

Hornet RISC-V Core Microarchitecture Reference Manual

Yavuz Selim Tozlu
oyavuz0@gmail.com

July 4, 2021

Preface

This manual describes the design of the Hornet core in detail. It elaborates on the design of each module in the core.

This document is provided so that those who wish to extend or modify the core can learn exactly how the core operates. The design diagrams are also presented to aid the reader.

Contents

Preface	ii
1 Introduction	1
2 Module Descriptions	2
2.1 Core	2
2.1.1 Pipeline Stages	2
2.1.1.1 Instruction Fetch Stage	2
2.1.1.2 Instruction Decode Stage	3
2.1.1.3 Execute Stage	4
2.1.1.4 Memory Stage	4
2.1.1.5 Writeback Stage	5
2.1.2 Submodule Functions	5
2.1.2.1 ALU	5
2.1.2.2 Control Unit	6
2.1.2.3 CSR Unit	6
2.1.2.4 Load-Store Unit	6
2.1.2.5 Immediate Decoder	6
2.1.2.6 Forwarding Unit	7
2.1.2.7 Hazard Detection Unit	7
2.2 ALU	7
2.3 Control Unit	7
2.4 CSR Unit	8
2.5 Load-Store Unit	9
2.6 Immediate Decoder	10
2.7 Forwarding Unit	10
2.8 Hazard Detection Unit	10
Appendix A	10

Chapter 1

Introduction

Hornet is a 32-bit RISC-V core developed by a couple of undergraduate students as part of their senior design project at Istanbul Technical University. The core is developed to be fully open-source, and for whatever purposes the public may see fit.

The core is designed to comply with the following RISC-V standards,

- RV32I Integer Instruction Set, Version 2.1 [1]
- "Zicsr", Control and Status Register (CSR) Instructions, Version 2.0 [1]
- Machine-Level ISA, Version 1.11 [2]

The RTL description of the core is written in Verilog HDL. The core is fully synthesizable, and FPGA proven.

The core has a classic, 5-stage pipelined microarchitecture. The design is heavily inspired by Patterson&Hennessy's computer architecture book[3]. Appendix A contains the complete pipeline diagram with full detail.

The core consists of 8 main modules, as listed below,

- Core - Describes the pipeline
- ALU - Arithmetic Logic Unit
- Control Unit - Generates control signals for muxes, ALU etc.
- CSR Unit - Contains CSRs. Manages trap handling
- Load-Store Unit - Generates the data memory interface signals
- Immediate Decoder - Generates 32-bit immediates
- Forwarding Unit - Implements forwarding logic
- Hazard Detection Unit - Implements pipeline stall logic

Chapter 2

Module Descriptions

This chapter describes each module in detail, along with the design diagrams.

2.1 Core

The Core module is the top module of the design. It instantiates all the other modules, and describes the 5-stage pipeline.

2.1.1 Pipeline Stages

This section describes the pipeline stages.

2.1.1.1 Instruction Fetch Stage

IF stage consists of an Instruction Memory, a Program Counter and 4 muxes.

The Instruction Memory is a Read-only Memory, a ROM. The PC is a register that acts as the input register of the Instruction Memory. As SRAM blocks usually have input registers embedded in them, the core outputs the `pc_i` as the instruction address, rather than `pc_o`. This is in contrast with what is seen on the pipeline diagram.

At the input of the PC, there are 4 muxes,

- Mux 1 chooses between the `mepc` value and the trap handler address. When a trap occurs, Mux 1 will output the trap handler address. When an `mret` instruction is executed, Mux 1 will output the `mepc` value to return from the trap. It is controlled by the CSR Unit.
- Mux 2 is used to stall the core. It either outputs the next `pc` value, `pc+4`, or the current `pc` value to stall the core.
- Mux 3 outputs the branch target address when a branch occurs. It is controlled by the branch decision logic in the EX stage.

- Mux 4 chooses between the outputs of Mux 1 and Mux 3. When an interrupt occurs, or an `mret` instruction is executed, it propagates Mux 1's output. Otherwise it propagates Mux 3's output.

IF stage is flushed whenever a branch is taken or a trap occurs.

IF stage is stalled whenever a data hazard or a misaligned memory access occurs.

2.1.1.2 Instruction Decode Stage

ID stage consists of a Control Unit, a Register File, an Immediate Decoder, and 3 muxes.

The Control Unit is responsible for generating control signals for muxes and the ALU; write enable signals, and exception signals. See the respective section for more details.

The Register File houses all 32 of the 32-bit RISC-V registers. It has two ports for reads, one port for writes. Note that, Register File is not a separate module, instead, it is defined in the `core` module. When a reset occurs, all the registers are reset to zero, asynchronously. Registers are written on the falling edge of the clock, and read on the rising edge of the clock.

The Immediate Decoder is responsible for generating the 32-bit immediate values as defined in the spec. It also generates the immediate values for the CSR instructions.

The 3 muxes group the control signals into 3 regions,

- Mux 1 outputs the control signals that are used in the WB stage.
- Mux 2 outputs the control signals that are used in the MEM stage.
- Mux 3 outputs the control signals that are used in the EX stage.

Figures below show the content of each region of control signals,

14	13	12	11	10	9	8	7	6	5	0
B	J	EX_MUX7	EX_MUX6	EX_MUX5	EX_MUX3	EX_MUX1	ALU			
1	1	1	1	1	2	2	6			

Figure 2.1: EX stage control region

2	1	0
MEM_LEN	MEM_WEN	
2	1	

Figure 2.2: MEM stage control region

6	5	4	3	2	1	0
WB_MUX	WB_SIGN	WB_RF_WEN	WB_CSR_WEN	WB_LEN		
2	1	1	1	2		

Figure 2.3: WB stage control region

When a pipeline stall occurs due to a data hazard, these muxes will output the signals associated with the `nop` instruction.

ID stage is flushed whenever a branch is taken or a trap occurs.

ID stage is stalled whenever a data hazard or a misaligned memory access occurs.

2.1.1.3 Execute Stage

EX stage consists of an ALU, a Load-Store Unit, branch logic, and 8 muxes.

The ALU performs the arithmetic operations for instructions. See the respective section for more details on ALU.

In the EX stage, the Load-Store Unit generates the data memory interface signals; `data_o`, `data_addr_o`, and `data_wmask_o`. See the respective section for more details.

The branch logic in EX stage consists of an adder that calculates the target address, and a simple logic circuit to determine if the branch will be taken. The circuit will decide on taking a branch if the instruction is a jump, or if the instruction is a branch and the branch condition is satisfied.

The 8 muxes operate as follows,

- Mux 1 chooses between the `rs1` data, the `pc`, and the CSR. The CSR input is chosen for some CSR instructions only. For other instructions, the appropriate input is chosen.
- Mux 2 implements forwarding for `rs1` data.
- Mux 3 chooses between the `rs1` data, the 32-bit immediate, and the CSR. The CSR input is chosen for some CSR instructions only. For other instructions, the appropriate input is chosen.
- Mux 4 implements forwarding for `rs2` data.
- Mux 5 chooses between the `rs1` data and the `pc`. Only JALR instruction uses the `rs1` data input. All other control transfer instructions use the `pc` input.
- Mux 6 chooses between the ALU output and the CSR. All CSR instructions choose the CSR, All other instructions choose the ALU output. The output of the mux propagates to the WB stage. It may be written to a register in the Register File, depending on the instruction.
- Mux 7 chooses between the ALU output and the 32-bit immediate value. All CSR instructions choose the ALU output. Only the LUI instruction chooses the 32-bit immediate value. The output of the mux propagates to the WB stage, and depending on the instruction, it may be written to a register in the Register File, or a CSR in the CSR Unit.
- Mux 8 implements forwarding for CSRs.

2.1.1.4 Memory Stage

MEM stage consists of a Data Memory, and a Load-Store Unit.

The Data Memory is a RAM. It is where the core stores its data.

In the MEM stage, the Load-Store Unit is responsible for shifting or concatenating the data coming in from the data memory. See the respective section for more details.

2.1.1.5 Writeback Stage

WB stage consists of the sign-extension circuitry, and a mux.

The sign-extension circuitry either zero-extends or sign-extends the data, depending on the instruction.

The WB mux chooses between the ALU output, the Data Memory output, and the 32-bit immediate value.

- Load instructions choose the Data Memory output.
- LUI instruction chooses the 32-bit immediate value.
- All other instructions choose the ALU output.

Output of this mux is the write data input for the Register File.

2.1.2 Submodule Functions

This section describes what each submodule does in the core. It does not elaborate on the inner working mechanisms of these submodules, however.

2.1.2.1 ALU

ALU is responsible for executing arithmetic operations for instructions, as listed below,

- For branch instructions, ALU does the register comparison operations, i.e. greater-or-equal, less-than, equal-to, not-equal-to. The address calculation for branch instructions are done in a separate adder outside the ALU.
- For jump instructions, ALU simply calculates $pc+4$. The address calculation for jump instructions are also done in a separate adder outside the ALU.
- For Load and Store instructions, ALU calculates the address of the data.
- For all arithmetic instructions, ALU executes the associated arithmetic operation.
- For the AUIPC instruction, ALU calculates the sum of the 32-bit immediate and the pc .

ALU also performs the necessary arithmetic operations for CSR instructions.

- For CSRRW and CSRWI instructions, ALU does nothing. It simply passes one of the inputs to the output.
- For CSRRS and CSRRSI instructions, ALU logically ORs the CSR with the input register or the 32-bit immediate.
- For CSRRC and CSRRCI instructions, ALU first logically inverts the register input or the 32-bit immediate, then logically ANDs the result with the CSR.

2.1.2.2 Control Unit

Control Unit generates the following control signals for the rest of the pipeline,

- Control signals for the ALU
- Control signals for muxes 1, 3, 5, 6 and 7 in the EX stage.
- Branch and Jump signals, B and J, which are used to determine branches.
- Length and the sign of a memory operation.
- Control signal for the WB mux.
- Write enable signals for Data Memory, Register File, and CSR Unit.

In addition to the control signals listed above, Control Unit is also responsible for detecting illegal instructions, ECALL, EBREAK and MRET instructions. It should be noted that the Control Unit is only responsible for the detection of these instructions. The necessary actions for these instructions are taken in the CSR Unit.

2.1.2.3 CSR Unit

CSR Unit houses the CSRs, handles traps, calculates trap handler addresses and flushes the pipeline upon a trap. It also controls a couple of muxes in the IF stage. The CSRs are also written on the falling edge of the clock, and read on the rising edge of the clock.

When a CSR instruction is executed, the CSR is read from the CSR Unit. It is modified appropriately in the ALU, and written back to the CSR Unit.

When a trap occurs, the CSR Unit saves the `pc` of the trapped instruction, flushes the pipeline and outputs the trap handler address. The pipeline diagram shows two muxes at the CSR Unit's `pc` input. These muxes choose the `pc` of the trapped instruction. However, what is not shown in the diagram is the input register for the `pc`. The `pc` input of the CSR Unit is registered outside the CSR Unit. This is necessary because the `mepc` register is written after the pipeline is flushed, and when the pipeline is flushed, the `pc` of the trapped instruction is also flushed.

2.1.2.4 Load-Store Unit

As the pipeline diagram shows, LSU operates in both the EX stage and the MEM stage.

In the EX stage, it generates the Data Memory interface signals. In the MEM stage, it processes the incoming data from the Data Memory.

LSU also handles misaligned memory accesses. First, it determines if the access is misaligned. If it is, the load or the store instruction is split into two parts, and two separate back-to-back accesses are performed. If it is a load instruction, LSU also does the combining of the two separate load accesses. Meanwhile, the IF and the ID stages are stalled.

2.1.2.5 Immediate Decoder

Immediate Decoder generates 32-bit immediate values for instructions that need it.

2.1.2.6 Forwarding Unit

Forwarding Unit forwards both integer registers and CSRs. It will forward data from MEM or WB stage to EX stage whenever a data hazard happens. It controls the muxes 2, 4 and 8 in EX stage.

2.1.2.7 Hazard Detection Unit

Hazard Detection Unit detects data hazards that require a pipeline stall. It stalls the pipeline whenever the instruction in the ID stage depends on a load instruction that is currently in the EX stage. In that case, IF and ID stages are stalled, and a NOP instruction is inserted to the EX stage on the next cycle.

2.2 ALU

ALU is a completely combinational circuit. It can perform the following arithmetic operations on two operands,

- Add, Subtract
- Logical XOR, OR, AND
- Set less than — signed and unsigned
- Shift left, right, right arithmetical
- Set if equal
- Set if not equal
- Set if greater or equal — signed and unsigned
- Add 4
- Pass

The **func1** input determines which of the above operations is used. In addition to the **func1** input, ALU has a **func2** input which determines if an input will be logically inverted before an AND operation. This functionality is only used for CSR instructions. Additionally, **func2** can determine which of the two inputs will be passed to the output for Pass operation.

The RTL description of the ALU is quite straightforward. We did not explicitly implement any arithmetic operations, instead took advantage of Verilog's arithmetic operators. This means it is up to the synthesis tool to implement the arithmetic operations. Luckily, modern synthesis tools are capable of synthesizing these operations.

2.3 Control Unit

Control Unit is a fully combinational circuit. Initially, it determines the major opcode, **func3** and **func7** fields of the instruction. Then, it generates the control signals by decoding these fields.

If no matching instructions are found, Control Unit will assert the illegal instruction signal.

If an MRET, ECALL or an EBREAK instruction is detected, Control Unit will assert the associated signal.

2.4 CSR Unit

CSR Unit implements a finite-state machine (FSM) to handle traps. Figure 2.4 shows a high-level FSM diagram.

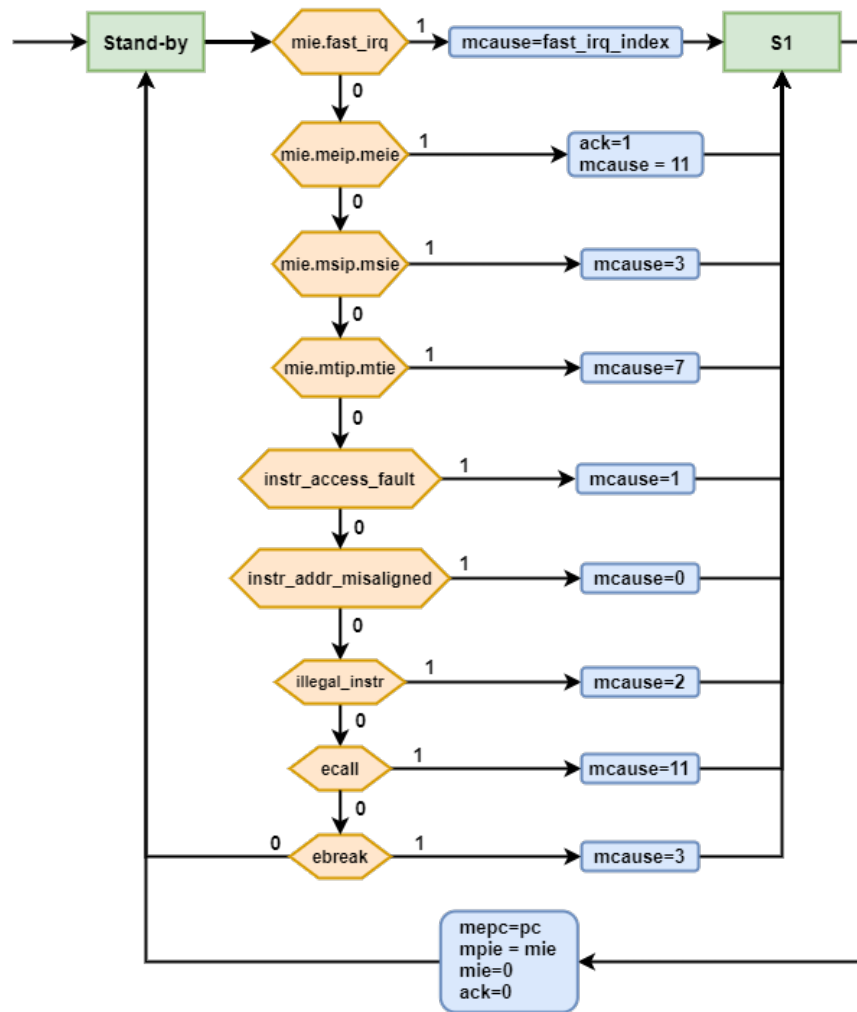


Figure 2.4: CSR Unit finite-state machine. The most-significant bit of the `mcause` is omitted in the diagram.

The FSM is initially at the **Stand-by** state. On every rising edge of the clock, the FSM moves to **S1** state if an interrupt is pending or an exception occurs. If that is the case, the `mcause` register will be updated appropriately.

Upon reaching the **S1** state, the FSM will always move back to the **Stand-by** state, while saving PC to `mepc`, and updating the necessary bits in the `mstatus` register.

The **ack** signal is asserted for one clock cycle when an external interrupt occurs. This signal is used to notify the interrupt handler outside the core.

Trap handling priority is in compliance with the spec. Fast interrupts have the highest priority. Then comes the external, software and timer interrupts, in order. Exceptions have the lowest priority.

CSR Unit implements a priority encoder in order to realize a priority scheme among multiple fast interrupt sources. The smallest indexed fast interrupt source has the highest priority, while the largest indexed fast interrupt source has the lowest priority.

CSR Unit also controls flush signals of the pipeline stages. When a particular pipeline stage is flushed, naturally, all the prior stages are also flushed. The PC of the latest flushed stage is saved. When a trap occurs, the pipeline is flushed according to the following algorithm,

- When an interrupt occurs, the latest pipeline stage without a dummy instruction is flushed. For example, if the instruction in MEM stage is a dummy instruction, but the instruction in the EX stage is a valid instruction, then IF, ID and EX stages are flushed, and PC of EX stage is saved in **mepc**.
- When an exception occurs, IF and ID stages are always flushed. If the exception is an instruction address misaligned exception, then EX stage is also flushed. If the exception is an instruction access fault exception, then EX and MEM stages are also flushed.

CSR Unit can generate trap handler addresses in either vectored mode or direct mode, as defined in the spec. In vectored mode operation, the address calculation requires the addition of the **mcause** register and the **mtvec** register. Since the **mcause** register is only updated on the falling edge of the clock, and the result of the addition must be ready before the next rising edge of the clock, this addition must be completed in half a clock cycle. To relax this constraint, the **mcause_buf** register is introduced. This register is updated on the rising edge of the clock, and the addition is calculated based on this register. Of course, the **mcause** register is updated to its new value on the consequent falling edge of the clock.

*It is possible to avoid implementing any adder circuit for this task. Implementations often constrain the values **mtvec** register can have. For example, an implementation might require that the **mtvec** value be 32-byte aligned. In that case, the 5 low-order bits of **mtvec** are guaranteed to be zero. Thus, for **mcause** values less than 32, the addition requires no logic operations, but simple wiring. We considered, but did not impose such constraints.*

When an MRET instruction reaches the ID stage, and there is no branch decision in the EX stage, CSR Unit will output the value in the **mepc** register. When this MRET instruction reaches the WB stage, CSR Unit will restore the interrupt enable bit from the **mpie** bit of **mstatus** register.

2.5 Load-Store Unit

LSU is a completely combinational circuit. It generates the address, the mask, and the data for memory operations, detects and handles misaligned accesses, and processes the data read from memory. The memory is assumed to work with 32-bit data. Thus, LSU only generates 4-byte aligned addresses. The write mask is used to write to a specific byte within the 32-bit data range.

In the EX stage, LSU generates the appropriate mask value based on the offset bits of the data address, which are two least-significant bits. In a similar manner, it also shifts the data as required.

In the MEM stage, LSU will shift the data read from memory based on the offset bits of the data address.

When a misaligned access occurs, LSU will perform two back-to-back memory accesses. For example, if a load instruction attempted to read 32-bit data from the address `0x0000_FFF1`, LSU will first read the 32-bit value at `0x0000_FFF0`, and place the upper 24-bit at the pipeline register. Then, it will read the 32-bit value at `0x0000_FFF4`, and place the lower 8-bit at the pipeline register. Thus, it will complete the misaligned access.

2.6 Immediate Decoder

Immediate decoder is a fully combinational circuit. It generates 32-bit immediate values as defined in the spec.

2.7 Forwarding Unit

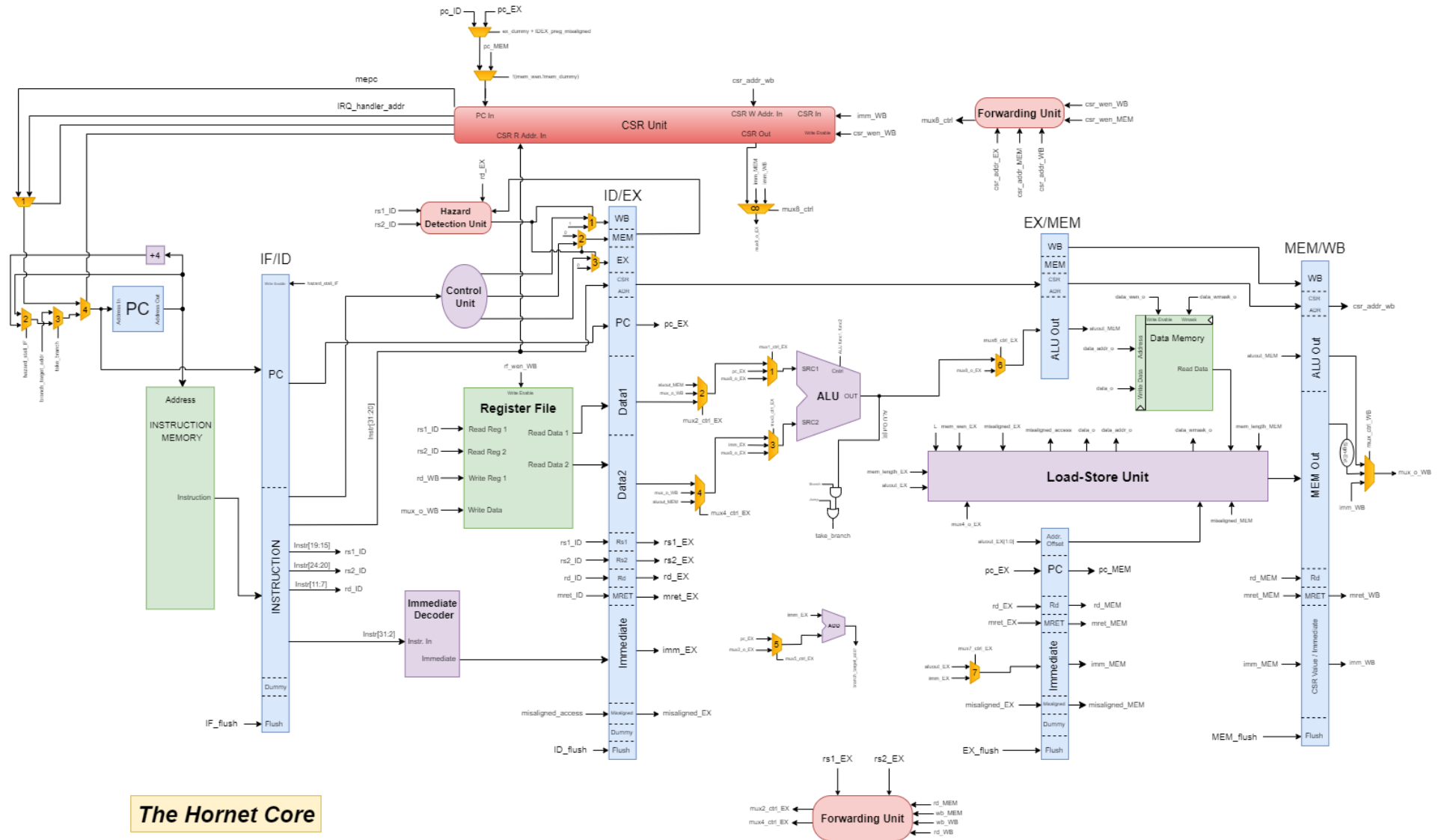
Forwarding Unit (FU) is a fully combinational circuit. It forwards both integer registers and CSRs.

FU will first check if the instruction in the MEM stage is going write to a register that the instruction in the EX stage needs. If that's the case, it will forward data from MEM to EX. If not, it will check the WB stage, and forward from there to EX, if necessary.

2.8 Hazard Detection Unit

Hazard Detection Unit (HDU) is a fully combinational circuit. It controls the stall signal of the pipeline. In order to detect a data hazard that requires a stall, it will first see if the instruction in the ID stage actually uses either `rs1`, `rs2`, or both. Then, it will see if the instruction in the EX stage is a load instruction, and if it is going to write to a register that the instruction in the ID stage uses. If these conditions are met, it will assert the stall signal.

Appendix A: The complete pipeline diagram



The Hornet Core

Bibliography

- [1] "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20191213", Editors Andrew Waterman and Krste Asanović, RISC-V Foundation, December 2019.
- [2] "The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20190608-Priv-MSU-Ratified", Editors Andrew Waterman and Krste Asanović, RISC-V Foundation, June 2019.
- [3] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. Morgan Kaufman, 2012.