**A computer chip with many wires

Description automatically generated**

**Five Stage Pipelined Processor**

**Prepared By**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Section** | **BN** | **ID** |
| **Amr Magdy** | **1** | **30** | **9210780** |
| **Fares Atef** | **1** | **33** | **9210797** |
| **Ghaith Mohamed** | **1** | **32** | **9213006** |
| **Amr Mohamed** | **1** | **31** | **9210781** |

**December 5, 2023**

**Phase 1**

**Version 3**

Contents

[Introduction 2](#_Toc152797745)

[Instruction Format/ Op Codes 2](#_Toc152797746)

[Instruction Bits Details 3](#_Toc152797747)

[Control Signal Table 3](#_Toc152797748)

[Schematic Diagram of The Processor 4](#_Toc152797749)

[Pipeline Stages Design 5](#_Toc152797750)

[Pipeline Hazards and Solutions 7](#_Toc152797751)

[VHDL Code 7](#_Toc152797752)

[Notes about Our Code / Design: 8](#_Toc152797753)

[References 8](#_Toc152797754)

# Introduction

The processor in this project has a RISC-like instruction set architecture. There are eight 4-byte general purpose registers; R0, till R7. Another two specific registers, one works as a program counter (PC). And the other works as a stack pointer (SP).

# Instruction Format/ Op Codes

|  |  |
| --- | --- |
| Instruction | Op Code |
| NOP | 00000 |
| NOT Rdst | 10000 |
| NEG Rdst | 10001 |
| INC Rdst | 10010 |
| DEC Rdst | 10011 |
| CMP Rsrc1, Rsrc2 | 10101 |
| IN Rdst | 00001 |
| OUT Rdst | 10100 |
| SWAP Rsrc, Rdst | 00010 |
| ADD Rdst, Rsrc1, Rsrc2 | 00011 |
| SUB Rdst, Rsrc1, Rsrc2 | 00101 |
| AND Rdst, Rsrc1, Rsrc2 | 00110 |
| OR Rdst, Rsrc1, Rsrc2 | 00111 |
| XOR Rdst, Rsrc1, Rsrc2 | 01000 |
| ADDI Rdst, Rsrc1, Imm | 00100 |
| BITSET Rdst, Imm | 10110 |
| RCL Rsrc, Imm | 10111 |
| RCR Rsrc, Imm | 11000 |
| LDM Rdst, Imm | 01010 |
| LDD Rdst, EA | 01011 |
| STD Rsrc, EA | 11010 |
| PUSH Rdst | 11001 |
| POP Rdst | 01001 |
| PROTECT Rsrc | 11011 |
| FREE Rsrc | 11100 |
| JZ Rdst | 11101 |
| JMP Rdst | 11110 |
| CALL Rdst | 11111 |
| RET | 01100 |
| RTI | 01101 |

# Instruction Bits Details

After considering all the instructions in our instruction set, we have decided on the following design of the instruction bits:

**N.B.** In case of using immediate values, which are 16-bits long we use the **next slot** in memory for the retrieval of that value.

Also, in the case of using an effective address which is 20-bits long we use the last 4 bits in the instruction, since it is unused, along with the 16-bits available in the next memory slot to be able to deliver a 20-bit value for execution.

# Control Signal Table

For a better view, we have attached an Excel spreadsheet for the table showing all the control signals VS. instructions here:



A table with numbers and letters

Description automatically generated

**Fig1: Control Signal Table**

# Schematic Diagram of The Processor

Like the control signal table, we have a link for a better view of our design attached here:

[**Five-Stage Processor Design**](https://www.canva.com/design/DAFzjgNgY9I/crOsA9dvig85tfbw5jPZ0A/edit?utm_content=DAFzjgNgY9I&utm_campaign=designshare&utm_medium=link2&utm_source=sharebutton)

**A diagram of a machine

Description automatically generated**

**Fig2: Schematic Design of The Processor**

# Pipeline Stages Design

To achieve a five-stage pipeline design, we have used four large to keep the all the data of each stage separate from all the other stages.

These registers are:

**IF/ID:** This is the register between the fetch and decode stages. **Size: 81-bits** in total split into:

* INT Signal**:** 1-bit
* PC+1**:** 32-bit
* Instruction**:** 16 bit
* Immediate Value:32 bit.

**ID/EX:** This is the register between the decode and execute stages. **Size:** 124-bitsin total split into:

* **19 1-Bit Signals** which are:   
  [ Register\_Write, Branch, Immediate, Mem\_Read, Mem\_Write  
  Mem\_2Reg, Port\_Write, Port\_Read, Protect\_Write, Protect\_Val ,Write\_Flag, Stack, Push, Call, Mem\_2PC, Swap, RTI, RST, INT, PUSH\_INT\_PC]
* **3 32-Bit** Register Valueswhich are:  
  [ Reg1\_Value, Reg2\_Value, PC+1]
* **3 3-Bit** Register Addresseswhich are:  
  [ Rdst,Rs,Rt]
* **5-Bit Op Code**

**EX/MEM:** This is the register between the execute and memory stages.  
**Size:** 124-bitsin total split into:

* **16 1-Bit Signals** which are:   
  [ Register\_Write, Branch, Mem\_Read, Mem\_Write  
  Mem\_2Reg, Port\_Write, Port\_Read, Stack, Push, Call, Mem\_2PC, RTI, RST, INT, PUSH\_INT\_PC,Protect\_State]
* **4 32-Bit Register Values** which are:  
  [ ALU Result, Memory Data, PC+1, Stack Pointer]
* **3-Bit Destination Register Address**
* **3-Bit Flag Register**

**MEM/WB:** This is the register between the memory and write-back stages.  
 **Size:** 107-bitsin total split into:

* **8 1-Bit Signals** which are:   
  [ Register\_Write, Mem\_2Reg, Port\_Read, Call, Mem\_2PC, RTI, RST, INT, PUSH\_INT\_PC]
* **3 32-Bit Register Values** which are:  
  [ ALU Result, Memory Data, Port Data]  
  **3-Bit Destination Register Address**

# Pipeline Hazards and Solutions

We face three types of hazards in our design which are:

* Structural
* Data
* Control

For **Structural** we have faced the following:

* Using the same memory for instruction fetch and storing and retrieving data  
  **For that:** We have used two different memory elements: **Data** and **Instruction**.
* Reading from and writing to the register file during the same clock cycle.  
  **For that:** We ensure that writing is done at the first half of the clock cycle and reading is done in the second.

For **Data** We have faced the following:

* **RAW Dependency  
  For that:** we have added a forwarding unit to forward data from both the ALU and Memory before it is written back to the register file.
* **Load Use Scenario  
  For that:** We stall the pipeline for one cycle and leave the forward unit to do its job.

For **Control** We have faced the following:

* **Incorrect Prediction  
  For that:** We flush the incorrectly fetched instructions.

**N.B.,** we use a static (not taken) branch prediction approach.

# VHDL Code

Our VHDL Code for the design can be found in our GitHub repository:

[GitHub-Processor RTL Design](https://github.com/FaresAtef1/5-stage-pipeline-processor.git)

# Notes about Our Code / Design:

* The destination / write-back register is always the first register referenced in the instruction bits.
* Only the pipeline register is synchronized with the clock while all the hardware in every stage is purely combinational.
* A naming convention we have used in our code is to append the name of every signal with the name of the stage in which it exists. e.g.: **Push\_EX** refers to the push control signal in the execute stage.
* To take care of timing and to avoid some unexpected behaviors we have introduced a delay to some of the signals looping back to earlier stages to ensure everything is updated correctly e.g. **Register\_Write** signal coming back to the register file should be delayed for some time to ensure data consistency in the register file.

# References

* [Computer Organization and Design -- The Hardware/Software Interface -- 5th Edition, David Patterson and John Hennessy.](http://store.elsevier.com/Computer-Organization-and-Design/David-Patterson/isbn-9780124078864/)