Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

A Mini Project Report

on

"NPR 21-bit Computer Design"

[Code No: COMP 315]

Submitted By:

Nischal Baral (06)

Prashant Manandhar (30)

Reewaj Khanal (61)

Submitted To:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date:2024/01/14

ACKNOWLEDGEMENT

We wish to extend our heartfelt thanks to Mr. Dhiraj Shrestha, our mentor, for his invaluable guidance, direction, and unwavering support throughout the development of our computer design, rooted in the foundational principles of "Basic Computer." Our gratitude also extends to all those who facilitated our successful completion of this endeavor. Through this project, we deepened our comprehension of essential components like computer memory, registers, and flip-flops. Additionally, we appreciate the university and the Department of Computer Science and Engineering (DOCSE) for granting us the opportunity to undertake this project.

Abstract

The NPR 21-bit computer is a specialized architecture featuring a 21-bit word size, segmented into a 4-bit opcode for operation specification, a 15-bit address for memory referencing, and a 2-bit field for versatile addressing modes. This design accommodates four distinct addressing modes, enabling operations like immediate, direct, indirect, and indexed addressing. The computer incorporates essential registers such as the Program Counter, Instruction Register, Accumulator, Memory Address Register, and Memory Buffer Register. With an instruction set including operations like load, store, add, and subtract, the NPR 21-bit system can execute instructions by fetching them from memory, decoding their opcode and addressing mode, and subsequently performing the specified operations, thereby facilitating efficient computation within its 32,768 memory locations.

Keywords:

- CPU Central Processing Unit
- RAM Random Access Memory
- AC Accumulator
- PC Program Counter
- TR Temporary Register
- AR Address Register
- DR Data Register
- IR Instruction Register
- OUTR Output Register
- INPR Input Register
- SC Sequence Counter
- MRI Memory Reference Instructions
- RRI Register Reference Instructions
- IOI Input Output Instructions
- SEQ Skip if Equal
- LDA Load Accumulator
- STA Store Accumulator
- BUN Branch Unconditionally
- BSA Branch and Save Return Address
- XCHG Exchange
- ISZ Increment and skip if zero.
- DSZ Decrement and skip if zero.
- CLA Clear Accumulator
- CMA Complement Accumulator
- CLE Clear E
- CME Complement E

- CIR Circular shift right
- CIL Circular shift Left.
- INC Increment
- DEC Decrement
- SPA Skip if Positive in Accumulator
- SNA Skip if Negative in Accumulator
- SZA Skip if Zero in Accumulator
- SZE Skip if Zero in E
- HLT Halt
- INP Input AC
- OUT Output AC
- SIE Skip if Input flag is enabled.
- SID Skip if Input flag is disabled.
- SOE Skip if Output flag is enabled.
- SOD Skip if Output flag is disabled.
- ION Interrupt enable ON.
- IOF Interrupt enable OFF

Table of Contents

List of Figures

Chapter 1 Introduction

A basic computer serves as the foundational building block of modern computing systems, encapsulating the essential elements required for data processing, storage, and retrieval. At its core, a basic computer comprises components such as a central processing unit (CPU), which acts as the brain, executing instructions and coordinating tasks. This CPU communicates with memory modules, where data and instructions are temporarily stored for rapid access during operations. Additionally, input and output devices facilitate interaction with users, allowing data entry and displaying results. The architecture of a basic computer often encompasses a system bus, connecting various components and enabling seamless data transfer. While rudimentary in design compared to advanced computing systems, a basic computer exemplifies the fundamental principles and functionalities that underpin more sophisticated technological innovations in the digital age.

1.1 Stored Program Concept

The stored-program concept, pioneered by mathematician and computer scientist John von Neumann, revolutionized modern computing by introducing a unified approach to storing both instructions and data within a computer's memory. This pivotal innovation allowed for dynamic, programmable machines where the central processing unit (CPU) fetches, decodes, and executes instructions stored alongside data in memory, enabling unparalleled flexibility and adaptability. Prioritizing software over fixed, hardwired programs, the stored-program architecture laid the foundation for contemporary software development, operating systems, and high-level programming languages, catalyzing transformative advancements in computational capabilities and technological innovation.

1.2 Von Neumann Architecture

The von Neumann architecture, conceptualized by the renowned mathematician and physicist John von Neumann, stands as a pivotal milestone in the evolution of computing systems. This groundbreaking design integrates key components such as the central processing unit (CPU), memory, and input/output (I/O) devices, all interconnected via a unified system bus. A hallmark of this architecture is the stored-program concept, wherein both instructions and data reside within the computer's memory. This innovation allows the CPU to fetch, decode, and execute instructions sequentially, providing a flexible and programmable computing environment that has been instrumental in shaping modern software ecosystems and technological advancements.

However, while the von Neumann architecture introduced a revolutionary approach to computing with its programmable and sequential processing capabilities, it also brought about certain inherent limitations. One notable constraint is the potential bottleneck in data throughput due to its sequential nature, which has spurred alternative architectural approaches to address evolving computational demands. Nevertheless, despite its limitations, the von Neumann architecture remains foundational, laying the groundwork for contemporary software development, operating systems, and the pervasive digital innovations that continue to redefine our world.

Chapter 2 DESIGN CONSIDERATIONS

The "NPR 21-bit Computer" is a sophisticated computational framework meticulously designed to process a foundational instruction set akin to a basic computer, while also incorporating a range of additional directives. Operating with a 21-bit word length and fortified with a 32K memory capacity, this system features a 15-bit address line, enhancing memory accessibility and facilitating efficient data retrieval and storage. The architectural blueprint showcases four registers, each boasting a 21-bit capacity, supplemented by two pivotal 15-bit address-related registers: AR and PC, alongside an 8-bit input/output register and a 4-bit Sequence Counter. Essential to its operational prowess are two decoders: a 4x16 opcode decoder, pivotal for facilitating instruction interpretation, and a dedicated 4x16 decoder orchestrating timing signals.

In terms of instruction sets, the NPR 21-bit Computer seamlessly integrates a diverse array of Memory Reference Instructions (MRIs). These MRIs encompass immediate, direct, and indirect addressing modes, providing precise control over memory locations as specified within the directives. This comprehensive system encapsulates 15 MRIs, 8 Input/Output Instructions (IOIs), and 13 Register Reference Instructions (RRIs), ensuring unparalleled versatility and adaptability, thereby enabling the execution of a broad spectrum of computational tasks within its meticulously crafted and advanced architectural framework.

2.1 Registers

The NPR Computer uses the following nine registers for their respective purposes:

Table 2.1-1 List of Registers in 'NPR Computer'

Register Name	Register Symbol	Number of bits	Function
Data Register	DR	21	Holds Memory Operand
Address Register	AR	15	Holds Address of memory
Temporary Register	TR	21	Holds Temporary Data
Instruction Register	IR	21	Holds Instruction Code
Accumulator	AC	21	Processor Register
Program Counter	PC	15	Holds Address of Instruction
Input Register	INPR	8	Holds Input Character
Output Register	OUTR	8	Holds Output Character
Sequence Counter	SC	4	

2.2 Flip-Flop

There are Eight Flip-Flops in our basic computer:

1. I_0 flip flop.
2. I_1 flip flop.
3. S flip flop.
4. E flip flop.

5. R flip flop.
6. IEN flip flop.
7. FGI flip flop.
8. FGO flip flop.

2.3 Common Bus System

The 21-bit data bus used in the NPR 21-bit computer is made up of different multiplexers and encoders. The following are the responses based on different entries to the encoder for the bus selection.

Table 2.3-1 Encoder for bus selection circuit

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

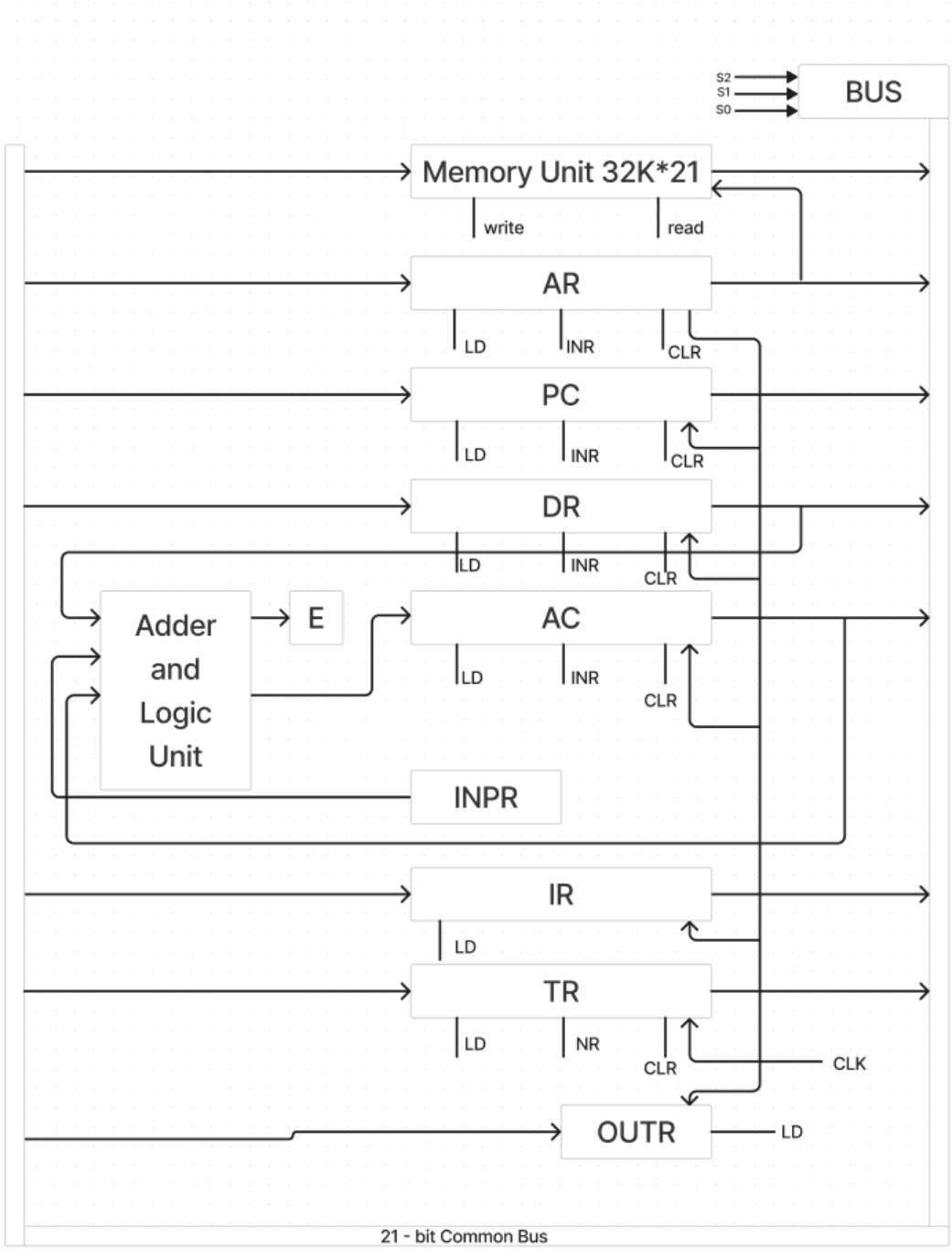

Figure 2.3-1 Common Bus for the NPR Computer

2.4 Control Unit

The Control Unit, an integral component of the Central Processing Unit (CPU), plays a crucial role in converting machine instructions into control signals that orchestrate the corresponding microoperations for their execution. By receiving the opcode and timing signals, the Control Unit generates the requisite control signals, enabling the CPU to operate seamlessly according to the specified directives. The Control Unit configuration for the NPR 21-bit Computer is detailed as follows:

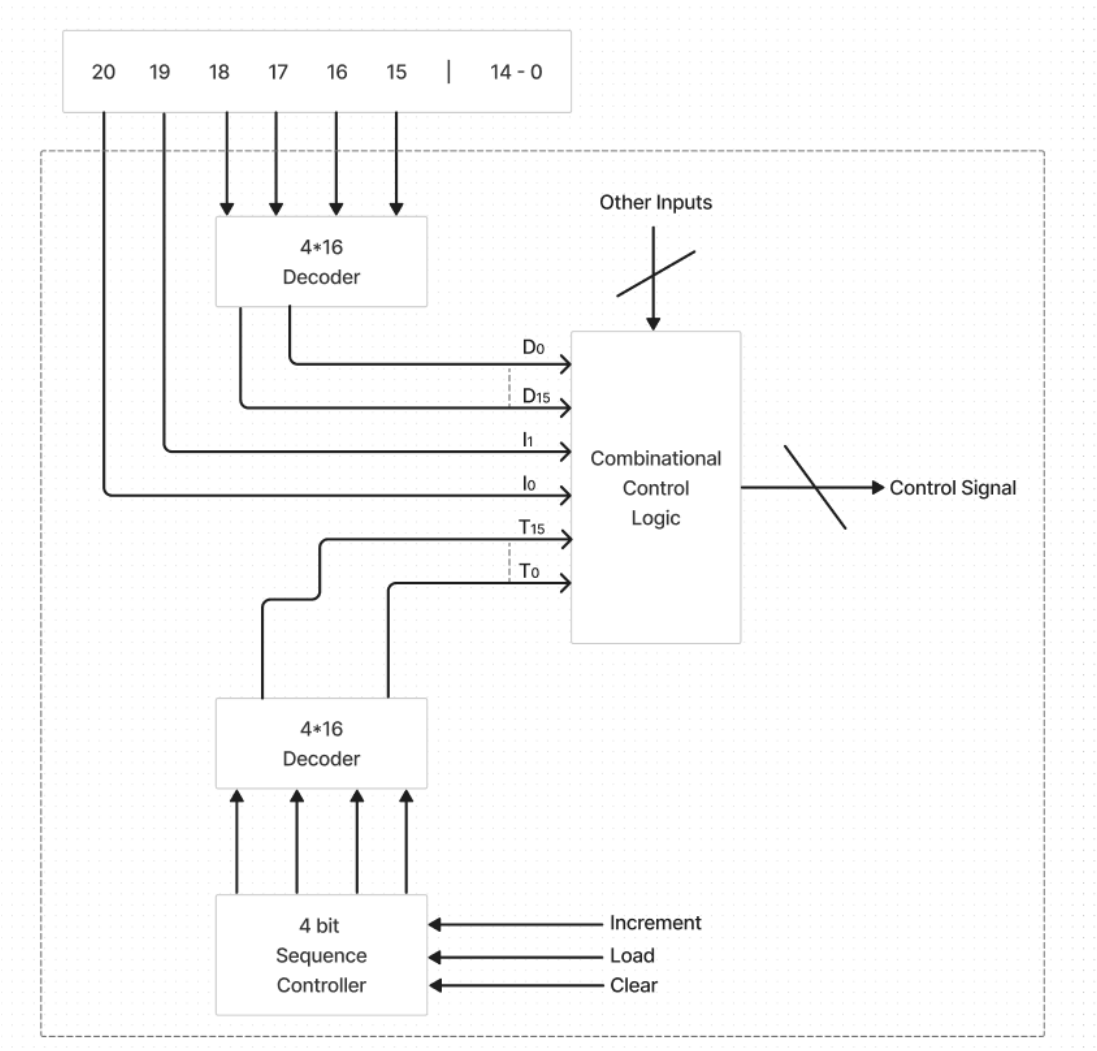

Figure 2.4-1 Control Unit of NPR Computer

2.5 Decoder and Encoder

Two decoders of 4*16 timing decoder,3*8 encoder and 4*16 operation decoders are used in the NPR 21-bit computer.

2.5.1 Timing Decoder

The timing decoder of 4*16 is used to decode the timing sequence from the clock pulse which is used to synchronize the all the elements.

2.5.2 Opcode Decoder

The opcode decoder of 4*16 is used to decode the opcode in the instructions fetch from the memory.

2.5.3 Encoder

The 3*8 encoder is used to encode the signals for passing through the common bus to select the required registers and the memory units. The encoded sequences are as follows:

Table 2.5-1 Encoder for bus selection circuit

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

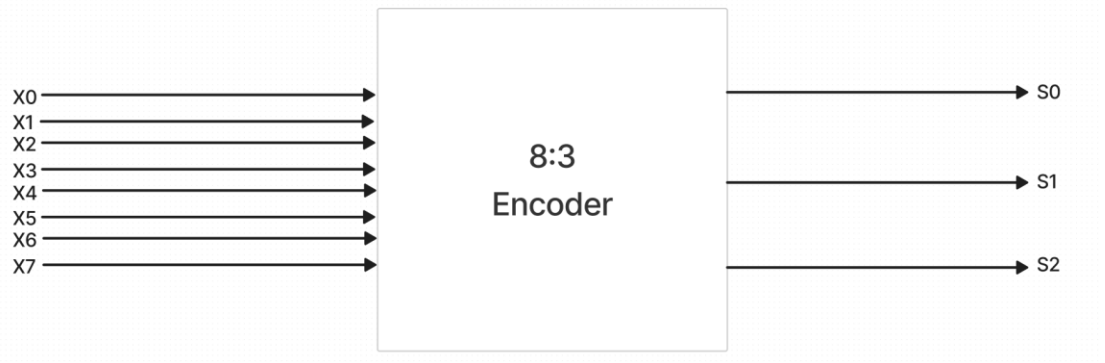

Figure 2.5-1 Encoder for Bus Selection Input

2.6 Instruction Set Architecture

The instruction consists of 21 bits. The first two bits are used for addressing modes, followed by 4 bits for opcode and the final 15 bits for the address in memory.

2.6.1 Memory Reference Instruction

Memory reference instruction can be categories by those instructions that require accessing or modifying data in the computer's memory. These instructions usually involve actions like putting data from registers into memory, loading data from memory into registers, and doing arithmetic operations on memory data. In our computer, the opcode of memory reference instructions is expected to be within the range of 0000 to 1110. The mode of the operation is determined by the 20^{th} bit and 19^{th} bit.

Table 2.6-1 Memory Reference Instruction

20	19	18 ... 14	0
I_0	I_1	OPCODE	Address

2.6.2 Addressing mode for Memory Reference Instruction

For memory reference instructions, the following are the combinations of the addressing mode bits:

Table 2.6-2 Addressing mode of 'NPR Computer.'

I_0	I_1	**Addressing mode**
0	0	Direct Addressing mode
0	1	Indirect Addressing mode
1	0	Immediate Addressing mode
1	1	NONE

2.6.3 Register-Reference Instruction

Register-Reference Instruction is defined as data kept in CPU registers is operated upon by instructions. Direct calculations or manipulations of data contained in registers are carried out via these instructions. Data transfers between registers, logical operations, and arithmetic computations are examples of common operations. Opcode with value 1111 is used for either register reference instruction or input output instructions. For register reference instruction the value of both addressing mode bits (i.e. 20^{th} bit and 19^{th} bit) must be 0.

Table 2.6-3 Register-Reference Instruction

20	19	18 … 14	0
0	0	1111	Address

2.6.4 Input-Output Instruction

Input-Output (I/O) Instructions are used for communication between the computer and external devices, such as keyboards, displays, or storage devices. The data

flow between the CPU and input/output devices is facilitated by these instructions. Input output instructions have an opcode value of 1111, and they also have the addressing mode bits (i.e. 20^{th} bit and 19^{th} bit) value of 1.

Table 2.6-4 Input-Output Instruction

20	19	18 — 14	0
1	1	1111	Address

2.7 Instruction Set

2.7.1 Memory Reference Instruction

The following are the combinations of opcode for the different memory reference instructions:

Table 2.7-1 Memory Reference Instruction

Symbol	**Hexadecimal Code**			**Description**
	$I_0I_1 = 00$	**$I_0I_1 = 01$**	**$I_0I_1 = 10$**	
OR	00(0-7)XXX	08(0-7)XXX	10(0-7)XXX	OR memory word to AC
AND	00(8-F)XXX	08(8-F)XXX	10(8-F)XXX	AND memory word to AC
SEQ	01(0-7)XXX	09(0-7)XXX	11(0-7)XXX	Skip if equal
ISZ	01(8-F)XXX	09(8-F)XXX	11(8-F)XXX	Increment and skip if zero

BUN	02(0-7)XXX	0A(0-7)XXX	12(0-7)XXX	Branch unconditionally
DSZ	02(8-F)XXX	0A(8-F)XXX	12(8-F)XXX	Decrement and skip if zero
XOR	03(0-7)XXX	0B(0-7)XXX	13(0-7)XXX	XOR memory word to AC
STA	03(8-F)XXX	0B(8-F)XXX	13(8-F)XXX	Store content of AC in memory
XCHG	04(0-7)XXX	0C(0-7)XXX	14(0-7)XXX	Exchange AC and memory
NOR	04(8-F)XXX	0C(8-F)XXX	14(8-F)XXX	NOR memory word to AC
NAND	05(0-7)XXX	0D(0-7)XXX	15(0-7)XXX	NAND memory word to AC
LDA	05(8-F)XXX	0D(8-F)XXX	15(8-F)XXX	Load memory to AC
ADD	06(0-7)XXX	0E(0-7)XXX	16(0-7)XXX	ADD memory word to AC
BSA	06(8-F)XXX	0E(8-F)XXX	16(8-F)XXX	Branch and save return address
SUB	07(0-7)XXX	0F(0-7)XXX	17(0-7)XXX	Subtract memory from AC

The following table shows the used decoder by the instruction set and their internal working:

Table 2.7-2 Decoder used by Memory Reference Instruction

Symbol	Operation Decoder	Symbolic description	Operation
OR	D_0	AC←ACvDR	OR memory word to AC
AND	D_1	AC←AC^DR	AND memory word to AC
SEQ	D_2	AC←AC⊕DR, (if (AC=0) then PC←PC+1)	Skip if equal
ISZ	D_3	M[AR]←M[AR]+1, (if (DR=0 then PC←PC+1)	Increment and skip if zero
BUN	D_4	PC←AR, SC←0	Branch unconditionally
DSZ	D_5	M[AR]←M[AR]-1, (if (DR=0 then PC←PC+1)	Decrement and skip if zero
XOR	D_6	AC←AC⊕DR	XOR memory word to AC
STA	D_7	M[AR]←AC	Store content of AC in memory
XCHG	D_8	TR←AC AC←DR DR←TR	Exchange AC and memory
NOR	D_9	AC←(ACvDR)’	NOR memory word to AC

NAND	D_{10}	AC← (AC^DR)’	NAND memory word to AC
LDA	D_{11}	AC←DR	Load memory to AC
ADD	D_{12}	AC←AC+DR	ADD memory word to AC
BSA	D_{13}	M[AR]←PC, PC←AR+1	Branch and save return address
SUB	D_{14}	AC←AC+DR’+1	Subtract memory from AC

2.7.2 Register Reference Instructions

The following table shows the combination of address bits for different instructions:

Table 2.7-3 Register Reference Instructions

Symbol	Hexadecimal Code	Description
SZE	7A000	Skip if zero in the extended accumulator (E)
CLA	79000	Clear the accumulator (AC)
CME	78800	Complement the extended accumulator (E)
HLT	78400	Halt computer, set S to zero
CIL	78200	Circulate the accumulator left
SZA	78100	Skip if zero in the accumulator (AC)
CIR	78080	Circulate the accumulator right
CMA	78040	Complement the accumulator (AC)

INC	78020	Increment the accumulator (AC)
CLE	78010	Clear the extended accumulator (E)
DEC	78008	Decrement the accumulator (AC)
SPA	78004	Skip if positive in the accumulator (AC)
SNA	78002	Skip if negative in the accumulator (AC)

The following table shows the decoders used and the operations done by the instruction set:

Table 2.7-4 Decoder used by Register Reference Instructions

Symbol		**Symbolic Description**	**Operation**
	r	SC←0	Clear SC
SZE	rB_{13}	If (E = 0) then (PC←PC+1)	Skip if zero in the extended accumulator (E)
CLA	rB_{12}	AC←0	Clear the accumulator (AC)
CME	rB_{11}	E←E'	Complement the extended accumulator (E)
HLT	rB_{10}	S←0	Halt computer, set S to zero
CIL	rB_{9}	AC← shl AC, AC (0) ←E, E←AC (20)	Circulate the accumulator left
SZA	rB_{8}	if (AC = 0) then	Skip if zero in the accumulator

		(PC←PC+1)	(AC)
CIR	rB_7	AC← shr AC, AC (20) ←E, E←AC (0)	Circulate the accumulator right
CMA	rB_6	AC←AC'	Complement the accumulator (AC)
INC	rB_5	AC←AC+1	Increment the accumulator (AC)
CLE	rB4	E←0	Clear the extended accumulator (E)
DEC	rB_3	AC←AC-1	Decrement the accumulator (AC)
SPA	rB_2	If (AC (20) = 0) then (PC←PC+1)	Skip if positive in the accumulator (AC)
SNA	rB_1	If (AC (20) = 1) then (PC←PC+1)	Skip if negative in the accumulator (AC)

2.7.3 Input Output Instructions

The following table shows the combination of address bits for different instructions:

Table 2.7-5 Input Output Instructions

Symbol	Hexadecimal Code	Description
INP	1F8100	Store content of INTR to AC (0-7)
SID	1F8080	Skip if input flag is disabled
OUT	1F8040	Store content of AC (0-7) to OUTR
ION	1F8020	Interrupt enable ON
IOF	1F8010	Interrupt enable OFF
SOE	1F8008	Skip if output flag is enabled
SIE	1F8004	Skip if input flag in enabled
SOD	1F8002	Skip if output flag is disabled

The following table shows the decoders used and the operations performed by the instruction set:

Table 2.7-6 Decoder used by Input Output Instructions

Symbol		Symbolic Description	Operation
	p	SC←0	Clear SC
INP	pB8	AC (0-7) ←INPR, FGI←0	Store content of INTR to AC

			(0-7)
SID	pB7	if (FGI=0) then (PC←PC+1)	Skip if input flag is disabled
OUT	pB6	OUTR← AC (0-7), FGO←0	Store content of AC (0-7) to OUTR
ION	pB5	IEN←1	Interrupt enable ON
IOF	pB4	IEN←0	Interrupt enable OFF
SOE	pB3	if (FGO=1) then (PC←PC+1)	Skip if output flag is enabled
SIE	pB2	if (FGI=1) then (PC←PC+1)	Skip if input flag in enabled
SOD	pB1	if (FGO=0) then (PC←PC+1)	Skip if output flag is disabled

2.8 Instruction Cycle

2.8.1 Fetch

The fetch cycle serves as the initial stage within the instruction execution process of a computer's Central Processing Unit (CPU). During this phase, the CPU retrieves the subsequent instruction from the system's memory by accessing the memory location specified by the Program Counter (PC). The fetched instruction is then stored temporarily within the Instruction Register (IR), facilitating its subsequent decoding and execution. The Program Counter is subsequently incremented to point to the subsequent memory address, ensuring the continuous retrieval and processing of instructions, thereby forming the foundation for the subsequent phases of the instruction cycle.

2.8.2 Decode

Following the fetch cycle, the decode cycle represents the subsequent phase wherein the CPU interprets and deciphers the fetched instruction stored within the Instruction Register (IR). During this pivotal phase, the CPU's Control Unit examines the opcode portion of the instruction to identify the specific operation or microoperation that the instruction mandates. Additionally, the Control Unit determines the requisite operands and memory locations, subsequently generating the corresponding control signals that guide the execution unit's subsequent operation. The decode cycle plays a crucial role in translating the machine-level instructions into actionable commands, facilitating the seamless execution of complex computational tasks within the CPU.

2.8.3 Execute the instruction.

After the decode cycle, the execute cycle represents the concluding phase within the instruction execution process, wherein the CPU proceeds to execute the decoded instruction's specified operation. Based on the decoded instruction and generated control signals, the CPU's execution unit performs the designated arithmetic, logic, or data transfer operation, manipulating data within the registers or transferring data between the registers and memory. Upon completing the specified operation, the CPU may update the relevant registers, flags, or memory locations as necessitated by the instruction's execution requirements. The execute cycle encapsulates the core computational activity within the CPU, enabling the realization of diverse computational tasks and operations specified by the fetched and decoded instructions.

Chapter 3 Register Transfer Language

3.1 Fetch Cycle:

Table 3.1-1 Register Transfer Language of Fetch Cycle

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

3.2 Decode Cycle:

Table 3.2-1 Register Transfer Language of Decode Cycle

$R'T_2$:	I_0←IR (20), I_1←IR (19), $D_0, D_1, \ldots, D_{15}$←Decode IR (18-15), AR←IR (0-14)

3.3 Interrupt Cycle:

Table 3.3-1 Register Transfer Language of Interrupt Cycle

$T_0'T_1'T_2'$. (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

3.4 Execution Cycle:

- **Direct addressing mode:**

Table 3.4-1 Register Transfer Language of Direct addressing mode

$D_{15}'. T_3.I_0'. I_1'$:	DR←M[AR]
$D_{15}'. T_4.I_0'. I_1'$:	Do Nothing

- **Indirect addressing mode:**

Table 3.4-2 Register Transfer Language of Indirect addressing mode

$D_{15}'. T_3.I_0'.I_1$:	AR←M[AR]
$D_{15}'. T_4.I_0'.I_1$:	DR←M[AR]

- **Immediate addressing mode:**

Table 3.4-3 Register Transfer Language of Immediate addressing mode

$D_{15}'. T_3.I_0.I_1'$:	DR←AR
$D_{15}'. T_4.I_0.I_1'$:	Do nothing

- **Register Reference Instruction:**

Table 3.4-4 Register Transfer Language of Register Reference Instruction

$D_{15}.T_3.I_0'. I_1'$:	Execute an RRI

- **Input Output Instruction:**

Table 3.4-5 Register Transfer Language of Input Output Instruction

$D_{15}.T_3.I_0.I_1$:	Execute an IOI

3.5 Memory Reference Instructions

Table 3.5-1 Register Transfer Language of Memory Reference Instructions

Instruction	Control Function	
OR	D_0T_5	AC←ACvDR, SC←0
AND	D_1T_5	AC←AC^DR, SC←0
SEQ	D_2T_5	TR←AC, AC←AC⊕DR,
	D_2T_6	If (AC = 0) then (PC=PC+1), AC←TR, SC←0
ISZ	D_3T_5	DR←DR+1
	D_3T_6	M[AR]←DR (if DR=0 then PC←PC+1), SC←0
BUN	D_4T_5	PC←AR, SC←0
DSZ	D_5T_5	DR←DR-1
	D_5T_6	M[AR]←DR (if DR = 0 then PC←PC+1), SC←0
XOR	D_6T_5	AC←AC⊕DR, SC←0
STA	D_7T_5	M[AR]←AC, SC←0
XCHG	D_8T_5	TR←AC
	D_8T_6	AC←DR
	D_8T_7	DR←TR, SC←0
NOR	D_9T_5	AC←(ACvDR)', SC←0
NAND	$D_{10}T_5$	AC← (AC^DR)', SC←0

LDA	$D_{11}T_5$	AC←DR,
ADD	$D_{12}T_5$	AC←AC+DR, E←C_{out}, SC←0
BSA	$D_{13}T_5$	M[AR]←PC, AR←AR+1
	$D_{13}T_6$	PC←AR, SC←0
SUB	$D_{14}T_5$	AC←AC+DR'+1, SC←0

3.6 Register Reference Instruction

Table 3.6-1 Register Transfer Language of Register Reference Instructions

r =$D_{15}I_0'I_1'T_3$ (Common to all register-reference instruction)		
B_i =IR(i), where i= 1, 2……,13		
	r	SC←0
SZE	rB_{13}	if (E = 0) then (PC←PC+1)
CLA	rB_{12}	AC←0
CME	rB_{11}	E←E'
HLT	rB_{10}	S←0
CIL	rB_9	AC← shl AC, AC (0) ←E, E←AC (20)
SZA	rB_8	if (AC = 0) then (PC←PC+1)
CIR	rB_7	AC← shr AC, AC (20) ←E, E←AC(0)
CMA	rB_6	AC←AC'
INC	rB_5	AC←AC+1

CLE	rB_4	E←0
DEC	rB_3	AC←AC-1
		AC(20)==0
SPA	rB_2	if(AC(20) = 0) then (PC←PC+1)
SNA	rB_1	if(AC(20) = 1) then (PC←PC+1)

3.7 Input-Output Instruction

Table 3.7-1 Register Transfer Language of Input Output Instruction

	$p=D_{15}I_0I_1T_3$ (Common to all input-output instructions)	
	$B_{i=}$ IR(i), where i =1, 2…8	
	p	SC←0
INP	pB_8	AC (0-7) ←INPR, FGI←0
SID	pB_7	if (FGI=0) then (PC←PC+1)
OUT	pB_6	OUTR← AC (0-7), FGO←0
ION	pB_5	IEN←1
IOF	pB_4	IEN←0
SOE	pB_3	if (FGO=1) then (PC←PC+1)
SIE	pB_2	if (FGI=1) then (PC←PC+1)
SOD	pB_1	if (FGO=0) then (PC←PC+1)

Chapter 4 Flowchart of Operations

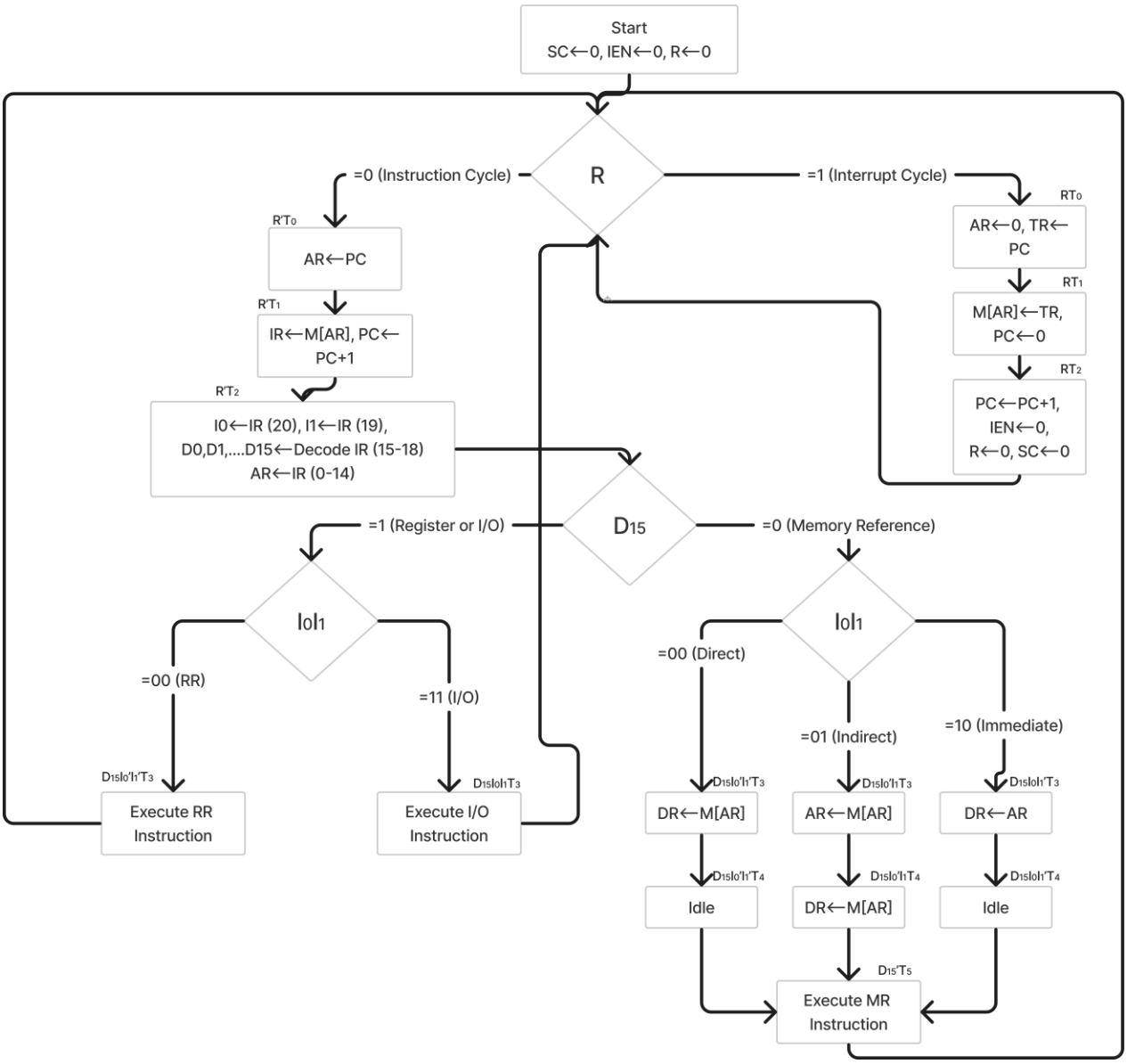

Figure 4.1 Flow Chart

Chapter 5 Design of Individual Components

5.1 Design of Registers

There are 9 different registers in the NPR 21-bit computer. They are used for storing the data temporarily. They are listed below:

5.1.1 Design of AR

AR stands for Address Register. The address register stores the address in the memory temporarily until the process is completed. AR is of 15 bits in our computer design.

Table 5.1-1 Design of AR

Control Function	**Operation**	
$R'T_0$	AR←PC	Load
$R'T_2$	AR←IR (0-14)	Load
$D_{15}'. T_3.I_0'. I_1$	AR←M[AR]	Load
RT_0	AR←0	Clear
$D_{13}T_5$	AR←AR+1	Increment
Load AR(LD)	$R'T_0 + R'T_2 + D_{15}'. T_3.I_0'. I_1$	
Increment AR (INCR)	$D_{13}\ T_5$	
Clear AR(CLR)	RT_0	

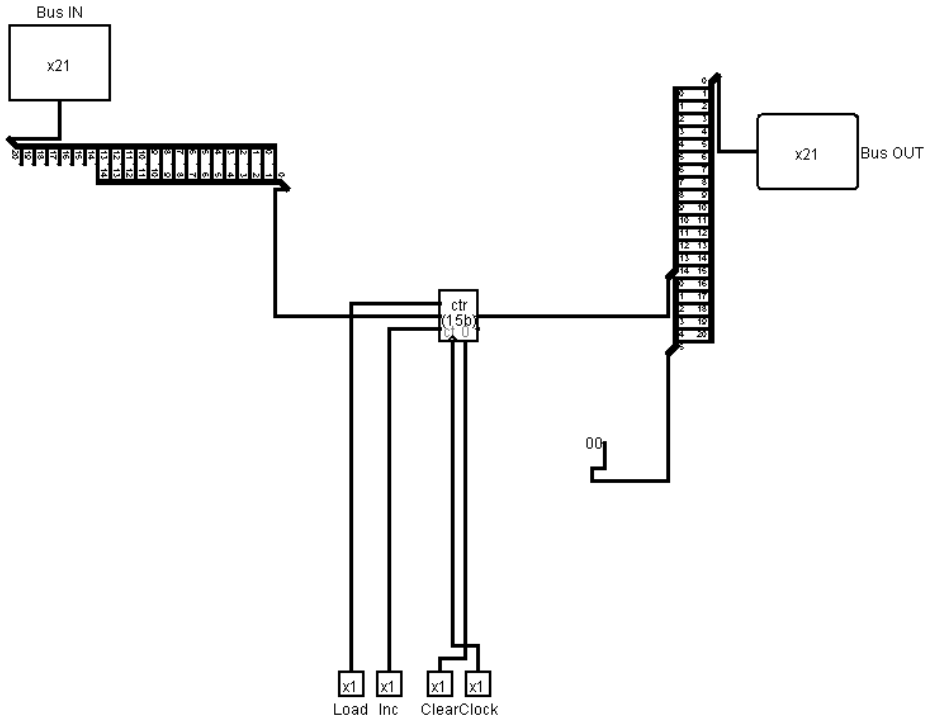

Figure 5.1-1 Design of AR

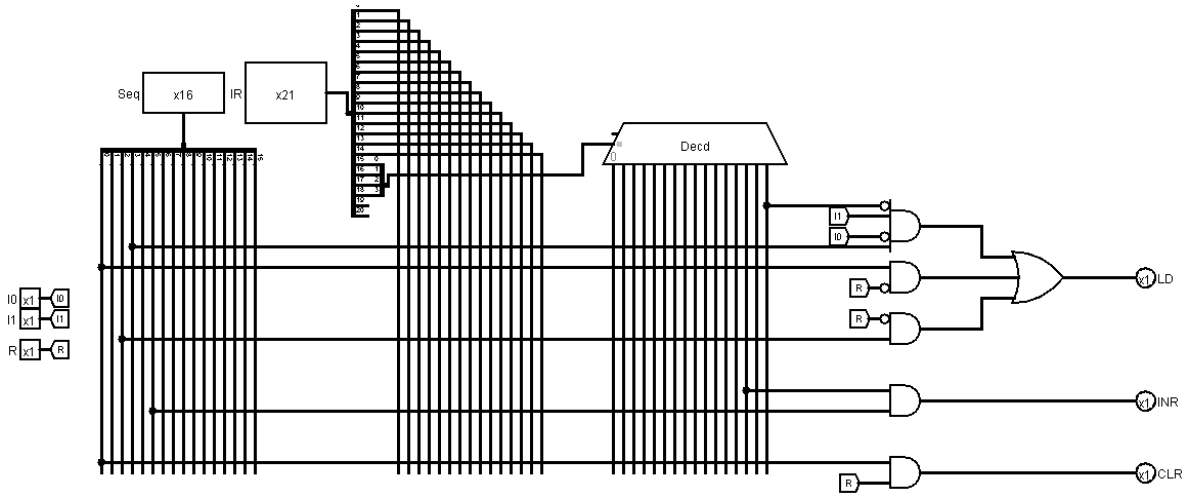

Figure 5.1-2 Control of AR

5.1.2 Design of PC

PC stands for Program Counter. It is used to count the number of steps run by the computer. The program counter helps to track the number of instructions executed. PC is of 15 bits in our computer design.

Table 5.1-2 Design of PC

Control Function	Operation	
$R'T_1$	PC←PC+1	Increment
RT_2	PC←PC+1	Increment
D_2T_6	If (AC = 0) then (PC=PC+1)	Increment
D_3T_6	(if DR=0 then PC←PC+1)	Increment
D_5T_6	(if DR = 0 then PC←PC+1)	Increment
rB_2	if (AC (20) = 0) then (PC←PC+1)	Increment
rB_1	if (AC (20) = 1) then (PC←PC+1)	Increment
rB_8	if (AC = 0) then (PC←PC+1)	Increment
rB_{13}	if (E = 0) then (PC←PC+1)	Increment
pB_3	if (FGI=1) then (PC←PC+1)	Increment
pB_7	if (FGI=0) then (PC←PC+1)	Increment
pB_2	if (FGO=1) then (PC←PC+1)	Increment
pB_1	if (FGO=0) then (PC←PC+1)	Increment

D_4T_5	PC←AR	Load
$D_{13}T_6$	PC←AR	Load
RT_1:	PC←0	Clear
Load PC (LD)	$D_4T_5 + D_{13}T_6$	
Increment PC (INC)	$R'T_1+RT_2+D_2T_6+D_3T_6+D_5T_6+ rB_2+ rB_4+ rB_8+ rB_{13}+ pB_7+ pB_3+ pB_2+ pB_1$	
Clear PC (CLR)	RT_1	

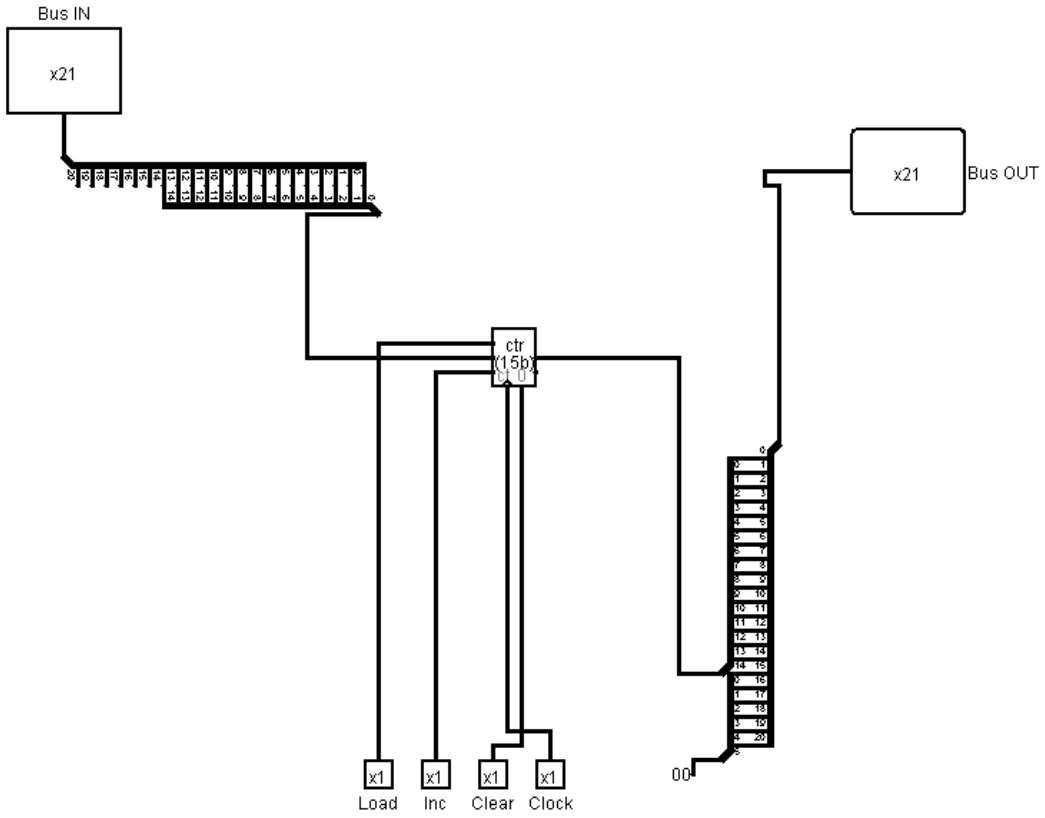

Figure 5.1-3 Design of PC

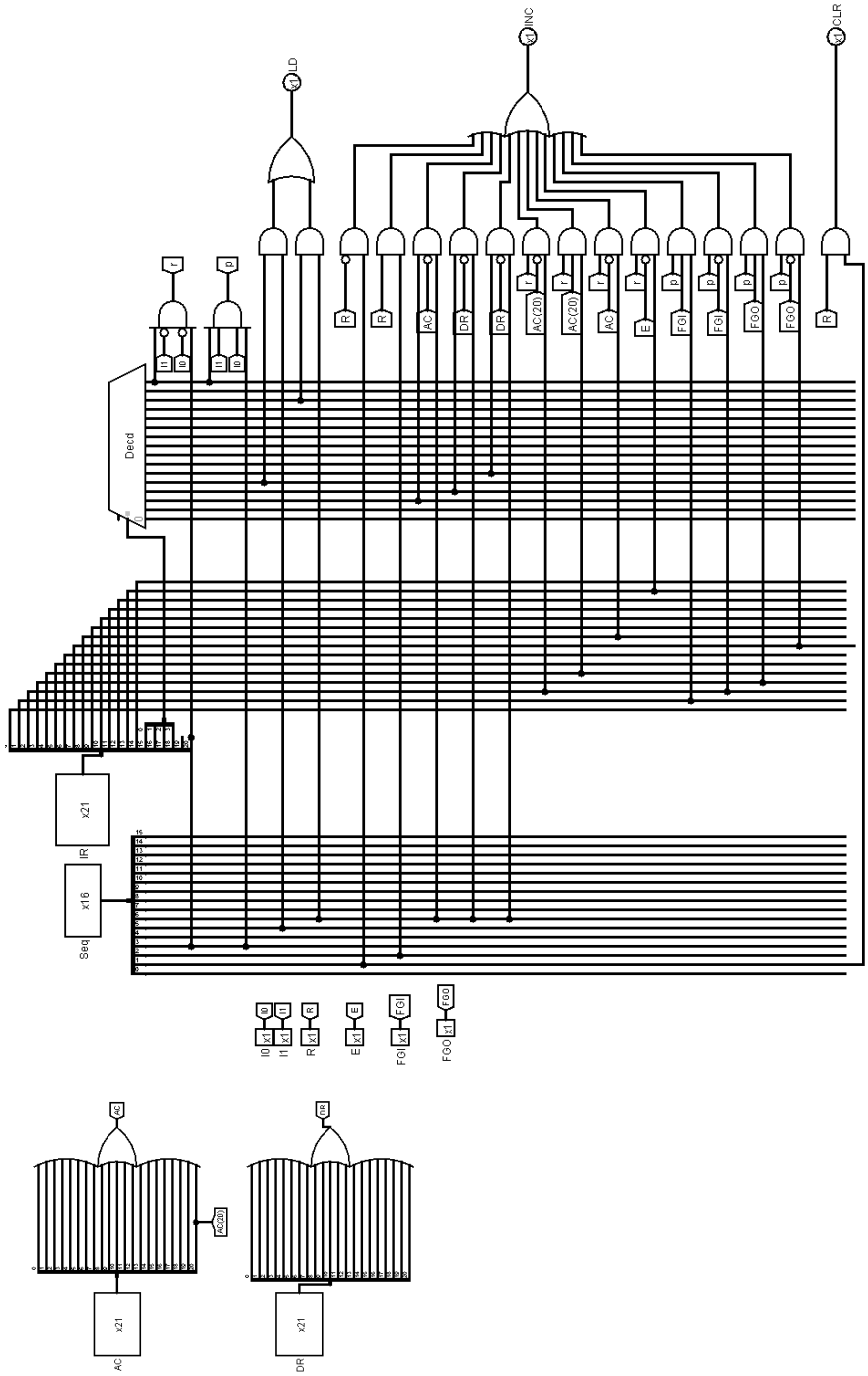

Figure 5.1-4 Control of PC

5.1.3 Design of TR

TR stands for Temporary Register. It holds the value of computation temporarily to further store it in either memory or output. It is 21 bits.

Table 5.1-3 Design of TR

Control Function	**Operation**	
RT_0	TR←PC	Load
D_2T_5	TR←AC	Load
D_8T_5	TR←AC	Load
Load TR (LD)	$=RT_0 + D_2T_5 + D_8T_5$	

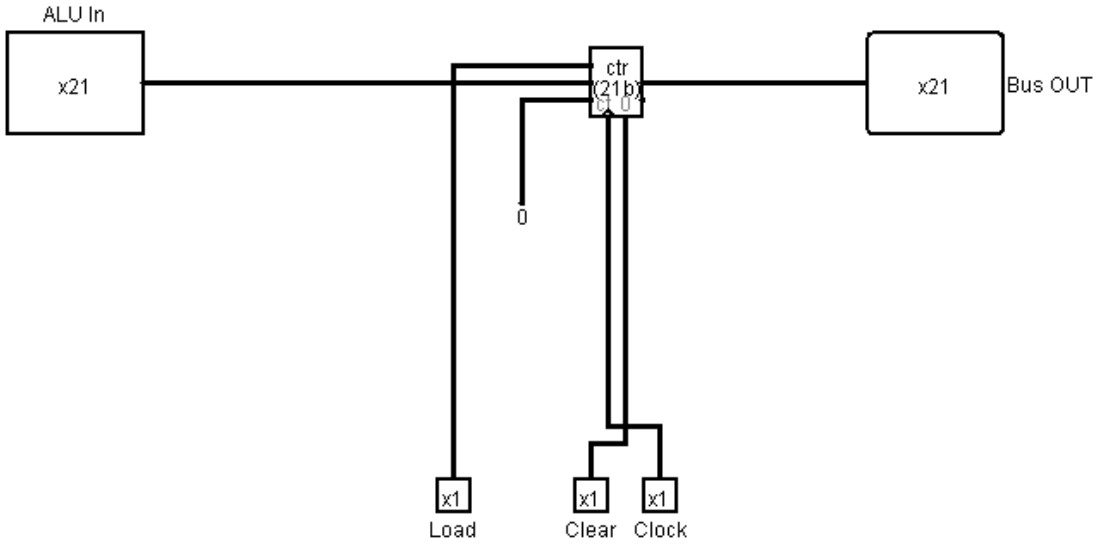

Figure 5.1-5 Control of TR

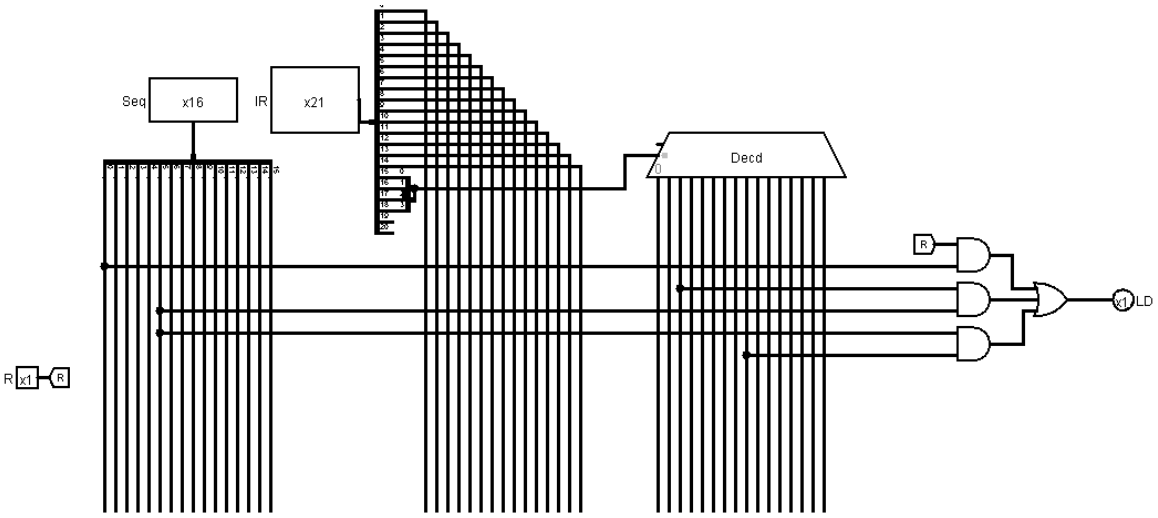

Figure 5.1-6 Control of TR

5.1.4 Design of SC

SC stands for sequence counter. As the name suggests it counts the sequence of the operations performed. After a process is completed, it is then reset to 0. It is of 4 bits in our design.

Table5.1-4 Design of SC

Control Function	Operation	
RT_2	SC←0	Clear
D_1T_5	SC←0	Clear
D_0T_5	SC←0	Clear
D_6T_5	SC←0	Clear
$D_{10}T_5$	SC←0	Clear
D_9T_5	SC←0	Clear
$D_{12}T_5$	SC←0	Clear

$D_{14}T_5$	SC←0	Clear
D_2T_6	SC←0	Clear
$D_{11}T_5$	SC←0	Clear
D_7T_5	SC←0	Clear
D_4T_5	SC←0	Clear
$D_{13}T_6$	SC←0	Clear
D_8T_7	SC←0	Clear
$D_{10}T_6$	SC←0	Clear
$D_{12}T_6$	SC←0	Clear
r	SC←0	Clear
p	SC←0	Clear
Clear SC (CLR)	$RT_2 + D_0T_5 + D_1T_5 + D_4T_5 + D_6T_5 + D_7T_5 + D_9T_5 + D_{11}T_5 + D_{12}T_5 + D_{14}T_5 + D_2T_6 + D_{13}T_6 + D_{10}T_6 + D_{12}T + p$	

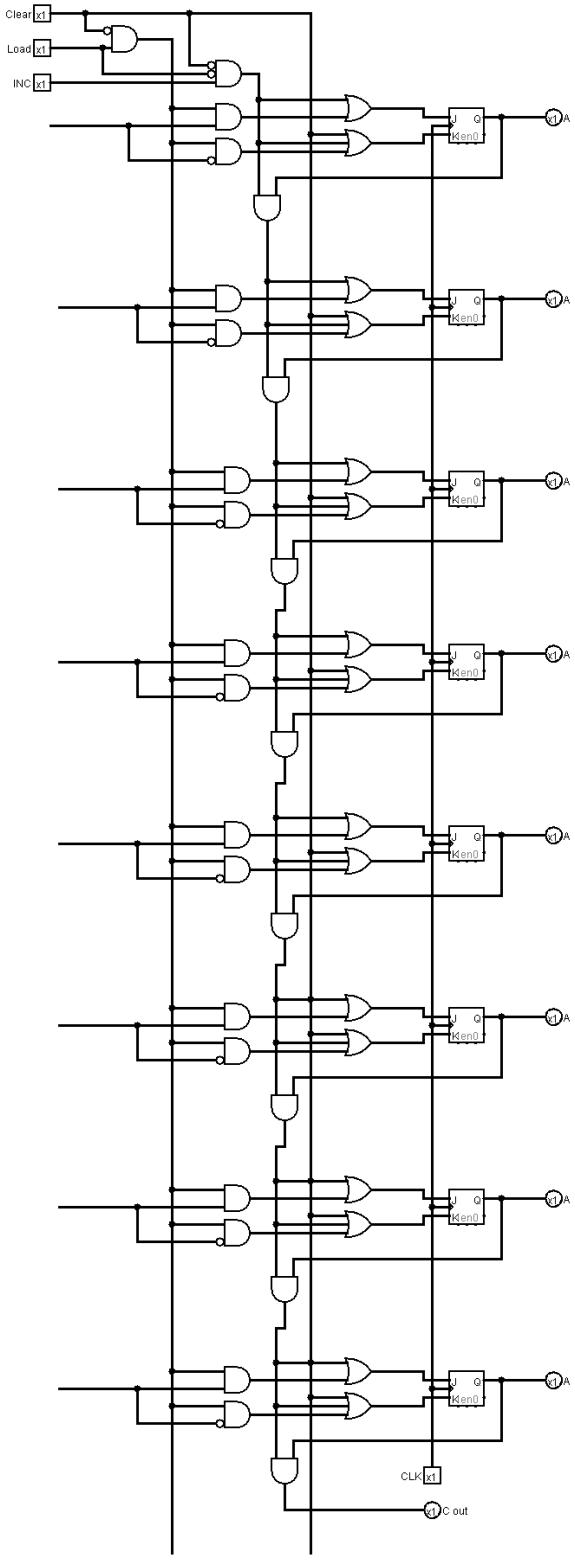

Figure 5.1-7 Design of SC

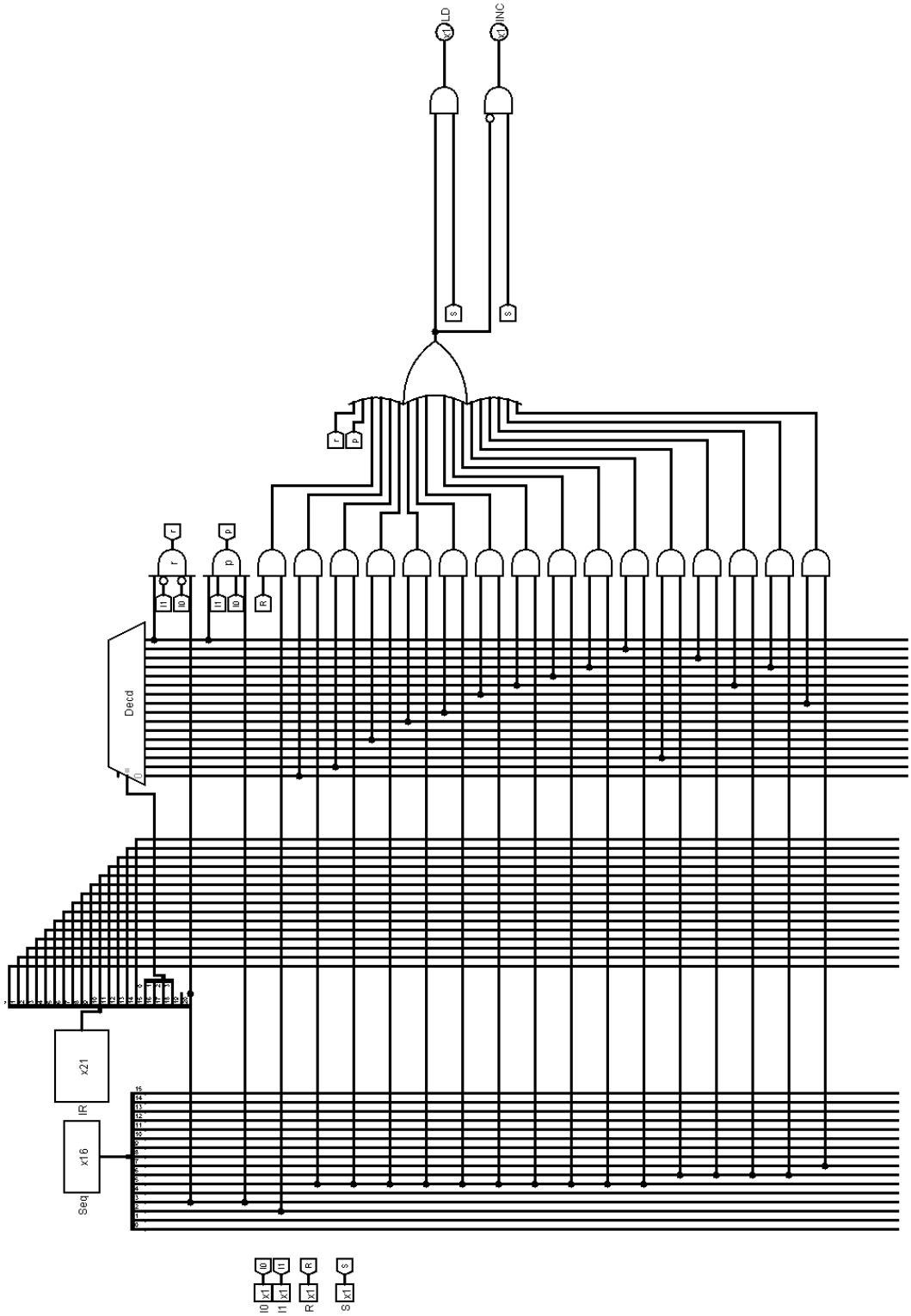

Figure 5.1-8 Control of SC

5.1.5 Design of IR

IR stands for Instruction Register. It holds the instruction to be executed in it. It is of 21 bits in our design.

Table5.1-5 Design of IR

Control Function	Operation	
$R'T_1$:	IR←M[AR]	Load

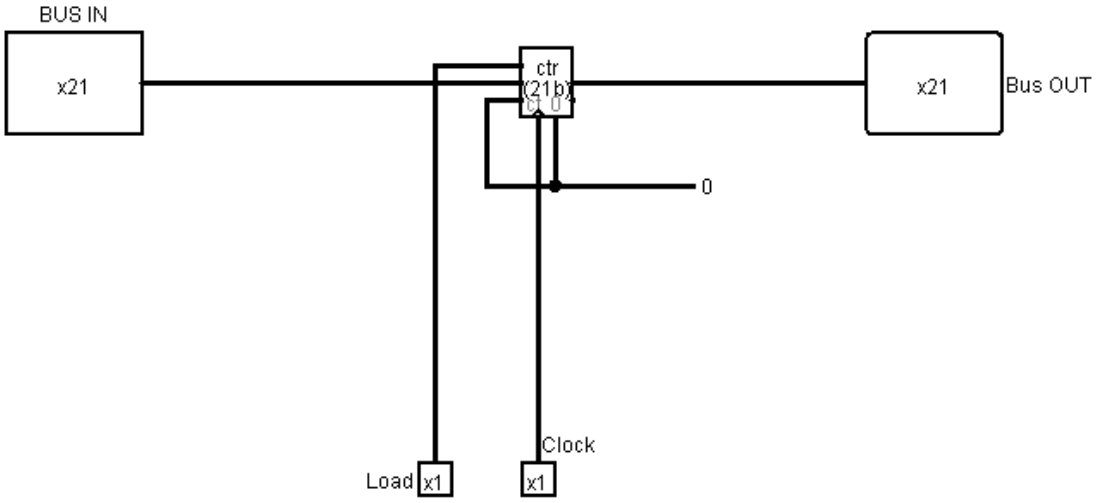

Figure 5.1-9 Design of IR

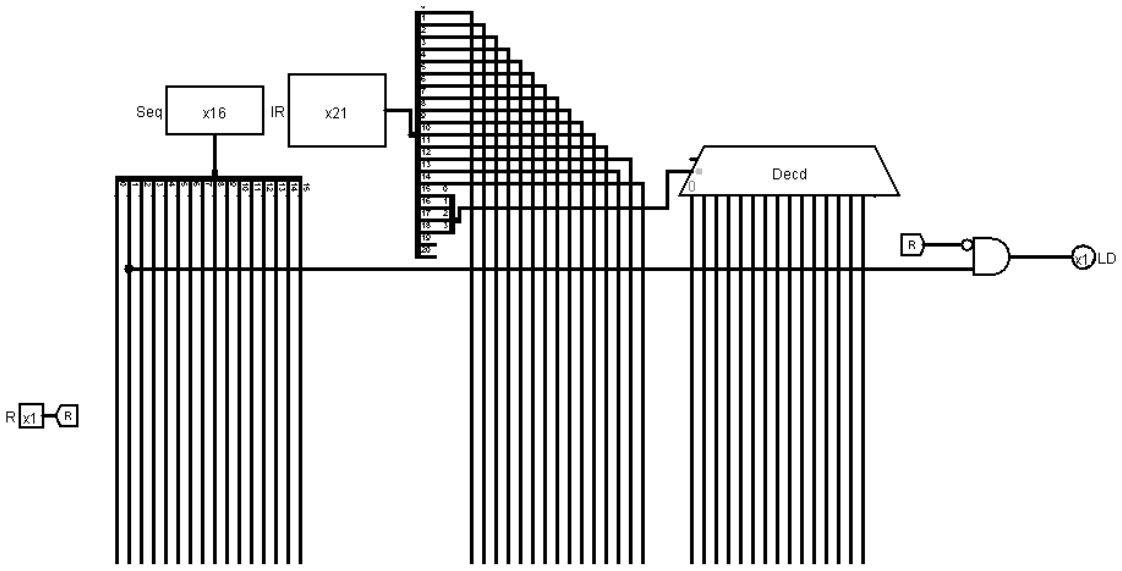

Figure 5.1-10 Control of IR

5.1.6 Design of AC

AC stands for accumulator. It holds the temporary value given by ALU for further computation. It is of 21 bits in our design.

Table 5.1-6 Design of AC

Control Function	Operation	
D_1T_5	AC←AC^DR	Load
$D_{12}T_5$	AC←AC+DR	Load
$D_{11}T_5$	AC←DR	Load
D_8T_6	AC←DR	Load
pB_{12}	AC (0-7) ←INPR	Load
D_0T_5	AC←ACvDR	Load
D_6T_5	AC←AC⊕DR	Load

D_2T_{60}	AC←TR	Load
$D_{10}T_5$	AC← (AC^DR)’	Load
D_9T_5	AC← (AC∨DR)’	Load
$D_{14}T_5$	AC←AC+DR’+1	Load
D_2T_5	AC←AC⊕DR	Load
rB_3	AC←AC-1	Load
rB_6	AC←AC’	Load
rB_7	AC←shr AC, AC (20) ←E	Load
rB_9	AC←shl AC, AC (0) ←E	Load
rB_{12}	AC←0	Increment
rB_5	AC←AC+1	Clear
Load AC(LD)	$D_1T_5 + D_{12}T_5 + D_{11}T_5 + D_8T_6 + pB_{12} + D_0T_5 + D_6T_5 + D_2T_6 + D_{10}T_5 + D_9T_5 + D_{14}T_5 + D_2T_5 + rB_3 + rB_6+rB_7 + rB_9$	
Increment AC (INC)	rB_{12}	
Clear AC (CLR)	rB_5	

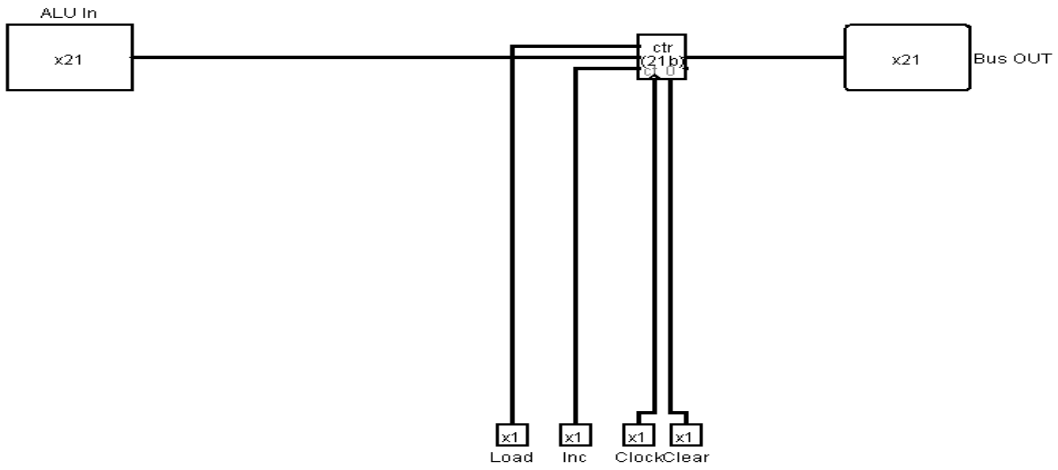

Figure 5.1-11 Design of AC

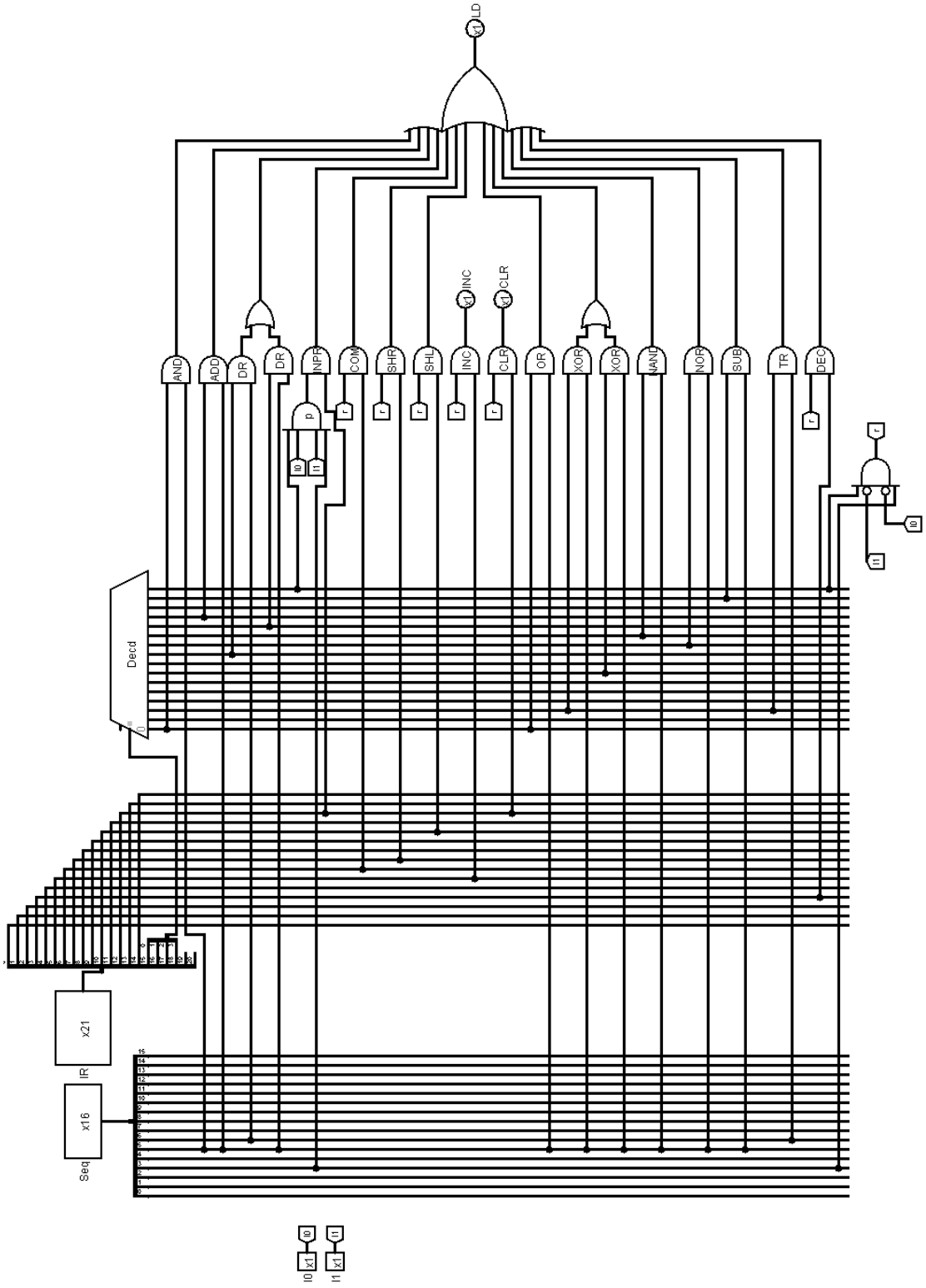

Figure 5.1-12 Control of AC

5.1.7 Design of DR

DR stands for Data Register. It fetches the data from the memory and passes it to ALU for computation. It is of 21bits in our design.

Table5.1-7 Design of DR

Control Function	**Operation**	
D_8T_7	DR←TR	Load
$D_{10}T_5$	DR←DR+1	Increment
$D_{12}T_5$	DR←DR-1	Load
$D_{15}'. T_3.I_0'. I_1'$:	DR←M[AR]	Load
$D_{15}'. T_4.I_0'. I_1$:	DR←M[AR]	Load
$D_{15}'. T_3.I_0.I_1'$:	DR←AR	Load
Load DR (LD)	$D_8T_7 + D_{12}T_5 + D_{15}'. T_3.I_0'. I_1' + D_{15}'. T_4.I_0'. I_1 + D_{15}'. T_3.I_0.I_1'$	
Increment DR (INC)	$D_{10}T_5$	

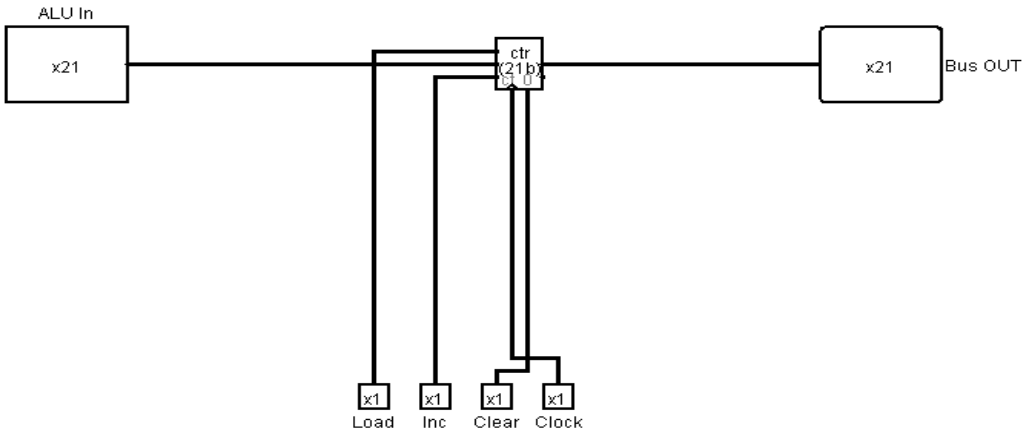

Figure 5.1-13 Design of DR

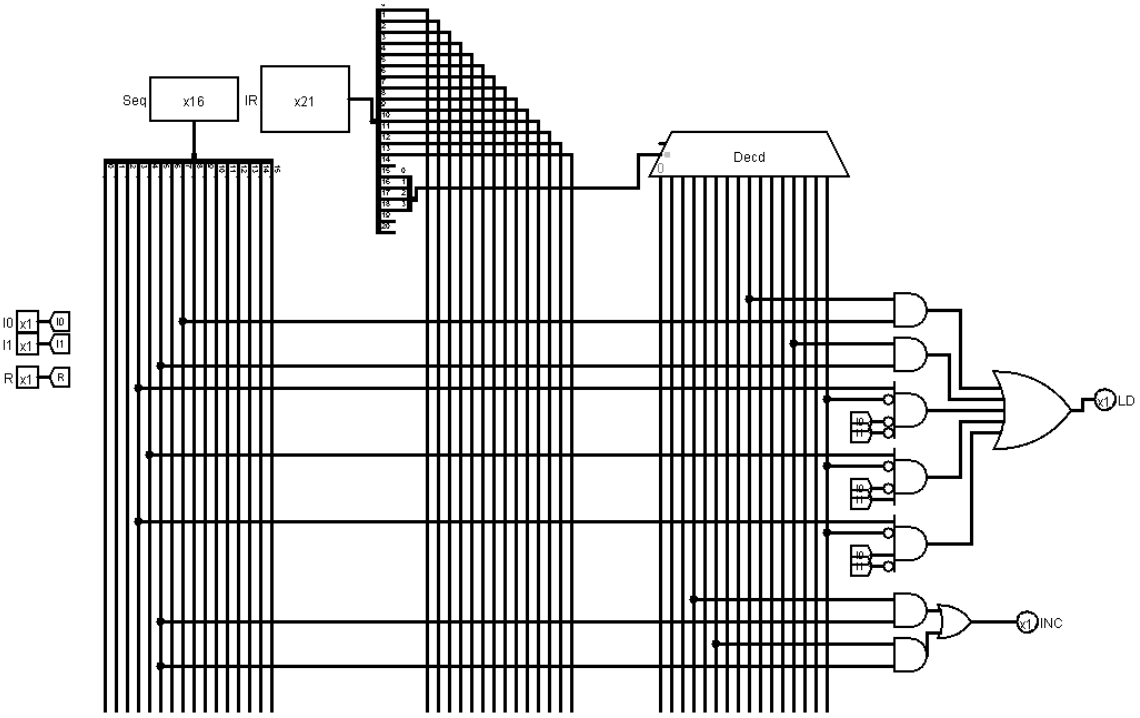

Figure 5.1-14 Control of DR

5.1.8 Design of OUTR

OUTR is a register used for temporary storage of data before sending to the output device. It is of 8 bits in our design.

Table 5.1-8 Design of OUTR

Control Function	Operation	
pB_6	IEN←1	

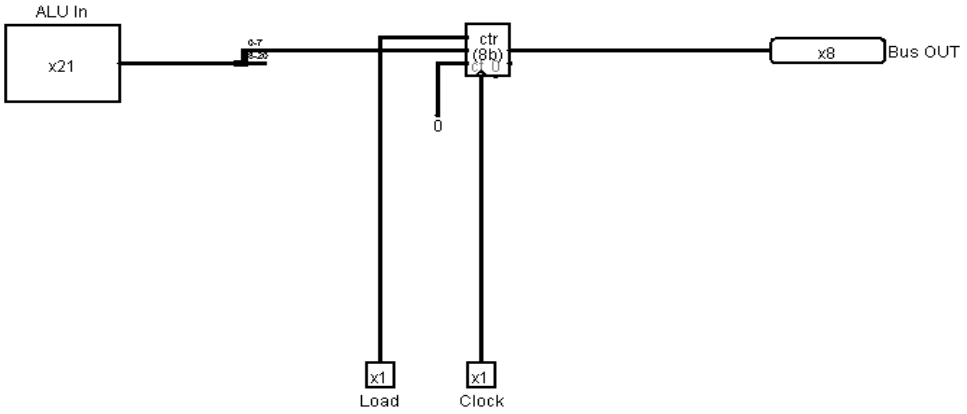

Figure 5.1-15 Design of OUTR

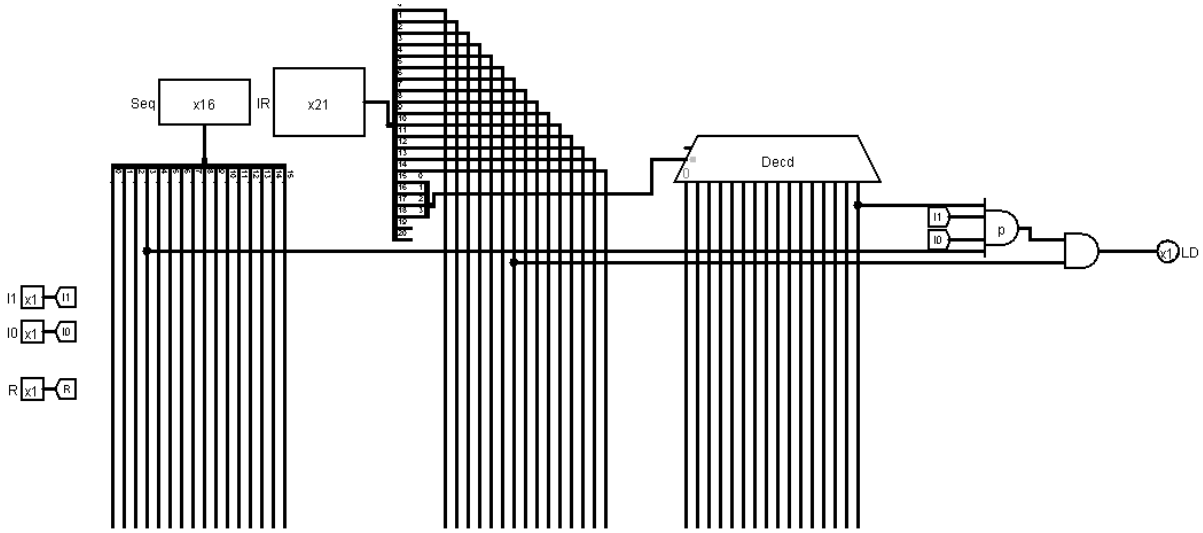

Figure 5.1-16 Control of OUTR

5.2 Design of flags

5.2.1 Design of IEN

IEN is a flag in our basic computer. It stands for Input Enable.

Table 5.2-1 IEN

Control Function	**Operation**	
pB_5	IEN←1	Set
pB_4	IEN←0	Reset
rT_2	IEN←0	Reset
Set	pB_5	
Reset	rT_2+ pB_4	

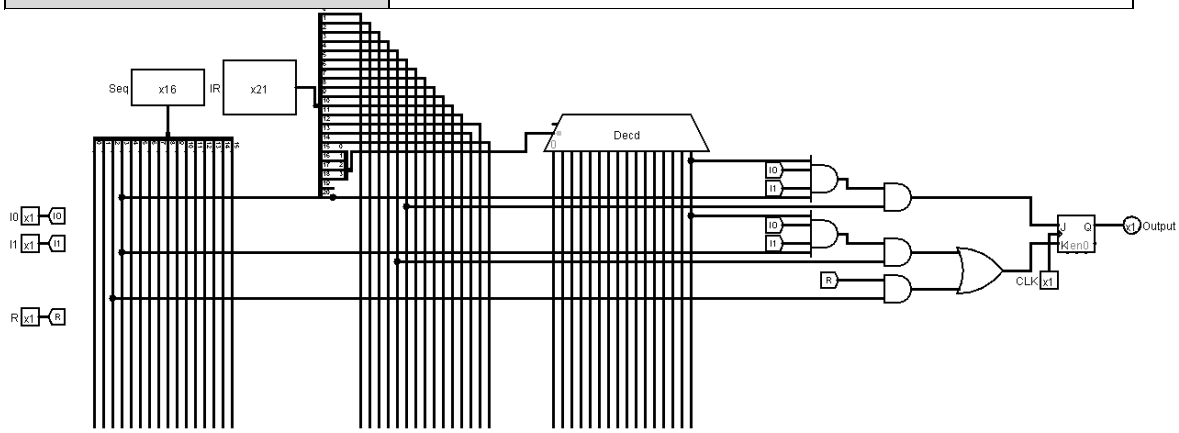

Figure 5.2-1 Design of IEN

5.2.2 Design of FGO

FGO stands for flag output.

Table 5.2-2 Design of FGO

Control Function	Operation	
pB_6	FGO←0	Reset
Reset	pB_6	

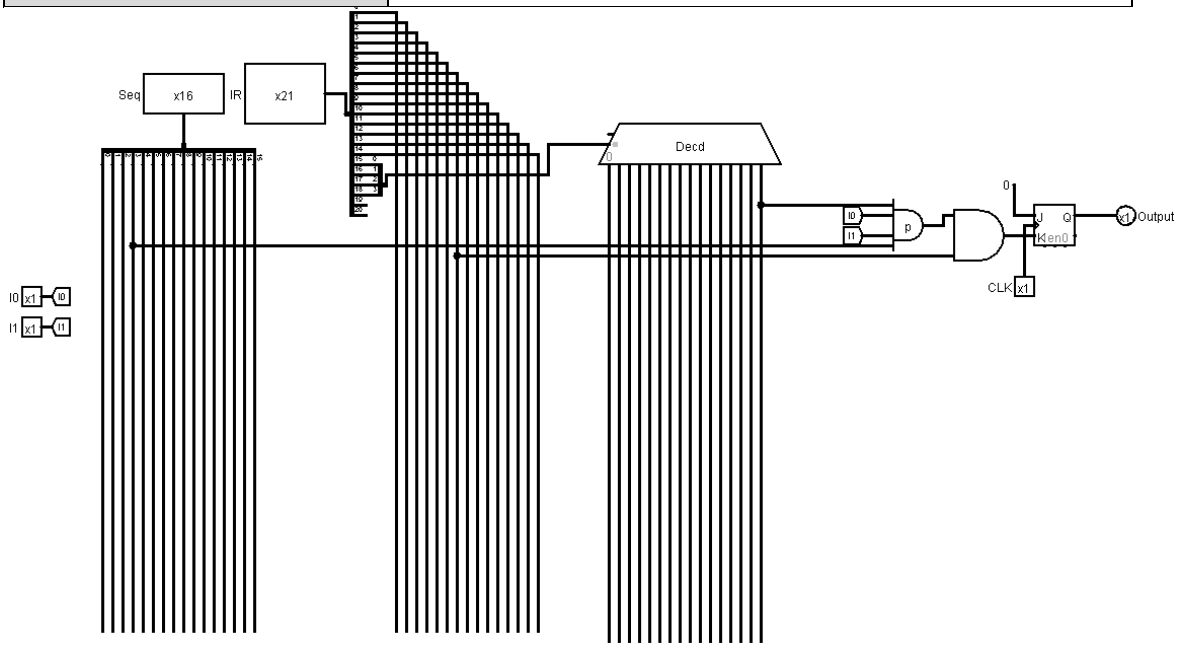

Figure 5.2-2 Design of FGO

5.2.3 Design of FGI

FGI stands for Flag Input.

Table 5.2-3 Design of FGI

Control Function	Operation	
pB_8	FGI←0	Reset
Reset	pB_8	

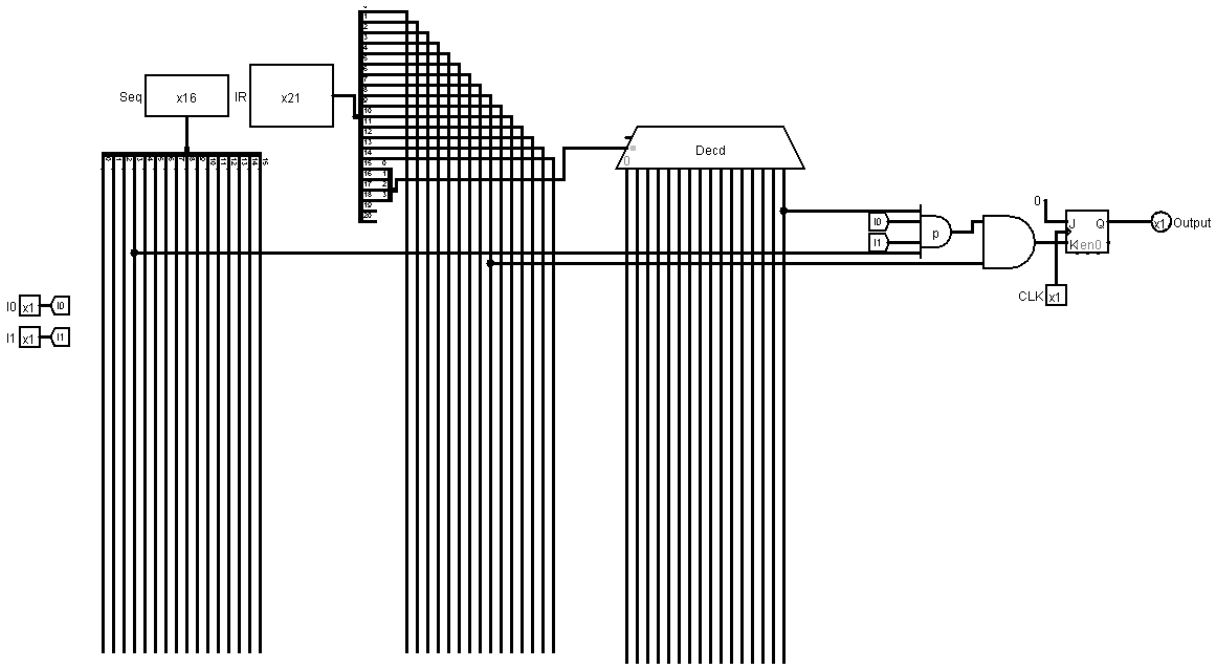

Figure 5.2-3 Design of FGI

5.2.4 Design of R

The R is an interrupt. When R is 0, the instruction cycle is proceeded, otherwise the interrupt cycle is processed.

Table 5.2-4 Design of R

Control Function	**Operation**	
T_0'T_1'T_2'. (IEN). (FGI+FGO):	R←1	Set
rT_2:	R←0	Reset
Reset	rT_2	
Set	T_0'T_1'T_2'. (IEN). (FGI+FGO)	

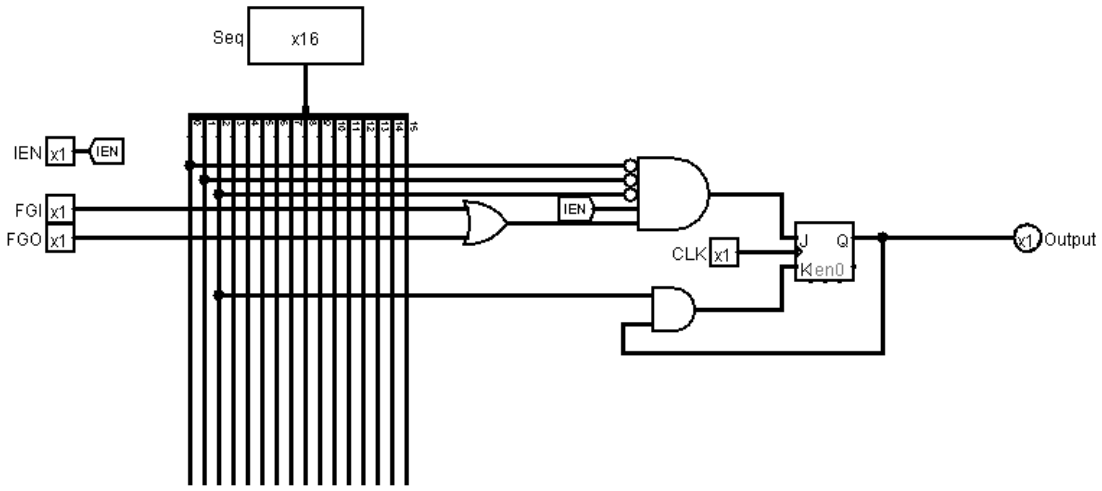

Figure 5.2-4 Design of R

5.2.5 Design of E

Table 5.2-5 Design of E

Control Function	Operation	
rB_4	E←0	Clear
rB_{11}	E←E’	Comp
rB_7	E←AC (0)	Load
rB_9	E←AC (20)	Load
$D_{12}T_5$	E←C_{out}	Load

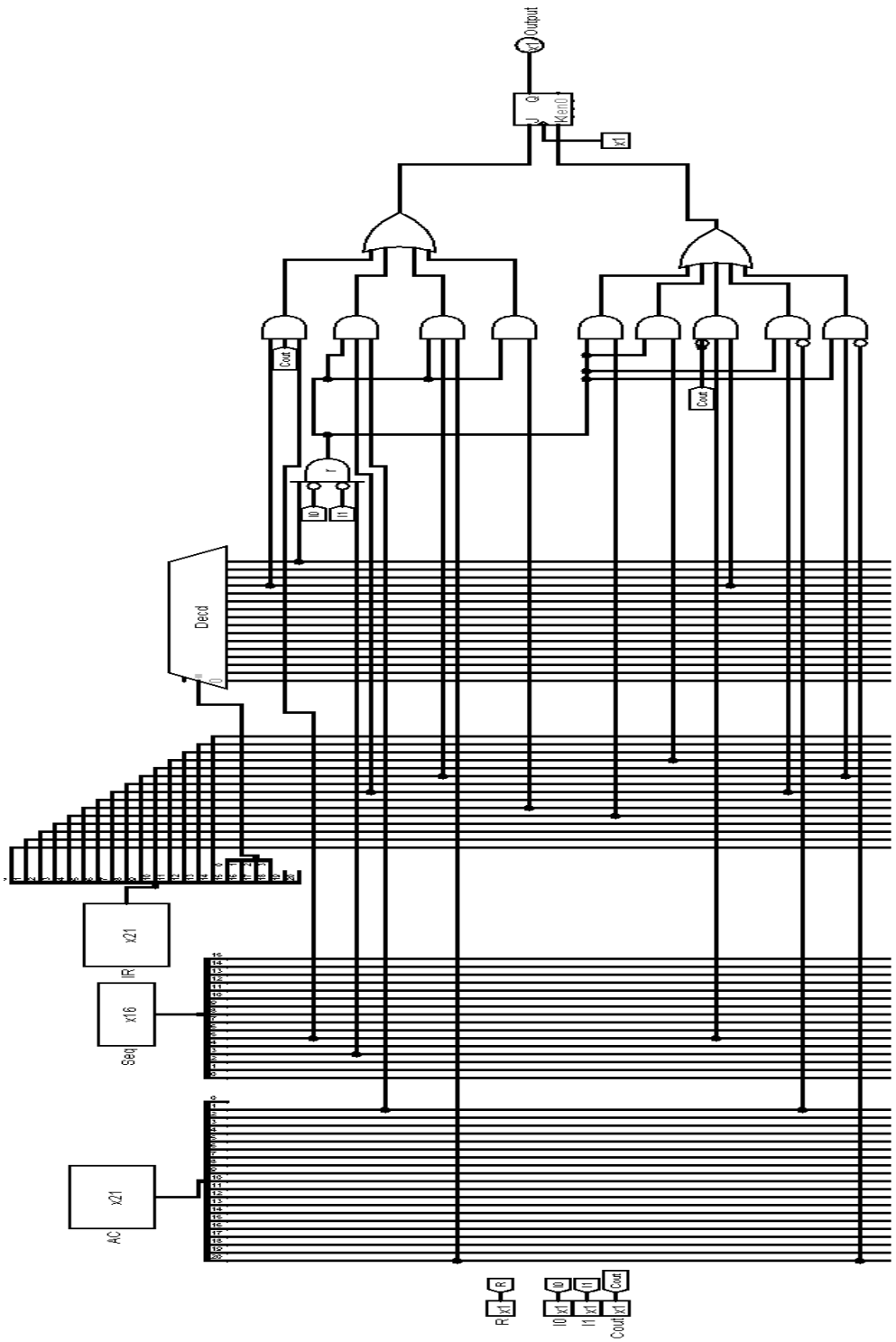

Figure 5.2-5 Design of E

5.2.6 Design of S

Table 5.2-6 Design of S

Control Function	Operation	
rB_{10}	S←0	Clear

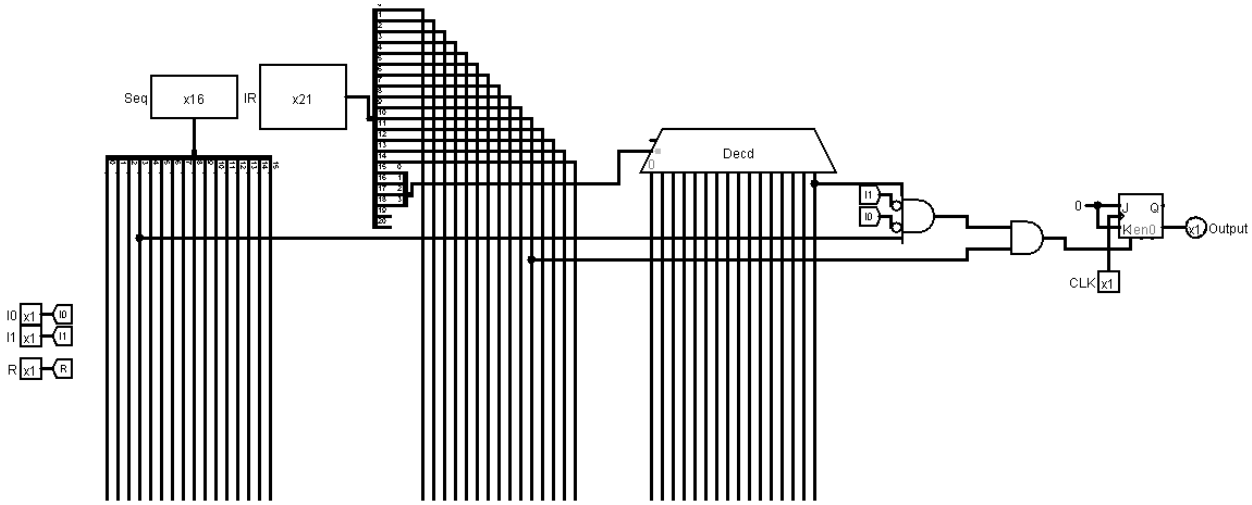

Figure 5.2-6 Design of S

5.2.7 Design of I_0

Table 5.2-7 of I0

Control Function	Operation	
$R'T_2$:	I0←IR (19)	

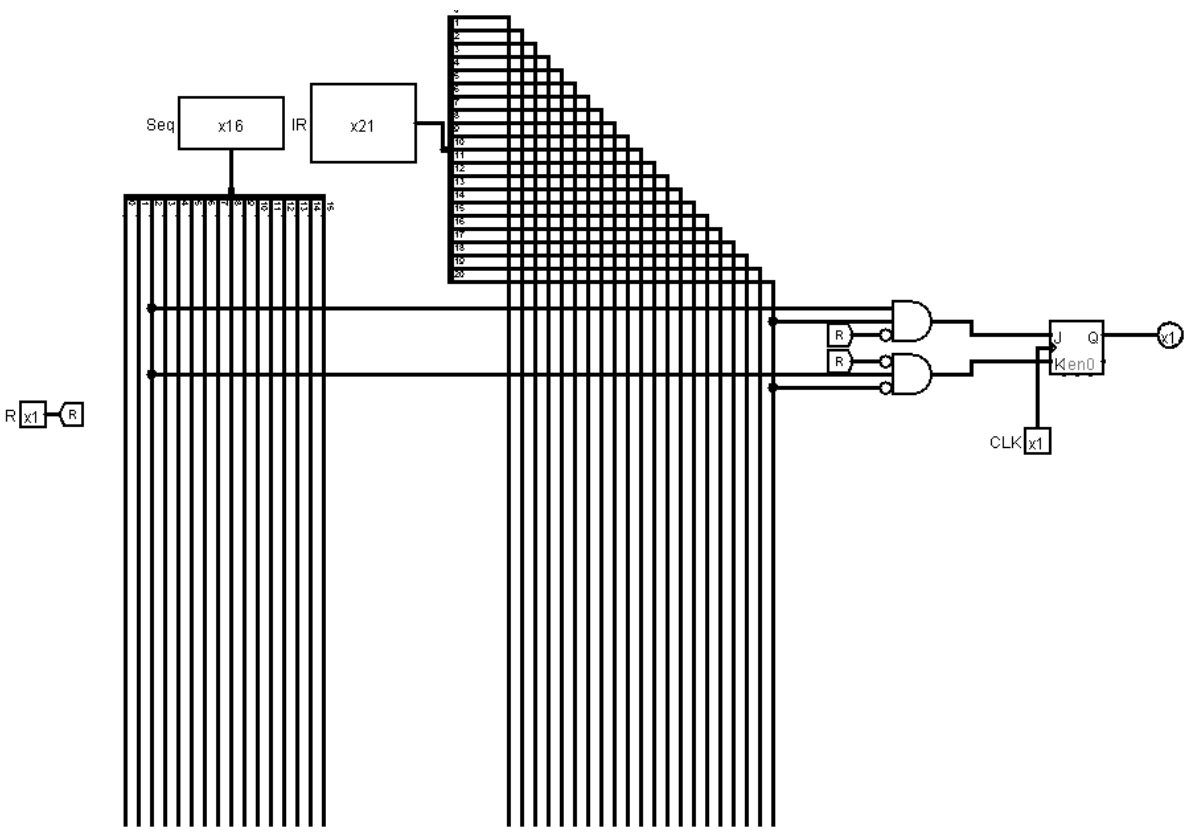

Figure 5.2-7 Design of I_0

5.2.8 Design of I1

Table 5.2-8 Design of I_1

Control Function	Operation	
$R'T_2$:	I1←IR (20)	

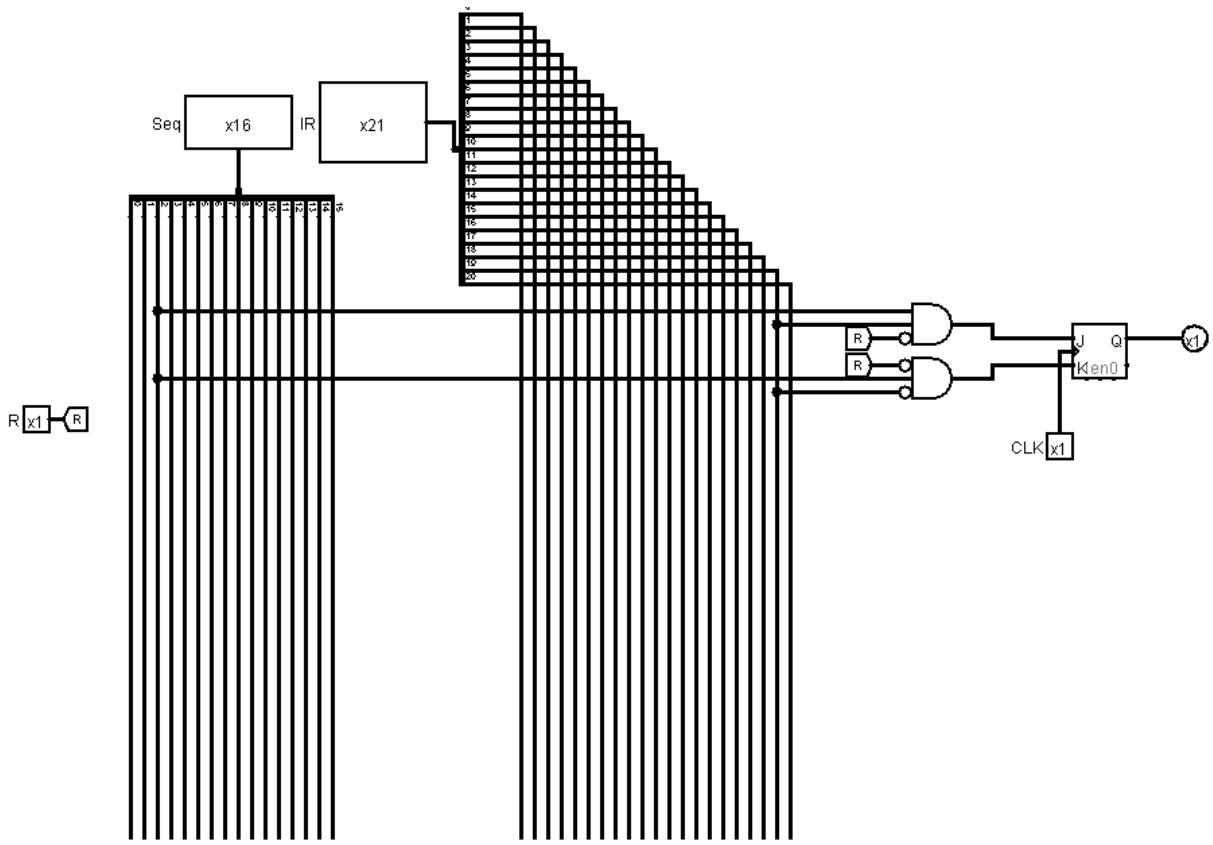

Figure 5.2-8 Design of I_1

5.3 Design of Memory

Table 5.3-1 Design of Memory

Control Function	Operation	
$R'T_1$:	IR←M[AR]	Read
$D_{15}'. T_3.I_0'. I_1'$:	DR←M[AR]	Read
$D_{15}'. T_3.I_0'. I_1$:	AR←M[AR]	Read
$D_{15}'. T_4.I_0'. I_1$:	DR←M[AR]	Read
RT_1:	M[AR] ←TR	Write

D_7T_5:	M[AR]←AC	Write
$D_{13}T_5$:	M[AR]←PC	Write
$D_{10}T_6$:	M[AR]←DR	Write
$D_{12}T_6$:	M[AR]←DR	Write
Read Memory	$R'T_1 + D_{15}'.\ T_3.I_0'.\ I_1' + D_{15}'.\ T_3.I_0'.I_1 + D_{15}'.T_4.I_0'.I_1$	
Write Memory	$RT_1 + D_7T_5 + D_{13}T_5 + D_{10}T_6 + D_{12}T_6$	

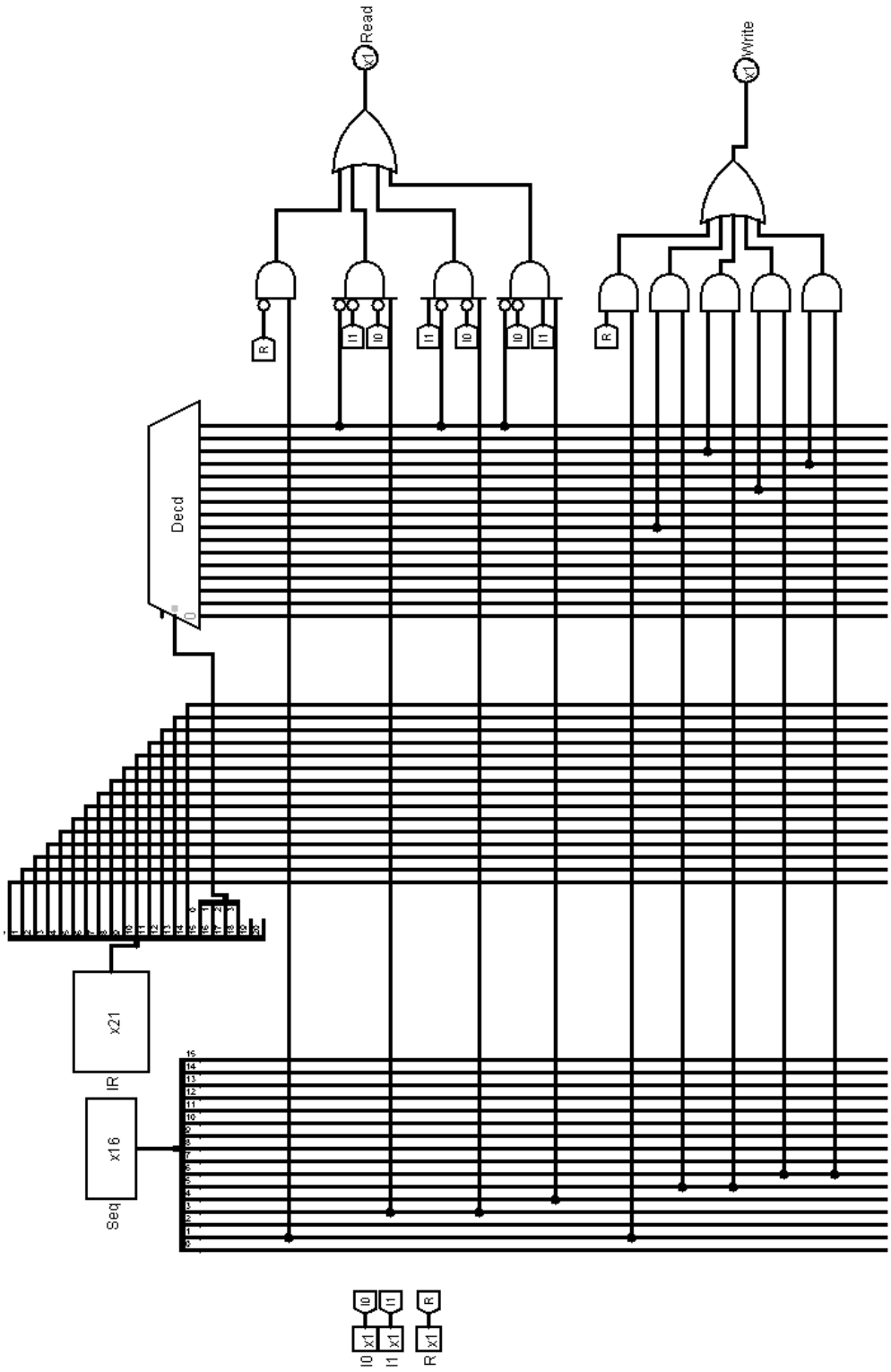

Figure 5.3-1 Design of Memory

5.4 Design of ALU

The term "ALU" refers to the Arithmetic and Logic Unit, a fundamental component within a computer system responsible for executing arithmetic and logical operations. The ALU manages a range of arithmetic tasks, such as addition, subtraction, shifting left (SHL), shifting right (SHR), among others. Additionally, it performs various logical operations, including AND and XOR operations, playing a critical role in processing and manipulating data within the computer.

Table 5.4-1 Design of ALU

Control Function	**Operation**	
D_1T_5:	AC←AC^DR	AND with DR
$D_{12}T_5$:	AC←AC+DR	ADD with DR
$D_{11}T_5$:	AC←DR	Transfer DR
D_8T_6:	AC←DR	Transfer DR
pB_{12}:	AC (0-7) ←INPR	INPR transfer
D_0T_5:	AC←ACvDR	OR with DR
D_6T_5:	AC←AC⊕DR	XOR with DR
D_2T_{60}:	AC←TR	Transfer TR
$D_{10}T_5$:	AC← (AC^DR)’	NAND with DR
D_9T_5:	AC← (ACvDR)’	NOR with DR
$D_{14}T_5$:	AC←AC+DR’+1	SUB with DR
D_2T_5:	AC←AC⊕DR	XOR with DR

rB_{12}:	AC←0	CLEAR AC
rB_5:	AC←AC+1	Increment AC
rB_3:	AC←AC-1	decrement AC
rB_6:	AC←AC’	Complement AC
rB_7:	AC←shr AC, AC (20) ←E	SHR AC
rB_9:	AC←shl AC, AC (0) ←E	SHL AC

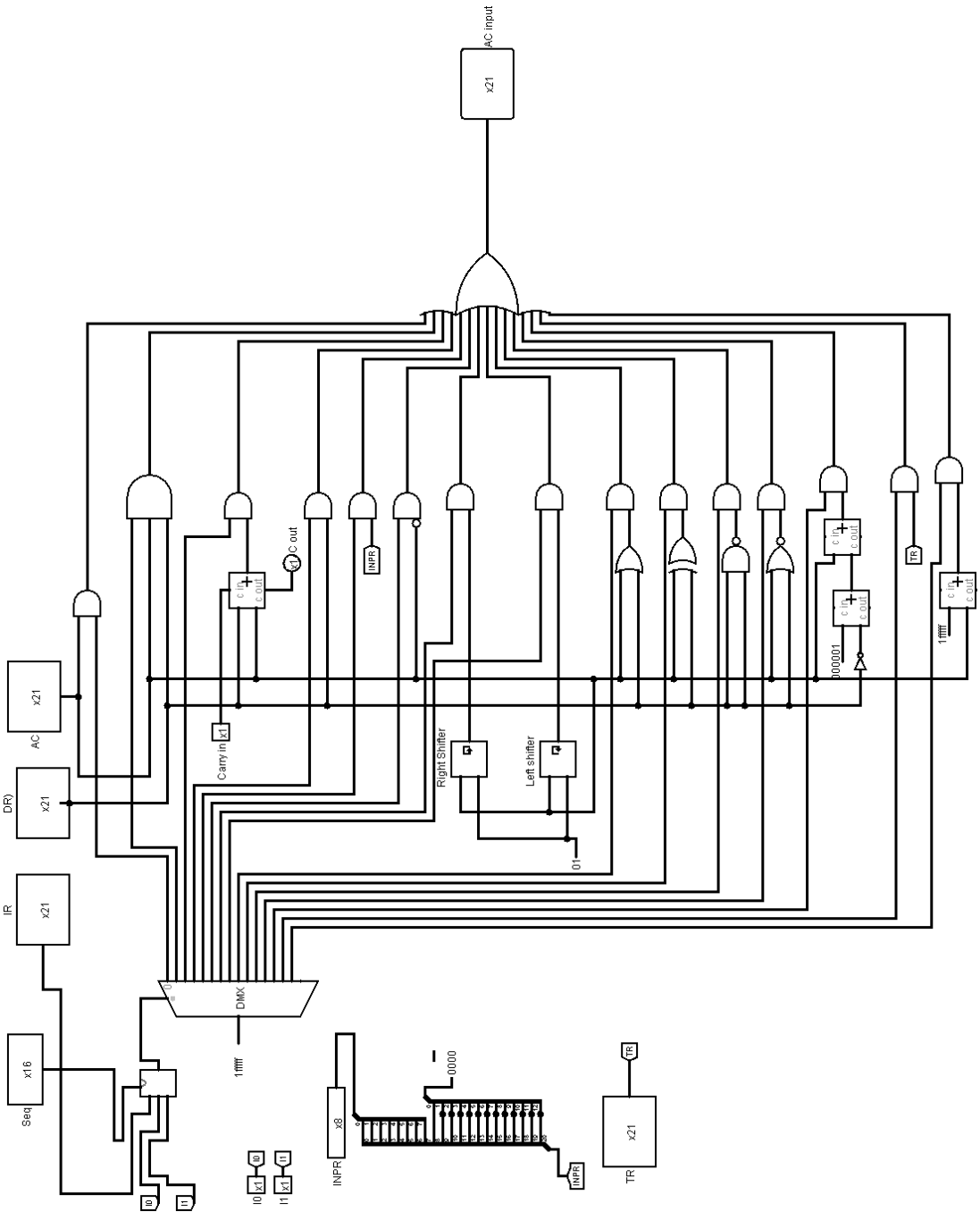

Figure 5.4-1 Design of ALU

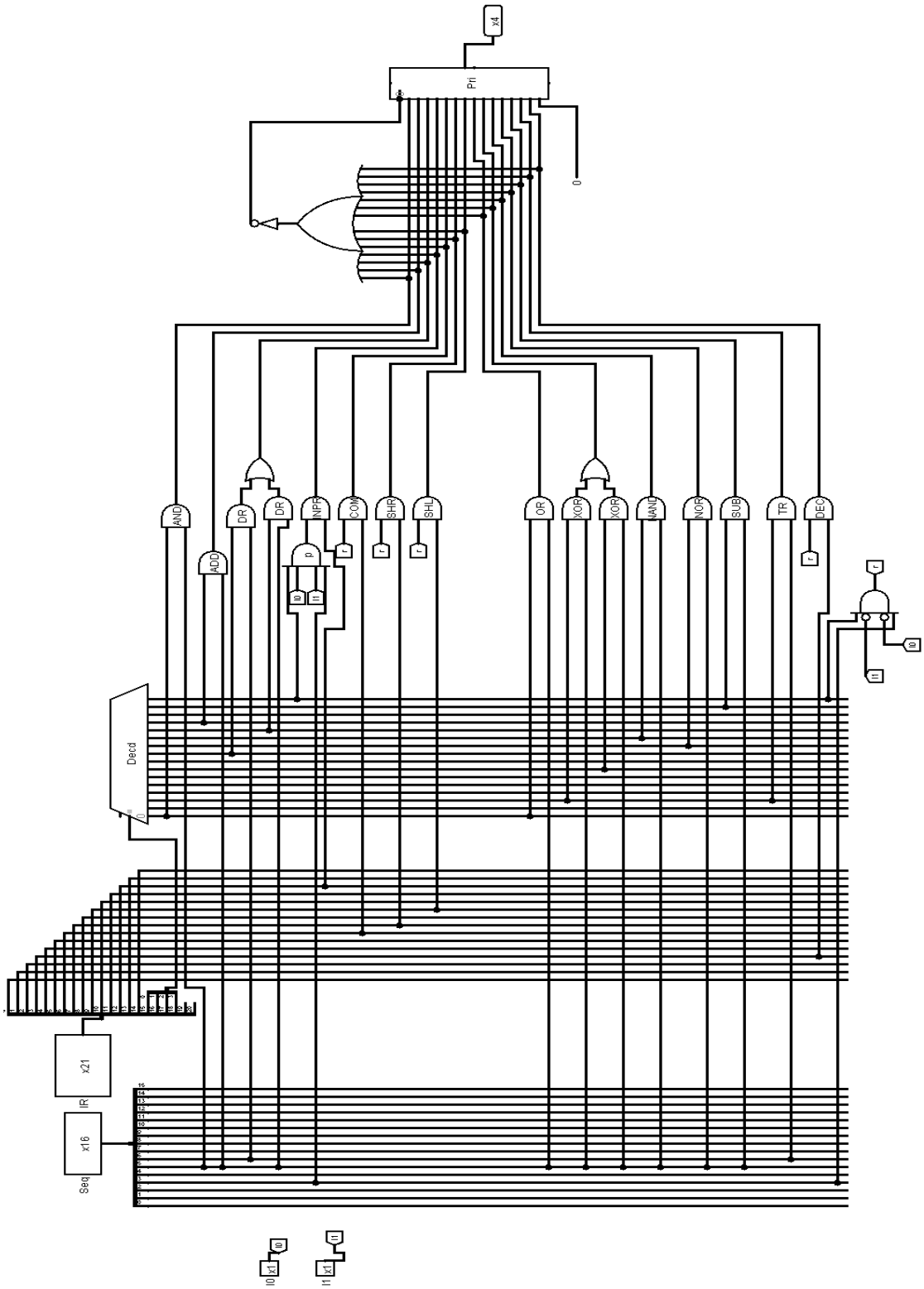

Figure 5.4-2 ALU Controller

5.5 Design of Common Bus Control

Table 5.5-1 Design of Common Bus Control

Design to Bus from	Control Function	
AR	D_{15}'. $T_3.I_0.I_1$ ':	DR←AR
	D_4T_5:	PC←AR
	$D_{13}T_6$:	PC←AR
PC	R'T_0:	AR←PC
	RT_0:	TR←PC
	$D_{13}T_5$:	M[AR]←PC
DR	$D_{11}T_5$:	AC←DR
	D_8T_6:	AC←DR
	D_3T_6:	M[AR]←DR
	D_5T_6:	M[AR]←DR
AC	D_2T_5:	TR←AC
	D_7T_5:	M[AR]←AC
	D_8T_5:	TR←AC
	pB_6:	OUTR← AC (0-7)
IR	R'T_2:	I_0←IR (20), I_1←IR (19)
TR	RT_1:	M[AR] ←TR
	D_2T_6:	AC←TR

	D_8T_7:	DR←TR
Memory	$R'T_1$:	IR←M[AR]
	$D_{15}'. T_3.I_0'. I_1'$:	DR←M[AR]
	$D_{15}'. T_3.I_0'. I_1$:	AR←M[AR]
	$D_{15}'. T_4.I_0'. I_1$:	DR←M[AR]

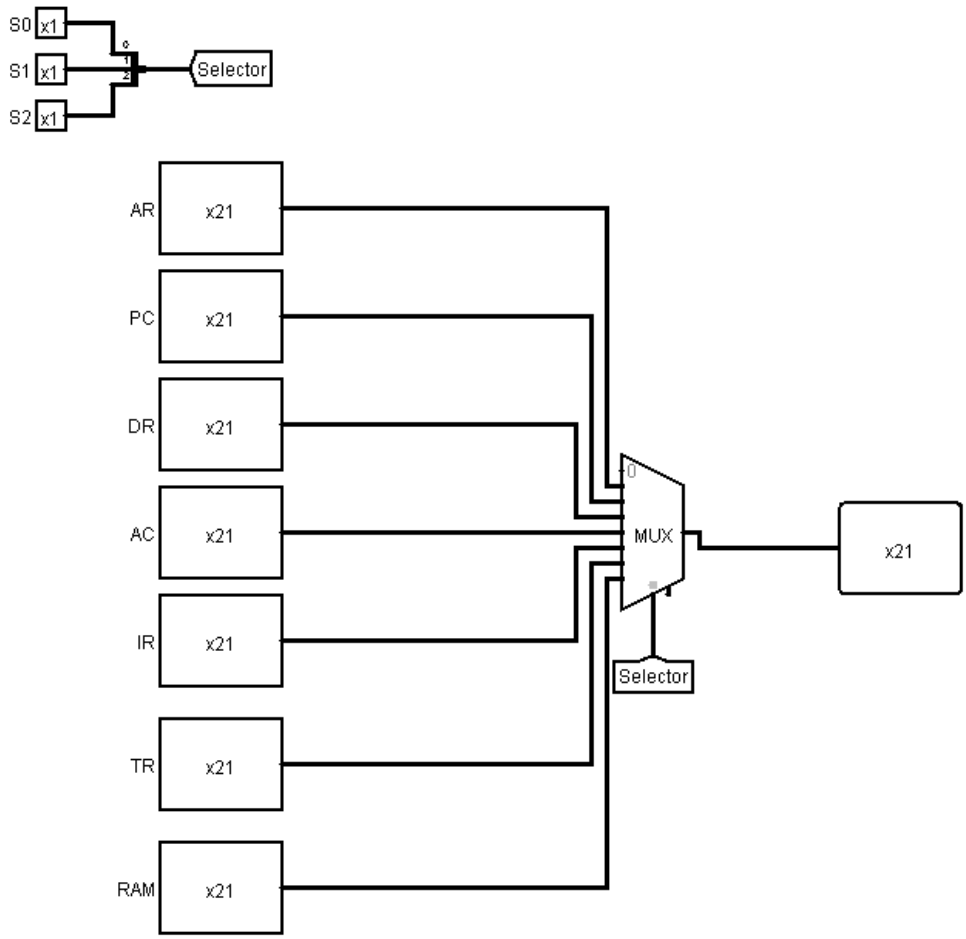

Figure 5.5-1 Design of BUS

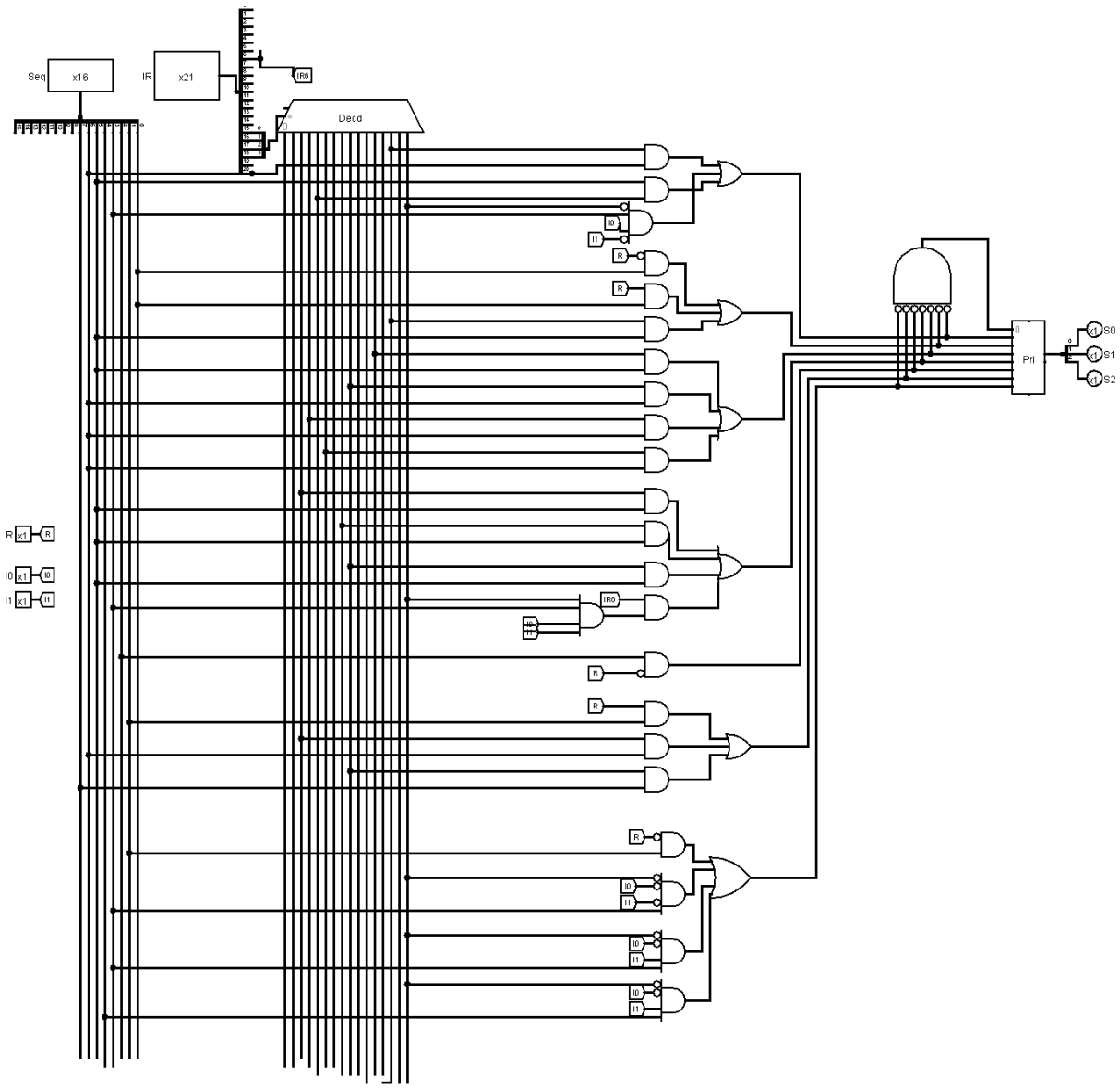

Figure 5.5-2 Control of Bus

5.6 LRN Basic Computer Diagram

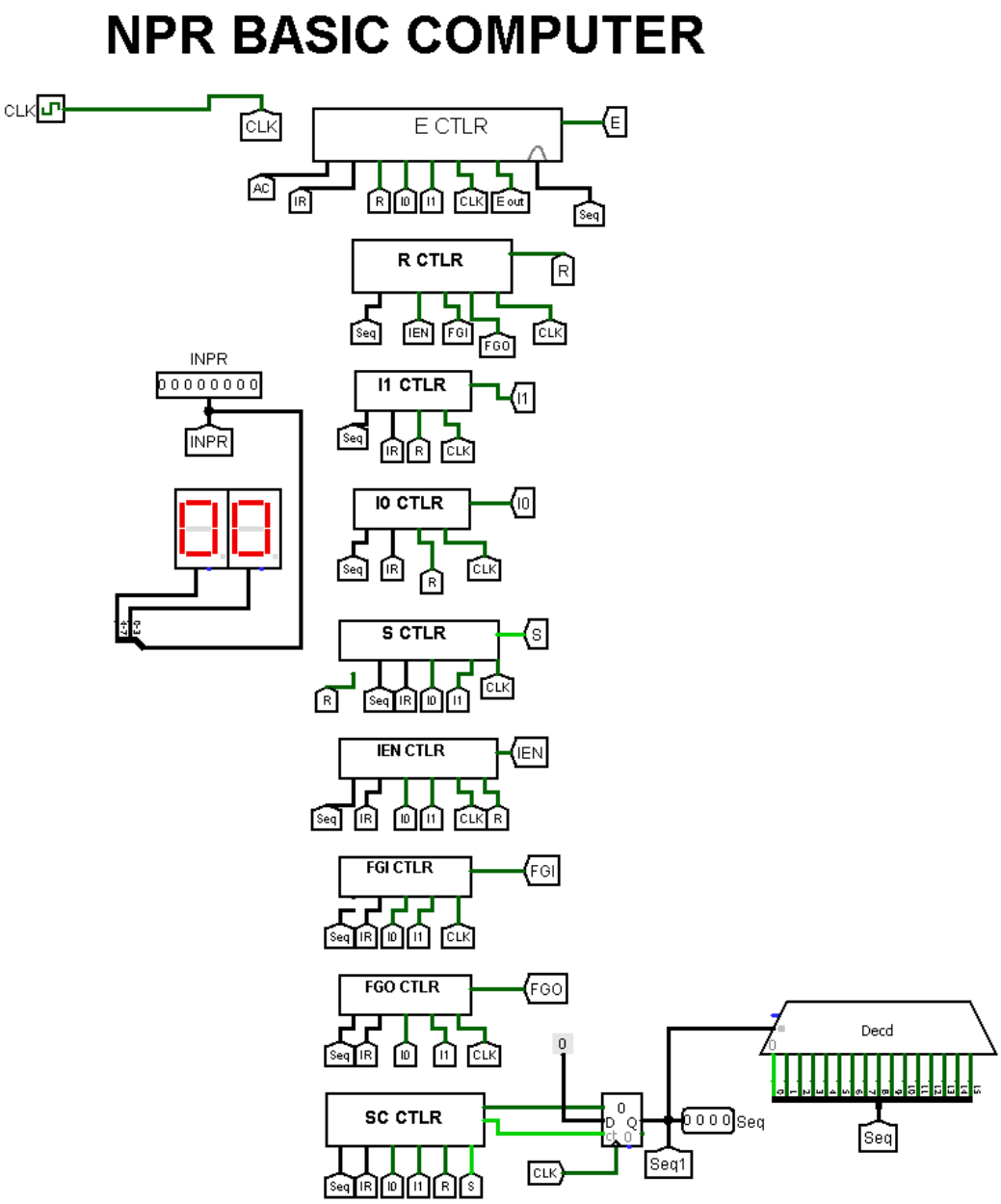

Figure 5.6-1 NPR Basic Computer-1

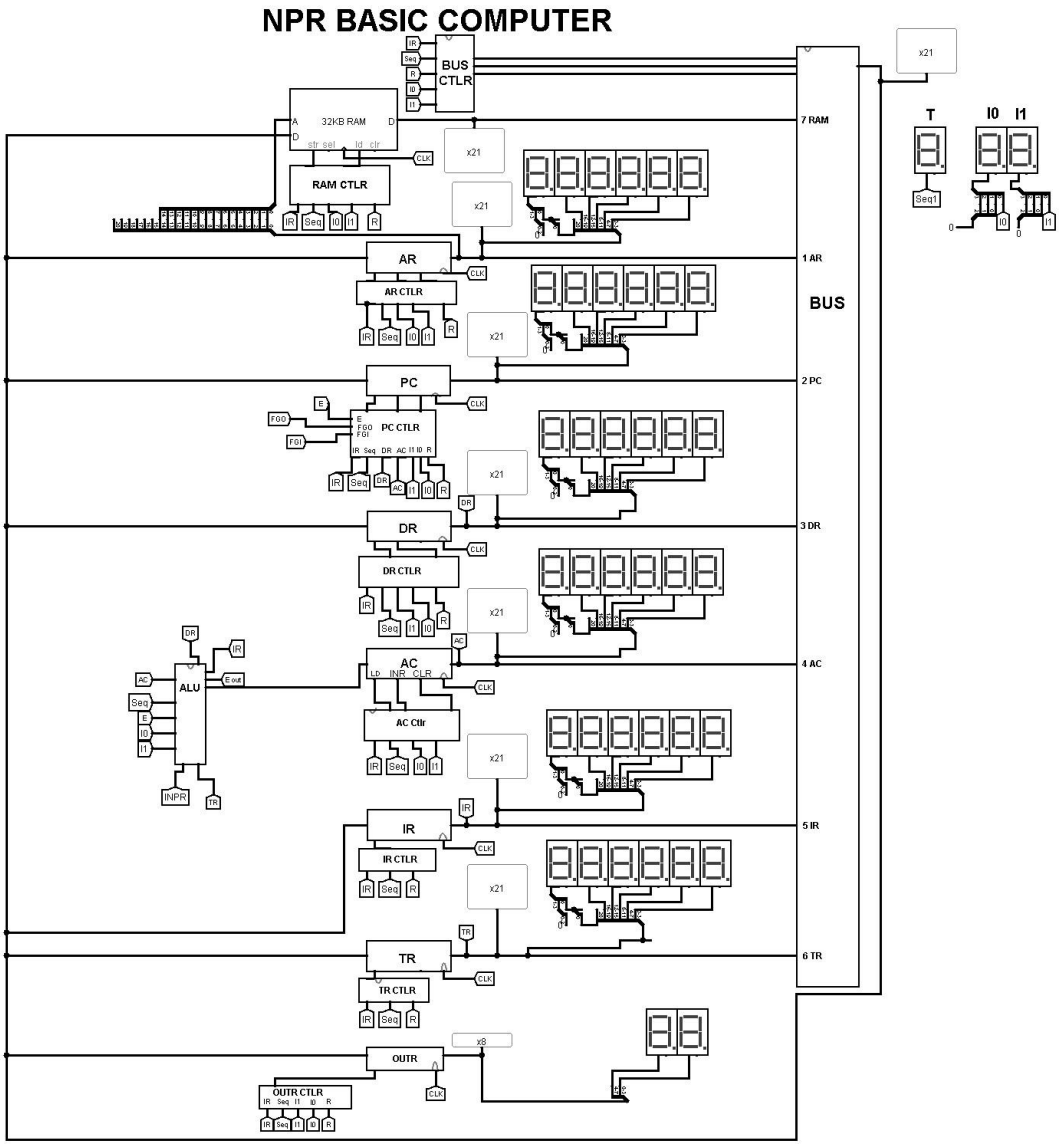

Figure 5.6-2 NPR Basic Computer-2

Chapter 6 Conclusion

Hence a 32K*21 bit “NPR 21-bit Computer” was designed and successfully simulated. The computer is capable of handling basic commands and calculations. It was designed using basic switches, registers, memory, and flip-flops. This is the implementation of Computer Architecture and Design. The implementation was done in Logisim application which helped in the development and understanding of the computer.