Hazard3

Table of Contents

Chapter 1. Introduction

Hazard3 is a 3-stage RISC-V processor, providing the following architectural support:

- `RV32I`: 32-bit base instruction set
- `M` extension: integer multiply/divide/modulo
- `C` extension: compressed instructions
- `Zicsr` extension: CSR access
- M-mode privileged instructions `ECALL`, `EBREAK`, `MRET`
- The machine-mode (M-mode) privilege state, and standard M-mode CSRs

The following are planned for future implementation:

- Support for `WFI` instruction
- Debug support
- `A` extension: atomic memory access
  - `LR`/`SC` fully supported
  - AMONone PMA on all of memory (AMOs are decoded but unconditionally trigger access fault without attempting memory access)
- Some nonstandard M-mode CSRs for interrupt control etc

Chapter 2. Instruction Cycle Counts

All timings are given assuming perfect bus behaviour (no stalls). Stalling of the `I` bus can delay execution indefinitely, as can stalling of the `D` bus during a load or store.

2.1. RV32I

Instruction	Cycles	Note
Integer Register-register		
`add rd, rs1, rs2`	1	
`sub rd, rs1, rs2`	1	
`slt rd, rs1, rs2`	1	
`sltu rd, rs1, rs2`	1	
`and rd, rs1, rs2`	1	
`or rd, rs1, rs2`	1	
`xor rd, rs1, rs2`	1	
`sll rd, rs1, rs2`	1	
`srl rd, rs1, rs2`	1	
`sra rd, rs1, rs2`	1	
Integer Register-immediate		
`addi rd, rs1, imm`	1	`nop` is a pseudo-op for `addi x0, x0, 0`
`slti rd, rs1, imm`	1	
`sltiu rd, rs1, imm`	1	
`andi rd, rs1, imm`	1	
`ori rd, rs1, imm`	1	
`xori rd, rs1, imm`	1	
`slli rd, rs1, imm`	1	
`srli rd, rs1, imm`	1	
`srai rd, rs1, imm`	1	
Large Immediate		
`lui rd, imm`	1	
`auipc rd, imm`	1	
Control Transfer		
`jal rd, label`	2[1]	
`jalr rd, rs1, imm`	2[1]	
`beq rs1, rs2, label`	1 or 2[1]	1 if nontaken, 2 if taken.

Instruction	Cycles	Note
`bne rs1, rs2, label`	1 or 2[1]	1 if nontaken, 2 if taken.
`blt rs1, rs2, label`	1 or 2[1]	1 if nontaken, 2 if taken.
`bge rs1, rs2, label`	1 or 2[1]	1 if nontaken, 2 if taken.
`bltu rs1, rs2, label`	1 or 2[1]	1 if nontaken, 2 if taken.
`bgeu rs1, rs2, label`	1 or 2[1]	1 if nontaken, 2 if taken.
Load and Store		
`lw rd, imm(rs1)`	1 or 2	1 if next instruction is independent, 2 if dependent.[2]
`lh rd, imm(rs1)`	1 or 2	1 if next instruction is independent, 2 if dependent.[2]
`lhu rd, imm(rs1)`	1 or 2	1 if next instruction is independent, 2 if dependent.[2]
`lb rd, imm(rs1)`	1 or 2	1 if next instruction is independent, 2 if dependent.[2]
`lbu rd, imm(rs1)`	1 or 2	1 if next instruction is independent, 2 if dependent.[2]
`sw rs2, imm(rs1)`	1	
`sh rs2, imm(rs1)`	1	
`sb rs2, imm(rs1)`	1	

2.2. M Extension

Timings assume the core is configured with `MULDIV_UNROLL = 2` and `MUL_FAST = 1`. I.e. the sequential multiply/divide circuit processes two bits per cycle, and a separate dedicated multiplier is present for the `mul` instruction.

Instruction	Cycles	Note
32 × 32 → 32 Multiply		
`mul rd, rs1, rs2`	1 or 2	1 if next instruction is independent, 2 if dependent.
32 × 32 → 64 Multiply, Upper Half		
`mulh rd, rs1, rs2`	18 to 20	Depending on sign correction
`mulhsu rd, rs1, rs2`	18 to 20	Depending on sign correction
`mulhu rd, rs1, rs2`	18	
Divide and Remainder		
`div`	18 or 19	Depending on sign correction
`divu`	18	
`rem`	18 or 19	Depending on sign correction
`remu`	18	

2.3. C Extension

All C extension 16-bit instructions on Hazard3 are aliases of base RV32I instructions. They perform identically to their 32-bit counterparts.

A consequence of the C extension is that 32-bit instructions can be non-naturally-aligned. This has no penalty during sequential execution, but branching to a 32-bit instruction that is not 32-bit-aligned carries a 1 cycle penalty, because the instruction fetch is cracked into two naturally-aligned bus accesses.

2.4. Privileged Instructions (including Zicsr)

Instruction	Cycles	Note
CSR Access		
`csrrw rd, csr, rs1`	1	
`csrrc rd, csr, rs1`	1	
`csrrs rd, csr, rs1`	1	
`csrrwi rd, csr, imm`	1	
`csrrci rd, csr, imm`	1	
`csrrsi rd, csr, imm`	1	
Trap Request		
`ecall`	3	Time given is for jumping to `mtvec`
`ebreak`	3	Time given is for jumping to `mtvec`

[1] A branch to a 32-bit instruction which is not 32-bit-aligned requires one additional cycle, because two naturally-aligned bus cycles are required to fetch the target instruction.

[2] If an instruction uses load data (from stage 3) in stage 2, a 1-cycle bubble is inserted after the load. Load-data to store-data dependency does not experience this, because the store data is used in stage 3. However, load-data to store-address (or e.g. load-to-add) does qualify.

Chapter 3. CSRs

The RISC-V privileged specification affords flexibility as to which CSRs are implemented, and how they behave. This section documents the concrete behaviour of Hazard3's standard and nonstandard M-mode CSRs, as implemented.

3.1. Standard CSRs

3.1.1. mvendorid

Address: `0xf11`

Read-only, constant. Value is configured when the processor is instantiated. Should contain either all-zeroes, or a valid JEDEC JEP106 vendor ID.

3.1.2. marchid

Address: `0xf12`

Read-only, constant. Architecture identifier for Hazard3, value can be altered when the processor is instantiated. Default is currently all zeroes as unregistered.

3.1.3. mimpid

Address: `0xf12`

Read-only, constant. Value is configured when the processor is instantiated. Should contain either all-zeroes, or some number specifiying a version of Hazard3 (e.g. git hash).

3.1.4. mstatus

blah blah

3.1.5. misa

Read-only, constant. Value depends on which ISA extensions Hazard5 is configured with.

3.2. Custom CSRs

These are all allocated in the space `0xbc0` through `0xbff` which is available for custom read/write M-mode CSRs, and `0xfc0` through `0xfff` which is available for custom read-only M-mode CSRs.

3.2.1. midcr

Address: `0xbc0`

Implementation-defined control register. Miscellaneous nonstandard controls.

Bits	Name	Description
31:1	**-**	RES0
0	`eivect`	Modified external interrupt vectoring. If 0, use standard behaviour: all external interrupts set interrupt `mcause` of 11 and vector to `mtvec + 0x2c`. If 1, external interrupts use distinct interrupt `mcause` numbers 16 upward, and distinct vectors `mtvec + (irq + 16) * 4`. Resets to 0. Has no effect when `mtvec[0]` is 0.

3.2.2. meie0

Address: `0xbe0`

External interrupt enable register 0. Contains a read-write bit for each external interrupt request IRQ0 through IRQ31. A `1` bit indicates that interrupt is currently enabled.

Addresses `0xbe1` through `0xbe3` are reserved for further `meie` registers, supporting up to 128 external interrupts.

An external interrupt is taken when all of the following are true:

- The interrupt is currently asserted in `meip0`
- The matching interrupt enable bit is set in `meie0`
- The standard M-mode interrupt enable `mstatus.mie` is set
- The standard M-mode global external interrupt enable `mie.meie` is set

`meie0` resets to **all-ones**, for compatibility with software which is only aware of `mstatus` and `mie`. Because `mstatus.mie` and `mie.meie` are both initially clear, the core will not take interrupts straight out of reset, but it is strongly recommended to configure `meie0` before setting the global interrupt enable, to avoid interrupts from unexpected sources.

3.2.3. meip0

Address: `0xfe0`

External IRQ pending register 0. Contains a read-only bit for each external interrupt request IRQ0 through IRQ31. A `1` bit indicates that interrupt is currently asserted. IRQs are assumed to be level-sensitive, and the relevant `meip0` bit is cleared by servicing the requestor so that it deasserts its interrupt request.

Addresses `0xfe1` through `0xfe3` are reserved for further `meip` registers, supporting up to 128 external interrupts.

When any bit is set in both `meip0` and `meie0`, the standard external interrupt pending bit `mip.meip` is also set. In other words, `meip0` is filtered by `meie0` to generate the standard `mip.meip` flag. So, an external interrupt is taken when *all* of the following are true:

- An interrupt is currently asserted in `meip0`

- The matching interrupt enable bit is set in `meie0`
- The standard M-mode interrupt enable `mstatus.mie` is set
- The standard M-mode global external interrupt enable `mie.meie` is set

In this case, the processor jumps to either:

- `mtvec` directly, if vectoring is disabled (`mtvec[0]` is 0)
- `mtvec + 0x2c`, if vectoring is enabled (`mtvec[0]` is 1) and modified external IRQ vectoring is disabled (`midcr.eivect` is 0)
- `mtvect + (mlei + 16) * 4`, if vectoring is enabled (`mtvec[0]` is 1) and modified external IRQ vectoring is enabled (`midcr.eivect` is 1). `
  - `mlei` is a read-only CSR containing the lowest-numbered pending-and-enabled external interrupt.

3.2.4. mlei

Address: `0xfe4`

Lowest external interrupt. Contains the index of the lowest-numbered external interrupt which is both asserted in `meip0` and enabled in `meie0`. Can be used for faster software vectoring when modified external interrupt vectoring (`midcr.eivect = 1`) is not in use.

Bits	Name	Description
31:5	-	RES0
4:0	-	Index of the lowest-numbered active external interrupt. A LSB-first priority encode of `meip0 & meie0`. Zero when no external interrupts are both pending and enabled.

3.2.5. Maybe-adds

An option to clear a bit in `meie0` when that interrupt is taken, and set it when an `mret` has a matching `mcause` for that interrupt. Makes preemption support easier.