

**COAL PROJECT**

**REPORT**

**Group Member:**

Asad Ali (2022903)

Hamayun Nasir (2022189)

Huzaifa Ghani (2022389)

Aqib Shakeel (2022104)

Sohaib Khan(2022551)

**TABLE OF CONTENTS:**

1. Abstract -----------------------------------------------------------------------------3
2. List of Registers --------------------------------------------------------------------3
3. Instruction Code Format----------------------------------------------------------6
4. Simulator Instruction--------------------------------------------------------------9
5. Memory reference register -------------------------------------------------------11
6. Register Reference Register -----------------------------------------------------12
7. Input/output Register -------------------------------------------------------------13

**Abstract**

The architecture and features of a computer system designed for certain purposes are presented in this Project Report. Highlighting the Address Register (AR), Data Register (DR), Accumulator (AC), Program Counter (PC), Temporary Register (TR), Multiplication Product Register (MPR), Input Register (INPR), Output Register (OUTR), and Address Mode Register (AM), the report explores the complex architecture of numerous registers and their functions within the system.   
  
In addition, it clarifies how memory is organized, with special attention to Random Access Memory (RAM) and how it stores binary data and instructions. The report also covers the FLAGS system, including the functions of the I bit for interrupt cycles, the Indirect bit (E), the Interrupt Enable (IEN), the Input Flag (FGI), and the Output Flag (FGO).

The report also offers information on addressing modes, common bus architecture in CPU, instructions structure, and a complete set of instructions for input/output, register reference, and memory reference. An extensive analysis of the instruction cycle, including obtaining, decoding, and executing instructions, comes at the end.   
  
All things considered, the Project Report provides an extensive rundown of the instruction set, memory organization, computer architecture, and functioning, offering insightful information for comprehending and making good use of the system.

**1. List of Registers:**

In this architecture there are a total of registers.

**Address Register (AR)**: This is a 16-bit register used to store the memory address. It stores the address of the data or instructions in the computer memory. It is used for memory access operations, from where data should be fetched or stored in the memory.

**Data Register (DR):** This is a 10-bit register used to store the data during the processing. It stores the operand on which instruction will be executed.

**Accumulator (AC):** This is a 10-bit special purpose register in the CPU. It is used for performing arithmetic and logic operations. It holds one operand and processes the arithmetic and logic instruction and then stores the result.

**Program counter (PC):** Program counter is a register used to stores the address of the next instruction to be executed. It is a 16-bit register in this architecture which is automatically incremented after every instruction is fetched and pointing to the next instruction stored in the memory.

**Temporary Register (TR):** Temporary register is used to store data temporarily during processing. Like the other registers in the CPU, it is not used for a special purpose. It is a 10-bit register used to hold the intermediate results during complex arithmetic and logic operations.

**Multiplication Product Register (MPR**): Multiplication product register is used to store the product of multiplication. As the product of multiplication is 32-bits so we cannot store it in accumulator, so we need a special register for this purpose.

**Input Register (INPR):** Input register is a special purpose register used to store input. When something is input, it is fetched from input register to accumulator.

**Output Register (OUTR):** Output register holds when something needs to be outputted. The operand is fetched from accumulator to output register then from output register it is shown on screen.

**Address Mode register (AM):** Address mode register is used to store the addressing bit of the instruction.

**Memory (RAM):**

Main memory stores data and instructions in binary format. Each memory location, also known as a memory address, holds a fixed number of bits, forming the smallest unit of storage in the memory. Main memory is typically byte-addressable, meaning that each memory location can be individually accessed and manipulated. In a system with 10-bit words, there can be 2^10 =1024unique memory locations. These locations are identified by unique binary addresses ranging from 0 to 1023.

Table

Description automatically generated

**FLAGS:**

1. **Interrupt enable IEN (you cannot take input if)**

IEN is a control system that can control external and internal signals. When IEN is on the computing system starts receiving signals from external and internal signals. When IEN is off it will start blocking the signal and instruction execute. This computing system is used where the input is necessary in a security system.

1. **FGI (Input Flag you cannot take input if FGI = 0 it must be FGI = 1)**

FGI (input flag) is a status indicator which shows that the input data is available or not available. So, when FGI is 0 means there is no input data. When FGI is 1 means there is input data is ready to processed the system.

1. **FGO (output Flag you cannot see output if FGO = 0 it must be FGO = 1)**

FGO (output flag) is a status indicator which shows whether the output data is display or not in system. So, when FGI is 0 means there is no output data to display. When FGI is 1 means there is output data ready to display on the system.

1. **I bit for interrupt cycle:**

The I bit is a status register which allow the processor to respond to external request when I bit is set. When I bit is cleared mean the processor discard all signals. The I bit works like a switch.

1. **E (Indirect bit)**

In indirect bit it tells the processor to read the data directly from a memory location or indirect through a register.

**Common Bus in CPU:** A common bus refers to a shared communication pathway that allows different registers of a computer system to exchange data and control signals. The common bus typically consists of multiple electrical lines or traces that carry various types of signals, including data, addresses, and control signals. These signals are transmitted using a protocol that defines how information is formatted, transferred, and interpreted by the connected components. One of the key advantages of common bus architecture is its simplicity and flexibility.

A diagram of a memory

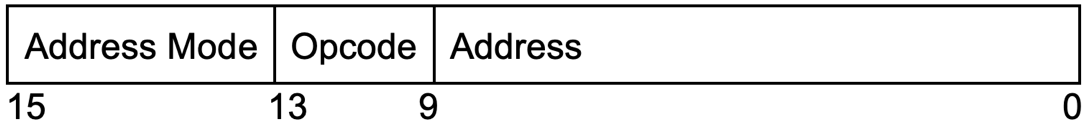
Description automatically generated

**Instructions Code Format**

Instruction Code consists of 21 bits.

* 2 addressing mode bits.
* 4 opcode bits
* 10 memory location bits

**Bits Sequence**



* 14-15 Addressing mode.
* 10-13 Opcode bits
* 9-0 address bits

**Instructions Cycle:**

Fundamental memory-based applications are executed by computerized computers. Information and a set of informational put away in (more often than not) successive memory locales make up a program. Each of these informational is considered to have been completely executed once it has completed an instruction cycle. These enlightening work in a successive mold. The three steps of an instruction cycle are gotten, decoded, and execute. The taking after gives a point by point portrayal of these:

1. **Fetch Instruction:**

The address of the ensuing instruction that must be retrieved from memory is put away within the Program Counter, or PC. This address in PC is moved (parallel) to the enlist AR amid the get stage in a single clock beat. The instruction is at that point augmented by 1 and set within the consequent memory address. The PC must at that point point to it within the ensuing clock beat. Also, the'read' line is actuated and get to is made to the memory M shown by the address in AR, or M[AR]. The command (or information) was moved to the framework common transport from M[AR]. Another, the Instruction Enroll (IR) is stacked with the data-bus substance. The bring stage closes when information or enlightening are entered into IR.

AR←PC

IR←M[AR], PC←PC+1

**2. Decode Instruction:**

AR is stacked with the lower ten bits. The 4x16 decoder, which could be a component of the hardwired control unit, gets the another 4 bits, or the opcode, and employments them to create the control flag Ki. The Mode Enroll gets the two most imperative bits from IR.

Decoder [10:13],

AM↞IR [14:15],

and AR↞IR [0:9]

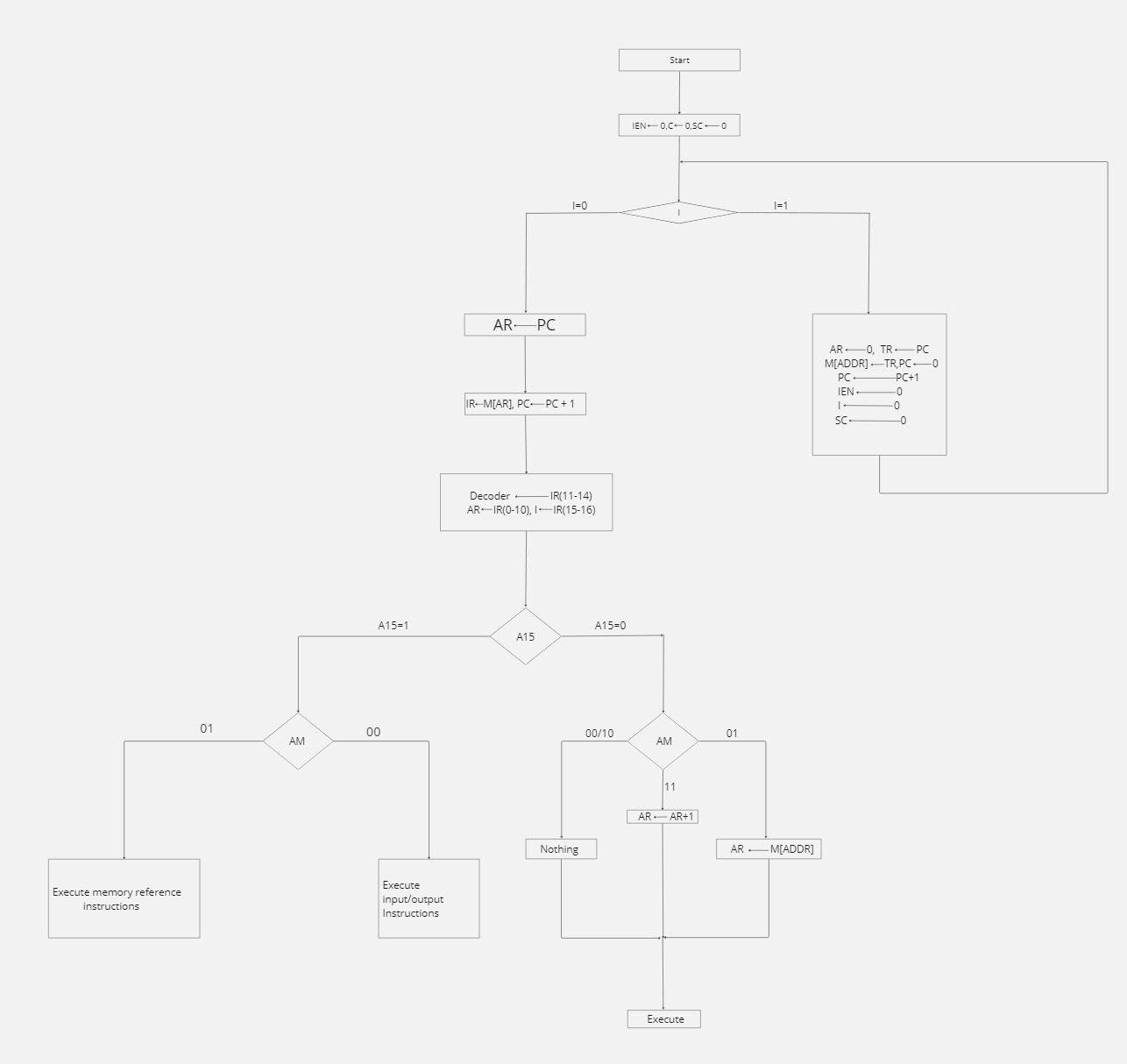
When a backhanded address happens, the operand's address—which is the esteem put away in memory—is brought to the address given by AR. Since the operand's address is as of now display in AR, this step is skipped in coordinate tending to. Essentially, within the case of immediate tending to, AR contains the operand itself. In all other cases, the operand is spared in DR utilizing this micro-operation T4, thus it ought to be moved to DR.

DR←M[AR]

**3.Execute Instruction:**

Any flag between K0 and K15 can be created by the 4x16 decoder for the opcodes 0000 through 1111, individually. Each Ki has an related instruction. Three distinctive sorts of information exist. Memory reference information are spoken to by the numbers 0001 through 1111. It is utilized as input/output or register instructions when it turns 0000. IO informational are carried out when the Mode Enlist MR [0]'s cleared outside bit is set to 0. Enroll reference information is carried out when MR [0] breaks even with 1.

**Flow diagram of Instruction cycle**



**SIMULATOR INSTRUCTIONS**

1. Memory Reference

Instructions that give a memory reference that data is stored in RAM.

1. Register Reference

Instructions that give a Register reference that data is stored in any Register.

1. Input/Output

Instructions that are used to take input bits or display output

**Addressing Modes:**

There are 4 addressing modes for different operations.

1. Direct
2. Indirect
3. Immediate
4. Auto Indexing

The addressing bits are used to access the memory location to retrieve data or write data.

**1. Direct Addressing:**

In direct addressing mode, the instruction specifies the exact memory address where the data is located. The CPU directly accesses this memory location to read or write data. Direct addressing is straightforward and efficient for accessing specific data stored in memory.

**2. Indirect Addressing:**

In indirect addressing mode, the instruction specifies a memory address that contains the actual memory address where the data is located. In this mode the memory address of the operand is fetched by going to the memory address stored in the address register.

**3. Immediate Addressing:**

In immediate addressing mode, the operand will not be fetched from the memory rather operand is present in the instruction itself. The operand is fetched directly from the instruction.

**4. Auto Indexing:**

When using auto indexing addressing mode, each time the CPU accesses a memory location, it automatically increases or decrements a register (such the address register). As a result, memory regions can be accessed in a sequential manner without needing to be specifically mentioned in each instruction. Auto indexing is useful for tasks that require effectively accessing data in a sequential fashion or looping over memory regions.   
  
These addressing modalities offer efficiency and flexibility in data access and command execution. With the advantages of each mode and its suitability for various activities, the system can successfully manage a wide range of jobs.

**List of Instructions:**

|  |  |  |
| --- | --- | --- |
| **Instructions** | **Hex**  **DIRECT, INDIRECT, IMMEDIATE, AUTO INDEXING** | **Explanation:** |
| LDA | 00XXX,10XXX,20XXX,30XXX | Load the instruction from memory to accumulator |
| STD | 01XXX,11XXX, NOT, 31XXX | Store the operand in the memory |
| ADD | 02XXX,12XXX,22XXX,32XX | Add accumulator and data register |
| AND | 03XXX,13XXX,23XXX,33XXX | AND the accumulator and data register |
| SUB | 04XXX,14XXX,24XXX,34XXX | Subtract the accumulator and data register |
| ISZ | 05XXX,15XXX, NOT,35XXX | Skip next instruction if it is zero |
| MULT | 06XXX,16XXX,26XXX,36XXX | Multiplication of accumulator and data register |
| OR | 07XXX,17XXX,27XXX,37XXX | OR the accumulator and data register |
| NOR | 08XXX,18XXX,28XXX,38XXX | NOR the accumulator and data register |
| NAND | 09XXX,19XXX,29XXX,39XXX | NAND the accumulator and data register |
| BUN | 0AXXX,1AXXX, NOT,3AXXX | Jumps the program to a specific memory address |
| BSA | 0BXXX,1AXXX, NOT ,3BXXX | Jumps the program to specific memory address and stores the return address |
| SKP | 0CXXX,1CXXX,2CXXX,3CXXX | Skip if the content in the memory and ac are same |
| STM | 0DXXX,1DXXX, NOT,3DXXX | Store the multiplication result |
| INCA | 0100 | Increment the accumulator |
| CLA | 0101 | Clear the accumulator |
| CLE | 0102 | Clear the extended accumulator |
| CPC | 0103 | Clear the program counter |
| CMA | 0104 | Complement accumulator |
| CME | 0105 | Complement extended accumulator |
| CIR | 0106 | Circular right shift accumulator and E |
| CIL | 0107 | Circular left shift with accumulator and E |
| DECA | 0108 | Decrement accumulator |
| DPC | 0109 | Decrement program counter |
| SPA | 010A | Skip next instruction if accumulator is positive |
| SNA | 010B | Skip next instruction if accumulator is negative |
| SZA | 010C | Skip next instruction if accumulator is zero |
| SZE | 010D | Skip next instruction if extended accumulator is zero |
| DECP | 010E | Decrement program counter |
| HALT | 010F | Halt the program |
| INPT | 0000 | Input in input register |
| OUTP | 0001 | Output in output register |
| IONN | 0002 | Interrupt ON |
| IOFF | 0003 | Interrupt off |
| SKPI | 0004 | Skip when input flag is on |
| SKPO | 0005 | Skip when output flag is on |

**Memory reference instructions:**

Memory reference instructions are used to store and manipulate data in RAM’s memory locations.

The Innovation in this instruction is multiplication.

|  |  |  |
| --- | --- | --- |
| **Instructions** | **OPCODE:** | **Execution Microoperations** |
| LDA | 0000 | AR ← M[ADDR], PC←PC+1  AC ← AR , SF←0 |
| STD | 0001 | M[ADDR] ← AC, SF←0 |
| ADD | 0010 | AC ← DR + AC, SF←0 |
| AND | 0011 | AC ← DR^AC, SF←0 |
| SUB | 0100 | AC←AC-DR, SF←0 |
| ISZ | 0101 | DR←DR+1  PC←PC+1  M[ADDR] ← DR, SF←0 |
| MULT | 0110 | MM ← DR \* AC, SF←0 |
| OR | 0111 | AC ← DR ∨ AC, SF←0 |
| NOR | 1000 | AC ← DR ∨ AC, SF←0 |
| NAND | 1001 | AC ← (DR ∧ AC)’, SF←0 |
| BUN | 1010 | PC ← AR, SF←0 |
| BSA | 1011 | M[ADDR] ← PC  AR ←AR+1  PC ← AR, SF ← 0 |
| SKP | 1100 | DR← M[ADDR]  IF(DR=AC) PC←PC+1,SF←0 |
| STM | 1101 | M[ADDR] ← MPR [0:15]  AR ← AR+1  M[ADDR] ← MPR [16:31], SF←0 |

**Register reference Instructions.**

Register reference instructions are used to store manipulated data in registers.

|  |  |  |
| --- | --- | --- |
| **Instructions** | **HEX CODE:** | **Execution Microoperations** |
| INCA | 0100 | AC←AC+1 |
| CLA | 0101 | AC←0 |
| CLE | 0102 | E←0 |
| CPC | 0103 | PC←0 |
| CMA | 0104 | AC←AC’ |
| CME | 0105 | E←E’ |
| CIR | 0106 | E←AC[0], AC←shr(AC), AC[15] ←E |
| CIL | 0107 | E←AC[15], AC←shl(AC), AC[0] ←E |
| DECA | 0108 | AC←AC-1 |
| DPC | 0109 | PC←PC-1 |
| SPA | 010A | If (AC[15]=0) :PC←PC+1 |
| SNA | 010B | If(AC[15]=1):PC←PC+1 |
| SZA | 010C | If (AC=0) :PC←PC+1 |
| SZE | 010D | If (E=0) :PC←PC+1 |
| DECP | 010E |  |
| HALT | 010F | S←0 |

**Input/Output instructions:**

Input/Output instructions are used to input bits and display output bits.

|  |  |  |
| --- | --- | --- |
| **Instructions** | **Hex code:** | **Execution microoperations** |
| INPT | 0000 | AC[0:7]←INPR, INFL←0 |
| OUTP | 0001 | OUTR←AC[0:7], OTFL←0 |
| IONN | 0002 | IEN←1 |
| IOFF | 0003 | IEN←0 |
| SKPI | 0004 | If (INFL=0) PC←PC+1 |
| SKPO | 0005 | If (OTFL=0) PC←PC+1 |

**ALU**

The Arithmetic Logic Unit (ALU) is like the brain of a computer's processor. The work of ALU is to store and run the instruction. It can add, subtract, multiply, and compare numbers, as well as perform logical operations like AND, OR, and NOT.

When you give a command to the computer, like asking it to add two numbers, the ALU springs into action. It takes those numbers from the memory or registers, performs the requested operation, and then delivers the result back to where it's needed.

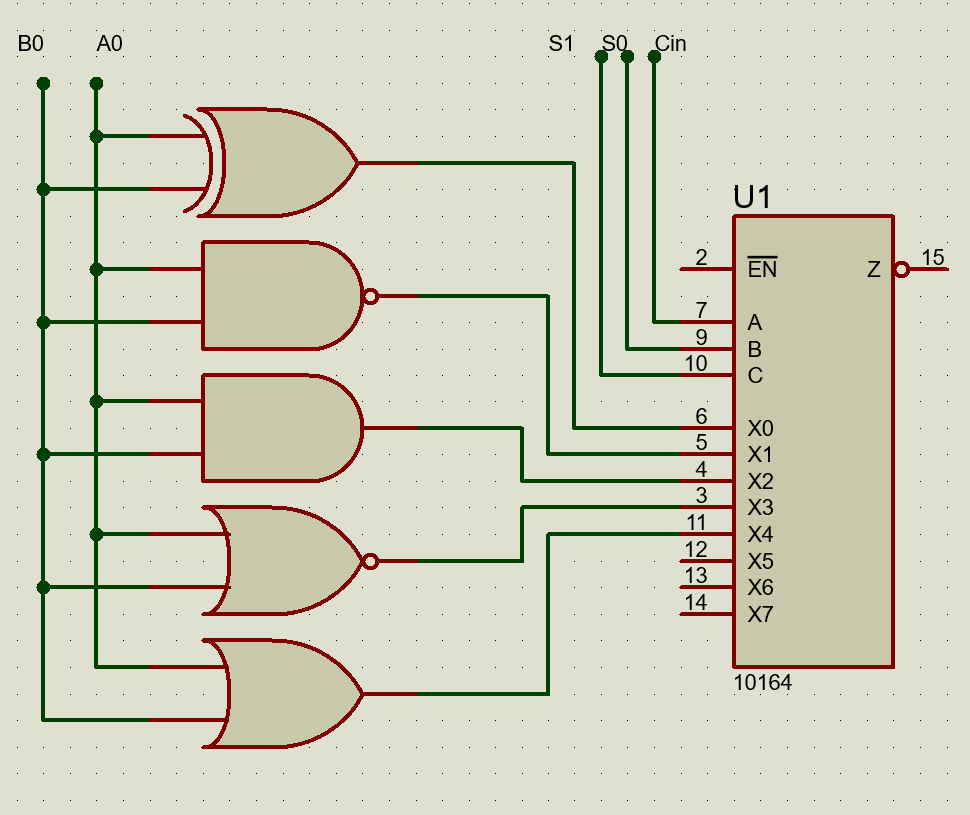
A diagram of a logic unit

Description automatically generated

A computer screen shot of a circuit board

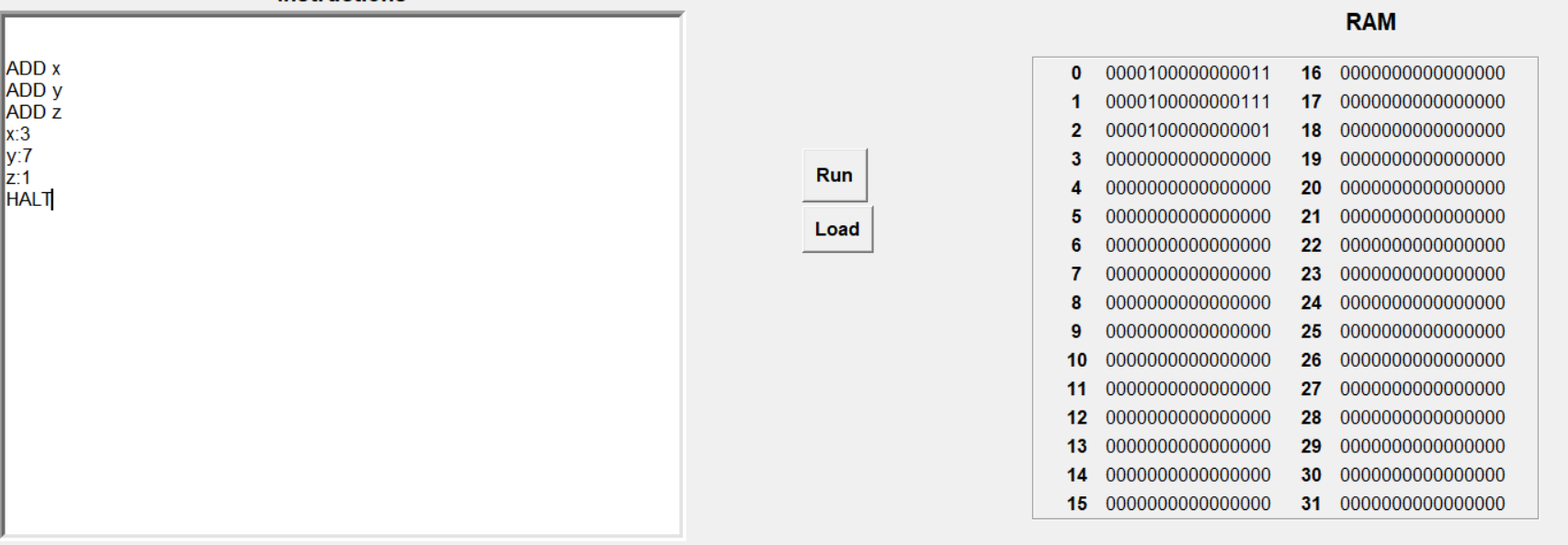
Description automatically generated

**LOGIC UNIT**

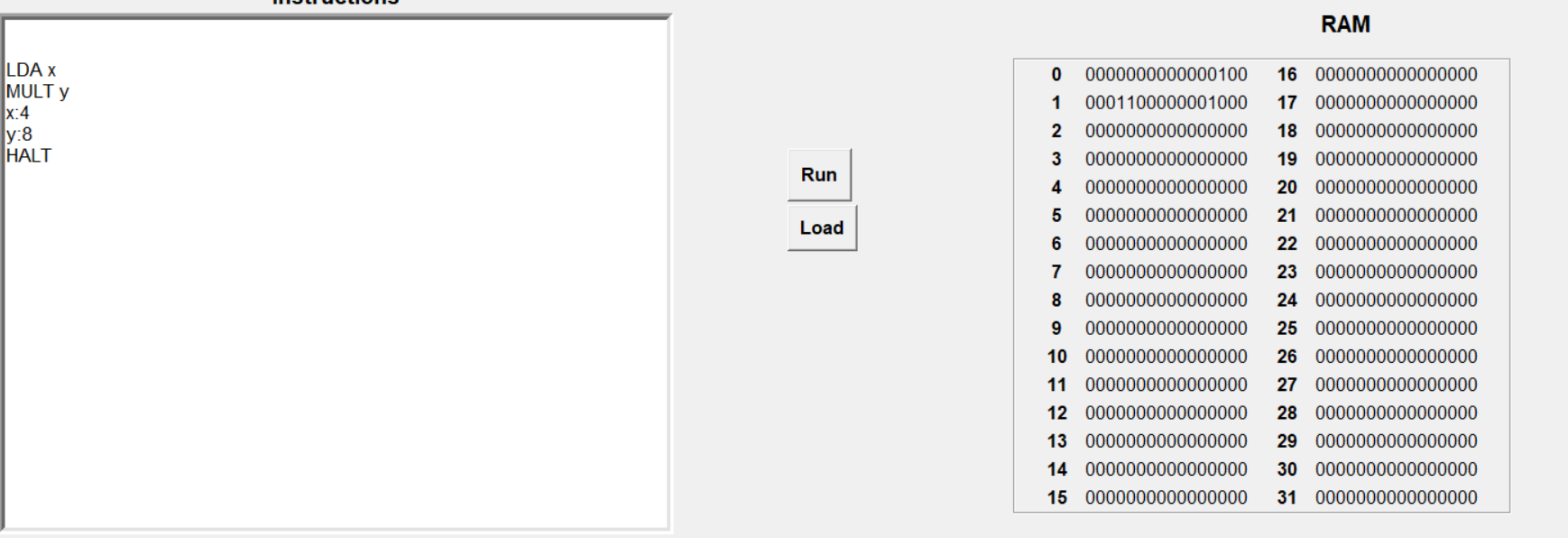


**Sample assembly programs:**

Program for addition:



Program for Multiplication:



**References:**

Computer Organization & Architecture by William Stallings