Arrow

Description automatically generated with low confidence

Birzeit University

Computer Systems Engineering

COMPUTER ARCHITECTURE

ENCS4370

Pipelined Processor

Prepared by :

Basil Mari 1191027

Islam Jihad 1191375

Yousra Sheikh Qasim 1192131

Instructor : Ayman Hroub

Date : June 10, 2022

Section : 3

**1 - Abstract:**

In this project, we have designed a 24-bit pipelined processor and the tool that was used is Logisim simulator.

The RISC processor has 3 different types that support 24 bits instructions ( R-type, I-type and J-type ) with different addressing modes, also eight 24-bit general-purpose registers: R0 through R7 and 24-bits program counter register were implemented.

The five stages of the pipelined processor were implemented and tested successfully.

Table of Contents

[Theory: 7](#_Toc105964415)

[Datapath: 7](#_Toc105964416)

[Pipelining Architecture: 7](#_Toc105964417)

[Project background: 7](#_Toc105964418)

[Design and Implementation 9](#_Toc105964419)

[Components 9](#_Toc105964420)

[Register file 9](#_Toc105964421)

[Instruction Memory 9](#_Toc105964422)

[ALU 10](#_Toc105964423)

[Next PC 12](#_Toc105964424)

[Data Memory 12](#_Toc105964425)

[INC PC 13](#_Toc105964426)

[IF/ID Buffer 14](#_Toc105964427)

[Extender 17 15](#_Toc105964428)

[Extender 10 16](#_Toc105964429)

[ID/EX Buffer 16](#_Toc105964430)

[EX/MEM 18](#_Toc105964431)

[MEM/WB 19](#_Toc105964432)

[Forwarding Units 20](#_Toc105964433)

[Hazard Detection 22](#_Toc105964434)

[Lws or Lw 27](#_Toc105964435)

[Pipeline Circuit 27](#_Toc105964436)

[Simulation and Testing 28](#_Toc105964437)

[Conclusion 37](#_Toc105964438)

[References 38](#_Toc105964439)

Table of figures

[Figure 1 Register file 9](#_Toc105964480)

[Figure 2 Instruction Memory 10](#_Toc105964481)

[Figure 3 ALU 10](#_Toc105964482)

[Figure 4 Next PC 12](#_Toc105964483)

[Figure 5 Data Memory 13](#_Toc105964484)

[Figure 6 INC PC 14](#_Toc105964485)

[Figure 7 IF/ID Buffer 15](#_Toc105964486)

[Figure 8 Extender 17 15](#_Toc105964487)

[Figure 9 Extender 10 16](#_Toc105964488)

[Figure 10 ID/EX Buffer 17](#_Toc105964489)

[Figure 11 EX/MEM 18](#_Toc105964490)

[Figure 12 MEM/WB 19](#_Toc105964491)

[Figure 13 Forwarding Unit 20](#_Toc105964492)

[Figure 14 Forwarding MEM to EX 20](#_Toc105964493)

[Figure 15 Forwarding WB to EX 20](#_Toc105964494)

[Figure 16 Forwarding WB to MEM 21](#_Toc105964495)

[Figure 17 Forwarding EX to ID 21](#_Toc105964496)

[Figure 18 Forwarding MEM to ID 22](#_Toc105964497)

[Figure 19 Hazard Detection 25](#_Toc105964498)

[Figure 20 Control Unit 26](#_Toc105964499)

[Figure 21 Lws or Lw 27](#_Toc105964500)

[Figure 22 Pipeline Circuit 28](#_Toc105964501)

[Figure 23 Testing 1 29](#_Toc105964502)

[Figure 24 Testing 2 30](#_Toc105964503)

[Figure 25 Testing 3 31](#_Toc105964504)

[Figure 26 Testing 4 32](#_Toc105964505)

[Figure 27 Testing 5 33](#_Toc105964506)

[Figure 28 Testing 6 34](#_Toc105964507)

[Figure 29 Testing 7 35](#_Toc105964508)

[Figure 30 Testing 8 36](#_Toc105964509)

[Figure 31 Testing 9 37](#_Toc105964510)

Table of tables

[Table 1 Instruction Encoding R-Type 8](#_Toc105964511)

[Table 2 Instruction Encoding I-type and J-Tpe 8](#_Toc105964512)

[Table 3 ALU truth table 11](#_Toc105964513)

[Table 4 Control unit truth table 27](#_Toc105964514)

# Theory:

The data path is a basic component if we want to build any computer.

## Datapath:

Is a set of functional units that carry out data processing operations. Data paths, with a control unit, make up the CPU (central processing unit) of a computer system. A larger data path can also be created by joining more than one together using multiplexers.

## Pipelining Architecture:

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages relate to one another to form a pipe like structure. Instructions enter from one end and exit from another end. Pipelining increases the overall instruction throughput.

# Project background:

The aim of this project is to implement and design a 24-bit pipelined processor. The five stages were constructed like the pipeline presented in the class lectures.

Pipeline registers between stages were added. The control logic to detect data dependencies among instructions was designed and the forwarding logic was implemented.

For branch and jump instructions, the delay to one cycle only was reduced. The pipeline for one clock cycle after a jump or a taken branch instruction was stalled.

The data path was designed to support the 3 different types of the instructions ( R-type, I-type and J-type ).

Table

Description automatically generated

Table Instruction Encoding R-Type

Table

Description automatically generated

Table Instruction Encoding I-type and J-Tpe

# Design and Implementation

## Components

### Register file

Register File consists of 8 × 24-bit registers

BusA and BusB are 24-bit output buses for reading and writing two registers, respectively. BusW is a 24-bit input bus for writing a register when RegWrite is 1.

In a cycle, two registers are read, and one is written.

Diagram, schematic

Description automatically generated

Figure 1 Register file

### Instruction Memory

Because Datapath does not write instructions and instead acts as combinational logic for reads, instruction memory just needs to allow read access. After the access time, the address chooses the instruction.

Diagram

Description automatically generated

Figure 2 Instruction Memory

### ALU

An arithmetic logic unit (ALU) is a digital combinational circuit that performs arithmetic and bitwise operations on integer binary integers in computing. A floating-point unit, on the other hand, works with floating-point numbers. In our CPU it will get 2 inputs of 24 bits and one output of 24 bits with ALU control signal to choose which operation to perform and some flags like ZERO flag.

Diagram, schematic

Description automatically generated

Figure 3 ALU

Table

Description automatically generated

Table ALU truth table

Following are the logic equations for the ALU operations :

1. AND = OP(0000).Funct(111) + OP(0001).

2. OR = OP(0000).Funct(110) + Op(0010).

3. MAX = OP(0000).Funct(101)

4. ADD = OP(0000).Funct(100) + OP(0000).Funct(011) + OP(0011) + OP(0100) + OP(0110) + OP(0111) + OP(1000).

5. SUB = OP(0000).Funct(010) + OP(1001)

6. SLT = OP(0000).Funct(001)

7. Shift = OP(1100)

### Next PC

It’s a circuit to determine and calculate the next program counter for the processer, and making aware of some conditions like Branch if not equal or jump conditions.

Diagram, schematic

Description automatically generated

Figure 4 Next PC

### Data Memory

Data memory where data is stored. It’s used for load and store

MemRead: enables output on Data\_out

Address selects the word to put on Data\_out

MemWrite: enables writing of Data\_in

Address selects the memory word to be written

The Clock synchronizes the write operation

Diagram

Description automatically generated

Figure 5 Data Memory

### INC PC

By employing a D-Flip-Flop that transfers the value at its input to its output at the rising edge of the clock, this component is responsible for incrementing the PC by 2 at each clock cycle. A wire from its output is labeled (PCToInstMem) as an output of this component, and another wire from its output is linked to an adder that adds 2 to the input, yielding IncPC, which contains the address of the next instruction to be fetched. The INC PC circuit is shown below, which has been implemented in Lgisim.Diagram

Description automatically generated

Figure 6 INC PC

### IF/ID Buffer

We require a buffer between each two stages for pipelining purposes, thus we have four buffers: IF/ID, ID/EX, EX/MEM, and EM/WB. All of them are implemented in the same way, with registers for each value to be put in the buffer, and they're all triggered on the rising edge of the clock. The signals at each buffer are illustrated below:

IF/ID: There are seven signals: funct, Rd, Rt, Rs, all of which are 3-bits, OP, which is 4-bits, imm\_10, which is 10-bits, and im\_17, which is 17-bits, all of which were obtained by utilizing splitters from the Fetched instruction. The final signal is the IncPC, which is a 24-bit signal that is acquired from the INCPC resulting by the INC PC component.

Diagram, schematic

Description automatically generated

Figure 7 IF/ID Buffer

### Extender 17

This unit take 17 bits as input and extend them to 24 bits, and they are signed

Diagram

Description automatically generated

Figure 8 Extender 17

### Extender 10

This unit take 10 bits as input and extend them to 24 bits, whether they are signed or unsigned. And the mux is to choose between signed or not.

Diagram

Description automatically generated

Figure 9 Extender 10

### ID/EX Buffer

ID/EX: We have 20 signals, including BUSA and BUSB of 16 bits, which represent the values of Reg(RS) and Reg(Rt) obtained from either the register file or the forwarding units, as well as imm IType and imm JType of 16 bits, which are obtained from the Extender 6 and Extender 12, respectively, as well as Rt, Rd, Rs, OP, funct, and IncPC signals obtained from the first bufferChart, diagram

Description automatically generated

Figure 10 ID/EX Buffer

### EX/MEM

EX/MEM: The following 11 signals are obtained from the second buffer: IncPC, BUSB, MemToReg, RegWrite, MemRead, MemWrite, Rt, Lb, LBu, as well as the ALUResult signal, which contains the address of the destination to write the result on, and the RW signal, which contains the address of the destination to write the result on.

Diagram, schematic

Description automatically generated

Figure 11 EX/MEM

### MEM/WB

MEM/WB: The ALUResult, IncPC, MemToreg, RegWrite, and RW signals from the previous buffer, as well as the MemoryData signal from the Memory stage, are all retrieved from the previous buffer.

Diagram, schematic

Description automatically generated

Figure 12 MEM/WB

### Forwarding Units

All forwarded units that have been implemented should be indicated in this section. When a stall is required, a forwarding unit is used to optimize the pipelined route, since certain processes need results from previously unfinished actions. The following forwarding unit is used in our datapath:

Diagram, engineering drawing, schematic

Description automatically generated

Figure 13 Forwarding Unit

#### Forwarding MEM to EX

When there is a read after write hazard between the instruction in the execute stage and the instruction in the memory stage, this unit is used to convey the result of the ALU provided in the memory stage to the execution stage. The rationale is as follows:

If (EX/MEM.RegWrite

and (EX/MEM.RegisterRd != 0)

and (EX/MEM.RegisterRd = ID/EX.RegisterRs))

then MEM\_to\_EX\_ALU1 = 1

If (EX/MEM.RegWrite

and (EX/MEM.RegisterRd != 0)

and (EX/MEM.RegisterRd = ID/EX.RegisterRt))

then MEM\_to\_EX\_ALU2 = 1

Then, the two produced signals are used as selection signals in the two multiplexers used in deciding the ALU inputs. Following is the implemented circuit for the unit.

Diagram

Description automatically generated

Figure 14 Forwarding MEM to EX

#### Forwarding WB to EX

When a dependence is discovered, this unit sends the result to be stored in the register file in the write-back stage to the execution stage:

If (MEM/WB.RegWrite

and (MEM/WB.RegisterRd != 0)

and ( EX/MEM.RegisterRd != ID/EX.RegisterRs )

and (MEM/WB.RegisterRd = ID/EX.RegisterRs))

then WB\_to\_EX\_ALU1 = 1

If (MEM/WB.RegWrite

and (MEM/WB.RegisterRd != 0)

and ( EX/MEM.RegisterRd != ID/EX.RegisterRt )

and (MEM/WB.RegisterRd = ID/EX.RegisterRt))

then WB\_to\_EX\_ALU2 = 1

Following is the implemented circuit for the unit.

Diagram

Description automatically generated

Figure 15 Forwarding WB to EX

#### Forwarding WB to MEM

When there is a read after write hazard between the instruction in the memory stage and the instruction in the write back stage, this unit seeks to transmit the final result in the Write Back stage, to be recorded into the register file, to the memory stage. The rationale is as follows:

If (MEM/WB.RegWrite

and (MEM/WB.RegisterRd != 0)

and (MEM/WB.RegisterRd = EX/MEM.RegisterRt))

then WB\_to\_MEM = 1

Then, the produced signal is used as a selection signal in the multiplexer used before the Data port of the Data\_Memory

Following is the implemented circuit for the unit.

Diagram

Description automatically generated

Figure 16 Forwarding WB to MEM

#### Forwarding EX to ID

If the following logic detects a dependence, this unit transfers the result of the ALU in the execution stage to the decode stage:

If ( ID/EX.RegWrite and (ID/EX.RegisterRd != 0)

and (ID/EX.RegisterRd = IF/ID.RegisterRs)

and (not ID/EX.MEMRead)) then ALU\_to\_ID\_RS = 1

If ( ID/EX.RegWrite

and (ID/EX.RegisterRd != 0)

and (ID/EX.RegisterRd = IF/ID.RegisterRt)

and (not ID/EX.MEMRead))

then ALU\_ALURes\_to\_ID\_RT = 1

Following is the implemented circuit for the unit.

Diagram

Description automatically generated

Figure 17 Forwarding EX to ID

#### Forwarding MEM to ID

The first sends the loaded value from memory in the memory stage to the decode stage when a read after write dependence between the instruction in the decode stage and the one in the memory stage is identified. This forwarding unit is mostly concerned with a load instruction, then any instruction, and finally an instruction that reads the loaded value. The following reasoning is used to determine the presence of a hazard:

If ( EX/MEM.RegWrite

and (EX/MEM.MEMRead)

and (EX/MEM.RegisterRd != 0)

and (ID/EX.RegisterRd != IF/ID.RegisterRs)

and (EX/Mem.RegisterRd = IF/ID.RegisterRs ))

then MEM\_to\_ID\_BusA = 1

If ( EX/MEM.RegWrite

and (EX/MEM.MEMRead)

and (EX/MEM.RegisterRd != 0)

and (ID/EX.RegisterRd != IF/ID.RegisterRt)

and (EX/MEM.RegisterRd = IF/ID.RegisterRt))

then MEM\_to\_ID\_BusB = 1

Following is the implemented circuit for the unit.

Diagram, schematic

Description automatically generated

Figure 18 Forwarding MEM to ID

### Hazard Detection

#### Hazard detection unit

This unit was created to determine when it was necessary to stall the pipeline for one cycle, i.e. when the forwarding fails to transmit the data at the appropriate moment, and by examining all of the instances, the forwarding fails and a stall cycle is required in the two circumstances below.

1. When a load instruction is followed by an instruction that is dependent on the data returned by the load instruction (R/W dependency), for example:
2. Load followed by an R-type or a Branch instruction that is dependent on the data returned by the load instruction (RS or Rt in the R-type or the Branch instruction need Rd from the load instruction).
3. Load followed by a store or JR instruction that uses the loaded data to determine the address of the required memory location in the case of a Store instruction, or to update the value of the PC in the case of a JR instruction (dependency between Rs in the store or JR instruction and Rt in the load instruction).
4. When the PC has to be updated due to J, JAL, JR, and BEQ commands.

**How to detect the stall cycles:**

1. The following logic is used to identify the stall cycle when a load instruction is followed by an instruction that relies on it (as shown above):

If (ID/EX.MEMRead and (ID/EX.RegisterRD != 0)

and (ID/EX.RegisterRd = IF/ID.RegisterRs

or (not IF/ID.MemWrite and ID/EX.RegisterRd = IF/ID.RegisterRt)))

1. In the event of a missprediction in BEQ, J, JAL, or JR, the following logic is used to identify the stall cycle:

if( PCSrc and

PC1 != PC2)

where PC1 is the one generated from the Next\_PC Component and the second is the IncPC.

**What we need to do to stall the pipelining:**

1. For the first type of stall (R/W dependency due to the Load instruction), we must disable the INC PC component and the IF/ID buffer using the signal generated by the detection logic shown above, as well as clear (flush) the second buffer for the first half cycle of the next cycle (after detecting the stall) by using a D-FlipFlop triggering at the falling edge and connecting its output to a and gate with the clock signal, so the resulted signal will be zero
2. For the second type of stall, we need to flush the first buffer (IF/ID buffer) when we detect the stall to prevent it from passing to the decode stage and processing the wrong instruction, and the way we control the stall in this case is by using a D-FlipFlop triggering at the falling edge of the clock to pass the signal generated from the detected logic of the second type, and anding its output with the not of the clock, so the flush for the first buffer is done in this way.

Note that flushing (clearing) means passing a nop instruction into the decoding stage (which includes setting all the bits in the instruction to be passed into the decoding stage) because clearing the buffer by resetting it results in a JR instruction and returns the program execution to the first instruction.

Diagram, schematic

Description automatically generated

Figure 19 Hazard Detection

#### Control Unit

The Main Control Unit is a core unit in the implemented datapath, and it is in charge of generating various signals that ensure proper coordination of all employed multiplexers, inputs to various units, and some of the work performed by various units. It takes the instruction's opcode and function fields as input and then goes about doing its job. The generated signals are listed in the truth table for the Control unit.

Following is the implemented Control unit after having the truth table and the expression for each signal ready.

Diagram, schematic

Description automatically generated

Figure 20 Control Unit

Control unit truth table:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| OP |  | ALUctrl | Regdest | ExtOp | RegWrite | ALUsrc | Beq | J | JAL | JR | LUI | MR | MW |
| AND | 0000 | 000 | 11 | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| COS | 0001 | 010 | 11 | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LWS | 0010 | 011 | 11 | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ADD | 0011 | 011 | 11 | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SUB | 0100 | 100 | 11 | X | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| CMP | 0101 | 101 | X | X | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JR | 0110 | X | X | X | 0 | X | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| ANDI | 0111 | 000 | 10 | X | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| ADDI | 1000 | 011 | 10 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LW | 1001 | 011 | 10 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| SW | 1010 | 011 | X | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| BEQ | 1011 | 100 | X | X | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| J | 1100 | X | X | X | 0 | X | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| JAL | 1101 | X | 01 | X | 1 | X | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| LUI | 1110 | 110 | 00 | X | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Table Control unit truth table

**Expiration signal:**

Regdest [0] = ¬(Jal x LUI)

[0] = (a`b`)+ bc`

And we calculated it for all other labels

BEQ = BEQ

J = J

JAL = JAL

LUI = LUI

LBU = LBU

LB = LB

### Lws or Lw

Shape, rectangle

Description automatically generated

Figure 21 Lws or Lw

### Pipeline Circuit

Pipelining is a method of executing many instructions at the same time. The pipeline is split into stages, each of which is joined to the next to create a pipe-like structure. Instructions come in from one end and leave from the other. Pipelining improves the total flow of instructions.

Diagram, schematic

Description automatically generated

Figure 22 Pipeline Circuit

# Simulation and Testing

We will test a case that contains many functions which are: addi, sub, beq, lw, J and the code is:

lw r1, 5

lw r2,1

loop:

beq r2, 5, exit

sub r1,r1,1

addi r2,r2,1

j loop

exit

And after translate it to Logisim we got this:

v2.0 raw

100405

100801

162002

082880

19fffd

So to follow the clocks of the machine we’ll look in every cycle.

Diagram, schematic

Description automatically generated

Figure Testing 1

lw r1, 5

At first clock the first line has been fetched to R1 by integer 5 as seen.

Diagram, schematic

Description automatically generated

Figure Testing 2

lw r2,1

At second clock the second line has been fetched to R2 by integer 1 as seen.

Diagram, schematic

Description automatically generated

Figure Testing 3

beq r2, 5, exit

In 3rd clock, branch instruction has been fetched and the previous 2 instruction are in execution and decode levels

Diagram, schematic

Description automatically generated

Figure Testing 4

In the 4th clock the **sub r1,r1,1** instruction is beingas seen 5 – 1 = 4 in ALU result when ALU first input = 5 and second input =1 and the operation is subtract.

Diagram, schematic

Description automatically generated

Figure Testing 5

In 5th cycle the answer were sent to memory after it finished the ALU operation as seen

Diagram, schematic

Description automatically generated

Figure Testing 6

In 6th clock cycle the result has been transferred from memory to memory write back

Diagram, schematic

Description automatically generated

Figure Testing 7

addi r2,r2,1

in this clock cycle it added 1 to R2 and put the result as 2 on the ALU result.

Diagram, schematic

Description automatically generated

Figure Testing 8

In the next clock cycle it repeated the subtract instruction as 4 – 1 = 3 and repeating this depends on the loop iterations.

Diagram, schematic

Description automatically generated

Figure Testing 9

After finishing the loop the beq instruction will work and branch the loop and as seen the next PC is jumped to the label and not 4 bits as usual

# Conclusion

We learnt a lot about pipelined structures and how to apply them in this project. We successfully implemented a pipelined 24-bit processor using Logisim simulator, achieving all the project's objectives, including implementing the forwarding technique and obtaining the fewest possible stall cycles, which is 1 stall cycle in the case of load followed by an instruction that relies on it (R/W) or a miss prediction in the case of a branch or jump instruction. As a long-term goal, this project motivates us

to learn more about computer architecture methodologies and how to improve the current designs.

# References

Doctor slides and contacting him sometimes

geeks for geeks

stack overflow