UNIT – 3 : BASIC COMPUTER ORGANIZATION AND DESIGN

Introduction:

Computer organization refers to the operational unit and their interconnection that realise the architectural specification.

Computer organization deals with how different part of a computer are organised and how various operations are performed between different part to do a specific task.

The organization of the computer is defined by its internal registers, timing and control structure, and set of instruction that is uses.

Von Neumann Architecture vs. Harvard Architecture:

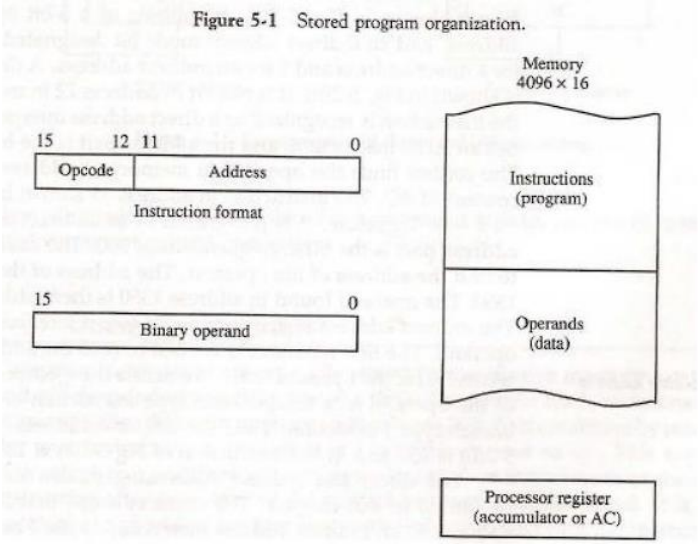
Von Neumann Architecture	Harvard Architecture
It is ancient computer architecture based on stored program computer concept.	It is modern computer architecture based on Harvard Mark I relay based model.
Same physical memory address is used for instructions and data.	Separate physical memory address is used for instructions and data.
There is common bus for data and instruction transfer.	Separate buses are used for transferring data and instruction.
Two clock cycles are required to execute single instruction.	An instruction is executed in a single cycle.
It is cheaper in cost.	It is costly than Von Neumann Architecture.
CPU cannot access instructions and read/write at the same time.	CPU can access instructions and read/write at the same time.
It is used in personal computers and small computers.	It is used in micro controllers and signal processing.

Stored Program Organization:

The ability to store and execute instructions is the most important property of a general-purpose computer. That type of stored program concept is called **stored program organization.**

The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. The first part specifies the operation to be performed and the second specifies an address.

The below figure shows the stored program organization –



Instructions are stored in one section of memory and data in another.

For a memory unit with 4096 words, we need 12 bits to specify an address since 2^{12} = 4096.

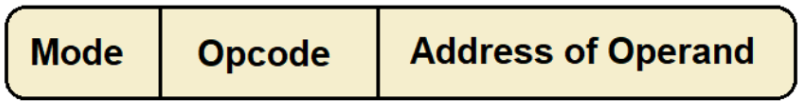
If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (abbreviated opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.

Instruction Codes:

Instruction codes are bits that instruct the computer to execute a specific operation. An instruction comprises groups called fields. These fields include:

An instruction comprises groups called fields. These fields include:

- The Operation code (Opcode) field determines the process that needs to perform.
- The Address field contains the operand's location, i.e., register or memory location.
- The Mode field specifies how the operand locates.



Instruction Format

Structure of an Instruction Code:

The instruction code is also known as an instruction set. It is a collection of binary codes. It represents the operations that a computer processor can perform. The structure of an instruction code can vary. It depends on the architecture of the processor but generally consists of the following parts:

- **Opcode:** The opcode (Operation code) represents the operation that the processor must perform. It might indicate that the instruction is an arithmetic operation such as addition, subtraction, multiplication, or division.
- **Operand(s):** The operand(s) represents the data that the operation must be performed on. This data can take various forms, depending on the processor's architecture. It might be a register containing a value, a memory address pointing to a location in memory where the data is stored, or a constant value embedded within the instruction itself.
- **Addressing mode:** The addressing mode represents how the operand(s) can be interpreted. It might indicate that the operand is a direct address in memory, an indirect address (i.e. a memory address stored in a register), or an immediate value (i.e. a constant embedded within the instruction).
- **Prefixes or modifiers:** Some instruction sets may include additional prefixes or modifiers that modify the behavior of the instruction. For example, they may specify that the operation should be performed only if a specific condition is met or that the instruction should be executed repeatedly until a specific condition is met.

Types of Instruction Code:

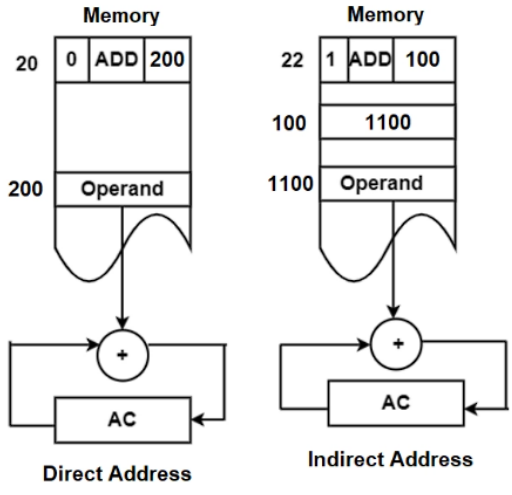
There are various types of instruction codes. They are classified based on the number of operands, the type of operation performed, and the addressing modes used. The following are some common types of instruction codes:

1. **One-operand instructions:** These instructions have one operand and perform an operation on that operand. For example, the "neg" instruction in the x86 assembly language negates the value of a single operand.
2. **Two-operand instructions:** These instructions have two operands and perform an operation involving both. For example, the "add" instruction in x86 assembly language adds two operands together.
3. **Three-operand instructions:** These instructions have three operands and perform an operation that involves all three operands. For example, the "fma" (fused multiply-add) instruction in some processors multiplies two operands together, adds a third operand, and stores the result in a fourth operand.

4. **Data transfer instructions:** These instructions move data between memory and registers or between registers. For example, the "mov" instruction in the x86 assembly language moves data from one location to another.
5. **Control transfer instructions:** These instructions change the flow of program execution by modifying the program counter. For example, the "jmp" instruction in the x86 assembly language jumps to a different location in the program.
6. **Arithmetic instructions:** These instructions perform mathematical operations on operands. For example, the "add" instruction in x86 assembly language adds two operands together.
7. **Logical instructions:** These instructions perform logical operations on operands. For example, the "and" instruction in x86 assembly language performs a bitwise AND operation on two operands.
8. **Comparison instructions:** These instructions compare two operands and set a flag based on the result. For example, the "cmp" instruction in x86 assembly language compares two operands and sets a flag indicating whether they are equal, greater than, or less than.
9. **Floating-point instructions:** These instructions perform arithmetic and other operations on floating-point numbers. For example, the "fadd" instruction in the x86 assembly language adds two floating-point numbers together.

Accumulator Register (AC): This register is found in single register processors (AC), and it performs all operations with memory operands.

Effective Address (EA) is the address of the operand or the target address. It defines the address that we can execute as a target address for a branch-type instruction or the address we can use directly to create an operand for a computation-type instruction without any changes.



Computer Registers:

Computer registers are memory storing units that operate at high speed. It's a component of a computer's processor. It can hold any type of data, including a bit sequence or a single piece of data.

Eight registers, a memory unit, and a control unit make up a basic computer. These devices must be connected on a regular basis.

Following is the list of some of the most common registers in computer:

TABLE 5-1 List of Registers for the Basic Computer

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

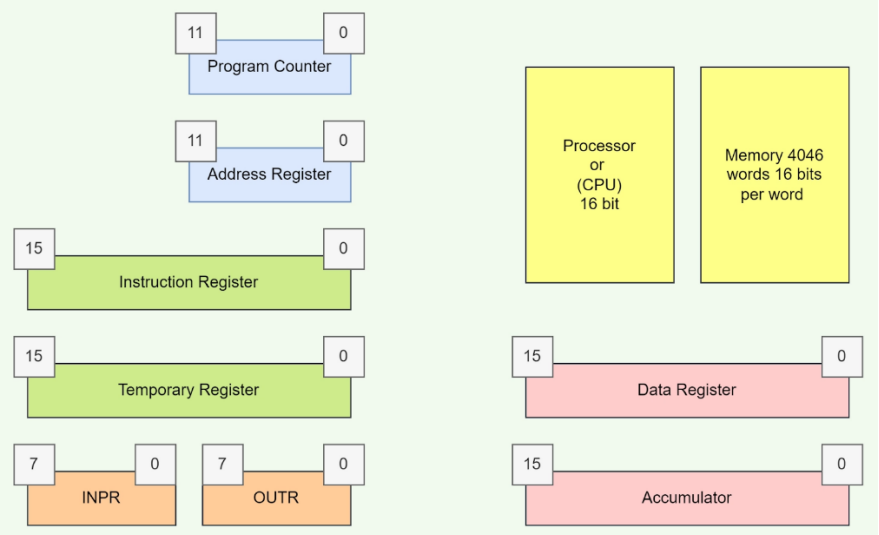
Types of Registers Used in Computer:

Registers are a type of computer memory used to accept, store, and transfer data and instructions used by the CPU right away. Processor registers refer to the registers used by the CPU.

During the execution of a program, registers are used to store data temporarily.

In most cases, the number of bits that a register can hold is used to determine its size.

The common registers in a computer and the memory are depicted in the diagram below:



The following are the various computer registers and their functions:

- **Accumulator Register (AC):** Accumulator Register is a general-purpose Register. The initial data to be processed, the intermediate result, and the final result of the processing operation are all stored in this register. If no specific address for the result operation is specified, the result of arithmetic operations is transferred to AC. The number of bits in the accumulator register equals the number of bits per word.
- **Address Register (AR):** The Address Register is the address of the memory location or Register where data is stored or retrieved. The size of the Address Register is equal to the width of the memory address is directly related to the size of the memory because it contains an address. If the memory has a size of 2n * m, then the address is specified using n bits.
- **Data Register (DR):** The operand is stored in the Data Register from memory. When a direct or indirect addressing operand is found, it is placed in the Data Register. This value was then used as data by the processor during its operation. It's about the same size as a word in memory.
- **Instruction Register (IR):** The instruction is stored in the Instruction Register. The instruction register contains the currently executed instruction. Because it includes instructions, the number of bits in the Instruction Register equals the number of bits in the instruction, which is n bits for an n-bit CPU.
- **Input Register (INPR):** Input Register is a register that stores the data from an input device. The computer's alphanumeric code determines the size of the input register.
- **Program Counter (PC):** The Program Counter serves as a pointer to the memory location where the next instruction is stored. The size of the PC is equal to the width of the memory address, and the number of bits in the PC is equal to the number of bits in the PC.
- **Temporary Register (TR):** The Temporary Register is used to hold data while it is being processed. As Temporary Register stores data, the number of bits it contains is the same as the number of bits in the data word.
- **Output Register (OUTR):** The data that needs to be sent to an output device is stored in the Output Register. Its size is determined by the alphanumeric code used by the computer.

What is the Common Bus System?

A bus is a pair of signal lines that allows multi-bit data to be transferred from one system to another. A common bus is a more efficient method of sending data in a system with multiple registers. The common bus connects the outputs of seven registers and memory. A bus provides a means for people to communicate with one another.

The basic computer registers and memory connection to a common bus system are depicted in the figure below:

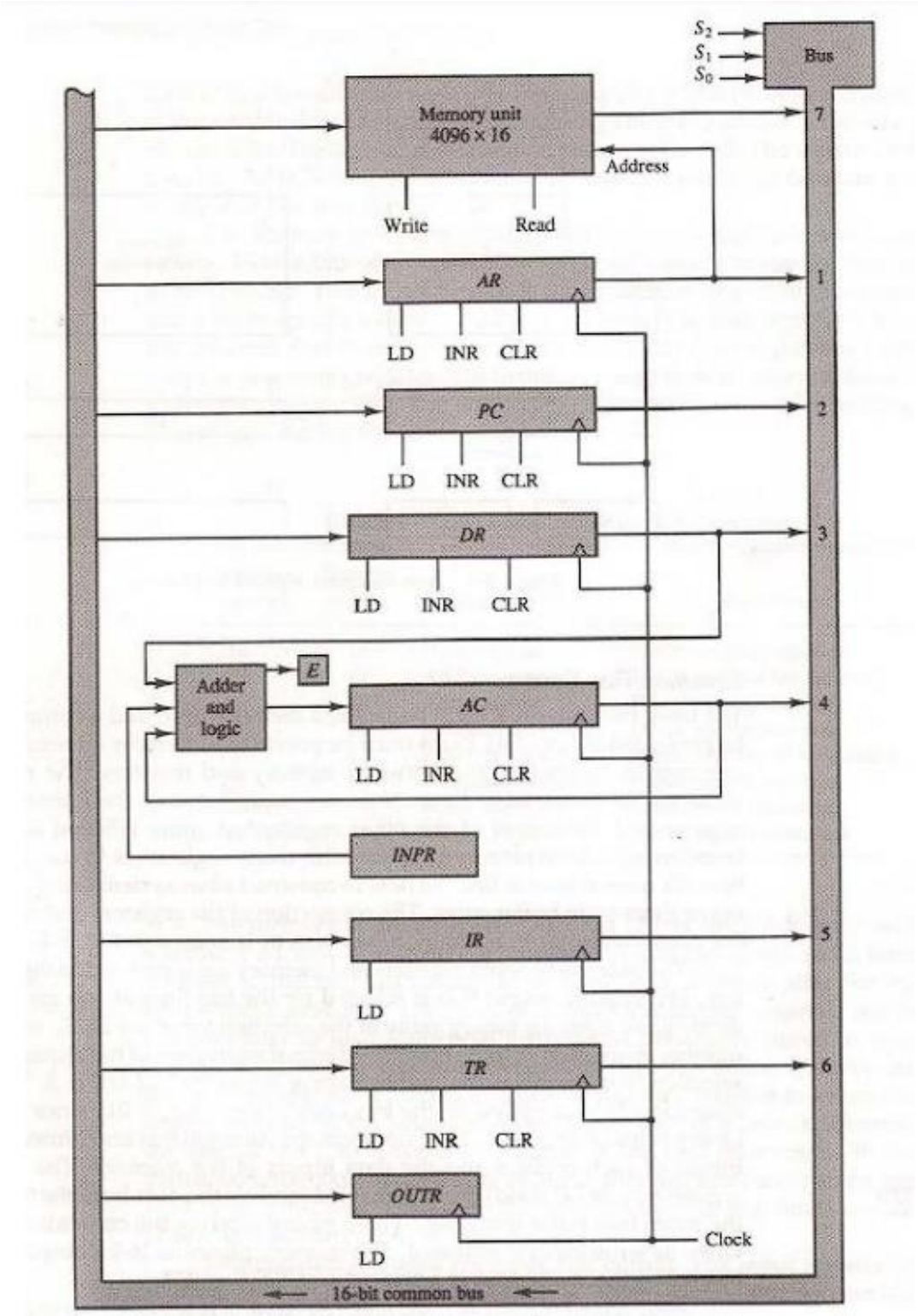


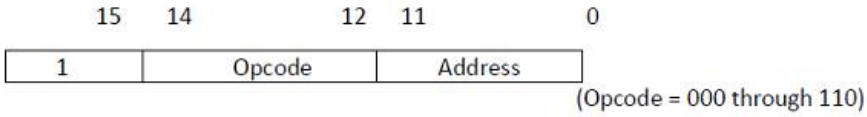
Figure 5-4 Basic computer registers connected to a common bus.

Computer Instructions:

Computer instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

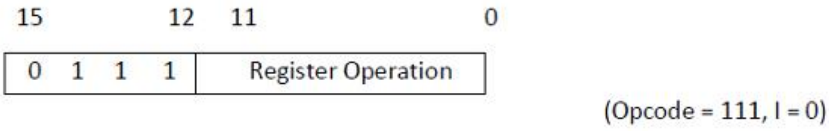
A basic computer has three instruction code formats which are:

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



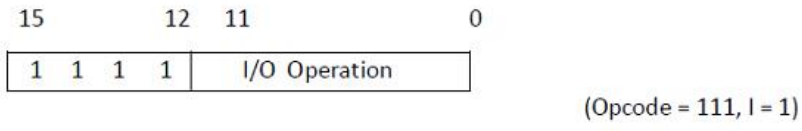
(a) Memory Reference Instruction

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.



(b) Register Reference Instruction

3. Input/Output – These instructions are for communication between computer and outside 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.



(c) Input-Output Instruction

The type of instruction is identified 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 similar 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 similar to 111, the control then examines 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.

TABLE 5-2 Basic Computer Instructions

Symbol	Hexadecimal code $I = 0$	Hexadecimal code $I = 1$	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in 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 instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
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

Uses of Basic Computer Instructions:

1. **Data manipulation:** Basic computer instructions are used to manipulate data stored in the computer system, including moving data between memory and the CPU, performing mathematical operations, and performing logical operations.
2. **Control flow:** Basic computer instructions are used to control the flow of instructions within the computer system. This includes branching to different parts of the program based on specified conditions and jumping to a specific memory location.
3. **Input/output operations:** Basic computer instructions are used to transfer data between the computer system and external devices, such as input devices (e.g. keyboard, mouse) and output devices (e.g. display screen, printer).

4. **Program execution:** Basic computer instructions are used to execute computer programs and run software applications. These instructions are used to load programs into memory, move data into and out of the program, and control the execution of the program.
5. **System maintenance:** Basic computer instructions are used to perform system maintenance tasks, such as memory allocation and deallocation, interrupt handling, and error detection and correction.

Issues of Basic Computer Instructions:

1. **Complexity:** Basic computer instructions can be complex and difficult to understand, particularly for novice programmers. This can make it challenging to write efficient and effective code.
2. **Limited functionality:** While basic computer instructions are versatile and can perform a wide range of tasks, they are still limited in their functionality. This can make it challenging to perform more complex operations and can require programmers to write additional code to accomplish their goals.
3. **Compatibility:** Basic computer instructions can be specific to a particular computer system or architecture, which can make it challenging to write code that is compatible with different systems. This can require programmers to write separate code for each system, which can be time-consuming and inefficient.
4. **Security:** Basic computer instructions can be vulnerable to security threats, such as buffer overflows and code injection attacks. This can make it challenging to write secure code and can require additional measures to be taken to protect the system.
5. **Maintenance:** Basic computer instructions can be difficult to maintain, particularly as systems become more complex and code becomes more extensive. This can require significant resources to maintain and update the code, which can be challenging for organizations with limited resources.

Instruction Set Completeness:

A set of instructions is said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

- Arithmetic, logical and shift instructions
- A set of instructions for moving information to and from memory and processor registers.
- Instructions which control the program together with instructions that check status conditions.
- Input and Output instructions

Arithmetic, logic and shift instructions provide computational capabilities for processing the type of data the user may wish to employ.

A huge amount of binary information is stored in the memory unit, but all computations are done in processor registers. Therefore, one must possess the capability of moving information between these two units.

Program control instructions such as branch instructions are used change the sequence in which the program is executed.

Input and Output instructions act as an interface between the computer and the user. Programs and data must be transferred into memory, and the results of computations must be transferred back to the user.

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 microoperations for the accumulator.
- There are two major types of control organization:
  1. Hardwired control
  2. Microprogrammed control
- The differences between hardwired and microprogrammed control are –

Hardwired Control Unit	**Microprogrammed Control Unit**
Hardwired control unit generates the control signals needed for the processor using logic circuits	Microprogrammed control unit generates the control signals with the help of micro instructions stored in control memory
Hardwired control unit is faster when compared to microprogrammed control unit as the required control signals are generated with the help of hardwares	This is slower than the other as micro instructions are used for generating signals here
Difficult to modify as the control signals that need to be generated are hard wired	Easy to modify as the modification need to be done only at the instruction level
More costlier as everything has to be realized in terms of logic gates	Less costlier than hardwired control as only micro instructions are used for generating control signals

It cannot handle complex instructions as the circuit design for it becomes complex	It can handle complex instructions
Only limited number of instructions are used due to the hardware implementation	Control signals for many instructions can be generated
Used in computer that makes use of Reduced Instruction Set Computers(RISC)	Used in computer that makes use of Complex Instruction Set Computers(CISC)

Control Unit with Timing Diagram:

The block diagram of the control unit is shown in figure 2.7.

Components of Control unit are

1. Two decoders
2. A sequence counter
3. Control logic gates

An instruction read from memory is placed in the instruction register (IR). In control unit the IR is divided into three parts:

1. the 1 bit,
2. the operation code (12-14) bit, and
3. bits 0 through 11.

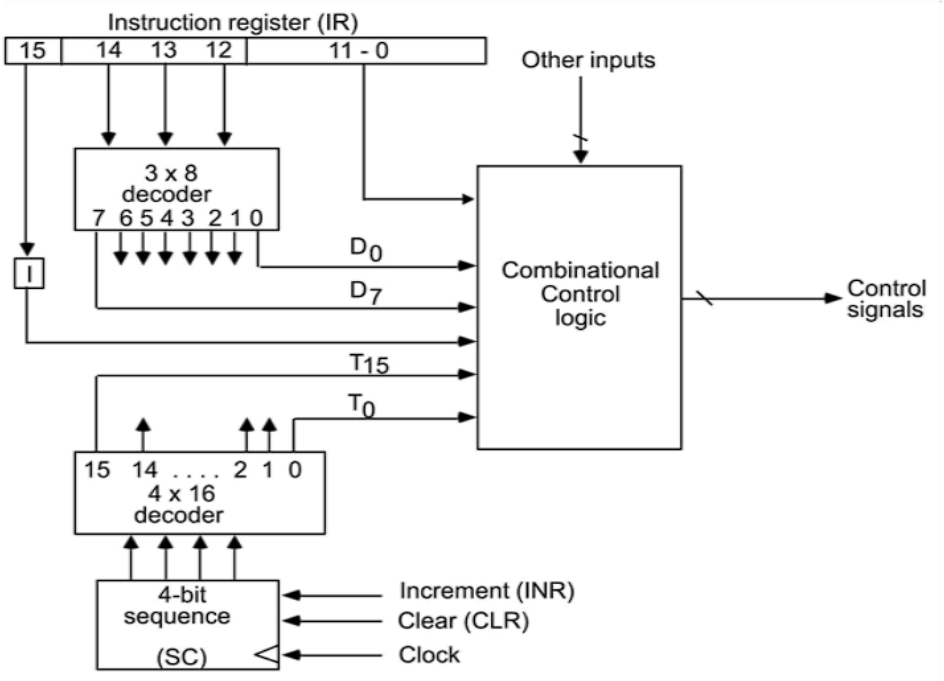


Figure 2.7: Control unit of basic computer

The operation code in bits 12 through 14 are 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

$$D3T4: SC \leftarrow 0$$

Timing Diagram:

The timing diagram of Fig. 5-7 shows the time relationship of the control signals.

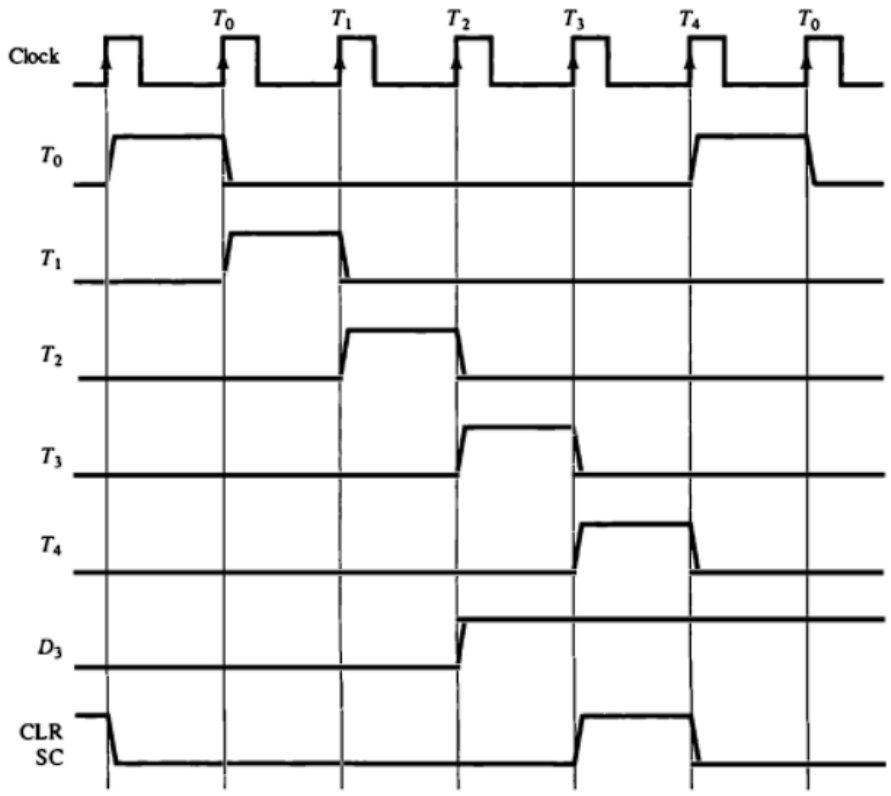


Figure 5-7 Example of control timing signals.

- The sequence counter SC responds to the positive transition of the clock.
- Initially, the CLR input of SC is active.
- The first positive transition of the clock clears SC to 0, which in turn activates the timing T0 out of the decoder. T0 is active during one clock cycle. The positive clock transition labeled T0 in the diagram will trigger only those registers whose control inputs are connected to timing signal T0.
- SC is incremented with every positive clock transition, unless its CLR input is active.
- This procedures the sequence of timing signals T0, T1, T2, T3 and T4, and so on. If SC is not cleared, the timing signals will continue with T5, T6, up to T15 and back to T0.
- The last three waveforms shows how SC is cleared when D3T4 = 1. Output D3 from the operation decoder becomes active at the end of timing signal T2. When timing signal T4 becomes active, the output of the AND gate that implements the control function D3T4 becomes active.
- This signal is applied to the CLR input of SC. On the next positive clock transition the counter is cleared to 0. This causes the timing signal T0 to become active instead of T5 that would have been active if SC were incremented instead of cleared.
- A memory read or write cycle will be initiated with the rising edge of a timing signal. It will be assumed that a memory cycle time is less than the clock cycle time. According to this assumption, a memory read or write cycle initiated by a timing signal will be completed by the time the next clock goes through its positive transition.
- The clock transition will then be used to load the memory word into a register. This timing relationship is not valid in many computers because the memory cycle time is usually longer than the processor clock cycle. In such a case it is necessary to provide wait cycles in the processor until the memory word is available. To facilitate the presentation, we will assume that a wait period is not necessary in the basic computer.
- To fully comprehend the operation of the computer, it is crucial that one understands the timing relationship between the clock transition and the timing signals. For example, the register transfer statement

$$T_0: AR \leftarrow PC$$

specifies a transfer of the content of PC into AR if timing signal T_0 is active. T_0 is active during an entire clock cycle interval. During this time the content of PC is placed onto the bus (with $S_2S_1S_0 = 010$) and the LD (load) input of AR is enabled. The actual transfer does not occur until the end of the clock cycle when the clock goes through a positive transition. This same positive clock transition increments the sequence counter SC from 0000 to 0001. The next clock cycle has T_1 active and T_0 inactive.

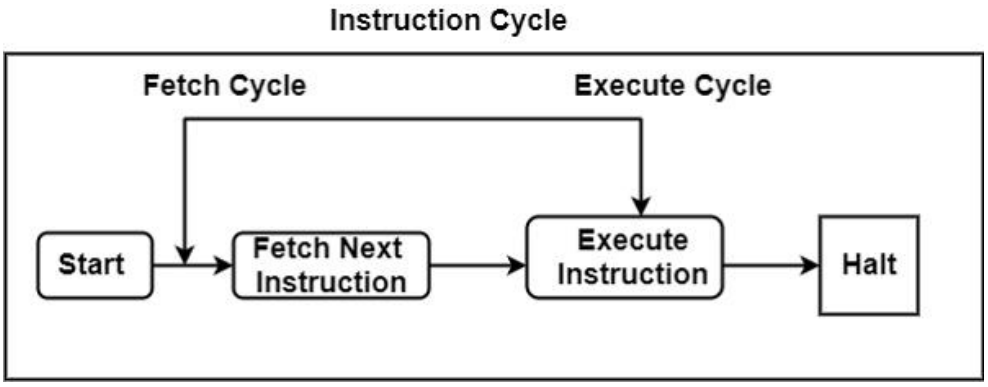
Instruction Cycle:

The instruction cycle is defined as the basic cycle in which a computer system fetches an instruction from memory, decodes it, and then executes it. **Fetch-Execute-Cycle** is another name for it. All instructions in a computer system are executed in the RAM of the computer system. The CPU is in charge of carrying out the instruction.

In the basic computer, each instruction cycle includes the following procedures –

- It can fetch instruction from memory.
- It is used to decode the instruction.
- It can read the effective address from memory if the instruction has an indirect address.
- It can execute the instruction.

After the following four procedures are done, the control switches back to the first step and repeats the similar process for the next instruction. Therefore, the cycle continues until a **Halt** condition is met. The figure shows the phases contained in the instruction cycle.



As display in the figure, the halt condition appears when the device receive turned off, on the circumstance of unrecoverable errors, etc.

Fetch Cycle:

The address instruction to be implemented is held at the program counter. The processor fetches the instruction from the memory that is pointed by the PC.

Next, the PC is incremented to display the address of the next instruction. This instruction is loaded onto the instruction register. The processor reads the instruction and executes the important procedures.

Execute Cycle:

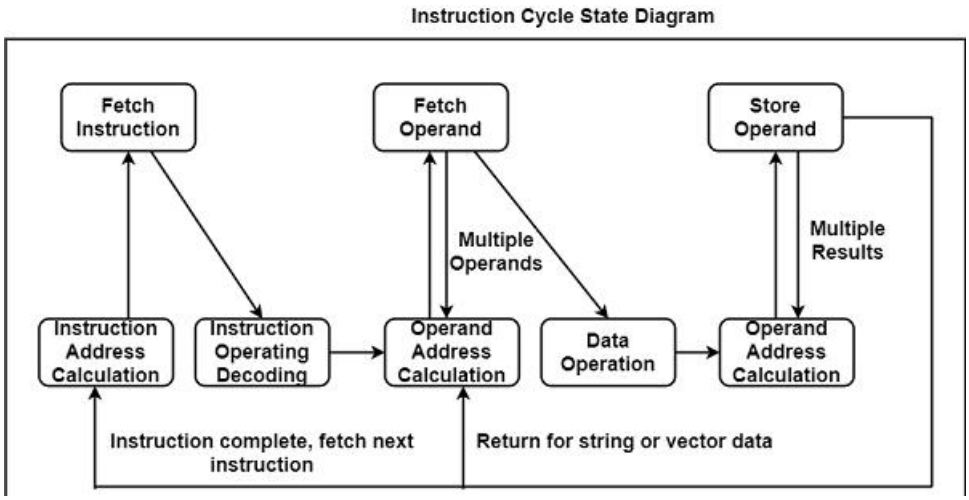
The data transfer for implementation takes place in two methods are as follows −

- **Processor-memory** − The data sent from the processor to memory or from memory to processor.
- **Processor-Input/Output** − The data can be transferred to or from a peripheral device by the transfer between a processor and an I/O device.

In the execute cycle, the processor implements the important operations on the information, and consistently the control calls for the modification in the sequence of data implementation. These two methods associate and complete the execute cycle.

State Diagram for Instruction Cycle:

The figure provides a large aspect of the instruction cycle of a basic computer, which is in the design of a state diagram. For an instruction cycle, various states can be null, while others can be visited more than once.



- **Instruction Address Calculation** − The address of the next instruction is computed. A permanent number is inserted to the address of the earlier instruction.
- **Instruction Fetch** − The instruction is read from its specific memory location to the processor.
- **Instruction Operation Decoding** − The instruction is interpreted and the type of operation to be implemented and the operand(s) to be used are decided.
- **Operand Address Calculation** − The address of the operand is evaluated if it has a reference to an operand in memory or is applicable through the Input/Output.
- **Operand Fetch** − The operand is read from the memory or the I/O.
- **Data Operation** − The actual operation that the instruction contains is executed.
- **Store Operands** − It can store the result acquired in the memory or transfer it to the I/O.

Register-Reference Instructions:

- Register-reference instructions are recognized by the control when D7 = 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).
- The control functions and microoperations for the register-reference instructions are listed in Table 5-3.
- These instructions are executed with the clock transition associated with timing variable T_3.
- Control function needs the Boolean relation $D_7I'T_3$, which we designate for convenience by the symbol r.
- By assigning the symbol B_i to bit i of *IR*, all control functions can be simply denoted by rB_i.

TABLE 5-3 Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0–11)$ that specifies the operation]

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

- For example, the instruction CLA has the hexadecimal code 7800, which gives the binary equivalent 0111 1000 0000 0000. The first bit is a zero and is equivalent to I′.
- The next three bits constitute the operation code and are recognized from decoder output D_7.
- Bit 11 in IR is 1 and is recognized from B_{11}. The control function that initiates the microoperation for this instruction is $D_7I'T_3\ B_{11} = rB_{11}$.
- The execution of a register-reference instruction is completed at time T_3.
- The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal T_0.
- The first seven register-reference instructions perform clear, complement, circular shift, and increment microoperations on the AC or E registers.
- The next four instructions cause a skip of the next instruction in sequence when a stated condition is satisfied. The skipping of the instruction is achieved by incrementing *PC* once again.
- The condition control statements must be recognized as part of the control conditions.
- The *AC* is positive when the sign bit in AC(15) = 0; it is negative when AC(15) = 1. The content of *AC* is zero *(AC* = 0) if all the flip-flops of the register are zero.
- The HLT instruction clears a start-stop flip-flop *S* and stops the sequence counter from counting.

Memory- Reference Instructions:

- Table 5-4 lists the seven memory-reference instructions.
- The decoded output D_i for i = 0, 1, 2, 3, 4, 5, and 6 from the operation decoder that belongs 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 execution of the memory-reference instructions starts with timing signal T_4.
- The symbolic description of each instruction is specified in the table in terms of register transfer notation.

TABLE 5-4 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 $PC \leftarrow PC + 1$

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: $DR \leftarrow M[AR]$
D_0T_5: $AC \leftarrow AC \wedge DR, \quad SC \leftarrow 0$

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: $DR \leftarrow M[AR]$
D_1T_5: $AC \leftarrow AC + DR, \quad E \leftarrow C_{out}, \quad SC \leftarrow 0$

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: $DR \leftarrow M[AR]$
D_2T_5: $AC \leftarrow DR, \quad SC \leftarrow 0$

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.

$$D_3T_4: \quad M[AR] \leftarrow AC, \quad SC \leftarrow 0$$

BUN: Branch Unconditionally

- This instruction transfers the program to the instruction specified by the effective address.
- 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:

$$D_4T_4: \quad PC \leftarrow AR, \quad SC \leftarrow 0$$

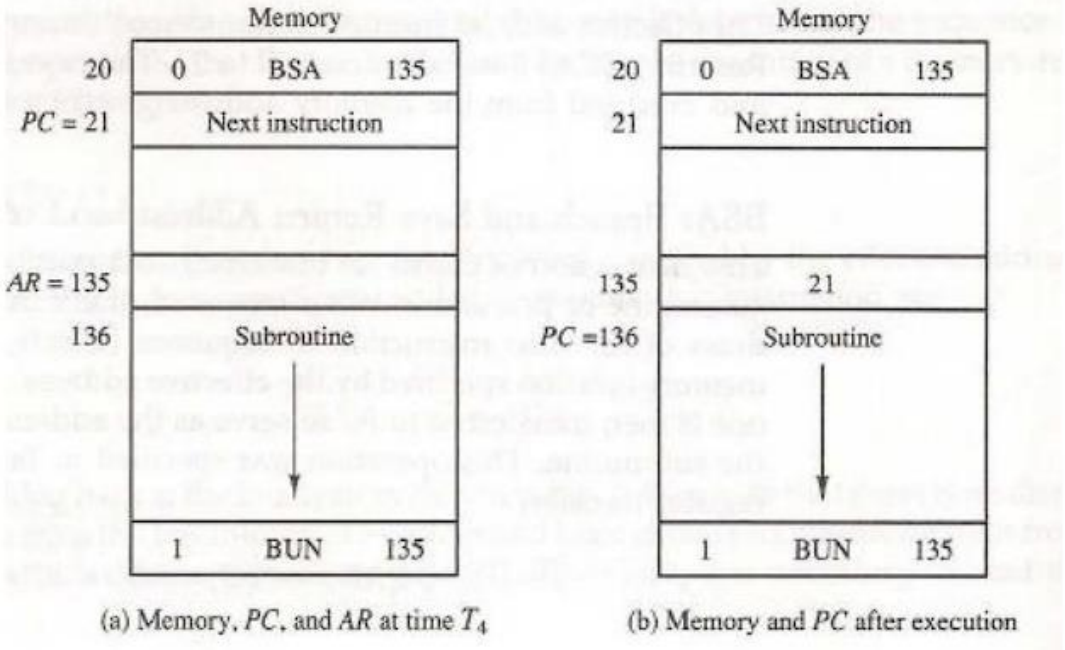
BSA: Branch and Save Return Address

- This instruction is useful for branching to a portion of the program called a subroutine or procedure.
- When executed, the BSA instruction stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.
- The effective address plus one is then transferred to *PC* to serve as the address of the first instruction in the subroutine.
- This operation was specified with the following register transfer:

$$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$$

- A numerical example that demonstrates how this instruction is used with a subroutine is shown in Fig. 5-10.

Figure 5-10 Example of BSA instruction execution.



- The BSA instruction is assumed to be in memory at address 20.
- The I bit is 0 and the address part of the instruction has the binary equivalent of 135.
- After the fetch and decode phases, *PC* contains 21, which is the address of the next instruction in the program (referred to as the return *address).* AR holds the effective address 135.
- This is shown in part (a) of the figure.
- The BSA instruction performs the following numerical operation:

$$M[135] \leftarrow 21, \quad PC \leftarrow 135 + 1 = 136$$

- The result of this operation is shown in part (b) of the figure.
- The return address21 is stored in memory location 135 and control continues with the subroutine program starting from address 136.
- The return to the original program (at address 21) is accomplished by means of an indirect BUN instruction placed at the end of the subroutine.
- When this instruction is executed, control goes to the indirect phase to read the effective address at location 135, where it finds the previously saved address 21.
- When the BUN instruction is executed, the effective address 21 is transferred to PC.
- The next instruction cycle finds *PC with* the value 21, so control continues to execute the instruction at the return address.
- The BSA instruction must be executed with a sequence of two microoperations:

$$D_5T_4: \quad M[AR] \leftarrow PC, \quad AR \leftarrow AR + 1$$
$$D_5T_5: \quad PC \leftarrow AR, \quad SC \leftarrow 0$$

ISZ: Increment and Skip if Zero

- This instruction increment the word specified by the effective address, and if the incremented value is equal to 0, PC is incremented by 1 to skip the next instruction in the program.
- Since 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:

$$D_6T_4: \quad DR \leftarrow M[AR]$$
$$D_6T_5: \quad DR \leftarrow DR + 1$$
$$D_6T_6: \quad M[AR] \leftarrow DR, \quad \text{if } (DR = 0) \text{ then } (PC \leftarrow PC + 1), \quad SC \leftarrow 0$$

Control Flowchart:

- A flowchart showing all microoperations for the execution of the seven memory-reference instructions is shown in Fig. 5.11.

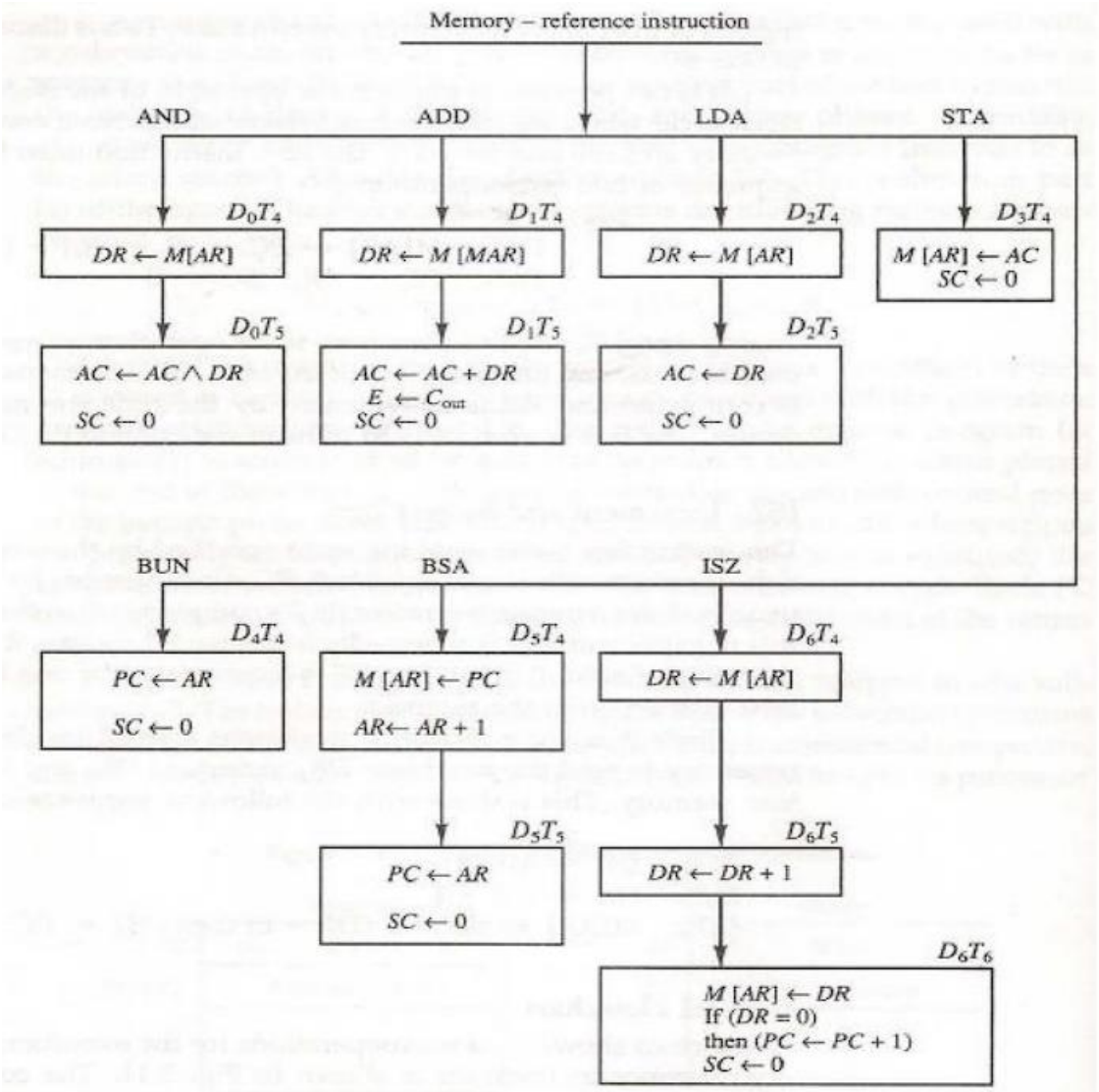


Figure 5-11 Flowchart for memory-reference instructions.

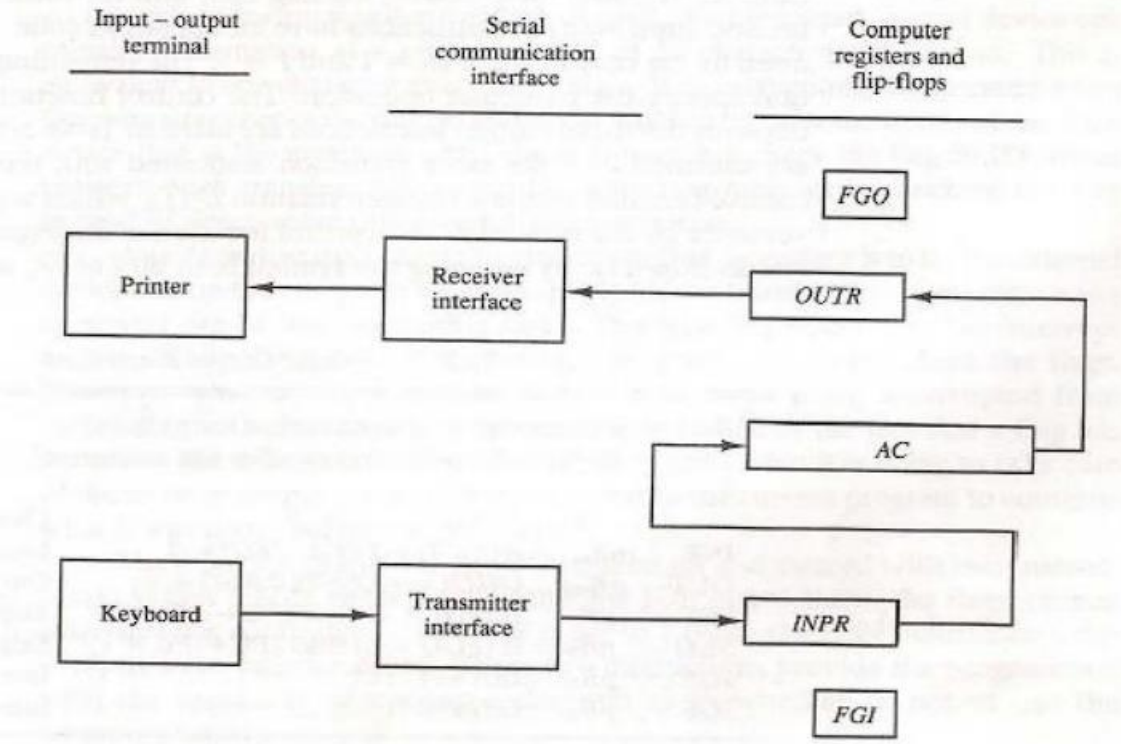
Input-Output and Interrupt:

- Instructions and data stored in memory must come from some input device.
- Computational results must be transmitted to the user through some output device.
- To demonstrate the most basic requirements for input and output communication, we will use as an illustration a terminal unit with a keyboard and printer.

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 shown in Fig. 5-12.

Figure 5-12 Input-output configuration.



- The input register INPR consists of eight bits and holds alphanumeric input information.
- The 1-bit input flag *FGI* is a control flip-flop.
- The flag bit is set to 1 when new information is available in the input device and is cleared to 0 when the information is accepted by the computer.
- 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.

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 D7 = 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 5-5.

TABLE 5-5 Input-Output Instructions

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

	p:	$SC \leftarrow 0$	Clear SC
INP	pB_{11}:	$AC(0\text{–}7) \leftarrow INPR$, $FGI \leftarrow 0$	Input character
OUT	pB_{10}:	$OUTR \leftarrow AC(0\text{–}7)$, $FGO \leftarrow 0$	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

- These instructions are executed with the clock transition associated with timing signal T_3.
- Each control function needs a Boolean relation D_7IT_3, which we designate for convenience by the symbol p.
- The control function is distinguished by one of the bits in IR (6-11).
- By assigning the symbol B_i to bit *i* of IR, all control functions can be denoted by pB_i for i = 6 though 11.
- The sequence counter *SC* is cleared to 0 when $p = D_7IT_3 = 1$.
- The last two instructions set and clear an interrupt enable flip-flop IEN.

Program Interrupt:

- The computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.
- The difference of information flow rate between the computer and that of the input—output device makes this type of transfer inefficient.
- An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer.
- In the meantime the computer can be busy with other tasks. This type of transfer uses the interrupt facility.
- While the computer is running a program, it does not check the flags.
- When a flag is set, the computer is momentarily interrupted from the current program.
- The computer deviates momentarily from what it is doing to perform of the input or output transfer.
- It then returns to the current program to continue what it was doing before the interrupt.
- The interrupt enable flip-flop IEN can be set and cleared with two instructions.
    - When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer.
    - When *IEN* is set to (with the ION instruction), the computer can be interrupted.
- The way that the interrupt is handled by the computer can be explained by means of the flowchart of Fig. 5-13.
- An interrupt flip-flop R is included in the computer. When *R* = 0, the computer goes through an instruction cycle.
- During the execute phase of the instruction cycle IEN is checked by the control.
- If it is 0, it indicates that the programmer does not want to use the interrupt,so control continues with the next instruction cycle.
- If IEN is 1, control checks the flag bits. If both flags are 0, it indicates that neither the input nor the output registers are ready for transfer of information. In this case, control continues with the next instruction cycle.
- If either flag is set to 1 while *1EN* = 1, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

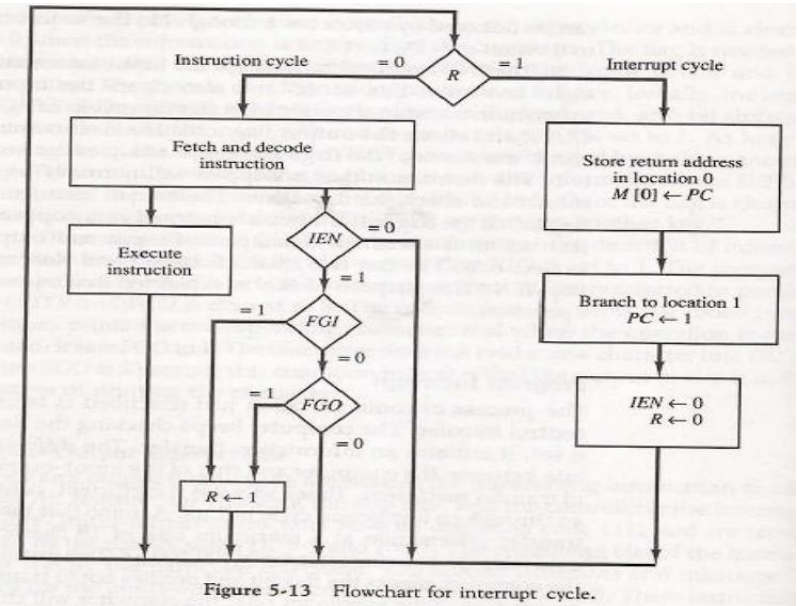
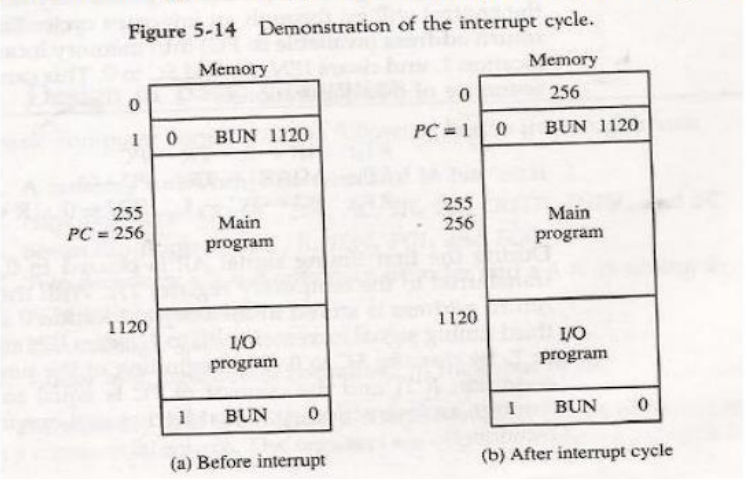


Figure 5-13 Flowchart for interrupt cycle.

Interrupt cycle:

- The interrupt cycle is a hardware implementation of a branch and save return address operation.
- The return address available in PC is stored in a specific location.
- This location may be a processor register, a memory stack, or a specific memory location.
- An example that shows what happens during the interrupt cycle is shown in Fig. 5-14.

Figure 5-14 Demonstration of the interrupt cycle.



- When an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255.
- At this time, the returns address 256 is in *PC*.
- The programmer has previously placed an input—output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1. This is shown in Fig. 5.14(a).
- When control reaches timing signal $T_0$and finds that *R = 1,* it proceeds with the interrupt cycle.
- The content of *PC* (256) is stored in memory location 0, *PC* is set to 1, and R is cleared to 0.
- The branch instruction at address 1 causes the program to transfer to the input—output service program at address 1120.
- This program checks the flags, determines which flag is set, and then transfers the required input or output information.
- Once this is done, the instruction ION is executed to set IEN to 1 (to enable further interrupts), and the program returns to the location where it was interrupted.
- This is shown in Fig. 5-14(b).

Design of a Basic Computer:

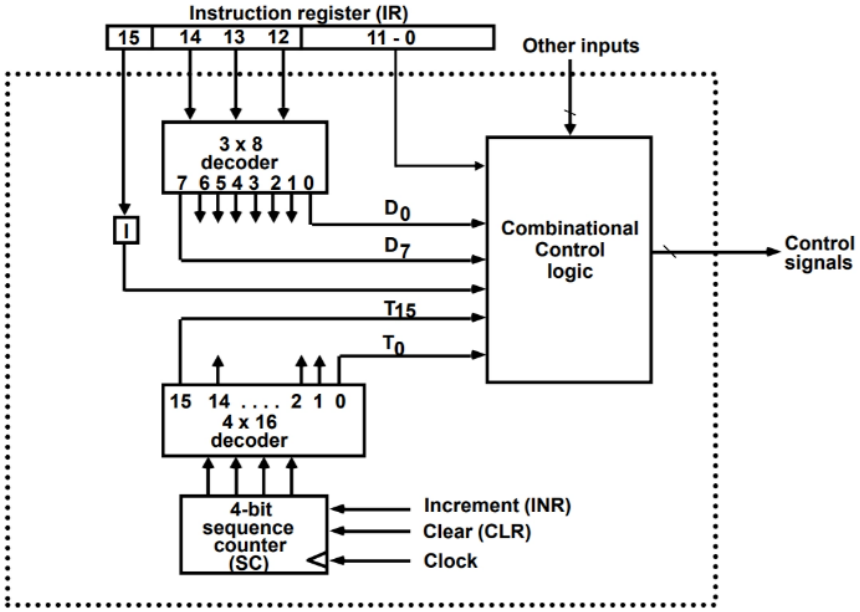
A basic computer consists of the following hardware components -

1. A memory unit with 4096 words of 16 bits each
2. Registers: AC (Accumulator), DR (Data register), AR (Address register), IR (Instruction register), PC (Program counter), TR (Temporary register), SC (Sequence Counter), INPR (Input register), and OUTR (Output register).
3. Flip-Flops: I, S, E, R, IEN, FGI and FGO

Note: FGI and FGO are corresponding input and output flags which are considered as control flip-flops.

4. Two decoders: a 3 x 8 operation decoder and 4 x 16 timing decoder
5. A 16-bit common bus
6. Control Logic Gates
7. The Logic and Adder circuits connected to the input of AC.

Control Logic Gates:



The input for the control logic gate comes from:

1. Input from I flip-flops
2. Input from the two decoders
3. Input from 0-11 bits of IR (Instruction Register)
4. Other inputs to the control logic gate include
    1. AC (Accumulator) bits 0-15, to check if AC=0 and to detect the sign bit in AC(15)
    2. DR (Data Register) bits 0-15, to check if DR=0 and check the values of seven flip-flops.

The output of the control logic circuit is as follows:

1. Signals to control the read and write inputs of memory
2. Signals to control the inputs of the eight registers.
3. Signals to control the AC adder and logic circuit
4. Signals to control the S2, S1, and S0 to select a register for the bus
5. Signals to set, clear or complement the flip-flops

Control of Registers and Memory:

Control inputs or signals of the registers are:

1. LD (load)
2. INR (increment)
3. CLR (clear)

Let's consider the case of AR to understand the control of registers and memory.

If we want to derive the gate structure associated with the control input of AR, we consider all the statements that change the content of AR:

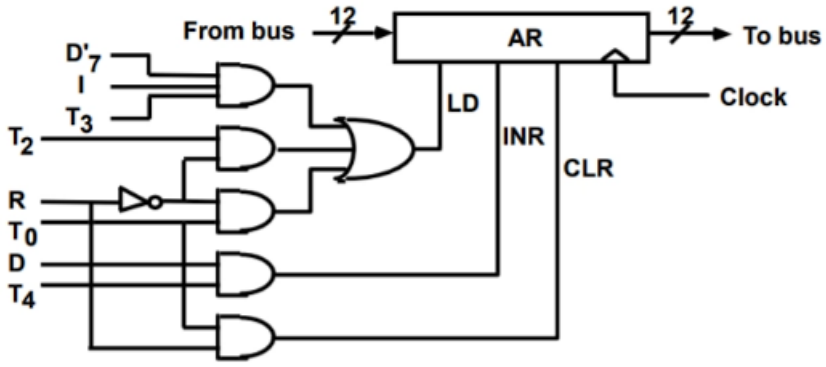
1. R'T0: AR ← PC
2. R`T2: AR ← IR (0-11)
3. D7`IT3 : AR ← M [AR]
4. RT0 : AR ← 0
5. D5T4: AR ← AR+1

1, 2, and 3 are responsible for transferring information from a register or memory to AR. The content of the source register memory is placed on the bus, and the content of the bus is transferred into AR by enabling its LD control input. 4 clears AR to 0 and 5 increments AR by one.

The control function can be combined into three boolean expressions as follows:

1. LD (AR) = R`T0 + R`T2 + D`7IT3 (load input of AR)
2. CLR (AR) = RT0 (clear of AR)
3. INR (AR) = D5T4 (increment input of AR)

The control gate logic associated with AR is shown below:



The read operation is recognized from the symbol ←M[AR]

Read = R`T1 + D`7IT3 + (D0 + D1 + D2 + D6)T4

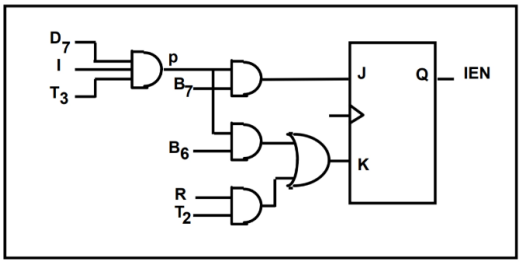
The control logic gate for the above control function can be designed the same way we designed AR. In the above operation, addition refers to OR, and multiplication refers to AND. The output of the logic gates that implement the boolean expression above must be connected to the read input of memory.

Control of Single Flip-Flops:

To understand the control of flip-flops, we will observe the control of IEN (Interrupt enable on) flip-flop. Now, the IEN may change as a result of the two instructions, ION and IOF

PB7: IEN ← 1
PB6: IEN ← 0
Where P = D7IT3
And B7 B6 are bits 7 and 6 of IR, respectively.
At the end of the interrupt cycle, IEN is cleared to 0
RT2 : IEN ← 0

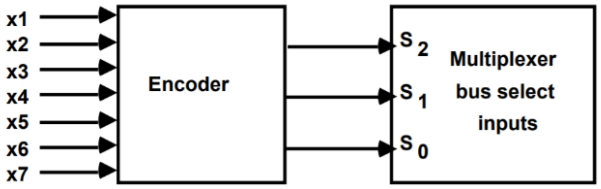
The control logic gate for IEN is given below:



Control of Common Bus:

We use an encoder for the bus selection circuit where the output of the encoder determines the data of which register is to be transferred on the bus.

The block diagram of the encoder used is given below:



The input and output of the above encoder are as shown below in the table.

x1	x2	x3	x4	x5	x6	x7	S2	S1	S0	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

We can see in the above table that for each output value, there is a register selected. If we observe in the above table what events trigger S0 to be 1, we find that S0 is 1 when X1 or X3 or X5 or X7 is 1. Similarly, for S1 and S2. From this observation, we can deduce that each binary number is associated with a boolean variable (X1-X7), corresponding to the gate structure that must be active to select the register or memory for the bus.

The boolean function for the encoder can be written as

S0 = X1 + X3 + X5 + X7

S1 = X2 + X3 + X6 + X7

S2 = X4 + X5 + X6 + X7

It's necessary to find the control functions that place the corresponding register onto the bus to determine the logic for each encoder input. Let's find this for X1. First, we will scan all the register transfer statements and extract the statements having AR as a source. We get the following instructions after this:

D4T4 : PC ← AR

D5T5: PC ← AR

So the boolean expression for X1 becomes

X1 = D4T4 + D5T5

Similarly, we can find the gate logic for X2 to X7 for different registers.

Flowchart for Computer Operation:

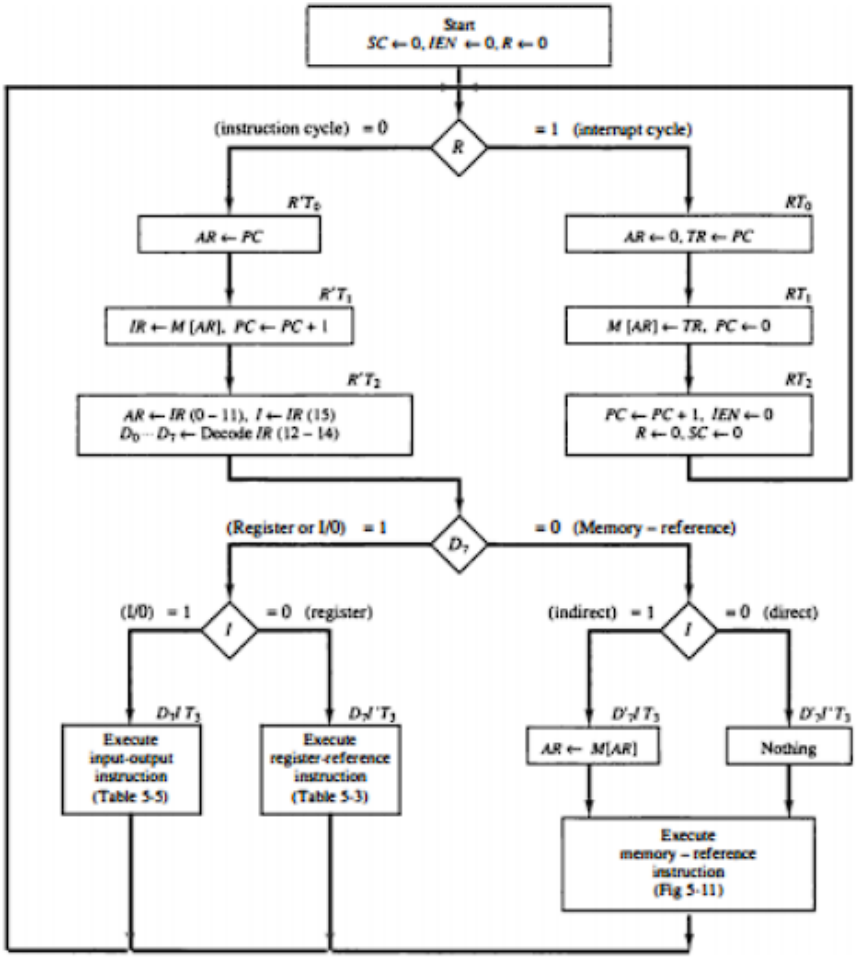


Figure 5-15 Flowchart for computer operation.

The instruction cycle is about how instruction is getting executed, and in the interrupt cycle, how Interrupt works when it occurs.

- We start with Sequence counter set to 0, Interrupt enable set to 0, and R flip-flop set to 0.
- R flip-flop is checked, which decides whether it is an instruction or an interrupt cycle.
- When R is 0, then it is the instruction cycle.
  - At time R'T0, the PC contents are transferred to AR, which contains the address of the instruction to be executed.
  - At time R'T1, contents of memory pointed by AR are stored in IR. Along with that, PC is incremented by one as we have to shift to the next instruction. This step completes the fetch part of the process.
  - At time R'T2
    - decoding will take place from (12-14) bits of IR register which will generate the corresponding (D0-D7) bits
    - (0-11) bits of IR are copied to the AR register.
    - The 15th bit of the IR register is copied into I flip-flop.
  - After decoding, we check the D7 bit.
  - If the D7 bit is 1, it is either register or i/o instruction.
    - We check for I; if it is 1, then it is I/O instruction, and we immediately execute the I/O instruction at time T3. after this, we make the sequence counter 0.
    - If I value 0, then it is a register reference instruction, so we execute it at time T3 and set SC to 0
  - If the D7 bit is 0, (D0-D6) can be 1. So it becomes a memory reference instruction
    - We check for I bit; if it is 1, it is indirect addressing. So at time T3, the effective address is fetched from memory and stored in the AR register, and then after some time, memory reference instruction would get executed, and SC will be set to 0
    - If I bit 0, then it is direct addressing. So at time T3, we are supposed to do nothing, and then simply after that, memory reference instruction is executed, and SC is set to 0
  - After this, it goes back to the fetch part.
- When R is 1, then it is an interrupt cycle.
  - At time RT0
    - store 0 in AR because interrupt cycle is similar to BSA (Branch and Save return Address), but the branching is fixed in case of Interrupt, and it is to be branched at address 0
    - address of next instruction which is stored in PC is stored in TR
  - At time RT1
    - Store the TR address in memory. We are storing the written address at address 0 here.

    - set the PC(Program Counter) to 0 because when Interrupt occurs then, we need to execute the service routine, which is at address 0
- At time RT2, we
    - increment PC by 1
    - set IEN to 0 because earlier it was 0 due to interrupt cycle
    - set R to 0 because now it becomes our normal instruction cycle
    - SC is set to 0.
- After this, it goes back to the normal instruction cycle execution.