Basic computer organization and design

By,
Er. Nabaraj Bahadur Negi

Contents

- Instruction Code, Operation Code, Stored Program Concept
- Registers and memory of Basic Computer, Common Bus System for Basic Computer.
- Instruction Format, Instruction Set Completeness, Control Unit of Basic Computer, Control Timing Signals
- Instruction Cycle of Basic computer, Determining Type of Instruction,
- Memory Reference Instructions, Input-Output Instructions, Program Interrupt & Interrupt Cycle.
- Description and Flowchart of Basic Computer

Introduction

- The organization of the computer is defined by its internal register, the timing and control structure, and the set of instructions that it uses.
- The internal organization of a digital system is defined by the sequence of microoperations it performs on data stored in its registers.
- Every different processor type has its own design (different registers, buses, microoperations, machine instructions, etc.)
- Modern processor is a very complex device
- It contains
  - Many registers
  - Multiple arithmetic units, for both integer and floating point calculations
  - The ability to pipeline several consecutive instructions to speed execution

- However, to understand how processors work, we will start with a simplified processor model
- This is similar to what real processors were like ~25 years ago
- M. Morris Mano introduces a simple processor model he calls the Basic Computer
- We will use this to introduce processor organization and the relationship of the RTL model to the higher level computer processor

The basic computer

- The Basic Computer has two components, a processor and memory
- The memory has 4096 words in it
  - $4096 = 2^{12}$, so it takes 12 bits to select a word in memory
- Each word is 16 bits long

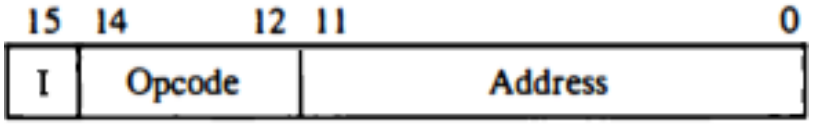
Instructions

- A computer instruction is a binary code that specifies a sequence of microoperations for the computer.
- Instruction codes together with data are stored in memory.
- The computer reads each instruction from memory, the CPU reads the next instruction from memory and places it in in (**Instruction Register (IR)**) a control register.
- The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of microoperations.
- Every computer has its own unique instruction set.

Instruction Format

- A computer instruction is often divided into two parts
  - An **opcode (Operation Code)** is the portion of a machine language instruction that specifies the operation for that instruction. also known as instruction machine **code**, instruction **code**, instruction syllable.
  - The operation code of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement.
  - An **address** that specifies the registers and/or locations in memory to use for that operation
- In the Basic Computer, since the memory contains 4096 (= 2^{12}) words, we needs 12 bit to specify which memory address this instruction will use
- In the Basic Computer, bit 15 of the instruction specifies the **addressing mode** (0: direct addressing, 1: indirect addressing)

- Since the memory words, and hence the instructions, are 16 bits long, It consists of a 3-bit operation code, a 12-bit address, and an indirect address mode bit designated by I. the mode bit is 0 for a direct and 1 for an indirect address.



(a) Instruction format

Stored Program Organization

- The simplest way to organize a computer is to have one processor register and instruction code format with two parts.
- The first part specifies the operation to be performed and the second specifies and address.
- The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register.
- Figure below depicts this type of organization. Instructions are stored in one section of memory and a data in another.
- For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$.

- If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated as opcode) to specify one out of 16 possible operation and 12 bits to specify the address of an operand.
- The control reads a 16-bit instruction from the program portion of memory. It uses the 12 bit address part of the instruction to read and 16 bit operand from the data portion of memory. It then executes the operation specified by the operation code.

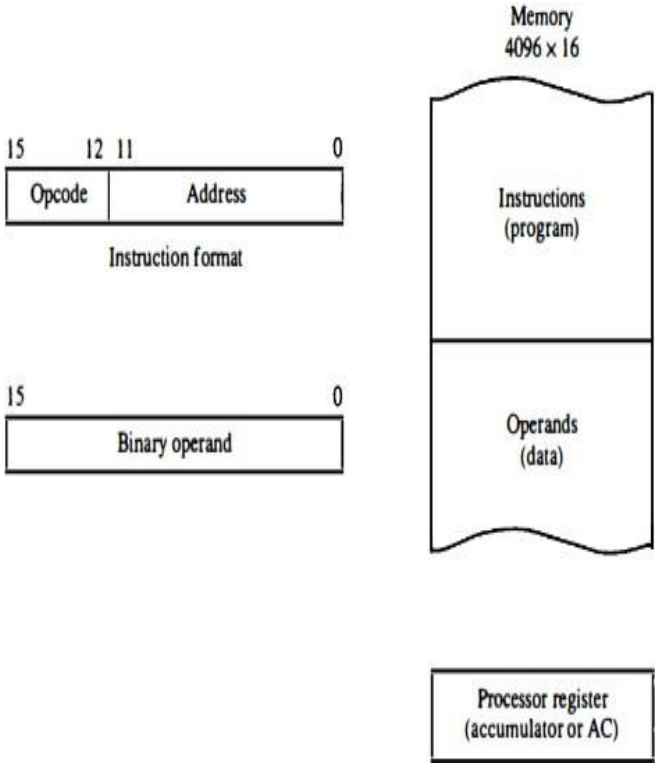


Figure : Stored program organization.

- Computer that have a single processor register usually assign to it the name accumulator and label it AC. The operation is performed with the memory operand and the content of AC.
- If an operation in an instruction code does not need and operand from memory, the rest of the bits in the instruction can be used for other purposes.
- For example, operations such as clear AC, complement AC, and increment AC operate on data stored in the AC register.

Addressing Modes

- The address field of an instruction can represent either
  - Direct address: the address in memory of the data to use (the address of the operand), or
  - Indirect address: the address in memory of the address in memory of the data to use

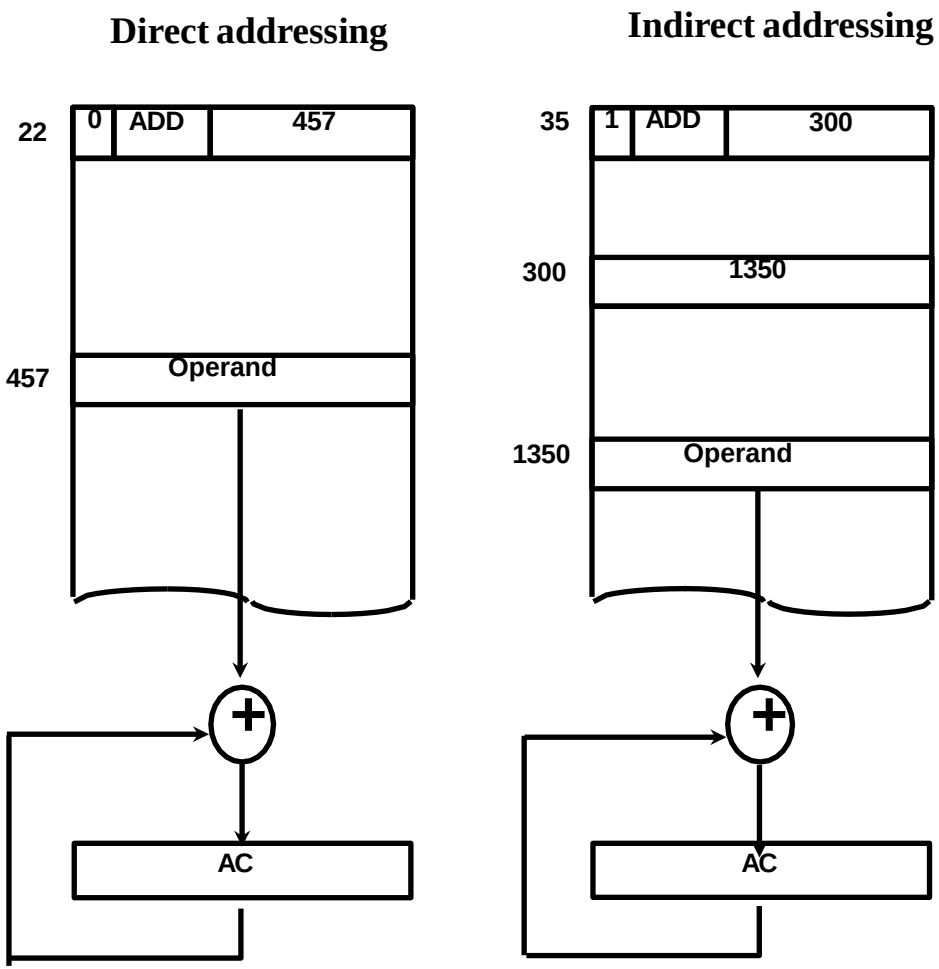


Figure : Demonstration of direct and indirect address.

- A direct address instruction is shown in figure : b It is placed in address 22 in memory. The I bit is 0, so the instruction is recognized as a direct address instruction. The opcode specifies an ADD instruction, and the address part is the binary equivalent of 457. The control finds the operand in memory at address 457 and adds it to the content of AC.
- The instruction in address 35 shown in figure : c has a mode bit I = 1. Therefore, it is recognized as an indirect address instruction. The address part is the binary equivalent of 300. The control goes to address 300 to find the address instruction needs two references to memory to fetch and operand. The address of the operand in this case is 1350. The operand found in address 1350 is then added to the content of AC.
- Effective Address(EA) is the location where operand is present , or as the target address for a branch-type instruction

Basic Computer Registers and Memory

- 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. This type of instruction sequencing needs a counter to calculate the address of the next instruction after execution of the current instruction is completed.
- The computer needs processor registers for manipulating data and a register for holding a memory address.
- The memory unit has a capacity of 4096 words and each word contains 16 bits. Twelve bits of an instruction word are needed to specify the address of an operand.

- A processor has many registers to hold instructions, addresses, data, etc
- The data register (DR) holds the operand read from memory. The accumulator (AC) register is a general-purpose processing register.
- The instruction read from memory is placed in the instruction register (IR). The temporary register (TR) is used for holding temporary data during the processing
- The processor has a register, the Program Counter (PC) that holds the memory address of the next instruction to be read from memory after the current instruction is executed. Since, the memory in the Basic Computer only has 4096 locations, the PC only needs 12 bits

- In a direct or indirect addressing, the processor needs to keep track of what locations in memory it is addressing: The Address Register (AR) is used for this
- The Input Register (INPR) holds an 8 bit character gotten from an input device
- The Output Register (OUTR) holds an 8 bit character to be send to an output device

Register symbol	Number of bits	Register name	Function
DR	16	Data Register	Holds memory operand
AR	12	Address Register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction Register	Holds instruction code
PC	12	Program Counter	Holds address of instruction
TR	16	Temporary Register	Holds temporary data
INPR	8	Input Register	Holds input character
OUTR	8	Output Register	Holds output character

Table : List of Registers for the Basic Computer

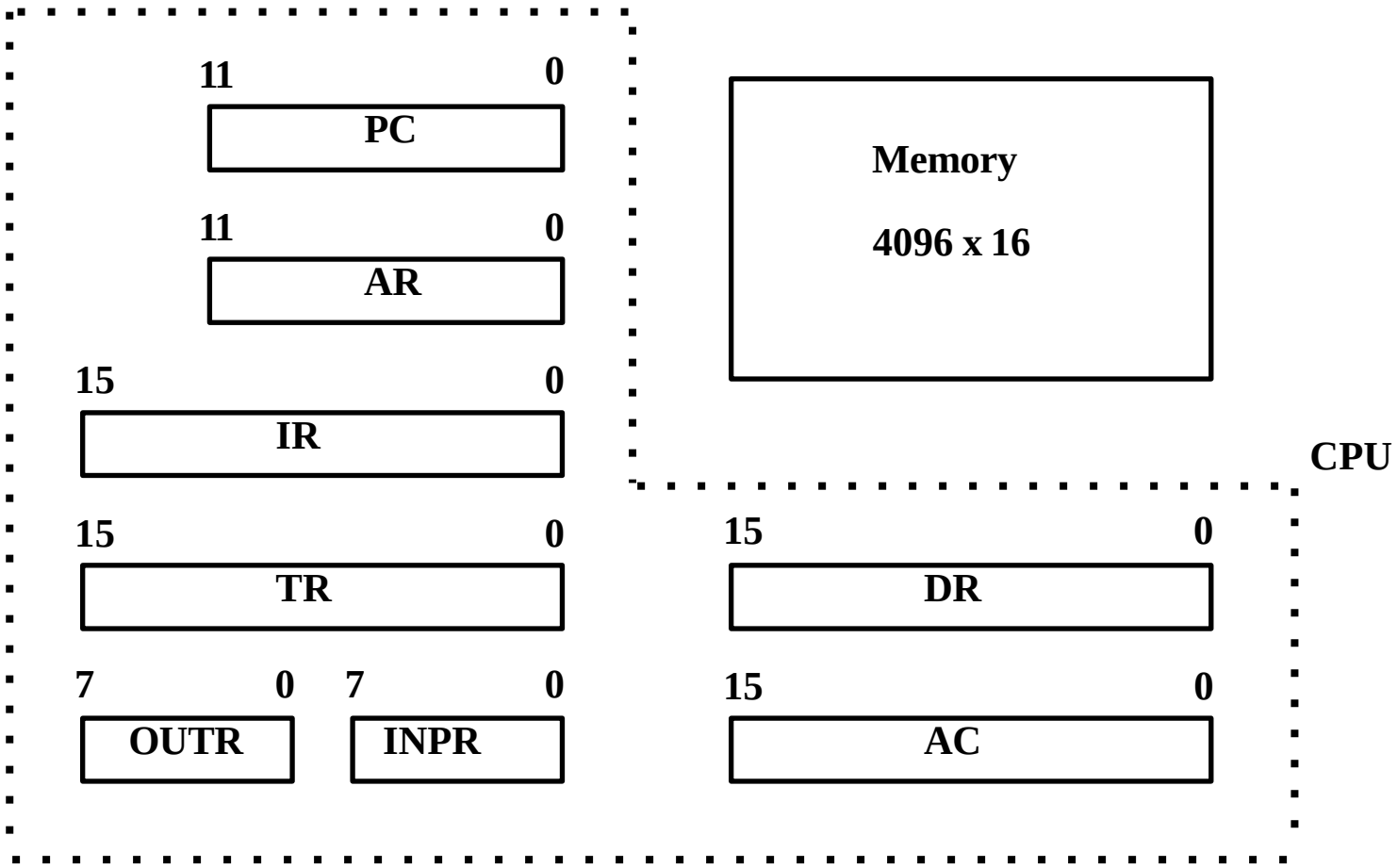


Figure : Basic computer registers and memory.

Common Bus System

- The basic computer has eight registers, a memory unit, and a control unit. Paths must be provided to transfer information from one register to another and between memory and register.
- The registers in the Basic Computer are connected using a bus
- This gives a savings in circuitry over complete connections between registers
- The outputs of seven registers and memory are connected to the common bus.
- Three control lines, S_2, S_1, and S_0 control which register the bus selects as its input.
- For example, the number along the output of DR is 3. The 16-bit outputs of DR are placed on the bus lines when $S_2S_1S_0 = 011$ since this is the binary value of decimal 3.

S_2	S_1	S_0	Register
0	0	0	x
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

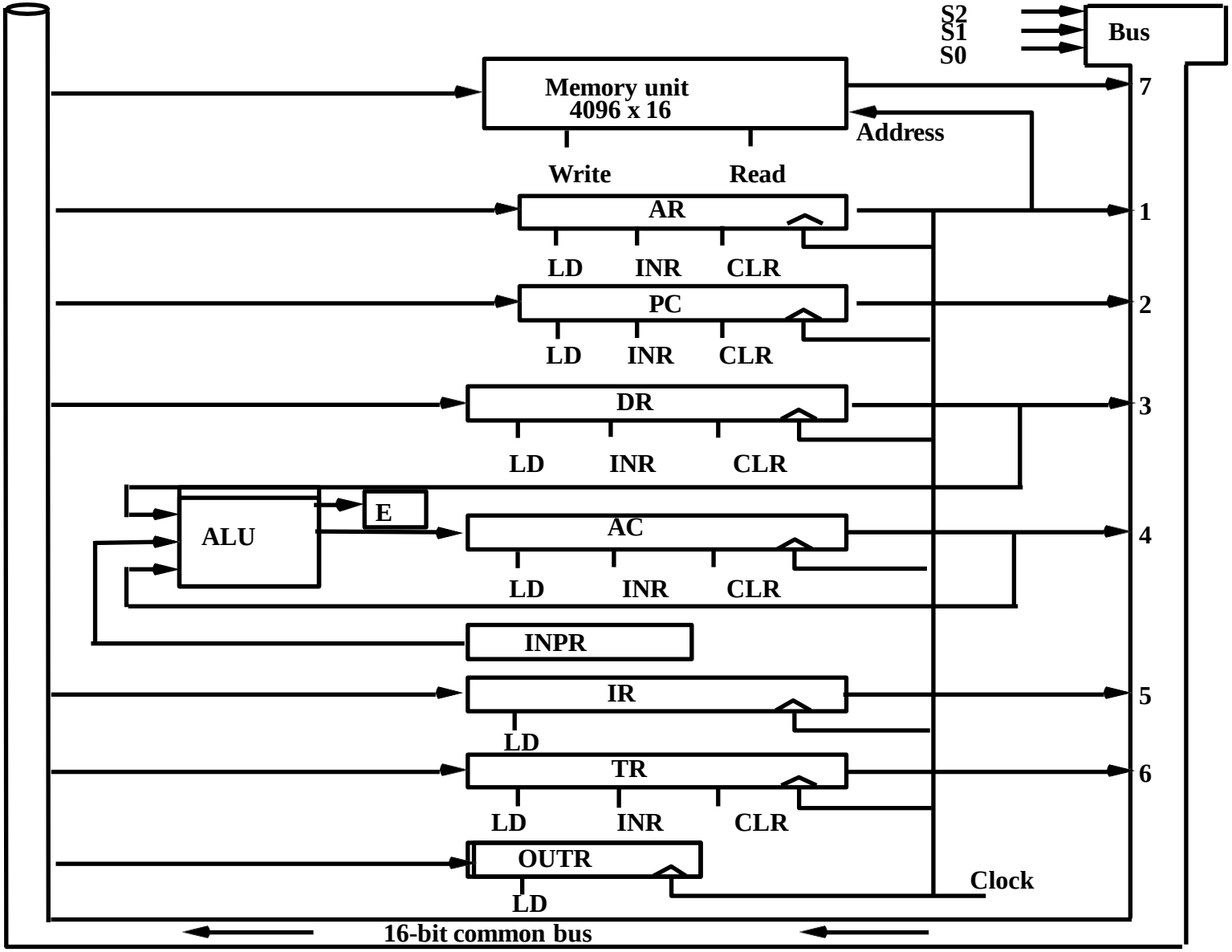


Figure : Basic computer registers connected to a common bus.

Example:

One Address Instructions

Expression: X = (A+B)*(C+D)

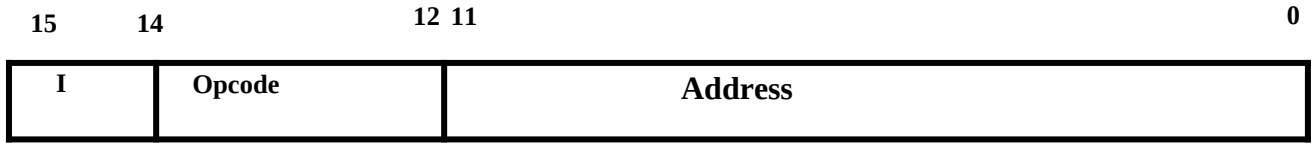
- AC is accumulator M[] is any memory location M[T] is temporary location

LOAD	A	AC = M[A]
ADD	B	AC = AC + M[B]
STORE	T	M[T] = AC
LOAD	C	AC = M[C]
ADD	D	AC = AC + M[D]
MUL	T	AC = AC * M[T]
STORE	X	M[X] = AC

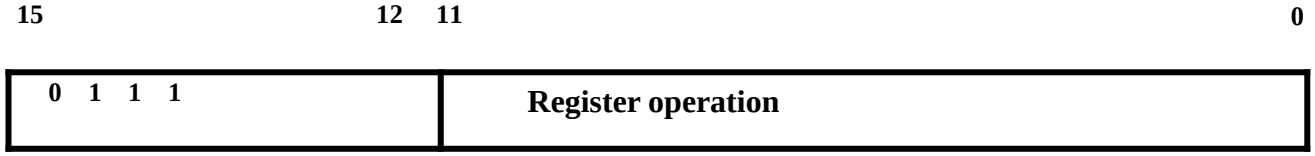
Basic Computer Instructions

- The basic computer has three instruction code formats. Each format has 16 bits. The operation code (opcode) part of the instruction contains three bits and the meaning of the remaining 13 bits depends on the operation code encountered.
- A memory-reference instruction uses 12 bits to specify and address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.
- The register reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction.
- A register-reference instruction specifies an operation on or a test of the AC register. An operand from memory is not needed; therefore, the other 12 bits are used to specify the operation or test to be executed.

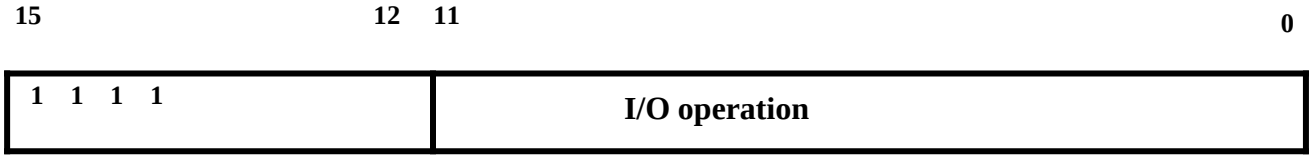
- Similarly, an input output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type of input-output operation or test performed.
- The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction.
- If the three opcode bits in positions 12 through 14 are not equal to 111, the instruction is a memory-reference type and the bit in position 15 is taken as the addressing mode I. If the 3-bit opcode is equal to 111, control then inspects the bit in position 15.
- If this bit is 0, the instruction is a register - reference type.
- If the bit is 1, the instruction is an input-output type.
- Note that the bit in position 15 of the instruction code is designated by the symbol I but is not used as a mode bit when the operation code is equal to 111.



(a) Memory-Reference Instructions (Opcode = 000 ~ 110)



(b) Register-Reference Instruction (OP-code = 111, I = 0)



(c) Input-Output Instructions (OP-code =111, I = 1)

Figure : Basic computer instruction formats.

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

Table : Basic Computer Instruction

❖ **Register- reference instructions**

- Register-reference instructions are recognized by the control when $D_7 = 1$ and $I = 0$.
- These instructions use bits 0 through 11 of the instruction code to specify one of 12 instructions. These 12 bits are available in IR(0-11). They were also transferred to AR during time T_2
- 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.[bit in IR(0-11) that specifies the operation]

	r:	SC ← 0	Clear SC
CLA	rB_{11}:	AC ← 0	Clear AC
CLE	rB_{10}:	E ← 0	Clear E
CMA	rB_9:	AC ← AC’	Complement AC
CME	rB_8:	E ← E’	Complement E
CIR	rB_7:	AC ← shr AC, AC(15) ← E, E ←AC(0)	Circulate right
CIL	rB_6:	AC ← shl AC, AC(0) ← E, E ←AC(15)	Circulate left
INC	rB_5:	AC ← AC + 1	Increment AC
SPA	rB_4:	if (AC(15) = 0) then (PC ← PC+1)	Skip if positive
SNA	rB_3:	if (AC(15) = 1) then (PC ← PC+1)	Skip if negative
SZA	rB_2:	if (AC = 0) then (PC ← PC+1)	Skip if A C zero
SZE	rB_1:	if (E = 0) then (PC ← PC+1)	Skip if E zero
HLT	rB_0:	S ← 0 (S is a start-stop flip-flop)	Halt computer

Table: Execution of Register –Reference instructions

❖ Memory - Reference Instructions

- The table below lists the seven memory-reference instructions.
- The decoded output D, for i = 0,1, 2, 3, 4, 5, and 6 from the operation decoder that belongs effective address to each instruction is included in the table.
- The effective address of the instruction is in the address register AR and was placed there during timing signal T_2 when I = 0, or during timing signal T_3 when I=1.
- The symbolic description of each instruction is specified in the table in terms of register transfer notation. The actual execution of the instruction in the bus system will require a sequence of microoperations.
- The execution of MR instruction starts with T_4

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

Table : Memory-Reference Instructions

AND to AC

- This is an instruction that performs the AND logic operation on pairs of bits in AC and the memory word specified by the effective address. The result of the operation is transferred to AC . The microoperations that execute this instruction are:

D_0T_4: $DR \leftarrow M[AR]$ //Read operand

D_0T_5: $AC \leftarrow AC \wedge DR, SC \leftarrow 0$ //AND with AC

ADD to AC

- This instruction adds the content of the memory word specified by the effective address to the value of AC . The sum is transferred into AC and the output carry C_{out} is transferred to the E (extended accumulator) flip-flop. The microoperations needed to execute this instruction are

D_1T_4: DR ← M[AR] //Read operand

D_1T_5: AC ← AC + DR, E ← C_{out}, SC ← 0 //Add to AC and store carry in E

LDA: Load to AC

- This instruction transfers the memory word specified by the effective address to AC . The microoperations needed to execute this instruction are

D_2T_4: DR ← M[AR] //Read operand

D_2T_5: AC ← DR, SC ← 0 //Load AC with DR

STA: Store AC

- This instruction stores the content of AC into the memory word specified by the effective address. Since the output of AC is applied to the bus and the data input of memory is connected to the bus, we can execute this instruction with one microoperation

D_3T_4: M[AR] ← AC, SC ← 0 //store data into memory location

BUN: Branch Unconditionally

- PC is incremented at time T1 to prepare it for the address of the next instruction in the program sequence. The BUN instruction allows the programmer to specify an instruction out of sequence and we say that the program branches (or jumps) unconditionally. The instruction is executed with one microoperation:

D_4T_4: PC ← AR, SC ← 0 //Branch to specified address

BSA: Branch and Save Return Address

D_5T_4: M[AR] ← PC, AR ← AR + 1 //save return address and increment AR

D_5T_5: PC ← AR, SC ← 0 //load PC with AR

- This is shown in part (a) of the figure. The BSA instruction performs the following numerical operation:

M[135] ← 21, PC ← 135 + 1 = 136 The result of this operation is shown in part (b) of the figure.

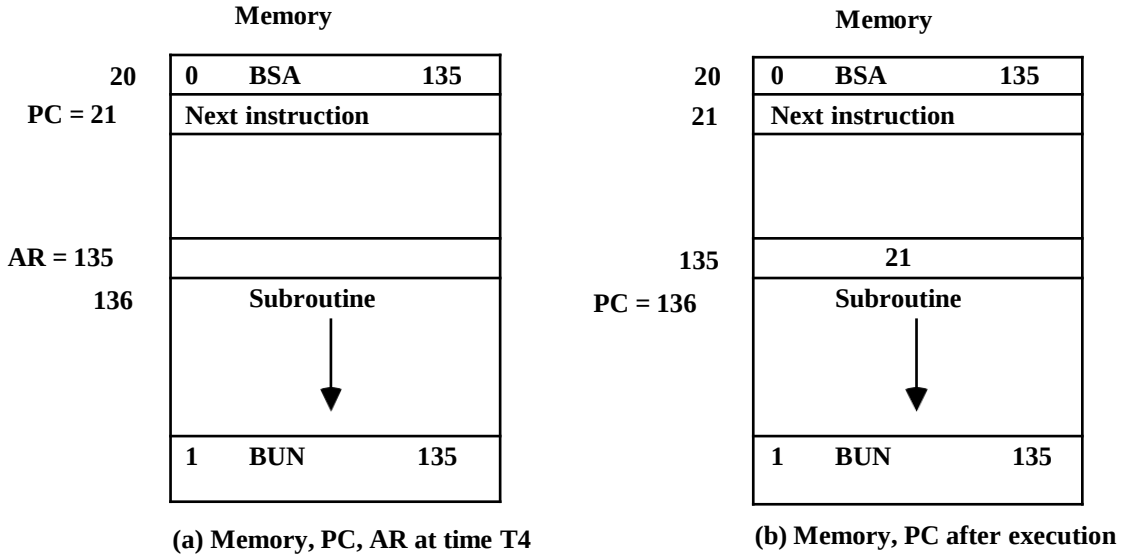


Figure : Example of BSA instruction execution.

ISZ: Increment and Skip-if-Zero

- This instruction increments the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1.
- It is not possible to increment a word inside the memory, it is necessary to read the word into DR, increment DR, and store the word back into memory. This is done with the following sequence of microoperations:

D_6T_4: DR ← M[AR] //load data into DR

D_6T_5: DR ← DR + 1 //increment data

D_6T_4: M[AR] ← DR, if (DR = 0) then (PC ← PC + 1), SC ← 0

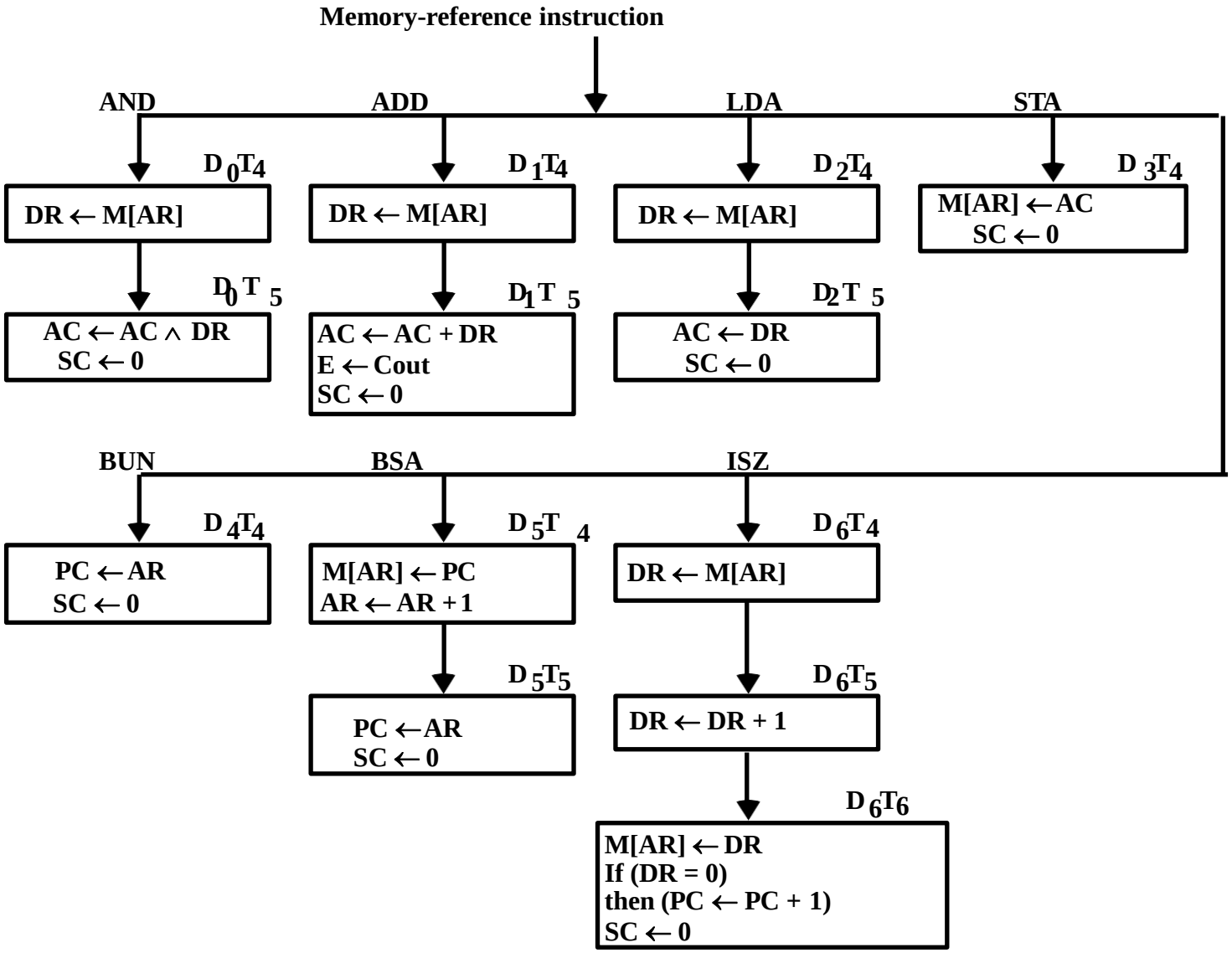


Figure : Flowchart for memory-reference instructions.

❖ Input-output instructions

- Input and output instructions are needed for transferring information to and from AC register, for 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 $D_7 = 1$ and I = 1.The remaining bits of the instruction specify the particular operation.
- The control functions and microoperations for the input-output instructions are listed in Table. D_7 I T_3 = p (common to all input-output instructions)

IR(i) = Bi [bit in IR (6-ll) that specifies the instruction]

	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

Instruction set completeness

- A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable.
- The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories:
  - **Functional Instructions**
    - Arithmetic, logic, and shift instructions
    - Example: ADD, complement AC(CMA), increment the accumulator AC (INC), circulate the AC right (CIR), circulate the accumulator left (CIL), AND, clear AC(CLA)
  - **Transfer Instructions**
    - Data transfers between the main memory and the processor registers
    - Examples: loading the AC(LDA), storing the accumulator (STA)

- **Control Instructions**
  - Program sequencing and control
  - Example: BUN, BSA, ISZ
  - The branch instructions BUN, BSA, and ISZ, together with the four skip instructions, provide capabilities for program control and checking of status conditions.
- **Input/Output Instructions**
  - Input and output
  - Example: INP, OUT
  - The input (INP) and output (OUT) instructions cause information to be transferred between the computer and external devices.

Timing and Control Unit

- 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 microoperations for the accumulator.
- **Control Unit:** Control unit (CU) of a processor translates from machine instructions to the control signals for the microoperations that implement them.

- Control units are implemented in one of two ways: **hardwired control and microprogrammed control.**

Hardwired control:

- The control logic is implemented with gates, flip-flops, decoders, and other digital circuits.
- It has the advantage that it can be optimized to produce a fast mode of operation.
- Requires changes in the wiring among the various components if the design has to be modified or changed, i.e. modification very difficult.
- Adding new feature is difficult.
- Expensive ,high error and used in RISC processor.

Microprogrammed control:

- The Control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations.
- In the microprogrammed control, any required changes or modifications can be done by updating the microprogram in control memory.
- Slow ,cheap
- Used in CISC processor.

……..

- The block Diagram of the hardware control unit is shown in figure below. It consists of two decoders, a sequence counter, and a number of control logic gates.
- An instruction read from memory is placed in the instruction register (IR)

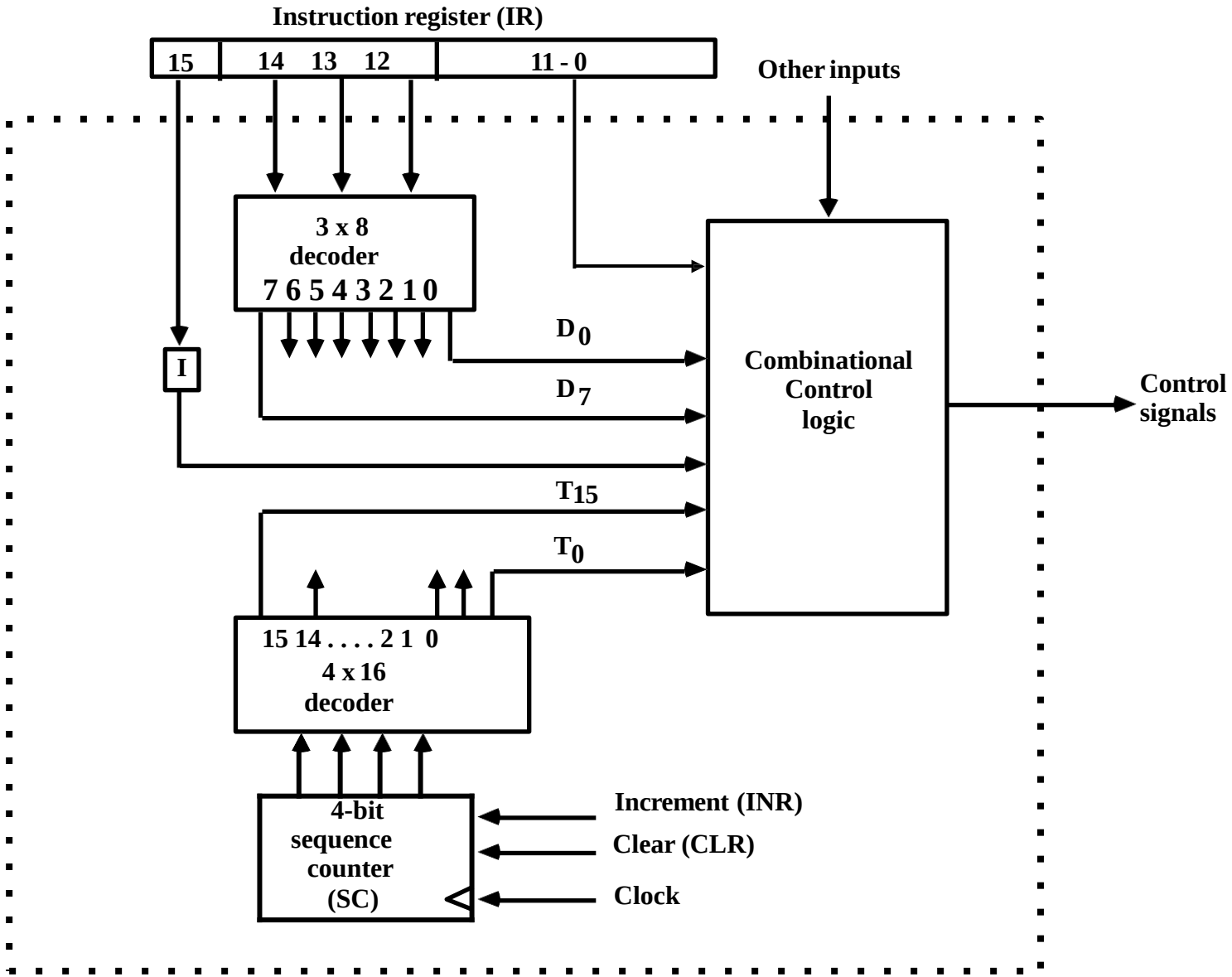


Figure : Hardwired Control organization.

- In an instruction register, instruction read from memory is sored where it is divided into three parts the I bit, the operation code and bits 0 through 11. The operation code in bits12 through 14 are decoded with a 3 x 8 decoder.
- The eight outputs of the decoder are designated by the binary value of the corresponding operation code. Bit 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.

Timing Signals

- Most of the time the counter is incremented to provide the sequence of timing signals out of the 4 x 16 decoder. Once in awhile, the counter is cleared to 0, causing the next active timing signal to be T_0.
- 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 $\leftarrow$ 0

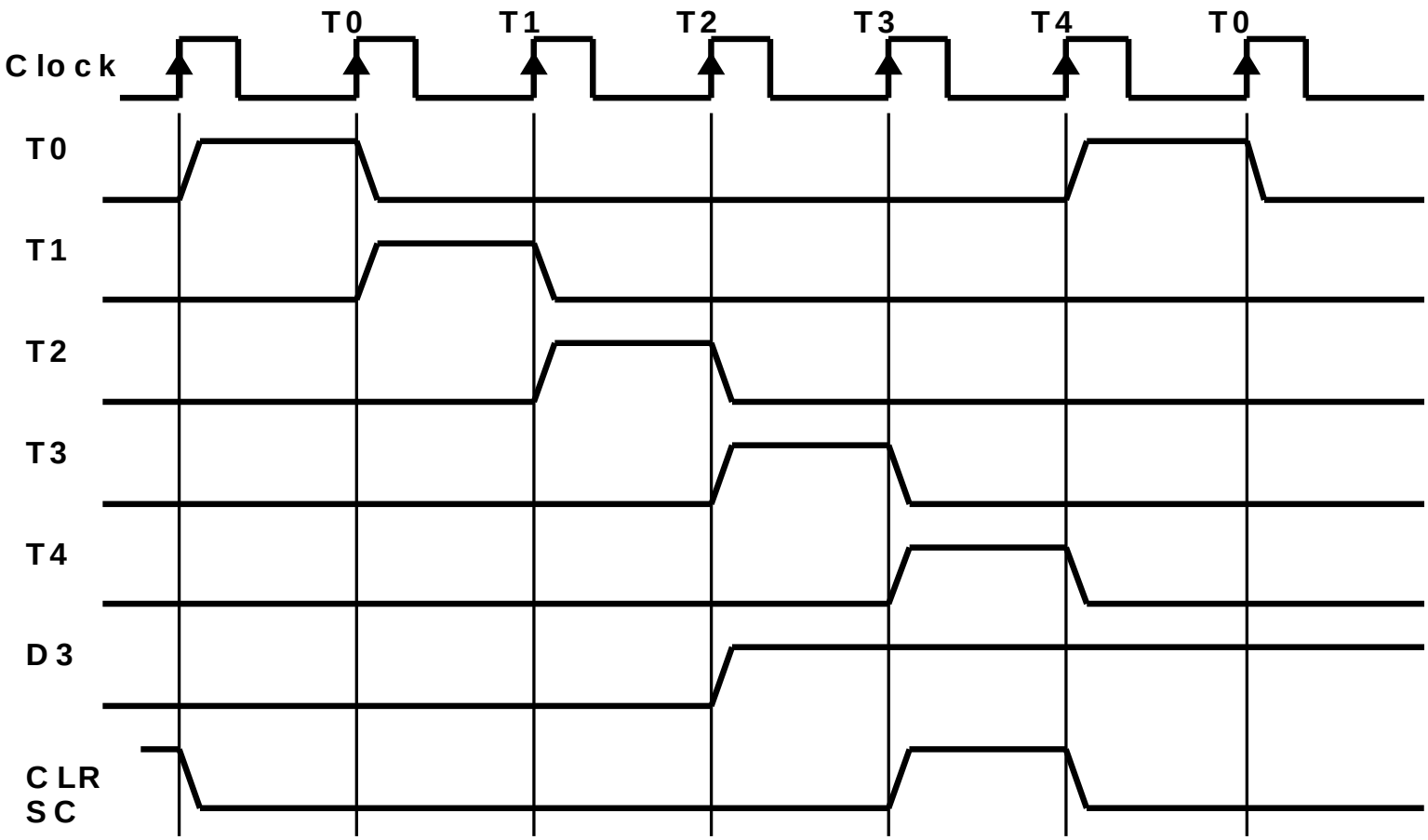


Figure : Example of control timing signals

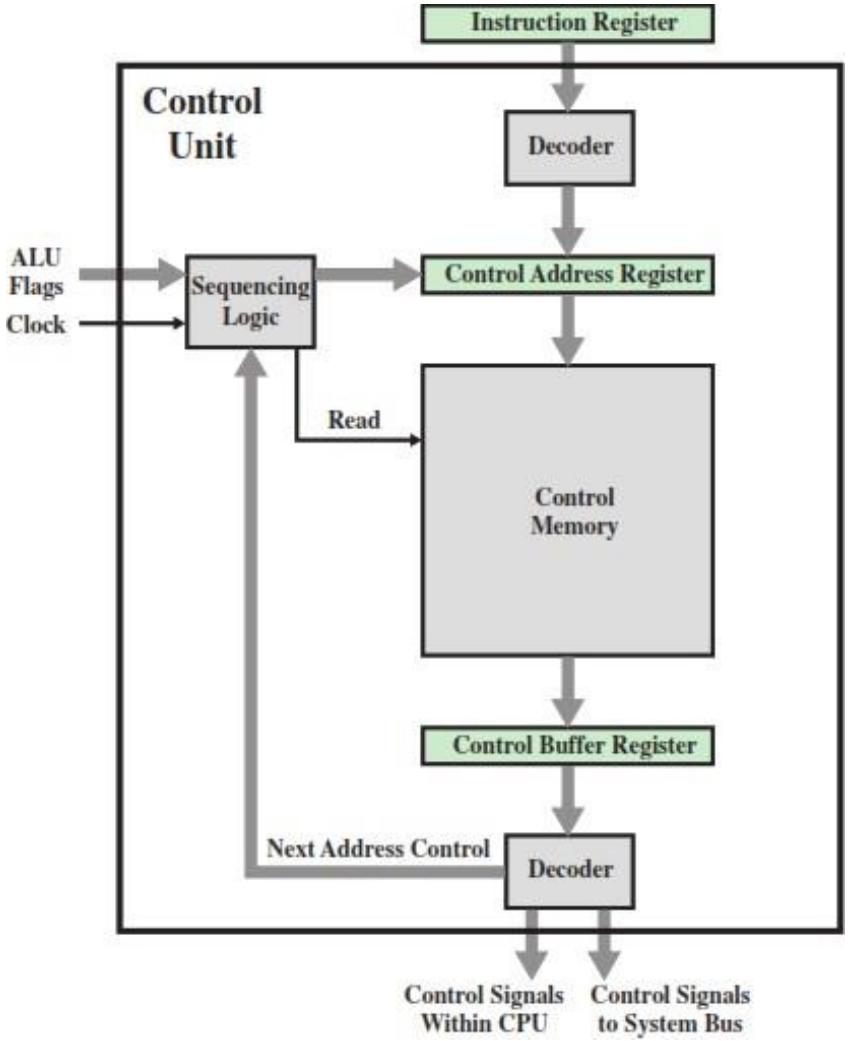


Figure: Functioning of Microprogrammed Control Unit

- The inputs (IR, ALU flags, clock) and outputs (control signals). The control unit functions as follows:
  - To execute an instruction, the sequencing logic unit issues a READ command to the control memory.
  - The word whose address is specified in the control address register is read into the control buffer register.
  - The content of the control buffer register generates control signals and next-address information for the sequencing logic unit.
  - The sequencing logic unit loads a new address into the control address register based on the next-address information from the control buffer register and the ALU flags.

Instruction Cycle

- A program residing in the memory unit of the computer consists of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction.

- In Basic Computer, a machine instruction is executed in the following cycle:

    1. Fetch an instruction from memory
    2. Decode the instruction
    3. Read the effective address from memory if the instruction has an indirect address
    4. Execute the instruction

- Upon the completion of step 4, the control goes back to step 1 to fetch, decode, and execute the next instruction. This process continues indefinitely unless a HALT instruction is encountered.

Note: Every different processor has its own (different) instruction cycle

Fetch and Decode

- Initially, the program counter PC is loaded with the address of the first instruction in the program.
- The sequence counter SC is cleared to 0, providing a decoded timing signal T_0, T_1, T_2, and so on.
- The microoperations for the fetch and decode phases can be specified by the following register transfer statements.

T_0: AR←PC

T_1: IR←M[AR],PC←PC +1

T_2: D_0, ….D_7←decode IR (12-14) ,AR←IR (0-11) ,I←IR(15)

- Since only AR is connected to the address inputs of memory, it is necessary to transfer the address from PC to AR during the clock transition associated with timing signal T_0.
- The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T_1.
- At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.
- At time T_2, the operation code in IR is decoded, the indirect bit is transferred to flip-flop I, and the address part of the instruction is transferred to AR.

Note: The sequence counter SC incremented after each clock pulse to produce the sequence T_0, T_1 and T_2.

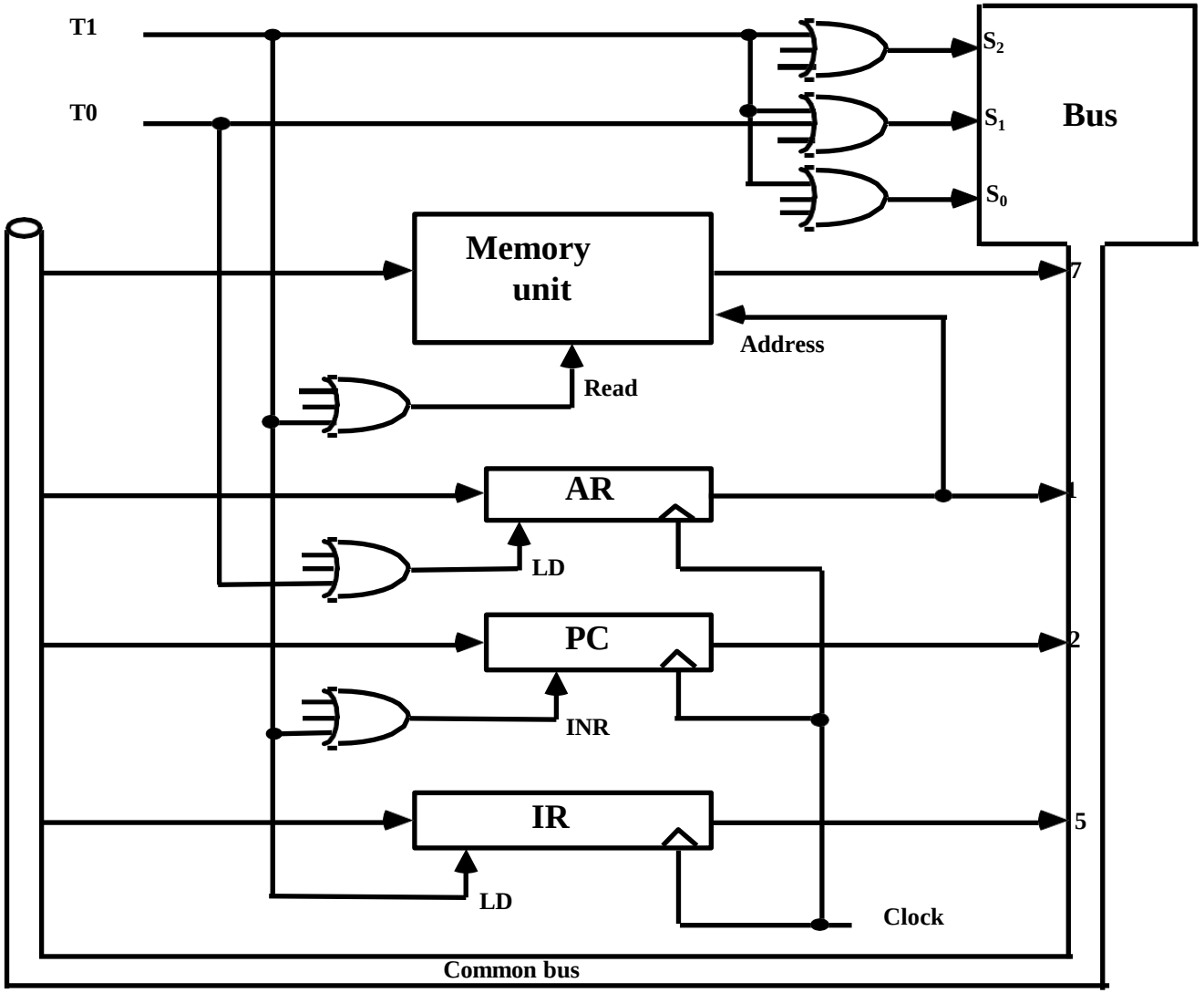


Figure : Register transfers for the fetch phase.

T_0 to achieve the following connection:

1.Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.

2. Transfer the content of the bus to AR by enabling the LD input of AR.

The next clock transition initiates the transfer from PC to AR since T_0=1. In order to implement the second statement

T_1: IR <- M[AR], PC <- PC + 1

it is necessary to use timing signal T1 to provide the following connections in the bus system.

1. Enable the read input of memory.

2. Place the content of memory onto the bus by making $S_2 S_1S_0$ = 1ll.

3. Transfer the content of the bus to IR by enabling the LD input of IR

4. Increment PC by enabling the INR input of PC .

Determine the type of instruction

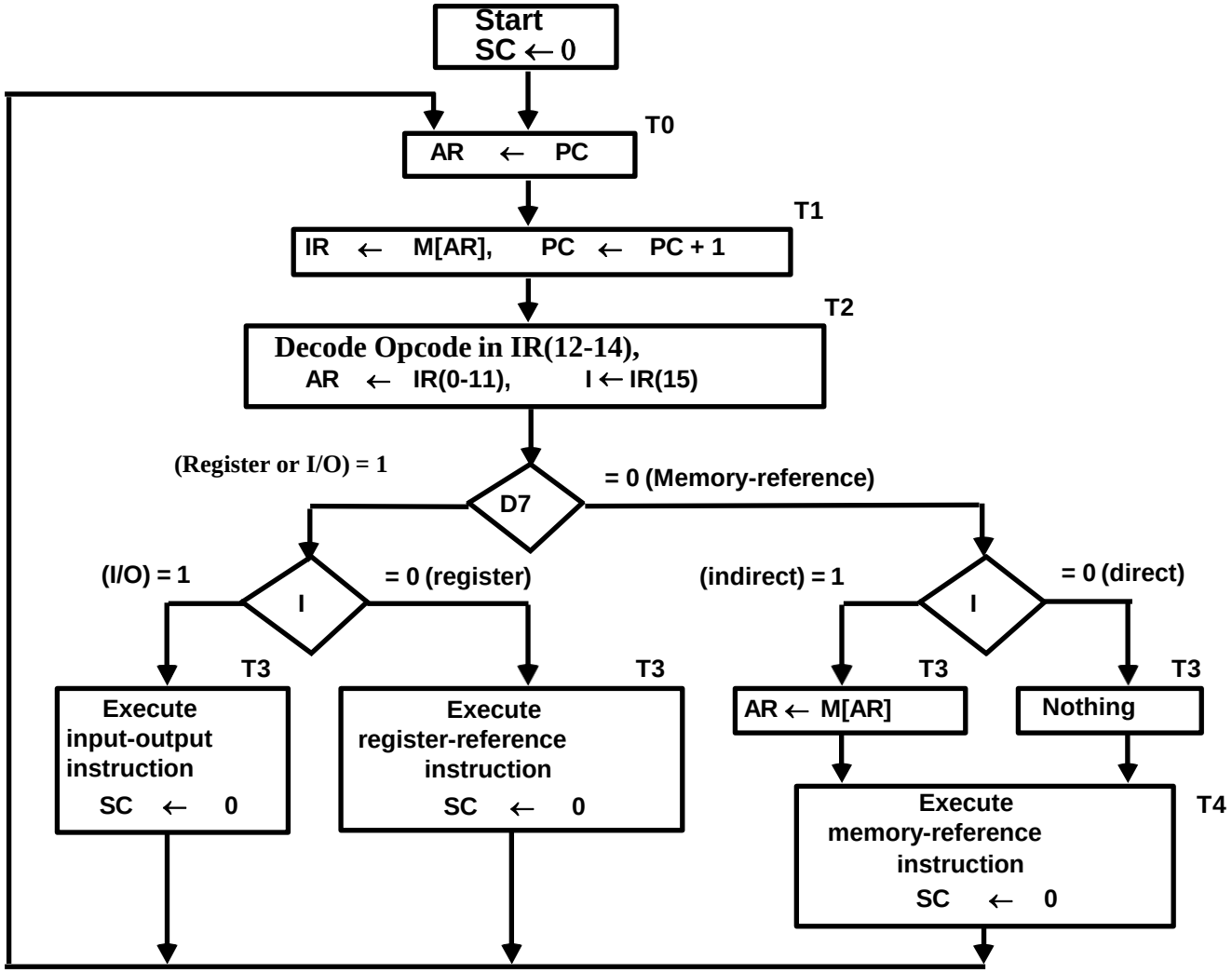


Figure : Flowchart for instruction cycle (initial configuration).

- The timing signal that is active after the decoding is T3. During time T3 the control unit determines the type of instruction that was just read from memory.
- The flowchart of figure above presents an initial configuration for the instruction cycle and shows how the control determines the instruction type after the decoding.
- We determine that if $D_7 = 1$, the instruction must be a register-reference or input output type. If $D_7 = 0$, the operation code must be one of the other seven values 000 through 110, specifying a memory reference instruction.
- The three instruction types are subdivided into four separate paths. The selected operation is activated with the clock transition associated with timing signal T3. This can be symbolized as follows:

  $D_7 I T_3$: AR + M[AR]

  $D_7 I' T_3$: Nothing

  $D_7 I' T_3$: Execute a register-reference instruction

  $D_7 I T_3$: Execute an input-output instruction

Input-Output and Interrupt

- Instructions and data stored in memory must come from some input device. Computational results must be transmitted to the user through some output device. Commercial computers include many types of input and output devices.

Input-Output Configuration

- The terminal sends and receives serial information. Each quantity of information has eight bits of an alphanumeric code.
- The two registers communicate with a communication interface serially and with the AC in parallel.
- 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.

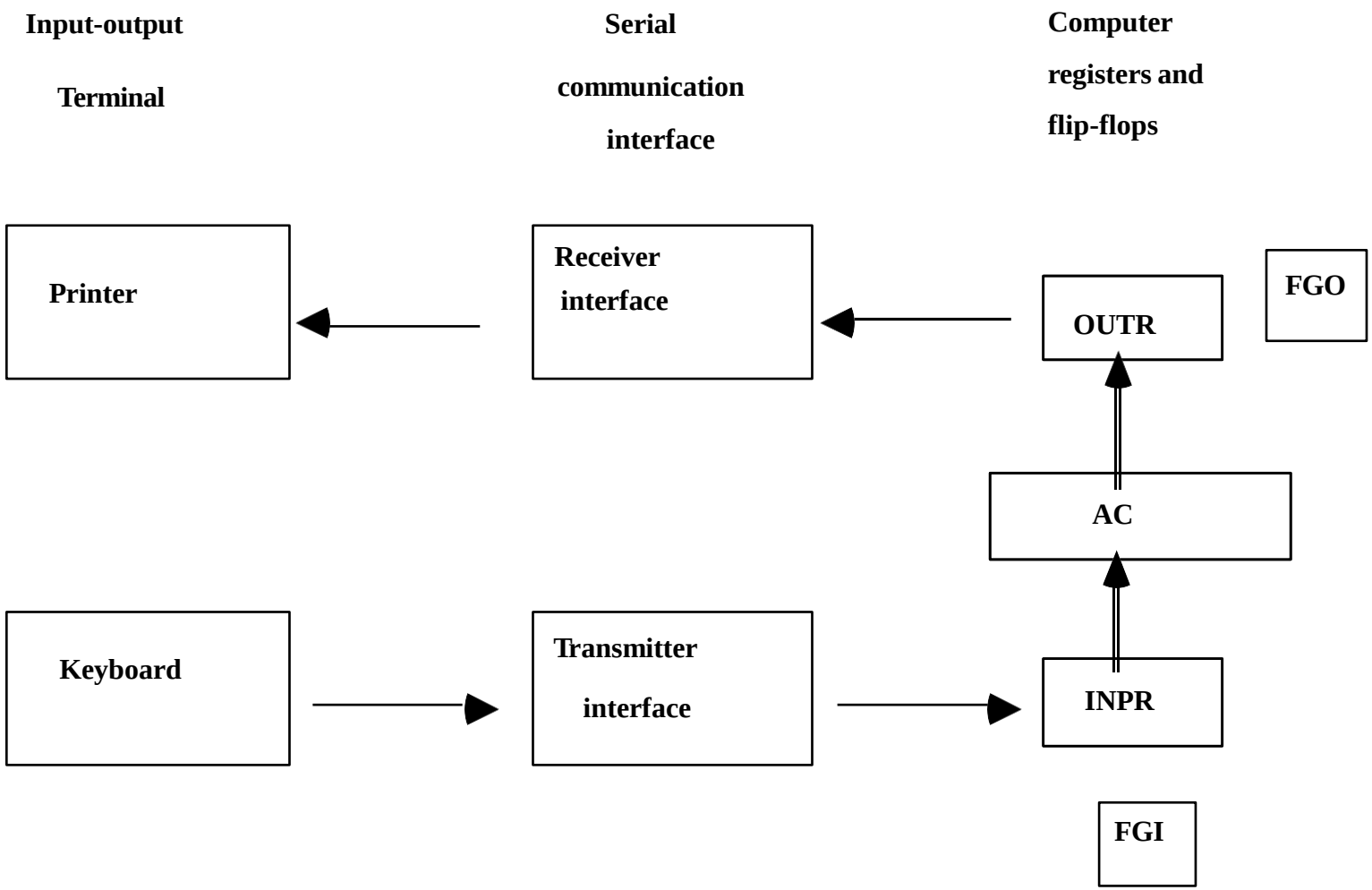
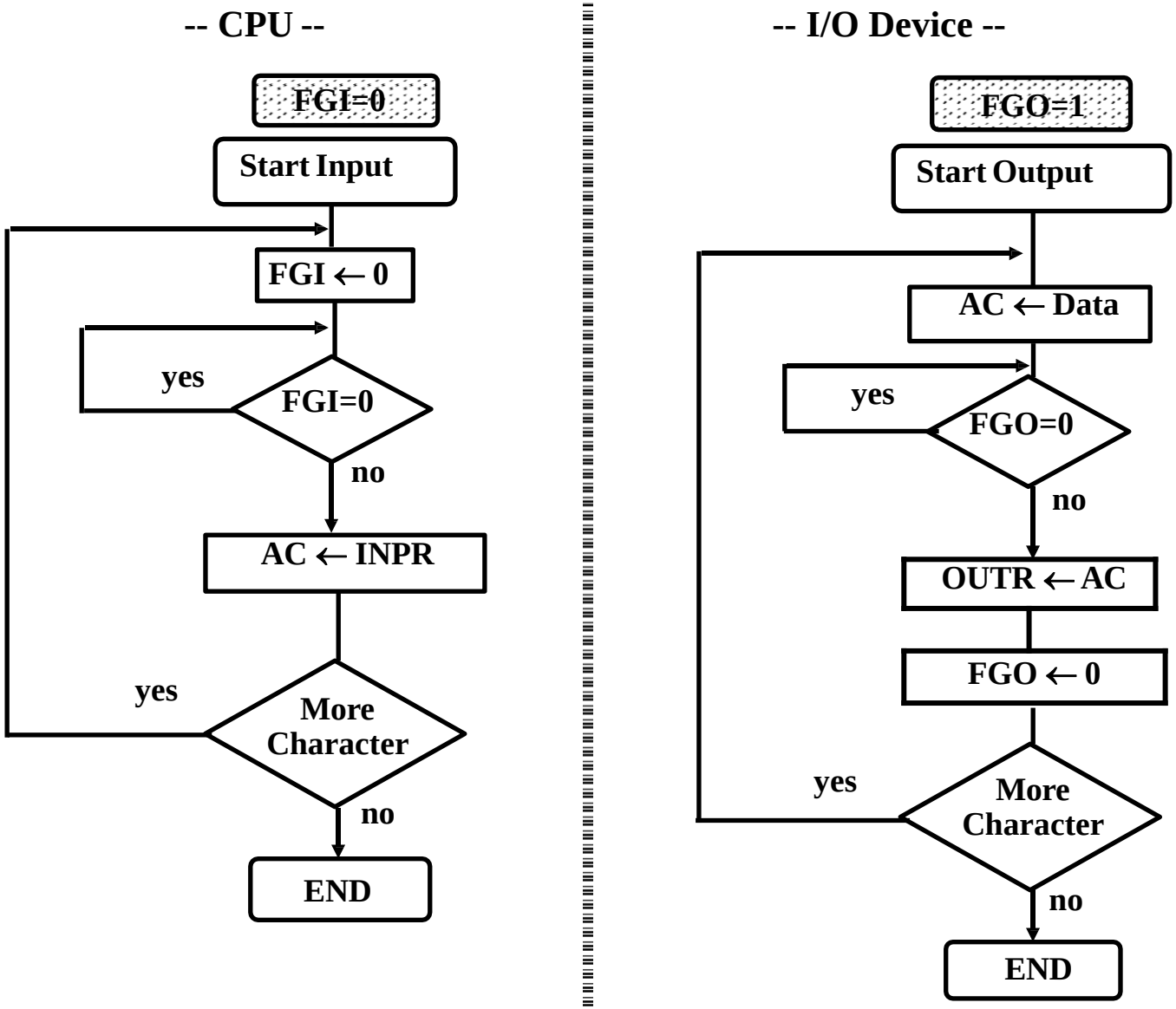


Figure : Input-output configuration

- INPR Input register - 8 bits, OUTR Output register - 8 bits, FGI Input flag - 1 bit, FGO Output flag - 1 bit, IENInterrupt enable - 1 bit
- Initially, the input flag FGI is cleared to 0. When a key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and the input flag FGI is set to 1. As long as the flag is set, the information in INPR cannot be changed by striking another key. The computer checks the flag bit; if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.
- The output register OUTR works similarly but the direction of information flow is reversed. Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.

Program controlled data transfer



Program Interrupt

- The process of communication just described is referred to as programmed control transfer. The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.
- Consider a computer that can transfer information at a maximum rate of 10 characters per second. This is equivalent to one character every 100,000µs. Two instructions are executed when the computer checks the flag bit and decides not to transfer the information. This means that at the maximum rate, the computer will check the flag 50000 times between each transfer. The computer is wasting time while checking the flag instead of doing some other useful processing task.
- The interrupt enable flip-flop IEN can be set and cleared with two instructions. When IEN is cleared to 0 the flags cannot interrupt the computer.

- When IEN is set to 1 the computer can be interrupted. These two instructions provide the programmer with the capability of making a decision as to whether or not to use the interrupt facility.

Interrupt Cycle

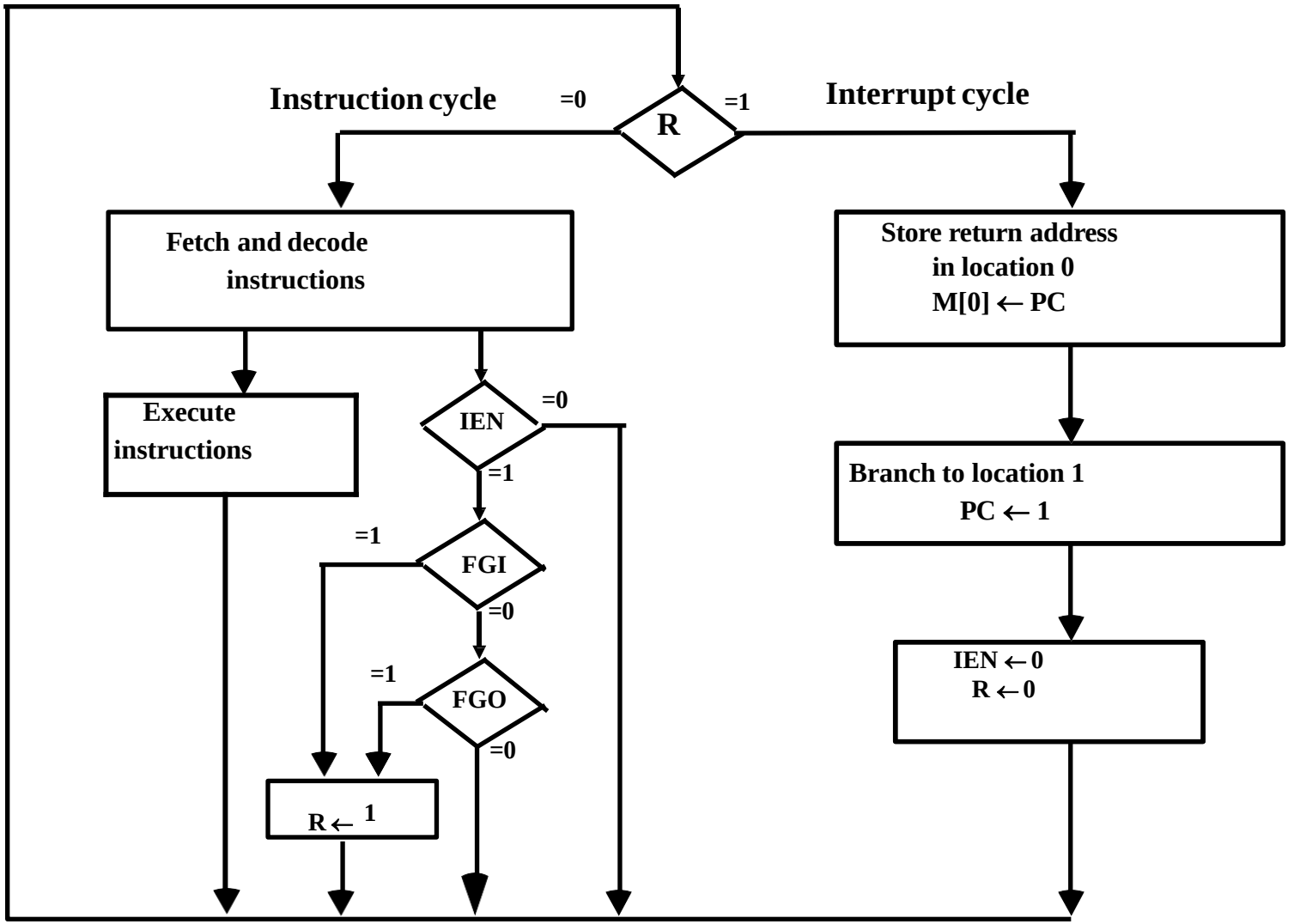


Figure : Flowchart for interrupt cycle

- The interrupt cycle is a HW implementation of a branch and save return address operation.
- At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1.
- At memory address 1, the programmer must store a branch instruction that sends the control to an interrupt service routine
- The instruction that returns the control to the original program is "indirect BUN 0"

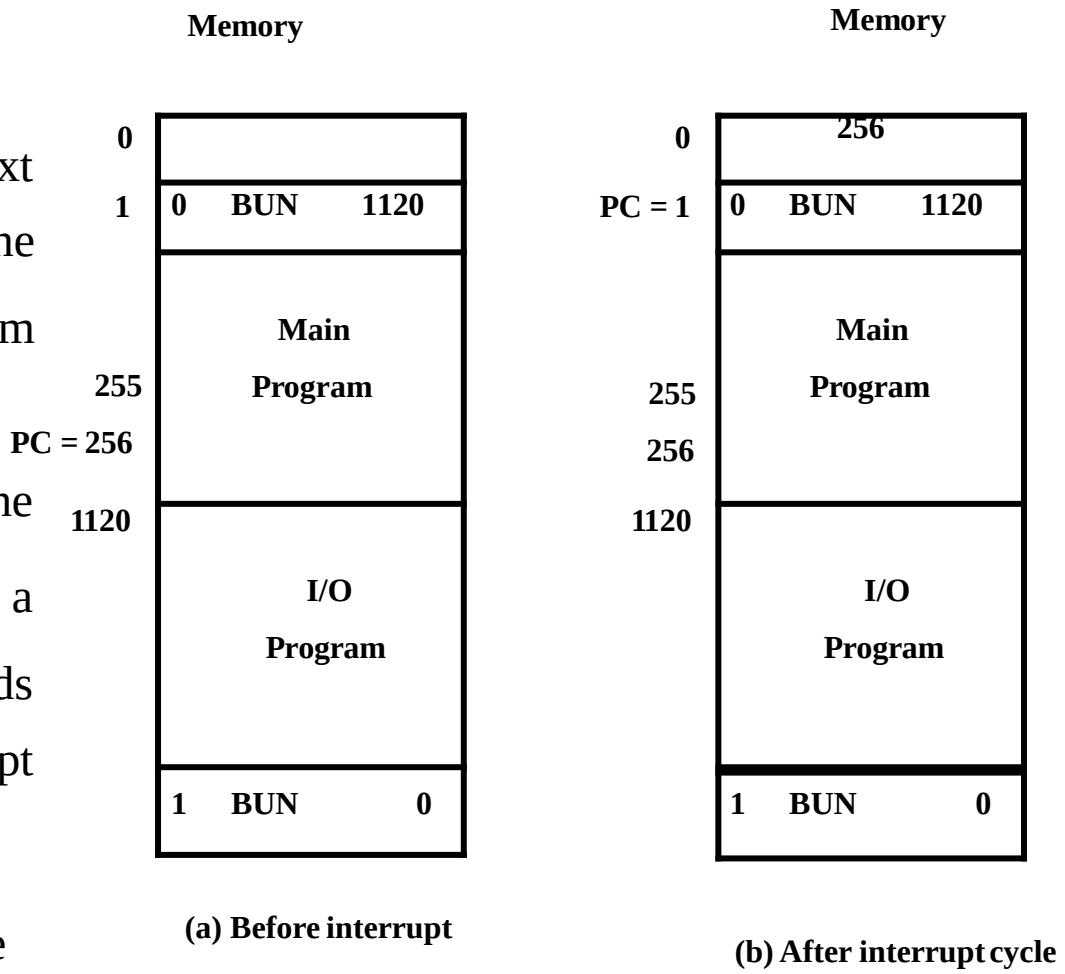


Figure : Demonstration of the interrupt cycle

- Register Transfer Statements for Interrupt Cycle, The interrupt cycle is initiated after the last execute phase if the interrupt flip-flop R is equal to 1. This flip-flop is set to 1 if IEN = 1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T_0, T_1 or T_2 are active.

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

- The fetch and decode phases of the instruction cycle must be modified: Replace T_0, T_1, T_2 with $R'T_0$, $R'T_1$, $R'T_2$
- The interrupt cycle :

  RT_0: $AR \leftarrow 0$, $TR \leftarrow PC$

  RT_1: $M[AR] \leftarrow TR$, $PC \leftarrow 0$

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

Complete Computer Description

Design of Basic computer :

The basic computer consists of the following hardware components:

- A memory unit with 4096 words of 16 bits each
- Nine registers : AR, PC, DR, AC, IR, TR. OUTR, INPR, and SC
- Seven flip-flop: I, S, E,R, IEN. FGI, and FGO
- Two decoders: a 3 x 8 operation decoder and a 4 x 16 timing decoder
- A 16 bit common bus.
- Control logic gates.
- Adder and logic circuit connected to the input of AC

Flowchart

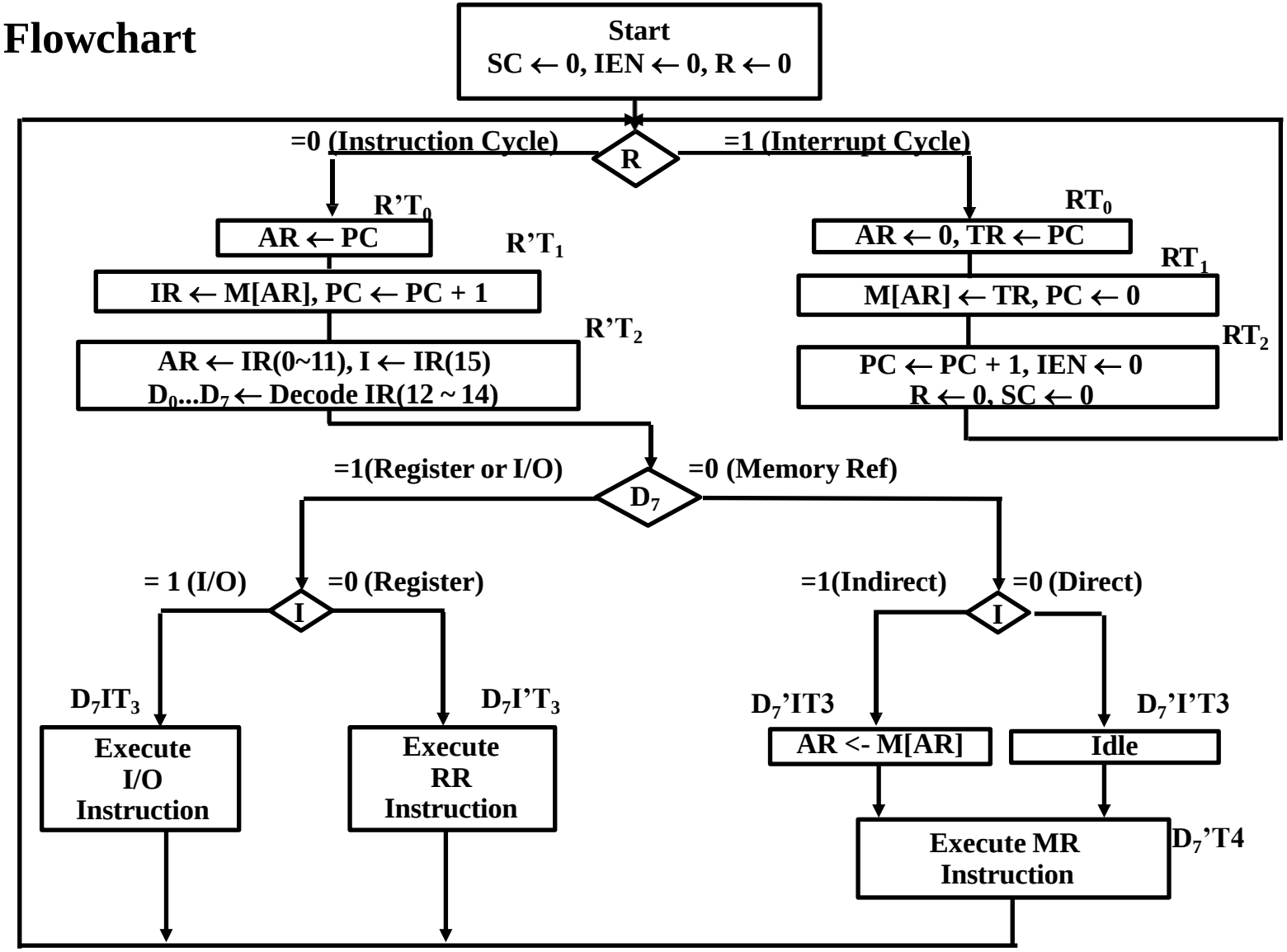


Figure : Flowchart for computer operation.

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC + 1$
Decode	$R'T_2$:	$D0, ..., D7 \leftarrow$ Decode $IR(12 \sim 14)$, $AR \leftarrow IR(0 \sim 11), I \leftarrow IR(15)$
Indirect	$D_7I'T_3$:	$AR \leftarrow M[AR]$
Interrupt		
	$T_0'T_1'T_2'(IEN)(FGI + FGO)$:	$R \leftarrow 1$
	RT_0:	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC + 1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory-Reference		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$
ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR, SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC, SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR, SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC, AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR, SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$, if(DR=0) then $(PC \leftarrow PC + 1)$, $SC \leftarrow 0$

Register-Reference		
	$D_7IT_3 = r$	(Common to all register-reference instr)
	$IR(i) = B_i$	(i = 0,1,2, ..., 11)
	r:	$SC \leftarrow 0$
CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow shr\ AC,\ AC(15) \leftarrow E,\ E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC,\ AC(0) \leftarrow E,\ E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	$If(AC(15) = 0)\ then\ (PC \leftarrow PC + 1)$
SNA	rB_3:	$If(AC(15) = 1)\ then\ (PC \leftarrow PC + 1)$
SZA	rB_2:	$If(AC = 0)\ then\ (PC \leftarrow PC + 1)$
SZE	rB_1:	$If(E = 0)\ then\ (PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$
Input-Output	$D_7IT_3 = p$	(Common to all input-output instructions)
	$IR(i) = B_i$	(i = 6,7,8,9,10,11)
	p:	$SC \leftarrow 0$
INP	pB_{11}:	$AC(0\text{-}7) \leftarrow INPR,\ FGI \leftarrow 0$
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{-}7),\ FGO \leftarrow 0$
SKI	pB_9:	$If(FGI = 1)\ then\ (PC \leftarrow PC + 1)$
SKO	pB_8:	$If(FGO = 1)\ then\ (PC \leftarrow PC + 1)$
ION	pB_7:	$IEN \leftarrow 1$
IOF	pB_6:	$IEN \leftarrow 0$

Control Logic Gates

Inputs:

- The inputs from the two decoders,
- I flip-flop
- IR-bits (0-11)
- AC bits 0 -15 to check if AC = 0 and to detect the sign bit in AC(15)
- DR bits 0 -15 to check if DR 0 and
- the values of the seven flip-flops.

Outputs:

- Signals to control the inputs of the nine registers
- Signals to control the read and write inputs of memory
- Signals to set, clear, or complement the flip-flops
- Signals for 52, 51, and 50 to select a register for the bus
- Signals to control the AC adder and logic circuit

Control of registers and memory

- The control inputs of the registers are LD (load), INR (increment), and CLR (clear). Suppose that we want to derive the gate structure associated
- Address Register; AR, Scan all of the register transfer statements that change the content of AR:

$R'T_0$: $AR \leftarrow PC$ LD(AR)

$R'T_2$: $AR \leftarrow IR(0\text{-}11)$ LD(AR)

D'_7IT_3: $AR \leftarrow M[AR]$ LD(AR)

RT_0: $AR \leftarrow 0$ CLR(AR)

D_5T_4: $AR \leftarrow AR + 1$ INR(AR)

$$LD(AR) = R'T_0 + R'T_2 + D'_7IT_3$$

$$CLR(AR) = RT_0$$

$$INR(AR) = D_5T_4$$

- where LD(AR) is the load input of AR, CLR(AR) is the clear input of AR, and INR(AR) is the increment input of AR . The control gate logic associated with AR is shown in Figure.
- In a similar fashion we can derive the control gates for the other registers as well as the logic needed to control the read and write inputs of memory. The read operation is recognized from the symbol <-M[AR].

$$Read = R'T_1 + D'_7IT_3 + (D_0 + D_1 + 0_2 + D_6)T_4$$

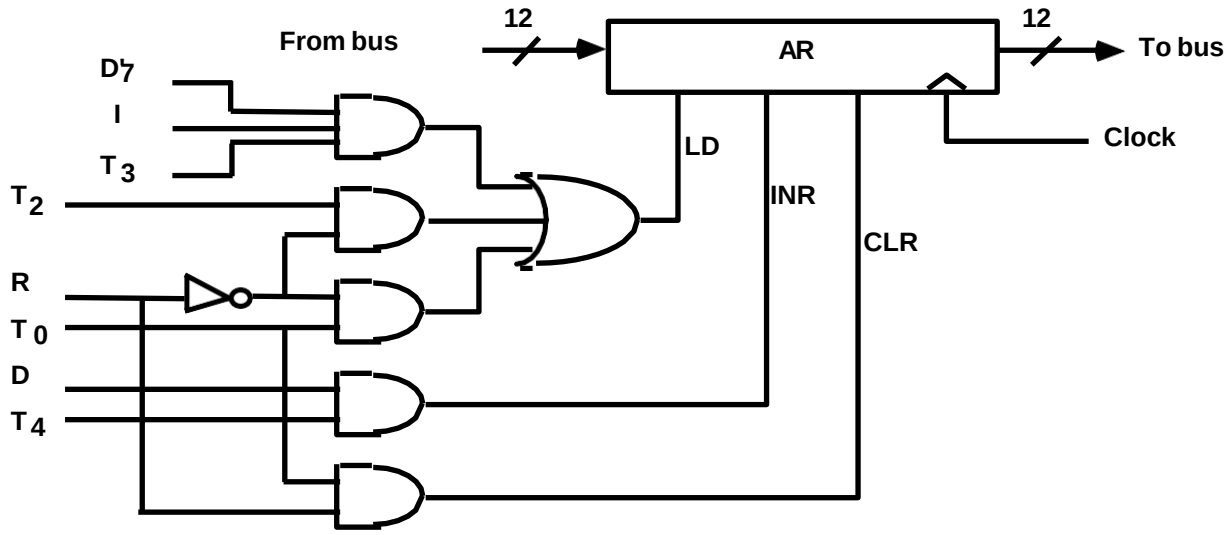


Figure : Control gates associated withAR.

Control of Flags

- The control gates for the seven flip-flops can be determined in a similar manner

•IEN: Interrupt Enable Flag pB_7:

IEN ← 1 (I/O Instruction)

pB_6: IEN ← 0 (I/O Instruction)

RT_2: IEN ← 0 (Interrupt) where, $p = D_7IT_3$ (Input/Output Instruction) and B, and B, are bits 7 and 6 of IR.

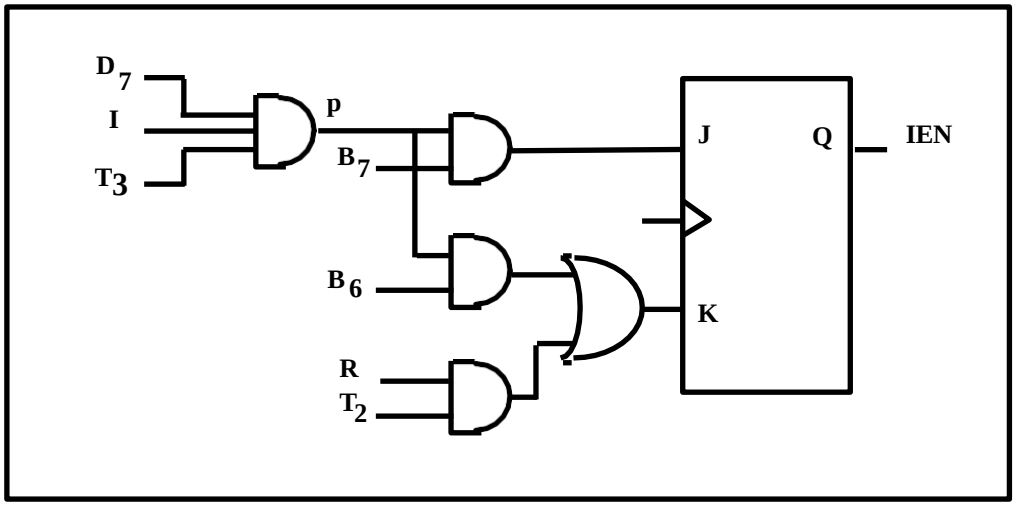


Figure : Control inputs for IEN.

Control of Common Bus

- The 16-bit common bus is controlled by the selection inputs S2, S1, and S_0 The decimal number shown with each bus input specifies the equivalent binary number that must be applied to the selection inputs in order.
- Each binary number is associated with a Boolean variable x_1 through x_7, corresponding to the gate structure that must be active in order to select the register or memory for the bus.

Inputs							Outputs			Register selected for 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	0	0	0	1	*AR*
0	1	0	0	0	0	0	0	1	0	*PC*
0	0	1	0	0	0	0	0	1	1	*DR*
0	0	0	1	0	0	0	1	0	0	*AC*
0	0	0	0	1	0	0	1	0	1	*IR*
0	0	0	0	0	1	0	1	1	0	*TR*
0	0	0	0	0	0	1	1	1	1	Memory

Table : Encoder for Bus Selection Circuit

- For example, to find the logic that makes $x_1 = I$, we scan all register transfer statements that have AR as a source.

  D_4T_4: PC $\leftarrow$ AR
  D_5T_5: PC $\leftarrow$ AR

Therefore, the Boolean function for x_1 is $x_1 = D_4T_4 + D_5T_5$

- Similarly, The data output from memory are selected for the bus when x7 = 1 and $S_2\,S_1\,S_0 = 1ll$. The gate logic that generates x_7 must also be applied to the read input of memory. Therefore, the Boolean function for x7 is the same as the one derived previously for the read operation.

  $x_7 = R'T_4 + D_7'IT_3 + (D_0 + D_1 + D_2 + D_6)T_4$

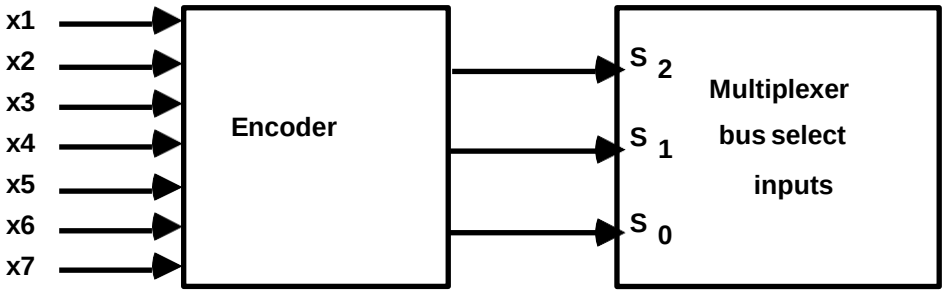


Figure :Encoder for bus selection inputs.

Design of Accumulator Logic

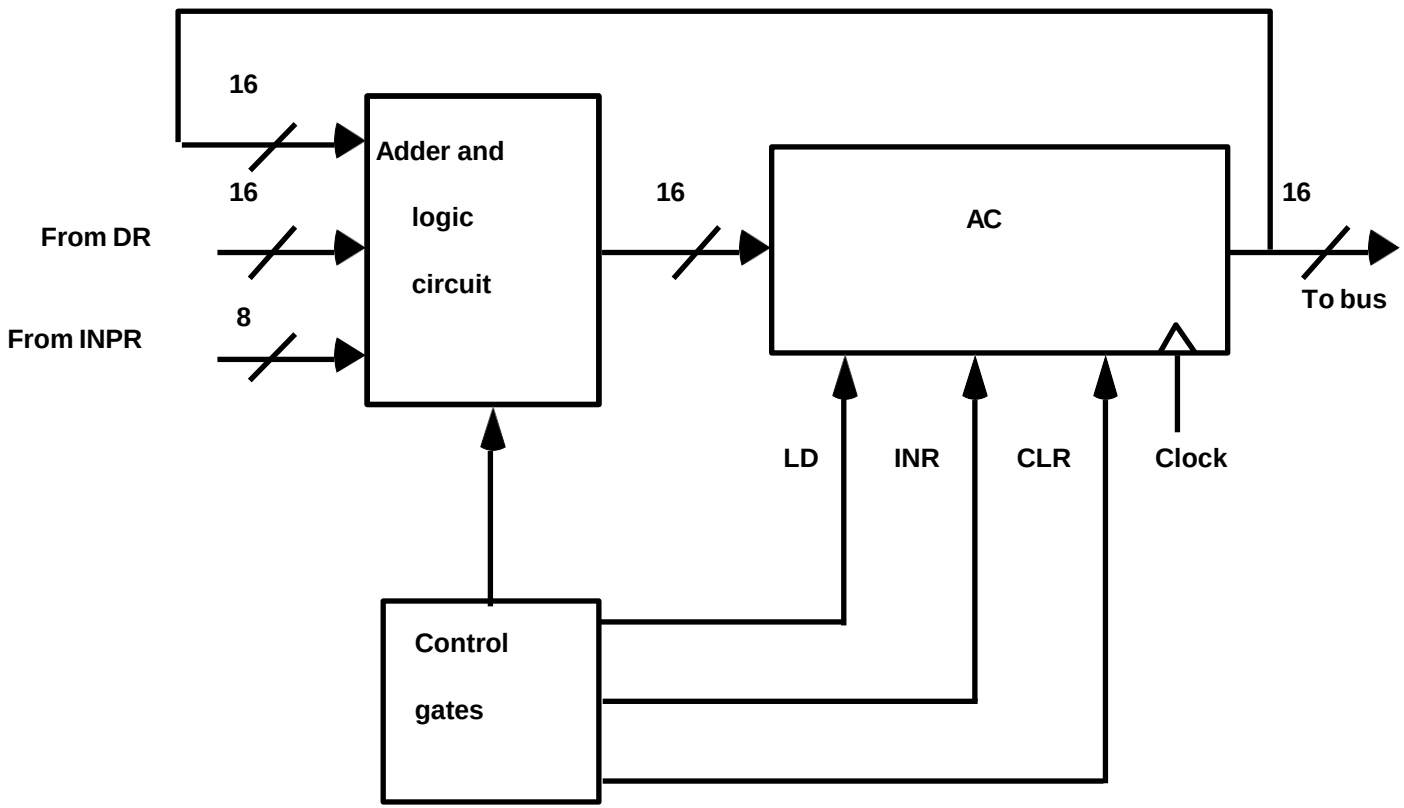


Figure : Circuits associated withAC.

D_0T_5: AC ← AC ∧ DR	AND with DR
D_1T_5: AC ← AC + DR	Add with DR
D_2T_5: AC ← DR	Transfer from DR
pB_{11}: AC(0-7) ← INPR	Transfer from INPR
rB_9: AC ← AC′	Complement
rB_7 : AC ← shr AC, AC(15) ← E	Shift right
rB_6 : AC ← shl AC, AC(0) ← E	Shift left
rB_{11} : AC ← 0	Clear
rB_5 : AC ← AC + 1	Increment

Control of AC register

- Gate structures for controlling the LD, INR, and CLR of AC show in below:

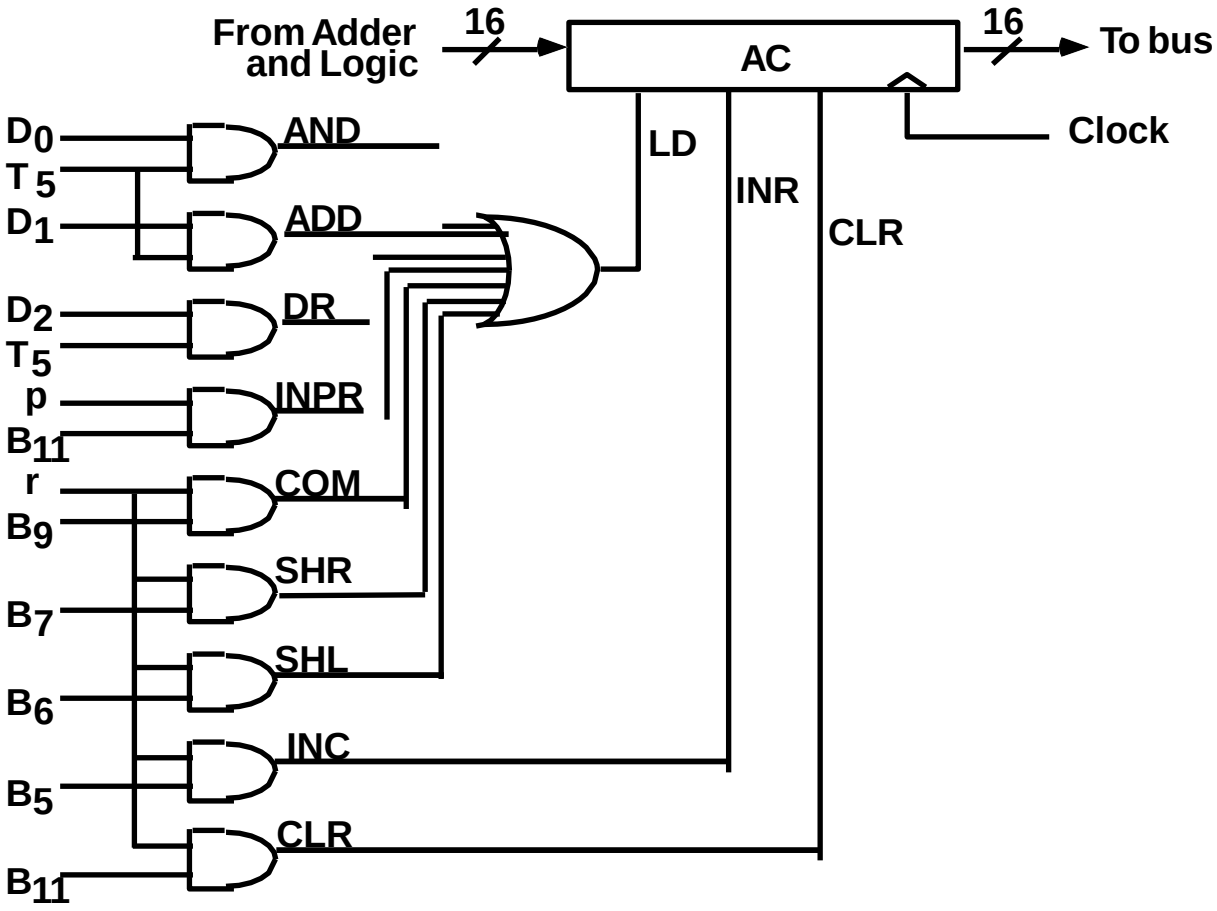


Figure : Gate structure for controlling the LD, INR, and CLR ofAC.

Adder and Logic circuit

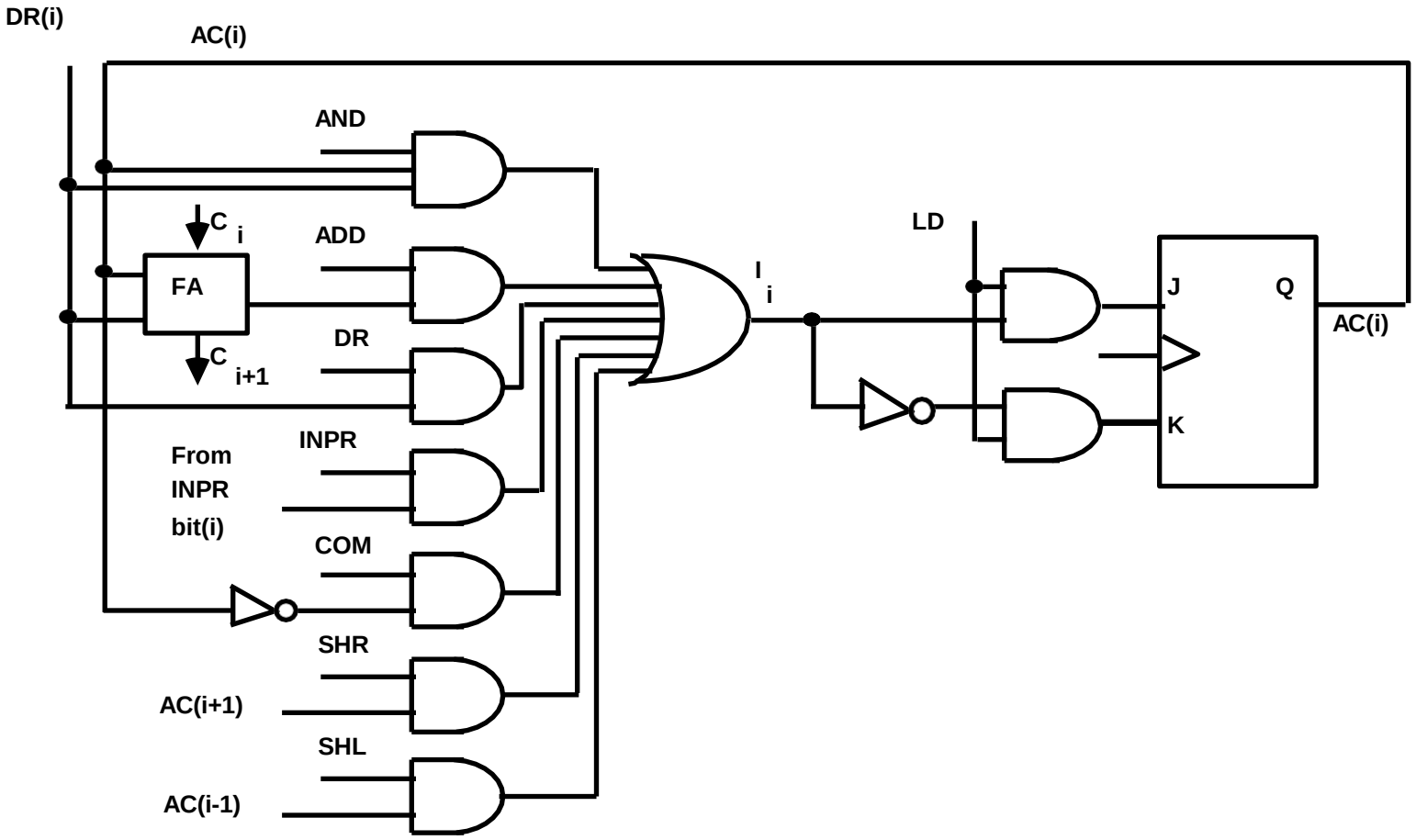


Figure : One stage of adder and logic circuit.