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

May 5, 2020

Contents

List of Figures

List of Tables

Part I

Introduction

0.1 Abstract

This report describes our design work of the 5-stage pipelined processor, that follows Harvard architecture.

This report contains:

- the overall system blocks and connections.
- the functionalities of the different blocks.
- the hazard solutions.

Part II

Overall System Design

0.2 Overall System Design Schema

Figure 1 shows the overall system design in detail. Each unit is described in details in its section.

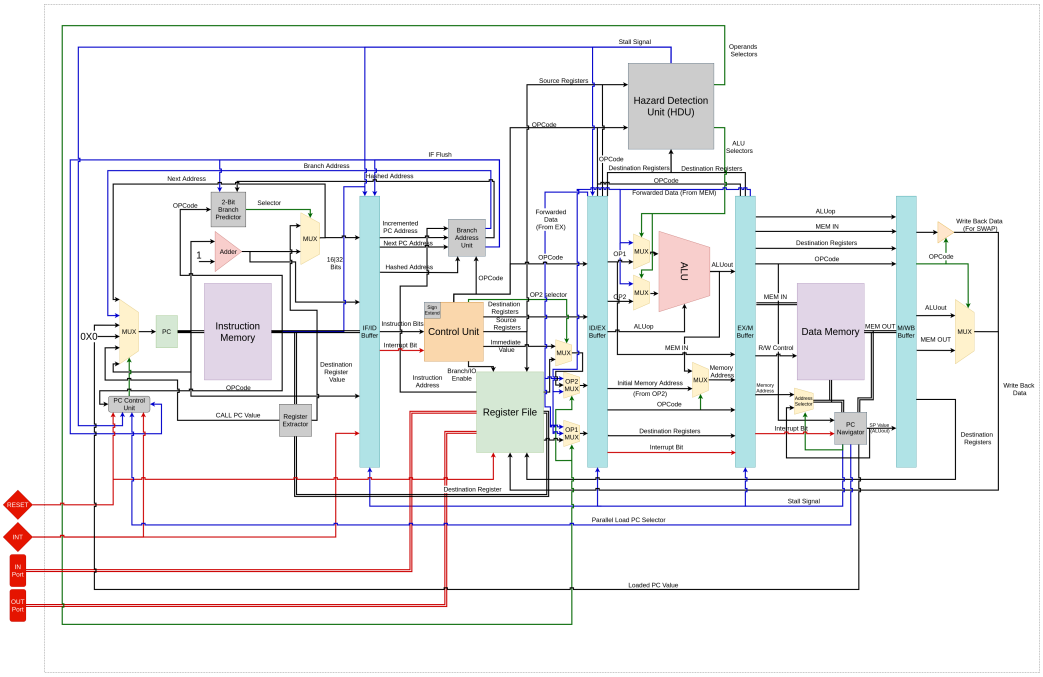

Figure 1: Overall System Design

0.3 Memory Specs

The cpu follows Harvard architecture and thus uses the following 2 separate memory units:

- Instructions Memory: Read-only, stores instructions.
  - **Word Width**: 16 bits.
  - **Address Bus Width**: 8 bits.
  - **Data Bus Width**: 16 bits.
  - **Total Number of Words**: 2^{16} words = 65,536 words = 131,072 bytes.

  - **Valid Address Range**: From 0x0000 inclusive to 0xFFFF inclusive.

- Data Memory: Read-Write, stores data and the stack.
  - **Word Width**: 16 bits.
  - **Address Bus Width**: 32 bits.
    For simulation, this memory will ignore bits from (31 to 10) inclusive, and only work with bits from (9 to 0) inclusive.
  - **Data Bus Width**: 32 bits.
    The higher bits (31 downto 16) are data at address A .
    The lower bits (15 downto 0) are data at address A+1 .
    where $A \ mod \ 2 = 0$.
    On read, data-memory loads data bus with data from A and A+1 .
    On write, data-memory stores data from data bus to both A and A+1 addresses.
  - **Total Number of Words**: 2^{32} words = 4,294,967,296 words = 8,589,934,592 bytes.
    For simulation, this memory will only have: 2^{10} words = 1,024 words = 2,048 bytes.
  - **Valid Address Range**: Even Adresses From 0x0000_0000 inclusive to 0xFFFF_FFFF exclusive. Formaly $A \in$[0x0000_0000, 0xFFFF_FFFF) and $A \ mod \ 2 = 0$.
    For simulation, range will become: Even Addresses From 0x0000_0000 inclusive to 0x0000_0400 exclusive. Formaly $A \in$[0x0000_0000, 0x0000_0400) and $A \ mod \ 2 = 0$.

0.4 PC Control Unit

0.4.1 Inputs

- IF Flush (1 bit)
- Stall Signal (1 bit)
- RESET Signal (1 bit)
- Interrupt Signal (1 bit)

- Current OPCode (7 bits)
- Parallel Load PC Selector (1 bit)

0.4.2 Outputs

- PC Mux Selectors (3 bits)

0.4.3 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
- PL PC Selector == 1, Output = 101
- Else, Output = 000

0.5 Dynamic Branch Prediction

Figure 2 shows the branch prediction unit.

0.5.1 Inputs

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

0.5.2 Outputs

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

0.5.3 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.

0.6 Branch Address Unit

Figure 3 shows the branch address unit.

0.6.1 Inputs

- Next PC Address (32 bits)
- Instruction Address (32 bits)
- Incremented PC Address (32 bits)
- Hashed Address (4 bits)
- OpCode (4 bits)
- CCR (3 bits)

0.6.2 Outputs

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

0.6.3 Logic

- Check if OpCode is of a conditional branch instruction, if true:
  - Check whether PC Next Address is equal to Instruction Address
  - If true:

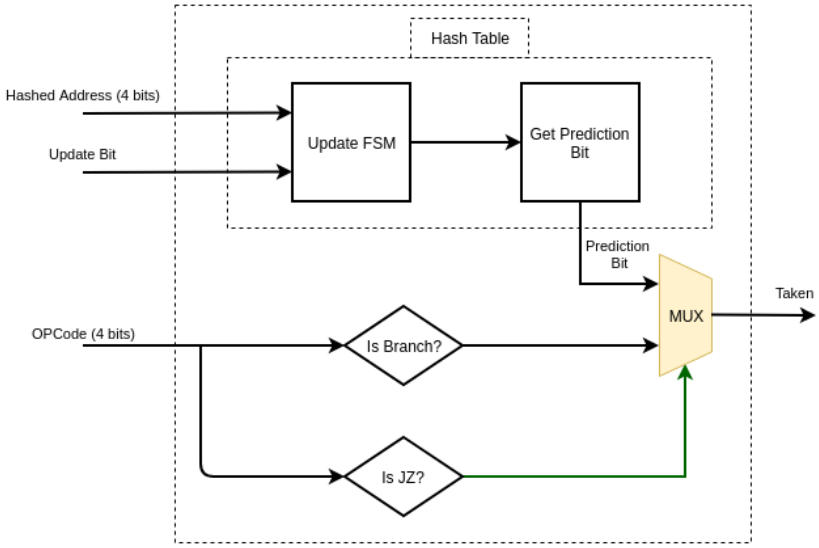

Figure 2: Branch Prediction Unit Diagram

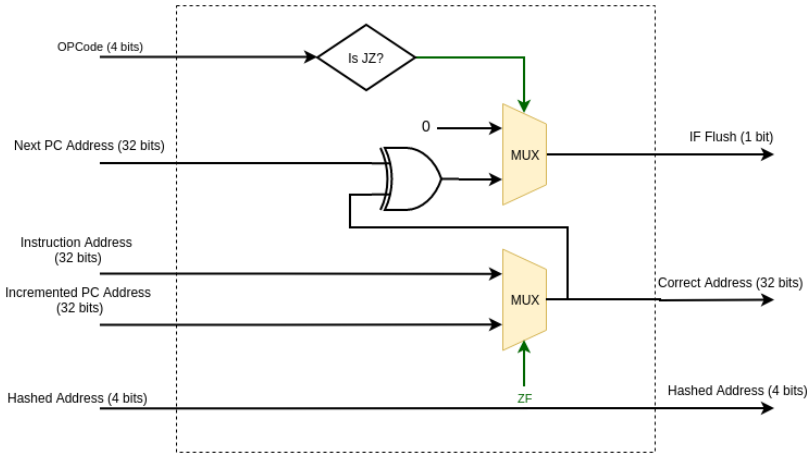

Figure 3: Branch Address Unit Diagram

    * IF Flush = 0, Branch Address = Instruction Address
  - If false:
    * IF Flush = 1, Branch Address = Instruction Address

0.7 Register File

Figure 4 shows the register file.

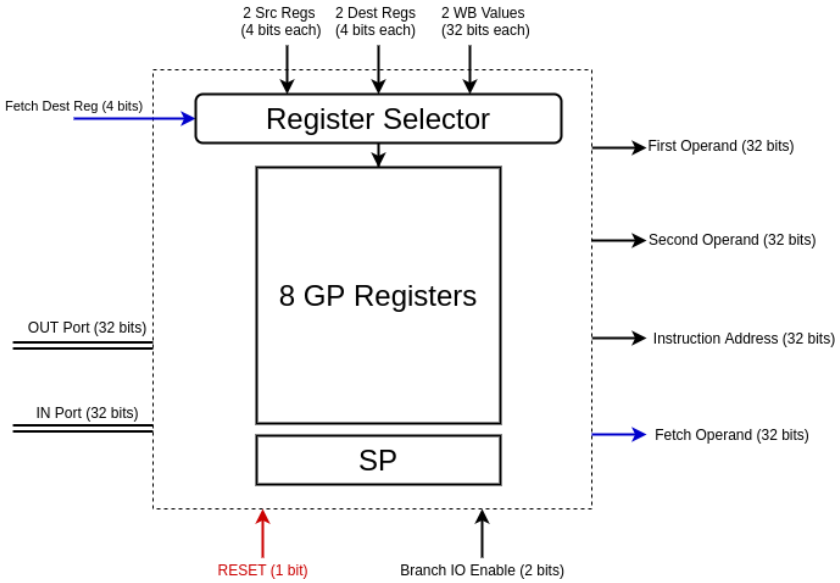

Figure 4: Register File Diagram

0.7.1 Registers

All internal registers are 32bit in width. Each one has a 4bit address to select it, either for reading or writing.

- 8 general purpose registers, addresses (0 to 7).
- Stack pointer (SP) register with address = 8.

0.7.2 Inputs

- Dest Regs: 2×4 bits (for destination selection)
- SRC Regs: 2×4 bits (for source selection)
- Fetch Reg: 4 bits (for fetch branch register selection)
- WB values: 2×32 bits (for write back values)
- RESET 1 bit: async, clears all registers.
- BR_IO_ENBL: 2 bits (to determine whether the operation is IO or branch)
  00 = Do Nothing
  01 = In
  10 = Out
  11 = Branch
- IN Port: 32 bits (IO input port).
- CLK: input operations are in rising edge, while output operations are in falling edge.

0.7.3 Outputs

- OP1: 32 bits (value of first operand (src0))
- OP2: 32 bits (value of second operand (src1))
- 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)

0.7.4 Logic

The register selector acts like a decoder to select the required operation and the register on which the operation performed. It operates in the following manner:

- if RESET == 1, then:
  - All registers are set to 0.
- Else if BR_IO_ENBL == 01, then:
  - Input from IN Port to first Dest Reg.
- Else if BR_IO_ENBL == 10, then:
  - Output from first Src Reg to OUT Port.
- Else if BR_IO_ENBL == 11, then:
  - Output from first Src Reg to Instruction Address.
- Else if BR_IO_ENBL == 00, then:
  - Do normal read/write operations based on Src and Dest Regs, as well as WB values.

0.8 ALU

0.8.1 Inputs

- ALUop: 4 bits (refer to ALU Operations below)
- Operands: 2×32 bits (2 input operands)

0.8.2 Outputs

- ALUout: 32 bits (operation result)
- CCR: 3 bits

0.8.3 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 with the value of second operand)
- 1001 – SHR – (shift first operand to the right with the value of second operand)
- 1010 – INC2 – (first operand + 2)
- 1011 – DEC2 – (first operand - 2)

0.8.4 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.

0.9 PC Navigator

Figure 5 shows the PC Navigator.

0.9.1 Inputs

- Interrupt Bit (1 bit)
- OPCode Signals (2 bits): to check whether the operation is RET or RTI
- Previous SP (32 bits): to increment or decrement it correspondingly to access Data Memory
- MEM OUT (32 bits): loaded PC from memory

0.9.2 Outputs

- Stall Signal (1 bit)
- New SP (32 bits)
- PC Selector (1 bit): to enable PC parallel load from Data Memory
- Loaded PC Value (32 bits)

0.9.3 Logic

- The Operation Checker checks the OPCode Signals and Interrupt Bit to check whether the operation is RET, RTI or Interrupt and produces its signals accordingly:
  - PC Return Enable is set.
  - Counter is set to 0 for RET, 1 for RTI and 2 for Interrupt.
  - Operation Selector is issued to the Adder/Subtractor to change the value of SP.
- Stall Counter counts the number of stalls produced by operation. It's set to 0 for RET, 1 for RTI and 2 for Interrupt. It counts down a number of cycles and then release the stall.
- Adder/Subtractor is used to update the SP every stall cycle, to get the right data from the memory.

- PC Return issues a signal to the PC Control Unit to enable parallel load from data memory and passes the loaded PC value.
- RET doesn't stall any cycles, as it only loads PC value from data memory.
- RTI stall only one cycle, as it loads PC value from data memory and then loads CCR.
- Interrupt stall two cycles, as it loads PC value from data memory, pushes CCR into stack and then pushes PC into stack.

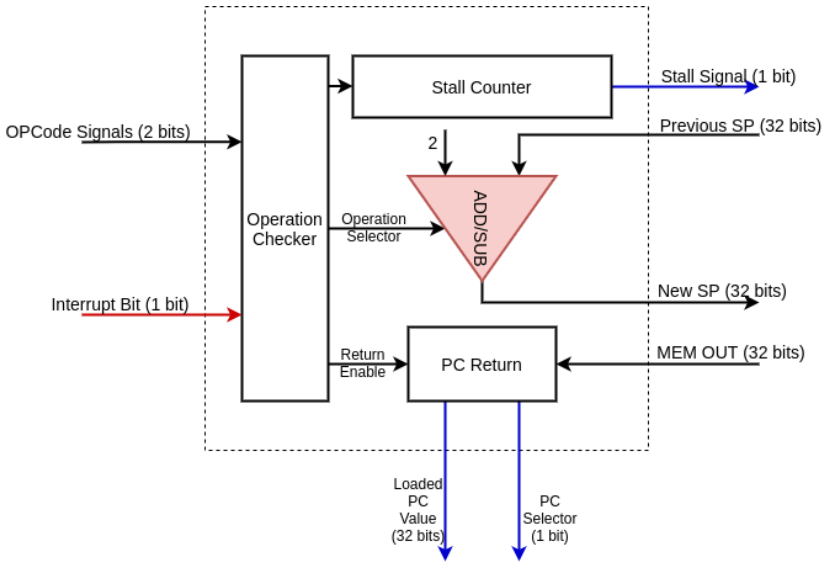

Figure 5: PC Navigator Diagram

Part III

Instruction Format

0.10 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 1: 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	————————

0.11 Special Operations

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

0.12 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 2: 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

0.13 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 3: 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	————————

0.14 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 4: 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 Unit (Signals)

0.15 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.

- It communicates with:
  - IF/ID buffer: for reading the instruction bits.
  - ID/EX buffer: for writing the appropriate registers, ALUop and signals.
  - Register file: for selecting the registers needed to be read (Rsrc1 and Rsrc2).
  - Hazards units (HDU and Branch Address Unit): for sending enables and needed signals.
- Unit Interface:
  - Inputs:
    - Instruction Bits (32 bits)
    - Interrupt bit (1 bit)
  - Outputs:
    - Rsrc2_val (32 bits) for immediate values or effective addresses
    - Rsrc1_sel (4 bits)
    - Rsrc2_sel (4 bits)
    - Rdst1_sel (4 bits)
    - Rdst2_sel (4 bits) used only in case of swap
    - Branch/IO Enable (2 bits)
    - OP2_sel (1 bit)
    - SP Enable (1 bit)
    - OpCode (7 bits)
    - Branch Enable (1 bit)
    - ALUop (4 bits)
    - R/W Control Signal (2 bits)
- Interpretation:

- **Rsrc2_val (32 bits):** occupies a single place in the ID/EX buffer. However, it's used in many different ways. It can be used as a register value extracted form register file. it can be used as an immediate value extracted from IF/ID buffer. Also, it can hold the stack pointer address, as well as the effective address (EA) sent to the memory for reading or writing.
- **Rdst2 (4 bits):** only used when dealing with a SWAP instruction, thus we need Op1 and Op2 and their new selectors.
- **OP2_sel (1 bit):** determines the value of Rsrc2 register in ID/EX buffer, whether it's immediate or register value.
- **Branch/IO Enable (2 bits):** informs the register file what operation of these are we executing (No/In/Out/Branch), however *Branch Enable* (1 bit) interacts with the Branch Address Unit, informing it what type of OpCode are we dealing with (branching or not).

0.16 Control Signals

In this section, instructions are divided into seven types based on the signals produced:

- One Operand (not,inc,dec,out,in).
- Two Operands (add,sub,and,or).
- Immediate Operand (iadd,shl,shr,ldm).
- Data (ldd,std).
- Stack (push,pop,call,ret,rti).
- Jump (jz, jmp).
- Special (nop,swap,reset,int).

0.16.1 One Operand Instructions

- IB[31:0] are the instruction bits.
- Inserting (1111) to Rsrc/Rdst selectors informs the register file not to output any register values.
- 'x' indicates don't care.
- 0000 at the ALUop indicates no operation.
- Rsrc1_sel is the same as Rdst1_sel.

Table 5: One Operand Instruction Control Signals Part I

Instruction	OPCode	ALUop	Rsrc1 selector	Rsrc2 selector	Rdst1 selector	Rsrc2 value
NOT	IB[31:25]	0111	0 and IB[24:22]	1111	0 and IB[24:22]	x
INC	IB[31:25]	0001	0 and IB[24:22]	1111	0 and IB[24:22]	x
DEC	IB[31:25]	0010	0 and IB[24:22]	1111	0 and IB[24:22]	x
OUT	IB[31:25]	0000	0 and IB[24:22]	1111	0 and IB[24:22]	x
IN	IB[31:25]	0000	0 and IB[24:22]	1111	0 and IB[24:22]	x

Table 6: One Operand Instruction Control Signals Part II

Instruction	OP2 selector	Rdst2 (swap)	Branch /IO Enable	SP Enable	Branch Enable	R/W Control Signal
NOT	x	1111	00	0	0	00
INC	x	1111	00	0	0	00
DEC	x	1111	00	0	0	00
OUT	x	1111	01	0	0	00
IN	x	1111	10	0	0	00

0.16.2 Two Operand Instructions

- OP2_sel: 0 the register value and 1 the imm/ea value.

Table 7: Two Operands Instruction Control Signals Part I

Instruction	OPCode	ALUop	Rsrc1 selector	Rsrc2 selector	Rdst1 selector	Rsrc2 value
ADD	IB[31:25]	0011	0 and IB[27:25]	0 and IB[24:22]	0 and IB[21:19]	x
SUB	IB[31:25]	0100	0 and IB[27:25]	0 and IB[24:22]	0 and IB[21:19]	x
AND	IB[31:25]	0101	0 and IB[27:25]	0 and IB[24:22]	0 and IB[21:19]	x
OR	IB[31:25]	0110	0 and IB[27:25]	0 and IB[24:22]	0 and IB[21:19]	x

Table 8: Two Operands Instruction Control Signals Part II

Instruction	OP2 selector	Rdst2 (swap)	Branch /IO Enable	SP Enable	Branch Enable	R/W Control Signal
ADD	0	1111	00	0	0	00
SUB	0	1111	00	0	0	00
AND	0	1111	00	0	0	00
OR	0	1111	00	0	0	00

0.16.3 Immediate Operand Instructions

- Rsrc1_sel is the same as Rdst1_sel, in SHL and SHR cases. However, in IADD case, it's a different register and in LDM case, there's no need for Rsrc, it's just a destination.
- In IADD case, Rsrc != Rdst.
- In LDM case, there's no Rsrc, it's Rdst.
- Rsrc2_val is the immediate value extracted from the IF/ID buffer.

- R/W memory (11) is write and (10) is read.
- Sign extend unit is used to adjust the (16 bits) immediate value to (32 bits).
- SE: sign extend enable (0/1).

Table 9: Immediate Operand Instruction Control Signals Part I

Instruction	OPCode	ALUop	Rsrc1 selector	Rsrc2 selector	Rdst1 selector	Rsrc2 value
IADD	IB[31:25]	0011	0 and IB[27:25]	1111	0 and IB[24:22]	0XSE and IB[15:0]
SHL	IB[31:25]	1000	0 and IB[27:25]	1111	0 and IB[27:25]	0XSE and IB[15:0]
SHR	IB[31:25]	1001	0 and IB[27:25]	1111	0 and IB[27:25]	0XSE and IB[15:0]
LDM	IB[31:25]	0000	1111	1111	0 and IB[27:25]	0XSE and IB[15:0]

Table 10: Immediate Operand Instruction Control Signals Part II

Instruction	OP2 selector	Rdst2 (swap)	Branch /IO Enable	SP Enable	Branch Enable	R/W Control Signal
IADD	1	1111	00	0	0	00
SHL	1	1111	00	0	0	00
SHR	1	1111	00	0	0	00
LDM	1	1111	00	0	0	11

0.16.4 Data Instructions

Note that:

- Effective address does not need a sign extend, that's why it's always zero extended with only 12 bits.
- OP2_sel is 1 to pass the EA.
- R/W memory (11) is write and (10) is read.

Table 11: Data Instruction Control Signals Part I

Instruction	OPCode	ALUop	Rsrc1 selector	Rsrc2 selector	Rdst1 selector	Rsrc2 val
LDD	IB[31:25]	0000	0 and IB[27:25]	1111	1111	0x000 and IB[19:0]
STD	IB[31:25]	0000	1111	1111	0 and IB[27:25]	0x000 and IB[19:0]

Table 12: Data Instruction Control Signals Part II

Instruction	OP2 selector	Rdst2 (swap)	Branch /IO Enable	SP Enable	Branch Enable	R/W Control Signal
LDD	1	1111	00	0	0	10
STD	1	1111	00	0	0	11

0.17 Stack Instructions

- Rsrc2_val is the stack pointer, as it's the address of the operation.
- ALUop's Inc2 and Dec2 are used to manipulate the stack pointer, thus the output of the ALU will be the new stack pointer.
- In case of Call, Rsrc1_sel is none, as no register is used. It is the PC pushed at the memory.
- In case of Call, Rdst1_sel, is the register holding the new address.

- In case of Ret and Rti, no registers are affected, as the PC is updated at the fetch stage.
- R/W memory (11) is write and (10) is read.

Table 13: Stack Instruction Control Signals Part I

Instruction	OPCode	ALUop	Rsrc1 selector	Rsrc2 selector	Rdst1 selector	Rsrc2 val
PUSH	IB[31:25]	1011	0 and IB[27:25]	1111	1111	SP(32 bits)
POP	IB[31:25]	1010	1111	1111	0 and IB[27:25]	SP(32 bits)
CALL	IB[31:25]	1011	1111	1111	0 and IB[27:25]	SP(32 bits)
RET	IB[31:25]	1010	1111	1111	1111	SP(32 bits)
RTI	IB[31:25]	1100	1111	1111	1111	SP(32 bits)

Table 14: Stacks Instruction Control Signals Part II

Instruction	OP2 selector	Rdst2 (swap)	Branch /IO Enable	SP Enable	Branch Enable (JZ)	R/W Control Signal
PUSH	1	1111	00	1	0	11
POP	1	1111	00	0	0	10
CALL	1	1111	00	0	0	11
RET	1	1111	00	0	0	10
RTI	1	1111	00	0	0	10

0.18 Jump Instructions

- Rsrc1_sel is the address we are jumping to, that's why we need to verify that our prediction at the JZ case is correct.
- Branch/IO Enable is (11) as it is a branching instruction.

- Branch enable (1) to detect if the JZ operated correctly.

Table 15: Jumpers Instruction Control Signals Part I

Instruction	OPCode	ALUop	Rsrc1 selector	Rsrc2 selector	Rdst1 selector	Rsrc2 val
JMP	IB[31:25]	0000	1111	1111	1111	x
JZ	IB[31:25]	0000	0 and IB[27:25]	1111	1111	x

Table 16: Jumpers Instruction Control Signals Part II

Instruction	OP2 selector	Rdst2 (swap)	Branch /IO Enable	SP Enable	Branch Enable (JZ)	R/W Control Signal
JMP	x	1111	11	0	0	00
JZ	x	1111	11	0	1	00

0.19 Special Instructions

There's no interrupt instruction, but there's a bit called Interrupt, sent to the Control Unit as an input to indicate an interrupt signal was triggered.

Table 17: Specials Instruction Control Signals Part I

Instruction	OPCode	ALUop	Rsrc1 selector	Rsrc2 selector	Rdst1 selector	Rsrc2 val
NOP	IB[31:25]	0000	1111	1111	1111	x
SWAP	IB[31:25]	0000	0 and IB[27:25]	0 and IB[24:22]	0 and IB[24:22]	x
Reset	IB[31:25]	0000	1111	1111	1111	x
Int	IB[31:25]	0000	1111	1111	1111	x

Table 18: Specials Instruction Control Signals Part II

Instruction	OP2 selector	Rdst2 (swap)	Branch /IO Enable	SP Enable	Branch Enable (JZ)	R/W Control Signals
NOP	x	1111	00	0	0	00
SWAP	0	0 and IB[27:25]	00	0	0	11
Reset	x	1111	00	0	0	00
Int	x	1111	00	0	0	00

Part V

Pipeline Stages

0.20 Overview

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

0.20.1 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.
- Reads from register file in the second half of cycle.

0.20.2 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.
- Reads from register file in the second half of cycle.

0.20.3 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.

0.20.4 Memory Stage

- Responsible for Data Memory IO.

0.20.5 Write-Back Stage

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

0.21 Intermediate Buffers

0.21.1 IF/ID Buffer

Registers

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

Control Signals

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

0.21.2 ID/EX Buffer

Registers

- Operand Registers ($2X32$ bits)
- Destination Register (4 bits)
- OpCode Register (7 bits)

- R/W Register (2 bits)
- Interrupt Register (1 bit)

Control Signals

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

0.21.3 EX/M Buffer

Registers

- ALUout Register (32 bits)
- MEM IN Register (32 bits)
- Opcode Register (7 bits)
- Destination Register (4 bits)
- R/W Register (2 bits)
- Interrupt Register (1 bit)

Control Signals

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

0.21.4 M/WB Buffer

Registers

- ALUout Register (32 bits)
- MEM OUT (32 bits)
- OpCode (7 bits)

- Destination Register (4 bits)
- Interrupt Register (1 bit)

0.22 Special Workflows

0.22.1 CALL Workflow

- Rdest value is loaded in fetch stage (like branches) and stored in PC.
- The current value of PC is propagated through the pipe, until it reaches the memory stage, where it's stored in data memory.

0.22.2 RET Workflow

- Compiler inserts 3 NOPs after each RET instruction to avoid any hazards.
- Once the RET operation reaches the memory stage it loads the PC value from stack (like a normal POP) and uses PC Navigator to write it to the PC.
- **NOTE:** Data hazards related to SP are handled normally through hazard detection unit.

0.22.3 Interrupt Workflow

- Interrupt signal is passed to the PC Control Unit and IF/ID Buffer, the fetch stage is stalled for two cycles to fetch the interrupt address and the Interrupt Bit propagates through the whole pipe, until it reaches the memory stage.
- In the memory stage, the interrupt stalls the pipe one cycles to be able to push both PC and CCR into stack.

0.22.4 RTI Workflow

- Compiler inserts 3 NOPs after each RTI instruction to avoid any hazards.

- Once the RTI operation reaches the memory stage it loads the PC value from stack (like a normal POP) and uses PC Navigator to write it to the PC.
- However, RTI stalls the pipe for one cycle to be able to load CCR, too.
- **NOTE:** Data hazards related to SP are handled normally through hazard detection unit.

Part VI

Pipeline Hazards and solutions

0.23 Structural Hazards

0.23.1 Detection

The structural hazard occurs in data memory and register file.

0.23.2 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 register reading from decode and fetch to happen in the second half.

0.24 Data Hazards

Figure 7 shows the hazard detection unit.

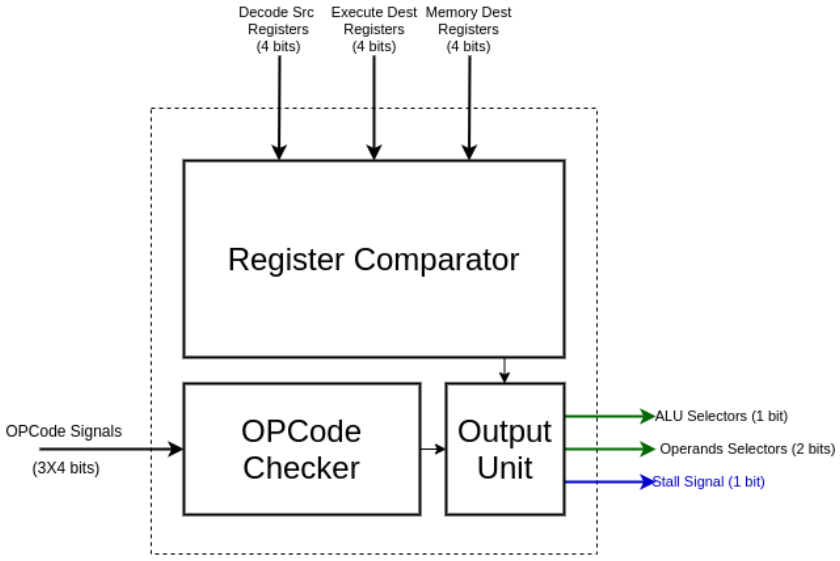

Figure 6: Hazard Detection Unit Diagram

0.24.1 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. 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. Also, it compares the execute source registers with the destination registers of the memory stage.
- **Output Unit:** outputs stall signal in case of load and pop instructions (considering the branch special case). Also, it outputs ALU and decode operands selectors.

0.24.2 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.

0.25 Control Hazards

0.25.1 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.

0.25.2 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.

0.26 Structural Hazards

0.26.1 Detection

The structural hazard occurs in data memory and register file.

0.26.2 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 register reading from decode and fetch to happen in the second half.

0.27 Data Hazards

Figure 7 shows the hazard detection unit.

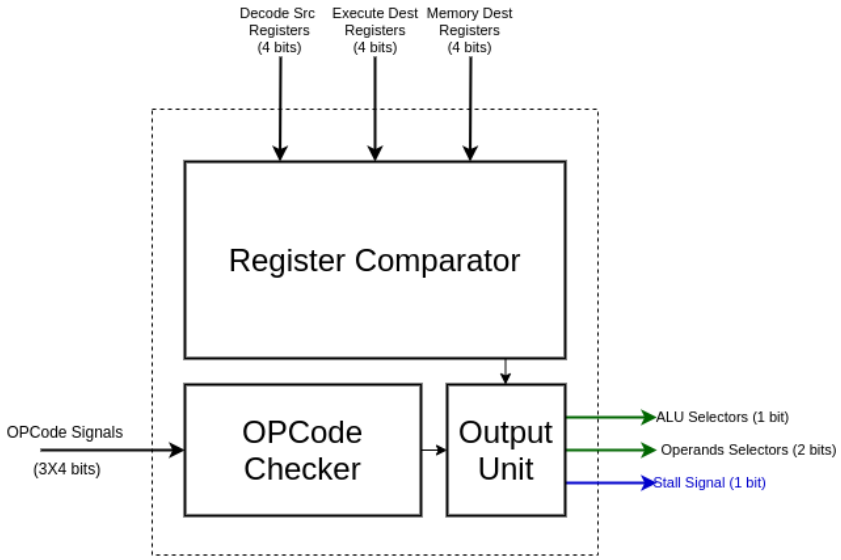

Figure 7: Hazard Detection Unit Diagram

0.27.1 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. 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. Also, it compares the execute source registers with the destination registers of the memory stage.
- **Output Unit:** outputs stall signal in case of load and pop instructions (considering the branch special case). Also, it outputs ALU and decode operands selectors.

0.27.2 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.

0.28 Control Hazards

0.28.1 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.

0.28.2 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.