Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report
on
"AD-18"

[Code No.: COMP 315]

(For partial fulfillment of Year III / Semester I in Computer Engineering)

Submitted by
Dipen Khatri (23)
Asim Shrestha (48)

Submitted to
Dhiraj Shrestha
Department of Computer Science and Engineering

Jan 14, 2024

Acknowledgement

We would like to express our heartfelt appreciation to Mr. Dhiraj Shrestha for his inspiring guidance and continuous encouragement throughout the development of our computer design project. His support has been pivotal in helping us effectively apply our knowledge of a ”Basic Computer.” Our thanks also extend to all those who provided us with the opportunity to complete this project.
Engaging in this endeavor has deepened our understanding of essential computer components such as memory, registers, and flip-flops. A special acknowledgment is reserved for the entire university and the Department of Computer Science and Engineering (DOCSE) for granting us the space and resources to undertake this project. We are grateful for the backing of the academic community, which has played a vital role in our achievements.

Abstract

This report presents a thorough exploration of computer architecture through the lens of the AD-18 project, showcasing the development and implementation of a sophisticated 18-bit Central Processing Unit (CPU) with an 8KB memory. The AD-18 project stands out for its innovative design principles, featuring 18-bit registers, a robust common bus system, and a diverse set of instructions. The CPU's instruction format, encapsulating 13 bits for address selection, 4 bits for opcode, and 1 bit for addressing modes, serves as a testament to the essence of computer system architecture. The project leverages Logisim for manual simulation, providing a dynamic platform for testing and refining the CPU's behavior during instruction fetch and execution.

Keywords: *AD-18, Computer Architecture, CPU Design, 18-bit System, Logisim Simulation*

Contents

List of Tables

List of Figures

Abbreviations

ALU Artithmetic and Logic Unit. 8

AR Address Register. 5, 20

CU Control Unit. 13

DR Data Register. 5

IR Instruction Register. 5

MRI Memory Reference Instruction. 15

MU Memory Unit. 10

OPCODE Operation Code. 3

OUTR Output Register. 6

PC Program Counter. 5, 20

RRI Register Reference Instruction. 16

TR Temporary Register. 6

Chapter 1: Introduction

Computer architecture defines how computer system and programs operate. In this complicated realm of computer engineering, the design and architecture of a computer system play pivotal roles in defining its functionality, organization, and ultimate performance. The evolutionary path of computer architecture traces back to two pivotal paradigms: the Von Neumann Architecture, and the Harvard Architecture. These two architectures stand as the bedrock, influencing and shaping the design principles of all modern computer systems.

Computer architecture serves as the blueprint that outlines the systematic design, functionality, and organizational structures of a computer system. Organization of a computer is defined by its internal register, the timing and control structures, and the set of instruction that it uses. At its core, the internal architecture is shaped by logic gates, forming connections between registers, memory, and buses.

The heart of any computer system lies within its Central Processing Unit (CPU), a hardware marvel that executes the instructions of a computer program. This execution encompasses arithmetical, logical, and input/output operations, forming the backbone of a computer's functionality. Designing a CPU is both a complex and indispensable process, requiring a careful synthesis of diverse ideas from computer system architecture.

Embarking on the exploration of the AD-18 computer, this report offers a comprehensive journey into its design considerations, components, and architectural nuances. With an 8KB memory and an 18-bit word size, the AD-18 redefines the landscape of computing, promising a harmonious blend of technological prowess and innovative design.

1.1 AD-18 Architecture

The Central Processing Unit (CPU) stands as the hardware linchpin within a computer, orchestrating the execution of a computer program through fundamental arithmetical, logical, and input/output operations. At the heart of computer organization lies a meticulous interplay of internal registers, timing and control structures, and a set of instructions that collectively define a system's architecture. The intricate process of designing a CPU not only embodies complexity but is also an indispensable step, harmonizing theoretical knowledge with practical implementation.

In the case of the AD-18 computer, the pursuit of this intricate design takes the form of an 18-bit CPU, a significant endeavor translating theoretical principles into

tangible functionality. This computational marvel is meticulously crafted with a memory configuration of 8K*18, featuring 8192 words—each 18 bits long. The memory, with its 13-bit address lines, allows for the selection of words, showcasing the fusion of theoretical concepts into a practical design.
The AD-18's instruction format reflects a thoughtful organization:

- 13 bits for address selection
- 4 bits for opcode
- 1 bit for addressing modes

This format, encapsulated in the opcode and address bits, delineates the pathways for the CPU to execute a diverse range of instructions, embodying the essence of computer system architecture.
The distinctive characteristics of our AD-18 computer include:

- 18-bit registers, forming the foundation for data storage and manipulation
- An 18-bit common bus system, facilitating seamless communication between components
- 13-bit address lines, enabling precise memory location selection
- A repertoire of 10 Memory Reference Instructions, providing access to stored data
- 13 Register Reference Instructions, facilitating efficient register operations
- 6 Input-Output Instructions, bridging the gap between the internal system and external devices.

In the chapters that follow, we embark on a detailed exploration of the AD-18's architecture, unraveling its design considerations, components, and the nuanced interplay that positions it as a testament to the synthesis of theoretical acumen and practical ingenuity in the ever-evolving landscape of computer engineering.

Chapter 2: Design Considerations

This part explains how we designed our computer, what's inside it, and the set of instructions it can understand.

2.1 Instruction Format

Instruction formats refer to the way instructions are encoded and represented in machine language. A computer instruction is a binary code that specifies a sequence of micro-operations for the computer. Instruction codes together with data are stored in memory. The computer reads each instruction from memory and places it in a control register.An instruction code is a group of bits that instruct the computer to perform a specific operation. It is usually divided into parts, each having its own particular interpretation. They are:

- **Operation Code (OPCODE) :** The Operation Code (OPCODE) instructs the processor on a specific operation to perform. It serves as the command telling the CPU what task to execute.
- **Address :** This part defines how data is represented in the instruction. It can either indicate that the data is in the computer's memory or stored in the CPU's register.
- **Addressing Modes :** Addressing modes specify how the operands (data or addresses of data) are determined for the operation. It describes the way in which the operands are specified in the instruction. Depending on the addressing mode, the operands might be immediate values (constants), values in registers, or values stored in memory locations. The addressing mode provides flexibility in how operands are specified in instructions.

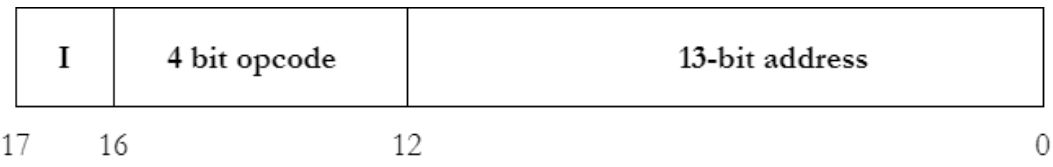

Figure 2.1: Instruction Format

In our basic computer design, with 8K*18-bit memory, we allocate 13 bits for addressing, 1 bit for addressing modes, and 4 bits for opcodes in the instruction format. This allows us to specify memory locations, indicate addressing modes, and command specific operations efficiently.

2.2 Addressing Modes

There are two addressing modes in our computer:

- Direct Addressing **(I=0)**
  In simple terms, in direct addressing mode, the instruction tells the computer to look in a specific memory location for the data it needs. The address of that memory location is stored within the instruction itself.

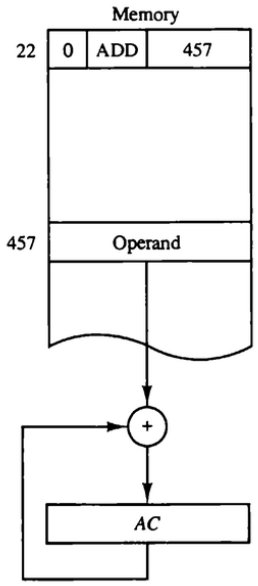

Figure 2.2: Direct Addressing

- Indirect Addressing **(I=1)**
  In the indirect addressing mode, the instruction directly contains the address of the operand. In other words, the instruction itself holds the information about where to find the data.

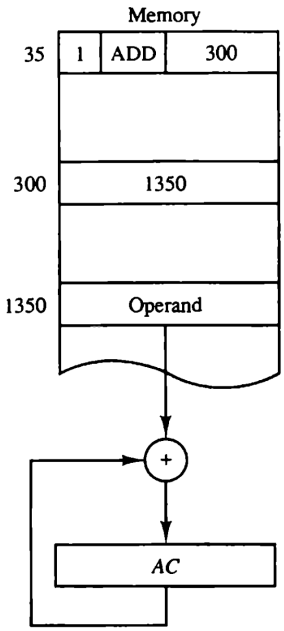

Figure 2.3: Indirect Addressing

2.3 Components

2.3.1 Registers

- **Instruction Register (IR)**
  IR stores 18 bits of instructions from memory. During each instruction cycle, the Program Counter's address goes to the AR, and the content of that memory location is moved to the IR through the Common Bus System . The IR is a register specifically for holding the instruction code.

- **Address Register (AR)**
  AR is of size of 13 bits. It holds main memory addresses for data and instructions, or a portion of the address used in calculating complete addresses. It is responsible for storing the memory location of the data or instruction that the computer needs to access.

- **Program Counter (PC)**
  PC is of size of 13 bits. It stores the address of the next instruction to be executed. It keeps the track of memory address of the next instruction to be executed in a program.

- **Data Register (DR)**
  DR is of size of 18 bits. It is a storage unit that temporarily holds data during

processing in a computer. It stores information actively used by the CPU for calculations or operations.

- **Accumulator**
  Accumulator is of size of 18 bits. It stores the result after each computation. It receives input from the ALU's output. The accumulator's input isn't directly connected to the Common Bus System (CBS). The accumulator's output goes to the Common Bus System and serves as the first operand for the ALU, with the second operand being the content of the Data Register.

- **Output Register (OUTR)**
  OUTR is of 8 bits and is responsible for delivering computer output to the external world. It receives its output from the Common Bus System, usually sourced from the accumulator. The content of the Output Register is then connected to an external LED display.

- **Temporary Register (TR)**
  TR is of 18 bits and are short-term storage locations within a computer system used for holding intermediate data during various processing stages. These registers facilitate efficient data manipulation and transfer within the CPU, playing a crucial role in temporary storage and retrieval of information during computational tasks.

Register	No of Bits	Description
DR	18	Data Register
AR	13	Address Register
AC	18	Accumulator
IC	18	Instruction Register
PC	13	Program Counter
TR	18	Temporary Register
INPR	8	Input Register
OUTR	8	Output Register

Table 2.1: Registers and their bit sizes in AD Computer

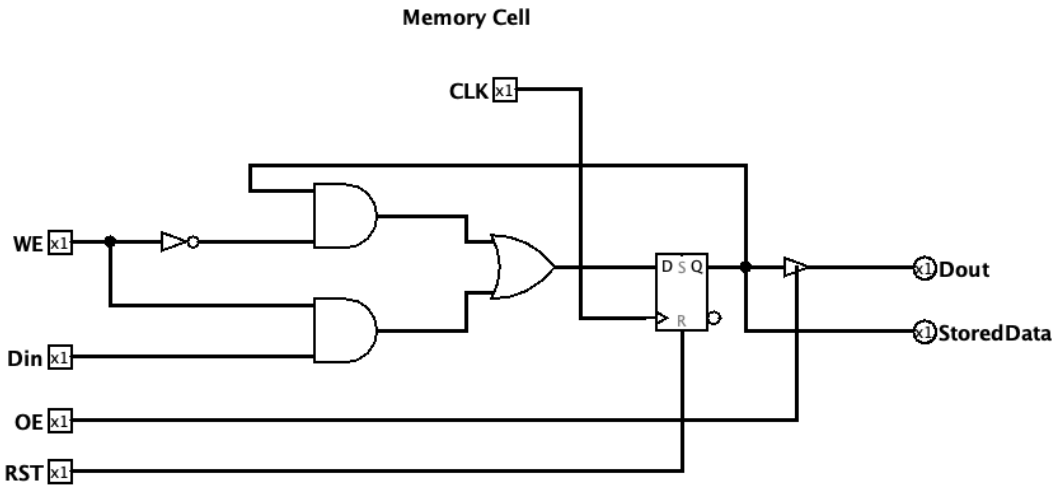

Figure 2.4: 1 bit Memory Cell

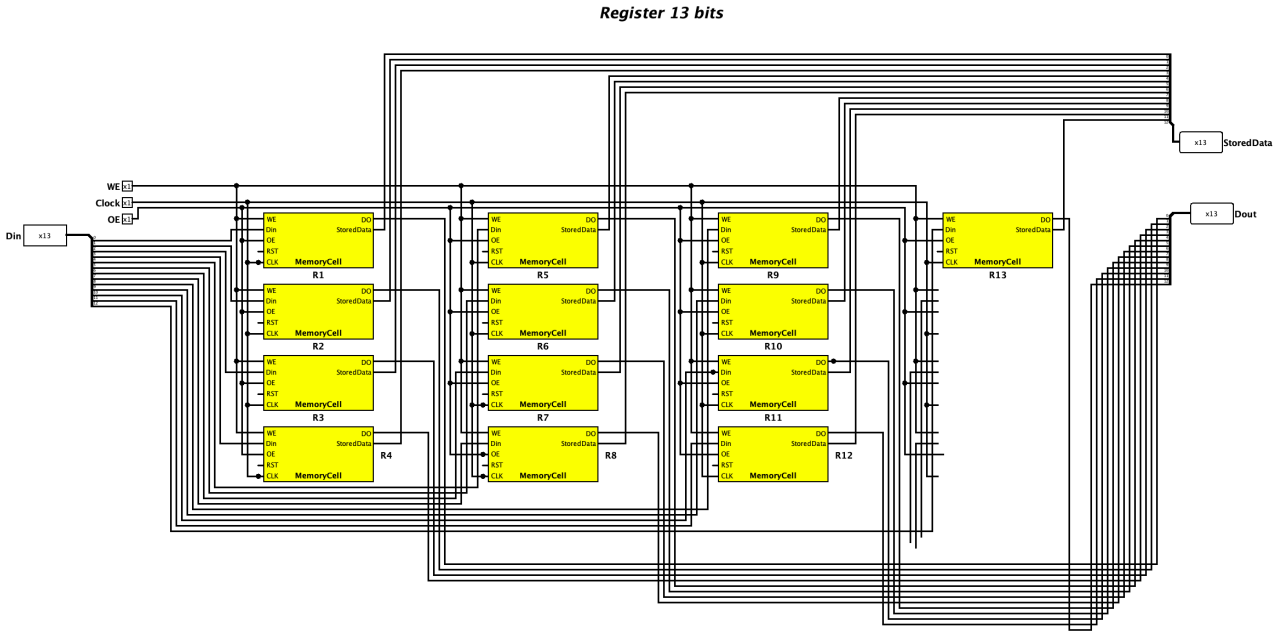

Figure 2.5: 13-bit Register

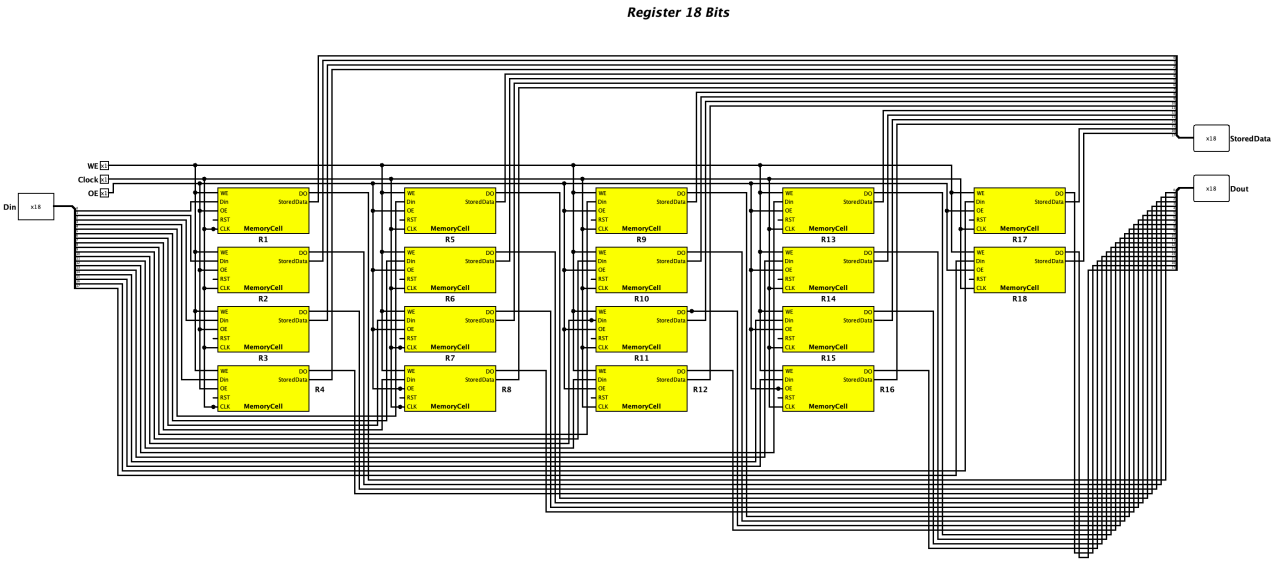

Figure 2.6: 18-bit Register

2.3.2 Artithmetic and Logic Unit (ALU)

ALU is of 18 bits and its main function is to take binary inputs, execute the operation, and create, store, and distribute binary output. ALU make arithmetic and logical calculations.

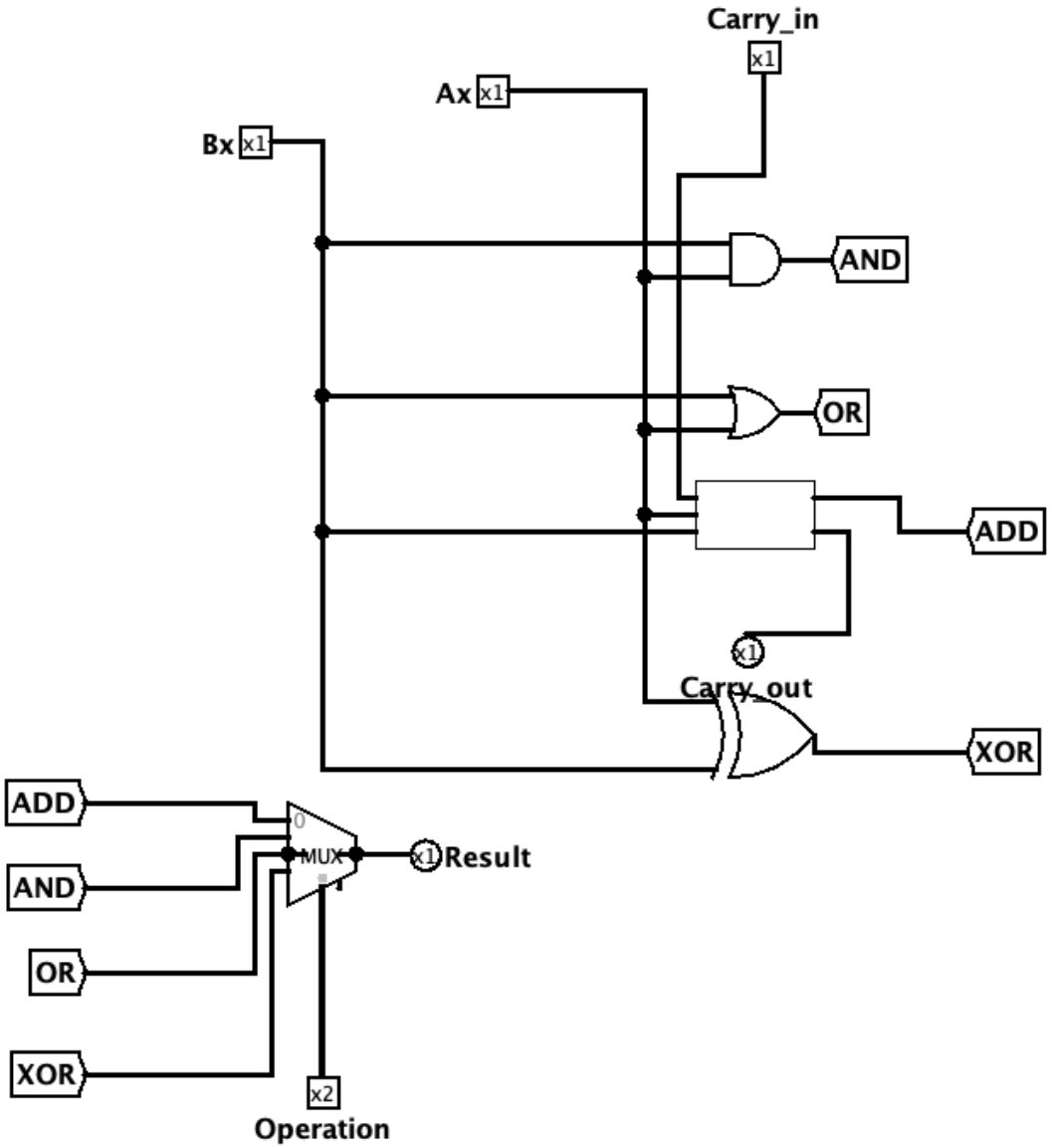

Figure 2.7: 1 bit ALU

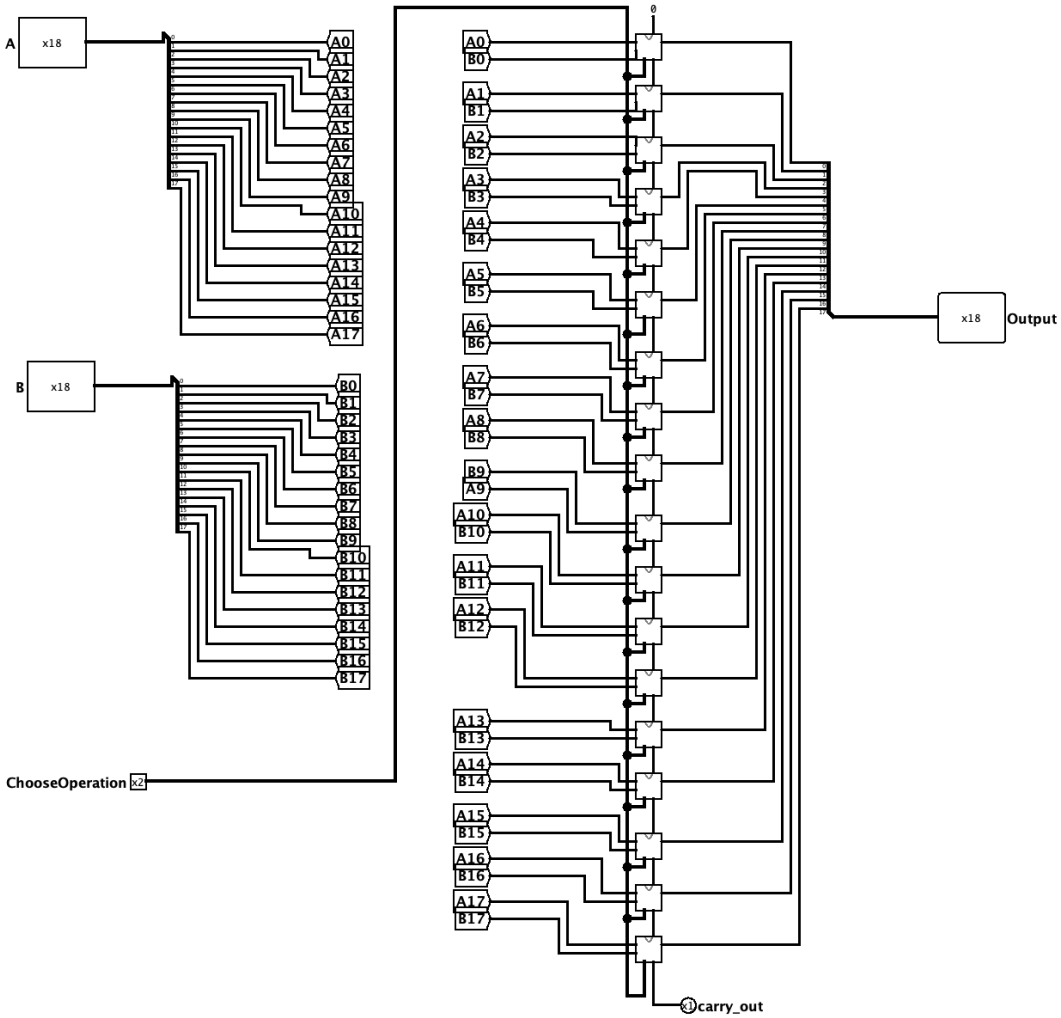

Figure 2.8: 18 bit ALU

2.3.3 Memory Unit (MU)

MU in the AD-18 computer is a pivotal component for storing and retrieving instructions and data. With a memory size of 8K*18, offering 8192 words of 18 bits each, it serves as a fundamental element in the computer's architecture. The memory basics involve organizing information into 18-bit words, with each word comprising bytes. The AD-18 memory configuration demands a 13-bit address line for precise word selection, with the Address Register (AR) facilitating the interaction, and the Common Bus System (CBS) managing data transfer.

2.3.4 Common Bus System

In the AD-18 computer, the Common Bus System (CBS) is like a central hub connecting important parts through an 18-bit pathway. This includes Memory, Accu-

mulator, Address Register, Program Counter, Instruction Register, Temporary Register, and Data Register, each having an 18-bit slot. The Data Register talks directly to the Arithmetic Logic Unit (ALU), creating a vital link with the control unit and the 18-bit Accumulator.

The CBS smartly handles the sharing of information among these registers and the memory. Binary values of signals S2, S1, and S0 decide which part gets to communicate on the 18-bit road. Each part has its spot on the road, determined by a number. When the clock ticks, the part with the green light (Load input enabled) gets the data. If it's Memory's turn, it shares when asked nicely (write input activated), or it puts its info on the road when someone wants to know (read input activated). This organized system ensures smooth communication and teamwork among components, making the AD-18 computer run seamlessly.

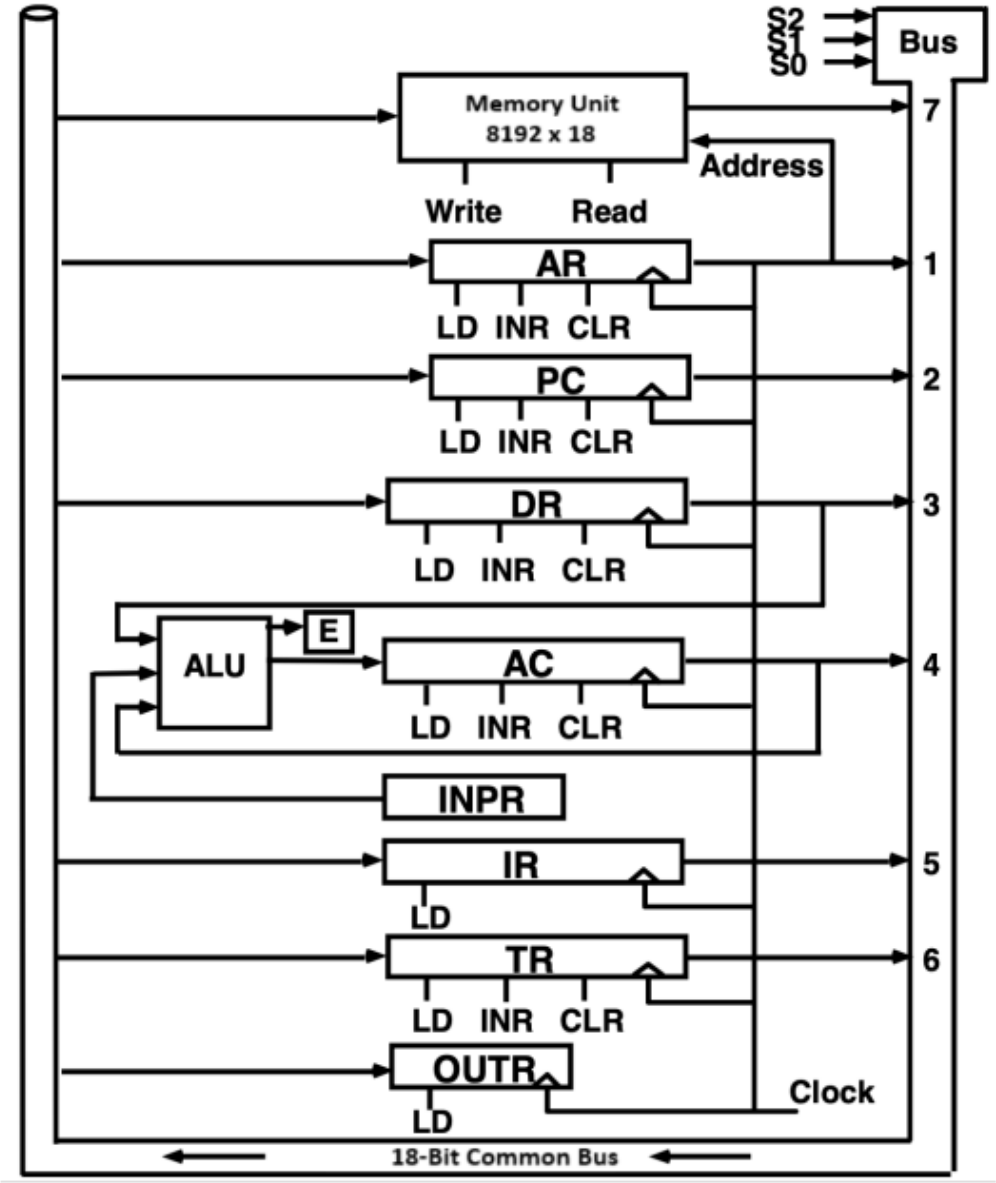

Figure 2.9: Common Bus System

2.3.5 Flags/Flip Flops

Flip Flop	Description
S	Start Stop Flip Flop
R	Interrupt Enable
E	End Around Carry
IEN	Interrupt Enable
FGI	Input Flag
FGO	Output Flag

Table 2.2: Flip-Flops

2.4 Control Unit

Control Unit (CU) refers to the circuitry embedded within a computer's processor responsible for overseeing and coordinating operations. The control signals are generated in the Control Unit (CU) and provide control inputs for the multiplexers in the common bus, control inputs in processor registers, and micro-operations for the accumulator. There are two major types of control organization: hardwired control and micro-programmed control. In the hardwired organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits. It has the advantage that ti can be optimized to produce a fast mode of operation. In the micro-programmed organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of micro-operations.

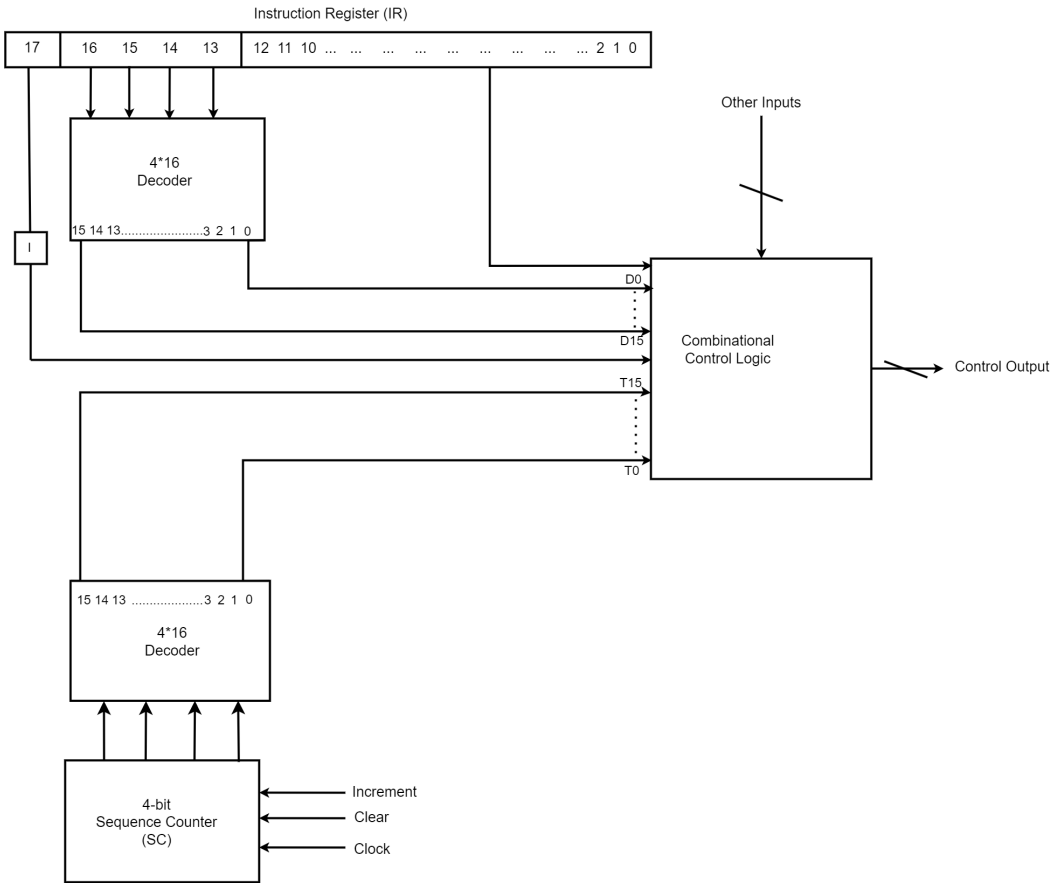

Figure 2.10: Control Unit

The block diagram of the control unit is shown in Fig. 2.10. It consists of two decoders, a sequence counter, and a number of control logic gates. An instruction read from memory is placed in the instruction register (IR). The position of this register in the common bus system is indicated in Fig. 2.9. The instruction register is shown again in Fig. 2.10, where it is divided into three parts: the I bit, the operation code, and bits 0 through 12. The operation code in bits 13 through 16 are decoded with a 4 x 16 decoder. The sixteen outputs of the decoder are designated by the symbols D0 through D15. The sub scripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 17 of the instruction is transferred to a flip-flop designated by the symbol .Bits 0 to 12 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T0 through T15

2.5 Instruction Type and Format

2.5.1 Memory Reference Instruction

The instruction code format for the Memory Reference Instruction (MRI) is shown below:

- Direct Addressing (**I=0**)

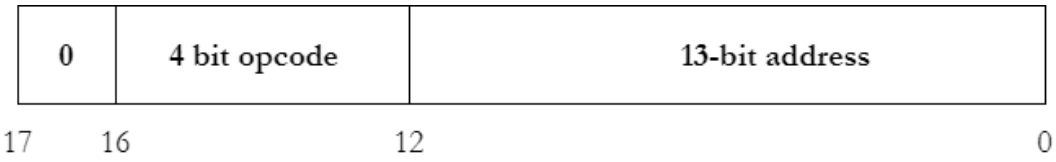

Figure 2.11: Direct Addressing

- Indirect Addressing (**I=1**)

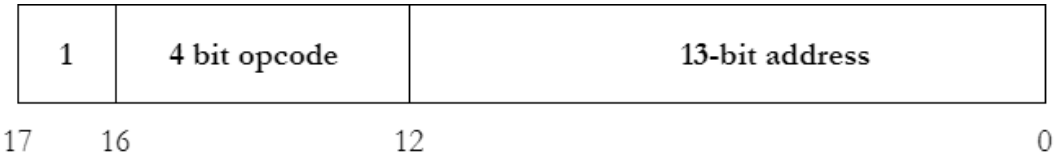

Figure 2.12: Indirect Addressing

MRI load values into and store values from the general registers. In our design, we have the following memory reference instructions:

Symbol	BITS	Description
ADD	I 0000 xxxxxxxxxxxxx	ADD Memory word to AC
SUB	I 0001 xxxxxxxxxxxxx	Subtract memory word from AC
INC	I 0010 xxxxxxxxxxxxx	Increment the value in a memory location
DEC	I 0011 xxxxxxxxxxxxx	Increment the value in a memory location
AND	I 0100 xxxxxxxxxxxxx	AND Memory word to AC
LDA	I 0101 xxxxxxxxxxxxx	Load the immediate value into the AR
STA	I 0110 xxxxxxxxxxxxx	Store the value from the AR into memory.
CML	I 0111 xxxxxxxxxxxxx	Complement the value
OR	I 1000 xxxxxxxxxxxxx	OR to AC
XOR	I 1001 xxxxxxxxxxxxx	XOR to AC
JMP	I 1010 xxxxxxxxxxxxx	Unconditional Jump to memory address
JZ	I 1011 xxxxxxxxxxxxx	Jump if Zero
JN	I 1100 xxxxxxxxxxxxx	Jump if Negative
SHL	I 1101 xxxxxxxxxxxxx	Shift the bits in the AC to the left
SHR	I 1110 xxxxxxxxxxxxx	Shift the bits in the AC to the Right

Table 2.3: Memory Reference Instruction Set Table

2.5.2 Register Refrence Instruction

Register Reference Instruction (RRI) works with register only. In the AD computer, the MSB of the register reference instructions is always equal to 0 i.e I = 0 and at the same time the opcode is always 1111 while the remaining address bits change for each instruction.

The instruction code format for the RRI is shown below:

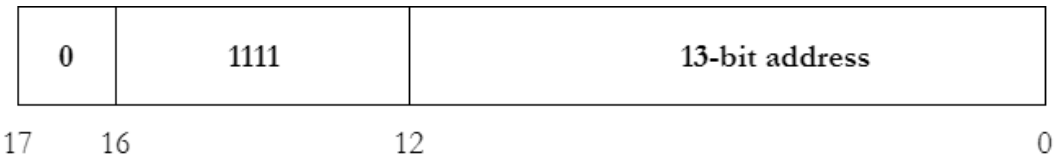

Figure 2.13: Register Reference Instruction

Instructions	Instruction Format	Instruction Description
CLA	0 1111 1000000000000	Clear AC
CLE	0 1111 0100000000000	Clear E
CMA	0 1111 0010000000000	Complement AC
CME	0 1111 0001000000000	Complement E
CIR	0 1111 0000100000000	Circulate right
CIL	0 1111 0000010000000	Circulate left
INC	0 1111 0000001000000	Increment AC
DEC	0 1111 0000000100000	Decrement AC
SPA	0 1111 0000000010000	Skip if positive
SNA	0 1111 0000000001000	Skip if negative
SZA	0 1111 0000000000100	Skip if AC=0
SZE	0 1111 0000000000010	Skip if E=0
HLT	0 1111 0000000000001	Halt the program

Table 2.4: Register reference Instruction Set Table

2.5.3 Input Output Instructions

In the AD computer, the MSB of the register reference instructions is always equal to 1 i.e I = 1 and at the same time the opcode is always 1111 while the remaining address bits change for each instruction
The instruction code format is shown below:

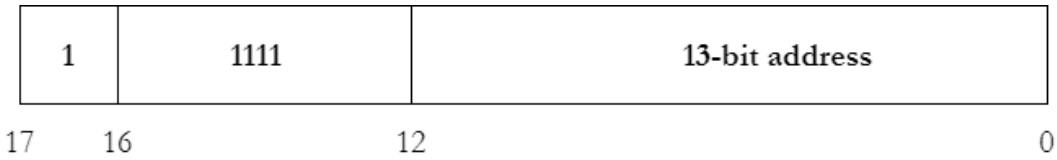

Figure 2.14: Input Output Instruction

Instructions	Instruction Format	Instruction Description
INP	1 1111 1000000000000	Input character to AC
OUT	1 1111 0100000000000	Output character from AC
SKI	1 1111 0010000000000	Skip on Input Flag
SKO	1 1111 0001000000000	Skip on Output Flag
ION	1 1111 0000100000000	Interrupt enable on
IOF	1 1111 0000010000000	Interrupt enable off

Table 2.5: Input Output Instruction Set Table

Chapter 3: Design of Individual Components

To create the individual components of our computer, it is essential to identify and verify the control functions and micro-operations. These micro-operations are executed within specific instruction cycles, and each of these cycles is detailed as follows:

3.1 Instruction Cycle

The instruction cycle is a key process in computer architecture, guiding the sequential steps a computer takes to execute a program. It involves fetching instructions from memory, decoding them, and executing the corresponding operations. The program counter determines the instruction address, initiating the cycle. Additional steps, like reading addresses from memory or executing specific operations, depend on the instruction type. A machine instruction is executed in the following phases:

- **Fetch** an instruction from memory.
- **Decode** the instruction.
- **Read** the effective address from memory if the instruction has an indirect address.
- **Execute** the instruction.

3.1.1 Fetch Cycle:

In this initial phase, the program counter retrieves the next instruction's memory address. The instruction is then fetched from memory and placed in the instruction register.
T_0: AR $\leftarrow$ PC
T_1: IR $\leftarrow$ M [AR], PC $\leftarrow$ PC+1

3.1.2 Decode Cycle:

Following fetching, the instruction undergoes decoding. The control unit interprets the opcode, determining the type of operation to be performed. It may involve deciphering addressing modes and extracting operands.
T_2: I $\leftarrow$ IR[17],R'T_2: (D0- D15) $\leftarrow$Decode(IR[13-16]),
AR $\leftarrow$ IR[0-12]

Determining Type of Instruction

The timing signal that si active after the decoding is T,. During time T, the control unit determines the type of instruction that was just read from memory.

Memory Reference Instruction:

- **Direct Addressing**
  $D'_{15}I'T_3$: NOP
- **Indirect Addressing**
  $D'_{15}IT_3$: AR ← M[AR]

$D_{15}I'T_3$: Register Reference Instruction

$D_{15}IT_3$: Input-Output Reference Instruction

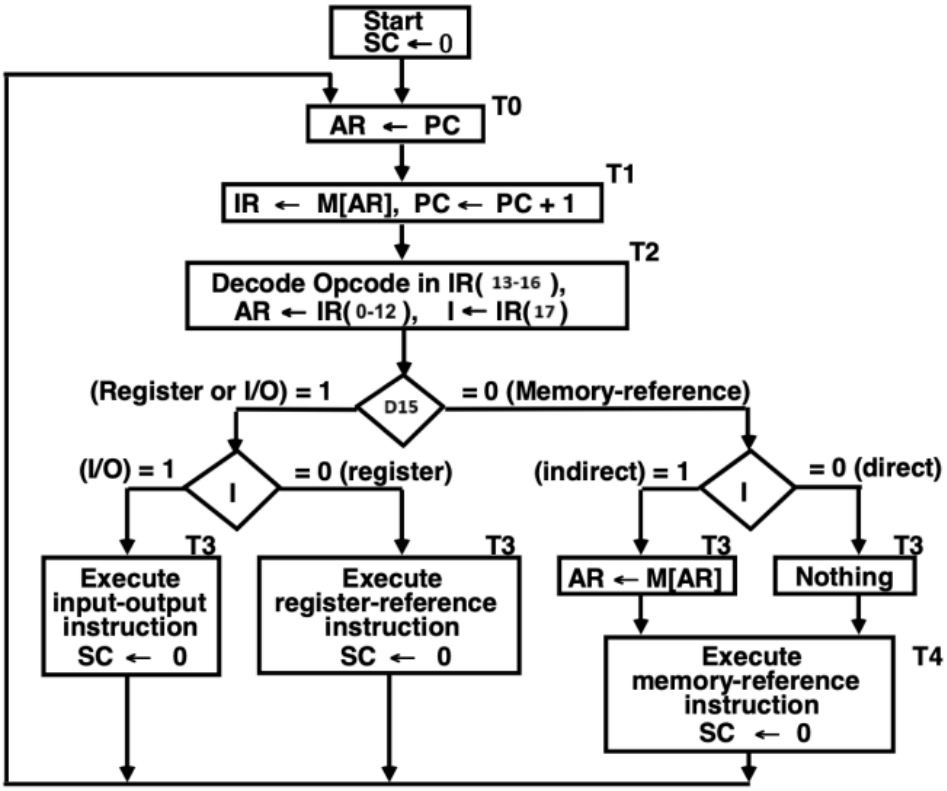

Figure 3.1: Determine Type of Instruction

3.1.3 Execute Cycle:

Once the instruction is decoded, the actual operation or computation takes place. This phase executes the instruction, manipulating data in registers or memory, and may involve arithmetic or logical operations.

Memory Reference Instruction

Instruction Type	RTL Instructions
ADD	D_0T_4: DR ←M[AR] D_0T_5: AC←AC + DR, E←Cout, SC←0
SUB	D_1T_4: DR ← M[AR] D_1T_5: AC ← AC - DR; SC ← 0
INC	D_2T_4: DR ← M[AR] D_2T_5: DR ← DR + 1 D_2T_6: M[AR] ← DR; SC ← 0
BSA	D_3T_4: M[AR] ← PC D_3T_5: AR ← AC + 1 D_3T_6: PC ← AR; SC ← 0
AND	D_4T_4: DR ←M[AR] D_4T_5: AC←AC ^ DR, SC←0
LDA	D_5T_4: DR ←M[AR] D_5T_5: AC←DR, SC←0
STA	D_6T_4: M[AR], ←AC, SC←0
OR	D_7T_4: DR ← M[AR] D_7T_5: AC ← AC | DR, SC ← 0
XOR	D_8T_4: DR ← M[AR] D_8T_5: AC ← AC ^ DR, SC ← 0
BUN	D_9T_4: PC ← AR, SC ← 0
ISZ	$D_{10}T_4$: DR ← M[AR] $D_{10}T_5$: DR ← DR + 1 $D_{10}T_6$: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0
CLM	$D_{11}T_4$: DR ← 0 $D_{11}T_5$: M[AR] ← DR, SC ← 0
BZ	$D_{12}T_4$: DR ← M[AR]; AR ← AR + 1 $D_{12}T_5$: if DR = 0 then PC ← AR, SC ← 0
XCZ	D_{13} T_4: DR ← M[AR] D_{13} T_5: M[AR] ← AC D_{13} T_6: AC ← DR; SC ← 0
CMP	D_{14} T_4: DR ← M[AR] D_{14} T_5: AC ← DR D_{14} T_6: AC ← AC',SC ← 0

Table 3.1: Memory Reference Instructions of AD Computer

Register Reference Instruction

r=$D_{15}I^{'}T_3$: Register Reference Instruction

Register Reference	
HLT	rB_0: S ← 0; SC ←0
SZE	rB_1: if (E=0) then (PC ← PC + 1); SC ←0
SZA	rB_2: if (AC=0) then (PC ← PC + 1); SC ←0
SNA	rB_3: if (AC[17]=1) then (PC ← PC + 1); SC ←0
SPA	rB_4: if (AC[17]=0) then (PC ← PC + 1): SC ←0
DEC	rB_5: AC ← AC - 1; SC ←0
INC	rB_6: AC ← AC + 1; SC ←0
CIL	rB_7: AC ← SHL AC; AC[0] ← E; E ← AC[17]; SC ←0
CIR	rB_8: AC ← SHR AC; AC[17] ← E; E ← AC[0]; SC ←0
CME	rB_9: E ← E'; SC ←0
CMA	rB_{10}: AC ← AC'; SC ←0
CLE	rB_{11}: E ← 0; SC ←0
CLA	rB_{12}: AC ← 0; SC ←0

Table 3.2: Register Reference Instructions of AD Computer

Input-Output Instructions

p=$D_{15}IT_3$

I/O Reference	
IOF	pB_7: IEN ← 0
IEN	pB_8: IEN ← 1
SKO	pB_9: if (FGO=1) then (PC ← PC + 1)
SKI	pB_{10}: if (FGI=1) then (PC ← PC + 1)
OUT	pB_{11}: OUTR ← AC(0-7); FGO ← 0
INP	pB_{12}: AC(0-7) ← INPR; FGI ← 0

Table 3.3: Input Output Operations

3.2 Interrupt Cycle

An Interrupt is a signal emitted by an I/O device attached to a computer or from a program within the computer

$T_0'T_1'T_2'(\text{IEN})(\text{FGI} + \text{FGO}) : R \leftarrow 1$

RT_0 : AR ← 0, TR ← PC

RT_1 : $M[\text{AR}]$ ← TR, PC ← 0

RT_2 : PC $\leftarrow$ PC $+ 1$, IEN $\leftarrow 0, R \leftarrow 0$, SC $\leftarrow 0$

Flowchart for determining all the operations is shown below:

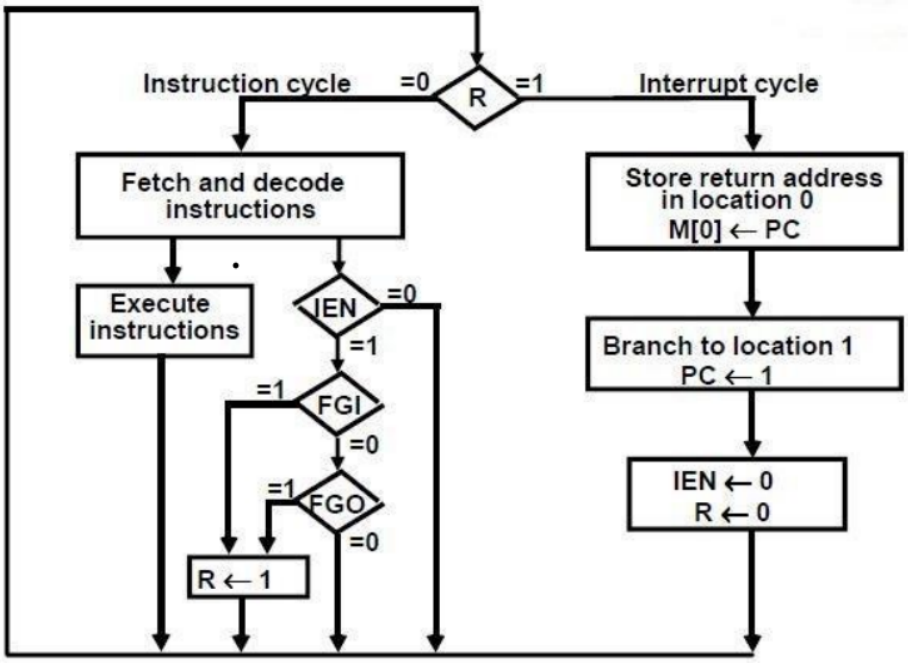

Figure 3.2: Flowchart for Interrupt Cycle

3.3 Design and Expressions of Registers and Memory

we have used r and p as a combination of signals which is further described below:

r=$D_{15}I'T_3$

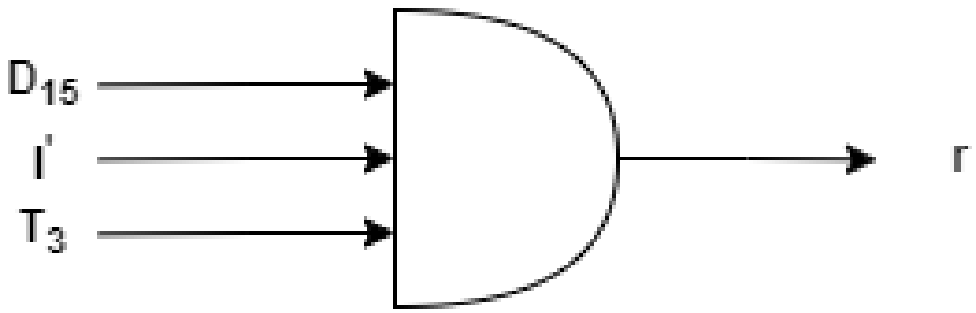

Figure 3.3: r Signal

p=$D_{15}IT_3$

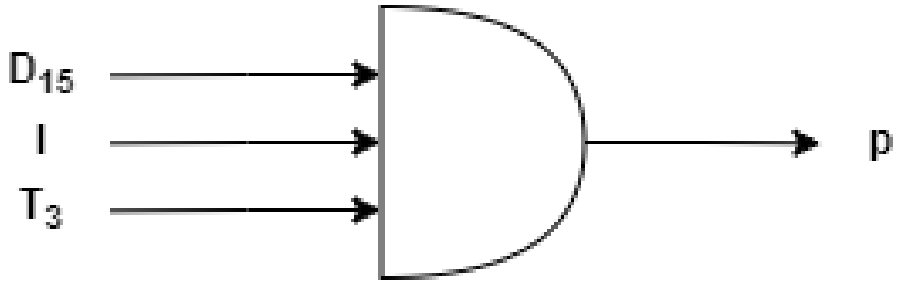

Figure 3.4: p Signal

3.3.1 AR

$R'T_0$: AR $\leftarrow$ PC

$R'T_2$: AR $\leftarrow$ IR$(0-17)$

$D'_{15}IT_3$: AR $\leftarrow M[$AR$]$

RT_0 : AR $\leftarrow 0$

$D_{10}T_4$: AR $\leftarrow$ AR $+ 1$

D_5T_4 : AR $\leftarrow$ AC $+ 1$

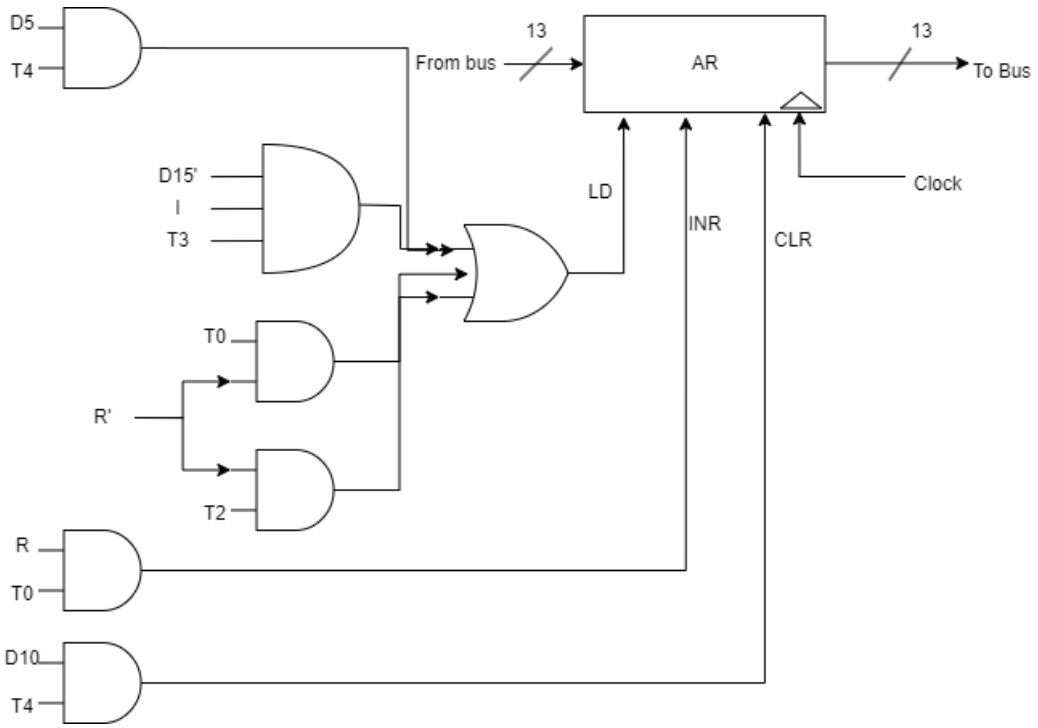

Figure 3.5: Logical Diagram of AR

3.3.2 PC

LOAD: $D_3T_6 + D_9T_4 + D_{12}T_5$
INCR: $RT_2 + D_{10}T_6 + rB_1 + rB_2 + rB_3 + rB_4 + pB_9 + pB_{10}$
CLR: RT_1

3.3.3 IR

$R'T_1$: IR $\leftarrow M[\text{AR}]$

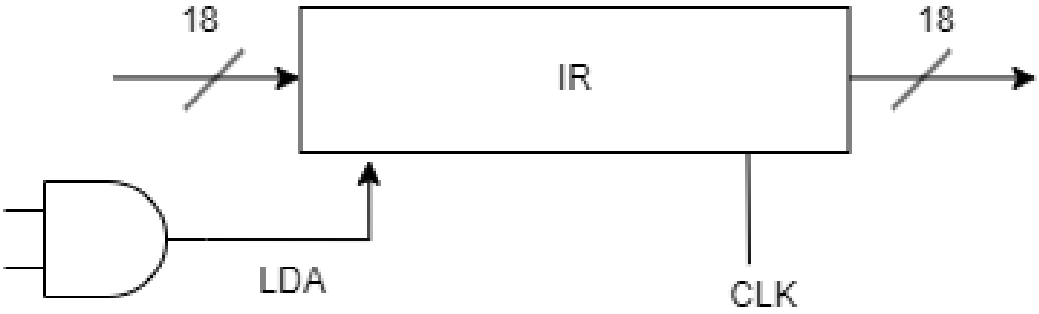

Figure 3.6: Logical Diagram of IR

3.3.4 TR

RT_0 : TR $\leftarrow$ PC

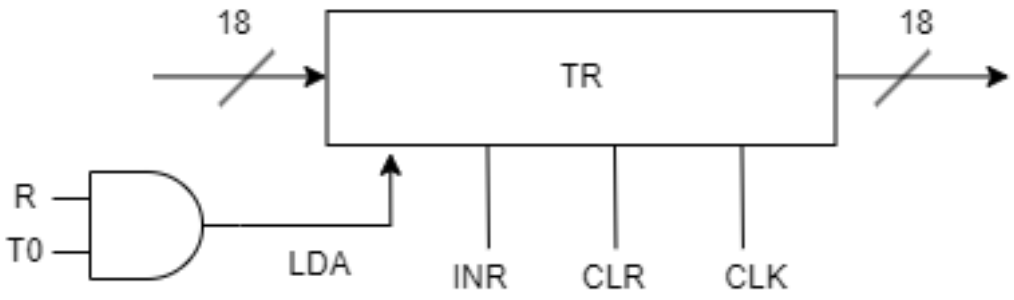

Figure 3.7: Logical Diagram of TR

3.3.5 DR

Load:

D_0T_4: DR $\leftarrow$ M[AR]

D_1T_4: DR $\leftarrow$ M[AR]

D_2T_4: DR $\leftarrow$ M[AR]

D_4T_4: DR $\leftarrow$ M[AR]

D_5T_4: DR $\leftarrow$ M[AR]

D_7T_4: DR $\leftarrow$ M[AR]

D_8T_4: DR $\leftarrow$ M[AR]

$D_{10}T_4$: DR $\leftarrow$ M[AR]

$D_{10}T_5$: DR $\leftarrow$ DR + 1

$D_{12}T_4$: DR $\leftarrow$ M[AR]

$D_{13}T_4$: DR $\leftarrow$ M[AR]

$D_{14}T_4$: DR $\leftarrow$ M[AR]

INCR:

D_{25}: DR $\leftarrow$ DR + 1

CLR:

D_{114}: DR $\leftarrow$ 0

3.3.6 OUTR

$p_{B_{11}}$: OUTR $\leftarrow$ AC$(0-7)$

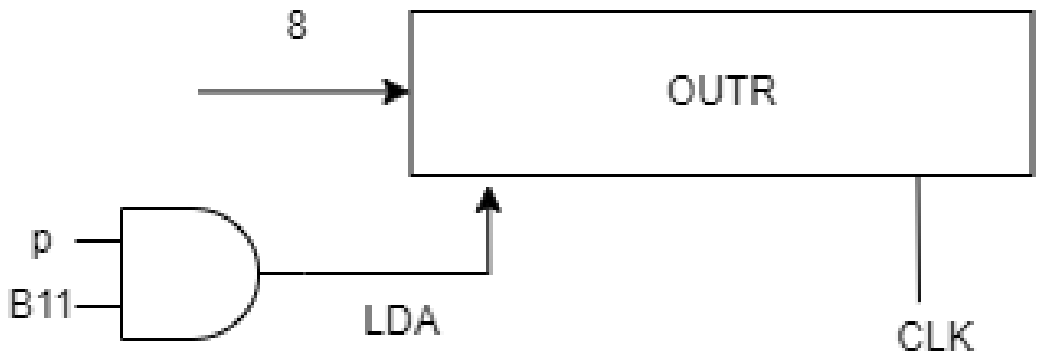

Figure 3.8: Logical Diagram of OUTR

3.3.7 RAM

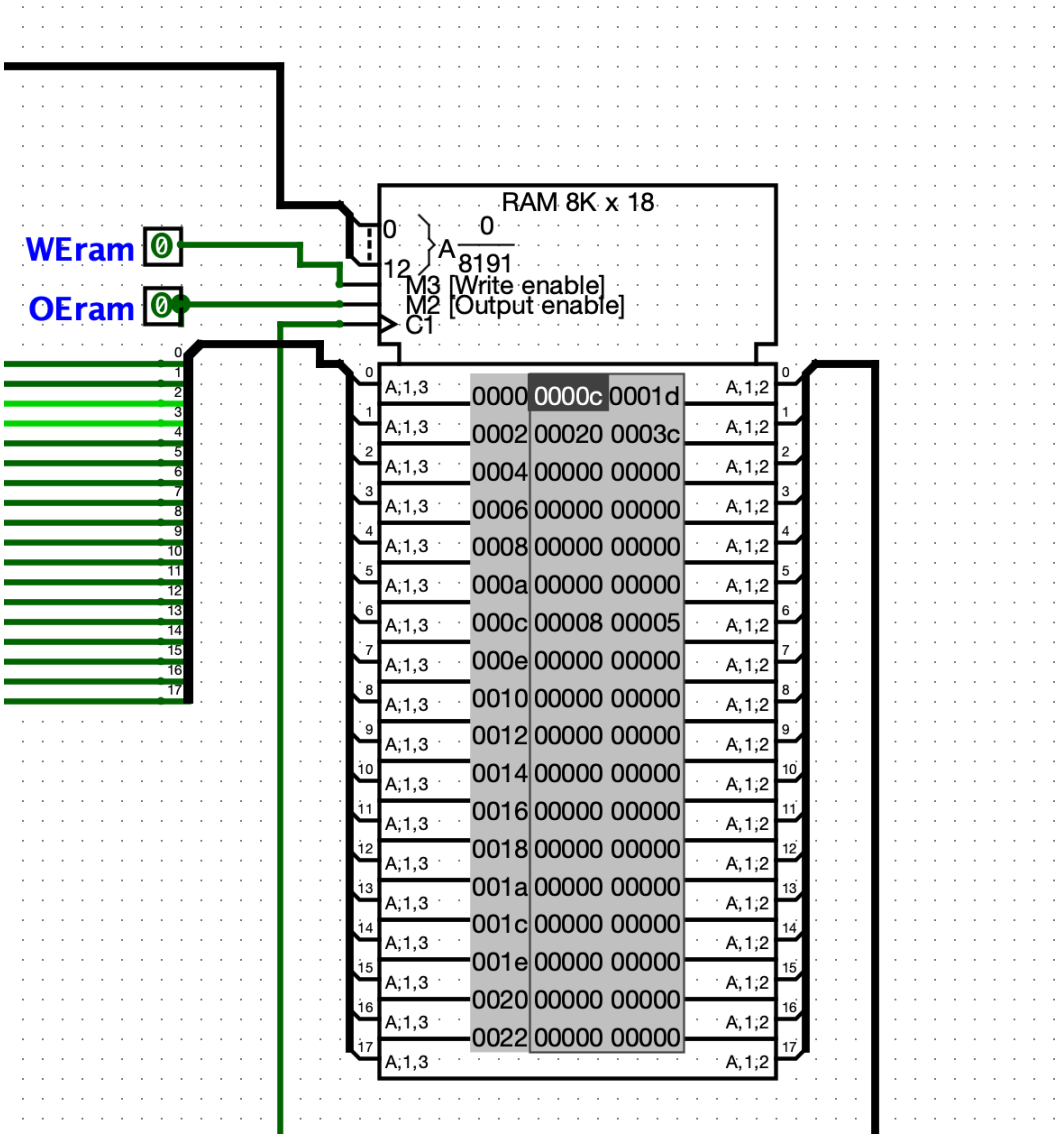

Figure 3.9: Memory Unit of AD-18

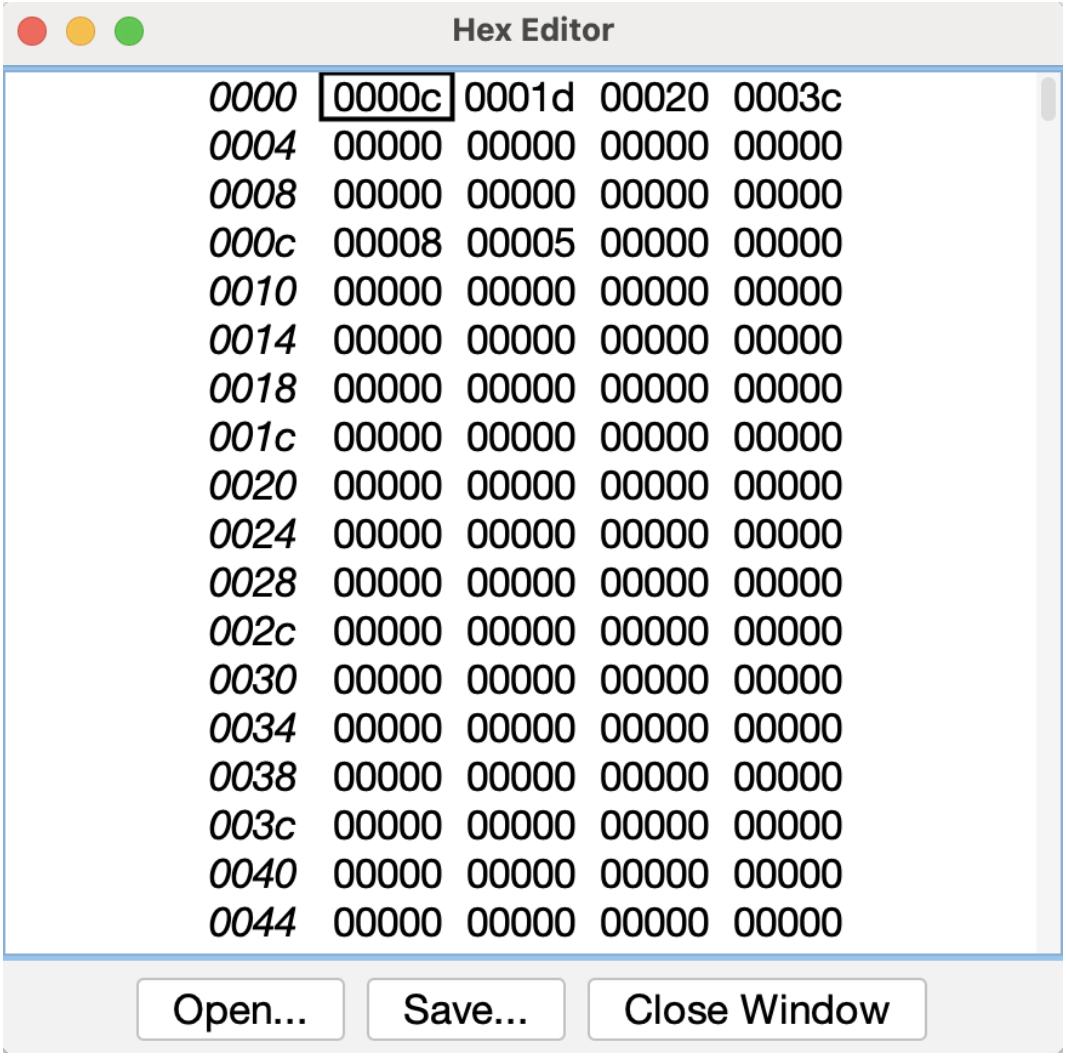

Figure 3.10: Initial Content of RAM

3.4 Design and Expression of Flip Flops and their Inputs

3.4.1 E Flip-Flop

- **SET:**
  D_0T_5: E←Cout
  rB_7: E←AC[17]
  rB_8: E←AC[0]

- **RESET:**
  rB_9: E←E'
  rB_{13}: E←0

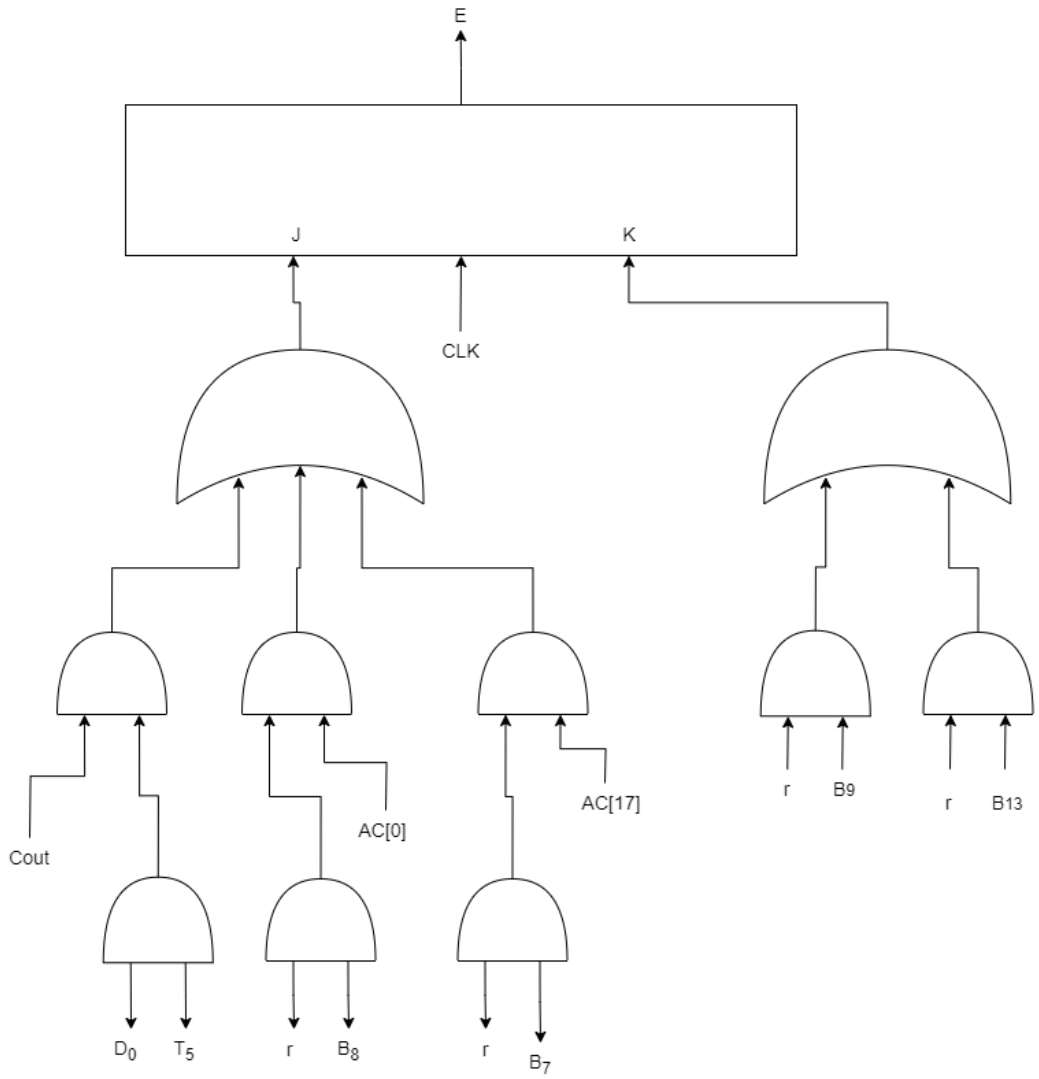

Figure 3.11: Logical Diagram of E Flip-Flop

3.4.2 R Flip-Flop

- **SET:**

  $T_0'T_1'T_2'(\text{IEN})(\text{FGI} + \text{FGO}) : R \leftarrow 1$

- **RESET:**

  $RT_2 : R \leftarrow 0$

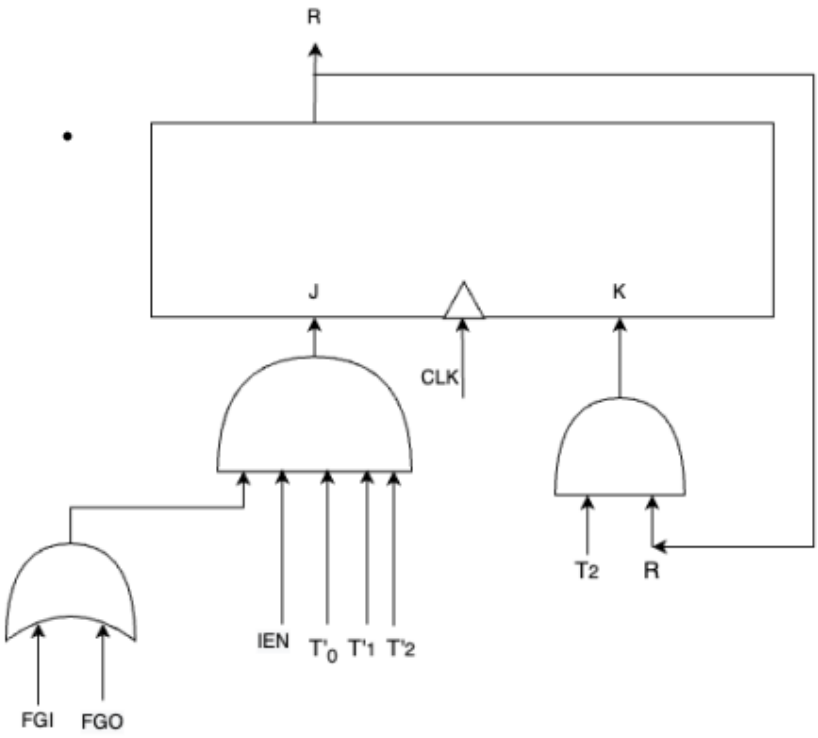

Figure 3.12: Logical Diagram of R Flip-Flop

3.4.3 FGI Flip-Flop

- **RESET:**
  pB_{12}:FGI $\leftarrow 0$

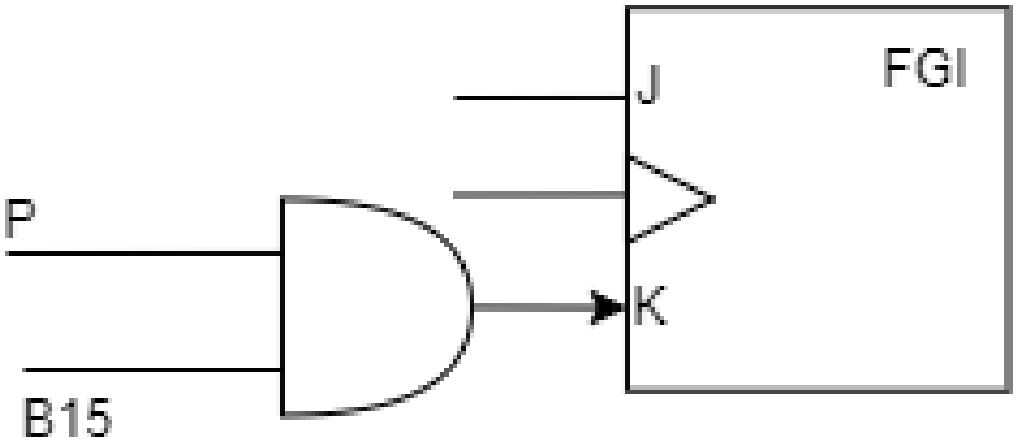

Figure 3.13: Logical Diagram of FGI Flip-Flop

3.4.4 FGO Flip-Flop

- **RESET:**

  pB_{11}: FGO $\leftarrow 0$

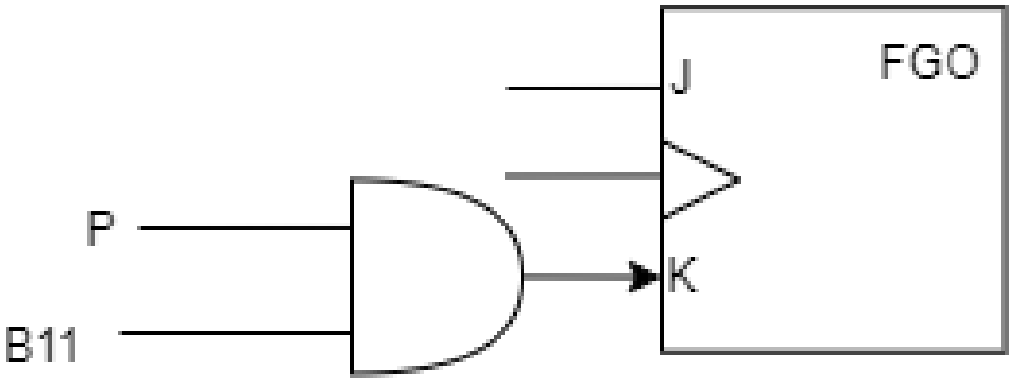

Figure 3.14: Logical Diagram of FGO Flip-Flop

3.4.5 IEN Flip-Flop

- **SET:**

  pB_8: IEN $\leftarrow 1$

- **RESET:**

  pB_7: IEN $\leftarrow 0$

  RT_2 : IEN $\leftarrow 0$

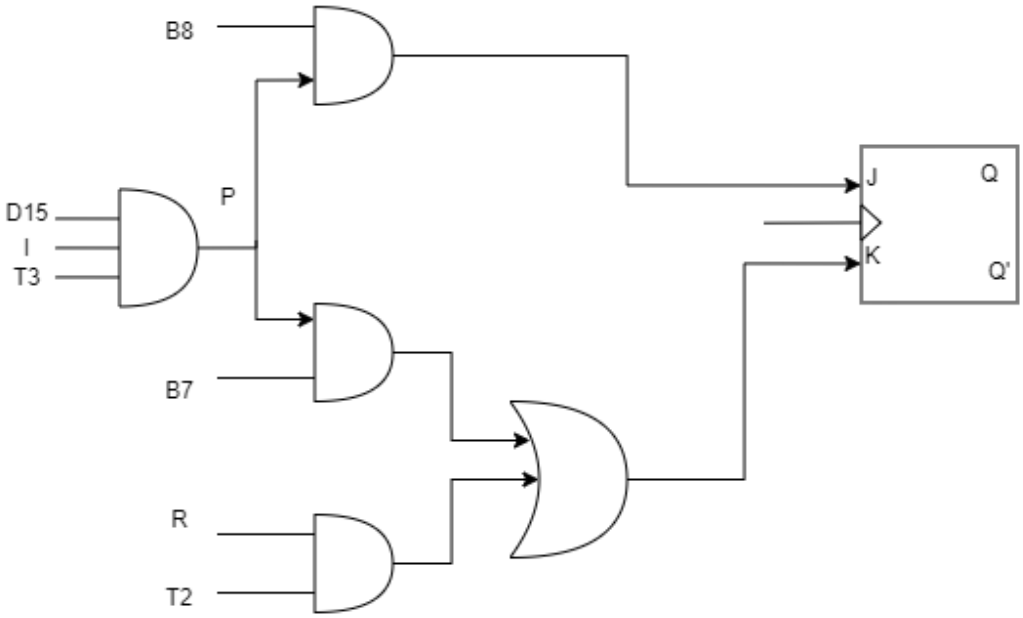

Figure 3.15: Logical Diagram of IEN Flip-Flop

3.4.6 S Flip-Flop

- **RESET:** :
  rB_0: S $\leftarrow$ 0

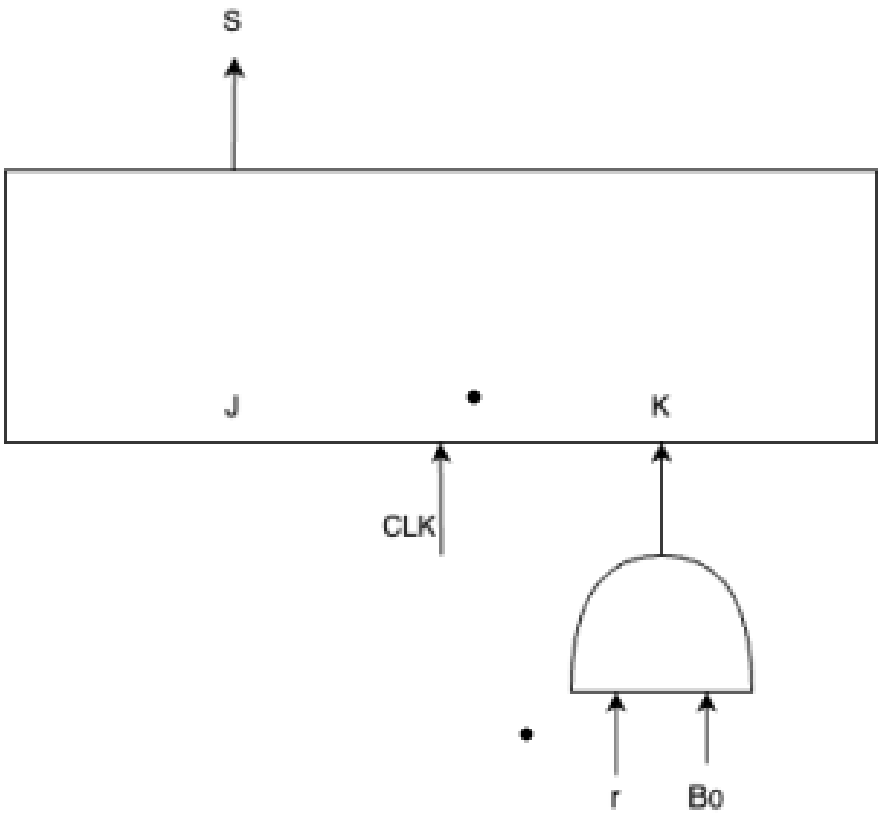

Figure 3.16: Logical Diagram of S Flip-Flop

3.5 Control of Common Bus

The 18-bit common bus is managed through selection inputs S2, S1, and S0, each representing binary numbers that correspond to specific registers. Boolean variables X1 through X7 are associated with the binary numbers, determining the selection of individual registers on the common bus.

X1	X2	X3	X4	X5	X6	X7	S0	S1	S2	Selected Register
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

Table 3.4: Common Bus of AD

Chapter 4: AD-18 Simulation

4.1 Introduction to Simulation

The simulation of the AD-18 computer in Logisim serves as a pivotal phase in the project, allowing for a detailed analysis of the CPU's behavior during instruction fetch and execution. Unlike relying on automated control unit signals, our approach involves a manual simulation to meticulously trace the flow of instructions and data within the system.

4.2 Fetching and Executing Instructions

4.2.1 Instruction 1

Fetch Cycle

- **T0:** Address of the next instruction is stored on PC, so we activate OE of PC to place its content on internal address bus of A0 Tunnel. Then we activate WE of MR to store the content of the internal address bus of AO into MAR.
- **T1:** Address of next instruction is stored now in MAR, so we activate OE of MAR to place its content on external address bus. Then we enable OE of RAM to give us the instruction stored in address copied in the external address bus, then the content of the instruction will be copied in the external data bus. Then we activate WE of MBR to store the instruction sent from the RAM over the external data bus. Then we activate count enable of PC to point to the address of the next instruction (PC=PC+1)
- **T2:** Then we activate internal OE of MBR to place its content on internal data bus. Then we activate WE of IR to store the instruction placed in the internal data bus into IR.

Execute Cycle

- **T3:** Now, the LOAD A instructions is stored in IR, next step is to fetch the operand from memory by activate OE of IR. Then we activate WE of MAR to store the address of the operand placed on the internal address bus to MAR.
- **T4:** we activate OE of MAR to place the address of the operand on the external address bus. Then, we activate OE of the Ram to place the operand on the

external data bus. Then, we activate WE of MBR to store the content of the external data bus into MBR.

- **T5:** Next, we Activate internal OE of the MBR to place the operand value onto the internal data bus. Finally, we activate WE of register A to store the value of the operand from the internal data bus.

4.2.2 Instruction 2

Fetch Cycle

- **T0 - T2: Same as above Fetch cycle**

Execute Cycle

- **T3 - T4: Same as above Execute cycle**
- **T5:** Next, we activate internal OE of MBR to place the operand value onto the internal data bus. Finally, we activate register B to store the value of the operand from the internal data bus.

4.2.3 Instruction 3

Fetch Cycle

- **T0 - T2: Same as above Fetch cycle**

Execute Cycle

- **T3** Now instruction 3(ADD) is in IR, the next step is to activate OE of ALU to place the sum to to Reg A and Reg B onto the internal data bus. Then we activate WE of the output register to store the content of the internal data bus onto output register.

4.2.4 Instruction 4

Fetch Cycle

- **T0 - T2: Same as above Fetch cycle**

Execute Cycle

- **T3:** Now, the instruction 3 (STORE) instruction is stored in the IR, next strep is to store the value stored in Output Register to the memory location from IR. So Start doing that by activating OE of IR. Next we activate WE of MAR to store the address of the operand placed on the internal address bus to MAR. Next we activate OE of out reg to place its content on the internal data bus. Next we activate WE of MBR to store the content of the internal data bus to MBR.

- **T4:** Next, we activate OE of MAR to place its address of the operand on the external address bus. Then, we activate OE of MBR to place its value of the operand on the external data bus. Finally, we activate WE of RAM to store the operand on the RAM at the address that came from external address bus.

4.3 Repetition of the Instruction Cycle

The CPU repeats the instruction cycle, fetching and executing instructions sequentially, demonstrating the cyclic nature of program execution.
In Conclusion, The manual simulation in Logisim provides a detailed insight into the functioning of the AD-18 computer. This hands-on approach allows for a nuanced understanding of the interplay between the CPU, memory, and control signals, emphasizing the synthesis of theoretical knowledge and practical implementation. The repetitive nature of the instruction cycle showcases the systematic execution of instructions, forming the backbone of the AD-18's computational capabilities.

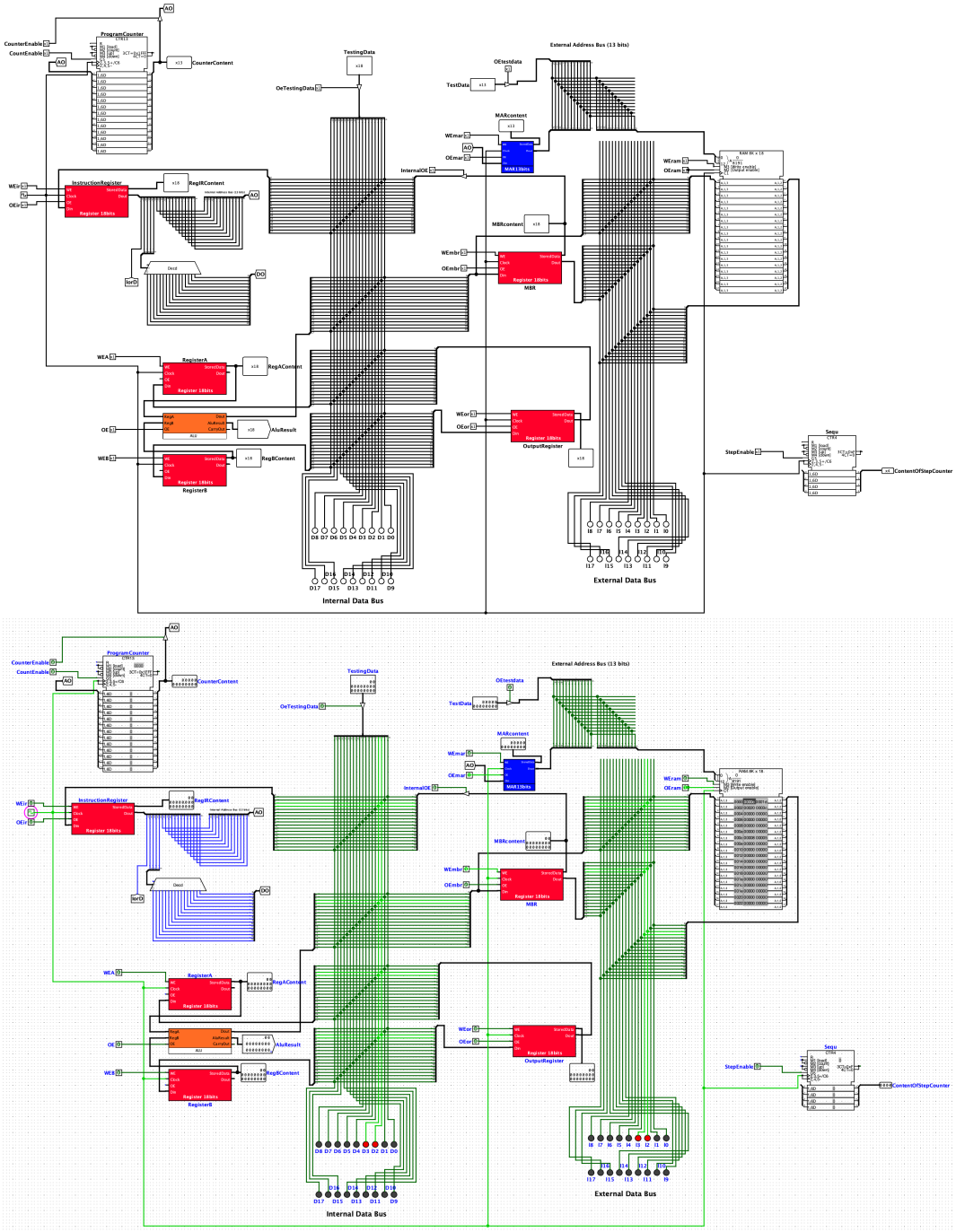

Figure 4.1: Logisim CPU Design

Chapter 5: Conclusion

In the culmination of our AD-18 computer project, we reflect on the journey from conceptualization to realization, achieving the objective of designing a sophisticated 18-bit system. Guided by foundational principles of computer architecture, we meticulously crafted an architecture featuring 18-bit registers, an 8K*18 memory configuration, and a comprehensive set of instructions. The Logisim simulation provided valuable insights through manual testing, aligning theoretical principles with practical implementation. Challenges in CPU design, memory configuration, and instruction set formulation served as learning opportunities. Looking ahead, we see future prospects in expanding the instruction set and optimizing CPU architecture. The AD-18 computer stands as a collaborative achievement, and we express gratitude to all contributors. This project deepened our understanding of computer engineering, leaving an indelible mark on our journey as aspiring engineers, and the AD-18 stands as a beacon of innovation in the dynamic field of computer architecture.