Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Project Report
On
"SKIP COMPUTER DESIGN"

[Code No: COMP 315]

Submitted by:
Siza Adhikari (03)
Isha Adhikary (04)
Pranima Kansakar (20)
Krisha Shrestha (44)

Submitted to:
Mr Pankaj Dawadi
Assistant Professor
Department of Computer Science and Engineering

Submission Date: 07-09-2021

Acknowledgement

As a matter of words, a project is like a bridge between theoretical and practical working and getting a chance to be a part of it. We are feeling highly obliged in taking the opportunity to sincerely thank our respected teacher Mr Pankaj Dawadi Sir for including this mini-project in our syllabus of **COMP 315**. We are also thankful for giving us this wonderful opportunity to explore our abilities and knowledge in the field of computing. We tried to give our best in this project.

Sincerely,

Siza Adhikari (03)

Isha Adhikary (04)

Pranima Kansakar (20)

Krisha Shrestha (44)

Abstract

A computer can understand and execute a few numbers of instructions that are hardwired in its design, these instructions form the instruction set of the computer. An **18 bit CPU** is a complex and yet essential process needed to implement all the ideas of Computer System Architecture together. A thorough understanding of computer organization and architecture is necessary to complete this task. For that, we have implemented **26 different instructions** based on the suitability of our project. Our computer has a 4-bit opcode, 12-bit address length and 2-bit Addressing Modes. With that being said, Our computer **SKIP** consists of the following hardware components:

1. A memory of **4096x18**.

2. **Nine registers:** AR, PC, DR, AC, IR, TR, OUTR, INPR and SC.

3. **Six flip-flops:** R, S, E, IEN, FGI and FGO.

4.**Three decoders**: a 4 x16 opcode decoder, a 4x16

timing decoder and a 2x4 addressing mode decoder.

5. A **18-bit common bus.**

6. Control logic gates.

7. Adder and logic circuit connected to the input of AC.

As we have used 4096x18 sized memory, it will allow us to have 4096 different slots wherein each slot, 18 different bits can be stored and executed.

As 4096 different slots have been allocated, we do need registers for storing and carrying out different sorts of instructions to be performed, this leads us to choose 8 different registers. To make the flow of instructions simple yet effective, we have implemented the Common bus System which is an 18-bit bus system.

These hardware components which when assembled accordingly along with many logical gates like AND, OR gates will help to operate the 29 different instructions that we have set and help us to get the desired output.

List of Figures

List of Tables

Table of Contents

Chapter 1: Introduction

As we know, the computer understands as well as executes the operation in binary code. This binary code is called the instruction. A computer can understand and execute a few numbers of instructions that are hardwired in its design, these instructions form the instruction set of the computer. The selection of binary code for the instruction and the instruction itself is the main task of the computer designer.

Designing an **18 bit CPU** is a complex and yet essential process needed to implement all the ideas of Computer System Architecture together. A thorough understanding of computer organization and architecture is necessary to complete this task.

On our computer, we have implemented **29 different instructions** based on the suitability of our project. Our computer has a 4-bit opcode, 12-bit address length and 2-bit addressing mode.

Our computer consists of the following hardware components:

1. A memory of **4096 x 18**.
2. **Nine registers:** AR, PC, DR, AC, IR, TR, OUTR, INPR and SC.
3. **Six flip-flops:** R, S, E, IEN, FGI and FGO.
4. **Three decoders**: a 4x16 opcode decoder, a 4x16 timing decoder and a 2x4 addressing mode decoder.
5. A **18-bit common bus.**
6. Control logic gates.
7. Adder and logic circuit connected to the input of AC.

Chapter 2: Design Considerations

The following section describes the design process and the internal architecture of our computer as well as the instruction set supported by this computer.

2.1. Instruction Format

There are two parts an instruction can be divided into. They are:

Operation Code (Opcode): It specifies the operation for an instruction.
Address: It specifies the registers and/or locations in memory to use for a particular operation.

In the SKIP computer, since the memory contains $4096 = 2^{12}$ words, we need 12 bits to specify which address the instruction will use. Since the words are 18 bits each, we reserve 12 LSBs for the address. Similarly, the 2 MSBs are reserved to represent addressing modes(AM). Finally, the remaining 4 bits are used to represent opcodes of the instructions.

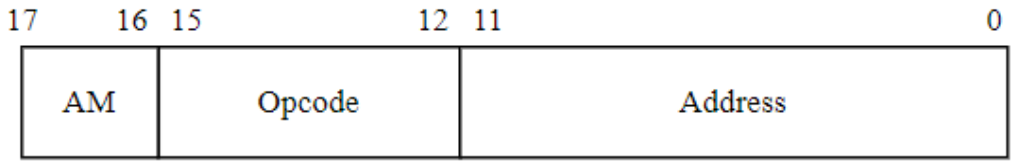

Figure 1: Instruction Format

2.2. Addressing Modes

There are 3 addressing modes used in the SKIP computer. They are:

	Addressing Mode Field	**Addressing Mode**
I_0	00	Direct
I_1	01	Indirect

I_2	10	PC Relative
I_3	11	N/A

Table 1: SKIP Computer Addressing Modes

2.2.1. Direct Addressing Mode

The address of the operand is stored in the memory location specified in the instruction in the direct addressing mode. An example is shown in figure 2.

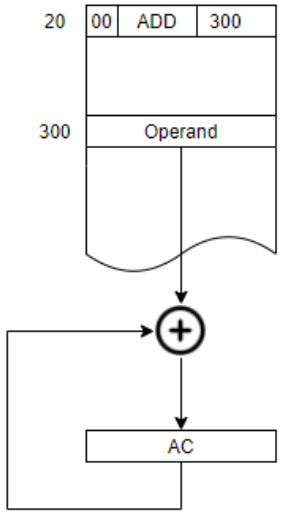

Figure 2: Direct Addressing Mode

2.2.2. Indirect Addressing Mode

The address of the operand is stored in the instruction in the indirect addressing mode. An example is shown in figure 3.

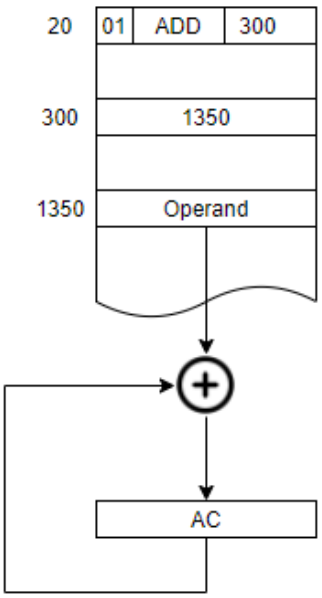

Fig 3: Indirect Addressing Mode

2.2.3. PC Relative Addressing Mode

The address stored in the instruction in the PC relative addressing mode is added with the address of the next instruction i.e., the current content of the PC. The resultant value gives the address of the operand. An example is shown in figure 4.

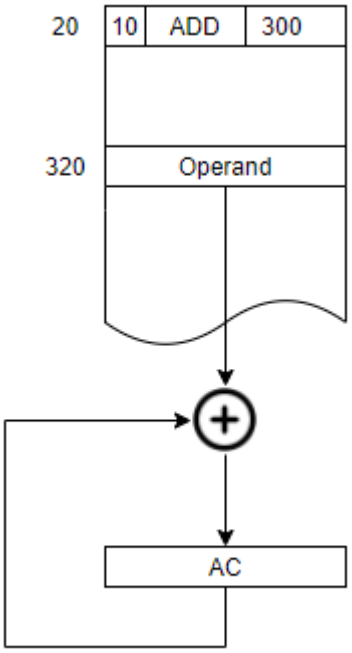

Figure 4: PC Relative Addressing Mode

2.3. Components

2.3.1. Registers

SKIP Computer is composed of 9 registers. They are as follows:

2.3.1.1. Instruction Register (IR)

Size: 18 bits

IR stores the 18 bits of instruction read from the memory. At the start of each instruction cycle, the address contained in the PC is transferred to the AR, and the content of the memory location pointer by AR is then transferred to the IR through the common bus system (CBS). Once the instruction is loaded in the IR it is then decoded to obtain the decoded lines (I_0 - I_3), (D_0 - D_{15}) and B_0 - B_{11} which along with the timing signals is used to provide the necessary control signals for the computer to execute the stated operation.

2.3.1.2. Program Counter

Size: 12 bits

The program counter contains the address of the memory location where the next instruction is located. At the beginning of each instruction cycle (fetch), the content of the program counter is transferred to AR and then is incremented by one to point to the next consecutive location.

In the case of subroutines, the content of the program counter (PC) is saved in the memory and is loaded with the location of the memory containing the subroutine procedure. Finally, at the end of the subroutine call, the PC is loaded with the previously saved address and the execution cycle continues.

2.3.1.3. Data Register

Size: 18 bits

The data register is used to store the operand read from the memory that is to be processed. The output of the data register is an input to the ALU.

2.3.1.4. Accumulator

Size: 18 bits

The accumulator is the main register for most of the operations. It stores the result of any operation after its completion. The input of the accumulator comes from the output of the ALU. The input of the accumulator is not directly interfaced with the CBS. The output of the AC goes to the common bus system as well as to ALU as the first operand, the second being the content of the Data register.

2.3.1.5. Input Register

Size: 9 bits

Input register (INPR) is connected to the ALU and the content of the input register is transferred to the ALU on the selection of the transfer operation.

2.3.1.6. Output Register

Size: 9 bits

Output register (OUTR) is used by the computer to provide output to the external world. It receives its output from the common bus system which usually comes from the accumulator. This output may be then connected to an external display.

2.3.1.7. Temporary Register

Size: 18 bits

An 18-bit non-programmable register used to hold data during an arithmetic and logic operation. It is used to hold intermediate results which the programmer can't access at all. It is temporarily stored inside the microprocessor.

In our SKIP computer design, the temporary register came into play, when we were performing the SUB operation. And the instruction set along with the use of the temporary register looks like this:

$D_{10}T_8$:	DR← M[AR]
$D_{10}T_9$:	TR← AC
$D_{10}T_{10}$:	AC ← DR
$D_{10}T_{11}$:	AC ← AC'
$D_{10}T_{12}$:	AC ← AC + 1, DR ← TR
$D_{10}T_{13}$:	AC ← AC + TR, SC ←0

Table 2: Use of Temporary Register

2.3.1.8. Sequence Counter

It is used for data rarely written to (e.g., system time), where the reader wants a consistent set of information and is willing to retry if that information changes. In SKIP, a 4-bit sequence counter is used as shown in figure 5 below which is connected to a 4 x 16-bit decoder.

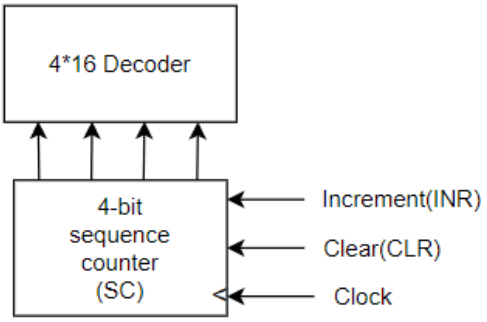

Figure 5: Sequence Counter(SC)

2.3.1.9. Address Register

Size:18 bits

It contains main memory addresses of data and instructions or a portion of the address that is used in the calculation of the complete addresses. It holds the memory location of data that needs to be accessed.

2.3.2. Arithmetic and Logic Unit (ALU)

The arithmetic and logic unit in our computer performs arithmetic and logical operations. We will be discussing the ALU of the SKIP computer further in section 3.4.

2.3.3. Memory Unit

*Size: 4096*18*

The memory size 4096 x 18 indicates a basic computer having 4096 different slots where 18 bits of instructions can be stored and executed in each slot. Memory stores both the instruction and the data on which processing is to be performed. We have selected a memory of 4096 words with a word size of 18-bits for our computer. This means that we require a 12-bit address line to address all the words of the memory. Memory receives its address from the Address Register (AR), it receives/sends its data from the common bus system, read or write operations are distinguished from the control signal.

2.3.4. Common Bus System

The registers in SKIP Computer are connected using an 18 bits Bus. The bus connects Memory of 18 bits, Accumulator of 18 bits, Address Register of 12 bits, Program Counter of 12, Instruction Register of 18 bits, Temporary Register of 18 bits along with Data Register of 18 bits. The Data register is connected to the ALU which is connected with the control unit and the Accumulator of 18 bits. The Control Unit is also connected to Control Signals. The Instruction Register passes decoded instruction to the Control Unit. Output Register of 9 bit is connected to the Bus while the Input Register is connected with ALU to provide data from input. With the help of 8x1 Multiplexer, the register is selected for data transfer. The selection lines to the multiplexer comes from the output of a 8x3 encoder. We can see the block diagram of the CBS in figure 6 below.

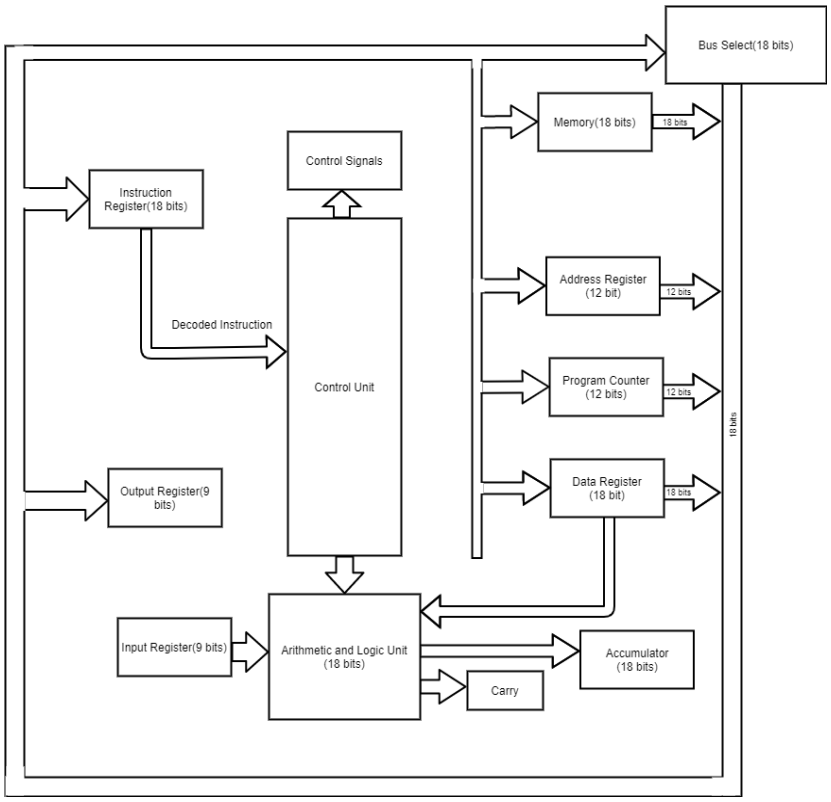

Figure 6: Block diagram of Common Bus System

2.3.5. Flip Flops/Flags

There are 6 flip flops used in SKIP Computer which are listed below.

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

Table 3: List of Flip Flops

2.3.6. Control Unit

The Control Unit (CU) generates the necessary control and timing signals to carry out the sequences of micro-operations to execute the given instruction. We will be discussing the control unit of the SKIP computer further in section 2.5.

2.4. Instructions

2.4.1. Instruction Types and Instruction Format

The different types of instruction formats with their instruction types in SKIP Computer is shown below:

a. Memory Reference Instruction (MRI)

In memory reference instruction 6-bits read from the memory during the fetch and decode cycle is used to specify the memory operation, and the next consecutive 12 bits read from the memory contains the memory address of the operand.
The instruction code format for the MRI is:

For direct addressing,

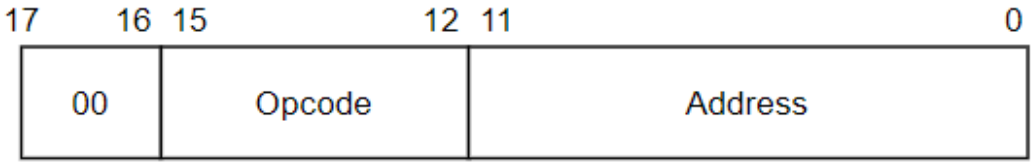

Figure 7: MRI Instruction for Direct Addressing Mode

For indirect addressing,

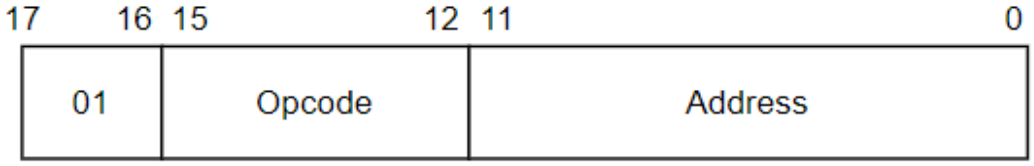

Figure 8: MRI Instruction for Indirect Addressing Mode

For PC relative addressing,

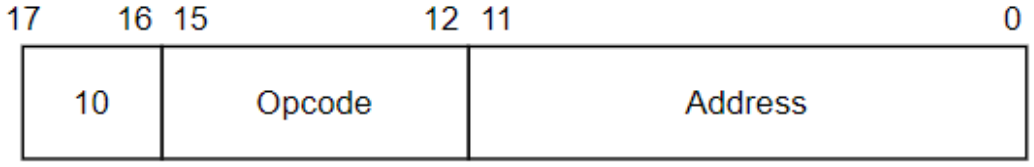

Figure 9: MRI Instruction for PC Relative Addressing Mode

b. Register Reference Instruction (RRI)

In RRI, the higher 6 bits of the instruction code is always 001111. The lower 12 bit of the instruction code is then utilized to specify the specific register reference

instruction. These lower 12 bits are then decoded to generate necessary control signals for executing the given instruction. The instruction format for RRI is given below:

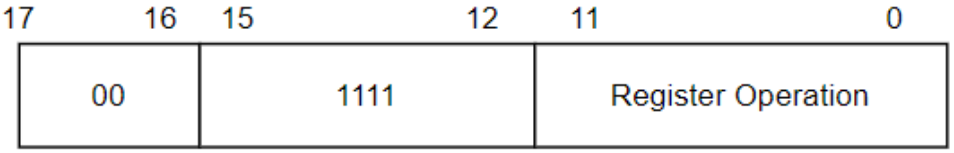

Figure 10: RRI Instruction

c. Input-Output Instruction (IOI)

In IOI, the higher 6 bits of the instruction code is always 111111.The lower 12 bits of the instruction code is then utilized to specify the specific IOI. These lower 12 bits are then decoded to generate necessary control signals for executing the given instruction. The instruction format for IOI is given below:

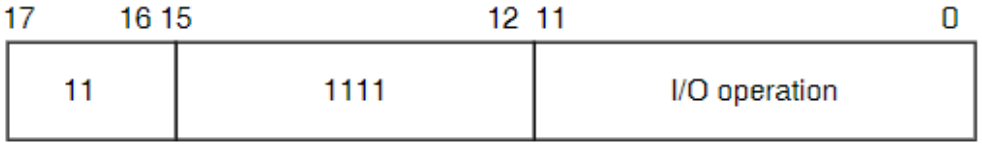

Figure 11: I/O instruction

2.4.2 SKIP Computer Instructions

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

Symbol	Opcode			Description
	I = 00	I = 01	I=10	
AND	000000X	010000X	100000X	AND the AC with memory word
ADD	000001X	010001X	100001X	Add the memory word with AC and save to AC
LDA	000010X	010010X	100010X	Load accumulator with memory word
STA	000011X	010011X	100011X	Store value of AC to memory
BUN	000100X	010100X	100100X	Branch unconditionally
BSA	000101X	010101X	100101X	Branch and save return address
ISZ	000110X	010110X	100110X	Increment and skip if zero
OR	000111 X	010111 X	100111X	OR the AC with memory word
XOR	001000X	011000X	101000X	XOR the AC with memory word
XCHG	001001X	011001X	101001X	Exchange the AC with memory word
SUB	001010X	011010X	101010X	Subtract the memory word from AC

Table 4: Memory Reference Instructions

Here, 'X' is a 12-bit memory address

b. Register Reference Instructions

Register reference instructions work with registers only.

In our design, we have the following register reference instructions:

Symbol	Opcode	Description
CLA	001111100000000000	Clear AC
CLE	001111010000000000	Clear E
CMA	001111001000000000	Complement AC
CME	001111000100000000	Complement E
CIR	001111000010000000	Circulate Right AC and E
CIL	001111000001000000	Circulate Left AC and E
INC	001111000000100000	Increment AC
SPA	001111000000010000	Skip next instruction if AC is positive
SNA	001111000000001000	Skip next instruction if AC is negative
SZA	001111000000000100	Skip next instruction if AC is zero
SZE	001111000000000010	Skip next instruction if E is 0
HLT	001111000000000001	Halt computer

Table 5: Register Reference Instructions

c. Input-Output Instructions

Symbol	Opcode	Description
INP	111111100000000000	Input character to AC
OUT	111111010000000000	Output character from AC

SKI	111111001000000000	Skip on input flag
SKO	111111000100000000	Skip on output flag
ION	111111000010000000	Interrupt Enable on
IOF	111111000001000000	Interrupt Enable off

Table 6: Input-Output Instructions

2.4.3. Instruction Set Completeness

The SKIP computer consists of the following instructions:

- **Transfer Instructions**

Here, the data transfers between the main memory and the processor's registers take place which brings us to the transfer instructions LDA, STA, XCHG in our operations.

- **Functional Instructions**

Here, the Arithmetical, Logical and Shift Instructions take place which brings us to the Functional instructions like ADD, SUB, CMA, INC, CIR, CIL, AND, OR, XOR, CLA in our operations.

- **Control Instructions**

Here, the generation of control signals for the smooth flow of Instructions takes place which brings us to the Control instructions like BUN, ISZ, BSA in our operations.

- **Input/Output Instructions**

Here, when the instructions between the CPU/memory or cache with the peripheral devices come into play, the instructions like INP and OUT help in the smooth running of the I/O operations.

2.5. Control Unit

The control unit of the SKIP computer is implemented by using a **hardwired control organization**. In hardwired organizations, the control logic is implemented with gates, flip flops, decoders and other digital circuits. It is used to produce a fast mode of operation.

2.5.1. Timing and Control

The timing for all registers in the SKIP computer is controlled by a master clock generator. The clock pulses are applied to all flip flops and registers in the system, including the flip flops and registers in the control unit. The clock pulses do not change the state of the register unless the register is enabled by a control signal. The control signals are generated in the control unit and provide control inputs for the multiplexers in the common bus, control inputs in the processor register and micro-operations hardwired organization for the accumulator.

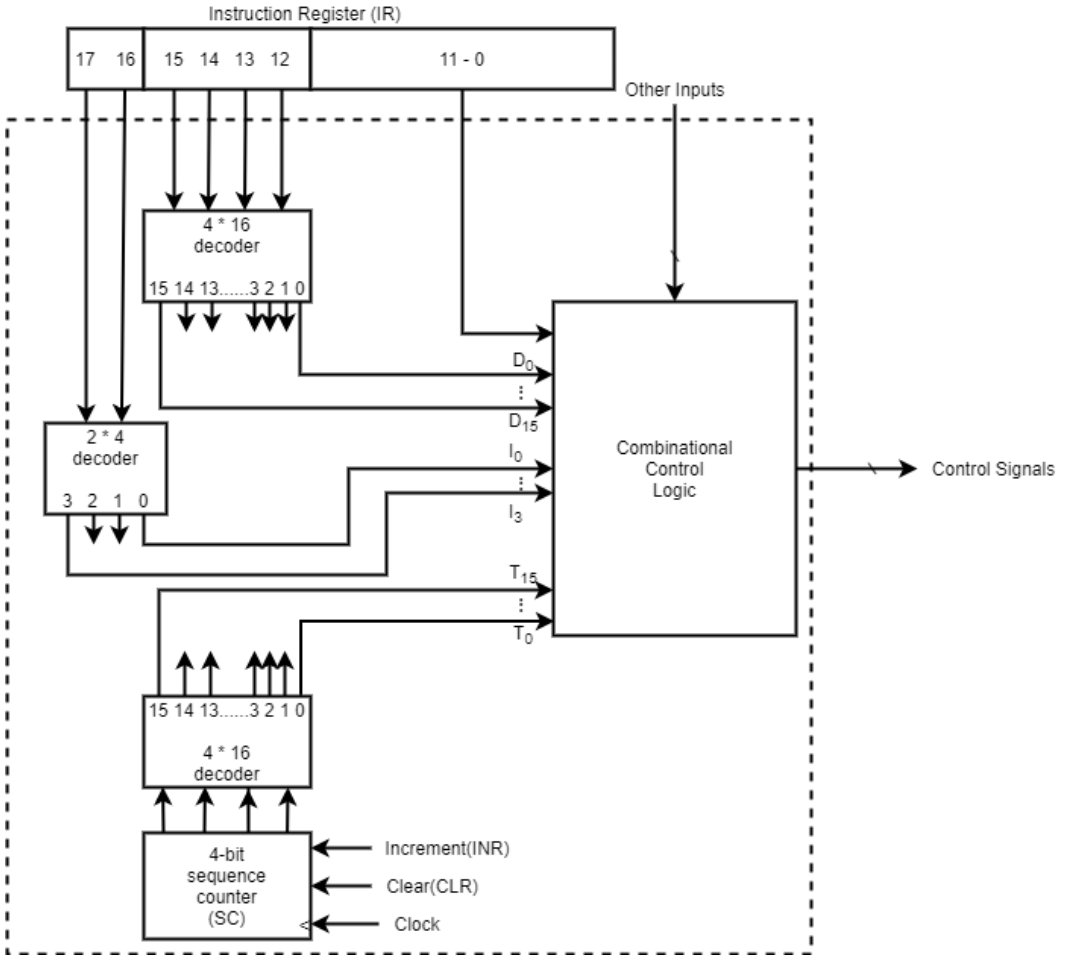

Figure 12: Control Unit of SKIP Computer

The block diagram of the control unit is shown in figure 12. It consists of three decoders, a sequence counter, and several control logic gates. An instruction read from memory is placed in the instruction register (IR). The IR consists of three parts: the addressing mode field, the operation code and the address. The addressing mode field in bits 16 and 17 are decoded with a 2*4 decoder. The four outputs of the addressing mode field decoder are designated by the symbols I_0 through I_3. The subscripted decimal number is equivalent to the binary value of the corresponding address field. Similarly, the operation codes in bits 12 through 15 are decoded with a 4*16 decoder. The sixteen outputs of the opcode decoder are designated by the symbols D_0 through D_{15}. The subscripted decimal number is equivalent to the binary

value of the corresponding opcode. Bits 0 through 11 are applied to the control logic gates.

2.5.2. Timing Signals

The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}. The sequence counter can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4x16 decoder. Once in a while, after completion of any instruction, the counter is cleared to 0 causing the next active timing signal to be T_0.

2.6. Instruction Cycle

A computer can understand and execute a few numbers of instructions that are hardwired in its design, these instructions form the instruction set of the computer. The selection of binary code for the instruction and the instruction itself is the main task of the computer designer.

A machine instruction is executed in the following cycle:

1. **Fetch Cycle:** Here, the instruction is fetched from the memory.
2. **Decode Cycle:** After fetching the instructions, they are decoded in this cycle. This mainly happens in the T_2 cycle. Here, the decision is taken such that the instructions are sent accordingly to the respective addressing modes.
3. **Execution Cycle:** Once, the decision of sending to the addressing modes are taken, the instructions are finally executed in this phase.

2.6.1. Fetch Cycle

During the fetch cycle, the instruction is read from the memory and loaded in the instruction register. For this, the address of the next location containing the instruction in the memory is contained in the program counter. This address is transferred to the Address Register during the first T0 timing signal.

On the next timing signal, the content of the memory pointed by the address register is copied to the instruction register; in the meantime, the Program counter and the

address register are incremented by one.

Following RTL denotes the fetch cycle.

T_0: AR ← PC

T_1: IR←M [AR], PC ← PC+1

2.6.2. Decode Cycle

In the decoding cycle, the instruction register's instruction is decoded to determine the type of instruction. Necessary decoding lines are generated to specify the operation. Following is the RTL for the decoding cycle:

T_2: $D_0 \ldots D_{15}$← Decode IR (12-15)], $I_0,...I_3$← Decode IR (16-17),

AR ← IR(0-11)

Determine The Type of Instruction

After the instruction being fetched from the memory till the process of the instruction being decoded in the decode phase in the T_2 cycle. Now, comes the time to decide on the instruction before sending it to the Execution cycle.

As we can see in the figure, for the decision process, the decoded values I_0 to I_3 and D_0 to D_{15} evaluates the instruction to be the Memory Referenced Instruction, Register Referenced or Input-Output instruction. When the instruction is sent to the MRI it either goes to the Direct addressing, Indirect addressing or the PC Relative addressing before the execution and when the bit comes to the RRI or I/O again the 16^{th} and the 17^{th} bit is checked before the execution phase begins.

A flowchart for the fetch and decode cycle is shown in figure 13.

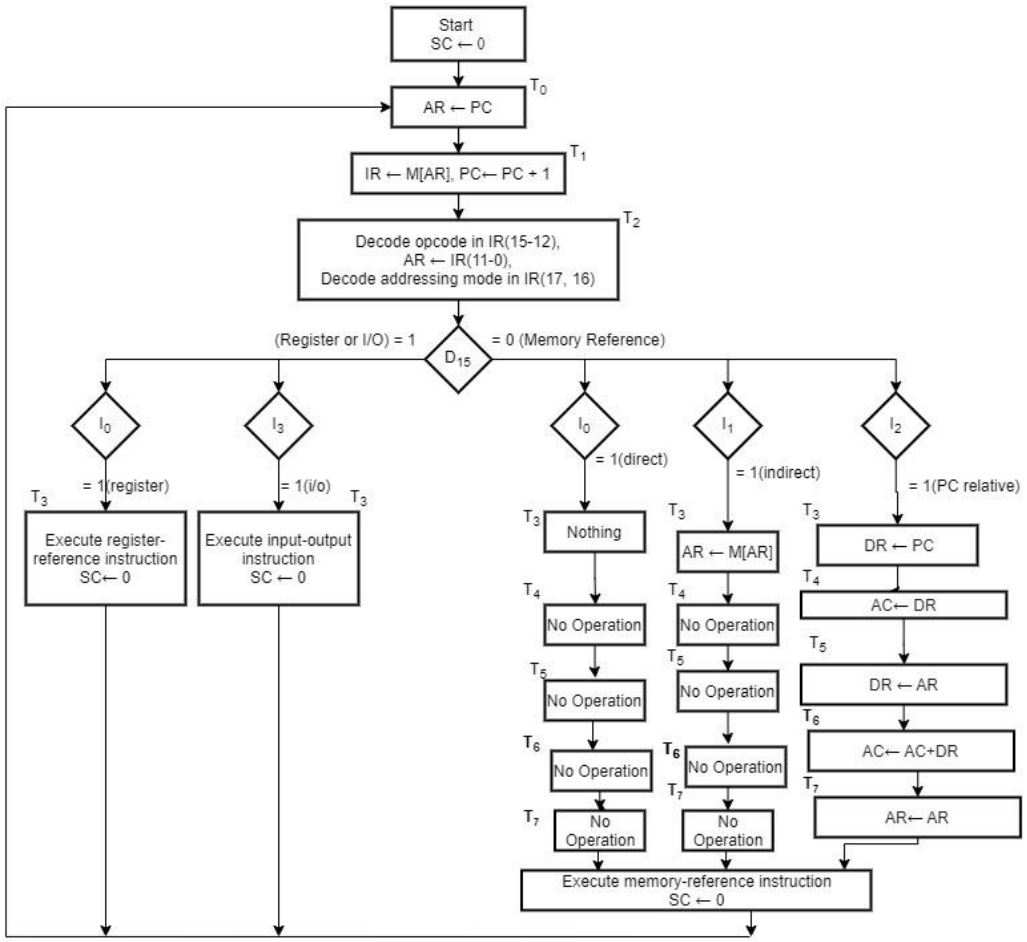

Figure 13: Determine the Type of Instruction

From figure 13,

$D_{15}.I_0.T_3$: Execute register-reference instruction

$D_{15}.I_3.T_3$: Execute input-output instruction

$D_{15}'.I_0.T_3$: Nothing

$D_{15}'.I_0.T_4$: No Operation

$D_{15}'.I_0.T_5$: No Operation

$D_{15}'.I_1.T_3$: AR ← M[AR]

$D_{15}'.I_1.T_4$: No Operation

$D_{15}'.I_1.T_5$: No Operation

$D_{15}'.I_2.T_3$: DR ← PC

$D_{15}'.I_2.T_4$: AC ← DR

$D_{15}'.I_2.T_5$: DR ← AR

$D_{15}'.I_2.T_6$: AC← AC + DR

$D_{15}'.I_2.T_7$: AR ← AC

2.6.3. Execution Cycle

After the decision taken in the T_2 cycle, the instructions are sent accordingly to their respective addressing modes, where in the T_8 cycle, the execution takes place. The instructions are sent to three different reference instructions, they are: Register Reference instructions, Memory Reference instructions and Input-Output instructions.

2.6.3.1. Register Reference Instructions

In this, the instructions take the reference of the Registers for its execution.

$D_{15}I'T_3 = r$ (Common to all register-reference instruction)

CLA

This instruction clears the content of the accumulator. A clear signal of the AC is activated to clear its content.

rB_{11}	AC←0

CLE

This instruction clears the carry flag E.

rB_{10}	E← 0

CMA

The content of the AC is complemented by this Instruction. The content of the AC is passed through the ALU where the complement operation is selected, finally, the complemented result is stored back in the ALU.

rB_9	AC ← (AC)’

CME

Carry flag E is complemented using. J=1 and K=1 (toggle) condition.

rB_8	E← (E)'

CIR

This is the circular right shift instruction and causes the content of the accumulator to be circularly shifted right without loss of any bit. This operation is carried out in ALU.

rB_7	AC←shr AC, AC(15) ←E, E←AC(0)

CIL

This is the circular left shift instruction and causes the content of the accumulator to be circularly shifted left without loss of any bit. This operation is carried out in ALU.

rB_6	AC←shl AC, AC(0) ←E, E←AC(15)

INC

It increments the content of AC by one. This is carried out by applying the INR signal to the AC.

rB_5	AC← AC+1`

SPA

Skip is positive AC causes the computer to skip the next instruction if the content of the accumulator is positive.

rB_4	If(AC(15)=0) then (PC←PC+1)

SNA

Skip is negative AC causes the computer to skip the next instruction if the content of

the accumulator is negative.

rB_3	If(AC(15)=1) then (PC←PC+1)

SZA

Skip on zero AC causes the computer to skip the next instruction if the content of the accumulator is zero.

rB_2	If(AC=0) then (PC←PC+1)

SZE

Skip on zero carry flag causes the computer to skip the next instruction if the carry flag is cleared.

rB_1	If(E=0) then (PC←PC+1)

HLT

Stop the execution of the program by resetting the start-stop flip-flop.

rB_0	S ← 0(S is a start-stop flip-flop)

2.6.3.2. Memory Reference Instructions

For the execution of the instructions, they take the reference of the memory.

AND <address>

This instruction copies the operand specified by the address to the data register during the first cycle of its execution. On the next cycle, this data is then ANDed with the content of the accumulator and the result is stored in the accumulator. The ANDing operation is performed inside the ALU. RTL for this instruction is:

D_0T_8:	DR ← M [AR]
D_0T_9:	AC←AC ^ DR, SC ←0

ADD <address>

During the first cycle, the operand specified by the address is copied to the DR. On the next cycle, the content of DR is added to the current content of the AC and the result is stored back to AC. If any carry is generated then the carry flag is set. The operation of adding two data is performed inside the ALU. RTL for this instruction is:

D_1T_8:	DR ← M [AR]
D_1T_9:	AC←AC+DR, E ←Cout, SC ←0

LDA <address>

This instruction loads the content of the memory location specified by the address. It takes two cycles to execute this instruction, during the first cycle the content of the memory The location pointed by the address register is copied to the DR. In the next cycle, the content of the data register is transferred to AC through the ALU. RTL for this instruction is given
below:

D_2T_8:	DR ← M [AR]
D_2T_9:	AC ←DR, SC ←0

STA <address>

This instruction stores the content of AC into the memory word specified by the effective address. The RTL for this instruction is:

D_3T_8:	M[AR] ← AC, SC ← 0

BUN <address>

This instruction causes the computer to branch unconditionally to the given address

and continue the execution of an instruction for the location. It takes only one step to execute this instruction. To execute this instruction, the content of the address is simply transferred to the program counter and the sequence counter is reset.

D_4T_8:	PC ← AR, SC ←0

BSA <address>

This instruction is especially helpful in implementing subroutine calls. It causes the computer to branch to the memory location specified by the address and in the meantime save the current address of the PC so that after the completion of the CALL, the program can continue its normal execution.

During the first cycle, the content of the program counter is saved in the memory location pointer by the AR. AR is also incremented by one. On the next execution state, the address stored in the AR is copied to the PC and SC is reset. RTL for this instruction is given below:

D_5T_8:	M [AR] ← PC, AR ← AR + 1

ISZ <address>

Increment and skip on zero instruction increments the content of the memory location specified by the address and if the result after incrementing is zero then it skips the next instructions.

In the first timing signal, the content of the memory The location pointed by the address is copied to the data register, on the next cycle data register, is incremented by one, finally on the last timing signal the content is copied back to the same memory location and if the content is zero then the computer skips the next instruction in the memory by incrementing the program counter by one. RTL for the given instruction is:

D_6T_8:	DR← M[AR]

D_6T_9:	DR ←DR + 1
D_6T_{10}:	M[AR] ← DR, if (DR = 0) then (PC ←PC + 1), SC ←0

OR <address>

This instruction copies the operand specified by the address to the data register during the first cycle of its execution. On the next cycle, this data is then ORed with the content of the accumulator and the result is stored in the accumulator. The OR operation is performed inside the ALU. RTL for this instruction is:

D_7T_8:	DR← M[AR]
D_7T_9:	AC←AC∨DR, SC ←0

XOR <address>

This instruction copies the operand specified by the address to the data register during the first cycle of its execution. On the next cycle, this data is then XORed with the content of the accumulator and the result is stored in the accumulator. The XOR operation is performed
inside the ALU. RTL for this instruction is:

D_8T_8:	DR← M[AR]
D_8T_9:	AC←AC⊕DR, SC ←0

XCHG <address>

This instruction copies the operand specified by the address to the data register during the first cycle of its execution. On the next cycle, this data is then exchanged with the content of the accumulator through the ALU. RTL for this instruction is:

D_9T_8:	DR← M[AR]
D_9T_9:	M[AR] ← AC, AC ← DR, SC ←0

SUB <address>

This instruction completes its execution in six cycles. During the first cycle, it copies the operand specified by the address to the data register. During the second cycle, the content of the accumulator is copied to the temporary register to prevent the loss of the data in the accumulator. Then, during the third cycle, the content of the data register is transferred to the accumulator. During the fourth cycle, the content of the accumulator is complemented and the result is stored back in the accumulator itself. During the fifth cycle, the content of the accumulator is incremented by 1 and the result is again stored back to the accumulator. Finally, the content of the temporary register is added to the current content of the accumulator. In this way, subtraction is performed by adding the content of the accumulator with the 2's complement of the operand specified by the address in the memory.

$D_{10}T_8$:	DR← M[AR]
$D_{10}T_9$:	TR← AC
$D_{10}T_{10}$:	AC ← DR
$D_{10}T_{11}$:	AC ← AC'
$D_{10}T_{12}$:	AC ← AC + 1, DR ← TR
$D_{10}T_{13}$:	AC ← AC + TR, SC ←0

2.6.3.3. Input-Output Instructions

Here, the instructions are referenced to and from input/output interfaces.

$D_{15}IT_3 = p$ (Common to all I/O instruction)

INP

This is an input instruction. It causes the computer to accept 4-bit data from the input register and clears the input flag. Clearing the input flag FGI indicates that the data has been accepted so that the inputting device can provide the next 4- bits of data.

pB_{11}	AC ← INPR, FGI← 0

OUT

This is an output instruction. It causes the computer to send 4-bit of data stored in the AC to be transferred to the output register and clear the output flag. Clearing the output flag indicates to the external reading device that the content is ready in the output register to be copied. Once the data is copied by the external device from the output register, the output flag is set allowing the computer to provide the next 4-bits of data.

pB_{10}	OUTR← AC, FGI← 0

SKI

Skip on the input flag causes the computer to skip the next instruction if the input flag is set.

pB_9	If(FGI==1) then PC← PC+1

SKO

Skip on the output flag causes the computer to skip the next instruction if the output flag is set.

pB_8	If(FGO==1) then PC← PC+1

ION

This instruction sets the interrupt to enable flip flops.

pB_7	IEN←1

IOF

This instruction resets the interrupt to enable a flip flop.

pB_6	IEN←0

2.7. I/O and Interrupt

The input and output devices are needed to make the communication in the computer. The terminal sends and receives serial information. The serial information from the keyboard is shifted in the INPR register and the serial information for the printer is stored in the OUTR register. These two registers communicate serially and with AC in parallel. The flag is needed to synchronize the timing difference between the input device and the computer. When the key is pressed on the keyboard, the 9 bits alphanumeric code is shifted in INPR and the FGI flip flop is set to 1.Then the computer checks if the FGI is 1, the information from INPR is transferred in parallel to AC and FGI is cleared to 0. If the FGI is 0 then new information can be shifted in INPR by pressing a key.

The OUTR register has the FGO flip flop set to 1. The computer checks if FGO is 1, the information from AC is transferred in parallel to OUTRand FG0 is cleared to 0. If FGO is 0 then the new character cannot be loaded by the computer as it indicates a character being printed.

2.7.1. Interrupt Cycle

Interrupt occurs in open communication when some data has to be passed i.e input is being fetched on INPR or transferring output from OUTR. The I/O interface watches the I/O devices instead of the CPU. When the interface finds that the I/O device is ready for data transfer, it generates an interrupt request to the CPU.

When an interrupt is detected, the CPU stops momentarily the work it is doing, branches to the service routine to process the transfer, and then returns to the task it was performing.

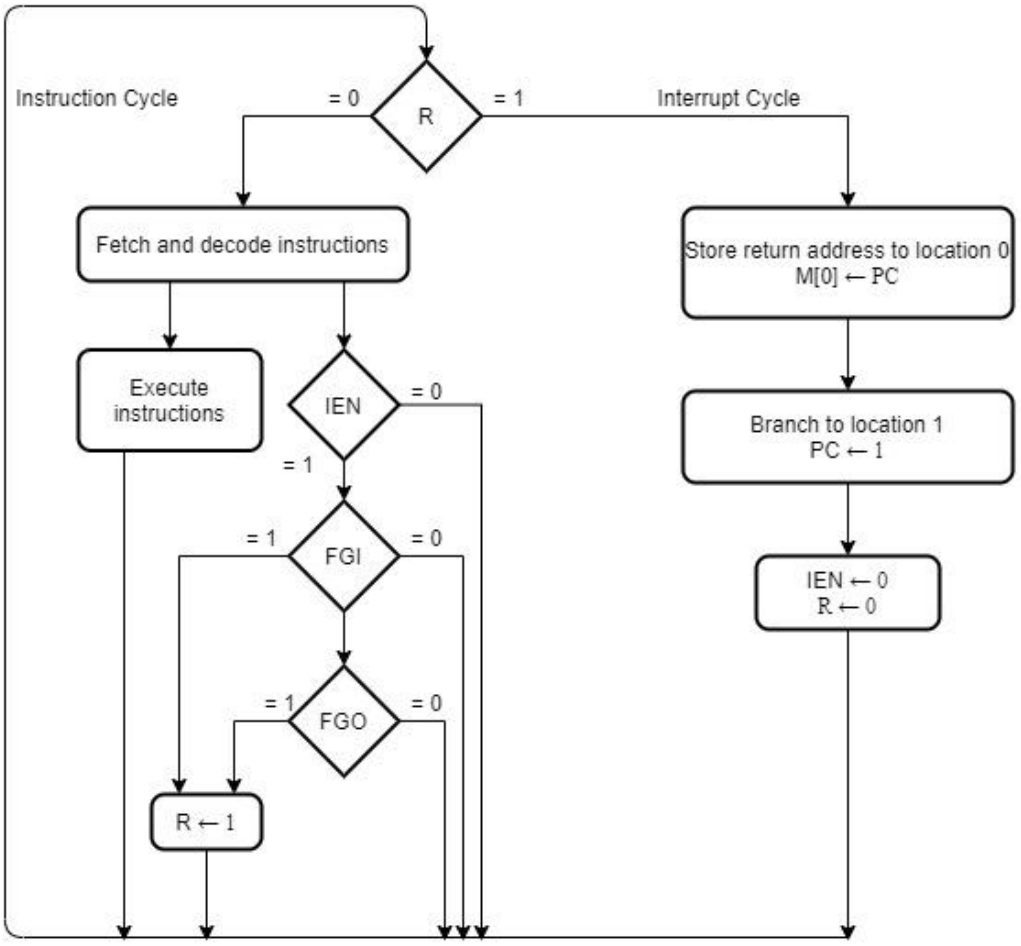

Figure 14: Flowchart of Interrupt Cycle

The way that the interrupt is handled by the computer can be explained using the flowchart of Figure 14**.** An interrupt flip-flop R is included in the computer. When R=0, the computer goes through an instruction cycle. During the execution phase of

the instruction cycle, IEN is checked by the control. If it is 0, it indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle. If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for the transfer of information. In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN =1, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

2.7.2. Register Transfer Operations in Interrupt Cycle

The interrupt cycle is initiated after the last execution phase if the interrupt flip flop R is equal to 1. This flip-flop is set to 1 if IEN =1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T_0, T_1, or T_2 are active. The condition for setting flip-flop R to 1 can be expressed with the following register transfer statement:

$$T_0T_1T_2(\text{IEN})(\text{FGI FGO}): R \leftarrow 1$$

The symbol + between FGI and FGO in the control function designates a logic OR operation. This is ANDed with IEN and $T_0'T_1'T_2'$.

We now modify the fetch and decode phases of the instruction cycle. Instead of using only timing signals T_0, T_1, and T_2 (as shown in ***Figure:13***)we will AND the three-timing signals with R so that the fetch and decode phases will be recognized from the three control functions RT_0, RT_1, and RT_2. The reason for this is that after the instruction is executed and SC is cleared to 0, the control will go through a fetch phase only if R=0. Otherwise, if R=1, the control will go through an interrupt cycle. The interrupt cycle stores the return address (available in PC) into memory location 0, branches to memory location 1, and clears IEN, R, and SC to 0. This can be done with the following sequence of microoperations:

RT_0: AR ← 0, TR ← PC

RT_1: M [AR] ← TR, PC ← 0

RT_2: PC ← PC+1, IEN ← 0, R ← 0, SC ← 0

During the first timing signal, AR is cleared to 0, and the content of the PC is transferred to the temporary register TR. With the second timing signal, the return address is stored in memory at location 0 and the PC is cleared to 0. The third timing signal increments PC to 1, clears IEN and R, and control goes back to T0 by clearing SC to 0. The beginning of the next instruction cycle has the condition RT_0 and the content of the PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

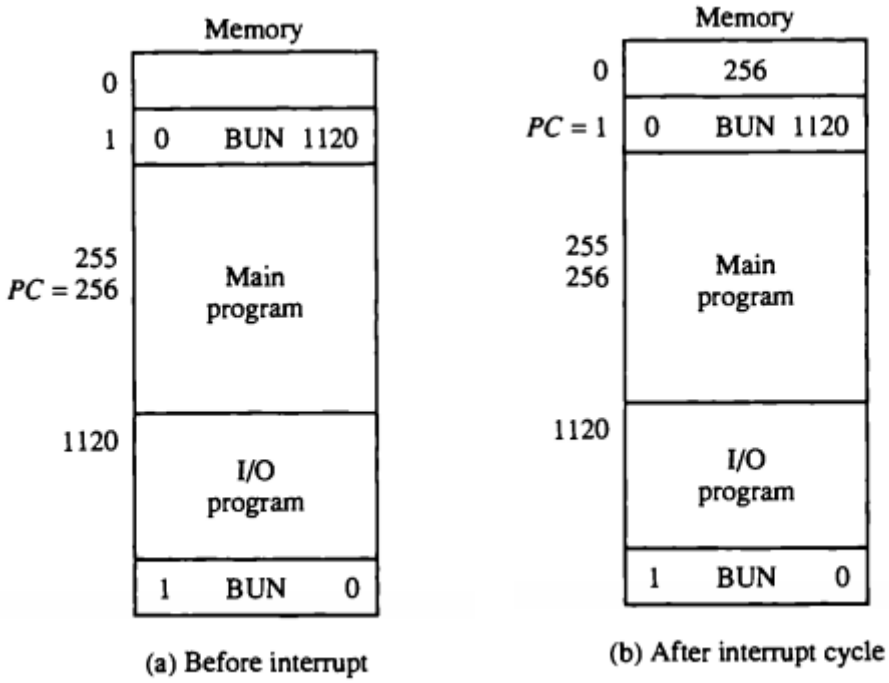

Figure 15: Demonstration of interrupt cycle

2.8. Complete Computer Description

Finally, from all the above design considerations, we design the SKIP computer, which is explained below.

2.8.1. Components

The following table gives the components used in the SKIP computer and their count.

S.N.	Component	Count
1.	Memory(2K x 18)	1
2	Registers	9
3	MUX (8x1)	18
4	Encoders (8x3)	1
5	Flags	5
6	Full Adders	18
7	Decoders(4 x 16), (4 x 16), (2 x 4)	3

Table 7: Components

2.8.2. Flowchart of Operations

The complete flowchart of operations including both the instruction cycle and the interrupt cycle is shown in figure 16 below.

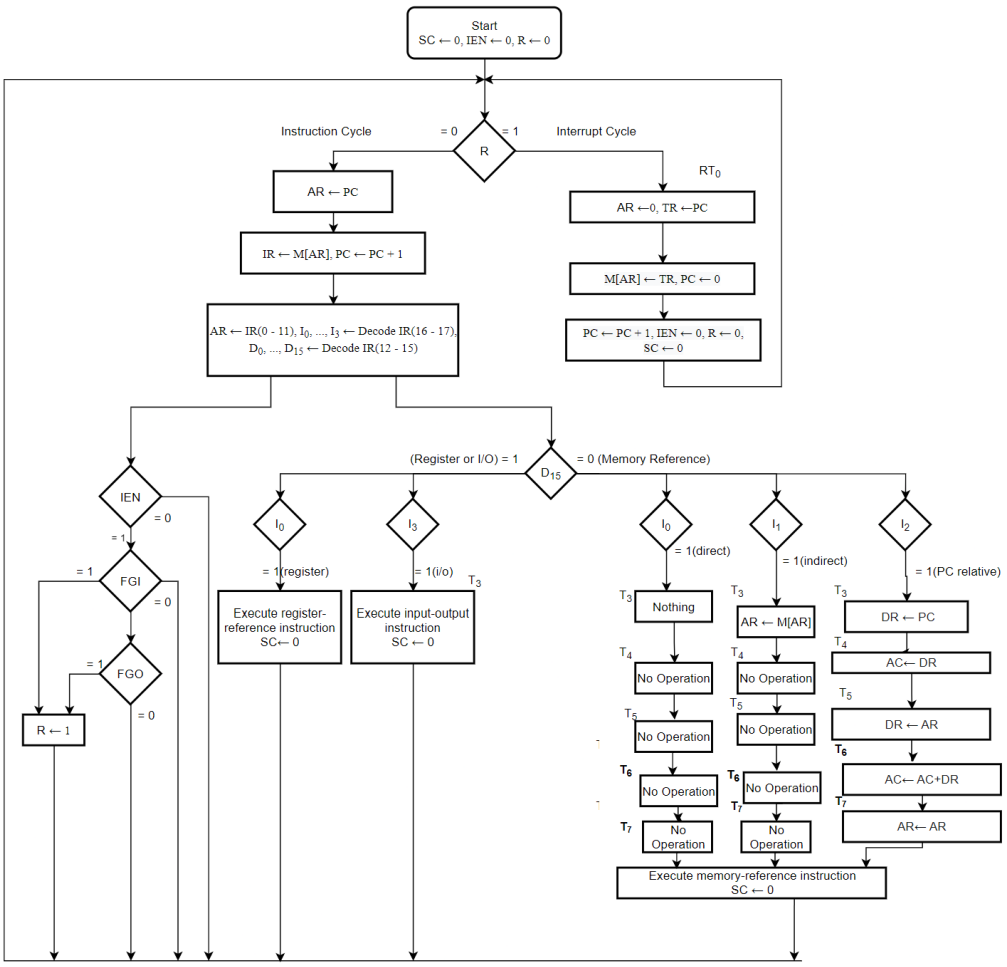

Figure 16: Flowchart of Operations

2.8.3. Micro operations

The Following represents the control functions and micro-operations for the SKIP computer.

Fetch

$R'T_0$:	AR ← PC
$R'T_1$:	IR ← M[AR], PC ← PC + 1

Decode

$R'T_2$:	$D_0, \ldots, D_{15} \leftarrow$ Decode IR(12-15), $I_0, \ldots, I_3 \leftarrow$ Decode IR(16-17), AR ← IR(0-11)

Indirect

$D_{15}'I_1T_3$:	AR ← M[AR]

PC Relative

$D_{15}'I_2T_3$:	DR ← PC
$D_{15}'I_2T_4$:	AC ← DR
$D_{15}'I_2T_5$:	DR ← AR
$D_{15}'I_2T_6$:	AC ← AC + DR
$D_{15}'I_2T_7$:	AR ← AC

Interrupt

$T_0'T_1'T_2'$(IEN)(FGI+FGO):	R ← 1
RT_0:	AR ← 0, TR ← PC
RT_1:	M[AR] ← TR, PC ← 0
RT_2:	PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

Memory Reference

AND	D_0T_8:	DR ← M [AR]

	D_0T_9:	AC←AC ^ DR, SC ←0
ADD	D_1T_8:	DR ← M [AR]
	D_1T_9:	AC←AC+DR, E ←Cout, SC ←0
LDA	D_2T_8:	DR ← M [AR]
	D_2T_9:	AC ←DR, SC ←0
STA	D_3T_8:	M[AR] ← AC, SC ← 0
BUN	D_4T_8:	PC ← AR, SC ←0
BSA	D_5T_8:	M [AR] ← PC, AR ← AR + 1
	D_5T_9:	PC ←AR, SC ←0
ISZ	D_6T_8:	DR← M[AR]
	D_6T_9:	DR ←DR + 1
	D_6T_{10}:	M[AR] ← DR, if (DR = 0) then (PC ←PC + 1), SC ←0
OR	D_7T_8:	DR← M[AR]
	D_7T_9:	AC←AC∨DR, SC ←0
XOR	D_8T_8:	DR← M[AR]
	D_8T_9:	AC←AC⊕DR, SC ←0
XCHG	D_9T_8:	DR← M[AR]
	D_9T_9:	M[AR] ← AC
	D_9T_{10}:	AC ← DR, SC ←0

SUB	$D_{10}T_8$:	DR← M[AR]
	$D_{10}T_9$:	TR← AC
	$D_{10}T_{10}$:	AC ← DR
	$D_{10}T_{11}$:	AC ← AC’
	$D_{10}T_{12}$:	AC ←AC + 1, DR ← TR
	$D_{10}T_{13}$:	AC ← AC + DR, SC ←0

Register Reference

$D_{15}I_0T_3$ = r (Common to all register-reference instruction)

IR(i) = B_i(i=0,1,2...,11) Specifies the particular RRI instruction

	r	SC←0
CLA	rB_{11}	AC←0
CLE	rB_{10}	E← 0
CMA	rB_9	AC ← (AC)’
CME	rB_8	E← (E)’
CIR	rB_7	AC←shr AC, AC(17) ←E, E←AC(0)
CIL	rB_6	AC←shl AC, AC(0) ←E, E←AC(17)
INC	rB_5	AC← AC+1`
SPA	rB_4	If(AC(17)=0) then (PC←PC+1)

SNA	rB_3	If(AC(17)=1) then (PC←PC+1)
SZA	rB_2	If(AC=0) then (PC←PC+1)
SZE	rB_1	If(E=0) then (PC←PC+1)
HLT	rB_0	S ← 0(S is a start-stop flip-flop)

Input-Output

$D_{15}I_3T_3$ = p (Common to all I/O instruction)

IR(i) = Bi (i=6, 7, 8,9,10, 11) Specifies the particular IOI instruction

	p	SC←0
INP	pB_{11}	AC ← INPR,FGI← 0
OUT	pB_{10}	OUTR← AC,FGO← 0
SKI	pB_9	If(FGI==1) then PC← PC+1
SKO	pB_8	If(FGO==1) then PC← PC+1
ION	pB_7	IEN ← 1
IOF	pB_6	IEN ← 0

Table 8: Control Functions and Microperation for SKIP computer

Chapter 3: Expression and Design of Individual Unit

3.1. Control of Registers and Memory

The following are the control of registers and memory used in SKIP computers. For simplification, we have used r and p as a combination of signals such as:

r→ $D_{15}I_0T_3$

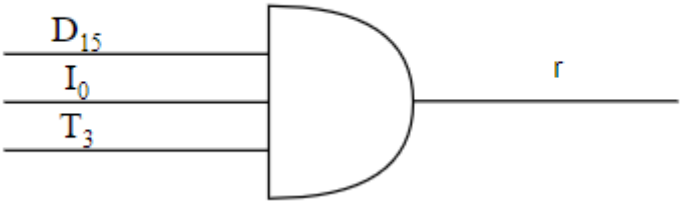

Figure 17: r Signal

p→ $D_{15}I_3T_3$

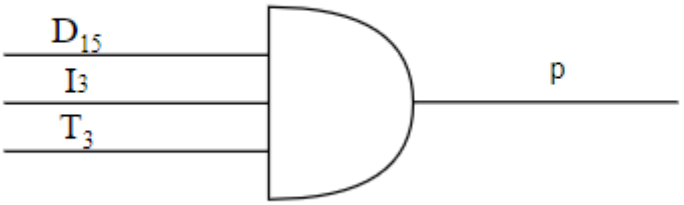

Figure 18: p Signal

3.1.1. AR

LD:

$R`T_o$: AR← PC

$R`T_2$: AR← IR(0-11)

$D_{15}`I_1T_3$: AR← M[AR]

$D_{15}`I_2T_7$: AR← AC

INR:

D_5T_8: AR← AR+1

CLR:

RT_o: AR← 0

i.e.

LD = R`$T_o$ + R`T_2 + D_{15}`$I_1T_3$ + $D_{15}$`I_2T_7

INR = D_5T_8

CLR = RT_o

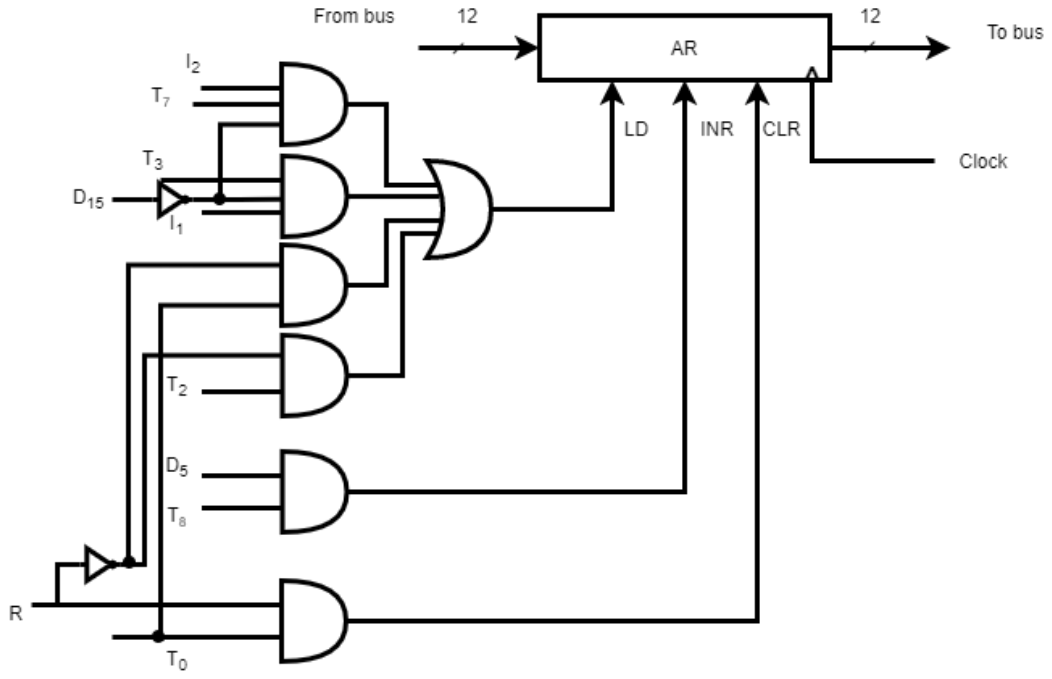

Figure 19: AR Control Signal

3.1.2. DR

LD:

D_{15}`I_2T_3: DR← AR

D_{15}`I_2T_5: DR← AR

D_0T_8: DR← M[AR]

D_1T_8: DR← M[AR]

D_2T_8: DR← M[AR]

D_6T_8: DR← M[AR]

D_7T_8: DR← M[AR]

D_8T_8: DR← M[AR]

D_9T_8: DR← M[AR]

$D_{10}T_8$: DR← M[AR]

$D_{10}T_{12}$: DR← TR

INR:

D_6T_9: DR← DR+1

i.e.

LD = $D_{15}`I_2T_3 + D_{15}`I_2T_5 + D_0T_8 + D_1T_8 + D_2T_8 + D_6T_8 + D_7T_8 + D_8T_8 + D_9T_8 + D_{10}T_8 + D_{10}T_{12}$

INR = D_6T_9

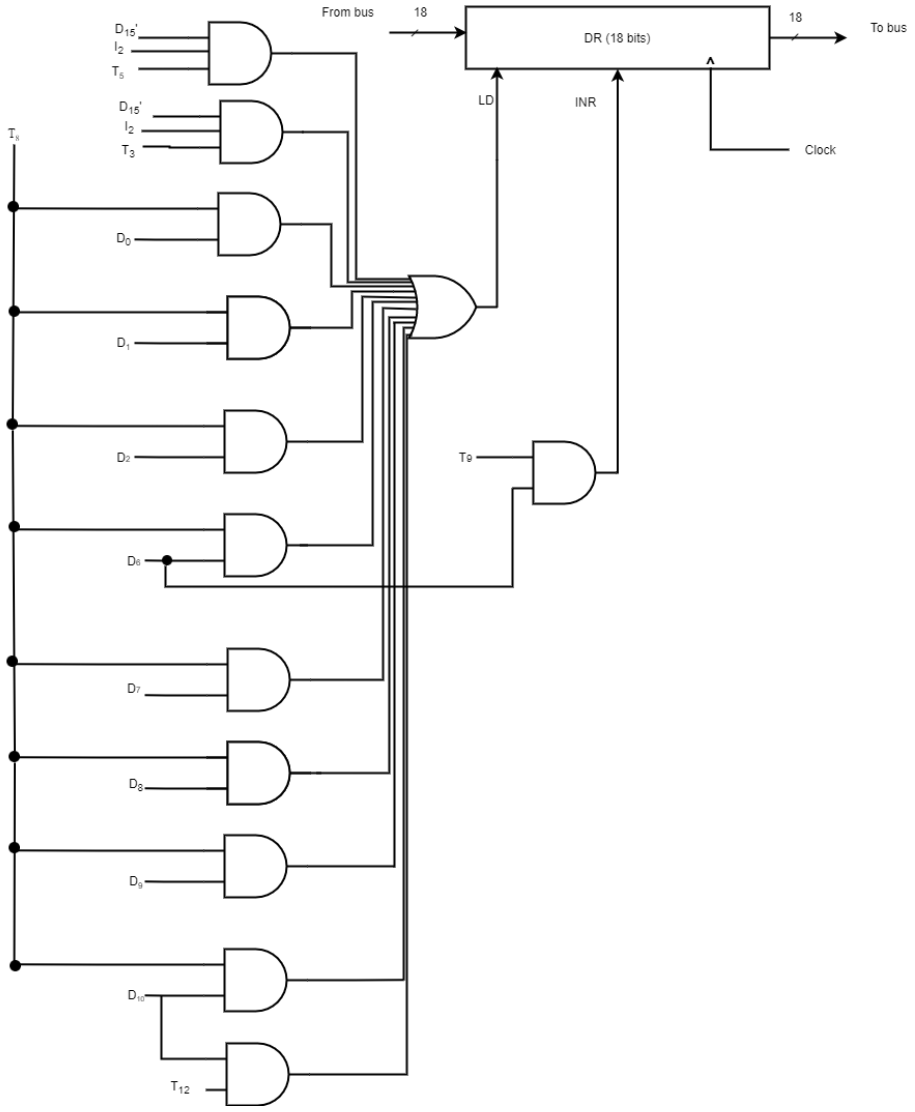

Figure 20: DR Control Signal

3.1.3. PC

LD:

D_4T_8: PC← AR

D_5T_9: PC← AR

INR:

R`T_1: PC← PC+1

RT_2: PC← PC+1

rB_4: If (AC(17)=0) then (PC← PC+1)

rB_3:If (AC(17)=1) then (PC← PC+1)

rB_2:If (AC=0) then (PC← PC+1)

rB_1:If (E=0) then (PC← PC+1)

pB_9:If (FGI=1) then (PC← PC+1)

pB_8:If (FGO=1) then (PC← PC+1)

D_6T_{10}: If (DR=0) then (PC← PC +1)

CLR:

RT_1: PC← 0

i.e.

LD = $D_4T_8 + D_5T_9$

INR = R`$T_1$ + $RT_2$ + $rB_4$(AC(17))` + rB_3(AC(17) + rB_2(AC(0) + AC(1) + AC(2) + … + AC(17))` + $rB_1$E` + pB_9FGI + pB_8FGO + D_6T_{10}(DR(0) + DR(1) + … + DR(17))`

CLR = RT_1

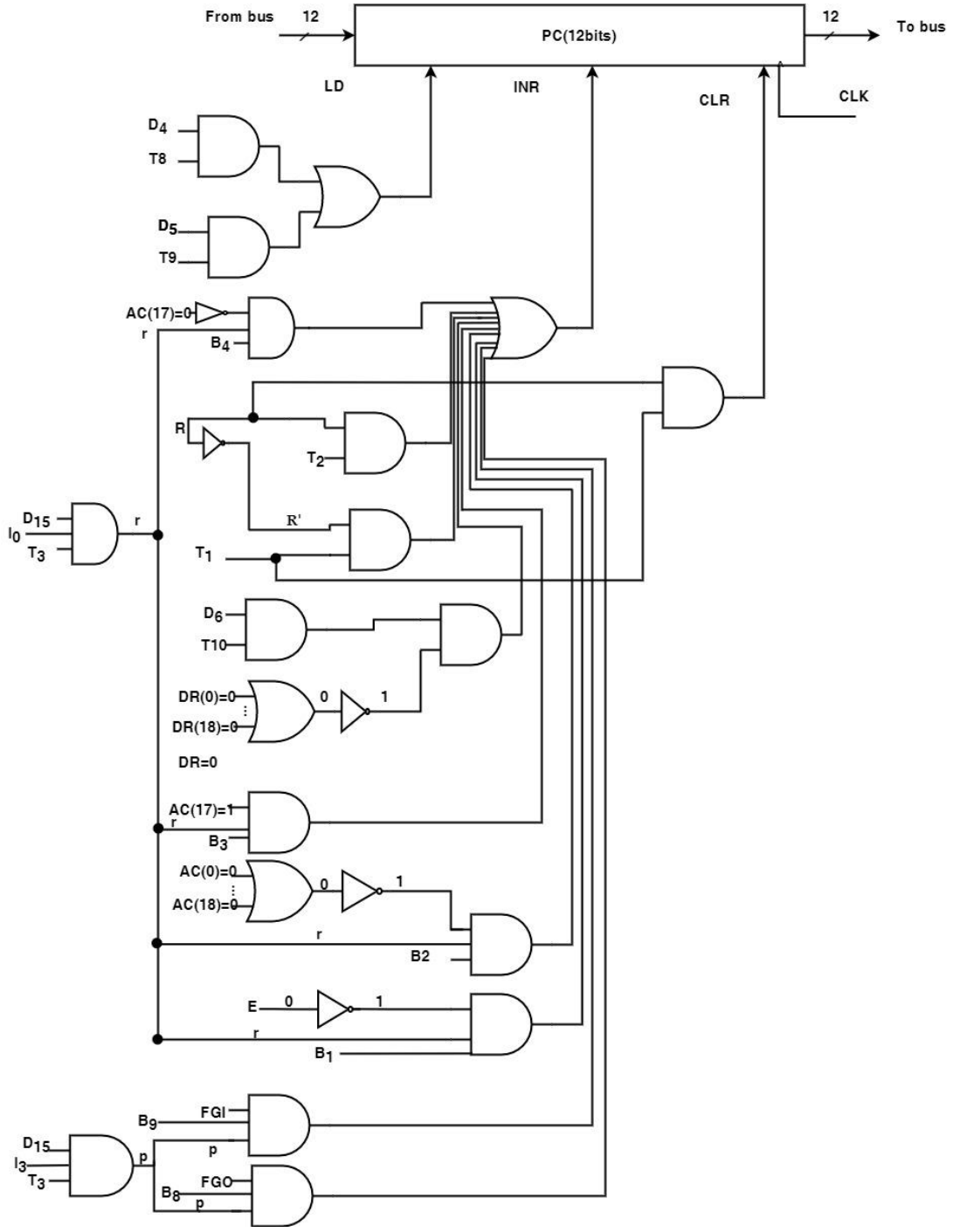

Figure 21:PC Control Signal

3.1.4. AC

LD:

$D_{15}`I_2T_4$: AC← DR

$D_{15}`I_2T_6$: AC← AC+ DR

D_0T_9: AC← AC Λ DR

D_1T_9: AC← AC+DR

D_2T_9: AC← DR

D_7T_9: AC← AC ∨ DR

D_8T_9: AC← AC ⊕DR

D_9T_{10}: AC← DR

$D_{10}T_{10}$: AC← DR

$D_{10}T_{11}$: AC← AC`

$D_{10}T_{13}$: AC← AC+DR

rB_9: AC← AC`

rB_7: AC← shr AC, AC(17) ← E

rB_6: AC← shl AC, AC(17) ← E

pB_{11}: AC← INPR

INR:

$D_{10}T_{12}$: AC← AC+1

rB_5: AC← AC+1

CLR:

rB_{11}: AC← 0

i.e.

LD = $D_{15}`I_2T_4 + D_{15}`I_2T_6 + D_0T_9 + D_1T_9 + D_2T_9 + D_7T_9 + D_8T_9 + D_9T_{10} + D_{10}T_{10} + D_{10}T_{11} + D_{10}T_{13} + rB_9 + rB_7 + rB_6 + pB_{11}$

INR = $D_{10}T_{12} + rB_5$

CLR = rB_{11}

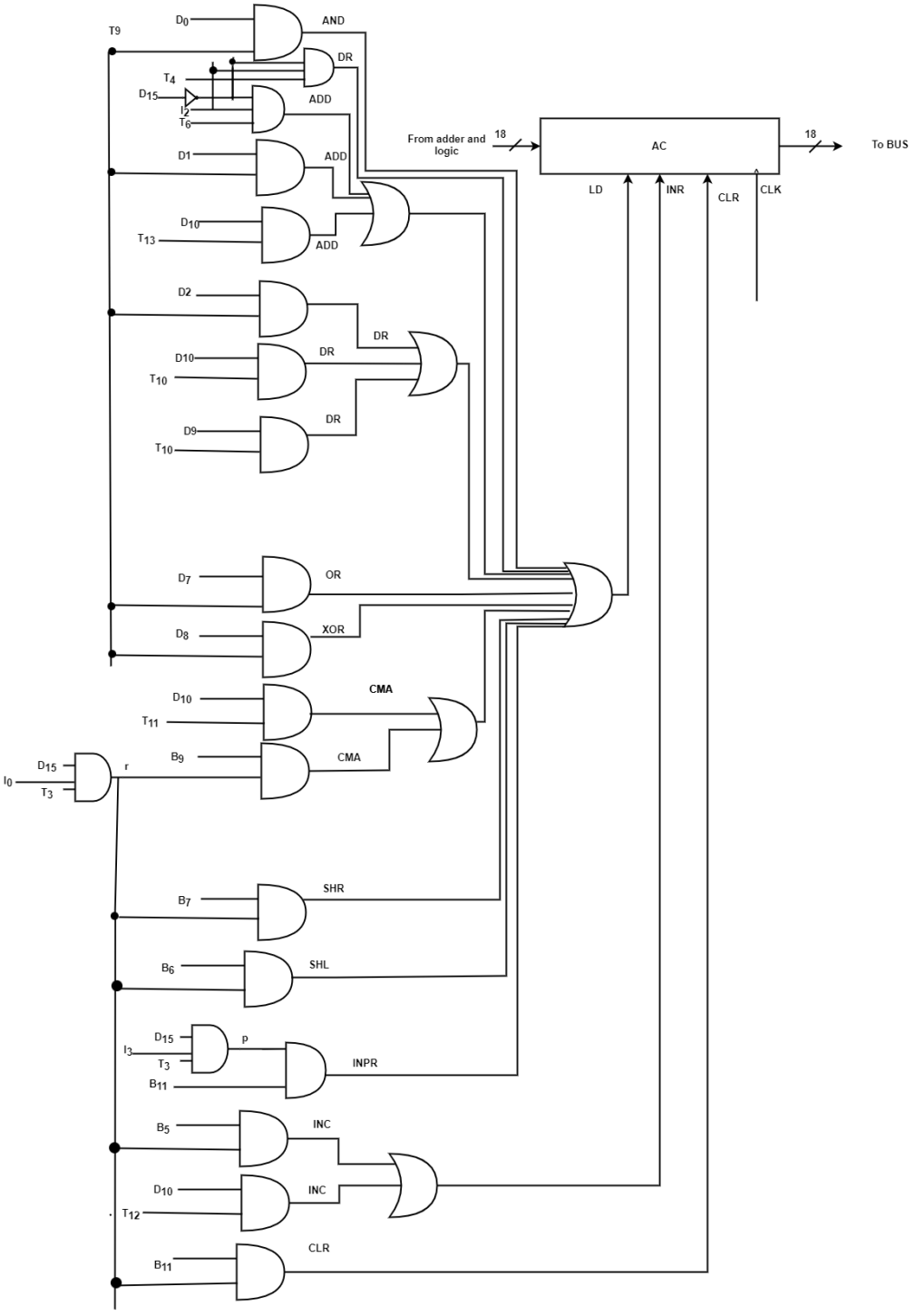

Figure 22: AC Control Signal

3.1.5. SC

CLR:

RT_2: SC← 0

D_0T_9: SC← 0

D_1T_9: SC← 0

D_2T_9: SC← 0

D_3T_8: SC← 0

D_4T_8: SC← 0

D_5T_9: SC← 0

D_6T_{10}: SC← 0

D_7T_9: SC← 0

D_8T_9: SC← 0

D_9T_{10}: SC← 0

$D_{10}T_{13}$:SC← 0

r: SC← 0

p: SC← 0

i.e. $CLR = RT_2 + D_0T_9 + D_1T_9 + D_2T_9 + D_3T_8 + D_4T_8 + D_5T_9 + D_6T_{10} + D_7T_9 + D_8T_9 + D_9T_{10} + D_{10}T_{13} + r + p$

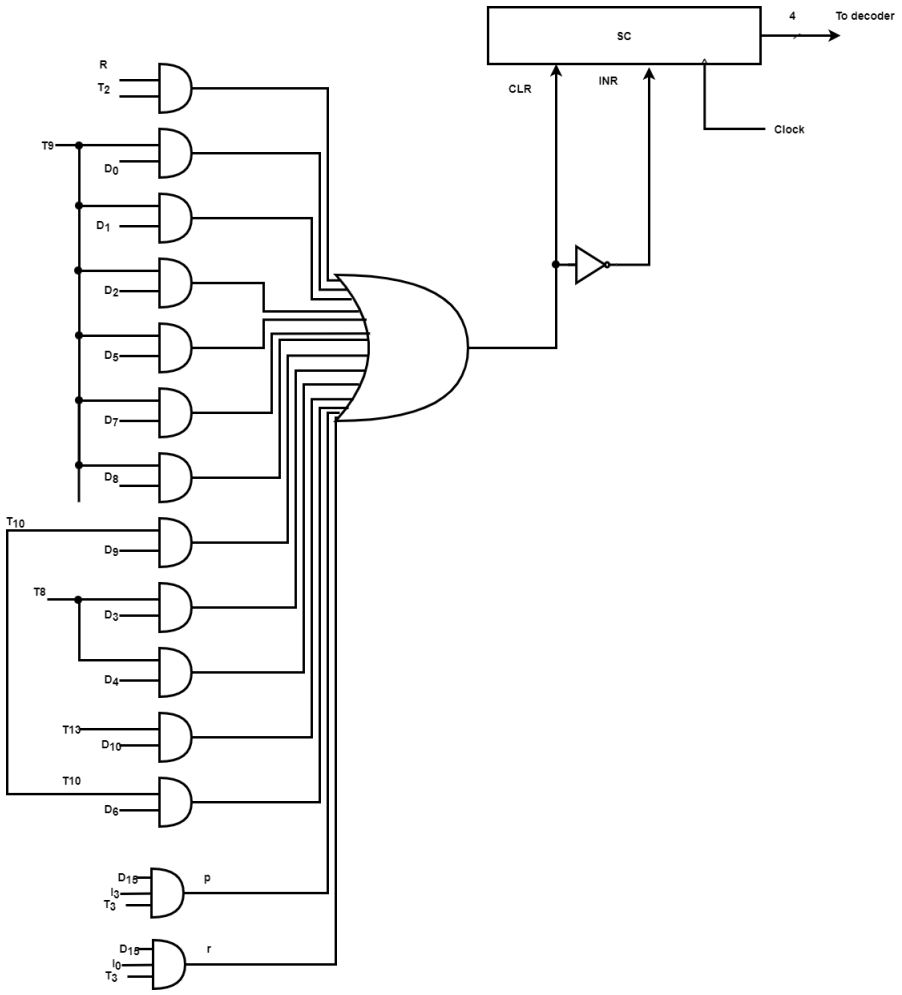

Figure 23: SC Control Signal

3.1.6. IR

LD

$R`T_1$: IR ← M[AR]

i.e.

LD = $R`T_1$

3.1.7. TR

LD

RT_0: TR← PC

$D_{10}T_9$: TR← AC

i.e.

LD = $RT_0+D_{10}T_9$

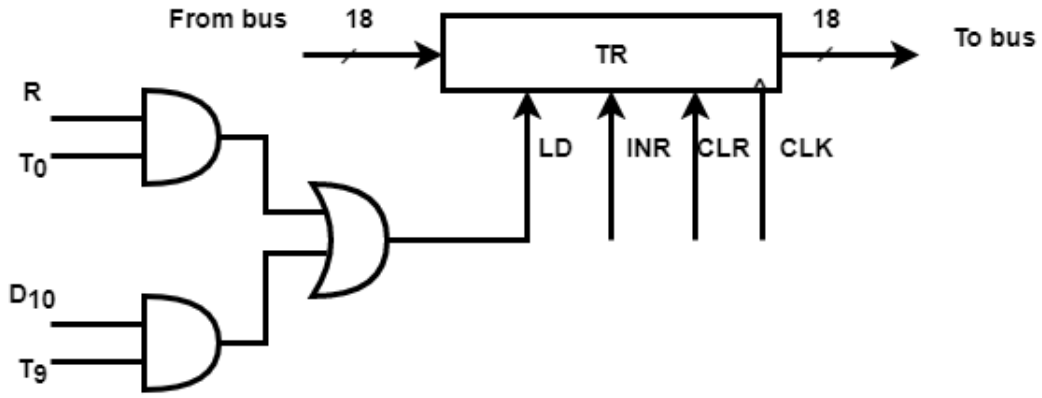

Figure 24: TR Control Signal

3.1.8 OUTR

LD

pB_{10}: OUTR← AC

i.e. **LD** = pB_{10}

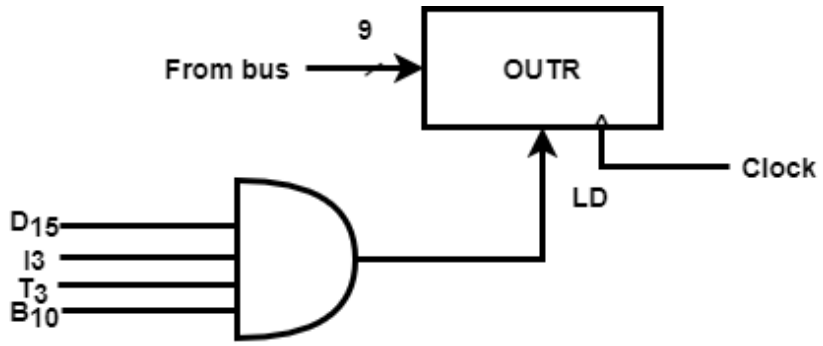

Figure 25: OUTR Control Signal

3.1.9. Memory

3.1.9.1. Memory Read:

R`T_1: IR ← M[MR]

D_{15}`I_1T_3: AR ← M[AR]

D_0T_8: DR← M[AR]

D_1T_8: DR← M[AR]

D_2T_8: DR← M[AR]

D_6T_8: DR← M[AR]

D_7T_8: DR← M[AR]

D_8T_8: DR← M[AR]

D_9T_8: DR← M[AR]

$D_{10}T_8$: DR← M[AR]

i.e. **READ** = R`$T_1$ + $D_{15}$`I_1T_3 + D_0T_8 + D_1T_8 + D_2T_8 + D_6T_8 + D_7T_8 + D_8T_8 + D_9T_8 + $D_{10}T_8$

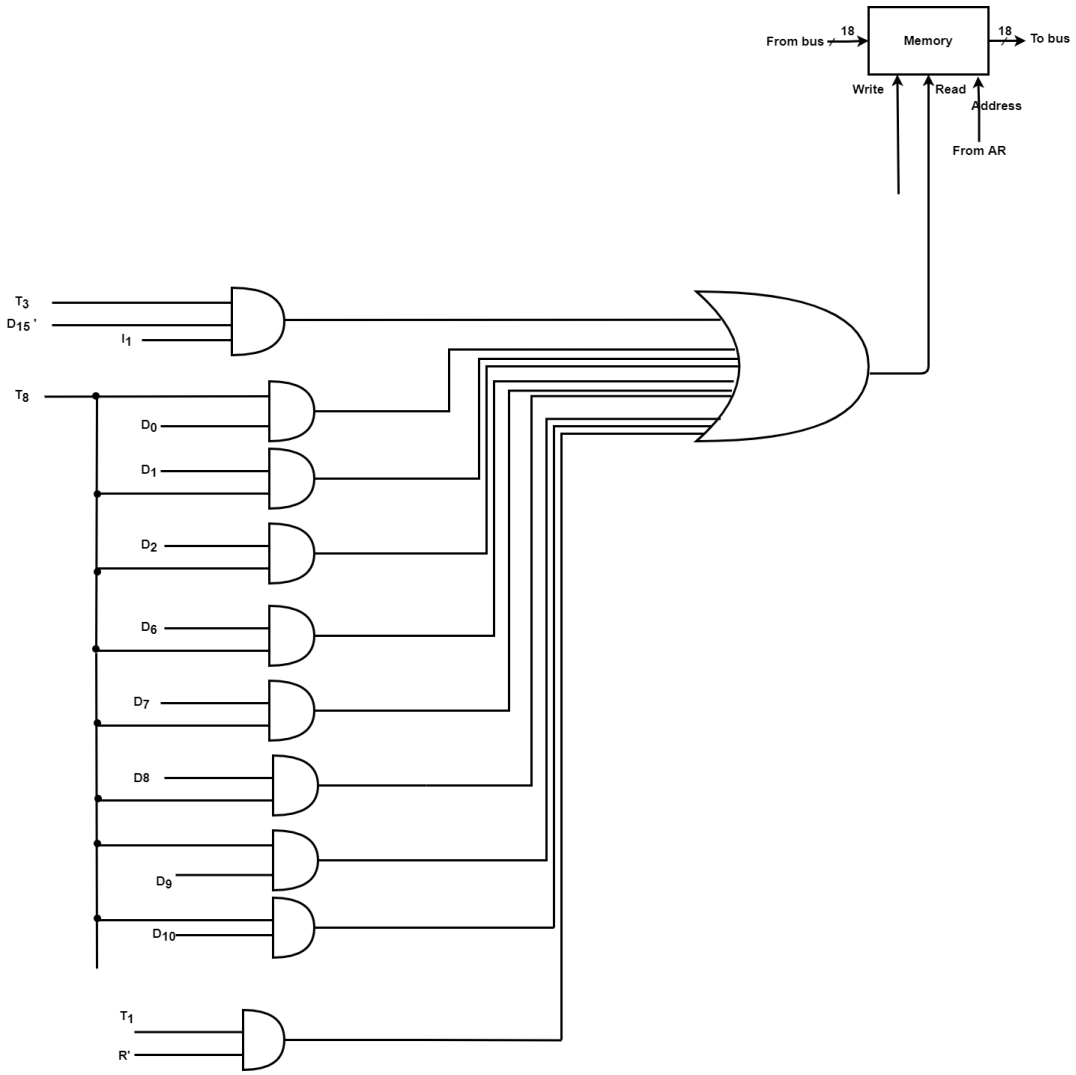

Figure 26: Memory Read Control Signal

3.1.9.2. Memory Write

RT_1: M[AR] ← TR

D_3T_8: M[AR] ← AC

D_5T_8: M[AR] ← PC

D_6T_{10}: M[AR] ← DR

D_9T_9: M[AR] ← AC

i.e. **WRITE** = $RT_1 + D_3T_8 + D_5T_8 + D_6T_{10} + D_9T_9$

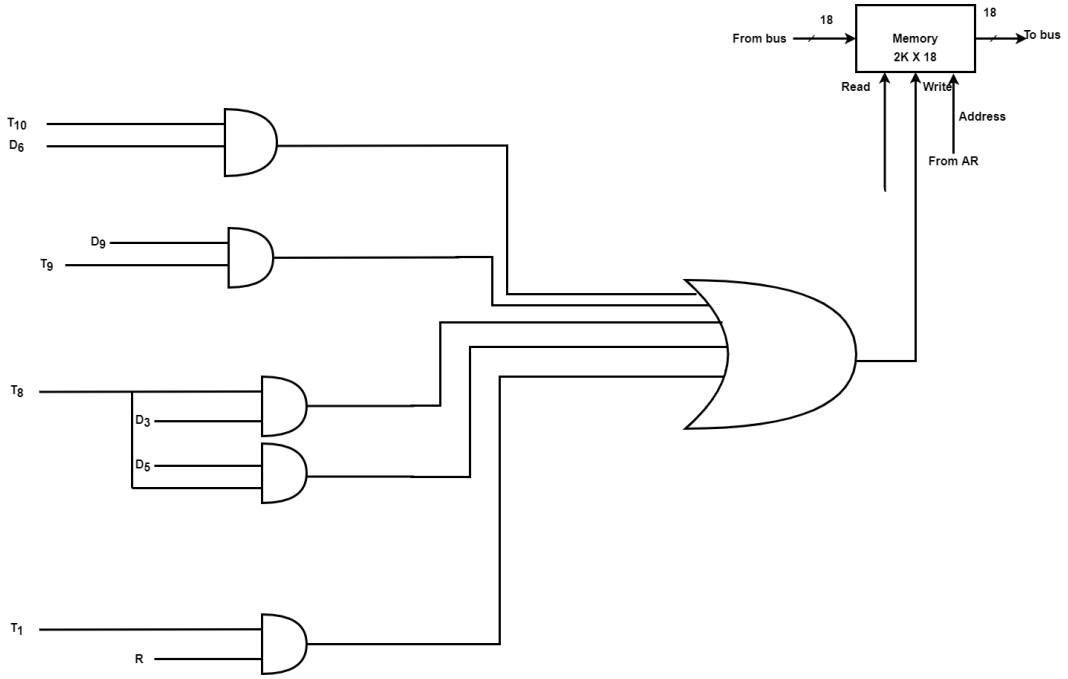

Figure 27: Memory Write Control Signal

3.2. Control of Single Flipflops

The control gates for the seven flip flops can be determined in the following manner.

3.2.1. Start-Stop Flip-flop (S)

The S may change as a result of the following control signals and operations:

rB_0: S ← 0

where $r = D_{15}I_0T_3$ and B_0 is the bit 0 of IR.

Using JK flip-flop for S, the control gate logic is shown in the following figure 28.

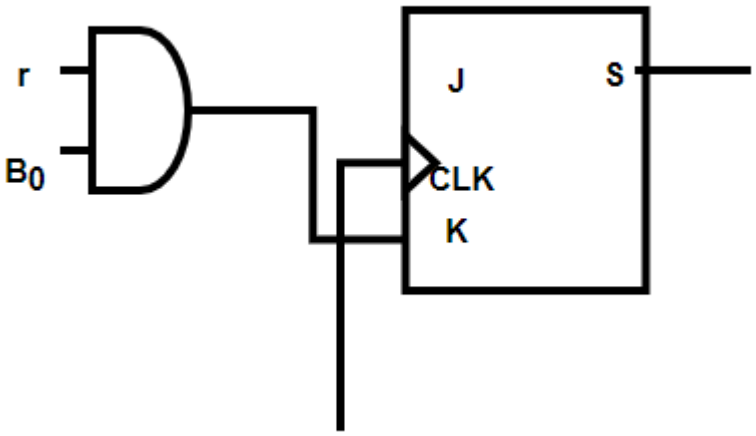

Figure 28: Control Input for S

3.2.2. End-around Carry (E)

The E may change as a result of the following control signals and operations:

rB_{10}: E ← 0

rB_8: E ← E'

rB_7: E ← AC(0)

rB_6: E ← AC(17)

D_1T_9: E ← C_{out}

Where,

$r = D_{15}I_0T_3$ and B_{10}, B_8, B_7 and B_6 are the bits 10, 8, 7 and 6 of IR respectively.

Using JK flip-flop for E, the control gate logic is shown in the following figure 29.

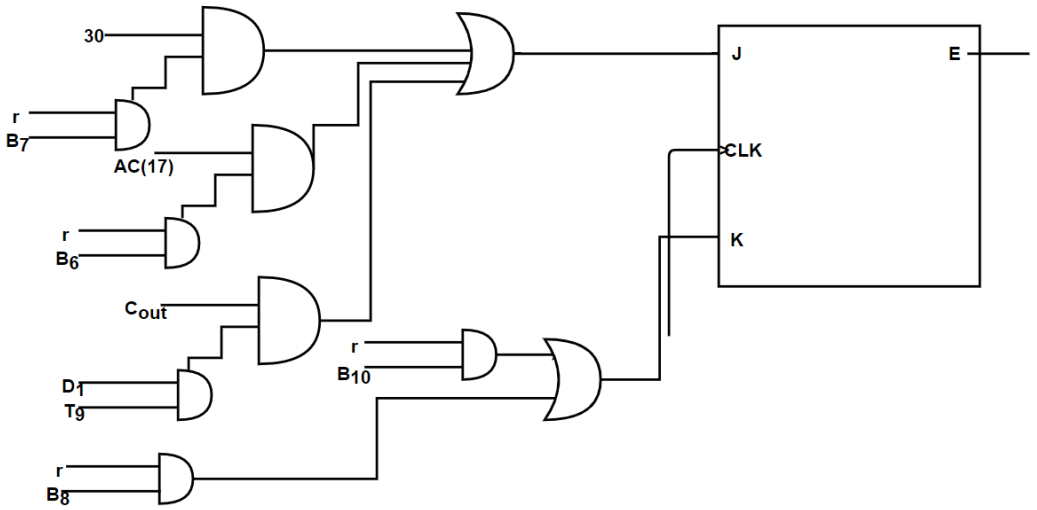

Figure 29: Control Input for E

3.2.3. Interrupt f/f (R)

The R may change as a result of the following control signals and operations:

$T_0' T_1' T_2'$ (IEN) (FGI + FGO): R ← 1

RT_2: R ← 0

Using JK flip-flop for R, the control gate logic is shown in the following figure 30.

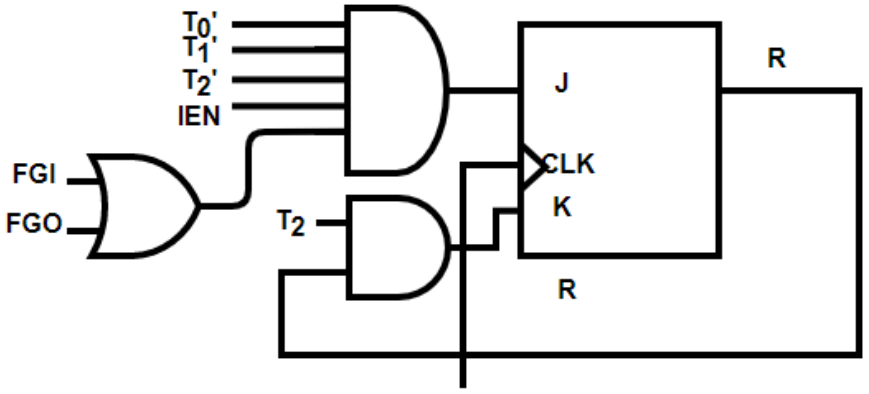

Figure 30: Control Input for R

3.2.4. Interrupt Enable (IEN)

The IEN may change as a result of the following control signals and operations:

pB_7: IEN ← 1

pB_6: IEN ← 0

RT_2: IEN ← 0

Where,

$p = D_{15}I_3T_3$ and B_7 and B_6 are the bits 7 and 6 of IR.

Using JK flip-flop for IEN, the control gate logic is shown in the following figure 31.

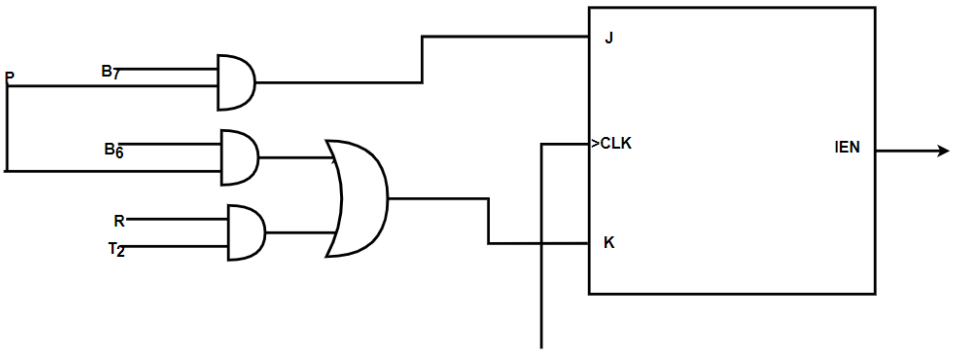

Figure 31: Control Input for IEN

3.2.5. Input Flag (FGI)

The FGI may change as a result of the following control signals and operations:

pB_{11}: FGI← 0

Where,

$p = D_{15}I_3T_3$ and B_{11} is the bit 11 of IR.

Using JK flip-flop for FGI, the control gate logic is shown in the following figure 32.

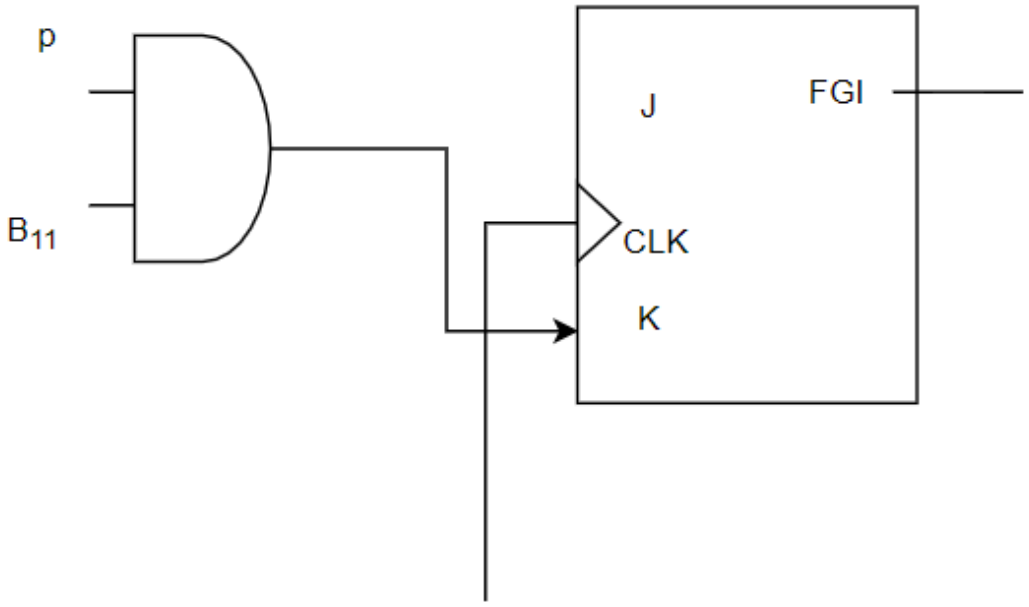

Figure 32: Control Input for FGI

3.2.6. Output Flag (FGO)

The FGO may change as a result of the following control signals and operations:

pB_{10}: FGI← 0

Where,

$p = D_{15}I_3T_3$ and B_{10} is the bit 10 of IR.

Using JK flip-flop for FGO, the control gate logic is shown in the following figure 33.

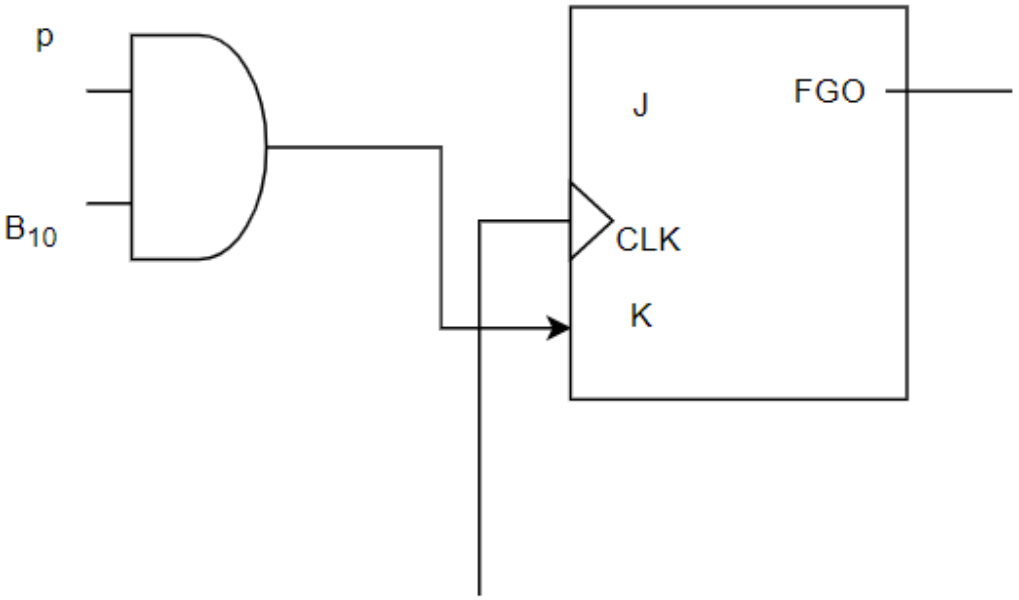

Figure 33: Control Input for FGO

3.3. Control of Common Bus

The 18-bit common bus is controlled by the selection inputs S_2, S_1, S_0. The binary numbers for $S_2S_1S_0$ select each register. Each binary number is associated with a Boolean variable X_1 through X_6.

X_1	X_2	X_3	X_4	X_5	X_6	X_7	S_2	S_1	S_0	Register selected for bus
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	0	1	TR
0	0	0	0	0	0	1	1	1	0	Memory

Table 9: Encoder for Bus Selection Circuit

The Boolean functions for the encoders are:

$S0 = X_1 + X_3 + X_5$

$S1 = X_2 + X_3 + X_6$

$S2 = X_4 + X_5 + X_6$

To find the logic that makes $X_1=1$, we extract all those statements with **AR** as a source.

D_4T_8: PC ← AR

D_5T_9: PC ← AR

$D_{15}'I_2T_5$: DR ← AR

So, the boolean function for X_1 is

$$\mathbf{X_1 = D_4T_8 + D_5T_9 + D_{15}'I_2T_5}$$

To find the logic that makes $X_2=1$, we extract all those statements with **PC** as a source.

D_5T_8: M[AR] ← PC

$R'T_0$: AR ← PC

$D_{15}'I_2T_3$: DR← PC

RT_0: TR ← PC

So, the boolean function for X_2 is

$$\mathbf{X_2 = D_5T_8 + RT_0 + R'T_0 + D_{15}'I_2T_3}$$

To find the logic that makes X_3=1, we extract all those statements with **DR** as a source.

D_6T_{10}: M[AR] ← DR

D_9T_{10}: AC ← DR

D_2T_9 : AC ←DR

$D_{10}T_{10}$: AC ← DR

$D_{15}'I_2T_4$:AC ← DR

So, the boolean function for X_3 is

$$\mathbf{X_3 = D_6T_{10} + D_9T_{10} + D_2T_9 + D_{10}T_{10} + D_{15}'I_2T_4}$$

To find the logic that makes X_4=1, we extract all those statements with **AC** as a source.

D_3T_8: M[AR] ← AC

D_9T_9: M[AR] ← AC

$D_{10}T_9$: TR← AC

pB_{10}:OUTR←AC

$D_{15}'I_2T_7$:AR←AC

So, the boolean function for X_4 is

$$\mathbf{X_4 = D_3T_8 + D_9T_9 + D_{10}T_9 + pB_{10} + D_{15}'I_2T_7}$$

To find the logic that makes X_5=1, we extract all those statements with **IR** as a source.

$R'T_2$:D0, … ,D15← Decode IR(12-15), I0, …, I3 ← Decode IR(16-17), AR ← IR(0-11)

So, the boolean function for X_5 is

$$\mathbf{X_5 = R'T_2}$$

To find the logic that makes X_6=1, we extract all those statements with **TR** as a source.

$D_{10}T_{12}$: DR ← TR

RT_1: M[AR] ← TR

So, the boolean function for X_6 is

$\mathbf{X_6 = D_{10}T_{12} + \ RT_1}$

Now for finding the logic that makes X_7=1, we extract all those statements that have **Memory** as a source.

D_0T_8: DR ← M [AR]

D_1T_8: DR ← M [AR]

D_2T_8: DR ← M [AR]

R'T1: IR ← M[AR]

D_6T_8:DR ← M[AR]

D_7T_8:DR ← M[AR]

D_8T_8: DR ← M[AR]

D_9T_8: DR ← M[AR]

$D_{10}T_8$: DR ← M[AR]

$D_{15}'I_1T3$: AR ← M[AR]

So, the boolean function for X_7 is

$\mathbf{X_7 = R'T_1 + D_0T_8 + D_1T_8 + D_2T_8 + R'T_1 + D_6T_8 + D_7T_8 + D_8T_8 + D_9T_8 + D_{10}T_8 + D_{15}'I_1T_3}$

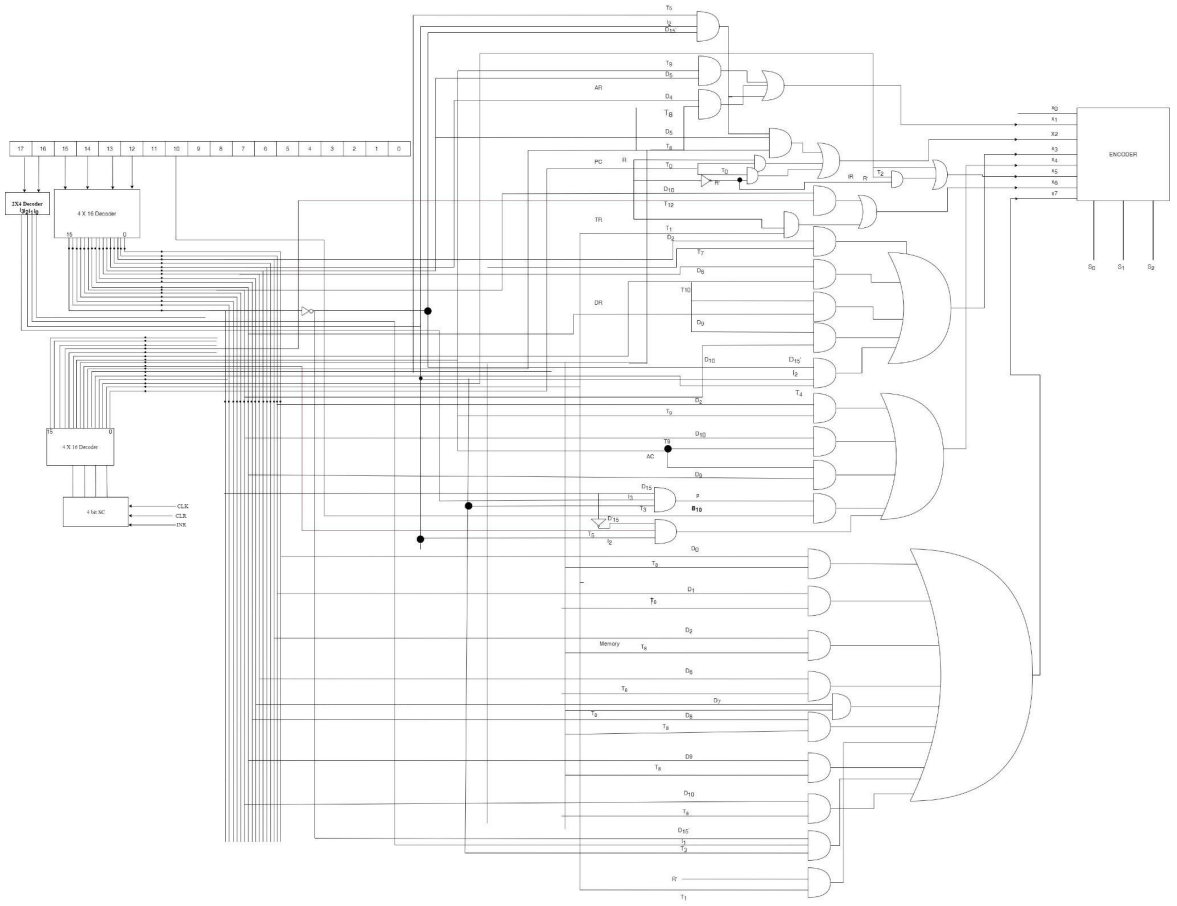

Figure 34: Control of Common Bus

3.4. ALU (Adder and Logic Circuit)

We have a total of 17 AND gates, and these AND gates are achieved by ANDing AC with corresponding bit in the Data Register.The ADD operator is achieved similarly by adding the Binary adder as shown in the figure. One stage of the adder uses the full-adder with the corresponding input and output carries.

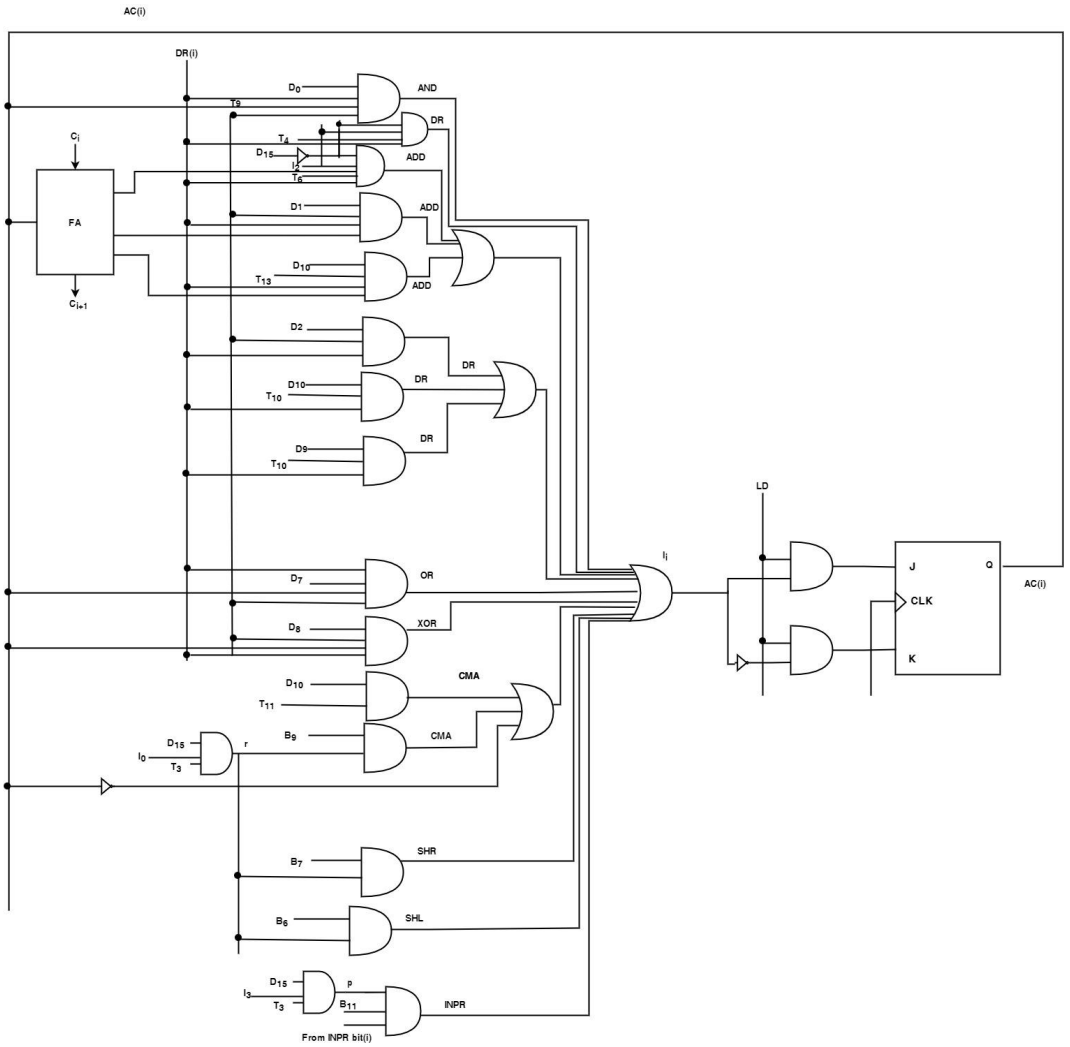

Figure 35: One Stage of ALU

Chapter 4: Conclusion

Designing an **18 bit CPU** was a complex and yet essential process needed to implement all the ideas of Computer System Architecture together. A thorough understanding of computer organization and architecture was necessary to complete this task.

After assembling all the hardware components accordingly in such a way that the flow of instructions that we had set i.e 29 different instructions could run properly without causing any problems, was not as easy as we thought when we began the first step of this project. But as working gradually, assembling all the signal and control lines, aligning the proper logic gates helped us to achieve the successful design of the computer which we named SKIP**.**