UIET CSJM UNIVERSITY,KANPUR

Computer Organisation

Lab Assignment (CSE-S205)
2k19

SUBMITTED BY:		SUBMITTED TO:
NAME	ROLL NO.	
ANKIT SINGH	11	DR.ALOK KUMAR
DEV PRATAP SINGH	17	
SHRADDHA CHOUDHARY	53	
SHREYA SACHAN	54	

COMPUTER1

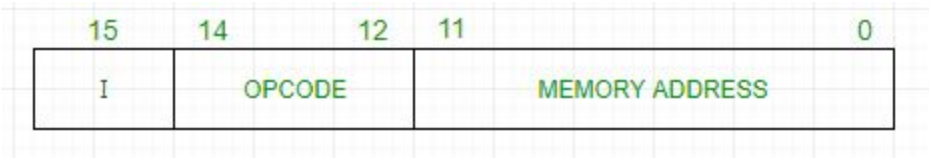
Computer1 is very small compared to commercial computers, It has the advantage of being simple enough so we can demonstrate the design process without too many complications.

Memory size:-4096×16

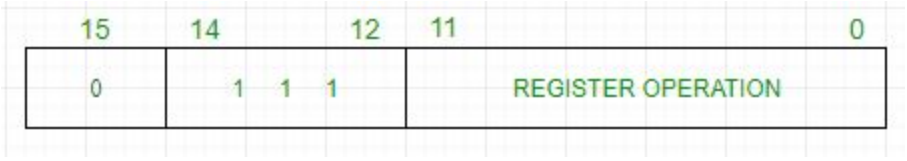
Instruction Code

The basic computer has a 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

1).Memory Reference – These instructions refer to memory address as an operand. The other operand is always an accumulator. Specifies 12-bit address, 3-bit opcode (other than 111), and 1-bit addressing mode for direct and indirect addressing.



2).Register Reference – These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.



3).Input/Output – These instructions are for communication between the computer and outside the environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.

Computer Registers

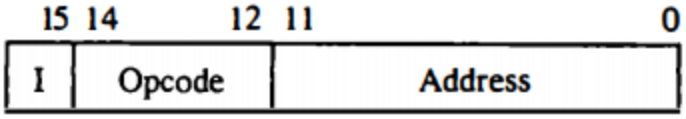
Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction from a specific address in memory and executes it. It then continues by reading the next instruction in sequence and executes it, and so on.

In Computer Architecture, the Registers are very fast computer memory which is used to execute programs and operations efficiently. This does this by giving access to commonly used values, i.e., the values which are in the point of operation/execution at that time. So, for this purpose, there are several different classes of CPU registers that work in coordination with the computer memory to run operations efficiently.

- **Accumulator:** This is the most frequently used register used to store data taken from memory. It is indifferent numbers in different microprocessors.
- **Memory Address Registers (MAR):** It holds the address of the location to be accessed from memory. MAR and MDR (Memory Data Register) together facilitate the communication of the CPU and the main memory.
- **Memory Data Registers (MDR):** It contains data to be written into or to be read out from the addressed location.
- **General Purpose Registers:** These are numbered as R0, R1, R2….Rn-1, and used to store temporary data during any ongoing operation. Its content can be accessed by assembly programming. Modern CPU architectures tend to use more GPR so that register-to-register addressing can be used more, which is comparatively faster than other addressing modes.
- **Program Counter (PC):** Program Counter (PC) is used to keep the track of the execution of the program. It contains the memory address of the next instruction to be fetched. PC points to the address of the next instruction to be fetched from the main memory when the previous instruction has been successfully completed. Program Counter (PC) also functions to count the number of instructions. The incrementation of PC depends on the type of architecture being used. If we are using a 32-bit architecture, the PC gets incremented by 4 every time to fetch the next instruction.
- **Instruction Register (IR):** The IR holds the instruction which is just about to be executed. The instruction from the PC is fetched and stored in IR. As soon as the instruction in place in IR, the CPU starts executing the instruction, and the PC points to the next instruction to be executed.

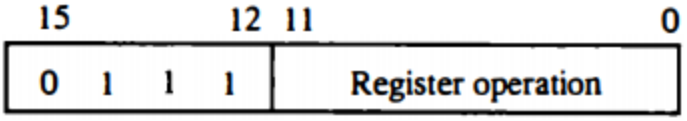
Computer Instruction

Figure Basic computer instruction formats.



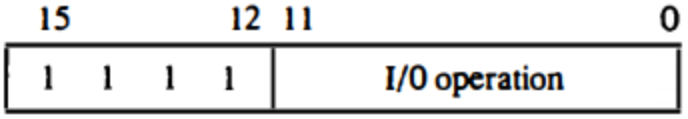
(Opcode = 000 through 110)

(a) Memory – reference instruction



(Opcode = 111, $I = 0$)

(b) Register – reference instruction



(Opcode = 111, $I = 1$)

(c) Input – output instruction

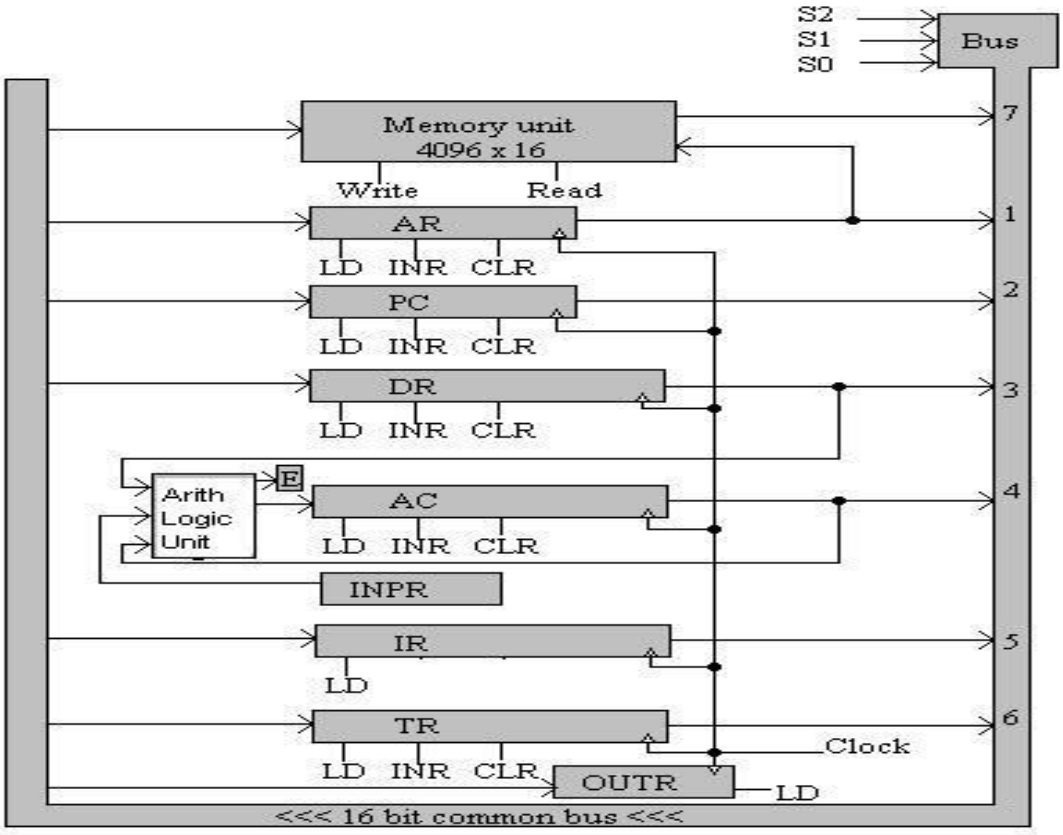
Basic Computer Instructions

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load AC from memory
STA	3xxx	Bxxx	Store content of AC into memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instr. if AC is positive
SNA	7008		Skip next instr. if AC is negative
SZA	7004		Skip next instr. if AC is zero
SZE	7002		Skip next instr. if E is zero
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Timing and Control

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 clock pulses do not change the state of a 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 processor registers, and micro-operations for the accumulator.

An instruction read from memory is placed in the instruction register (IR).position of this register in the common bus system.



The instruction register is shown again in Fig. 5.6, where it is divided into three parts:

1. the 1 bit,
2. the operation code, and
3. bits 0 through 11.

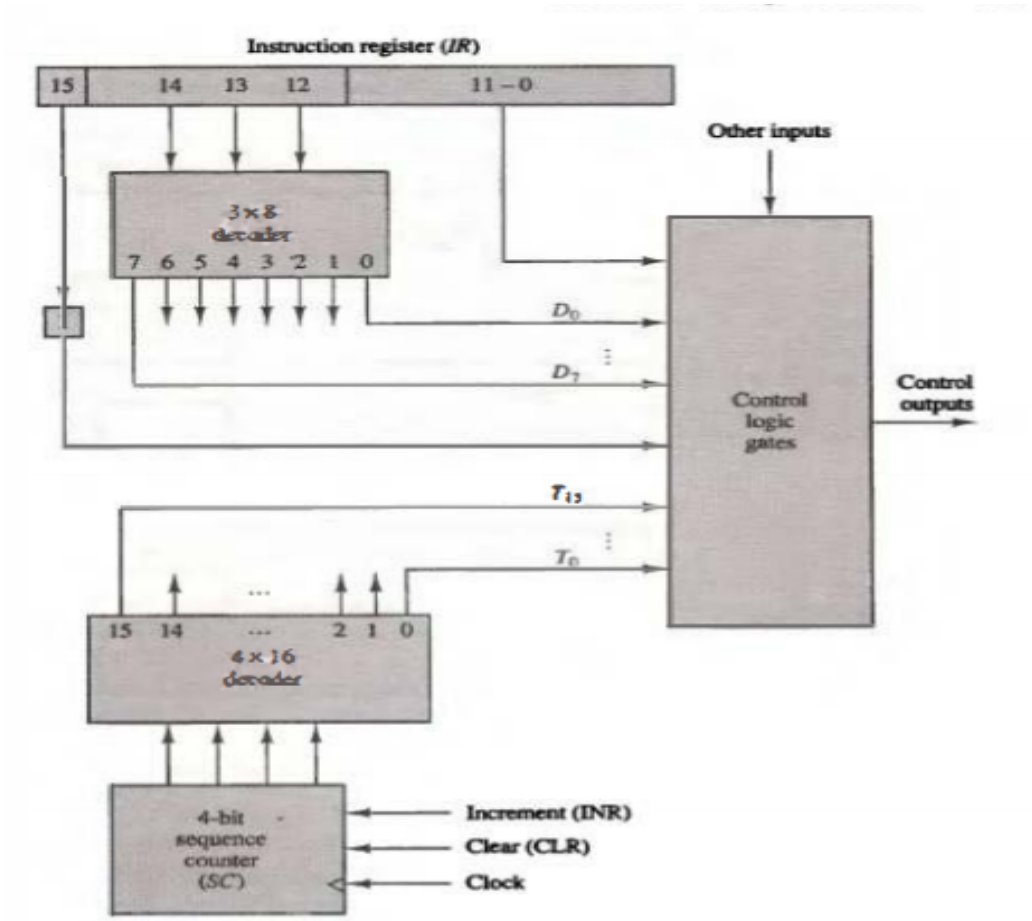


Figure 5-6 Control unit of basic computer.

The operation code in bits 12 through 14 is decoded with a 3 x 8 decoder. The eight outputs of the decoder are designated by the symbols D_0 through D_7. The subscripted decimal number is equivalent to the binary value of the corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits 0 through 11 are applied to the control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The outputs of the counter are decoded into 16 timing signals T_0 through T_{15}.

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.

As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3, and T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This is expressed symbolically by the statement D_3T_4: SC <— 0.

Logisim

• What is Logisim?

Logisim is a digital design tool for educational purposes designed by Carl Burch of Hendrix University. Logisim can be used for the logical design of circuits and is the tool you will be using for the design projects.

• Where to get Logisim?

You can find Logisim at http://ozark.hendrix.edu/~burch/logisim/index.html, or by Googling "Logisim".

Environment Layout

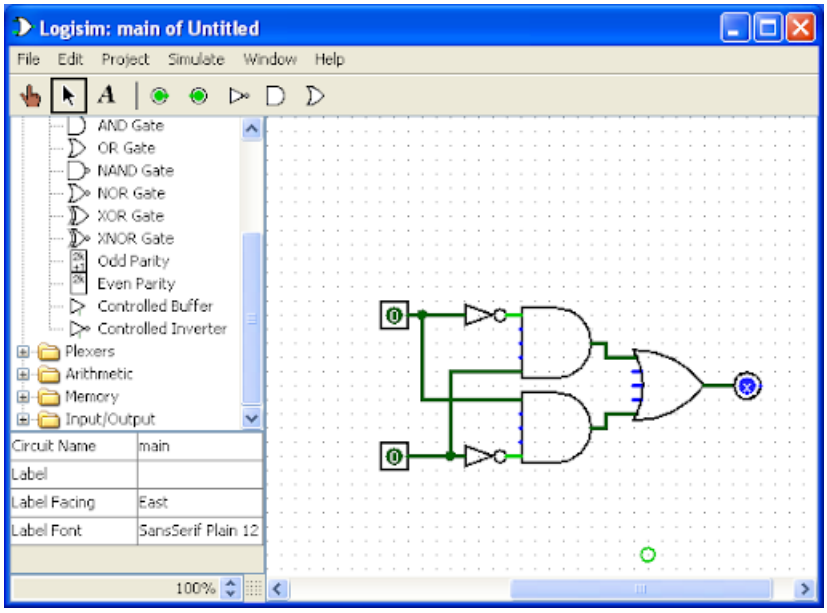
• Toolbar: The toolbar contains shortcuts to several commonly used items 1 – The poke tool (shaped like a hand) is used to alter input pins. – The input pin (green circle surrounded by a box) is used to send a signal through a wire. When placing the input on the canvas it initializes to 1-bit. This number of bits can be increased in the Attribute Table. – The output pin (green circle in a circle) is used to observe output from a gate. The output pin toggles in real-time as long as the simulation is enabled from the menu bar Simulate > Simulate enabled

• Explorer Pane: The list of wiring, gates, multiplexers, etc... that are available for digital design in Logisim. Please note not all items are allowed to be used in every project.

• Attribute Table: Gives detailed attributes of digital design components (e.g., AND, OR, XOR gates). The attribute table allows you to alter the number of inputs/outputs that a digital design component.

• Canvas: The canvas is the area for you to create your digital circuits. In the canvas area, you may simulate your circuits while designing in real-time.

Design



An XNOR gate created from AND, OR, and NOT gates.

Simulation

The types of simulation supported by Logisim are:

• Simulate Enabled: Let the circuit run based on its inputs. must be enabled for all tick simulations to work as well.

• Step Simulation: This allows the user to simulate a single step at a time. If an input changes in step simulation you must advance the signal through each gate by stepping (ctrl-i).

• Tick Simulation Used to tick a clock (found in Explorer Plane Wiring > Clock). This is vital for stateful circuits (e.g., RAM, flip-flops, etc...) – Tick Once: Tick the clock once (go from high to low or vice versa) – Ticks Enabled: Tick automatically at the rate of tick frequency – Tick Frequency: How often to tick the clock (measured in Hz).

Flip-Flops

Flip-Flops are used to enable circuits. Flip-Flops (FF) are found in the Explorer Plane > Memory . In this example, we select a typical D Flip-Flop (DFF), and show how it is used in concert with a clock. The DFF absorbs the input bit on the rising edge of the clock, which means when the clock transitions from 0 → 1.

There are several inputs on the DFF

- D: The value to input into the DFF on the next rising edge.
- Clock: The clock that controls the DFF
- Q: The output of the DFF.
- Asynchronous reset: pins DFF value to 0 when input is 1, found on the bottom left.
- Enable: When set to 0 the clock input is ignored and the DFF will maintain its current value.
- Asynchronous Set: pins DFF value to 1, while initialized to 1.

Instruction Set Of Computer1

The basic computer has a 16-bit instruction register (IR) which can denote either memory reference or register reference or input-output instruction.

1).Memory Reference – These instructions refer to memory address as an operand. The other operand is always an accumulator. Specifies 12-bit address, 3-bit opcode (other than 111), and 1-bit addressing mode for direct and indirect addressing.

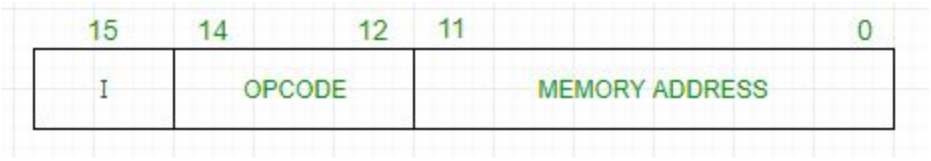
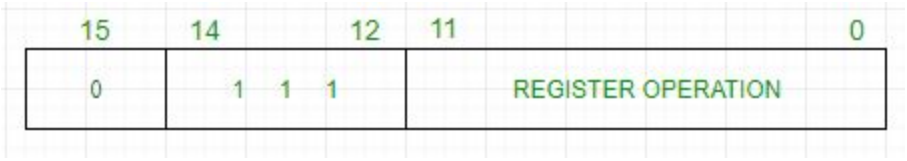


TABLE Memory-Reference Instructions

Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1,$ If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

2).Register Reference – These instructions perform operations on registers rather than memory addresses. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 0 (differentiates it from input/output instructions). The rest 12 bits specify register operation.



REGISTER REFERENCE INSTRUCTIONS

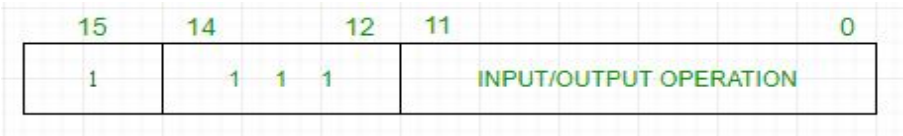
Register Reference Instructions are identified when

- $D_7 = 1$, $I = 0$
- Register Ref. Instr. is specified in $b_0 \sim b_{11}$ of IR
- Execution starts with timing signal T_3

$r = D_7 I' T_3$ => Register Reference Instruction
$B_i = IR(i)$, i=0,1,2,...,11

	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(15) ← E, E ← AC(0)
CIL	rB_6:	AC ← shl AC, AC(0) ← E, E ← AC(15)
INC	rB_5:	AC ← AC + 1
SPA	rB_4:	if (AC(15) = 0) then (PC ← PC+1)
SNA	rB_3:	if (AC(15) = 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)

3).Input/Output – These instructions are for communication between the computer and outside the environment. The IR(14 – 12) is 111 (differentiates it from memory reference) and IR(15) is 1 (differentiates it from register reference instructions). The rest 12 bits specify I/O operation.



Input-Output Instructions

unacademy

$D_7 I T_3 = p$
$IR(i) = B_i$, i = 6, ..., 11

	p:	SC ← 0	Clear SC
INP	pB_{11}:	AC(0-7) ← INPR, FGI ← 0	Input char. to AC
OUT	pB_{10}:	OUTR ← AC(0-7), FGO ← 0	Output char. from AC
SKI	pB_9:	if(FGI = 1) then (PC ← PC + 1)	Skip on input flag
SKO	pB_8:	if(FGO = 1) then (PC ← PC + 1)	Skip on output flag
ION	pB_7:	IEN ← 1	Interrupt enable on
IOF	pB_6:	IEN ← 0	Interrupt enable off

IMPLEMENTATION OF CONTROL UNIT(CU)

INSTRUCTION CYCLE

1)Instruction fetching
2)Instruction decoding
3)Effective address calculation
4)Operator fetching
5)Instruction execution

Instruction fetching(memory to cu):-

T_0 **AR<-PC**
T_1 **AC<-M[AR],PC<-PC+1**

Instruction decoding:-

T_2 **AC(0-7)<-Decode(Ir(12,13,14)),I<-IR(15),AR<-IR(0-11)**

Effective address calculation:-

$D_7'IT_3$ **AR<-M[AR]**

Operator fetching:-

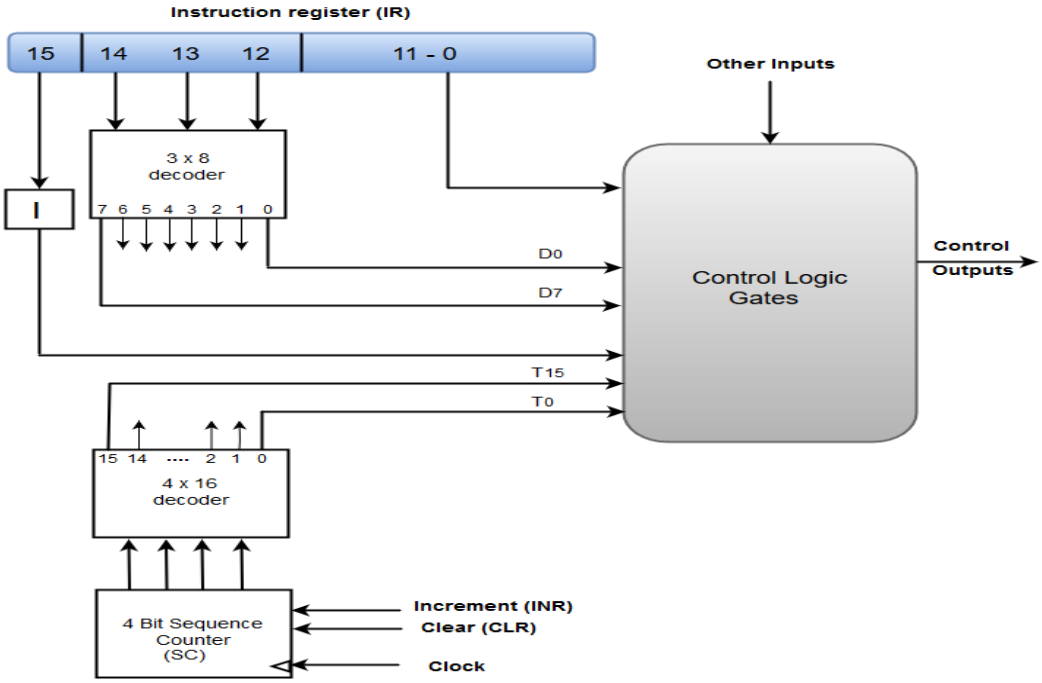
D_0T_4 **DR<-M[AR]**

Instruction execution:-

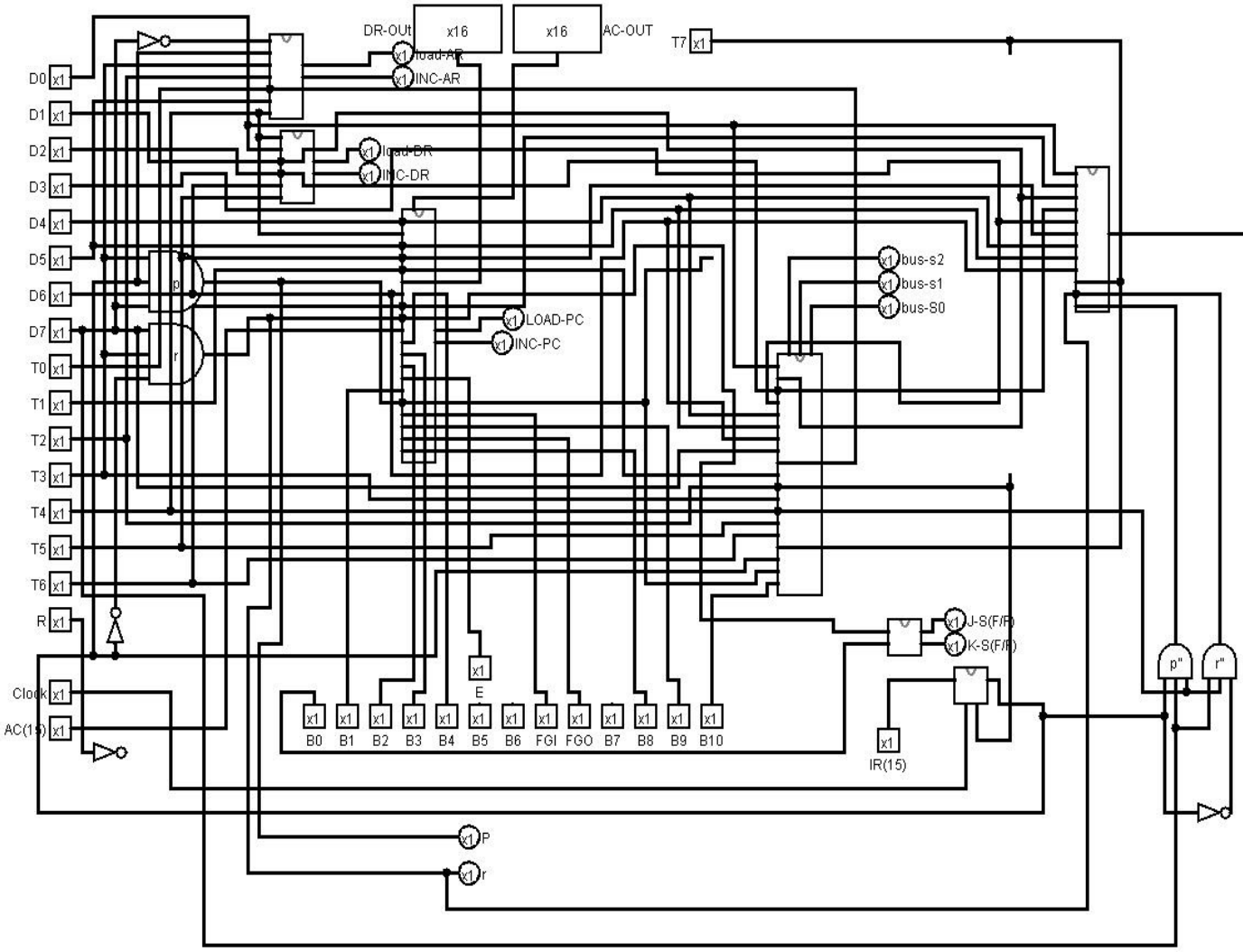
D_0T_5 **AC<-AR^DR,SC<-0**

The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.

Control Unit of a Basic Computer:



- A Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.
- An instruction fetched from the memory unit is placed in the instruction register (IR).
- The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.
- The operation code in bits 12 through 14 are coded with a 3 x 8 decoder.
- The outputs of the decoder are designated by the symbols D0 through D7.
- The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.
- The operation codes from Bits 0 through 11 are applied to the control logic gates.
- The Sequence counter (SC) can count in binary from 0 through 15.



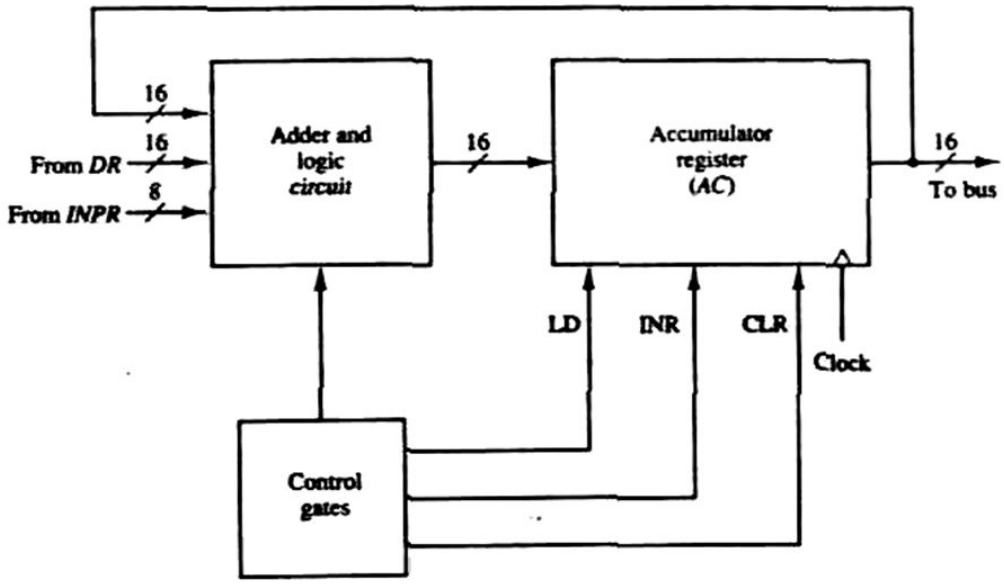
COMPUTER-1 CONTROL UNIT(CU)

IMPLEMENTATION OF ALU

ALU CONSTRUCTION:- The circuits associated with the AC register. The adder and logic circuit has three sets of inputs. One set of 16 inputs comes from the outputs of AC. Another set of 16 inputs comes from the data register DR. A third set of eight inputs comes from the input register INPR. The outputs of the adder and logic circuit provide the data inputs for the register. In addition, it is necessary to include logic gates for controlling the LD, INR, and CLR in the register and for controlling the operation of the adder and logic circuit. In order to design the logic associated with AC, it is necessary to go over the register transfer statements in Table 5-6 and extract all the statements that change the content of AC.

D_OT_5	AC<-AC^DR	AND with DR
D_1T_5	AC<-AC+DR	Add with DR
D_2T_5	AC<-DR	Transfer from DR
$D_7IT_3IR_{11}$	AC(0-7)<-INPR	Transfer from INPR
$D_7I'T_3IR_{11}$	AC<-AC'	Complement
$D_7I'T_3IR_{09}$	AC <- shr AC, AC(15) <-E	Shift right
$D_7I'T_3IR_{07}$	AC <- shl AC, AC(O) <-E	Shift left
$D_7I'T_3IR_{06}$	AC <- 0	Clear
$D_7I'T_3IR_{05}$	AC <-AC + 1	Increment

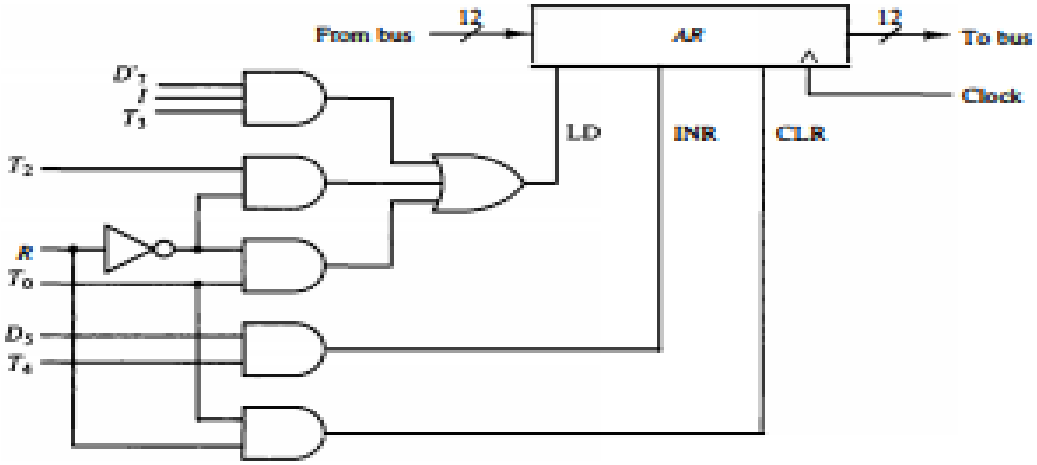
Block Diagram of Design an Accumulator Logic



Circuits associated with AC

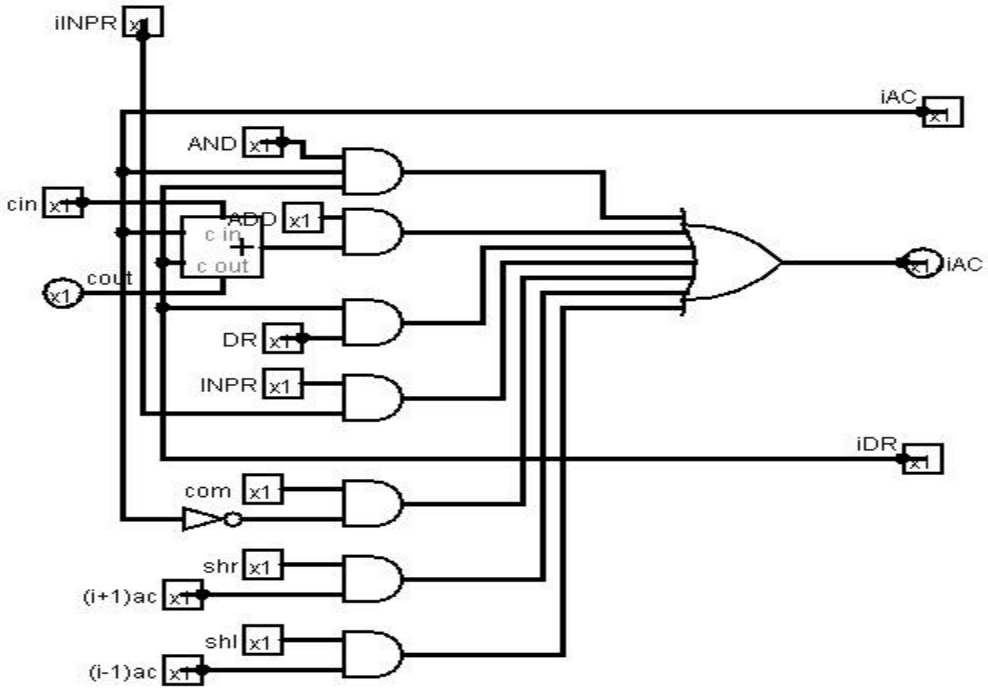
Control of AC Register: The gate structure that controls the LD, INR, and CLR inputs of AC . The gate configuration is derived from the control functions in the list above. The control function for the clear microoperation is rBn, where r = D7I'T3 and Bn = IR(ll). The output of the AND gate that generates this control function is connected to the CLR input of the register. Similarly, the output of the gate that implements the increment microoperation is connected to the INR input of the register. The other seven microoperations are generated in the adder and logic circuit and are loaded into AC at the proper time. The outputs of the gates for each control function is marked with a symbolic name. These outputs are used in the design of the adder and logic circuit.

Figure 5-16 Control gates associated with *AR*



Adder and Logic Circuit : The adder and logic circuit can be subdivided into 16 stages, with each stage corresponding to one bit of AC. The internal construction of the register. Looking back at that figure we note that each stage has a JK flip-flop, two OR gates, and two AND gates. The load (LD) input is connected to the inputs of the AND gates.one such AC register stage (with the OR gates removed). The input is labeled I; and the output AC(i). When the LD input is enabled, the 16 inputs I, for i = 0, 1, 2, ... , 15 are transferred to AC (0-15). One stage of the adder and logic circuit consists of seven AND gates, one OR gate and a full-adder (FA). The inputs of the gates with symbolic names come from the outputs of gates marked with the same symbolic name. For example, the input marked ADD is connected to the output marked ADD .The AND operation is achieved by ANDing AC(i) with the corresponding bit in the data register DR(i). The ADD operation is obtained using a binary adder similar to the one. One stage of the adder uses a full-adder INPR with the corresponding input and output carries. The transfer from to AC is only for bits 0 through 7. The complement microoperation is obtained by inverting the bit value in AC. The shift-right operation transfers the bit from AC(i + 1), and the shift-left operation transfers the bit from AC(i - 1). The complete adder and logic circuit consists of 16 stages connected together.

Computer 1 ALU:-



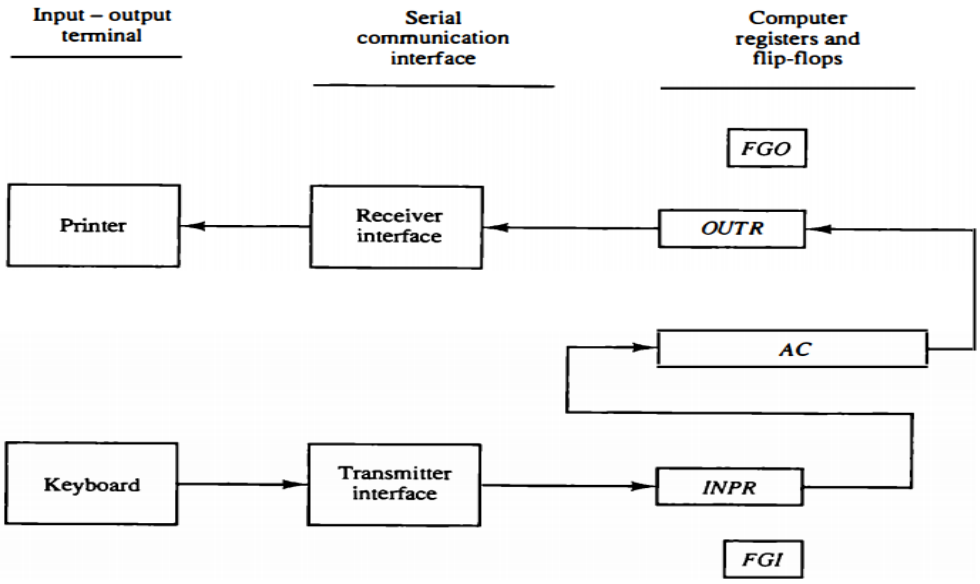
IMPLEMENTATION OF I/O SYSTEM

A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device.

Input-Output Configuration:-

The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code. The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register OUTR. These two registers communicate with a communication interface serially and with the AC in parallel. The input-output configuration is below. The transmitter interface receives serial information from the keyboard and transmits it to INPR. The receiver interface receives information from OUTR and sends it to the printer serially.

Figure Input-output configuration.



Input-Output Instructions

Input and output instructions are needed for transferring information to and from the AC register, checking the flag bits, and for controlling the interrupt facility. Input-output instructions have an operation code 1111 and are recognized by the control when D7 = 1 and I = 1. The remaining bits of the instruction specifies the particular operation. The control functions and micro operations for the input-output instructions. These instructions are executed with the clock transition associated with timing signal T, Each control function needs a Boolean relation D7IT3, which we designate for convenience by the symbol p. The control function is distinguished by one of the bits in IR(6-1 1). By assigning the symbol B, to the bit I of IR, all control functions can be denoted by pB; for i = 6 though II.

List of input-output instructions

$D_7IT_3 = p$ (common to all input–output instructions)
$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0–7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0–7)$, $FGO \leftarrow 0$	Output character
SKI	pB_9:	If $(FGI = 1)$ then $(PC \leftarrow PC + 1)$	Skip on input flag
SKO	pB_8:	If $(FGO = 1)$ then $(PC \leftarrow PC + 1)$	Skip on output flag
ION	pB_7:	$IEN \leftarrow 1$	Interrupt enable on
IOF	pB_6:	$IEN \leftarrow 0$	Interrupt enable off

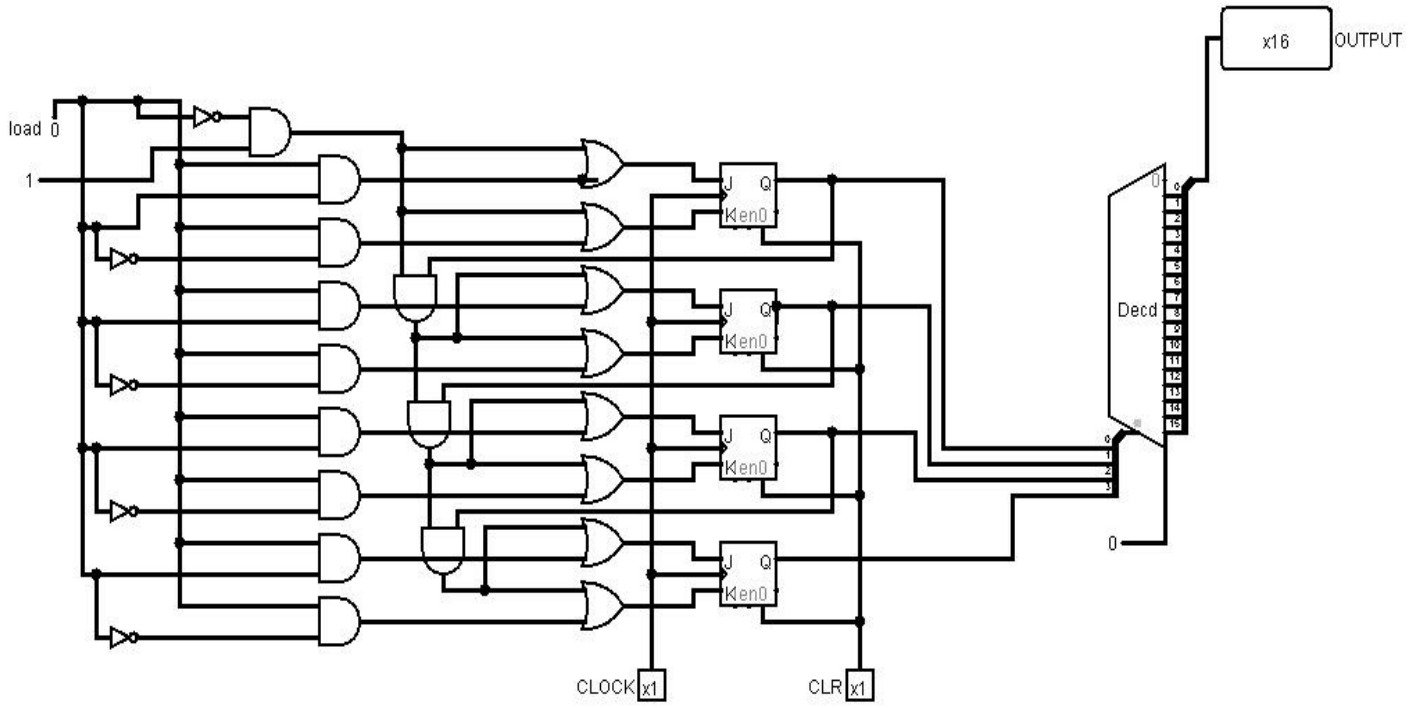
The sequence counter SC is cleared to 0 when p = D71T3 = I. The INP instruction transfers the input information from INPR into the eight low-order bits of AC and also clears the input flag to 0. The OUT instruction transfers the eight least significant bits of AC into the output register OUTR and clears the output flag to 0. The next two instructions check the status of the flags and cause a skip of the next instruction if the flag is I. The instruction that is skipped will normally be a branch instruction to return and check the flag again. The branch instruction is not skipped if the flag is 0. If the flag is I, the branch instruction is skipped and an input or output instruction is executed. The last two instructions set and clear an interrupt enable flip flop IEN. The purpose of IEN is explained in conjunction with the interrupt operation.

SYSTEM TESTING

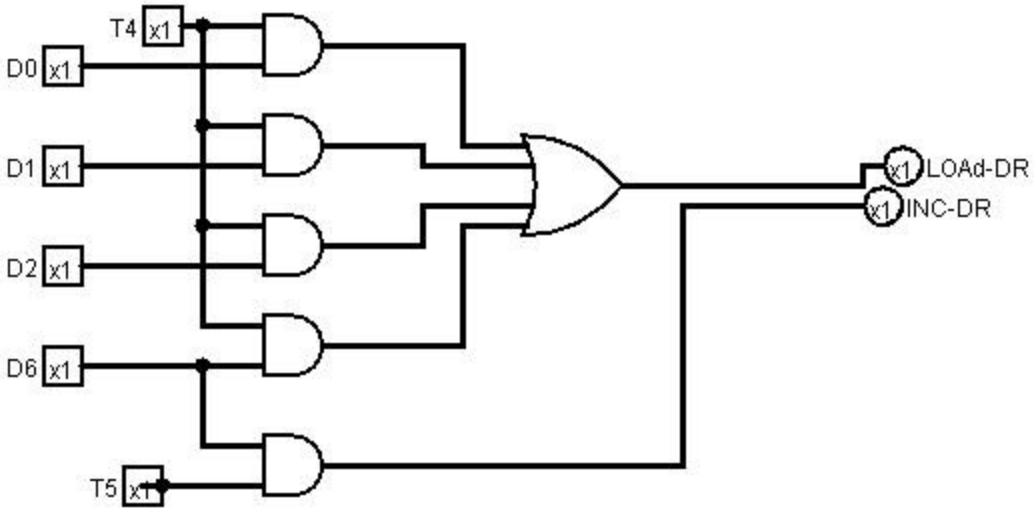
OPERATION	DIFFERENT TIME FRAME	MEMORY	AR	PC	DR	AC	IR	TR	OUTR	INPR	SC	I	S	E	R	IEN	FGI	FGO
LDA(load 5 in AC)	t_0	0	0	0	0	0	0	0	0	0	0	0			0	0	0	0
	t_0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
	t_1	200a	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
	t_2	200a	0	1	0	0	200a	0	0	0	2	0	0	0	0	0	0	1
	t_3	005	10	1	0	0	200a	0	0	0	3	0	0	0	0	0	0	1
	t_4	005	10	1	0	0	200a	0	0	0	4	0	0	0	0	0	0	1
	t_5	005	10	1	5	0	200a	0	0	0	5	0	0	0	0	0	0	1
	t_6	005	10	1	5	5	200a	0	0	0	6	0	0	0	0	0	0	1
ADD(add 6 to content of AC)	t_0	005	10	1	5	5	200a	0	0	0	0	0	0	0	0	0	0	1
	t_1	100b	1	1	5	5	200a	0	0	0	1	0	0	0	0	0	0	1
	t_2	100b	1	2	5	5	100b	0	0	0	2	0	0	0	0	0	0	1
	t_3	006	11	2	5	5	100b	0	0	0	3	0	0	0	0	0	0	1
	t_4	006	11	2	5	5	100b	0	0	0	4	0	0	0	0	0	0	1
	t_5	006	11	2	6	5	100b	0	0	0	5	0	0	0	0	0	0	1
	t_6	006	11	2	6	11	100b	0	0	0	6	0	0	0	0	0	0	1

OPERATION	DIFFERENT TIME FRAME	MEMORY	AR	PC	DR	AC	IR	TR	OUTR	INPR	SC	I	S	E	R	IEN	FGI	FGO
STA	t_0	006	11	2	6	11	100b	0	0	0	0	0	0	0	0	0	0	1
	t_0	006	11	2	6	11	100b	0	0	0	0	0	0	0	0	0	0	1
	t_1	300c	2	2	6	11	100b	0	0	0	1	0	0	0	0	0	0	1
	t_2	300c	2	3	6	11	300c	0	0	0	2	0	0	0	0	0	0	1
	t_3	000	13	3	6	11	300c	0	0	0	3	0	0	0	0	0	0	1
	t_4	000	13	3	6	11	300c	0	0	0	4	0	0	0	0	0	0	1
	t_5	11	13	3	6	11	300c	0	0	0	5	0	0	0	0	0	0	1
OUT	t_0	11	13	3	6	11	200a	0	0	0	0	0	0	0	0	0	0	1
	t_1	f400	3	3	6	11	200a	0	0	0	1	0	0	0	0	0	0	1
	t_2	f400	3	4	6	11	f400	0	0	0	2	1	0	0	0	0	0	1
	t_3	00	400	4	6	11	f400	0	0	0	3	1	0	0	0	0	0	1
	t_4	00	400	4	6	11	f400	0	11	0	4	1	0	0	0	0	0	1

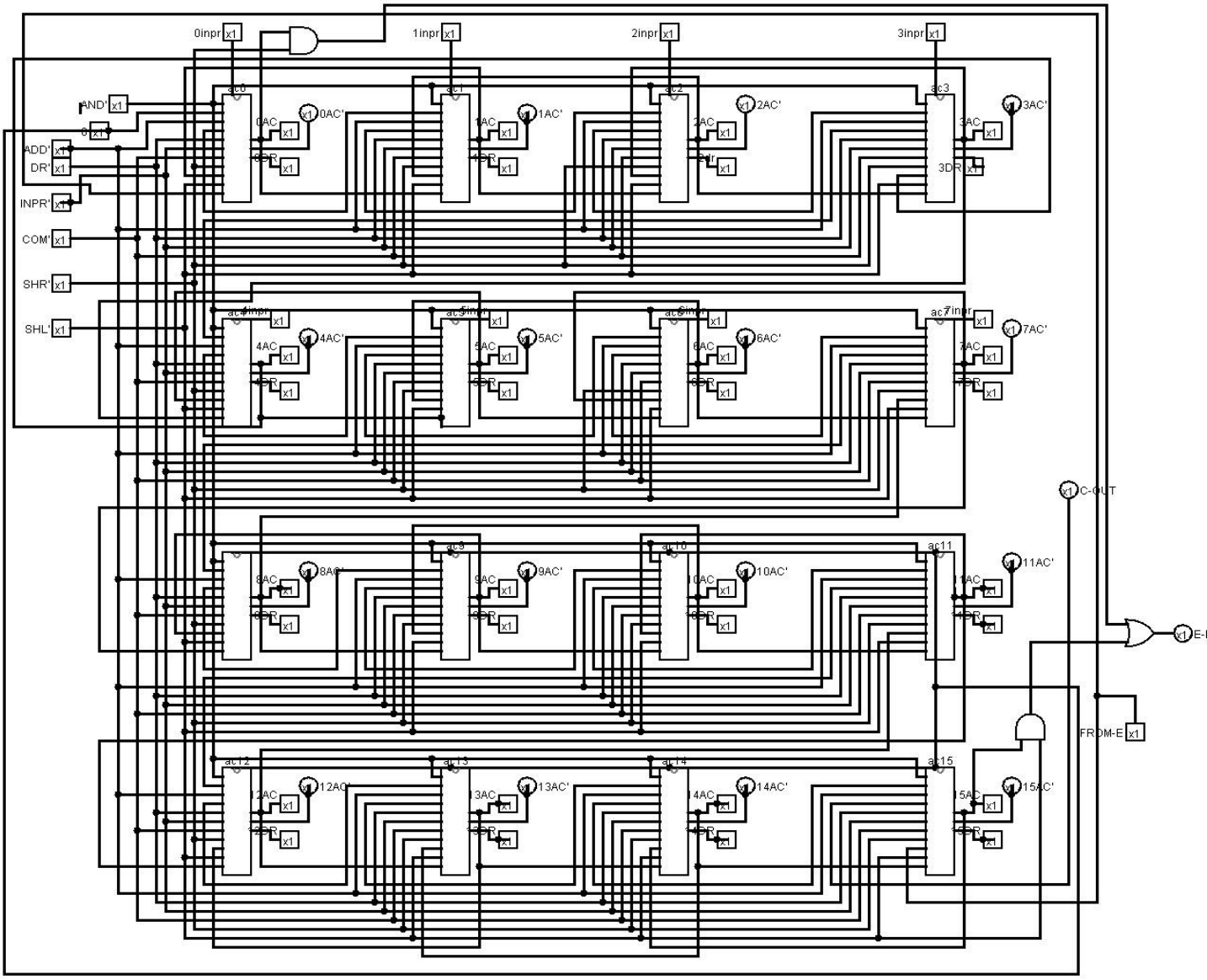
CIRCUIT OF 4BIT SEQUENCE COUNTER



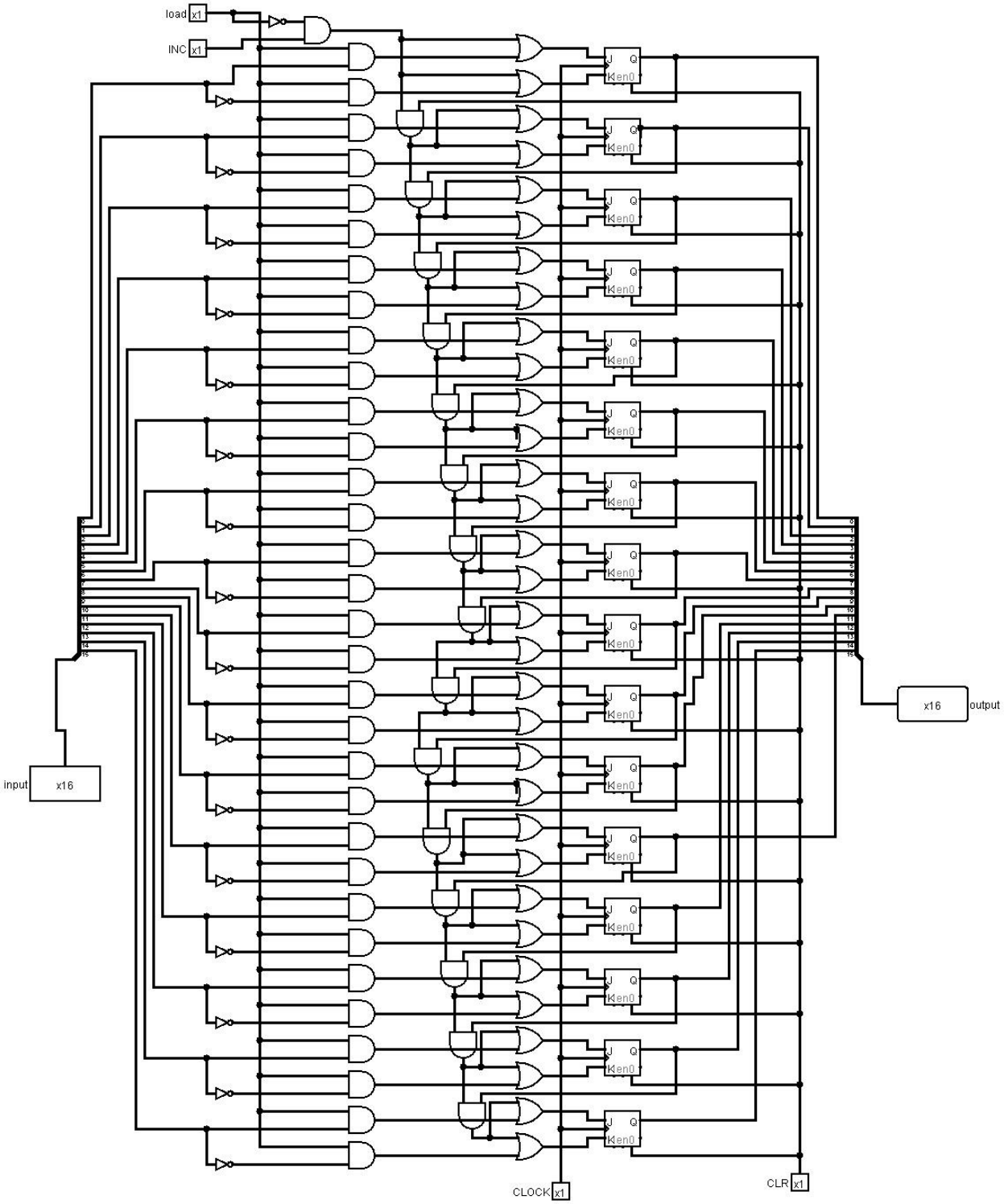
CIRCUIT OF DATA REGISTER



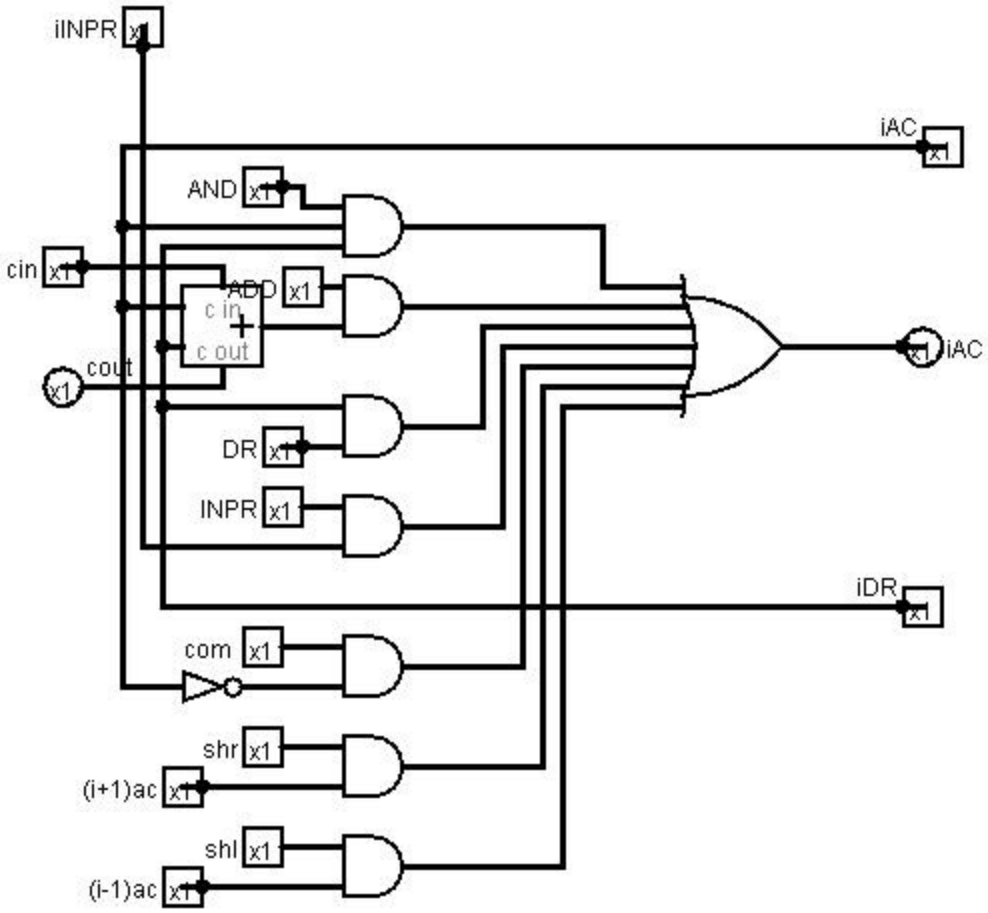
CIRCUIT OF ADD AND INCREMENT



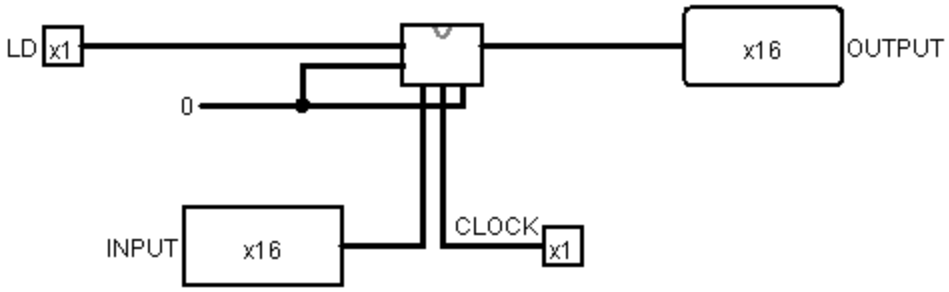
CIRCUIT OF 16 BIT REGISTER



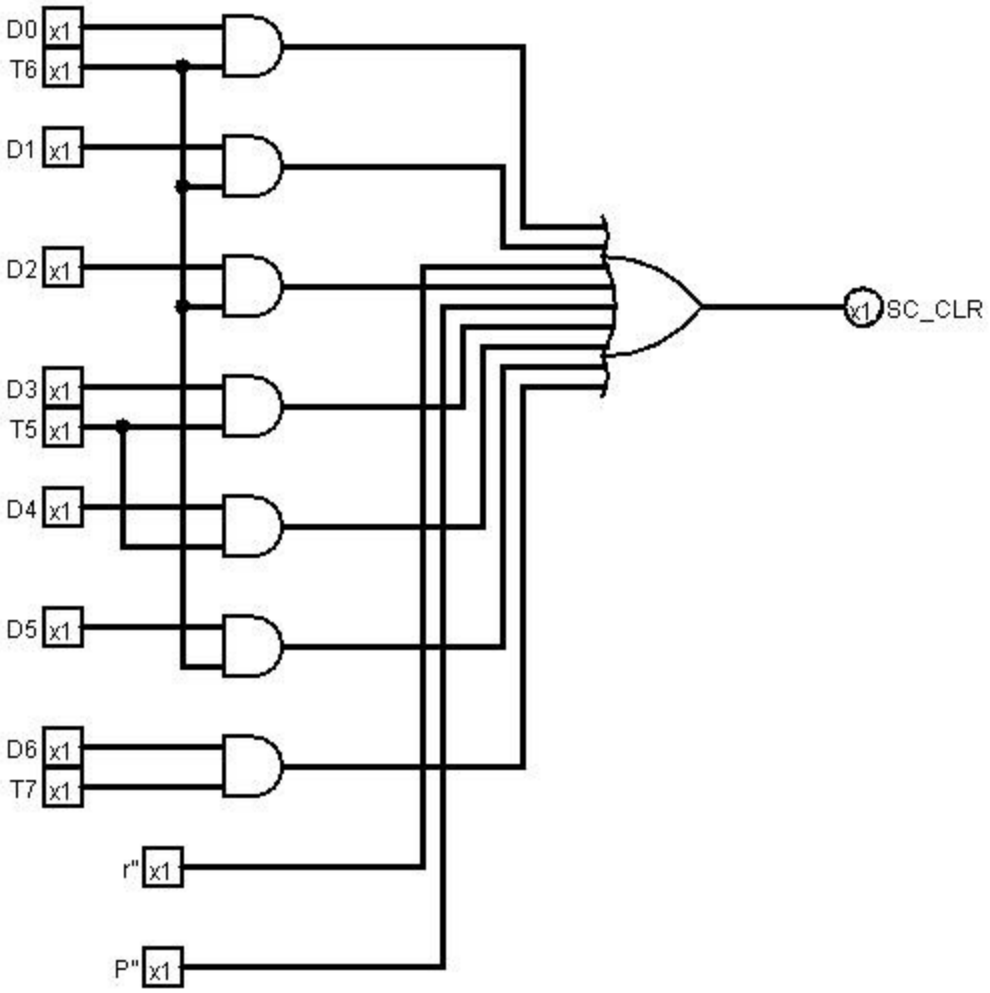
CIRCUIT OF ADDER AND LOGIC



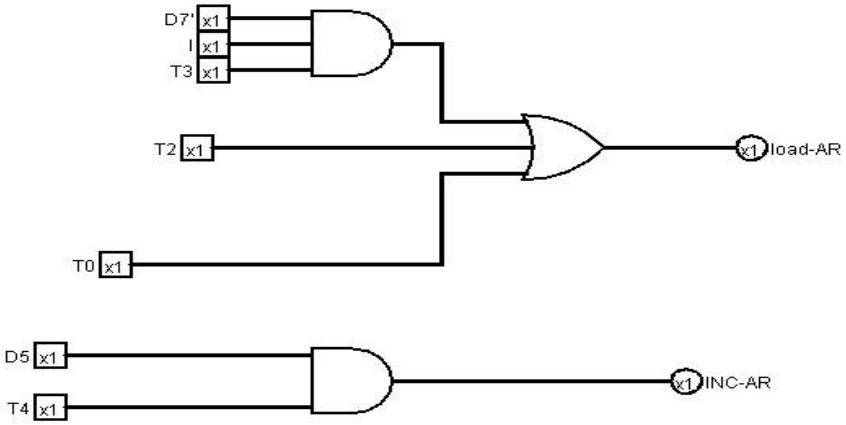
CIRCUIT OF TEMPORARY REGISTER



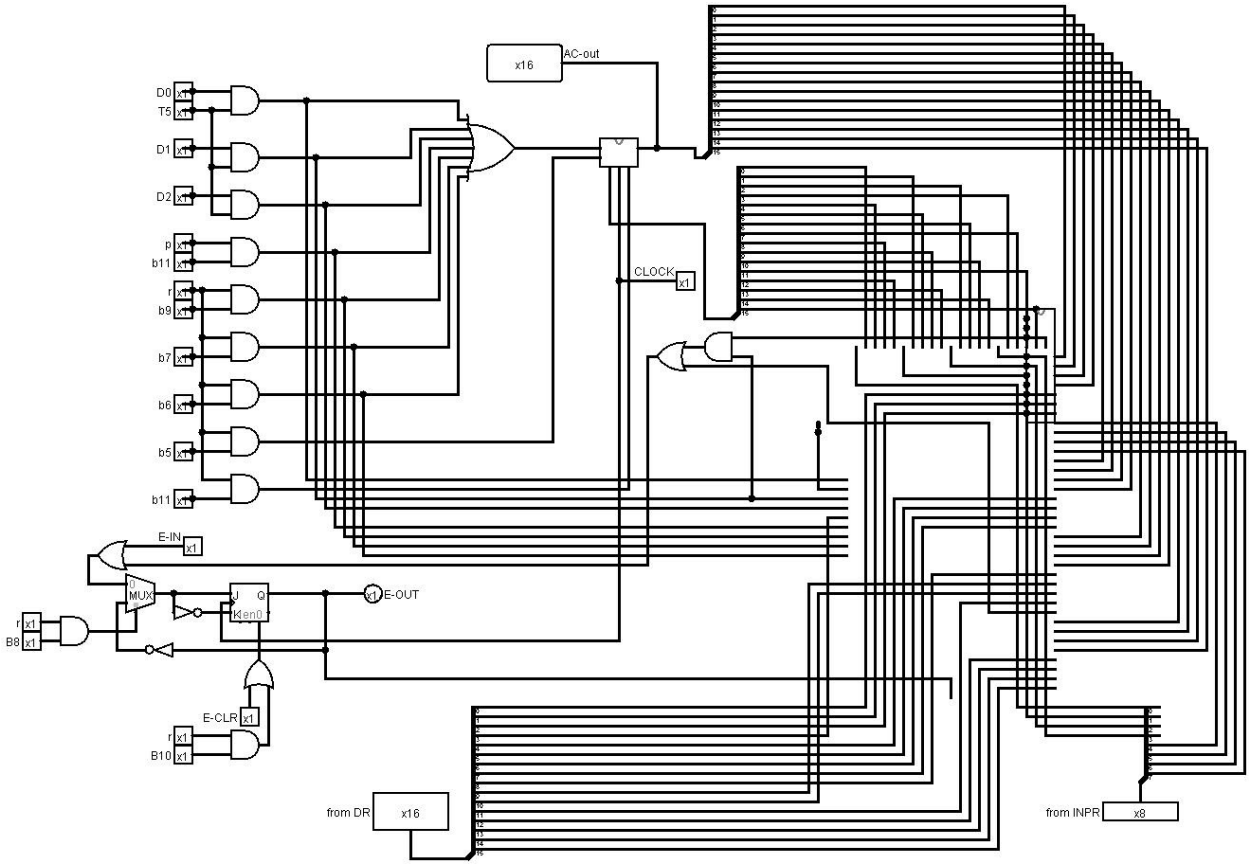
CIRCUIT OF SEQUENCE COUNTER



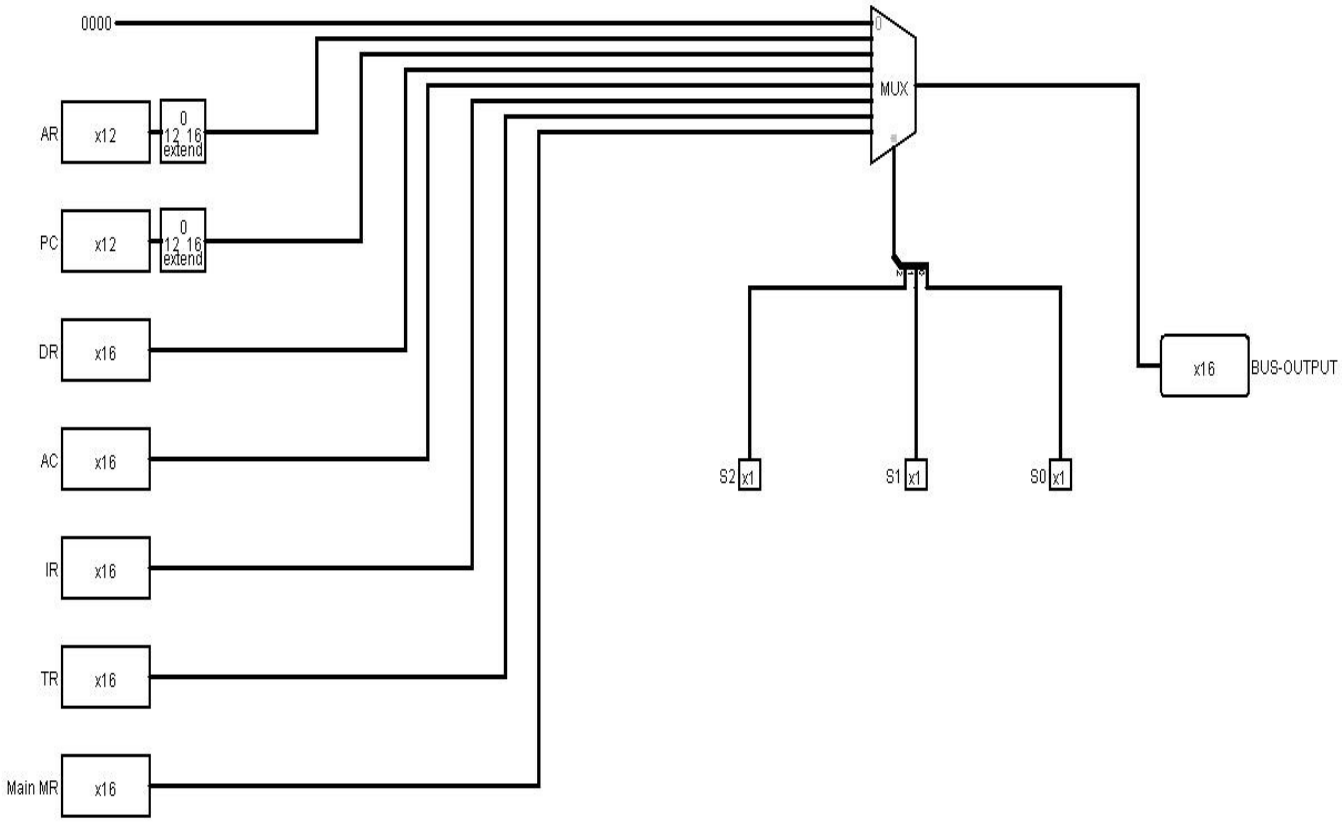
CIRCUIT OF ADDRESS REGISTER



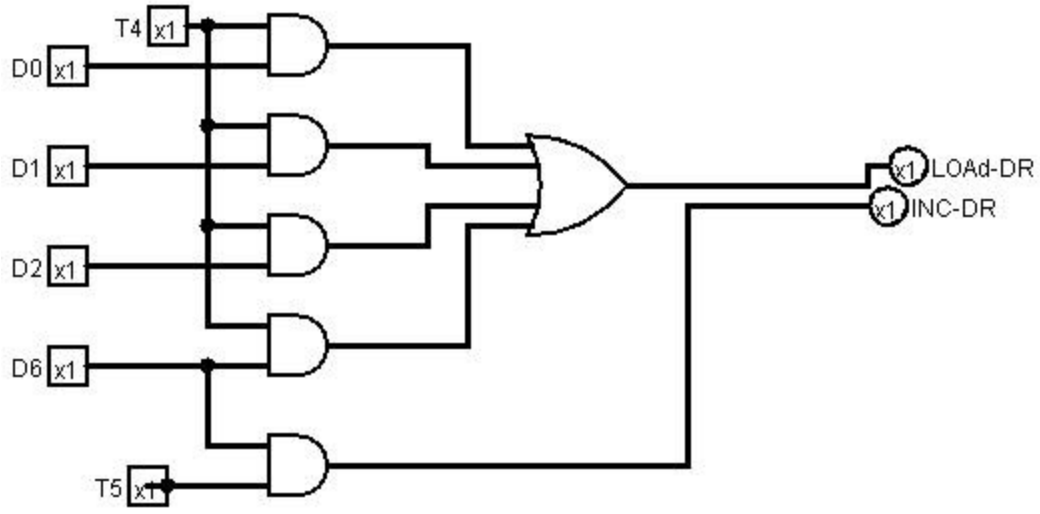
CIRCUIT OF ADDER AC BUS



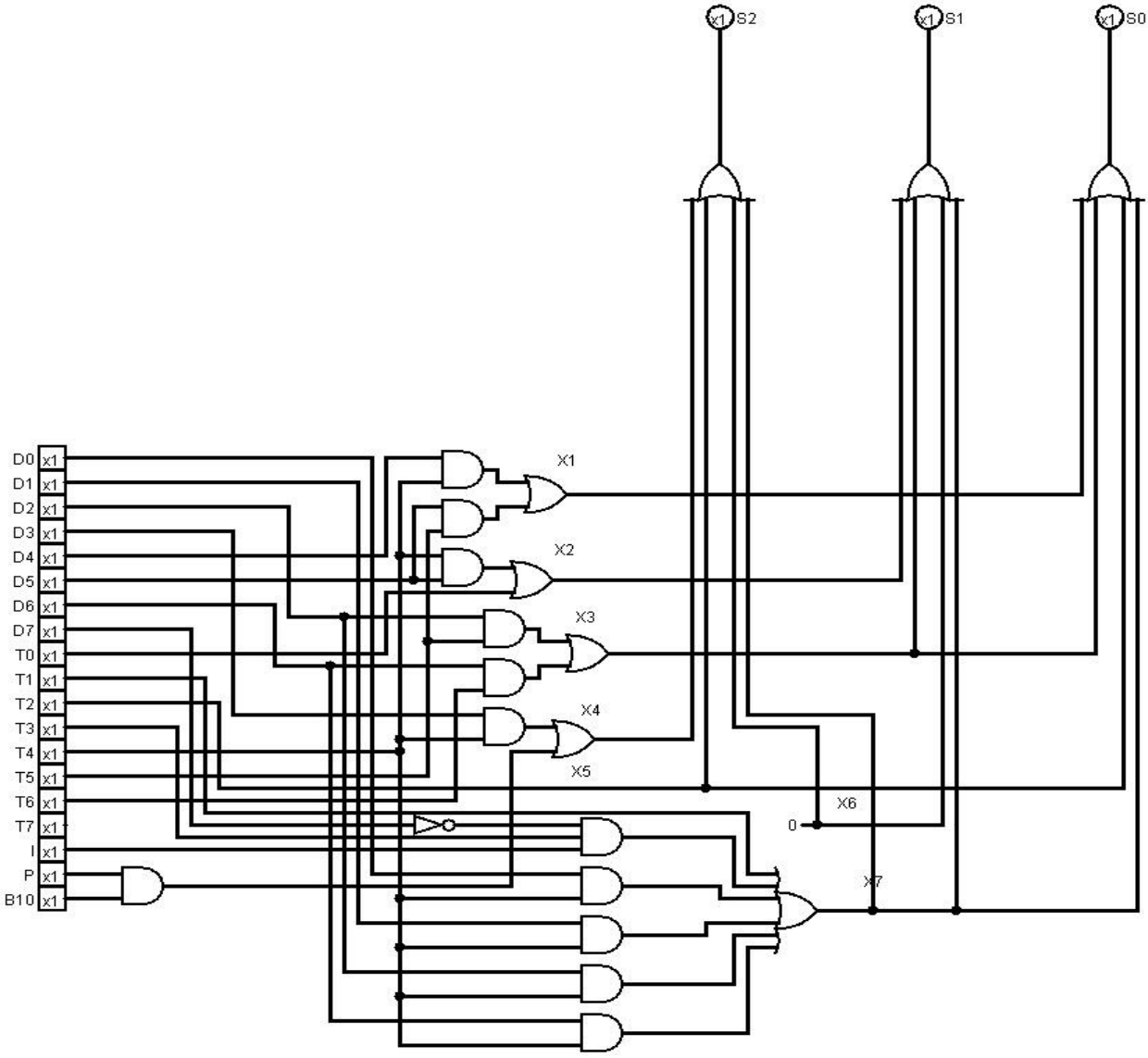
CIRCUIT OF BUS(MAIN)



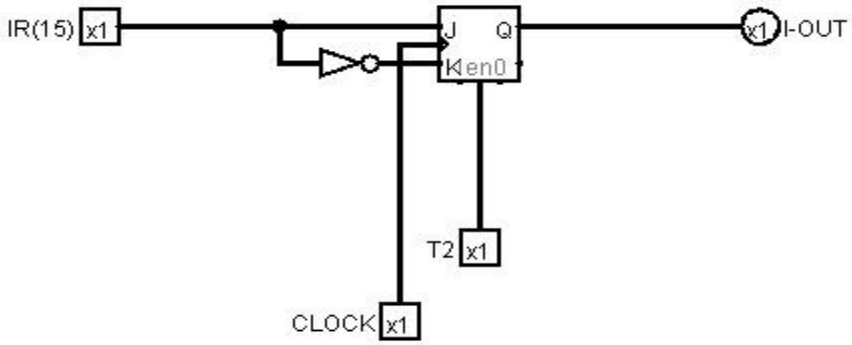
CIRCUIT OF DATA REGISTER



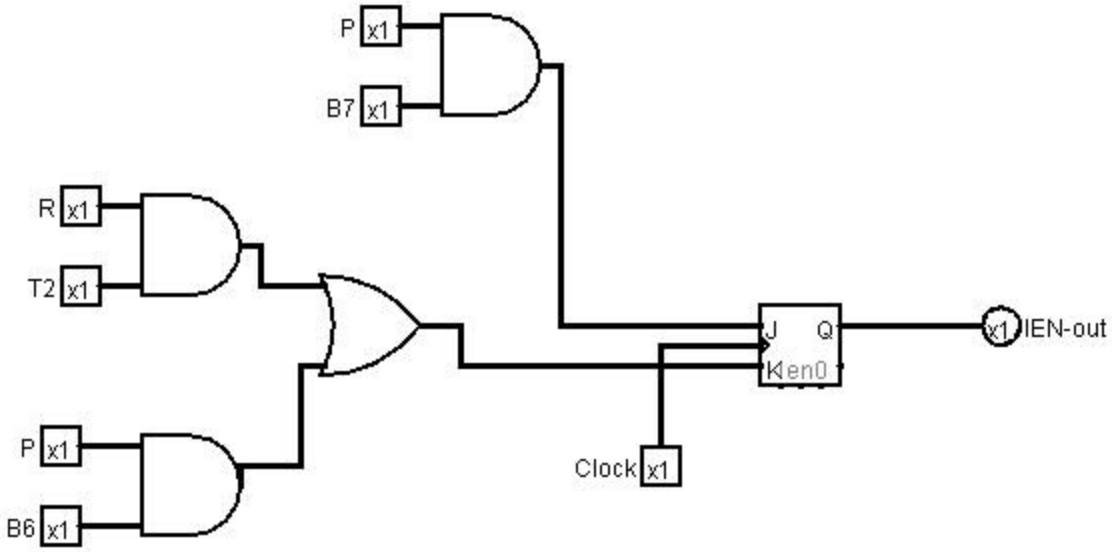
CIRCUIT OF BUS



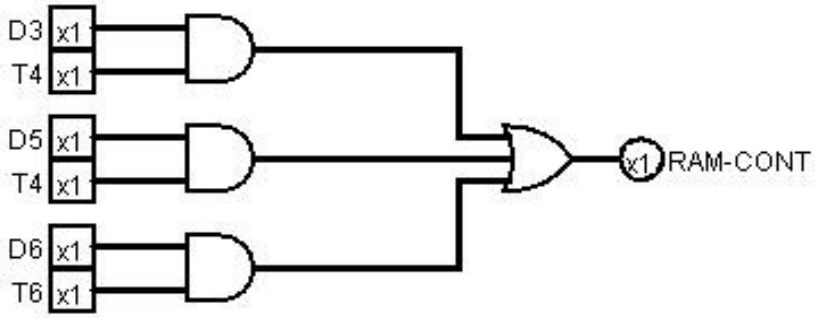
CIRCUIT OF I(FLIP FLOP)



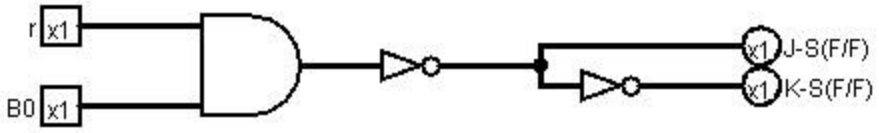
CIRCUIT OF IEN



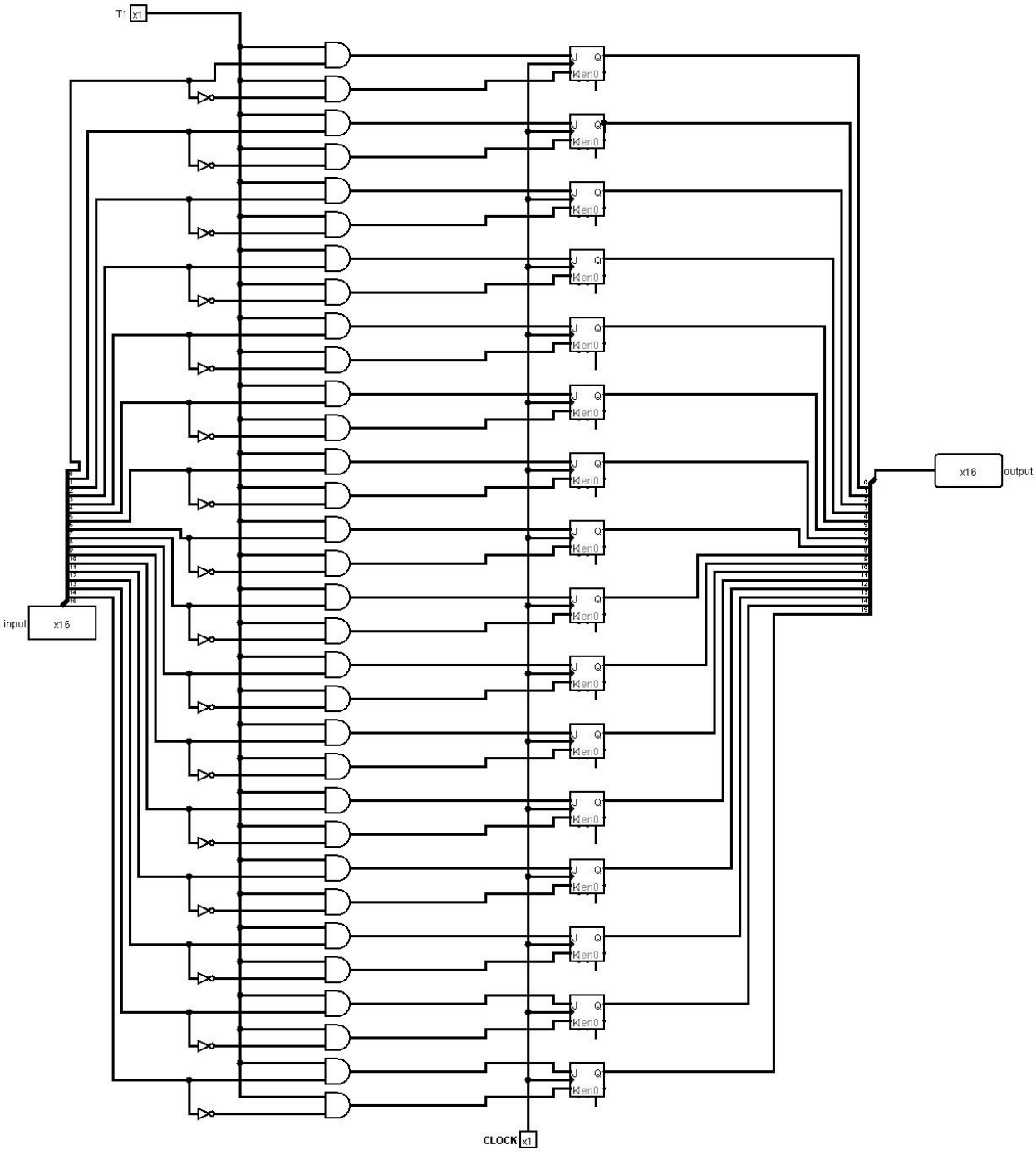
CIRCUIT OF RAM



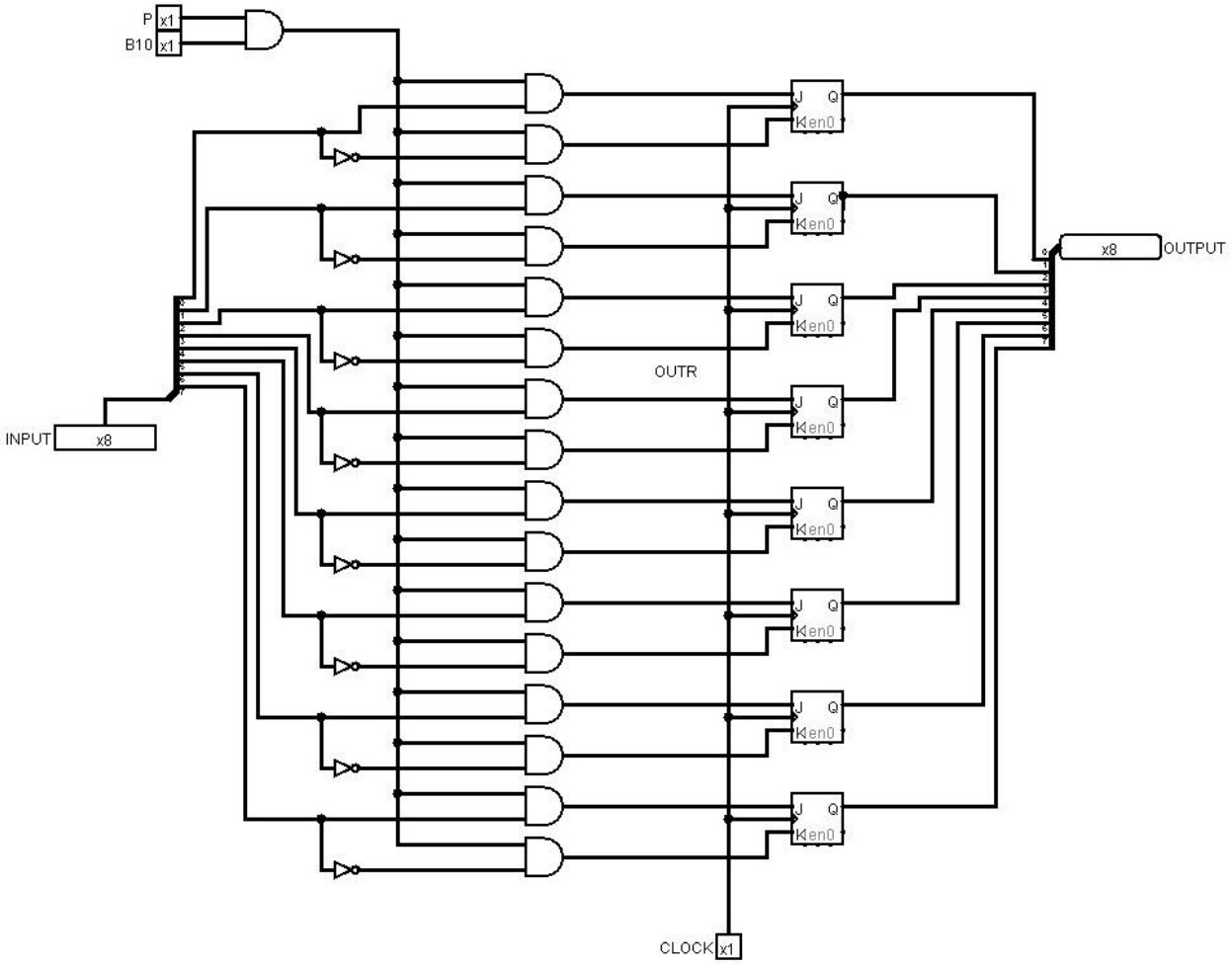
CIRCUIT OF S(FLIP FLOP)



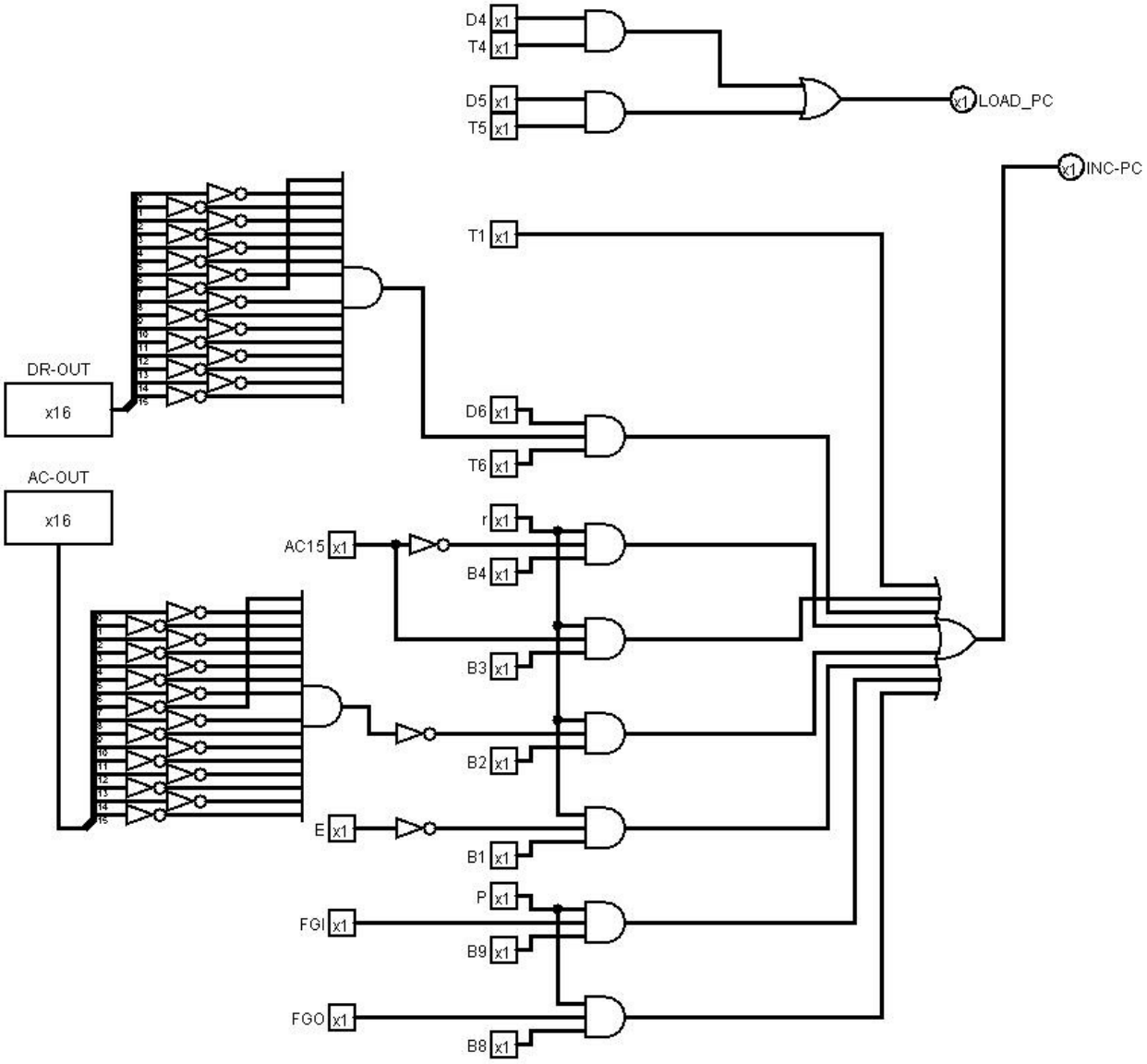
CIRCUIT OF INSTRUCTION REGISTER



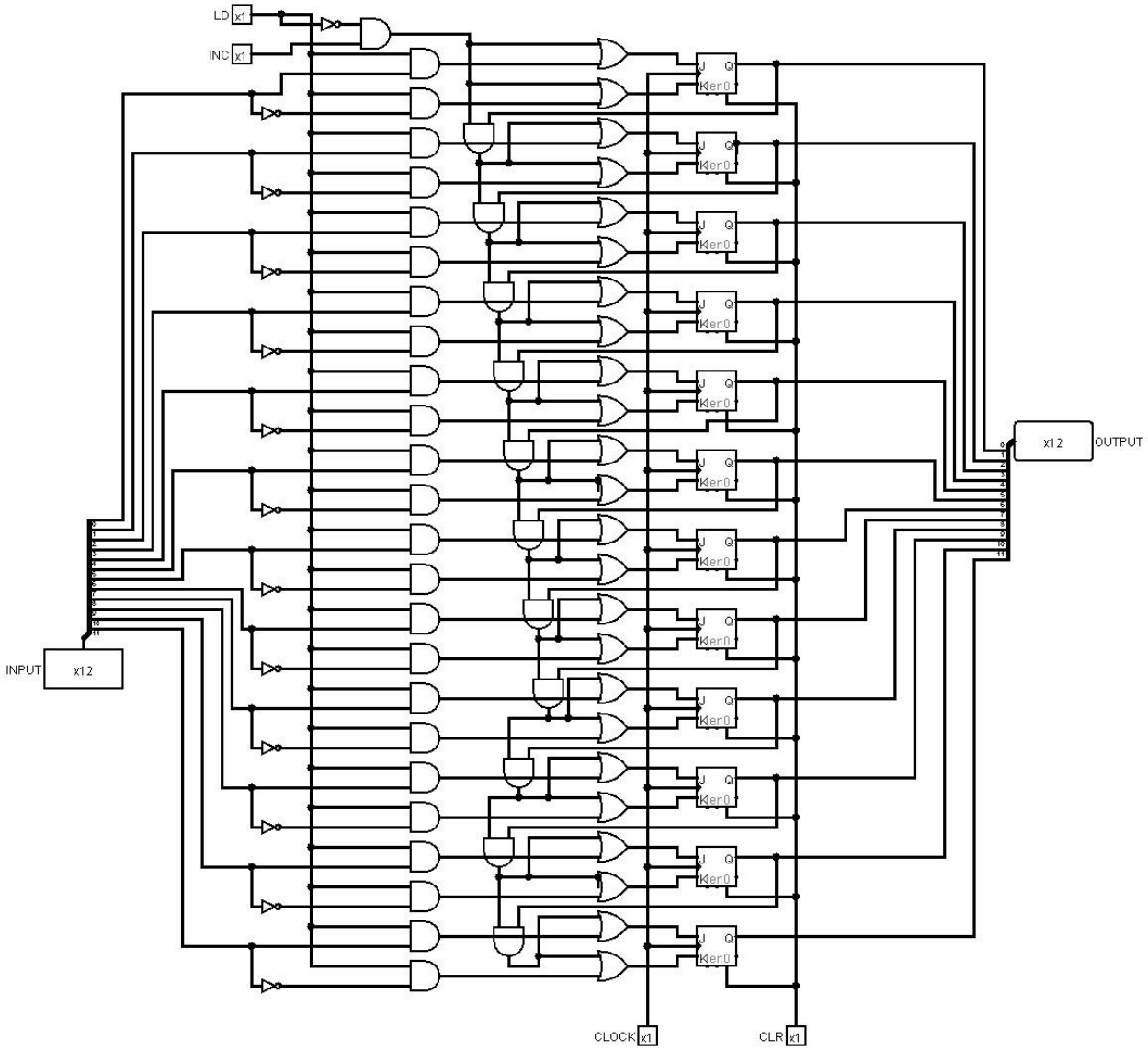
CIRCUIT OF OUTPUT REGISTER



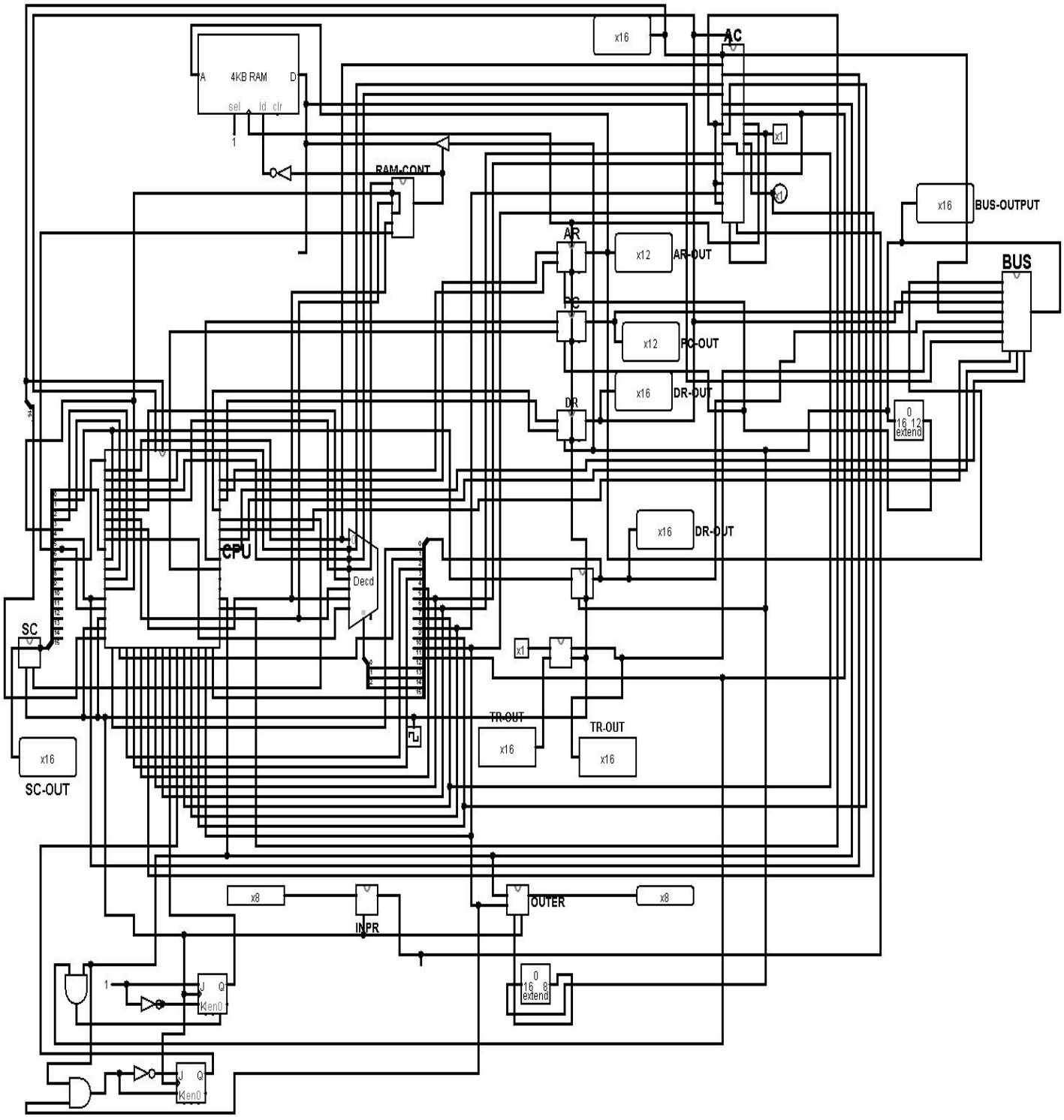
CIRCUIT OF PROGRAM COUNTER



CIRCUIT OF r12 LD,INR,CLR (AR,PC)



CIRCUIT OF THE WHOLE SYSTEM



RESULT:

1. We learnt how to design circuit 4bit sequence counter,DR,BUS,I(ff),IEN,RAM,S(ff),IR,OUTR,INPR,PC.
2. We learnt to implement computer instruction.
3. We learnt to implement of ALU.
4. We learnt to implement the I/O system.
5. We learnt to implement the Control Unit.

THANK YOU……