Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Mini Project Report

on

"SPAN COMPUTER DESIGN"

[Code No: COMP 315]

(For partial fulfillment of 3rd Year/1st Semester in Computer Engineering)

Submitted By

Nitin Ghimire(18)

Pranish Kafle (26)

Aayush Man Shakya (48)

Sumesh Shrestha (52)

Submitted To:

Mr Pankaj Raj Dawadi

Assistant Professor

Department of Computer Science and Engineering

Submission Date: 08-06-2024

Acknowledgement

We are grateful for the opportunity to participate in this project. We would like to express our sincere gratitude to our esteemed teacher, Mr. Pankaj Dawadi, for incorporating this mini-project into our COMP 315 syllabus. We appreciate the chance to explore and enhance our skills and knowledge in computing. We have tried to deliver our best in this project.

Sincerely,

Nitin Ghimire(18)

Pranish Kafle (26)

Aayush Man Shakya (48)

Sumesh Shrestha (52)

List of Tables

List of Figures

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 22 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 different instructions based on the suitability of our project. Our computer has a 4-bit opcode, 16-bit address length and 2-bit addressing mode.

Our computer consists of the following hardware components:

1. A memory of 65536 x 22.
2. Nine registers: AR(16), PC(16), DR(22), AC(22), IR(22), TR(22), 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 22-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

An instruction is divided into three parts:

Operation Code (Opcode): It specifies the operation for an instruction.

Address: It specifies the registers and/or memory locations used in an operation.

In the SPAN computer, memory contains 65536(2^16) words, requiring 16 bits to specify an address. Each word is 22 bits, with 16 least significant bits (LSBs) reserved for the address, the 2 most significant bits (MSBs) for addressing modes, and the remaining 4 bits for opcodes.

2.2. Addressing Modes

The SPAN computer uses two addressing modes:

	Addressing Mode Field	Addressing Mode
I_0	00	Direct Addressing Mode
I_1	01	Indirect Addressing Mode
I_2	10	Register Reference Instruction Mode
I_3	11	Input Output Instruction Mode

Table 1: 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.

2.2.2. Indirect Addressing Mode

The address of the operand is stored within the instruction itself in the indirect addressing mode.

2.3. Component

2.3.1. Registers

The SPAN Computer has nine registers:

Instruction Register (IR)

Size: 22 bits

IR holds 22 bits of instruction read from memory. The address in the PC is transferred to the Address Register (AR), and the content of the memory location pointed to by the AR is then transferred to the IR via the common bus system (CBS). The instruction is decoded to provide the necessary control signals for execution.

Program Counter (PC)

Size: 16 bits

It contains the address of the next instruction to be executed. When a byte (machine code) is fetched, the PC is incremented by one. So that it can fetch the next instruction. If the computer is reset or restarted, the program counter returns to zero value. For subroutines, the PC is saved and loaded with the subroutine's address, resuming with the saved address after the subroutine call.

Data Register (DR)

Size: 22 bits

It stores the operand read from memory to be processed. The output of the DR is an input to the Arithmetic and Logic Unit (ALU).

Accumulator (AC)

Size: 22 bits

It stores the result of operations. The input comes from the ALU's output and the out of the operation gets stored in the accumulator.

Input Register (INPR)

The Input Registers (INPR) holds the input characters given by the user.

Output Register (OUTR)

The Output Registers (OUTR) holds the output after processing the input data.. The output can be connected to an external display.

Temporary Register (TR)

Size: 22 bits

It holds data during arithmetic and logic operations. It is non-programmable and used for intermediate results.

Sequence Counter (SC)

Size: 4 bits

It is used to generate different Timing Signals. It is connected to a 4x16-bit decoder.

Address Register (AR)

Size: 22 bits

It is needed by processor to keep track of which locations in memory it is pointing. Denoted by AR.

2.3.2. Arithmetic and Logic Unit (ALU)

The ALU performs arithmetic and logical operations. It has the ability to perform all processes related to arithmetic and logic operations such as addition, subtraction, and shifting operations, including Boolean comparisons.

2.3.3. Memory Unit

Size: 65536 x 22

SPAN Computer has 65536 slots where 22 bit of instructions can be stored and executed in each slots. The memory receives its address from the AR and data from the Control Bus System, with read/write operations determined by control signals.

2.3.4. Common Bus System

The SPAN computer has nine registers ,a memory unit, and a control unit. The bus has a memory unit with 65536 slots each of 22 bits , Accumulator of 22 bits, Address Register of 16 bits, Program Counter of 16 bits , Instruction Register of 22 bits, Data Register of 22 bits and Temporary Register of 22 bits. Status Signals are used to choose particular registers or memory location for the Common bus system. Initially Address for the instruction to be executed is extracted from the Program Counter and then passed to Address Register through bus and then the data is fetched from memory and passed that data to the data register and though ALU specific operation is performed.

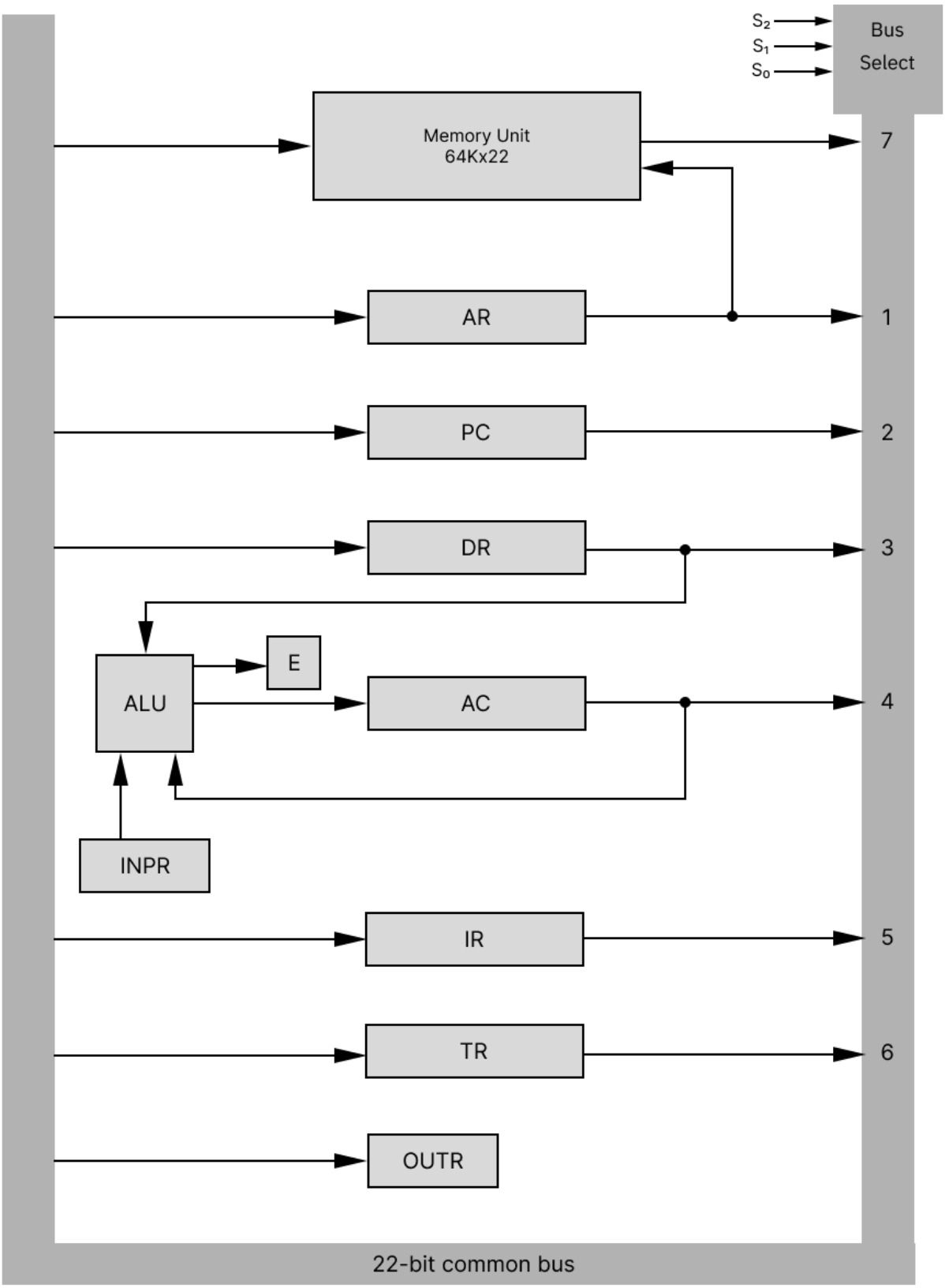

Figure 1:Common Bus System

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 Flipflop
E	End Around Carry
FGI	Input Flag
FGO	Output Flag
IEN	Interrupt Enable

Table 2: Flip Flops/F1ags

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

2.4. SPAN Instructions

Different Types of Instructions in SPAN computer are:

- Memory Reference Instruction (MRI)

  These instructions refer to the memory address as an operand. The other operand is always accumulator. 00 and 01 in Addressing modes Specifies the Memory Reference Instruction in SPAN computer.

I	OPCODE	MEMORY ADDRESS

Table 3: SPAN Instructions

For Direct Addressing,

00	OPCODE	MEMORY ADDRESS

Table 4: MRI Instructionfor Direct Addressing

For Indirect Addressing,

01	OPCODE	MEMORY ADDRESS

Table 5: MRI Instruction for Indirect Addressing

- Register Reference Instruction (RRI)

  These instructions perform operations on registers rather than memory addresses. 10 bits in Addressing mode specifies Register Reference Instruction.

10	OPCODE	MEMORY ADDRESS

Table 6: Register Reference Instruction

- Input-Output Instruction (IOI)

  These instructions are for communication between computer and outside environment. 11 in Addressing Modes specifies Input-Output Instruction

11	OPCODE	MEMORY ADDRESS

Table 7: Input-Output Instruction

2.4.1 SPAN Computer Instructions

a. Memory Reference Instructions

Symbol	**Instruction Code**		Operations
	I=00	I=01	
AND	00 0000 X	01 0000 X	AND the AC with memory word
ADD	00 0001 X	01 0001 X	Add the memory word with AC
LDA	00 0010 X	01 0010 X	Load accumulator with memory word
STA	00 0011 X	01 0011 X	Store value of AC to memory
BUN	00 0100 X	01 0100 X	Branch unconditionally
BSA	00 0101 X	01 0101 X	Branch and save return address
ISZ	00 0110 X	01 0110 X	Increment and skip if zero
OR	00 0111 X	01 0111 X	OR the AC with memory word
XOR	00 0100 X	01 0100 X	XOR the AC with memory word
XCHG	00 0101 X	01 0101 X	Exchange the AC with memory word

Table 8: Memory Reference Instructions

Here,'X' is a 16 bit memory address

b. Register Reference Instruction

Symbol	**Instruction Code**	Description
CLA	10 0000 XXXX	Clear AC
CLE	10 0001 XXXX	Clear E
CMA	10 0010 XXXX	Complement AC
CME	10 0011 XXXX	Complement E
CIR	10 0100 XXXX	Circulate Right AC and E
CIL	10 0101 XXXX	Circulate Left AC and E
INC	10 0110 XXXX	Increment AC
SPA	10 0111 XXXX	Skip next instruction if AC is positive
SNA	10 1000 XXXX	Skip next instruction if AC is negative
SZA	10 1001 XXXX	Skip next instruction if AC is zero
SZE	10 1010 XXXX	Skip next instruction if E is 0
HLT	10 1011 XXXX	Halt computer

Table 9: Register Reference Instruction

Here,'X' is a 4 bit memory address

c. Input-Output Instructions

Symbol	Instruction Code	Description
INP	10 0000 XXXX	Input character to AC
OUT	10 0001 XXXX	Output character from AC
SKI	10 0010 XXXX	Skip on input flag
SKO	10 0011 XXXX	Skip on output flag
ION	10 0100 XXXX	Interrupt Enable on
IOF	10 0101 XXXX	Interrupt Enable off

Table 10: Input-Output Instructions

Here,'X' is a 4 bit memory address

2.5 Control Unit and Timing

In the “SPAN”, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits.The timing for all registers in the basic 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 sequence counter SC can be incremented or cleared synchronously. Most of the time, the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in a while, the counter is cleared to 0, causing the next active timing signal to be T_0.

Control Unit contains three decoders two 4*16 decoder for Sequence Counter and Opcode of Instruction Register and one 2*4 decoder for Addressing Mode in Instruction Register.

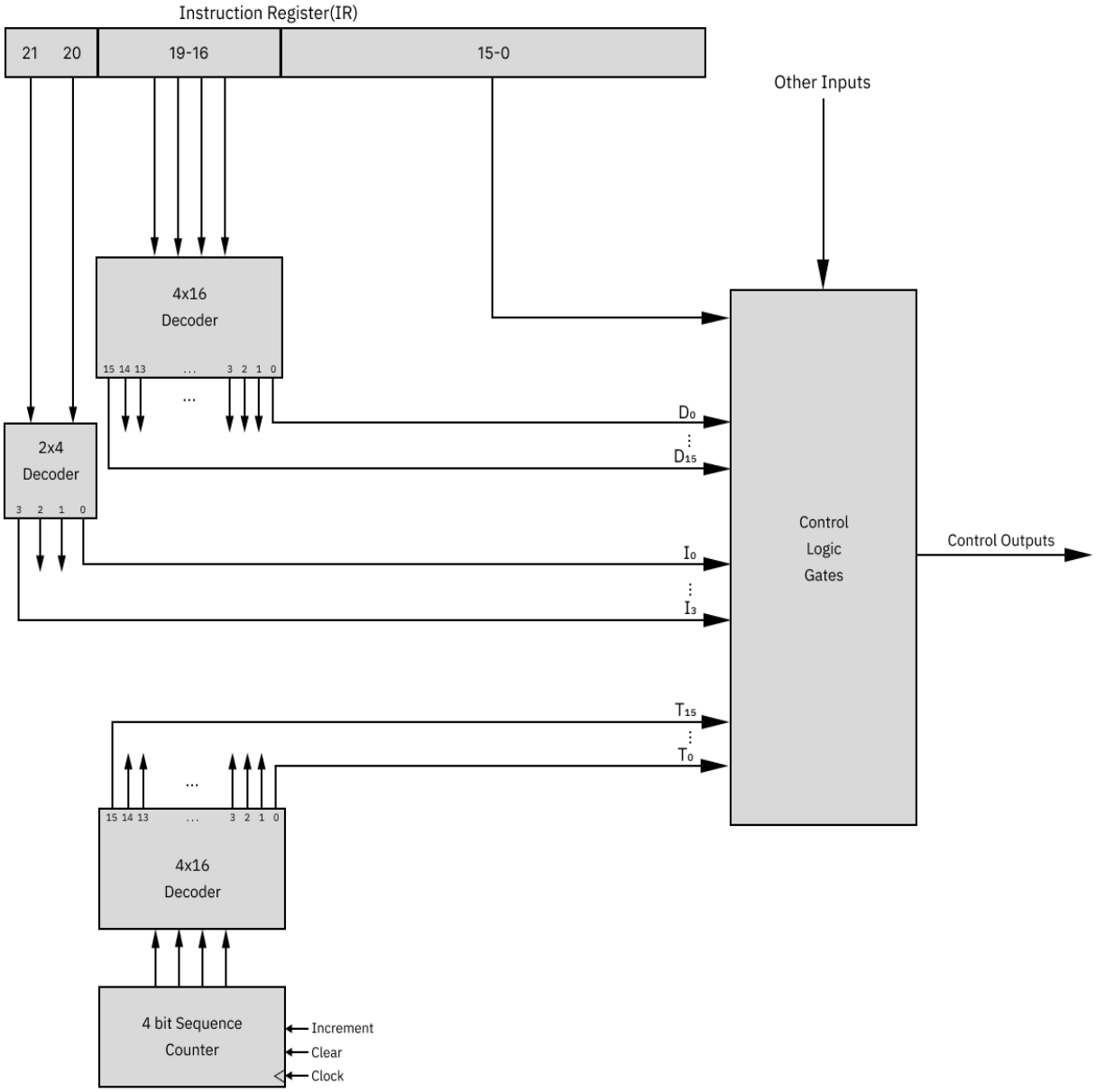

Figure 2: Control Unit

2.6 Instruction Cycle

The instruction cycle consists of several steps, each of which performs a specific function in the execution of the instruction. The major steps in the instruction cycle are:

1. Fetch Cycle: In the fetch cycle, the CPU retrieves the instruction from memory. The instruction is typically stored at the address specified by the program counter (PC). 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 and the PC is then incremented to point to the next instruction in memory.

   Following Register Transfer Language denotes the fetch cycle.

   T0: AR ← PC
   T1: IR←M [AR], PC ← PC+1

2. Decode Cycle: In the decode cycle, the CPU interprets the instruction and determines what operation needs to be performed. This involves identifying the opcode and any operands that are needed to execute the instruction.
   Following is the RTL for the decoding cycle:
   T_2: $D_0 \ldots D_{15}$← Decode IR (16-19)] , I_0...I3← Decode IR (20-21),
   AR ← IR(0-15)

3. Execute Cycle: In the execute cycle, the CPU performs the operation specified by the instruction. This may involve reading or writing data from or to memory, performing arithmetic or logic operations on data, or manipulating the control flow of the program

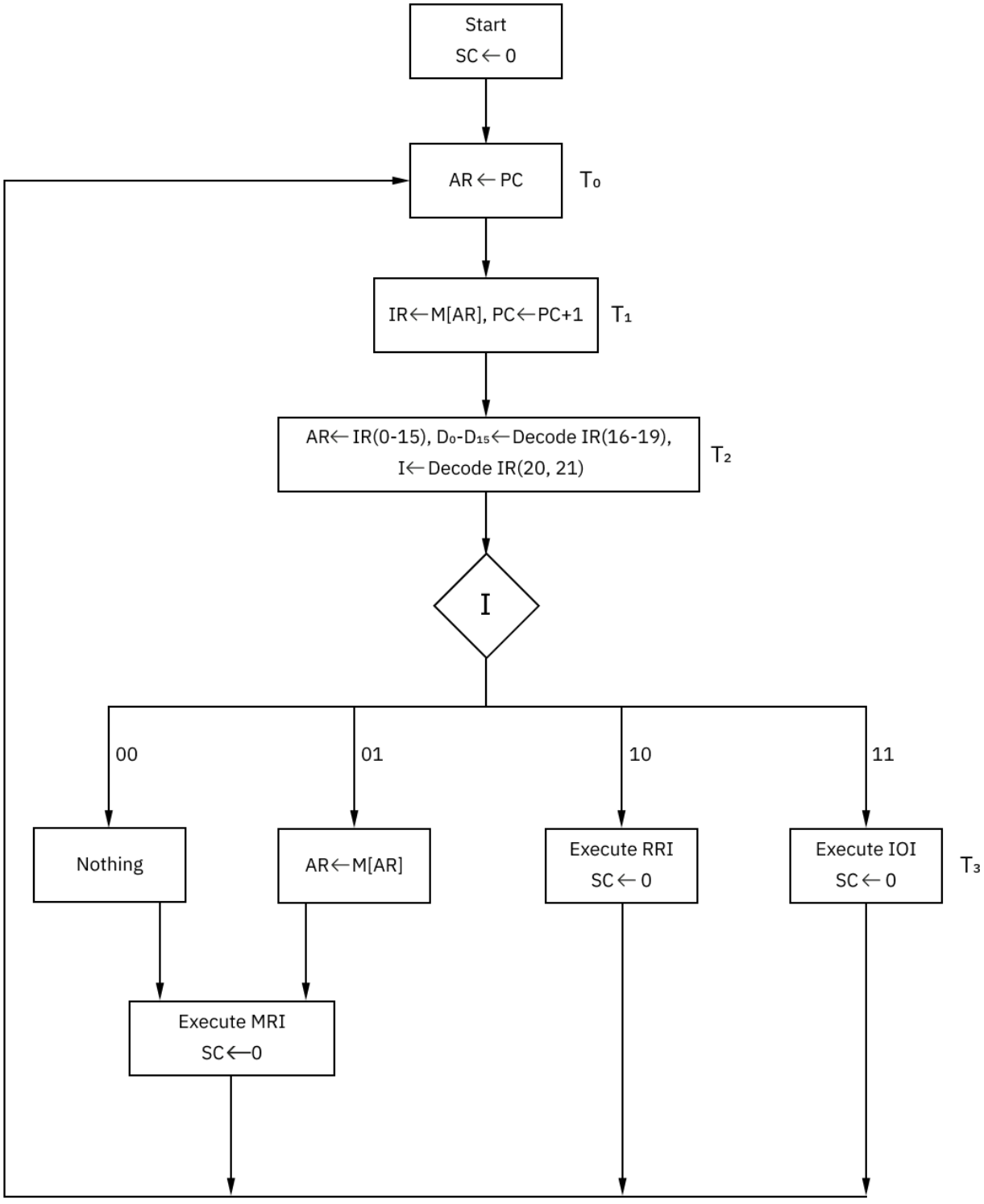

Figure 3: SPAN Instruction Cycle

2.6.1 Memory Reference Instructions

$m=(I_o+I_1)$

Symbol	Decoded Opcode	Operations $I=(I_0+I_1)$
AND	D_0	m D0 T4 : DR ← M[AR] m D0 T5 : AC ← AC ∧ DR, SC ← 0
ADD	D_1	m D_1T_4: DR←M[AR] m D_1T_5: AC←AC+DR, E ←Cout, SC ←0
LDA	D_2	m D_2T_4: DR ← M [AR] m D_2T_5: AC ←DR, SC ←0
STA	D_3	m D_3T_4: M[AR] ← AC, SC ← 0
BUN	D_4	m D_4T_4: PC ← AR, SC ←0
BSA	D_5	m D_5T_4: M [AR] ← PC, AR ← AR + 1 m D_5T_5: PC←AR, SC←0
ISZ	D_6	m D_6T_4: DR← M[AR] m D_6T_5: DR ←DR + 1 m D_6T_6: M[AR] ← DR, if (DR = 0) then (PC ←PC + 1), SC ←0
OR	D_7	m D_7T_4: DR← M[AR] m D_7T_5: AC←AC∨DR, SC ←0
XOR	D_8	m D_8T_4: DR← M[AR] m D_8T_5:AC←AC⊕DR, SC ←0
XCHG	D_9	m D_9T_4:DR←M[AR]

		m D_9T_5:M[AR] ←AC, AC ←DR, SC ←0

Table 11: Memory Reference Instructions

2.6.2 Register Reference Instruction

$r = I_2T_3$

Symbol	Decoded Opcode	Operation
		r: SC ← 0
CLA	D_O	rD_0: AC← 0
CLE	D_1	rD_1 : E ← 0
CMA	D_2	rD_2 : AC ← AC’
CME	D_3	rD_3 : E ← E’
CIR	D_4	rD_4 : AC ← shr AC, AC(15) ← E, E ←AC(0)
CIL	D_5	rD_5 : AC←shl AC, AC(0) ←E, E ←AC(15)
INC	D_6	rD_6 : AC ← AC + 1
SPA	D_7	rD_7 : if (AC(15) = 0) then (PC ← PC+1)
SNA	D_8	rD_8 : if (AC(15) = 1) then (PC ← PC+1)
SZA	D_9	rD_9 : if (AC = 0) then (PC ← PC+1)
SZE	D_{10}	rD_{10} : if (E = 0) then (PC ← PC+1)
HLT	D_{11}	rD_{11} : S ← 0 (S is a start-stop flip-flop)

Table 12: Register Reference operation

2.6.3 Input-Output Instruction

$p=I_3T_3$

Symbol	Decoded Output	Operations
		p: SC← 0
INP	D_0	pD_0 : AC ← INPR, FGI ← 0
OUT	D_1	pD_1 : OUTR ← AC, FGO ← 0
SKI	D_2	pD_2 : if(FGI = 1) then (PC ← PC + 1)
SKO	D_3	pD_3 : if(FGO = 1) then (PC ← PC + 1)
ION	D_4	pD_4 : IEN ← 1
IOF	D_5	pD_5 : IEN ← 0

Table 13: Input Output operation

2.7 I/O and Interrupt

When a key on the keyboard is pressed, an 8-bit alphanumeric code is transferred into the INPR via the interface. The INPR and OUTR communicate serially with the interfaces, but they communicate in parallel with the accumulator. Initially, the FGI value is set to 0. When data is shifted into the INPR, the FGI value is set to 1. Once the INPR value is transferred to the accumulator, the FGI is reset to 0. Similarly, the FGO value is set to 1. When the accumulator's contents are moved to the OUTR, the FGO value is reset to 0. After the OUTR value is sent to the output devices (like a printer), the FGO value is set to 1 again. While FGI is set to 1, no new input can be transferred from the input devices to the INPR through the interfaces. When FGO is set to 1, data from the input device cannot be transferred to the INPR, and when FGO is set to 0, data cannot be moved from the accumulator to the OUTR.

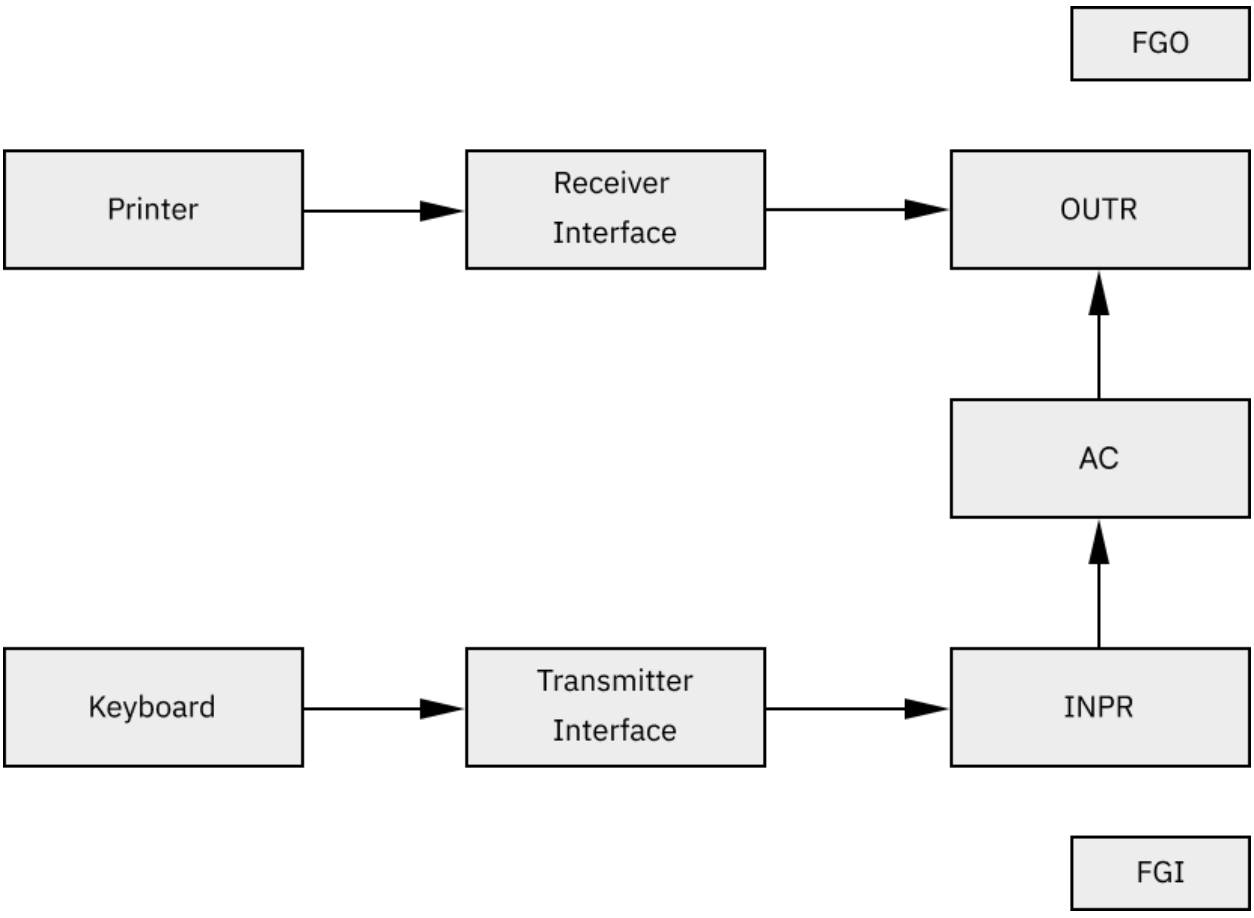

Figure 4: I/O and Interrupt

2.7.1 Interrupt Cycle

Interrupts occur in open communication when data needs to be transferred, either by fetching input into the INPR or sending output from the OUTR. The I/O interface monitors the devices instead of the CPU. When the interface detects that an I/O device is ready for data transfer, it generates an interrupt request to the CPU. Upon detecting an interrupt, the CPU temporarily halts its current task, branches to a service routine to handle the transfer, and then resumes its previous task.

The interrupt handling process is illustrated in flowchart below. An interrupt flip-flop R is used in the computer. When R=0, the computer executes its instruction cycle. During this cycle, the control checks the IEN. If IEN is 0, the program does not use interrupts, and control proceeds with the next instruction cycle. If IEN is 1, control checks the flag bits. If both flags are 0, indicating that neither the input nor output registers are ready for data transfer, the next instruction cycle continues. If either flag is set to 1 and IEN is also 1, flip-flop R is set to 1. At the end of the execute phase, control checks R, and if R=1, it enters an interrupt cycle instead of the next instruction cycle.

The Register Transfer Language for RTL is

$T0' T1' T2' (IEN)(FGI + FGO): R \leftarrow 1$

$RT0 : AR \leftarrow 0, TR \leftarrow PC$

$RT1 : M[AR] \leftarrow TR, PC \leftarrow 0$

$RT2 : PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$

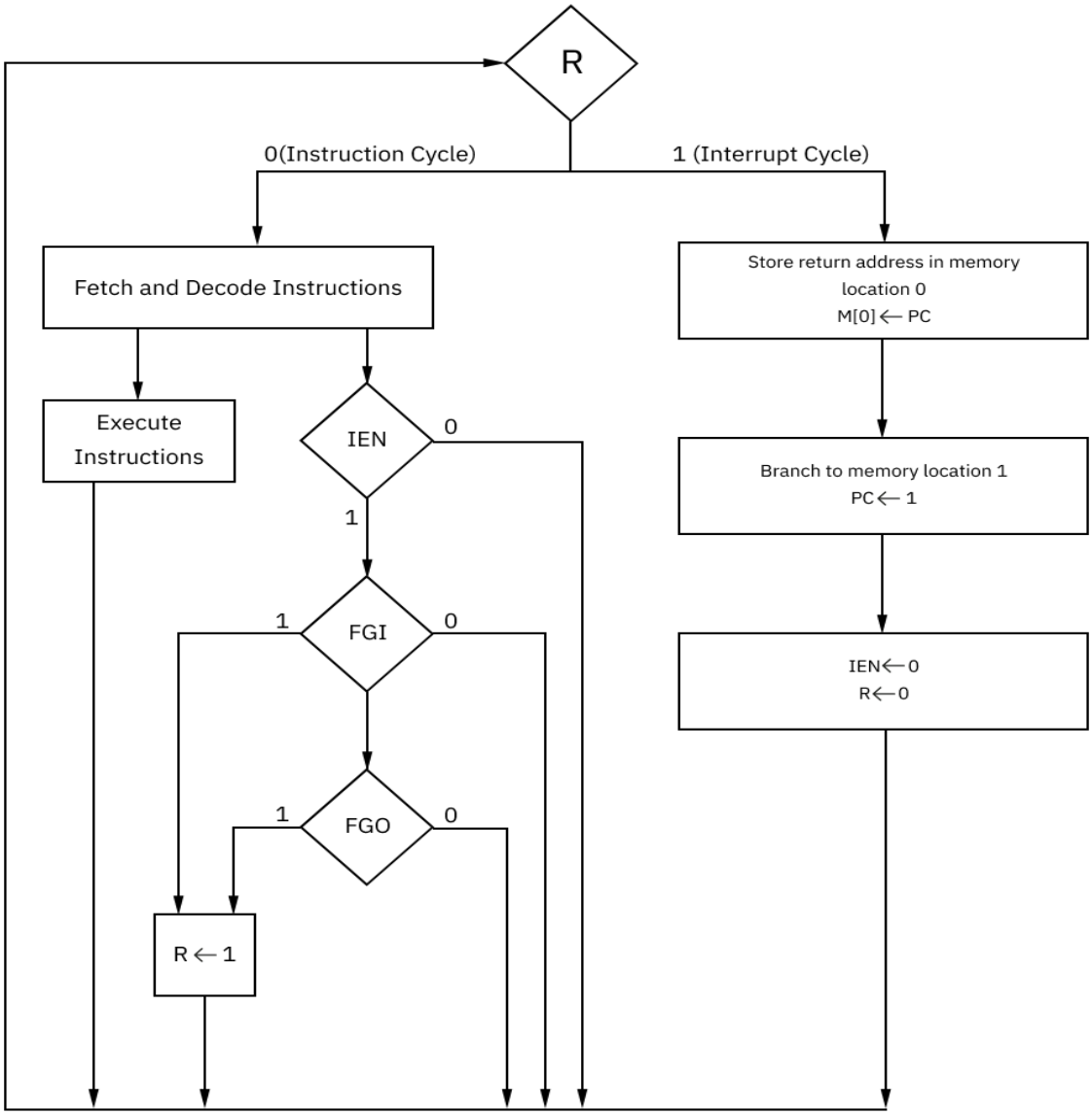

Figure 5: Interrupt Cycle

2.8 Flowchart of Operation

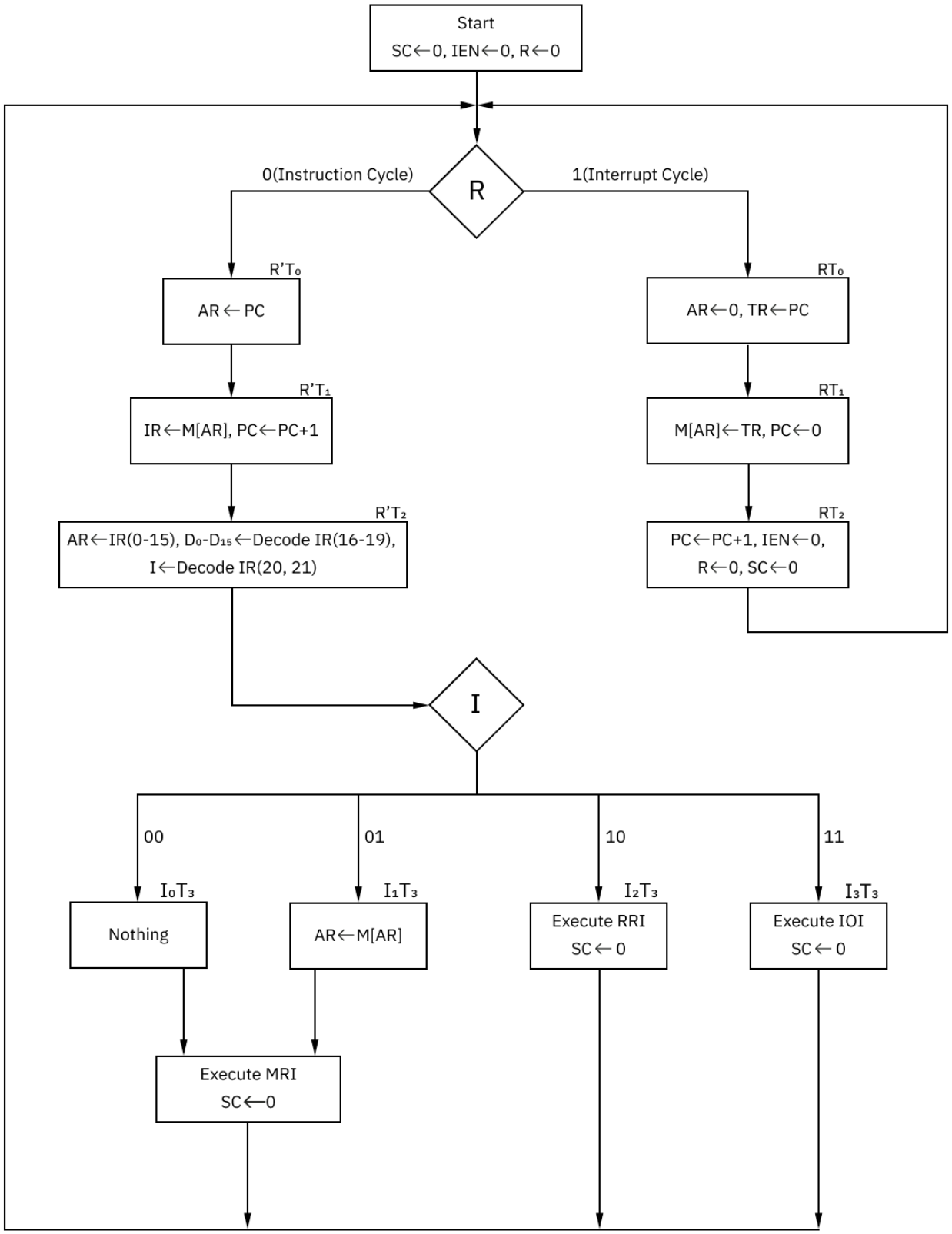

Figure 6: Flowchart of Operation

Chapter 3: Design of Individual Components

This chapter shows the circuits of the individual registers and memory used in the SPAN computer. For simplification, we use the following terminology:

$I_0+I_1 = m$

$I_2T_3 = r$

$I_3T_3 = p$

3.1 Control Registers and Memory

3.1.1 AR

LD:

$R'T_0$: AR ← PC

$R'T_1$: AR ← IR(0-15)

$R'I_1T_3$: AR ← M[AR]

INR:

mD_5T_4: AR ← AR + 1

CLR:

RT_0: AR ← 0

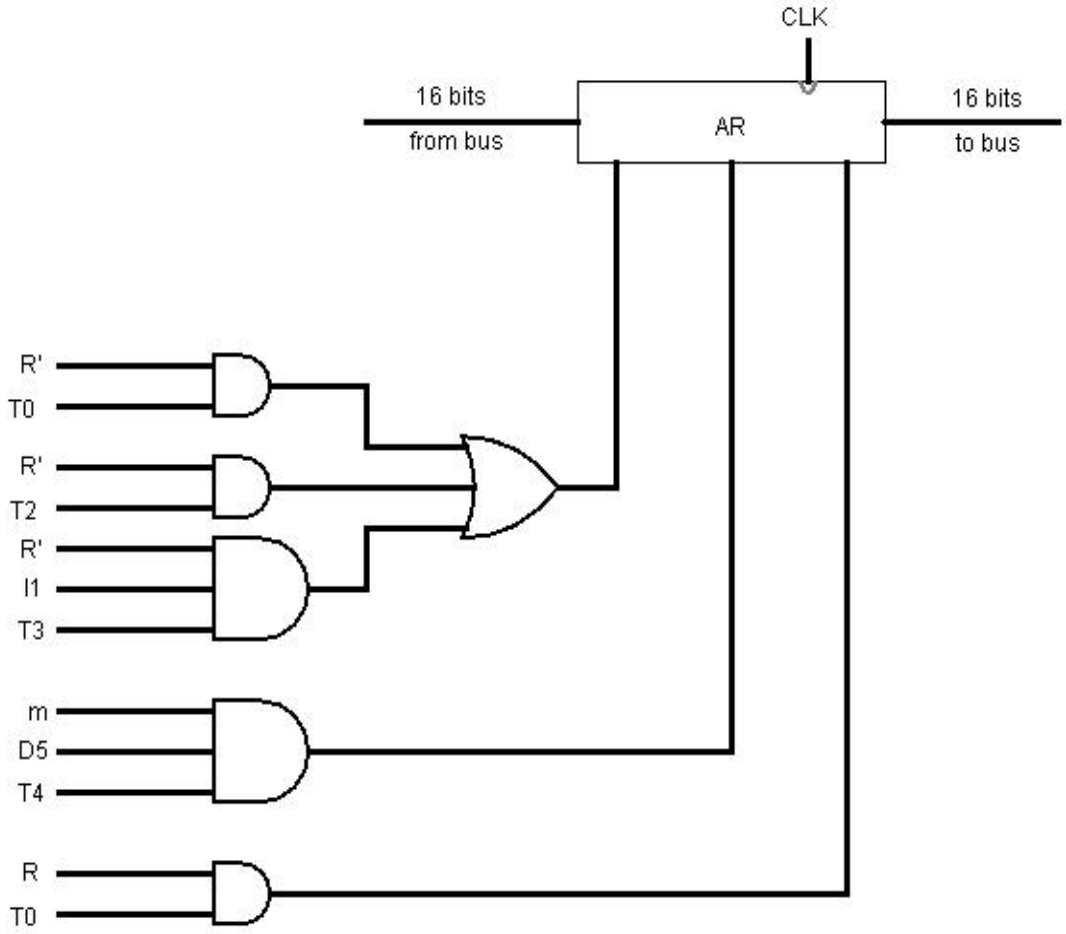

Figure 7: AR Control Signal

3.1.2 DR

LD:

mD_0T_4: DR ← M[AR]

mD_1T_4: DR ← M[AR]

mD_2T_4: DR ← M[AR]

mD_6T_4: DR ← M[AR]

mD_7T_4: DR ← M[AR]

mD_8T_4: DR ← M[AR]

CLR:

mD_6T_5: DR ← DR + 1

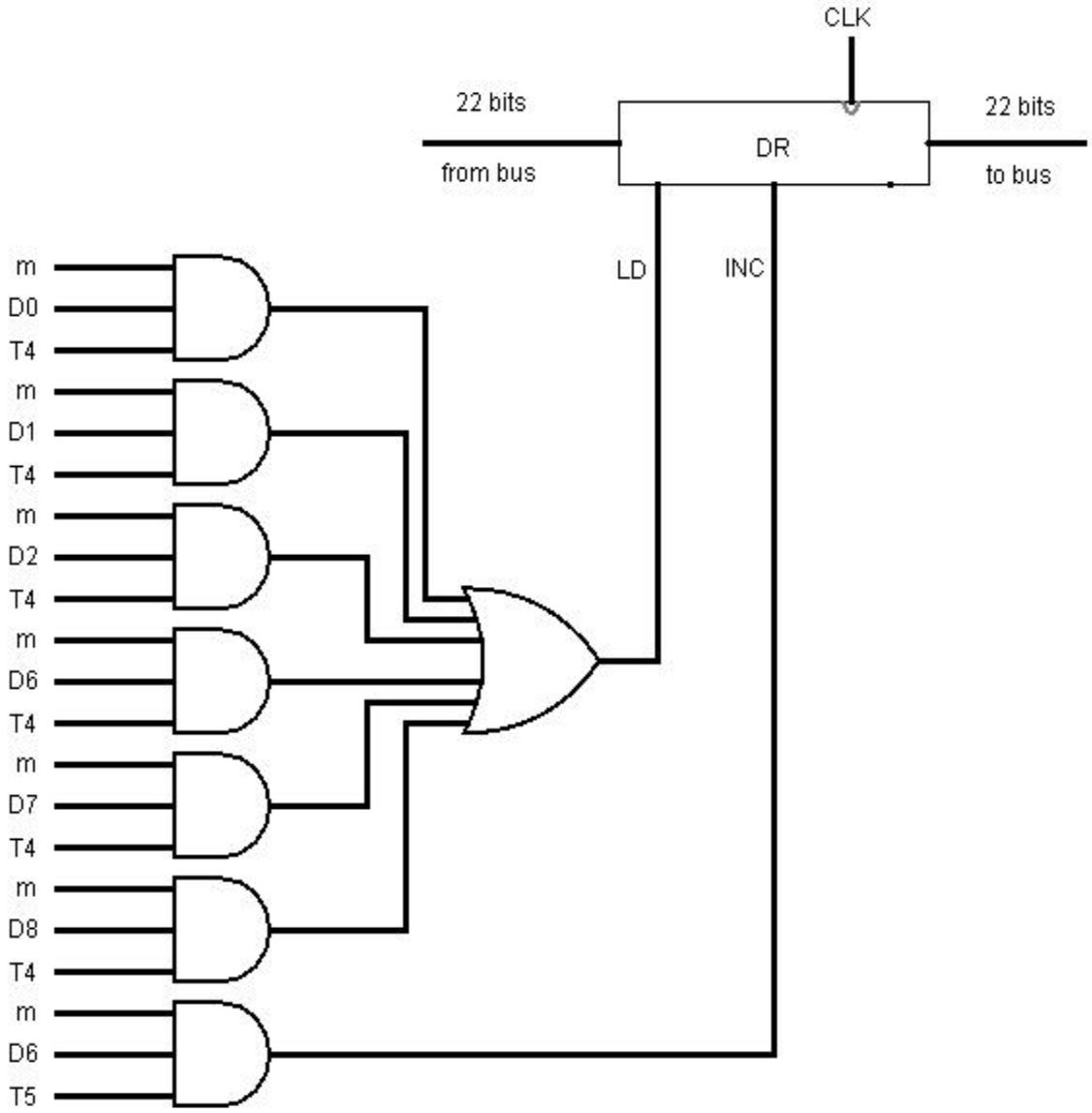

Figure 8: DR Control Signal

3.1.3 AC

LD:

mD_0T_5: AC←AC^DR

mD_1T_5:AC←AC+DR

mD_2T_5:AC←DR

mD_7T_5:AC←AC ∨ DR

mD_8T_5: AC ← AC ⊕ DR

mD_9T_5: AC ← DR

$rD_{2:}$:AC ← AC'

rD_4:AC ← shrAC, AC(15) ← E

rD_5: AC ← shlAC, AC(0) ← E

pD_0:AC ← INPR

INR:

rD_6:AC ← AC + 1

CLR:

rD_0 : AC ← 0

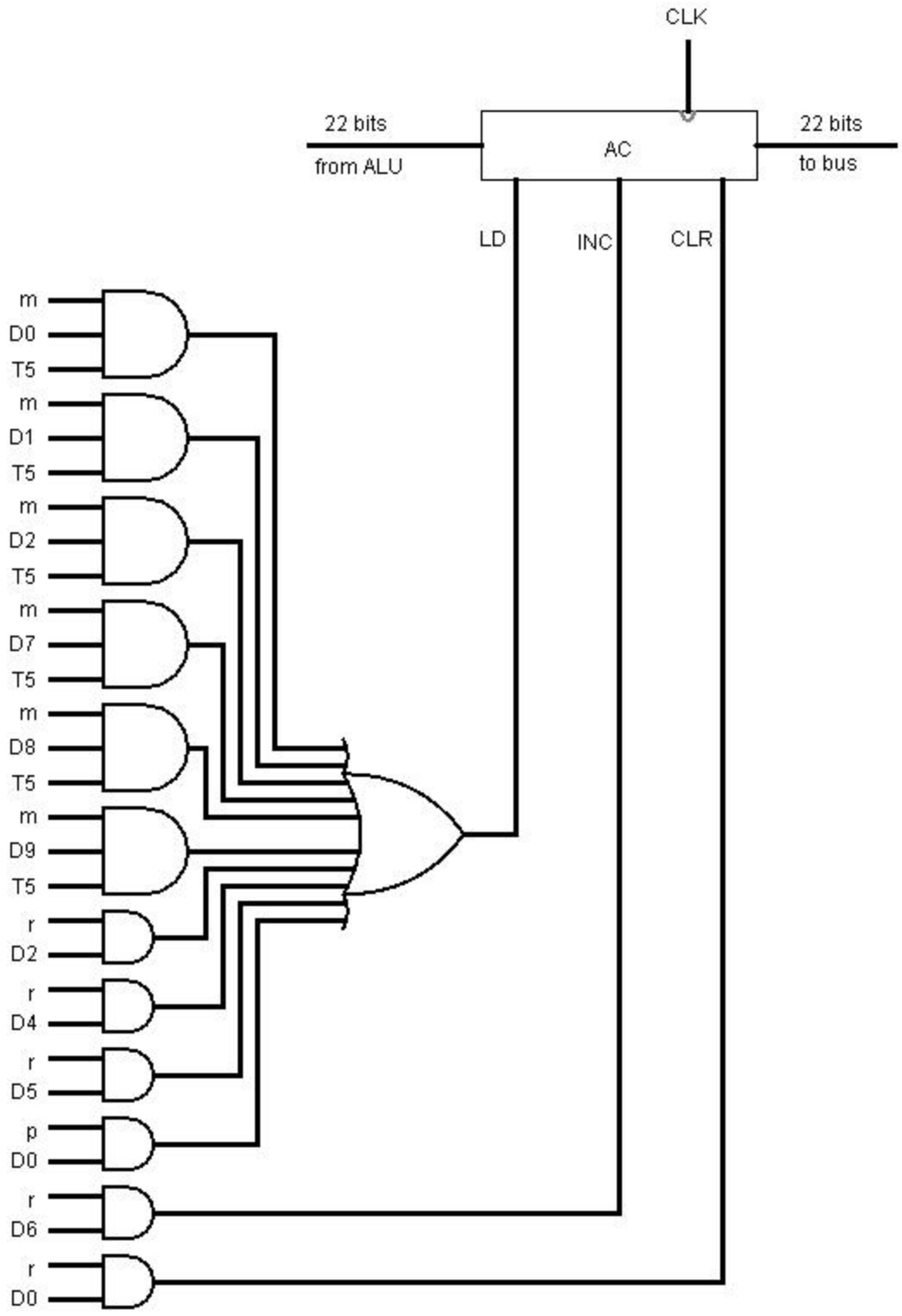

Figure 9: AC Control Signal

3.1.4 IR

LD:

$R'T_1$:IR ← M[AR]

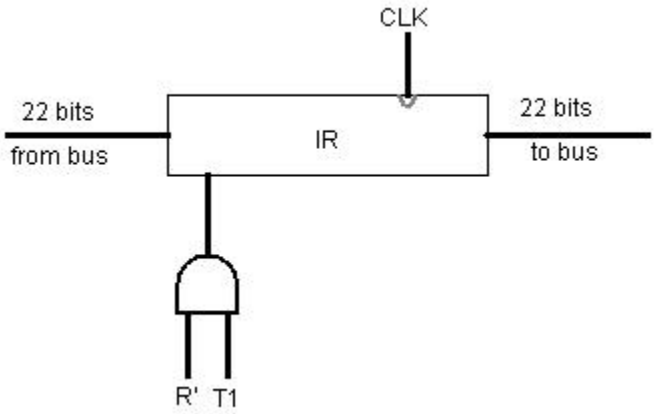

Figure 10: LD Control Signal

3.1.5 TR

LD:

RT_0:TR ← PC

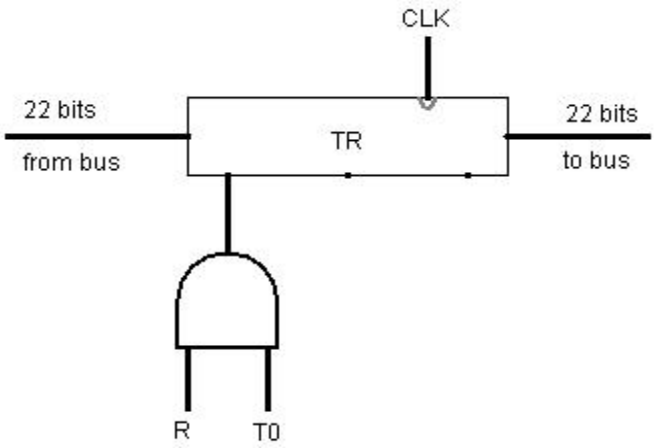

Figure 11: TR Control Signal

3.1.6 OUTR

LD:

pD_1:OUTR ← AC

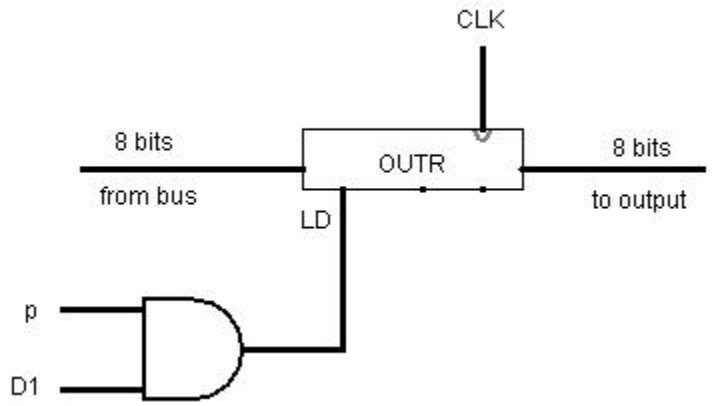

Figure 12: OUTR Control Signal

3.1.7 PC

LD:

mD_4T_4:PC ← AR

mD_5T_5:PC ← AR

INR:

$R'T_1$:PC ← PC + 1

RT_2 : PC ← PC+1

mD_6T_6:if (DR=0) then PC ← PC+1

rD_7:: if (AC(21)=0) then PC ← PC+1

rD_8:if (AC(21)=1) then PC ← PC+1

rD_9:if (AC=0) then PC ← PC+1

rD_{10}:if (E=0) then PC ← PC+1

pD_2:if (FGI=1) then PC ← PC+1

pD_3:if (FGO=1) then PC ← PC+1

CLR:

RT_1: PC ← 0

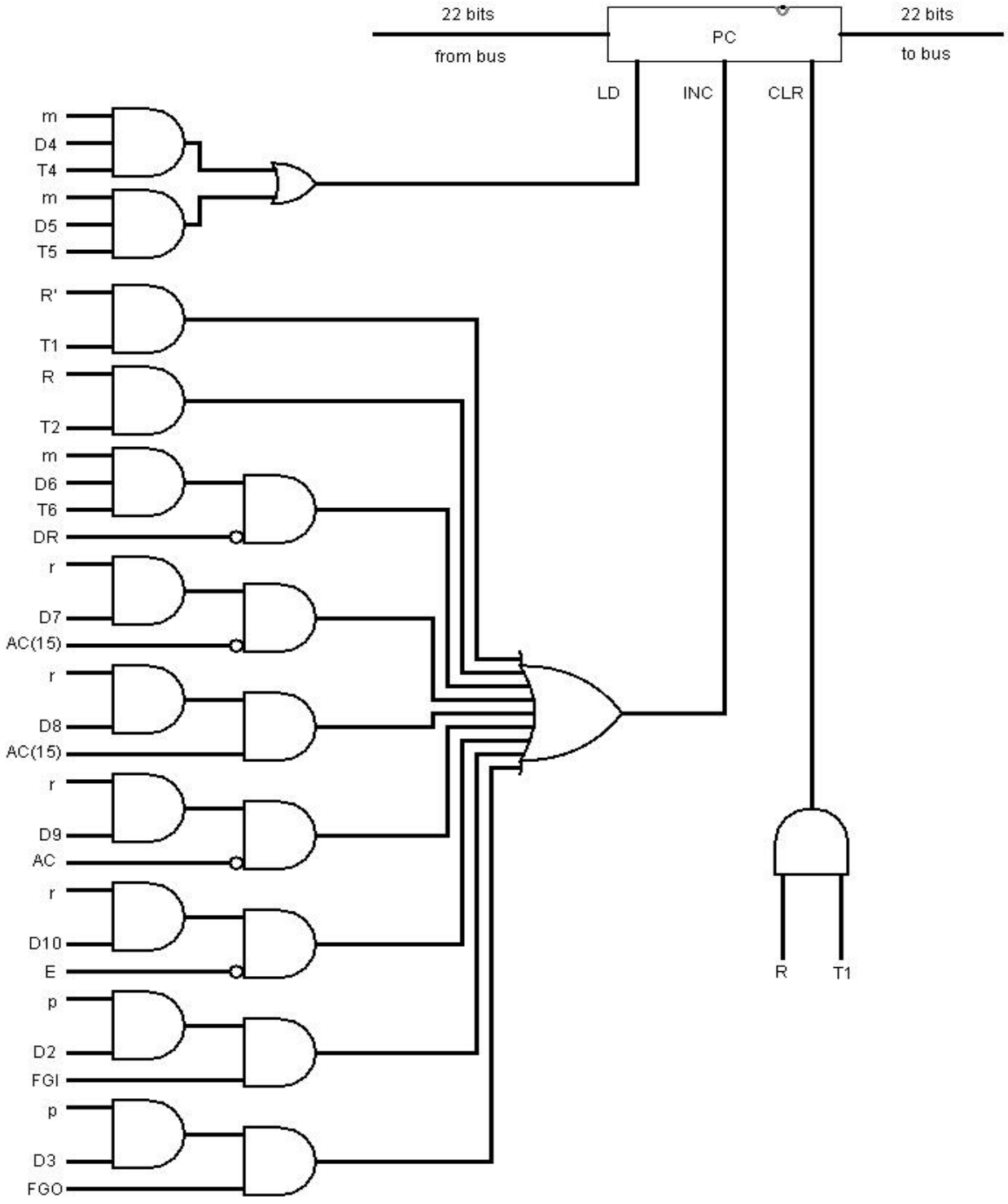

Figure 13: PC Control Signal

3.1.8 Memory Write

LD:

mD_3T_4:M[AR]← AC

mD_6T_6:M[AR] ← DR

mD_9T_5:M[AR] ← AC

RT_1 : M[AR] ← TR

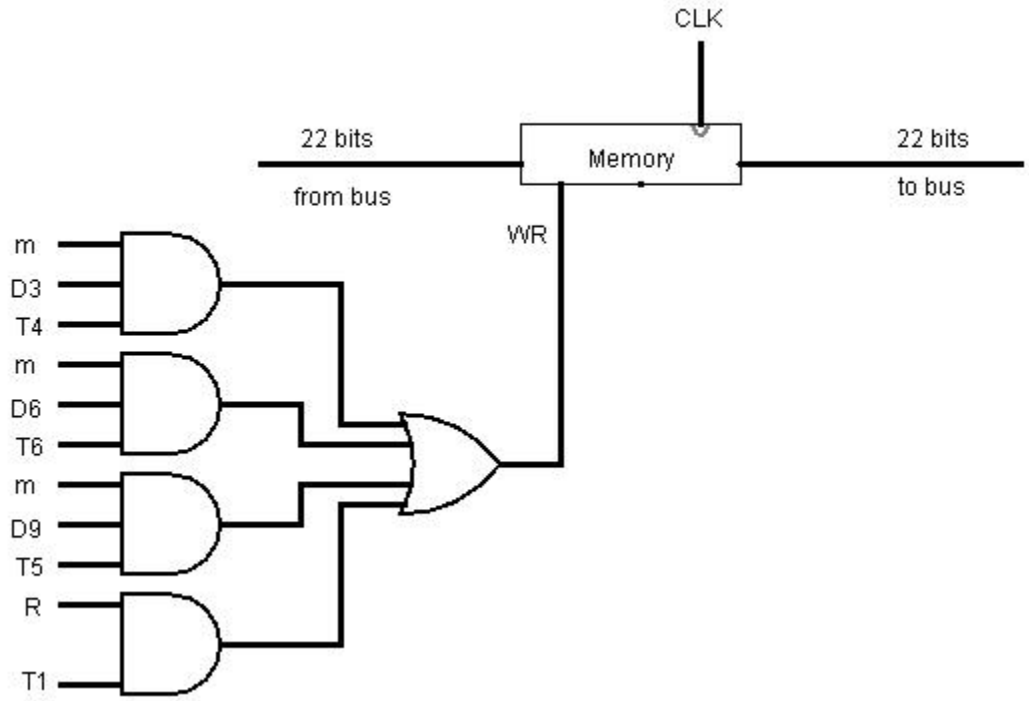

Figure 14: Memory Write Control Signal

3.1.9 SC

CLR:

mD_0T_5:SC←0

mD_1T_5:SC←0

mD_2T_5:SC←0

mD_3T_5:SC←0

mD_4T_5:SC←0

mD_5T_5:SC←0

mD_6T_5:SC←0

mD_7T_5:SC←0

mD_8T_5:SC←0

mD_9T_5:SC←0

r:SC←0

p:SC←0

RT_2:SC←0

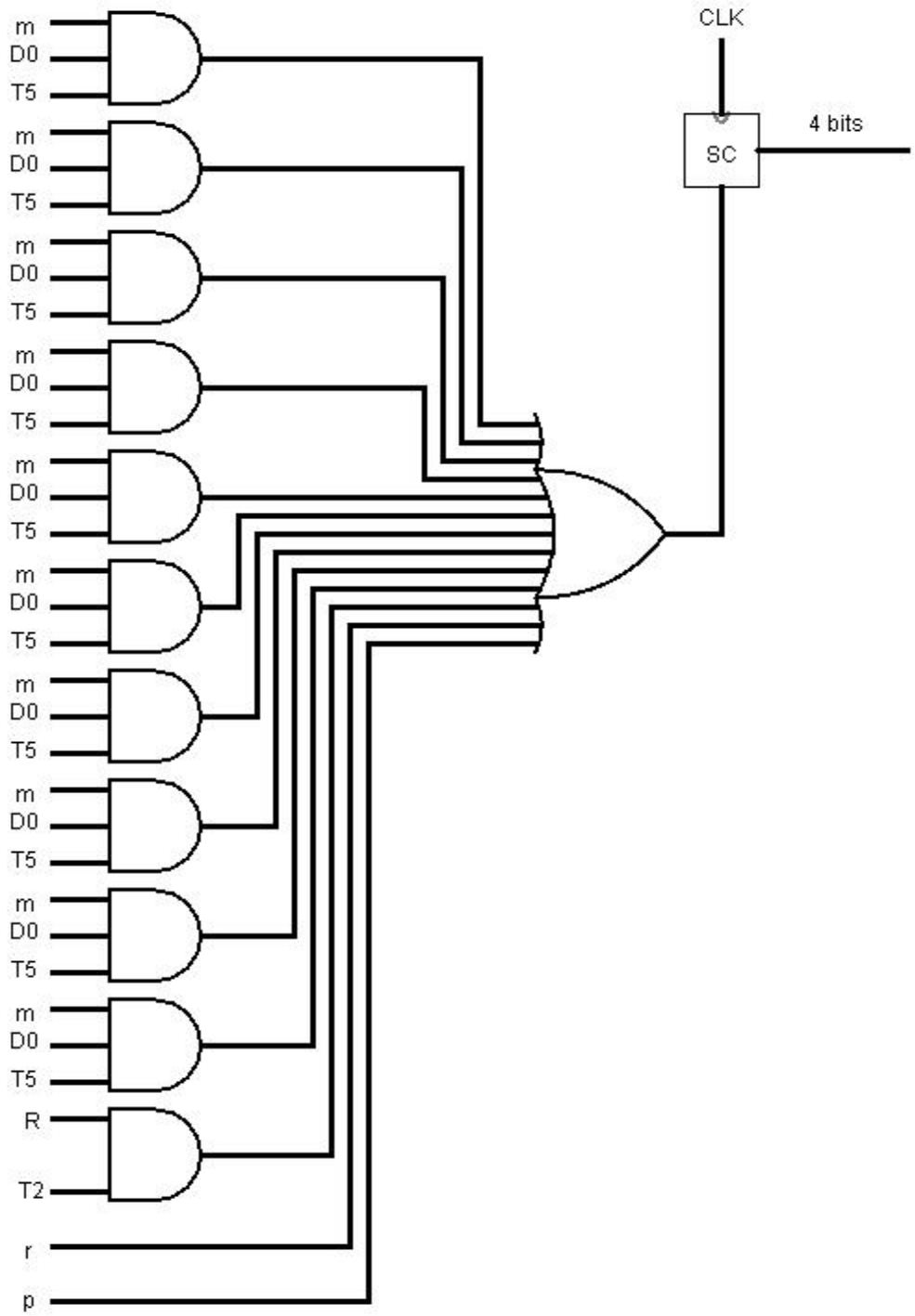

Figure 15: SC Control Signal

3.2 Control of Flip Flops

3.2.1 End-around Carry (E) Flip Flop

mD_1T_5:E←C_{out}

rD_1:E←0

rD_3:E←E’

rD_4:E←AC(0)

rD_5:E←AC(21)

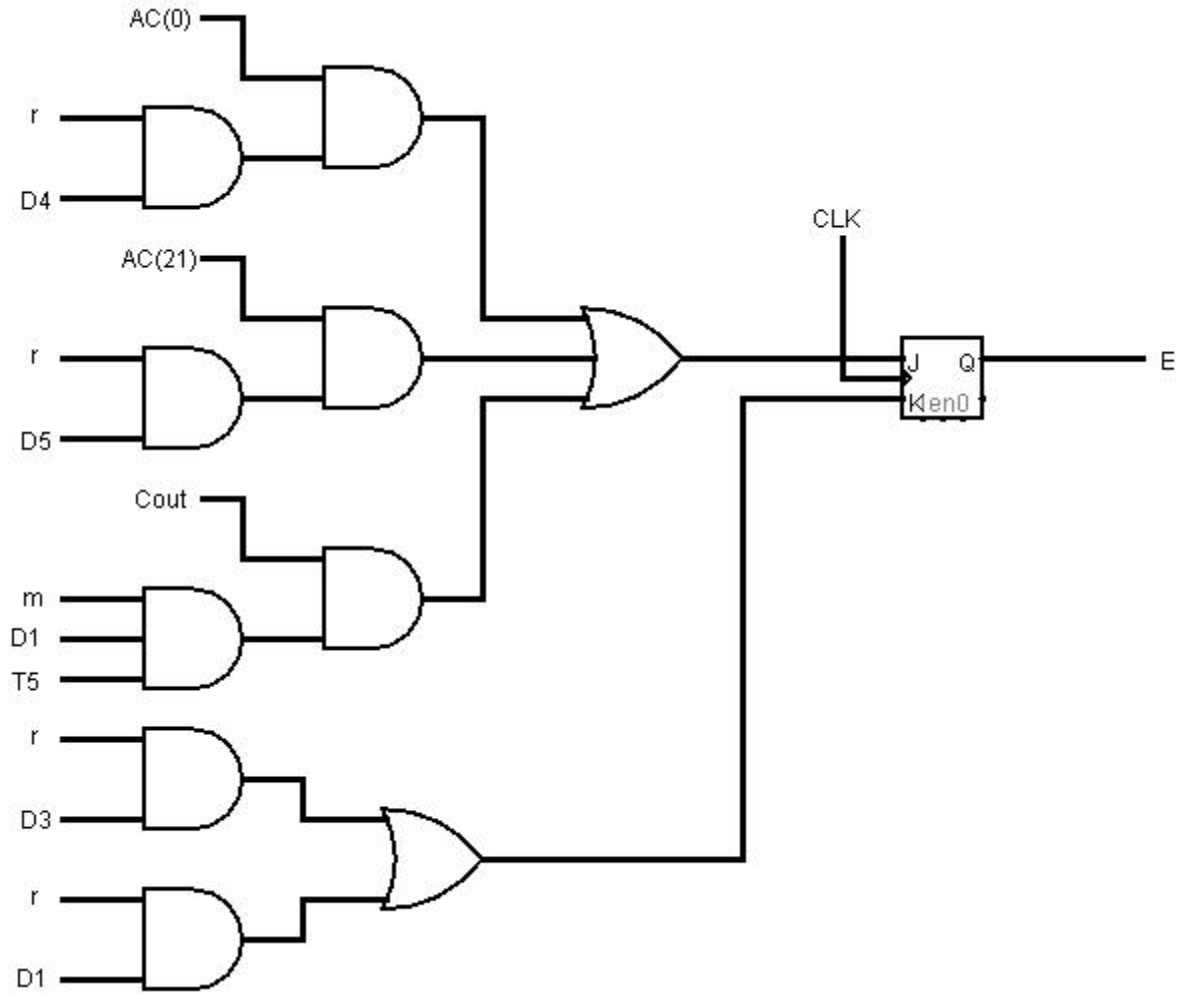

Figure 16: E flipflop control signal

3.2.2 R Flip Flop

$T0' T1' T2' (IEN)(FGI + FGO): R \leftarrow 1$

$RT_2: R \leftarrow 0$

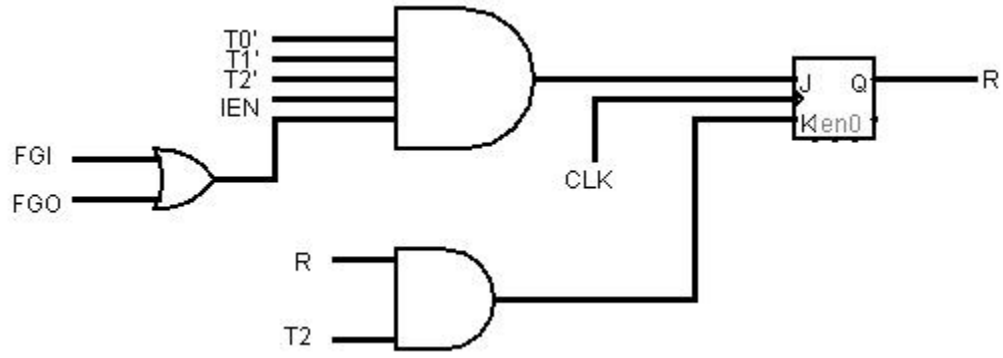

Figure 17: R Flipflop Control Signal

3.2.3 IEN

$RT_2: IEN \leftarrow 0$

$pD_5: IEN \leftarrow 0$

$pD_4: IEN \leftarrow 1$

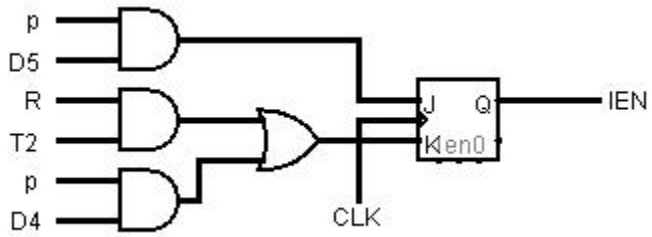

Figure 18: IEN Control Signal

3.2.3 Start-Stop Flip-flop (S)

rD_{11}: S ← 0

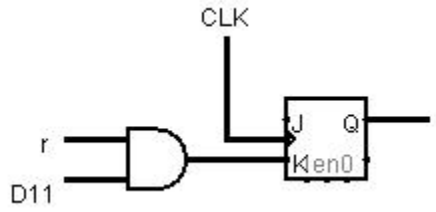

Figure 19: S Control Signal

3.2.4 FGI

pD_1: FGI← 0

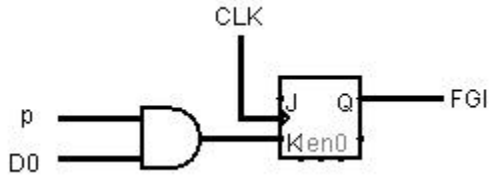

Figure 20: FGI Control Signal

3.2.5 FGO

pD_1: FGO ← 0

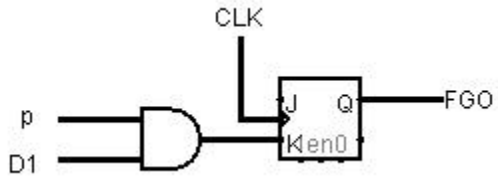

Figure 21: OUTR Control Signal

3.3 Control of Common Bus

The 22-bit common bus shown in fig. 21 is controlled by the selection inputs S2, S1, and S0. The decimal number shown with each bus input specifies the equivalent binary number that must be applied to the selection inputs in order to select the corresponding register. Table. 14 specifies the binary numbers for S_2, S_1, S_0 that select each register. Each binary number is associated with a Boolean variable x1 through x7, corresponding to the gate structure that must be active in order to select the register or memory for the bus.

Inputs							**Outputs**			**Register for selected bus**
X_1	**X_2**	**X_3**	**X_4**	**X_5**	**X_6**	**X_7**	**S_2**	**S_1**	**S_0**	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	o	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 14: Control of Common Bus

The placement of the encoder at the inputs of the bus selection logic is shown in fig. 22 . The Boolean functions for the encoder are:

$So = X_1 + X_3 + X_5 + X_7$

$S_1 = X_2 + X_3 + X_6 + X_7$

$S_3 = X_4 + X_5 + X_6 + X_7$

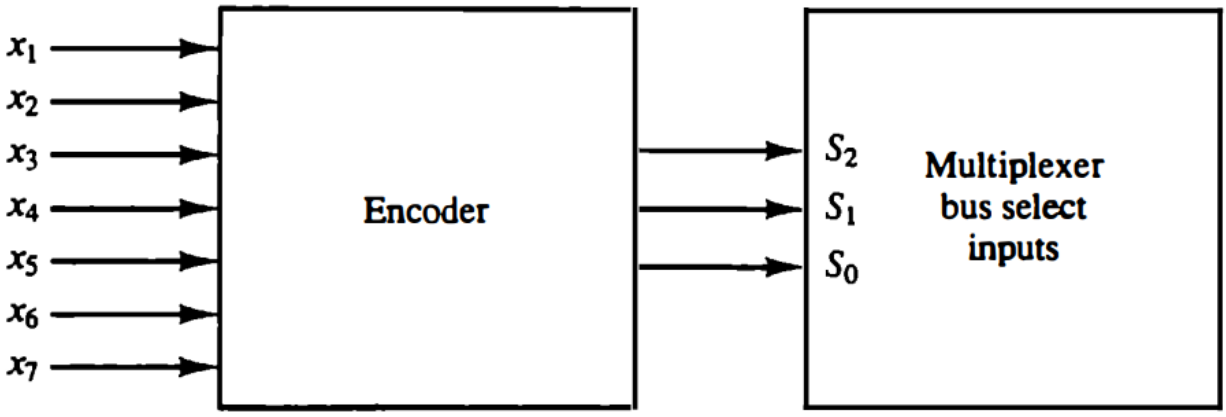

Figure 22: Encoder

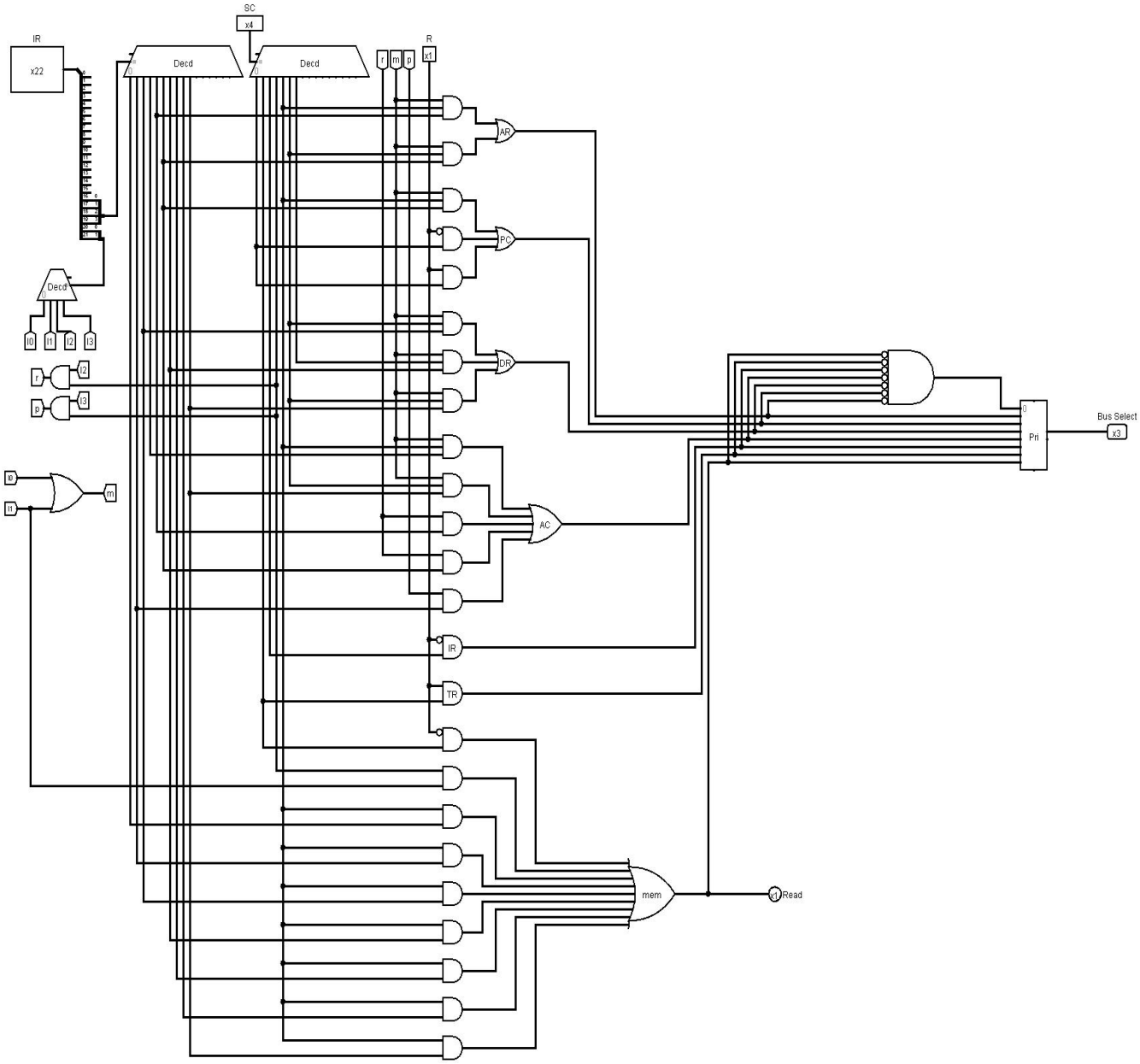

Figure 23: BUS Control Signal

3.4 ALU (Adder and Logic Circuit)

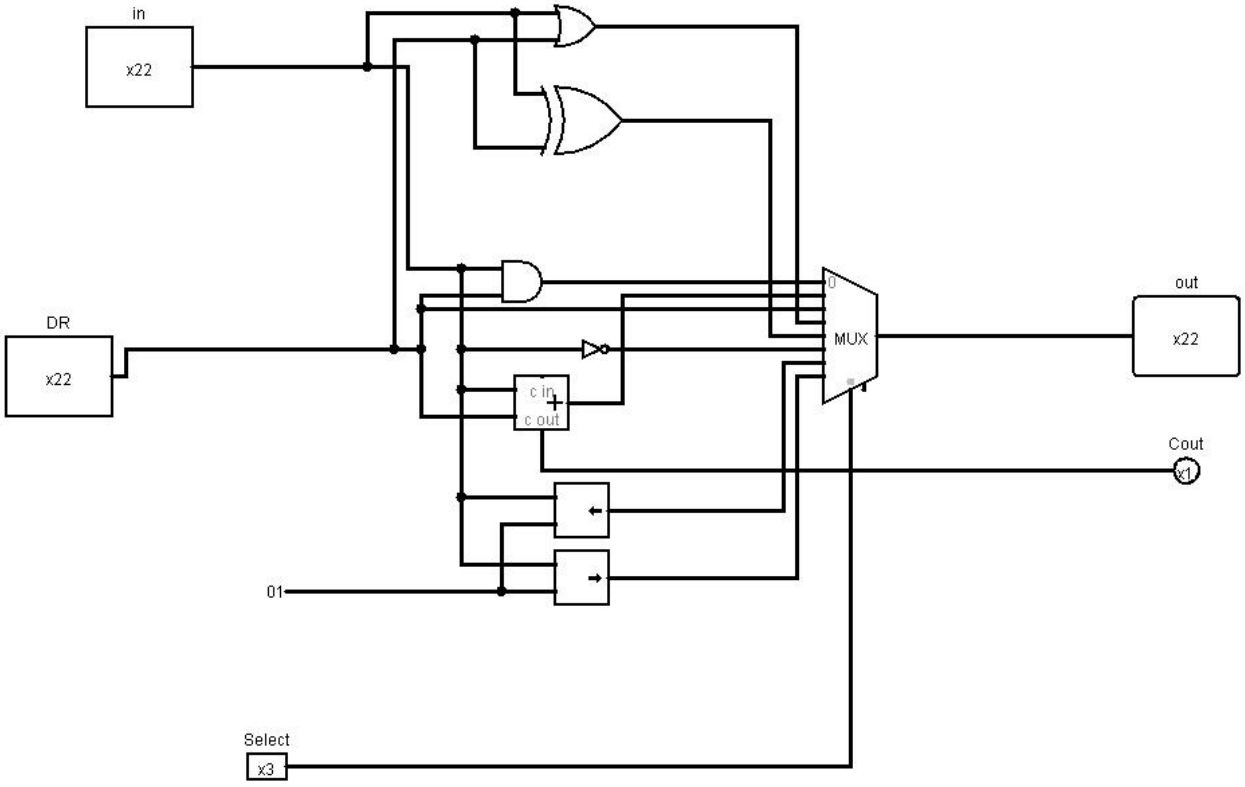

Figure 24: ALU Control Signal

Chapter 4: Conclusion

Designing an 18-bit CPU was a complex yet essential process, crucial for integrating all the concepts of Computer System Architecture. Completing this task required a thorough understanding of computer organization and architecture. However, by gradually working through the process, assembling all the signal and control lines, and aligning the proper logic gates, we successfully completed the design of the computer, which we named SPAN.