDBUX | LDCUX | LDPUX | LDDRX |
| 01xx | LDTX | LDOX |  |  | LDTUX | LDOUX | LEAX |  |
| 10xx |  |  |  |  |  |  |  |  |
| 11xx |  |  |  |  |  |  |  |  |

## Memory Stores

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode4 | xx00 | xx01 | xx10 | xx11 |
| 00xx | STB | STC | STP | STD |
| 01xx |  |  |  | STDC |
| 10xx | STT | STO | CAS | PUSHC |
| 11xx | TLB | PUSH | CACHE | {MSX} |

## Indexed Memory Stores / Misc

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Funct5 | xx000 | xx001 | xx010 | xx011 | xx100 | xx101 | xx110 | xx111 |
| 00xx | STBX | STCX | STPX | STDX |  |  |  | STDCX |
| 01xx | STTX | STOX | CASX |  |  |  | CACHE |  |
| 10xx |  |  |  |  |  |  |  |  |
| 11xx | MEMSB | MEMDB |  |  |  |  |  |  |

## Flow Control

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode4 | xx00 | xx01 | xx10 | xx11 |
| 00xx | Bcc | BLcc | BRcc | NOP |
| 01xx | FBcc | BBc | BEQ # | BNE # |
| 10xx | JAL | JMP | CALL | RET |
| 11xx | CHK # | CHK | RTI / REX / Misc | BRK |

## Floating Point

Opcode4 - Bits 6 to 9

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Opcode4 | xx00 | xx01 | xx10 | xx11 |
| 00xx |  | {FLT2} | FAND # | FOR # |
| 01xx | FMA | FMS | FNMA | FNMS |
| 10xx |  |  |  |  |
| 11xx |  |  |  |  |

Op6 – Bits 22 to 27

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  |  |  |  | FADD | FSUB | FCMP |  | FMUL | FDIV |  |  | FAND | FOR |  |  |
| 1x | FMOV |  | FTOI | ITOF | FNEG | FABS | FSIGN | FMAN | FNABS | FCVTSD |  | FCVTSQ | FSTAT | FSQRT |  |  |
| 2x | FTX | FCX | FEX | FDX | FRM |  |  |  |  | FCVTDS |  |  |  |  |  |  |
| 3x |  |  |  |  |  |  | FSYNC |  | FSLT | FSGE | FSLE | FSGT | FSEQ | FSNE | FSUN |  |

|  |  |  |
| --- | --- | --- |
| Unit |  |  |
| 1 | Branch |  |
| 2 | Integer |  |
| 3 | Floating Point |  |
| 4 | Memory Load |  |
| 5 | Memory Store |  |
| 6 |  |  |
| 7 |  |  |

## Integer ALU {bits 32, 31, Opcode4}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x |  | {R1} | {R3} | {R3} | ADD | CSR | CMP | CMPU | AND | OR | XOR | {BF} | BLEND |  |  |  |
| 1x | SLT | SGE | SLE | SGT | SLTU | SGEU | SLEU | SGTU | SEQ | SNE |  | {BF} |  |  |  |  |
| 2x | MUL | MULU | DIV | DIVU | MOD | MODU |  |  | MADF |  |  | {BF} |  |  |  |  |
| 3x | FXMUL |  |  |  | ADDS1 | ADDS2 | ADDS3 |  | ANDS1 | ANDS2 | ANDS3 | {BF} | ORS1 | ORS2 | ORS3 |  |

## ALU – R3 Format {Funct5, bit 6}

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | MULH | MULUH | ADDV | SUBV | ADD | SUB | CMP | CMPU | AND | OR | XOR |  | NAND | NOR | XNOR |  |
| 1x | SLT | SGE | SLE | SGT | SLTU | SGEU | SLEU | SGTU | SEQ | SNE | MOV | CMOVNZ | MIN | MAX | PTRDIF |  |
| 2x | MUL | MULU | DIV | DIVU | MOD | MODU |  |  | MADF | MUX |  |  |  |  |  |  |
| 3x | FXMUL | FXDIV | SHL | ASL | SHR | ASR | ROL | ROR | SHL # | ASL # | SHR # | ASR # | ROL # | ROR # | BMM |  |

## ALU – R1 Format Funct5

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | x0 | x1 | x2 | x3 | x4 | x5 | x6 | x7 | x8 | x9 | xA | xB | xC | xD | xE | xF |
| 0x | CNTLZ | CNTLO | CNTPOP | COM | ABS | NOT | ISPTR | NEG | ZXT | ZXC | ZXB | ZXP | ZXO |  |  |  |
| 1x |  |  |  |  |  |  |  |  | SXT | SXC | SXB | SXP | SXO |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Template | Slot0 | Slot1 | Slot2 |
| 00 | Mem Unit | Int Unit | Int Unit |
| 01 | Mem Unit | Int Unit | Int Unit |
| 02 | Mem Unit | Int Unit | Int Unit |
| 03 | Mem Unit | Int Unit | Int Unit |
| 04 | Mem Unit |  |  |
| 05 | Mem Unit |  |  |
| 06 |  |  |  |
| 07 |  |  |  |
| 08 | Mem Unit | Mem Unit | Int Unit |
| 09 | Mem Unit | Mem Unit | Int Unit |
| 0A | Mem Unit | Mem Unit | Int Unit |
| 0B | Mem Unit | Mem Unit | Int Unit |
| 0C | Mem Unit | FP Unit | Int Unit |
| 0D | Mem Unit | FP Unit | Int Unit |
| 0E | Mem Unit | Mem Unit | FP Unit |
| 0F | Mem Unit | Mem Unit | FP Unit |
| 10 | Mem Unit | Int Unit | Branch Unit |
| 11 | Mem Unit | Int Unit | Branch Unit |
| 12 | Mem Unit | Branch Unit | Branch Unit |
| 13 | Mem Unit | Branch Unit | Branch Unit |
| 14 |  |  |  |
| 15 |  |  |  |
| 16 | Branch Unit | Branch Unit | Branch Unit |
| 17 | Branch Unit | Branch Unit | Branch Unit |
| 18 | Mem Unit | Mem Unit | Branch Unit |
| 19 | Mem Unit | Mem Unit | Branch Unit |
| 1A |  |  |  |
| 1B |  |  |  |
| 1C | Mem Unit | FP Unit | Branch Unit |
| 1D | Mem Unit | FP Unit | Branch Unit |
| 1E |  |  |  |
| 1F |  |  |  |

Templates

|  |  |  |  |
| --- | --- | --- | --- |
| Template | Slot0 | Slot1 | Slot2 |
| 00 | Int | Int | Int |
| 01 | Mem | Int | Int |
| 02 | Int | Mem | Int |
| 03 | Mem | Mem | Int |
| 04 | Int | Int | Mem |
| 05 | Mem | Int | Mem |
| 06 | Int | Mem | Mem |
| 07 | Mem | Mem | Mem |
| 08 | Branch | Int | Int |
| 09 | Int | Branch | Int |
| 0A | Branch | Branch | Int |
| 0B | Int | Int | Branch |
| 0C | Branch | Int | Branch |
| 0D | Int | Branch | Branch |
| 0E | Branch | Branch | Branch |
| 0F | FP | Int | Int |
| 10 | int | FP | int |
| 11 | FP | FP | int |
| 12 | Int | Int | FP |
| 13 | FP | Int | FP |
| 14 | Int | FP | FP |
| 15 | FP | FP | FP |
| 16 | Branch | Mem | Mem |
| 17 | Mem | Branch | Mem |
| 18 | Branch | Branch | Mem |
| 19 | Mem | Mem | Branch |
| 1A | Branch | Mem | Branch |
| 1B | Mem | Branch | Branch |
| 1C | Branch | FP | FP |
| 1D | FP | Branch | FP |
| 1E | Branch | Branch | FP |
| 1F | FP | FP | Branch |

|  |  |  |  |
| --- | --- | --- | --- |
| Template | Slot0 | Slot1 | Slot2 |
| 20 | Branch | FP | Branch |
| 21 | FP | Branch | Branch |
| 22 | Mem | FP | FP |
| 23 | FP | Mem | FP |
| 24 | Mem | Mem | FP |
| 25 | FP | FP | Mem |
| 26 | Mem | FP | Mem |
| 27 | FP | Mem | Mem |
| 28 | MemSt | MemLd | MemLd |
| 29 | MemLd | MemSt | MemLd |
| 2A | MemSt | MemSt | MemLd |
| 2B | MemSt | MemLd | MemSt |
| 2C | MemLd | MemSt | MemSt |
| 2D | MemLd | MemLd | MemSt |
| 2E | MemLd | MemSt | Int |
| 2F | MemSt | MemLd | Int |
| 30 | Int | MemLd | MemSt |
| 31 | Int | MemSt | MemLd |
| 32 | MemLd | Int | MemSt |
| 33 | MemSt | Int | MemLd |
| 34 | Branch | MemLd | MemSt |
| 35 | Branch | MemSt | MemLd |
| 36 | MemLd | Branch | MemSt |
| 37 | MemSt | Branch | MemLd |
| 38 | MemLd | MemSt | Branch |
| 39 | MemSt | MemLd | Branch |
| 3A | FP | MemLd | MemSt |
| 3B | FP | MemSt | MemLd |
| 3C | MemLd | FP | MemSt |
| 3D | MemSt | FP | MemLd |
| 3E | MemLd | MemSt | FP |
| 3F | MemSt | MemLd | FP |

|  |  |  |  |
| --- | --- | --- | --- |
| Template | Slot0 | Slot1 | Slot2 |
| 40 | Int | Int | Int |
| 41 | Mem | Int | Int |
| 42 | Int | Mem | Int |
| 43 | Mem | Mem | Int |
| 44 | Int | Int | Mem |
| 45 | Mem | Int | Mem |
| 46 | Int | Mem | Mem |
| 47 | Mem | Mem | Mem |
| 48 | Branch | Int | Int |
| 49 | Int | Branch | Int |
| 4A | Branch | Branch | Int |
| 4B | Int | Int | Branch |
| 4C | Branch | Int | Branch |
| 4D | Int | Branch | Branch |
| 4E | Branch | Branch | Branch |
| 4F | FP | Int | Int |
| 50 | int | FP | int |
| 51 | FP | FP | int |
| 52 | Int | Int | FP |
| 53 | FP | Int | FP |
| 54 | Int | FP | FP |
| 55 | FP | FP | FP |
| 56 | Branch | Mem | Mem |
| 57 | Mem | Branch | Mem |
| 58 | Branch | Branch | Mem |
| 59 | Mem | Mem | Branch |
| 5A | Branch | Mem | Branch |
| 5B | Mem | Branch | Branch |
| 5C | Branch | FP | FP |
| 5D | FP | Branch | FP |
| 5E | Branch | Branch | FP |
| 5F | FP | FP | Branch |

|  |  |  |  |
| --- | --- | --- | --- |
| Template | Slot0 | Slot1 | Slot2 |
| 60 | Branch | FP | Branch |
| 61 | FP | Branch | Branch |
| 62 | Mem | FP | FP |
| 63 | FP | Mem | FP |
| 64 | Mem | Mem | FP |
| 65 | FP | FP | Mem |
| 66 | Mem | FP | Mem |
| 67 | FP | Mem | Mem |
| 68 |  |  |  |
| 69 |  |  |  |
| 6A |  |  |  |
| 6B |  |  |  |
| 6C |  |  |  |
| 6D |  |  |  |
| 6E |  |  |  |
| 6F |  |  |  |
| 70 |  |  |  |
| 71 |  |  |  |
| 72 |  |  |  |
| 73 |  |  |  |
| 74 |  |  |  |
| 75 |  |  |  |
| 76 |  |  |  |
| 77 |  |  |  |
| 78 |  |  |  |
| 79 |  |  |  |
| 7A |  |  |  |
| 7B |  |  |  |
| 7C |  |  |  |
| 7D |  |  |  |
| 7E |  |  |  |
| 7F |  |  |  |

# Instruction Descriptions

## ABS – Absolute Value

**Description:**

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

**Instruction Format:** Integer R1

**Clock Cycles:** 0.33

**Execution Units:** Integer ALU

**Operation:**

If Ra < 0

Rt = -Ra

else

Rt = Ra

**Exceptions:** none

## 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 Formats**: Integer RI and R3

The R3 format adds three registers together.

**Clock Cycles:** 0.33

**Execution Units:** All ALU’s

**Exceptions**: none

**Notes**:

## ADDS1 – Addition Shifted 22 Bits

**Description**:

Add a register and an immediate value shifted to the left 22 bits. The immediate constant associated with the RI form of the instruction is sign extended to the left and zero extended to the right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other shifting instructions.

**Instruction Format**: Integer RI

**Clock Cycles**: 0.33

**Execution Units: All** Integer ALUs

**Exceptions**: none

## ADDS2 – Addition Shifted 44 Bits

**Description**:

Add a register and an immediate value shifted to the left 44 bits. The immediate constant associated with the RI form of the instruction is sign extended to the left and zero extended to the right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other shifting instructions.

**Instruction Format**: Integer RI

**Clock Cycles**: 0.33

**Execution Units: All** Integer ALUs

**Exceptions**: none

## ADDS3 – Addition Shifted 66 Bits

**Description**:

Add a register and an immediate value shifted to the left 66 bits. The immediate constant associated with the RI form of the instruction is zero extended to the right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other shifting instructions.

**Instruction Format**: Integer RI

**Clock Cycles**: 0.33

**Execution Units: All** Integer ALUs

**Exceptions**: none

## Bcc – Conditional Branch

**Description**:

If the branch condition is true, the target address is computed and loaded into the program counter. The branch is relative to the address of the branch instruction. The target address uses displace-plus-offset addressing to determine the address to branch to. The branch ranges is approximately +/- 4MB. If the instruction branches back to itself, a branch target exception will occur.

Target Address Calculation:

The low order 13 bits of the target address are loaded directly into the instruction pointer from the To13 field of the instruction. The high order bits of the target are calculated as the sum of the high order instruction pointer bits and a sign extended constant found in the Td field of the instruction.

**Instruction Format**:

**Instruction Format**:

For the register form of the instruction, an absolute address is loaded into the target register if the branch condition is true.

|  |  |  |
| --- | --- | --- |
| Cond3 | Mne. |  |
| 0 | BEQ | Rs1 = Rs2 signed |
| 1 | BNE | Rs1 <> Rs2 |
| 2 | BLT / BGT | Rs1 < Rs2 or Rs2 > Rs1 |
| 3 | BGE / BLE | Rs1 >= Rs2 or Rs2 <= Rs1 |
| 4 | reserved |  |
| 5 | reserved |  |
| 6 | BLTU | Rs1 < Rs2 (unsigned) |
| 7 | BGEU | Rs1 >= Rs2 (unsigned) |

**Clock Cycles**:

Typically, 2 with correct branch outcome and target prediction.

**Execution Units:** FCU Only

**Exceptions**: branch target

## BFINSI – Bitfield Insert Immediate

**Description**:

A bitfield is inserted into the target register Rd by combining a value read from Rs3 with a constant shifted to the left. The bitfield may not be larger than six bits. To accommodate a larger field multiple instructions can be used, or a value loaded into a register and the BFINS instruction used.

**Instruction Format**: Integer BFI

|  |  |
| --- | --- |
| Rg2 Bit |  |
| 0 | 1= Bo is a register spec, 0 = Bo is a six bit immediate |
| 1 | 1. = Bw is a register spec, 0 = Bw is a six bit immediate |

Rg[1] bit should always be clear for this instruction

**Clock Cycles**: 1

**Execution Units:** ALU #0 Only

**Exceptions**: none

## BMM – Bit Matrix Multiply

BMM Rd, Rs1, Rs2

**Description**:

The BMM instruction treats the bits of register Rs1 and Rs2 as an 8x8 bit matrix, performs a bit matrix multiply of the two registers and stores the result in the target register. An alternate mnemonic for this instruction is MOR. Only the least significant 64 bits of the registers are used.

**Instruction Format**: Integer R3

|  |  |
| --- | --- |
| Func2 | Function |
| 0 | MOR |
| 1 | MXOR |
| 2 | MORT (MOR transpose) |
| 3 | MXORT (MXOR transpose) |

**Operation**:

for I = 0 to 7

for j = 0 to 7

Rt.bit[i][j] = (Ra[i][0]&Rb[0][j]) | (Ra[i][1]&Rb[1][j]) | … | (Ra[i][7]&Rb[7][j])

**Clock Cycles:** 1

**Execution Units:** ALU #0 only

**Exceptions**: none

**Notes**:

The bits are numbered with bit 63 of a register representing I,j = 0,0 and bit 0 of the register representing I,j = 7,7.

## 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 $F[…]FFC0100. 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 |  |
| instruction pointer | ip\_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 |

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.

Hardware interrupts will cause a BRK instruction to be inserted into the instruction stream.

Because instructions are fetched in bundles a hardware interrupt always intercepts at a bundle address, and always returns to a bundle address.

**Instruction Format: BRK**

H = 1 = software interrupt – return address is next instruction

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

IMask4 = 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 cause code is bitwise ‘or’d with the value in register Rs1 to set the the cause CSR. Usually either one of the cause code field or Rs1 will be zero.

The empty instruction fields may be used to pass constant data to the break handler.

|  |  |  |
| --- | --- | --- |
| Operating Level | Address (If TVEC[0] contains $FFFC0000) |  |
| 0 | $FFFC0000 | Handler for operating level zero |
| 1 | $FFFC0020 |  |
| 2 | $FFFC0040 |  |
| 3 | $FFFC0060 |  |

## CACHE – Cache Command

CACHE Cmd, d[Rn]

CACHE Cmd, d[Ra + Rc \* scale]

**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:**

|  |  |  |
| --- | --- | --- |
| Cmd3 | 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) |
|  |  |  |

Operation:

Register Indirect with Displacement Form

Line = round32(sign extend(memory[displacement + Ra]))

Register-Register Form

Line = round32(sign extend(memory[Ra + Rc \* scale]))

Notes:

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

## CALL – Call Method

**Description**:

This instruction loads the instruction pointer with a constant value specified in the instruction. In addition, the address of the instruction following the CALL is stored in the return address register r61. This instruction may be used to implement subroutine calls.

**Instruction Format**: CALL

This format has a 64GB range.

If an address range greater than 36 bits is required, then the JAL instruction must be used.

**Execution Units:** Branch Unit

Clock Cycles:

## CHK – Check Register Against Bounds

**Description**:

A register is compared to two values. If the register is outside of the bounds defined by Rs2 and Rs3 or an immediate value then an exception will occur. Rs1 must be greater than or equal to Rs2 and Rs1 must be less than Rs3 or the immediate.

**Instruction Format**: CHK, CHKI

**Clock Cycles**: 0.33

**Exceptions**: bounds check

Notes:

The system exception handler will typically transfer processing back to a local exception handler.

## OR – Bitwise Or

**Description**:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the size of the register.

**Instruction Format**: Integer RI and R3

**Clock Cycles**: 0.33

**Execution Units: All** Integer ALUs

**Exceptions**: none

## ORS1 – Bitwise Or Shifted 22 Bits

**Description**:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the left and right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other ‘or’ and ‘or’ shifting instructions.

**Instruction Format**: Integer RI

**Clock Cycles**: 0.33

**Execution Units: All** Integer ALUs

**Exceptions**: none

## ORS2 – Bitwise Or Shifted 44 Bits

**Description**:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the left and right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other ‘or’ and ‘or’ shifting instructions.

**Instruction Format**: Integer RI

**Clock Cycles**: 0.33

**Execution Units: All** Integer ALUs

**Exceptions**: none

## ORS3 – Bitwise Or Shifted 66 Bits

**Description**:

Perform a bitwise or operation between operands. The immediate constant associated with the RI form of the instruction is zero extended to the right of the constant supplied in the instruction. This instruction allows building a 80-bit constant in a register when combined with other ‘or’ and ‘or’ shifting instructions.

**Instruction Format**: Integer RI

**Clock Cycles**: 0.33

**Execution Units: All** Integer ALUs

**Exceptions**: none