**SAP based Microprocessor Design**

**Comprehensive Report**

**A logo with a number in a square

Description automatically generated**

Prepared By Version

**Team 5**  **1.0**

Table of Contents

[**1** **Introduction** 4](#_Toc153158359)

[**1.1** **Background** 4](#_Toc153158360)

[**1.2** **Objectives** 4](#_Toc153158361)

[**1.3** **Importance and Applications** 5](#_Toc153158362)

[**2** **Project Team** 6](#_Toc153158363)

[**2.1** **Supervision Team** 6](#_Toc153158364)

[**3** **Project Scope and Objectives** 7](#_Toc153158365)

[**3.1** **Microprocessor Design Overview** 7](#_Toc153158366)

[**3.2** **Project Goals and Constraints** 9](#_Toc153158367)

[**4** **Project Management** 10](#_Toc153158368)

[**4.1** **Milestones** 10](#_Toc153158369)

[**4.2** **Estimated Timeline** 12](#_Toc153158370)

[**4.3** **Challenges Faced and Solutions** 13](#_Toc153158371)

[**5** **Microprocessor Architecture** 14](#_Toc153158372)

[**5.1** **Choice of Microprocessor Architecture** 14](#_Toc153158373)

[**5.2** **Block Diagram and Components** 15](#_Toc153158374)

[**6** **Instruction Set Architecture (ISA)** 16](#_Toc153158375)

[**6.1** **Definition of Instruction Set** 16](#_Toc153158376)

[**6.2** **Instruction Formats** 17](#_Toc153158377)

[**6.2.1** **SR-Type Instruction** 19](#_Toc153158378)

[**6.2.2** **DR-Type Instruction** 20](#_Toc153158379)

[**6.2.3** **I-Type Instruction** 21](#_Toc153158380)

[**6.2.4** **J-Type Instruction** 22](#_Toc153158381)

[**6.2.5** **D-Type Instruction** 23](#_Toc153158382)

[**6.2.6** **O-Type Instruction** 24](#_Toc153158383)

[**6.3** **Opcode Assignment Technique** 25](#_Toc153158384)

[**6.4** **Macroinstructions** 26](#_Toc153158385)

[**6.5** **Assembler** 27](#_Toc153158386)

[**7** **Design Methodology** 28](#_Toc153158387)

[**7.1** **High-Level Design Overview** 28](#_Toc153158388)

[**7.2** **Control Unit Design** 28](#_Toc153158389)

[**7.3** **Data Path Design** 28](#_Toc153158390)

[**7.4** **Memory Hierarchy** 28](#_Toc153158391)

[**8** **Implementation** 28](#_Toc153158392)

[**8.1** **Register Transfer Level (RTL) Description** 28](#_Toc153158393)

[**8.2** **Simulation Results** 28](#_Toc153158394)

[**8.3** **Hardware Description Language (HDL) Code Snippets** 29](#_Toc153158395)

[**9** **Testing and Verification** 29](#_Toc153158396)

[**9.1** **Test Plan** 29](#_Toc153158397)

[**9.2** **Simulation Verification** 29](#_Toc153158398)

[**9.3** **Hardware Testing Strategies** 29](#_Toc153158399)

[**10** **Results and Analysis** 29](#_Toc153158400)

[**10.1** **Evaluation of Microprocessor Performance** 29](#_Toc153158401)

[**10.2** **Comparison with Project Requirements** 29](#_Toc153158402)

[**11** **Conclusion** 30](#_Toc153158403)

[**11.1** **Summary of Achievements** 30](#_Toc153158404)

[**11.2** **Lessons Learned** 30](#_Toc153158405)

[**12** **Resources** 31](#_Toc153158406)

[**13** **Appendices** 32](#_Toc153158407)

[**13.1** **Appendix 1: Microprocessor Full Instruction Set** 32](#_Toc153158408)

# **Introduction**

## **Background**

Microprocessors stand at the forefront of digital systems and Systems on Chip (SoCs), serving as the foundational building blocks that empower the modern computing landscape. The evolution of microprocessor design has been instrumental in shaping the efficiency, speed, and versatility of digital systems. Rooted in this context, our project focuses on the creation of a 'Basic' microprocessor, drawing inspiration from the Simple As Possible (SAP)-1 architecture.

The SAP architecture, known for its simplicity and educational value, provides a solid framework for students to delve into the core principles of digital design. This project serves as a bridge between theoretical concepts and practical application, offering an immersive experience in crafting a functional microprocessor.

## **Objectives**

The primary objective of this project is to design and implement an 8-bit microprocessor, adhering rigorously to established design best practices. The microprocessor specifications mandate a minimum of 2 arithmetic operations, 2 logic operations, and one branch operation. Beyond these foundational requirements, students with advanced skills are encouraged to explore additional functionalities, provided they enhance the design without compromising the quality of the design documentation.

A specific focus lies on the control unit, requiring a detailed breakdown of various blocks and a clear articulation of the teamwork plan. The project is designed to not only cultivate technical skills but also to showcase effective team management, reflecting real-world scenarios where collaborative efforts are crucial for success.

## **Importance and Applications**

The importance of microprocessor design transcends the boundaries of theoretical knowledge, extending into practical applications that drive innovation. Microprocessors serve as the central nervous system of electronic devices, enabling functionalities ranging from simple arithmetic operations to complex computations. The successful design and implementation of a microprocessor not only contributes to advancements in digital systems but also enhances the problem-solving capabilities of the designers.

In the broader context, microprocessors find applications in diverse fields, including embedded systems, IoT devices, communication systems, and beyond. The skills acquired through this project are directly transferable to real-world scenarios, making students well-equipped for challenges in the ever-evolving landscape of digital IC design. The project, therefore, holds significance not only in its educational value but also in its practical implications for future technological advancements.

# **Project Team**

In order to design and implement the outlined microprocessor, it is essential to assemble a proficient team consisting of members with expertise in digital IC design, control unit development, RTL implementation, simulation, verification and teamwork. Our team possesses the required skill set, comprising the following members:

|  |  |  |
| --- | --- | --- |
| **Name** | **AUC ID** | **Email** |
| Omar Hesham Elshopky | V23010251 | [omar.elshopky202@gmail.com](mailto:omar.elshopky202@gmail.com) |
| Mohamed Ahmed Kamal | V23010268 |  |
| Hoda Ashraf Mohamed | V23010471 | [hodashrafff@gmail.com](mailto:hodashrafff@gmail.com) |
| Ehab Ashraf Bolis | V23010363 |  |

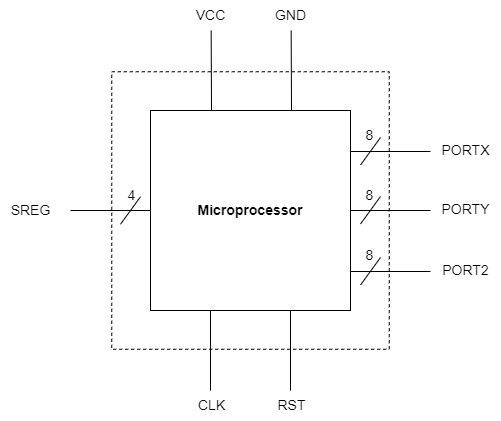
## **Supervision Team**

|  |
| --- |
| **Name** |
| Dr. Islam Yehia |
| Eng. Zeina Mohamed Samir |

# **Project Scope and Objectives**

## **Microprocessor Design Overview**

In this section, the microprocessor specifications are determined, considering the developed microprocessor as a black box tasked with performing the required functions.



*Fig. 1: A block diagram representing the input/output pins of the microprocessor as a black box.*

The RISC (Reduced Instruction Set Computing) architecture is followed for the ISA (Instruction Set Architecture) of an 8-bit microprocessor, enabling the execution of multiple arithmetic and logic operations. The microprocessor features four programmer-accessible registers, namely A, B, C, and D, which can be utilized in various operations.

The microprocessor also has three output ports that can be used to display its registers’ content to the external world, for instance, through a 7-segment display. This feature allows users to visualize the values stored in these registers during the execution of instructions.

Additionally, a dynamic stack is employed to enhance the capability of executing multiple function calls and branching. This dynamic stack facilitates efficient management of subroutine calls and branching instructions within the microprocessor.

Direct, immediate, and register-based addressing modes are supported in load and store instructions. Furthermore, the microprocessor incorporates a Status Register containing flags obtained from the operations. This register provides information about the status of the microprocessor after each operation.

A single bus is utilized following the Von Neumann Architecture. The technical specifications, including the mentioned features, are outlined in the following table:

|  |  |
| --- | --- |
| **Tech Specifications** | |
| Data Width | 8 bits |
| Clock Speed | 16 MHz |
| Memory (RAM) | 64 Kb |
| Program Memory/Data Memory | Adopting a flexible partitioning strategy guided by initialization instructions. (The low 48 KB for Program Memory and higher 16 KB for Data Memory allocated unless explicitly initialized) |
| Registers | 9x 8-bit Register File. Among them are 4 programmer-accessible registers, Status Register, and two 16-bit Stack Pointer and Program Counter. |
| Arithmetic Operations | Addition, Subtraction, Increment, Decrement, Multiplication and Division by 2 |
| Logic Operations | AND, OR, XOR, and Rotation |
| Branching Operations | Conditional and Unconditional Jump, and Call & Return |
| Stack | Dynamic Stack Size managed by a 16-bit Stack Pointer (SP) |
| Input Pins | RST (Reset Pin), CLK (Clock Pin) |
| Output Pins | Output ALU flags through a 4-bit SREG pins, and display the values of internal registers through three 8-bit {X|Y|Z}PORTs. |
| Power Consumption | X mW |
| Instruction Set Architecture | Reduced Instruction Set Computing (RISC) |
| Bus Architecture | Von Neumann Architecture |

*Table 1: The technical specifications of the microprocessor.*

## **Project Goals and Constraints**

The project aims to design and implement a microcontroller with the [specified characteristics](#_Microprocessor_Design_Overview), adhering to best practices in design, Verilog standards, and comprehensive documentation. The preferred approach is to prioritize completeness over complexity.

The SAP-based microprocessor design project concludes upon the completion of the following deliverables:

* Microprocessor design, ranging from high-level conceptualization to detailed sub-block designs.
* A programming guide outlining the instruction set in assembly, hexadecimal, and binary formats.
* Verilog implementation of the designed sub-blocks, integrated to form the desired microprocessor.
* Multiple test benches, including one for each sub-block and another for the top level to verify overall microprocessor functionality.
* An assembler developed in Python to convert assembly instructions into a binary file ready for execution.
* A demonstration video showcasing the microprocessor's verification on FPGA.
* A presentation summarizing the work done, highlighting the characteristics of the microprocessor design.

# **Project Management**

## **Milestones**

The project plan outlines specific milestones that collectively contribute to accomplishing the defined objectives and deliverables presented in the preceding section. These milestones are as follows:

1. **Define Project Objectives**

Establish the project's goals, scope, and objectives, identifying specific functionalities for the microprocessor.

1. **Select Microprocessor Architecture**

Explore various versions of SAP, conduct a thorough analysis, and select an appropriate microprocessor architecture that aligns with the project's specific requirements.

1. **Specifications Determination and Instruction Set Architecture**

Outline the microprocessor specifications, including data width, I/O signals, instruction set architecture, and register configuration.

1. **High-Level Design**

Create a comprehensive block diagram outlining major components, data paths, and control units in a high-level design.

1. **Control Unit Design**

Design the control unit along with its Finite State Machines (FSMs), responsible for managing instruction and data flow.

1. **Data Path and ALU Design**

Design the data path and incorporate the arithmetic logic unit to achieve precise manipulation of data.

1. **Memory Design**

Design the memory hierarchy components ensuring seamless interfacing and communication within the microprocessor.

1. **Control Unit Components Implementation**

Implement the FSMs and the components defined during the “Control Unit Design” milestone and perform unit testing on each component to ensure readiness for integration.

1. **Data Path Components Implementation**

Implement the data path and ALU components defined during the “Data Path and ALU Design” milestone and perform unit testing on each component to ensure readiness for integration.

1. **Memory Components Implementation**

Implement the memory components defined during the “Memory Design” milestone and perform unit testing on each component to ensure readiness for integration.

1. **Components Integration**

Integrate units into the complete microprocessor and conduct testing to verify proper communication and coordination.

1. **Timing Analysis**

Assess the timing characteristics of the microprocessor, ensuring compliance with specified constraints, involving analysis of propagation delays, setup times, and hold times for optimal performance and prevention of timing-related issues.

1. **Simulation and Verification**

Conduct simulations to validate the RTL design, analyzing microprocessor behavior under various conditions and inputs.

1. **Hardware Implementation**

Implement the microprocessor on hardware, an FPGA, conducting real-world hardware testing.

1. **Assembler Development**

Develop a crucial software component, the assembler, by designing algorithms for syntax parsing and object program generation. This facilitates the translation of assembly language programs into machine code, streamlining the programming process for improved efficiency.

1. **Documentation**

Create comprehensive documentation covering architecture, design specifics, implementation, and testing outcomes in a final report.

## **Estimated Timeline**

The project progresses through five (**5**) phases, outlined as follows:

1. **Phase 1 – Project Initiation**

In this initial phase, project objectives, scope, and deliverables are established. The optimal architecture, informed by research and a detailed review of SAP, is selected, and microprocessor specifications are meticulously set, forming a clear guide for the project's trajectory.

1. **Phase 2 – Microprocessor Architecture Design**

This phase is centered on the design of the microprocessor, progressing from a detailed block diagram to the construction of subblocks. Simultaneously, the instruction set architecture is defined, specifying opcode assignments, and addressing modes.

1. **Phase 3 – RTL Implementation & Unit Testing**

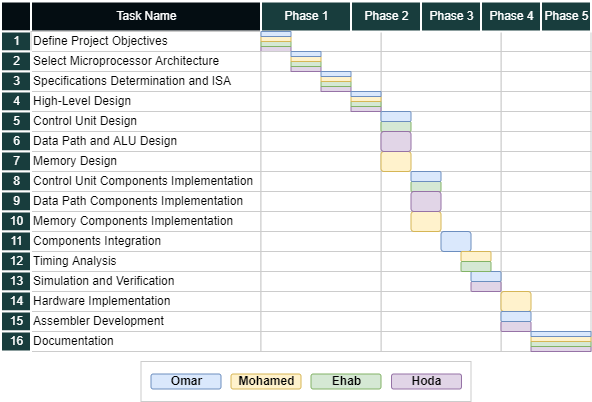
During Phase 3, the implementation of microprocessor blocks is undertaken, with rigorous unit testing conducted for each block individually before integration into one system.

1. **Phase 4 – Simulation and Verification**

During this phase, simulations are conducted to validate the microprocessor's implementation. Simultaneously, a defined testing plan is executed to ensure the functionality and correctness of the microprocessor.

1. **Phase 5 – Final Report Creation**

In the final phase, detailed documentation for the microprocessor is created by consolidating sub-documents, offering a comprehensive record of architecture, design, implementation, testing outcomes, and project lessons.

The following is the initial Gantt chart that provides a precise schedule and work plan for each milestone within the project.

## **Challenges Faced and Solutions**

Discuss any challenges encountered during the project and the corresponding solutions.

* Instruction Formats Creation and the usage of our background knowledge (subnetting)

# **Microprocessor Architecture**

## **Choice of Microprocessor Architecture**

Before embarking on the design phase and finalizing our microprocessor specifications, a comprehensive review of available microprocessor architectures was conducted. The goal was to select a base model upon which to build our microprocessor. Among the considered options were various variants of the SAP (Simple As Possible) computer, notably SAP-1, SAP-2, and SAP-3 (inspired by the Intel 8080/8085 with some instructions removed).

While SAP-1 offered a simple architecture with essential computer features, it fell short in meeting several points specified in our requirements. Progressing to SAP-2 and SAP-3, we identified advanced capabilities that could enhance our design.

SAP-2 introduced bidirectional registers, reducing wiring capacitance and the count of I/O pins. It also featured a larger memory, providing a more realistic option compared to the 16-byte memory used in SAP-1.

In SAP-3, the introduction of a dynamic stack proved advantageous for call-return applications, surpassing the two slots introduced in SAP-2. Additionally, SAP-3 offered a versatile register file, enabling programmers to reduce the number of memory-reference instructions by leveraging the provided registers in the architecture.

In light of these pivotal considerations, the microprocessor's base models were defined, and the subsequent sections elaborate on how these factors influenced our microprocessor architecture.

## **Block Diagram and Components**

Provide a high-level schematic or block diagram of the microprocessor, detailing its key components.

# **Instruction Set Architecture (ISA)**

## **Definition of Instruction Set**

The microprocessor instruction set includes **25** distinct operations that involve diverse initialization, memory-reference, register, arithmetic, logical, branching, stack, and output operations. These operations are outlined in Table 2 and discussed in detail in the subsequent sections.

Appendix 1 contains a comprehensive table detailing each operation across all registers, encompassing a total of **151** instructions.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Op Code** | **Addressing Mode** | **T states** | **Flags** | **Bytes** | **Type** | **Main Effect** |
| **Initialization Instructions** | | | | | | | |
| PRGM ***double-byte*** | 11010000 | Immediate |  | - | 3 | J | DataMemStart ← double-byte |
| **Memory-Reference Instructions** | | | | | | | |
| LDR ***Rd***, ***address*** | 111000XX | Direct |  | - | 3 | D | Rd ← Maddress |
| STR ***Rs***, ***address*** | 111001XX | Direct |  | - | 3 | D | Maddress ← Rs |
| **Register Instructions** | | | | | | | |
| MOV ***Rd***, ***Rs*** | 0100XXXX | Register |  | - | 1 | DR | Rd ← Rs |
| MVI ***Rd***, ***byte*** | 110000XX | Immediate |  | - | 2 | I | Rd ← byte |
| **Arithmetic Instructions** | | | | | | | |
| ADD ***Rd***, ***Rs*** | 0101XXXX | Register |  | ZCPS | 1 | DR | Rd ← Rd + Rs |
| SUB ***Rd***, ***Rs*** | 0110XXXX | Register |  | ZCPS | 1 | DR | Rd ← Rd - Rs |
| INR ***Rd*** | 000000XX | Register |  | Z-PS | 1 | SR | Rd ← Rd + 1 |
| DER ***Rd*** | 000001XX | Register |  | Z-PS | 1 | SR | Rd ← Rd - 1 |
| **Logical Instructions** | | | | | | | |
| ROR ***Rd*** | 000010XX | Register |  | -C-- | 1 | SR | Rd ← Rd x 2 (Rotate all right) |
| ROL ***Rd*** | 000011XX | Register |  | -C-- | 1 | SR | Rd ← Rd / 2 (Rotate all left) |
| ANR ***Rd***, ***Rs*** | 0111XXXX | Register |  | ZCPS | 1 | DR | Rd ← Rd & Rs |
| ORR ***Rd***, ***Rs*** | 1000XXXX | Register |  | ZCPS | 1 | DR | Rd ← Rd | Rs |
| XRR ***Rd***, ***Rs*** | 1001XXXX | Register |  | ZCPS | 1 | DR | Rd ← Rd ^ Rs |
| **Branching Operations** | | | | | | | |
| JMP ***address*** | 11010001 | Immediate |  | - | 3 | J | PC ← address |
| JZ ***address*** | 11010010 | Immediate |  | - | 3 | J | PC ← address if Z = 0 |
| **Stack Instructions** | | | | | | | |
| CALL ***address*** | 11010011 | Immediate |  | - | 3 | J | PC ← address |
| RET | 11110000 | - |  | - | 1 | O | PC ← return address |
| PUSH ***Rs*** | 000100XX | Register |  | - | 1 | SR | Mstack – 1 ← Rs |
| POP ***Rd*** | 000101XX | Register |  | - | 1 | SR | Rd ← Mstack |
| **Misc Instructions** | | | | | | | |
| OUTX ***Rs*** | 000110XX | Register |  | - | 1 | SR | PORTX ← Rs |
| OUTY ***Rs*** | 000111XX | Register |  | - | 1 | SR | PORTY ← Rs |
| OUTZ ***Rs*** | 001000XX | Register |  | - | 1 | SR | PORTZ ← Rs |
| NOP | 11111110 | - |  | - | 1 | O | Delay (No Operation) |
| HLT | 11111111 | - |  | - | 1 | O | Stop Processing |

*Table 2: The distinct operations composed in the instruction set.*

## **Instruction Formats**

The microprocessor relies on instructions to guide its sequential execution of tasks. These instructions must be loaded in machine code form at the outset —comprising 0s and 1s— enabling the machine to comprehend and execute them. Programmers commonly use assembly instructions like ADD, SUB, LDR, etc., which are later translated into machine code using an assembler, a software discussed in more detail in a [dedicated section](#_Assembler).

To standardize the instruction format, a generic structure is adopted for each individual instruction, as outlined below:

|  |  |
| --- | --- |
| **Opcode** | **Operand** |
| 8 bits | 8 or 16 bits |

The 8 bits allocated for the opcode allow the microprocessor to accommodate 255 different instructions. Although our design currently implements only 151 instructions, each may vary in lengths and layouts, making a random assignment of opcodes impractical.

To optimize the instruction encoding and decoding operations, specific formats should be defined to categorize the instructions into cohesive groups or types. Each group or type adheres to standardized method for encoding and decoding the instructions with similarities in length, layout, and memory addressing mode. This systematic approach not only streamlines the encoding and decoding processes but also facilitates smoother operation in the controller, particularly during the decode cycle.

The microprocessor categorizes instructions into six types:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Type** | **Instruction Layout** | **Instruction Length** | **Opcode** | **Addressing Mode** |
| [SR-Type](#_SR-Type_Instruction) | ASM R | 1 Byte | 00XXXXXX | Register |
| [DR-Type](#_DR-Type_Instruction) | ASM Rd, Rs | 1 Byte | 01XXXXXX  10XXXXXX | Register |
| [I-Type](#_I-Type_Instruction) | ASM Rd, byte | 2 Bytes | 1100XXXX | Immediate |
| [J-Type](#_J-Type_Instruction) | ASM address | 3 Bytes | 1101XXXX | Immediate |
| [D-Type](#_D-Type_Instruction) | ASM Rd, address | 3 Bytes | 1110XXXX | Direct |
| [O-Type](#_O-Type_Instruction) | ASM | 1 Byte | 1111XXXX | - |

*Table 3: The different instruction types in our microprocessor. ASM stand for Assembly Keyword.*

Further elaboration on each type is provided in the following sections.

### **SR-Type Instruction**

The Single Register type instruction typically performs a specific operation on the value stored in the designated register, adhering to the **register addressing mode** paradigm.

**Instruction Layout**

[INSTRUCTION\_KEYWORK] [REGISTER]

**Machine Code Format**

Utilize the opcode section of the generic format, excluding the operand.

|  |  |  |
| --- | --- | --- |
| **Opcode** | | |
| **Operation** | | **Register** |
| **SR-Type** | **Instruction Index** |
| 6 bits | | 2 bits |
| 00 | XXXX | XX |

**Instruction Length**

1 Byte

**Instructions**

9 Instructions: INR, DER, ROR, ROL, PUSH, POP, OUTX, OUTY, OUTZ

**Example**

INR B

|  |  |  |
| --- | --- | --- |
| **Opcode** | | |
| **Operation** | | **Register** |
| **SR-Type** | **Instruction Index** |
| 00 | 0000 | 01 |

### **DR-Type Instruction**

The Double Register type instruction typically carries out a specific operation on the values stored in the two provided registers. The result is then stored in the first register, following the **register addressing mode** paradigm.

**Instruction Layout**

[INSTRUCTION\_KEYWORK] [DESTINATION\_REGISTER] [SOURCE\_REGISTER]

**Machine Code Format**

Utilize the opcode section of the generic format, excluding the operand.

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | | | |
| **Operation** | | **Destination Register (Rd)** | **Source Register (Rs)** |
| **DR-Type** | **Instruction Index** |
| 4 bits | | 2 bits | 2 bits |
| 01 or 10 | XX | XX | XX |

**Instruction Length**

1 Byte

**Instructions**

6 Instructions: MOV, ADD, SUB, AND, ORR, XRR

**Example**

AND C, B

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | | | |
| **Operation** | | **Destination Register (Rd)** | **Source Register (Rs)** |
| **DR-Type** | **Instruction Index** |
| 01 | 11 | 10 | 01 |

### **I-Type Instruction**

The Immediate type instruction typically executes a specific operation on a specified register, with an immediate value provided as the operand. This follows **the immediate addressing mode** paradigm.

**Instruction Layout**

[INSTRUCTION\_KEYWORK] [DESTINATION\_REGISTER] [IMMIDEATE\_BYTE]

**Machine Code Format**

Utilize both the opcode and the operand sections of the generic format.

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | | | **Operand** |
| **Operation** | | **Destination Register (Rd)** | **Immediate** |
| **I-Type** | **Instruction Index** |
| 6 bits | | 2 bits | 8 bits |
| 1100 | XX | XX | XXXXXXXX |

**Instruction Length**

2 Bytes

**Instructions**

1 Instruction: MVI

**Example**

MVI D, 15H

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | | | **Operand** |
| **Operation** | | **Destination Register (Rd)** | **Immediate** |
| **I-Type** | **Instruction Index** |
| 1100 | 00 | 11 | 00011001 |

### **J-Type Instruction**

The Jump type instruction typically performs a specific operation on double-byte operand, which follows **the immediate addressing mode** paradigm.

**Instruction Layout**

[INSTRUCTION\_KEYWORK] [DOUBLE\_BYTES\_IMMIDIATE]

**Machine Code Format**

Utilize both the opcode and the operand sections of the generic format.

|  |  |  |
| --- | --- | --- |
| **Opcode** | | **Operand** |
| **Operation** | | **Immediate** |
| **J-Type** | **Instruction Index** |
| 8 bits | | 16 bits |
| 1101 | XXXX | XXXXXXXX XXXXXXXX |

**Instruction Length**

2 Bytes

**Instructions**

4 Instructions: PRGM, JMP, JZ, CALL

**Example**

JMP FF46H

|  |  |  |
| --- | --- | --- |
| **Opcode** | | **Operand** |
| **Operation** | | **Immediate** |
| **J-Type** | **Instruction Index** |
| 1101 | 0001 | 11111111 01000110 |

### **D-Type Instruction**

The Direct type instruction typically performs a specific operation on the specified register and the memory content at the provided address, which follows **the direct addressing mode** paradigm.

**Instruction Layout**

[INSTRUCTION\_KEYWORK] [REGSITER] [MEMORY\_ADDRESS]

**Machine Code Format**

Utilize both the opcode and the operand sections of the generic format.

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | | | **Operand** |
| **Operation** | | **Register** | **Address** |
| **I-Type** | **Instruction Index** |
| 6 bits | | 2 bits | 16 bits |
| 1110 | XX | XX | XXXXXXXX XXXXXXXX |

**Instruction Length**

3 Bytes

**Instructions**

2 Instructions: LDR, STR

**Example**

LDR A, F037H

|  |  |  |  |
| --- | --- | --- | --- |
| **Opcode** | | | **Operand** |
| **Operation** | | **Register** | **Address** |
| **I-Type** | **Instruction Index** |
| 1110 | 00 | 00 | 11110000 00110111 |

### **O-Type Instruction**

The Others type instruction typically perform special operations that are hardcoded into the microprocessor’s controller.

**Instruction Layout**

[INSTRUCTION\_KEYWORK]

**Machine Code Format**

Utilize the opcode section of the generic format, excluding the operand.

|  |  |
| --- | --- |
| **Opcode** | |
| **Operation** | |
| **O-Type** | **Instruction Index** |
| 8 bits | |
| 1111 | XXXX |

**Instruction Length**

1 Byte

**Instructions**

3 Instructions: RET, NOP, HLT

**Example**

HLT

|  |  |
| --- | --- |
| **Opcode** | |
| **Operation** | |
| **O-Type** | **Instruction Index** |
| 1111 | 1111 |

## **Opcode Assignment Technique**

8 bit -> 255 instructions

4 Reg -> 00 (2 bits)

|  |  |
| --- | --- |
| **Register** | **Encoding** |
| A | 00 |
| B | 01 |
| C | 10 |
| D | 11 |

SR 6 bits 9 Ins 00XXXX?? 00 (16 free)

DR 4 bits 6 Ins XXXX???? 01, 10 (8 free)

I 6 bits 1 Ins 1100XX?? 11 (4 free)

J 8 bits 4 Ins 1101XXXX 11 (16 free)

D 6 bits 2 Ins 1110XX?? 11 (4 free)

O 8 bits 3 Ins 1111XXXX 11 (16 free)

4 groups

All instructions have opcode that specifies the operations.

## **Macroinstructions**

Explain the addressing modes supported by the microprocessor.

## **Assembler**

Explain the addressing modes supported by the microprocessor.

# **Design Methodology**

## **High-Level Design Overview**

Provide an overview of the microprocessor's high-level design.

## **Control Unit Design**

Detail the design of the control unit and its role in instruction execution.

## **Data Path Design**

Explain the organization and components of the data path.

## **Memory Hierarchy**

Describe how memory is managed within the microprocessor.

# **Implementation**

## **Register Transfer Level (RTL) Description**

Offer a detailed description of the microprocessor's design at the RTL, including data and control flow.

## **Simulation Results**

Present results from simulations to verify the correctness and performance of the design.

## **Hardware Description Language (HDL) Code Snippets**

Include relevant portions of the HDL code.

# **Testing and Verification**

## **Test Plan**

Outline the plan for testing and verifying the microprocessor's functionality.

## **Simulation Verification**

Discuss how simulations were used to validate the design.

## **Hardware Testing Strategies**

Detail strategies for testing the microprocessor on hardware.

# **Results and Analysis**

## **Evaluation of Microprocessor Performance**

Assess the microprocessor's performance against specified criteria.

## **Comparison with Project Requirements**

Evaluate how well the designed microprocessor meets the initial project requirements.

# **Conclusion**

## **Summary of Achievements**

Summarize the key achievements of the project.

## **Lessons Learned**

Reflect on lessons learned during the design and implementation process.

# **Resources**

1. [Albert P. Malvino, Jerald A. Brown. “Digital Computer Electronics” (3rd Edition)](https://ia803000.us.archive.org/8/items/367026792DigitalComputerElectronicsAlbertPaulMalvinoAndJeraldABrownPdf1/367026792-Digital-Computer-Electronics-Albert-Paul-Malvino-and-Jerald-A-Brown-pdf%20%281%29.pdf)

# **Appendices**

## **Appendix 1: Microprocessor Full Instruction Set**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Instruction** | **Op Code** | **Addressing Mode** | **T states** | **Flags** | **Bytes** | **Type** | **Main Effect** |
| **Initialization Instructions** | | | | | | | |
| PRGM ***double-byte*** | 40 | Immediate |  | - | 3 | J | DataMemStart ← double-byte |
| **Memory-Reference Instructions** | | | | | | | |
| LDR ***A***, ***address*** | E0 | Direct |  | - | 3 | D | A ← Maddress |
| LDR ***B***, ***address*** | E1 | Direct |  | - | 3 | D | B ← Maddress |
| LDR ***C***, ***address*** | E2 | Direct |  | - | 3 | D | C ← Maddress |
| LDR ***D***, ***address*** | E3 | Direct |  | - | 3 | D | D ← Maddress |
| STR ***A***, ***address*** | E4 | Direct |  | - | 3 | D | Maddress ← A |
| STR ***B***, ***address*** | E5 | Direct |  | - | 3 | D | Maddress ← B |
| STR ***C***, ***address*** | E6 | Direct |  | - | 3 | D | Maddress ← C |
| STR ***D***, ***address*** | E7 | Direct |  | - | 3 | D | Maddress ← D |
| **Register Instructions** | | | | | | | |
| MOV ***A***, ***A*** | 40 | Register |  | - | 1 | DR | A ← A |
| MOV ***A***, ***B*** | 41 | Register |  | - | 1 | DR | A ← B |
| MOV ***A***, ***C*** | 42 | Register |  | - | 1 | DR | A ← C |
| MOV ***A***, ***D*** | 43 | Register |  | - | 1 | DR | A ← D |
| MOV ***B***, ***A*** | 44 | Register |  | - | 1 | DR | B ← A |
| MOV ***B***, ***B*** | 45 | Register |  | - | 1 | DR | B ← B |
| MOV ***B***, ***C*** | 46 | Register |  | - | 1 | DR | B ← C |
| MOV ***B***, ***D*** | 47 | Register |  | - | 1 | DR | B ← D |
| MOV ***C***, ***A*** | 48 | Register |  | - | 1 | DR | C ← A |
| MOV ***C***, ***B*** | 49 | Register |  | - | 1 | DR | C ← B |
| MOV ***C***, ***C*** | 4A | Register |  | - | 1 | DR | C ← C |
| MOV ***C***, ***D*** | 4B | Register |  | - | 1 | DR | C ← D |
| MOV ***D***, ***A*** | 4C | Register |  | - | 1 | DR | D ← A |
| MOV ***D***, ***B*** | 4D | Register |  | - | 1 | DR | D ← B |
| MOV ***D***, ***C*** | 4E | Register |  | - | 1 | DR | D ← C |
| MOV ***D***, ***D*** | 4F | Register |  | - | 1 | DR | D ← D |
| MVI ***A***, ***byte*** | C0 | Immediate |  | - | 2 | I | A ← byte |
| MVI ***B***, ***byte*** | C1 | Immediate |  | - | 2 | I | B ← byte |
| MVI ***C***, ***byte*** | C2 | Immediate |  | - | 2 | I | C ← byte |
| MVI ***D***, ***byte*** | C3 | Immediate |  | - | 2 | I | D ← byte |
| **Arithmetic Instructions** | | | | | | | |
| ADD ***A***, ***A*** | 50 | Register |  | ZCPS | 1 | DR | A ← A + A |
| ADD ***A***, ***B*** | 51 | Register |  | ZCPS | 1 | DR | A ← A + B |
| ADD ***A***, ***C*** | 52 | Register |  | ZCPS | 1 | DR | A ← A + C |
| ADD ***A***, ***D*** | 53 | Register |  | ZCPS | 1 | DR | A ← A + D |
| ADD ***B***, ***A*** | 54 | Register |  | ZCPS | 1 | DR | B ← B + A |
| ADD ***B***, ***B*** | 55 | Register |  | ZCPS | 1 | DR | B ← B + B |
| ADD ***B***, ***C*** | 56 | Register |  | ZCPS | 1 | DR | B ← B + C |
| ADD ***B***, ***D*** | 57 | Register |  | ZCPS | 1 | DR | B ← B + D |
| ADD ***C***, ***A*** | 58 | Register |  | ZCPS | 1 | DR | C ← C + A |
| ADD ***C***, ***B*** | 59 | Register |  | ZCPS | 1 | DR | C ← C + B |
| ADD ***C***, ***C*** | 5A | Register |  | ZCPS | 1 | DR | C ← C + C |
| ADD ***C***, ***D*** | 5B | Register |  | ZCPS | 1 | DR | C ← C + D |
| ADD ***D***, ***A*** | 5C | Register |  | ZCPS | 1 | DR | D ← D + A |
| ADD ***D***, ***B*** | 5D | Register |  | ZCPS | 1 | DR | D ← D + B |
| ADD ***D***, ***C*** | 5E | Register |  | ZCPS | 1 | DR | D ← D + C |
| ADD ***D***, ***D*** | 5F | Register |  | ZCPS | 1 | DR | D ← D + D |
| SUB ***A***, ***A*** | 60 | Register |  | ZCPS | 1 | DR | A ← A – A |
| SUB ***A***, ***B*** | 61 | Register |  | ZCPS | 1 | DR | A ← A – B |
| SUB ***A***, ***C*** | 62 | Register |  | ZCPS | 1 | DR | A ← A – C |
| SUB ***A***, ***D*** | 63 | Register |  | ZCPS | 1 | DR | A ← A – D |
| SUB ***B***, ***A*** | 64 | Register |  | ZCPS | 1 | DR | B ← B – A |
| SUB ***B***, ***B*** | 65 | Register |  | ZCPS | 1 | DR | B ← B – B |
| SUB ***B***, ***C*** | 66 | Register |  | ZCPS | 1 | DR | B ← B – C |
| SUB ***B***, ***D*** | 67 | Register |  | ZCPS | 1 | DR | B ← B – D |
| SUB ***C***, ***A*** | 68 | Register |  | ZCPS | 1 | DR | C ← C – A |
| SUB ***C***, ***B*** | 69 | Register |  | ZCPS | 1 | DR | C ← C – B |
| SUB ***C***, ***C*** | 6A | Register |  | ZCPS | 1 | DR | C ← C – C |
| SUB ***C***, ***D*** | 6B | Register |  | ZCPS | 1 | DR | C ← C – D |
| SUB ***D***, ***A*** | 6C | Register |  | ZCPS | 1 | DR | D ← D – A |
| SUB ***D***, ***B*** | 6D | Register |  | ZCPS | 1 | DR | D ← D – B |
| SUB ***D***, ***C*** | 6E | Register |  | ZCPS | 1 | DR | D ← D – C |
| SUB ***D***, ***D*** | 6F | Register |  | ZCPS | 1 | DR | D ← D – D |
| INR ***A*** | 0 | Register |  | Z-PS | 1 | SR | A ← A + 1 |
| INR ***B*** | 1 | Register |  | Z-PS | 1 | SR | B ← B + 1 |
| INR ***C*** | 2 | Register |  | Z-PS | 1 | SR | C ← C + 1 |
| INR ***D*** | 3 | Register |  | Z-PS | 1 | SR | D ← D + 1 |
| DER ***A*** | 4 | Register |  | Z-PS | 1 | SR | A ← A - 1 |
| DER ***B*** | 5 | Register |  | Z-PS | 1 | SR | B ← B - 1 |
| DER ***C*** | 6 | Register |  | Z-PS | 1 | SR | C ← C - 1 |
| DER ***D*** | 7 | Register |  | Z-PS | 1 | SR | D ← D - 1 |
| **Logical Instructions** | | | | | | | |
| ROR ***A*** | 8 | Register |  | -C-- | 1 | SR | A ← A x 2 (Rotate all right) |
| ROR ***B*** | 9 | Register |  | -C-- | 1 | SR | B ← B x 2 (Rotate all right) |
| ROR ***C*** | A | Register |  | -C-- | 1 | SR | C ← C x 2 (Rotate all right) |
| ROR ***D*** | B | Register |  | -C-- | 1 | SR | D ← D x 2 (Rotate all right) |
| ROL ***A*** | C | Register |  | -C-- | 1 | SR | A ← A / 2 (Rotate all left) |
| ROL ***B*** | D | Register |  | -C-- | 1 | SR | B ← B / 2 (Rotate all left) |
| ROL ***C*** | E | Register |  | -C-- | 1 | SR | C ← C / 2 (Rotate all left) |
| ROL ***D*** | F | Register |  | -C-- | 1 | SR | D ← D / 2 (Rotate all left) |
| ANR ***A***, ***A*** | 70 | Register |  | ZCPS | 1 | DR | A ← A & A |
| ANR ***A***, ***B*** | 71 | Register |  | ZCPS | 1 | DR | A ← A & B |
| ANR ***A***, ***C*** | 72 | Register |  | ZCPS | 1 | DR | A ← A & C |
| ANR ***A***, ***D*** | 73 | Register |  | ZCPS | 1 | DR | A ← A & D |
| ANR ***B***, ***A*** | 74 | Register |  | ZCPS | 1 | DR | B ← B & A |
| ANR ***B***, ***B*** | 75 | Register |  | ZCPS | 1 | DR | B ← B & B |
| ANR ***B***, ***C*** | 76 | Register |  | ZCPS | 1 | DR | B ← B & C |
| ANR ***B***, ***D*** | 77 | Register |  | ZCPS | 1 | DR | B ← B & D |
| ANR ***C***, ***A*** | 78 | Register |  | ZCPS | 1 | DR | C ← C & A |
| ANR ***C***, ***B*** | 79 | Register |  | ZCPS | 1 | DR | C ← C & B |
| ANR ***C***, ***C*** | 7A | Register |  | ZCPS | 1 | DR | C ← C & C |
| ANR ***C***, ***D*** | 7B | Register |  | ZCPS | 1 | DR | C ← C & D |
| ANR ***D***, ***A*** | 7C | Register |  | ZCPS | 1 | DR | D ← D & A |
| ANR ***D***, ***B*** | 7D | Register |  | ZCPS | 1 | DR | D ← D & B |
| ANR ***D***, ***C*** | 7E | Register |  | ZCPS | 1 | DR | D ← D & C |
| ANR ***D***, ***D*** | 7F | Register |  | ZCPS | 1 | DR | D ← D & D |
| ORR ***A***, ***A*** | 80 | Register |  | ZCPS | 1 | DR | A ← A | A |
| ORR ***A***, ***B*** | 81 | Register |  | ZCPS | 1 | DR | A ← A | B |
| ORR ***A***, ***C*** | 82 | Register |  | ZCPS | 1 | DR | A ← A | C |
| ORR ***A***, ***D*** | 83 | Register |  | ZCPS | 1 | DR | A ← A | D |
| ORR ***B***, ***A*** | 84 | Register |  | ZCPS | 1 | DR | B ← B | A |
| ORR ***B***, ***B*** | 85 | Register |  | ZCPS | 1 | DR | B ← B | B |
| ORR ***B***, ***C*** | 86 | Register |  | ZCPS | 1 | DR | B ← B | C |
| ORR ***B***, ***D*** | 87 | Register |  | ZCPS | 1 | DR | B ← B | D |
| ORR ***C***, ***A*** | 88 | Register |  | ZCPS | 1 | DR | C ← C | A |
| ORR ***C***, ***B*** | 89 | Register |  | ZCPS | 1 | DR | C ← C | B |
| ORR ***C***, ***C*** | 8A | Register |  | ZCPS | 1 | DR | C ← C | C |
| ORR ***C***, ***D*** | 8B | Register |  | ZCPS | 1 | DR | C ← C | D |
| ORR ***D***, ***A*** | 8C | Register |  | ZCPS | 1 | DR | D ← D | A |
| ORR ***D***, ***B*** | 8D | Register |  | ZCPS | 1 | DR | D ← D | B |
| ORR ***D***, ***C*** | 8E | Register |  | ZCPS | 1 | DR | D ← D | C |
| ORR ***D***, ***D*** | 8F | Register |  | ZCPS | 1 | DR | D ← D | D |
| XRR ***A***, ***A*** | 90 | Register |  | ZCPS | 1 | DR | A ← A ^ A |
| XRR ***A***, ***B*** | 91 | Register |  | ZCPS | 1 | DR | A ← A ^ B |
| XRR ***A***, ***C*** | 92 | Register |  | ZCPS | 1 | DR | A ← A ^ C |
| XRR ***A***, ***D*** | 93 | Register |  | ZCPS | 1 | DR | A ← A ^ D |
| XRR ***B***, ***A*** | 94 | Register |  | ZCPS | 1 | DR | B ← B ^ A |
| XRR ***B***, ***B*** | 95 | Register |  | ZCPS | 1 | DR | B ← B ^ B |
| XRR ***B***, ***C*** | 96 | Register |  | ZCPS | 1 | DR | B ← B ^ C |
| XRR ***B***, ***D*** | 97 | Register |  | ZCPS | 1 | DR | B ← B ^ D |
| XRR ***C***, ***A*** | 98 | Register |  | ZCPS | 1 | DR | C ← C ^ A |
| XRR ***C***, ***B*** | 99 | Register |  | ZCPS | 1 | DR | C ← C ^ B |
| XRR ***C***, ***C*** | 9A | Register |  | ZCPS | 1 | DR | C ← C ^ C |
| XRR ***C***, ***D*** | 9B | Register |  | ZCPS | 1 | DR | C ← C ^ D |
| XRR ***D***, ***A*** | 9C | Register |  | ZCPS | 1 | DR | D ← D ^ A |
| XRR ***D***, ***B*** | 9D | Register |  | ZCPS | 1 | DR | D ← D ^ B |
| XRR ***D***, ***C*** | 9E | Register |  | ZCPS | 1 | DR | D ← D ^ C |
| XRR ***D***, ***D*** | 9F | Register |  | ZCPS | 1 | DR | D ← D ^ D |
| **Branching Operations** | | | | | | | |
| JMP ***address*** | D1 | Immediate |  | - | 3 | J | PC ← address |
| JZ ***address*** | D2 | Immediate |  | - | 3 | J | PC ← address if Z = 0 |
| **Stack Instructions** | | | | | | | |
| CALL ***address*** | D3 | Immediate |  | - | 3 | J | PC ← address |
| RET | F0 | - |  | - | 1 | O | PC ← return address |
| PUSH ***A*** | 10 | Register |  | - | 1 | SR | Mstack – 1 ← A |
| PUSH ***B*** | 11 | Register |  | - | 1 | SR | Mstack – 1 ← B |
| PUSH ***C*** | 12 | Register |  | - | 1 | SR | Mstack – 1 ← C |
| PUSH ***D*** | 13 | Register |  | - | 1 | SR | Mstack – 1 ← D |
| POP ***A*** | 14 | Register |  | - | 1 | SR | A ← Mstack |
| POP ***B*** | 15 | Register |  | - | 1 | SR | B ← Mstack |
| POP ***C*** | 16 | Register |  | - | 1 | SR | C ← Mstack |
| POP ***D*** | 17 | Register |  | - | 1 | SR | D ← Mstack |
| **Misc Instructions** | | | | | | | |
| OUTX ***A*** | 18 | Register |  | - | 1 | SR | PORTX ← A |
| OUTX ***B*** | 19 | Register |  | - | 1 | SR | PORTX ← B |
| OUTX ***C*** | 1A | Register |  | - | 1 | SR | PORTX ← C |
| OUTX ***D*** | 1B | Register |  | - | 1 | SR | PORTX ← D |
| OUTY ***A*** | 1C | Register |  | - | 1 | SR | PORTY ← A |
| OUTY ***B*** | 1D | Register |  | - | 1 | SR | PORTY ← B |
| OUTY ***C*** | 1E | Register |  | - | 1 | SR | PORTY ← C |
| OUTY ***D*** | 1F | Register |  | - | 1 | SR | PORTY ← D |
| OUTZ ***A*** | 20 | Register |  | - | 1 | SR | PORTZ ← A |
| OUTZ ***B*** | 21 | Register |  | - | 1 | SR | PORTZ ← B |
| OUTZ ***C*** | 22 | Register |  | - | 1 | SR | PORTZ ← C |
| OUTZ ***D*** | 23 | Register |  | - | 1 | SR | PORTZ ← D |
| NOP | FE | - |  | - | 1 | O | Delay (No Operation) |
| HLT | FF | - |  | - | 1 | O | Stop Processing |