Harvard 5 Stages Pipeline CPU Report

December 6, 2024

0.1 Contributors information

Name	**Section**	**ID**
Youssef Tarek	2	9220990
Marwan Mohamed	2	9220808
Youssef Roshdy	2	9220985
Moamen Hefny	2	9220886

Contents

Chapter 1

Introduction

This is the introduction section of the report. Here you can provide an overview of the topic, background information, and the purpose of the report.

Chapter 2

Instruction Set Architecture

Instruction Types and Classification

This document defines the instruction set architecture (ISA) for a hypothetical processor. The instructions are categorized into three main types: **R-Type**, **I-Type**, and **J-Type**, with an additional category for miscellaneous instructions referred to as **Others**. The bit allocation for each instruction type is detailed along with the format used for decoding.

Instruction Categories

R-Type (Register-to-Register Instructions)

R-Type instructions perform operations between registers and store the result in a destination register. They typically involve arithmetic and logical operations.

Opcode	Function	Rsrc1	Rsrc2	Rdst
[15-14]	[13-11]	[10-8]	[7-5]	[4-2]

Unused bits: [1-0]
Function field: 3 bits to specify the operation.
Supported Instructions:

- `NOT Rdst, Rsrc1`
- `INC Rdst, Rsrc1`
- `MOV Rdst, Rsrc1`
- `ADD Rdst, Rsrc1, Rsrc2`
- `SUB Rdst, Rsrc1, Rsrc2`
- `AND Rdst, Rsrc1, Rsrc2`

I-Type (Immediate and Memory Instructions)

I-Type instructions involve immediate values or memory access, typically for arithmetic or data transfer operations.

Opcode	Function	Rsrc1	Rdst
[15-14]	[13-11]	[10-8]	[7-5]

Unused bits: [4-0]
Second Fetch (Immediate or Offset):

Immediate / Offset
[15-0]

Supported Instructions:

1. `IADD Rdst, Rsrc1, Imm`
2. `LDM Rdst, Imm`
3. `LDD Rdst, offset(Rsrc1)`
4. `STD Rsrc1, offset(Rsrc2)`

J-Type (Control Flow Instructions)

J-Type instructions handle program control flow, such as jumps and calls.

Opcode	Function	Rsrc1
[15-14]	[13-11]	[10-8]

Unused bits: [7-0]
Supported Instructions:

1. `JZ Rsrc1`
2. `JN Rsrc1`
3. `JC Rsrc1`
4. `JMP Rsrc1`
5. `CALL Rsrc1`
6. `RET`
7. `RTI`
8. `INT index`

Other Instructions

These instructions do not fit into the R-Type, I-Type, or J-Type categories and typically control processor state or interact with peripherals.

Supported Instructions:

1. `NOP` – No operation.
2. `HLT` – Halt the processor.
3. `SETC` – Set the carry flag.
4. `OUT Rsrc1` – Output data from `Rsrc1`.
5. `IN Rdst` – Input data into `Rdst`.
6. `PUSH Rsrc1` – Push `Rsrc1` onto the stack.
7. `POP Rdst` – Pop value from the stack into `Rdst`.

Bit Allocation Overview

- **Opcode:** 2 bits `[15-14]`.
- **Function:** 3 bits `[13-11]`.
- **Register Fields:**
  - `Rsrc1`: 3 bits `[10-8]`.
  - `Rsrc2`: 3 bits `[7-5]` (R-Type only).
  - `Rdst`: 3 bits `[4-2]` (R-Type and I-Type).

Encoding Summary

R-Type Encoding

Utilizes two registers as sources and one destination register. The function field specifies the operation.

[Opcode — Function — Rsrc1 — Rsrc2 — Rdst — Unused(2 bits)]

I-Type Encoding

Involves one source register and one destination register, with a second fetch for immediate values or offsets.

[Opcode — Function — Rsrc1 — Rdst — Unused(5 bits)]

[Immediate/Offset (Second Fetch)]

J-Type Encoding

Uses a single register as the jump target.

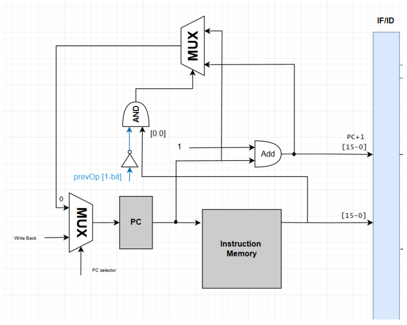
[Opcode — Function — Rsrc1 — Unused(8 bits)]

Pipeline Stages Design

The processor pipeline consists of five stages: **IF** (Instruction Fetch), **ID** (Instruction Decode), **EX** (Execution), **MEM** (Memory Access), and **WB** (Write Back). Each stage is responsible for specific tasks in the instruction execution process.

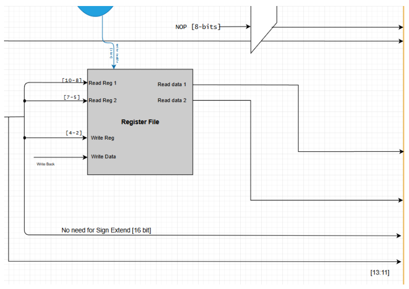
IF (Instruction Fetch)

- Fetch the instruction from memory using the Program Counter (PC).
- Increment the PC to point to the next instruction.
- Store the fetched instruction in the Instruction Register (IR).
- In case of a branch or jump instruction, update the PC accordingly.
- In case of a 'Halting' instruction, stop fetching new instructions by stalling the PC counter value.



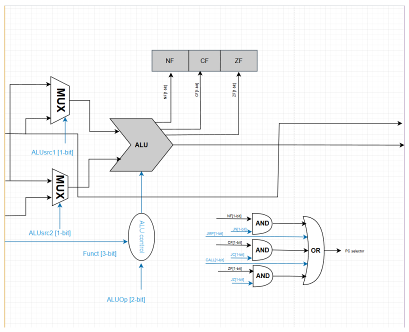
ID (Instruction Decode)

- Decode the instruction opcode and extract the necessary fields.
- Read the register file to get the values of the source registers.
- Determine the type of instruction and the required control signals.
- Calculate the effective address for memory operations if applicable.
- Forward register values and immediate operands to the next stage (EX).
- Stall the pipeline if a data hazard is detected and cannot be resolved with forwarding.



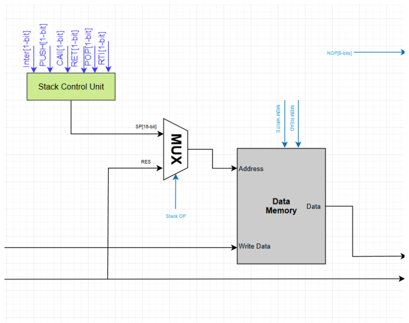
EX (Execution Stage)

- Select ALU inputs through multiplexers based on control signals (`ALUsrc1` and `ALUsrc2`).
- Perform the ALU operation based on the control signals (`ALUOp`) and the instruction's function field (`Func[3-bit]`).
- Generate the condition flags (`ZF, NF, CF`) based on the ALU result.
- Calculate the branch target address if the instruction is a branch or jump.
- Evaluate the branch condition using flags and control signals (`JMP`, `CALL`, `JC`, `JZ`, `JN`).
- Forward the ALU result to the Memory Access (MEM) stage.
- Stall the pipeline if a data hazard is detected and cannot be resolved with data forwarding.
- Flush the pipeline if a branch instruction is mispredicted.
- Generate control signals to update the Program Counter (PC) if a branch is taken.



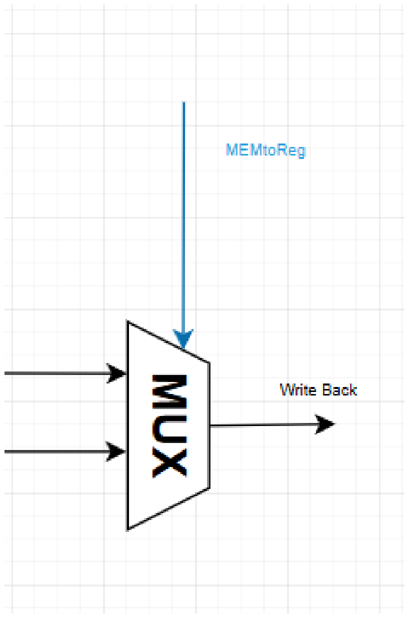
MEM (Memory Access)

- Access memory to read or write data based on the instruction type.
- Forward the memory read data to the Write Back (WB) stage.
- Stall the pipeline if a data hazard is detected and cannot be resolved with data forwarding.
- Stack control unit handles stack operations (PUSH, POP).
- Control unit generates signals to enable memory read/write operations.
- If the instruction doesn't involve memory access, forward the ALU result to the WB stage.



WB (Write Back)

- Update the register file with the new value if the instruction is a register operation.
- Update the Program Counter (PC) with the branch target address if a branch is taken.
- Stall the pipeline if a data hazard is detected and cannot be resolved with data forwarding.
- If the instruction is a jump or call, update the PC with the target address.
- If the instruction is a return, update the PC with the return address.
- If the instruction is a halt, stop the processor.



Conclusion

This ISA defines a compact instruction encoding scheme with distinct categories to handle arithmetic, memory, and control operations efficiently. The use of 2-bit opcodes and 3-bit function fields allows for flexible expansion while maintaining a clear structure for decoding.