5-stage Pipelined Processor Design Report

Team #4

Mohamed Shawky
SEC:2, BN:16

Remonda Talaat
SEC:1, BN:20

Evram Youssef
SEC:1, BN:9

Mahmoud Adas
SEC:2, BN:21

April 6, 2020

Contents

List of Figures

List of Tables

Part I

Introduction

0.1 System Overview

This document reports our design work of the 5-stage pipelined processor using Harvard architecture. We discuss the overall system blocks and connections, the functionalities of the different blocks and the hazard solutions.

0.2 Task Distribution

Table 1: Task Distribution

Team Member	Tasks
Mohamed Shawky	1) Overall system design. 2) Hazard detection and handling. 3) Document typing and formatting.
Remonda Talaat	1) Instruction format. 2) Interrupt Handling. 3) Overall system design.
Evram Youssef	1) Control unit and its signals. 2) Low level block design. 3) Pipeline buffers.
Mahmoud Adas	1) Low level block design. 2) Pipeline buffers. 3) Document typing and formatting.

Part II

Overall System

0.3 Overall System Design Schema

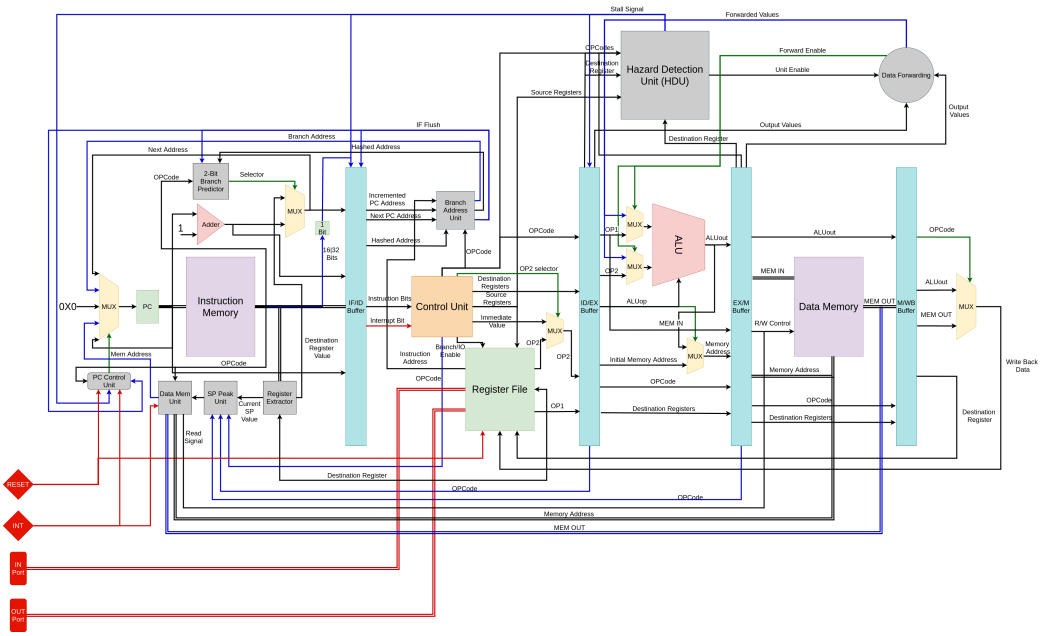

Figure 1: Overall System Design

0.4 Memory Specs

- We have 2 separate memory units, one for instructions and another for data and stack.
- Instruction Memory:
  - 2^{32} X 16 bits
  - 16-bit bus
- Data Memory:
  - 2^{32} X 16 bits
  - 32-bit bus
  - SP starts at 2^{32}-1

0.5 PC Control Unit

0.5.1 Inputs

- IF Flush (1 bit)
- Stall Signal (1 bit)
- RESET Signal (1 bit)
- Interrupt Signal (1 bit)
- Current OPCode (7 bits)

0.5.2 Outputs

- PC Mux Selectors (3 bits)

0.5.3 Logic

- If IF Flush == 1, Output = 001
- If RESET == 1, Output = 010
- If Stall == 1, Output = 011
- If Interrupt == 1 || OPCode == RET/RTI, Output = 100
- Else, Output = 000

0.6 Dynamic Branch Prediction

0.6.1 Inputs

- Hashed Address (4 bits)
- Update Bit (1 bit): *Taken or Not* to update FSM
- OPcode (4 bits)

0.6.2 Outputs

- Taken (1 bit): predict whether the branch taken or not

0.6.3 Logic

- Updates the FSM corresponding to the hashed address.
- Checks whether the OPCode is of a conditional branch instruction.
- Outputs the prediction bit *(Taken or Not)* accordingly.

0.7 Stack Pointer (SP) Peak Unit

0.7.1 Inputs

- Current SP (32 bits)
- Prev OPCodes ($3X7$ bits)

0.7.2 Outputs

- Expected SP (32 bits): stack pointer to read from after eliminating hazards

0.7.3 Logic

This unit peaks the value of the stack pointer on returning from a subroutine or an interrupt, in order to calculate the correct value of the address from which the original program counter read from memory. It checks for *POP/PUSH* instructions and use the count to update the address.
NOTE: In case we didn't use this unit, we will stall the pipe for three consecutive cycles to eliminate possible hazards.

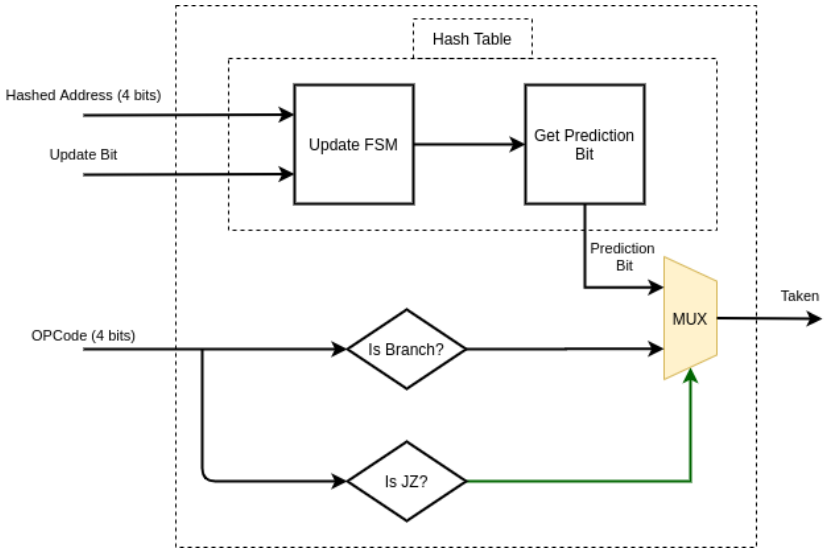

Figure 2: Branch Prediction Unit Diagram

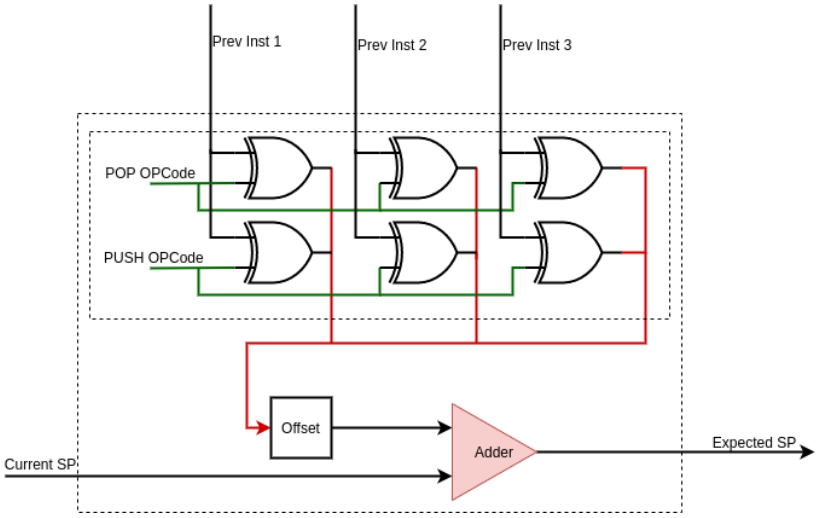

Figure 3: Stack Pointer (SP) Peak Unit Diagram

0.8 Branch Address Unit

0.8.1 Inputs

- Next PC Address (32 bits)
- Instruction Address (32 bits)
- Incremented PC Address (32 bits)
- Hashed Address (4 bits)
- OpCode (4 bits)

0.8.2 Outputs

- IF Flush (1 bit)
- Branch Address (32 bits)
- Hashed Address (4 bits)

0.8.3 Logic

- Check if OpCode is of a conditional branch instruction, if true:
  - Check whether PC Next Address is equal to Instruction Address
  - If true:
    - IF Flush = 0, Branch Address = Instruction Address
  - If false:
    - IF Flush = 1, Branch Address = Instruction Address

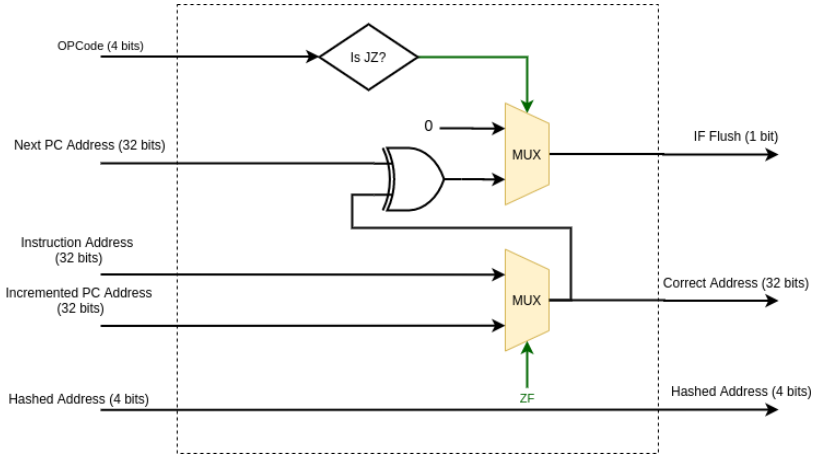

Figure 4: Branch Address Unit Diagram

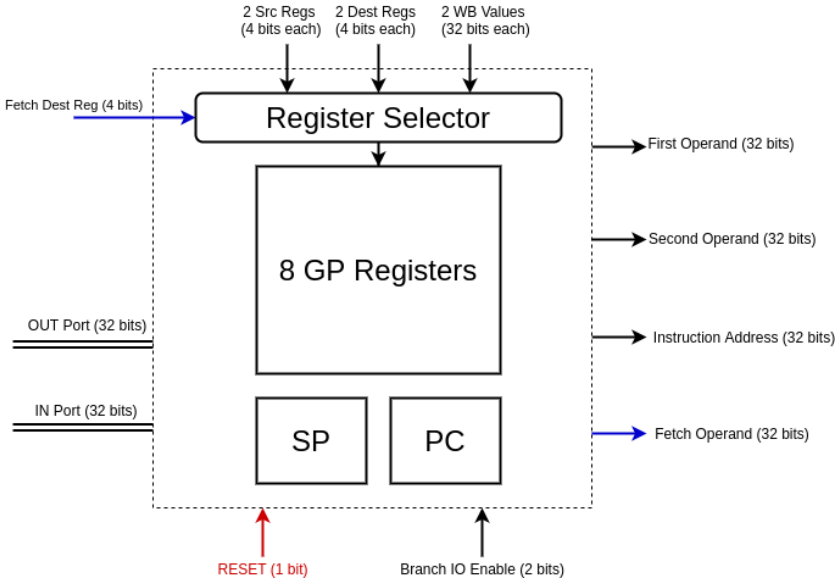

Figure 5: Register File Diagram

0.9 Register File

0.9.1 Registers

- 8 general purpose registers
- Stack pointer (SP) register
- Program counter (PC) register

0.9.2 Inputs

- Dest Regs: $2X4$ bits (for destination selection)
- SRC Regs: $2X4$ bits (for source selection)
- Fetch Reg: 4 bits (for fetch branch register selection)
- WB values: $2X32$ bits (for write back values)
- RESET: 1 bit (for registers clear).
- Branch/IO: 2 bits (to determine whether the operation is IO or branch)
- IN Port: 32 bits (IO input port)

0.9.3 Outputs

- OP1: 32 bits (value of first operand)
- OP2: 32 bits (value of second operand)
- Fetch Value: 32 bits (value of branch address required by fetch)
- Instruction Address: 32 bits (value of branch address)
- OUT Port: 32 bits (IO output port)

0.9.4 Logic

The register selector acts like a decoder to select the required operation and the register on which the operation performed.

0.10 ALU

0.10.1 Inputs

- ALUop: 4 bits (refer to ALU Operations below)
- Operands: $2X32$ bits (2 input operands)

0.10.2 Outputs

- ALUout: 32 bits (operation result)

0.10.3 ALU Operations

- 0000 – NOP – (no operation)
- 0001 – INC – (first operand + 1)
- 0010 – DEC – (first operand - 1)
- 0011 – ADD – (first operand + second operand)
- 0100 – SUB – (first operand - second operand)
- 0101 – AND – (first operand && second operand)
- 0110 – OR – (first operand || second operand)
- 0111 – NOT – (!first operand)
- 1000 – SHL – (shift first operand to the left)
- 1001 – SHR – (shift first operand to the right)
- 1010 – INC2 – (first operand + 2)
- 1011 – DEC2 – (first operand - 2)
- 1100 – INC4 – (first operand + 4)
- 1101 – DEC4 – (first operand - 4)

0.10.4 Logic

- ALU performs the operation and changes the CCR accordingly.
- The input operands of the ALU are multiplexed between forwarded data and register data, with selectors from data forwarding unit.

Part III

Instruction Format

0.11 One Operand Operations

- 4 bits (1111) for one operand instructions.
- 3 bits to define instruction.
- 3 bits for destination register.
- 1 bit to define the memory slots occupied by the instruction.
- Total of 11 bits, padded with 5 0's to fit 16 bits.

Table 2: One Operand Instruction Mapping

Operation	OpCode	Destination	16|32	Conditions
IN	1111000	000:111	0	————————
NOT	1111001	000:111	0	if !Rdst=0,Z=1 if !Rdst<0,N=1
INC	1111010	000:111	0	if Rdst+1=0,Z=1 if Rdst+1<0,N=1
DEC	1111011	000:111	0	if Rdst-1=0,Z=1 if Rdst-1<0,N=1
OUT	1111100	000:111	0	————————

0.12 Special Operations

- 16 0's to represent NOP (0000000000000000).

0.13 Two Operand Operations

- 4 bits to define instruction.
- 3 bits for each of Rsrc1, Rsrc2 and Rdst.
- 1 bit to define the memory slots occupied by the instruction.
- 16 bits for immediate values.
- Total of 14 bits in most cases with some exceptions mentioned below.

Table 3: Two Operand Instruction Mapping

Operation	OpCode	Rsrc1	Rsrc2	Rdst	imm	16|32	Conditions
SWAP	0001	000:111	—	000:111	—	0	———————— _
ADD	0010	000:111	000:111	000:111	—	0	if Result=0,Z=1 if Result<0,N=1
SUB	0011	000:111	000:111	000:111	—	0	if Result=0,Z=1 if Result<0,N=1
AND	0100	000:111	000:111	000:111	—	0	if Result=0,Z=1 if Result<0,N=1
OR	0101	000:111	000:111	000:111	—	0	if Result=0,Z=1 if Result<0,N=1
SHL	0110	000:111	—	—	16 bits	1	update carry flag
SHR	0111	000:111	—	—	16 bits	1	update carry flag
IADD	1000	000:111	—	000:111	16 bits	1	if Result=0,Z=1 if Result<0,N=1

0.14 Memory Operations

- 4 bits to define instruction.
- 3 bits for destination register.
- 1 bit to define the memory slots occupied by the instruction.
- 16 bits for immediate values.
- 20 bits for effective addresses.
- Total of 8 bits with no immediate values or effective addresses.
- Total of 24 bits with immediate values.
- Total of 28 bits with effective addresses.

Table 4: Memory Instruction Mapping

Operation	OpCode	Rdst	imm	EA	16|32	Conditions
PUSH	1001	000:111	—	—	0	————————
POP	1010	000:111	—	—	0	————————
LDM	1011	000:111	16 bits	—	1	————————
LDD	1100	000:111	—	20 bits	1	————————
STD	1101	000:111	—	20 bits	1	————————

0.15 Branch and Change Control Operations

- 4 bits (0000) for branching instructions.
- 3 bits to define instruction.
- 3 bits for destination register.
- 1 bit to define the memory slots occupied by the instruction.
- Total of 11 bits, padded with 5 0's to fit 16 bits.

Table 5: One Operand Instruction Mapping

Operation	OpCode	Destination	16|32	Conditions
JZ	0000001	000:111	0	————————
JMP	0000010	000:111	0	————————
CALL	0000011	000:111	0	————————
RET	0000100	—	0	————————
RTI	0000101	—	0	————————

Part IV

Control Signals

0.16 Overview

Control unit is responsible for generating the control signals that are used to activate several operations throughout the pipeline. Also, it's responsible for the extraction of specific information from instruction bits.

- Immediate values (16 bits)
- Effective addresses (20 bits)
- source registers (4 bits)
- destination registers (4 bits)
- OPCode (4:7 bits)
- Branch/IO Enable (2 bits)
- ALUop (4 bits)
- Interrupt Enable (1 bit)
- Memory R/W Enable (2 bits)
- second operand selector (1 bit)

TODO: Add related control signals for each instruction

Part V

Pipeline Stages

0.17 Overview

This section discusses the 5 stages of our system and their functionalities.

0.17.1 Fetch Stage

- Responsible for fetching the next instruction.
- Can take two cycles in case of 32-bit instructions.
- Contains a branch prediction unit to determine the next address to be fetched in case of branching.
- Outputs the instruction bits into IF/ID Buffer.
- The current instruction is fetched at the first half of cycle, then the next PC value calculations are done in the second half.

0.17.2 Decode Stage

- Responsible for decoding the instruction bits into control signals.
- Outputs the corresponding signals to ID/EX Buffer.
- Contains register file to output operand values and register-related operations.
- Determines the correct branch address in case of branching instructions by using Branch Address Unit.
- The control unit deduces the corresponding signals in the first half of cycle, then the register operations and branch address calculation are done in the second half of cycle.

0.17.3 Execute Stage

- Responsible for ALU operations.
- Determines the correct ALU output and pass it with other signals to EX/M Buffer.
- The ALU operations and CCR update are done in the first half of cycle.

0.17.4 Memory Stage

- Responsible for Data Memory IO.
- Memory read/write is done in the first half of cycle.

0.17.5 Write-Back Stage

- Responsible for passing correct output values to the destination registers.
- Write back is done in the first half of cycle.

0.18 IF/ID Buffer

0.18.1 Registers

- Instruction Register (32 bits)
- Next Address Register (32 bits)
- Incremented PC Register (32 bits)
- Hashed Address Register (4 bits)
- Interrupt Register (1 bit)

0.18.2 Control Signals

- Flush: clear buffer (1 bit)
- Stall: freeze buffer (1 bit)

0.19 ID/EX Buffer

0.19.1 Registers

- Operand Registers ($2X32$ bits)
- Destination Register (4 bits)

- OpCode Register (7 bits)
- R/W Register (2 bits)

0.19.2 Control Signals

- Stall (IN): freeze buffer (1 bit)
- Destination Register (OUT) (4 bits)
- Output Values (OUT) (32 bits)

0.20 EX/M Buffer

0.20.1 Registers

- ALUout Register (32 bits)
- MEM IN Register (32 bits)
- Opcode Register (7 bits)
- Destination Register (4 bits)
- R/W Register (2 bits)

0.20.2 Control Signals

- Destination Register (OUT) (4 bits)
- Output Values (OUT) (32 bits)

0.21 M/WB Buffer

0.21.1 Registers

- ALUout Register (32 bits)
- MEM OUT (32 bits)

- OpCode (7 bits)
- Destination Register (4 bits)

Part VI

Pipeline Hazards and solutions

0.22 Structural Hazards

0.22.1 Detection

The structural hazard occurs in data memory and register file.

0.22.2 Handling

The structural hazard in data memory is solved by using 2 memory units, one for instructions and one for data. Both have the same specs *(previously mentioned)*.
However structural hazard in register file is handled by forcing the write back to happen in the first half of the clock cycle and the decode to happen in the second half.

0.23 Data Hazards

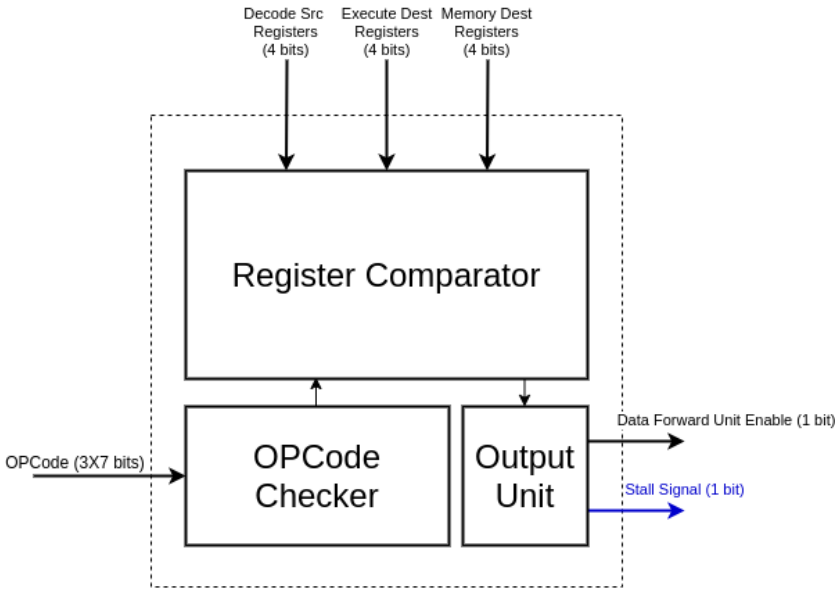

Figure 6: Hazard Detection Unit Diagram

0.23.1 Detection

Hazard Detection Unit (HDU)

HDU consists of 3 parts:

- **OPCode Checker:** checks the opcode of the current instruction to check whether it will cause data hazard or not, then activates the Register Comparator accordingly. Also, it checks for *load-use case*, in order to activate the stall signal.
- **Register Comparator:** compares the decode source registers with the destination registers of the execute and memory stages.
- **Output Unit:** outputs stall signal in case of load and pop instructions and data forward unit enable in case of other data hazards.

0.23.2 Handling

Stall

Occurs only at Fetch and Decode stage, due to load(pop) use case.

- Fetch same instruction (don't increment the program counter).
- Latch IF/ID buffer with the same values.
- Freeze Decode stage.
- Clear ID/EX buffer.

Data Forwarding

- EX/MEM buffer $->$ Execute / Decode.
- ID/EX buffer $->$ Decode.

0.24 Control Hazards

0.24.1 Detection

The branch address calculation occurs in the Decode stage. So, the hazard might affect only the Fetch stage, which will be flushed in case of wrong address prediction.

0.24.2 Handling

- At Fetch stage, always check the branch predictor and calculate the next address accordingly.
- At Decode stage, we have a *Branch Address Unit* that checks whether the OPCode is of a branch operation. If so, it passes the address to the program counter and compares the correct address with the address of the counter to decide whether to flush the Fetch stage or not.

Flush

Occurs only at Fetch Stage, due to wrong branch prediction at Decode stage.

- Load new address in the program counter.
- Remove fetched instructions from IF/ID buffer.

Dynamic Branch Prediction

We use 2-bit branch predictor, which is a hash table of *Finite State Machines* (FSMs) to predict whether the branch will be taken (1) or not (0) at each individual branch address.