# ADDIS ABABA UNIVERSITY INSTITUTE OF TECHNOLOGY


## SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
## COMPUTER ARCHITECTURE AND ORGANIZATION
## PROJECT
## PHASE -2


| **Name:** | **ID-No** |
|---|---|
| • Abenezer Alemayehu | UGR/8513/13 |
| • Eyosias Solomon | UGR/9004/14 |
| • Gutu Regassa | UGR/5449/13 |
| • Hayat Gebeyehu | UGR/0240/13 |
| • Helen Girma | UGR/4633/13 |


SUBMITTED TO:- Mr. Kinde

## Introduction

At its core, the primary goal of this project is to empower us with the freedom to conceptualize and design a processor of our own from inception to realization. The formation of the project is divided into two stages, each of which introduces considerable additional details and improvements. In phase one, we have submitted our proposal document containing the foundational elements of our proposed processor including lists of instructions for processor execution, detailed specifications, including instruction format, and considerations of design issues such as operation repertoire, registers, data types, memory hierarchy, and organizational structure. A schematic block diagrams were also included to visually represent the intended CPU design.

The second phase is the process of implementation and modification. Considering feedback received during the time we first presented our idea, we worked to incorporate our teacher’s constructive comments into the overall design of our project. The final step, in which we used Proteus to simulate the processor, shows the outcome of our work. This simulation provides a virtual setting for evaluating and verifying our CPU architecture, showing the design's practical applicability. In doing the project we had to face the fact that we were running out of time due to the demands of a busy semester. As a result, the proposal saw a necessary revision that included revisions that were considered important for the successful completion of the project

**Instruction Set**

| OPCODE | OPERAND 1 | OPERAND 2 |
|---|---|---|

The instruction set we thought we could implement easily was the one shown above containing 16 bits with the first 4 bits for the OPCODE and the rest 12 bit contains 2 operands each 6 bits width. The instruction format uses two-address instructions and the addressing mode would be direct.But with its complexity of application we had to reduce 16 bits it into 8-bit instruction set mentioned below and the new processor we built uses one addressing mode

| OPCODE(4-bit) | OPERAND(4-bit) |
|---|---|

The OPCODES that will be used in the updated processor along with their operation and descriptions are shown in the table below.

| Opcode | Operation | Instruction Format | Description |
|---|---|---|---|
| 0000 | Load | LDA X | Loads data on memory address “X” to register A. |
| 0001 | Add | ADD X | Adds data on register A with data from memory address “X”. |
| 0010 | Subtract | SUB X | Subtracts data of memory address “X” from data on register A. |
| 0011 | AND | AND X | Performs logical AND operation on data in Register A and memory address “X”. |
| 0100 | OR | OR X | Performs logical OR operation on data in Register A and memory address “X”. |
| 0101 | XOR | XOR X | Performs logical XOR operation on data in Register A and memory address “X”. |
| 0110 | STORE | STORE X | Stores data on Register A to memory address X. |
| 0111 | OUT | OUT | Displays the data on register A. |

## Design Issues

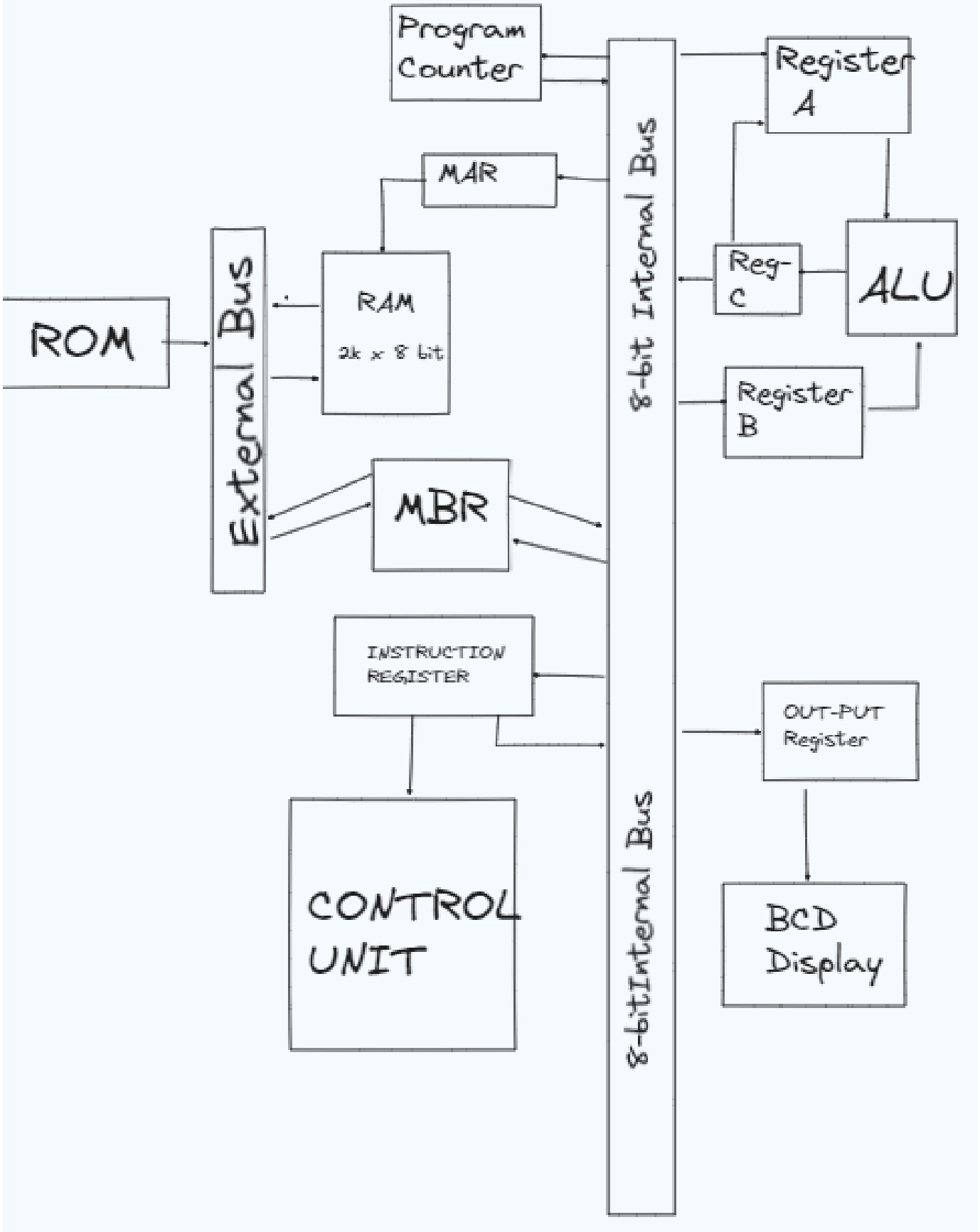
We have addressed main design issues in our work.

1. **Operation repertoire**: In our design we have used 2 arithmetic operations and 2 logical operations, namely addition, subtraction, AND, XOR.
2. **Registers**: We totally used around 8 Different Registers, Program Counter (PC), Input Register, MAR, MDR, Instruction Register, Register A, Register B, Register C , Output Register.
3. **Data Types**: The processor supports two data types, Integer and Boolean.
4. **Instruction Format**: Our instruction Format as mentioned above contains a 8bit of instruction which inside partitions four bit for Op-code and 4 bit operand.
5. **Memory Hierarchy and Organization:** In our design, we have adopted a SAP-type organization because, the last time we presented our thesis we thought of using RISC organization but as mentioned above because of a bit of complexity in our circuit we had to change it into SAP type organization. SAP organization that is simple as possible organization that performs a simple task like Add, Sub and other Logical Operations.

## List of Components used

1. Program counter (PC)
2. Memory address register (MAR)
3. Memory buffer register (MBR)
4. Register A
5. Register B
6. Register C
7. ALU
8. Instruction register (IR)
9. RAM
10. ROM (with Loading circuit)
11. Output register
12. Display
13. Control Unit

The block diagram below shows the basic design of the processor



## Explanation of Components:

- **Program Counter (PC)**: PC is a four bit register that holds the address of the next instruction. To implement PC we have used 74LS163 IC which is a 4-bit synchronous binary counter and 74LS244 an octal buffer/line driver with 3-state outputs. Additional logic gates are also used in setting up the internal operation of the PC. It has 5 input ports; two for control signals coming from the control unit (EPC, IPC), one for clock and the remaining two (LPC, clr) are shorted and given a low input. It also has 4 output ports for 4 bit address data.

- **Memory Address Register (MAR)**: MAR is a four bit register that holds the address of the instruction going to be executed. To implement MAR we have used 74LS173 IC which is a 4-bit D-type register with three-state outputs. The MAR has different input ports, one for MARI and CLK and 4 for input address data. It also has 4 output ports that are directly connected to the RAM.

- **Memory Buffer Register (MBR):** MBR is an eight bit register that holds a data that is read from RAM or going to be written on the RAM. To implement MBR we have used two 74LS173 IC by cascading them to support 8 bit data. The MBR has 8 input and output ports which are connected to both the system bus and internal bus. In addition it has two input ports for control signals (MBRI, MBRO) and for CLK.

- **Instruction Register (IR):** IR is an eight bit register that holds instructions fetched from the RAM. The implementation of IR is almost the same with MBR except their connection. Unlike the MBR, IR is connected to the internal bus and the Control Unit. It also has a CLK and control signals (IRI, IRO).

- **Register A (REG A):** REG A is an eight bit register that holds a data need for ALU operation. To implement REG A we have used two 74LS173 IC by cascading them to support 8 bit data. It has 8 input and output ports which are connected to internal and ALU. It also has a CLK and control signals (RAI, RAO).

- **Register B (REG B):** REG B is the same as REG A and has control signals (RBI, RBO).

- **Register C (REG C):** REG C is an eight bit register used to temporarily store the data coming from ALU. It has the same implementation with REG A and REG B. It also has a CLK and control signals (RCI, RCO).

- **Output Register (OP REG):** OP REG is also an eight bit data register used to store a data which is going to be displayed on the display. It has 8 input and output ports which are connected to the internal bus and the display circuit respectively. . It also has a CLK and control signals (ROPI, ROPO).

- **Arithmetic Logic Unit (ALU):** while building the ALU we were supposed to use the IC 74LS382 but that IC doesn't have a simulation as such we had to use 74LS181 which uses 5 selector as such we had to build our own circuit which takes 3 selector and

outputs 5 selector. We used a Karnaugh map to and basic knowledge of Digital Logic Design. Then we cascaded two 74LS181 in order to calculate or perform a certain operation on 8-bit data because the single IC can operate only on 4bit data.it takes the data from Register A and Register B and directly operates on the data's received and send the data to Register C as an accumulator.

- **Display:** the display follows the output register. When the control unit releases a signal for the output register for it to release its contents it will directly send it to the Display part which is connected to the 7-segment BCD. One 7-BCD takes 4 bit and converts into a decimal. Thus we had to cascade three 7segment BCD. But our input it is 8 bit hence for that to be displayed on three different 7 segment display we had to build our own circuit which decodes the 8 bit into 10 different o/p which the the 2 ports of the last 7seg-BCD will be grounded. Thus the decoder must import 8 bit data from the output register and convert it into 10bit data. Hence our circuit it incorporates that. Inorder to build the the decoder we used a basic Digital logic design and cascaded them to produce a 10 bit output. We used components like NOT gate, AND gate and OR gate, we used K-map like the ALU in order to figure out how we should connect them.

- **Control Unit (CU):** The control unit is the main part in the processor as it controls all the operations and sequence of everything the processor does. We implemented a hardwired controller on our design for ease of control and integration. The sequencing is implemented using a 74LS163 IC which is a 4-bit synchronous binary counter. It is designed to count from 0 to 7 then resets to 0 and counts again. This is done because each instruction takes 8 clock cycles to complete. The rest of the connection is done using basic logic AND and OR gates which connect each instruction (Load, Add, …) and the time cycle (t1, t2, … , t8) to determine which control signals should be activated.

- **RAM:** The ram is implemented using the IC 6116 which has 11 bit address to store a bit data. We only needed a 4 bit address ram thus the address ports A4 to A10 are grounded and we used address bits A0 to A3. Therefore, the ram holds 8 bit data on 16 addresses.

- **ROM:** For the ROM we used the IC 27C512 which is a 12kbit (64k x 8 bit) EPROM which has 16-bit address but we only used addresses A0 to A3 and grounded the rest. The main function of the programmable ROM is to load the data (Instructions) on it to the RAM so that it could be accessed by the processor. Therefore, we have also incorporated a **loading circuit** to load the data on the ROM to our RAM. This is implemented using 2 cascaded selector ICs 74HC157 which connect the RAM address pins and control inputs with the processor (MAR and Control Unit) when executing instructions or with the loading circuit when we want to load data on it. Then a 74S163 4-bit counter IC is connected to the address pins of both the RAM and ROM which counts the address from 0000 to 1111 as the RAM is being loaded during each count at that specific address. Finally control inputs for both the RAM and ROM are implemented using basic NAND and NOT gates which tell the ROM to release data and RAM to take the data in.

Something to Note here is that we didn't manage to fully separate the Internal and External bus that is connected to the MBR and this creates a bus contention when trying to load the

RAM. As a temporary solution we were loading the RAM on T7 or T8 cycles of instruction Load (0000) where the bus is free and then resetting the PC and Control unit and starting the control Unit again with the RAM loaded.

## Control Signals and their descriptions.

There are 21 control signals that our control unit sends to each component in the processor. The control signals and their descriptions are listed in the table below.

1. **EPC:** (Enable program counter) the PC releases its data to the bus.
2. **IPC:** (Increase program counter) increases PC by 1.
3. **MARI:** The MAR takes the data available on the bus.
4. **CE:** Activates the RAM data pins.
5. **OE:** The RAM takes the data in from MBR (CE must be active and WE must not.)
6. **WE:** The RAM releases data to the MBR (CE must be active and OE must not.)
7. **MBRI:** MBR takes the data from either RAM or Internal bus.
8. **MBRO:** MBR releases data to the RAM or the Internal bus.
9. **IRI:** The IR takes data from the bus.
10. **IRO:** The IR releases data (The 4 higher nibbles, 4 to 7) to the bus.
11. **RAI:** Register A takes data from the bus.
12. **RAO:** Register A releases data to the ALU.
13. **RBI:** Register B takes data from the bus.
14. **RBO:** Register B releases data to the ALU.
15. **RCI:** Register C takes data from the ALU.
16. **RCO:** Register C releases data to the bus.
17. **A0:** Control signal that tells the ALU what operation to perform.
18. **A1:** Control signal that tells the ALU what operation to perform.
19. **A2:** Control signal that tells the ALU what operation to perform.
20. **ROPI:** The output register takes data from the bus.
21. **ROPO:** The output register releases data to the Display.

## Conclusion

This project, which represents the completion of our work to design and build a processor, is evidence for the process from the conceptualization phase, where we described our intended design in an overall proposal, to the dynamic execution and modifying in the second phase.
We handled the complexities of our project with flexibility in spite of the difficulties brought on by a hard academic semester. Our first proposal was revised to better reflect our practical viewpoint.
The ultimate accomplishment is the Proteus simulation, which gives our processor a virtual platform to demonstrate its powers. This last phase of the project cycle verifies the efficiency of the designed processor. Upon reflection on this crucial practical experience, we appreciate the teamwork, creativity, and resiliency that have characterized our project.