COMPUTER ORGANIZATION AND ARCHITECTURE
UNIT – 1

TOPIC- INTRODUCTION, BLOCK DIAGRAM OF A DIGITAL COMPUTER

Introduction to Digital Computers

The word "computer" comes from the Latin word computare, which means "to calculate," "to count," "to think," or "to sum up". The word "computer" means a "device that performs computation" .A computer is an electronic device that can perform a variety of tasks, such as storing, processing, and outputting data.

Functions of a computer:

1) It accepts data or instructions through input,
2) It stores data,
3) It can process required data by the user,
4) It gives results as production, and
5) It controls all functions inside the computer.

Block diagram of digital computer and the functioning of its blocks

A digital Computer is a fast electronic calculating machine that:

• accepts digitized input information,
• processes it according to a list of internally stored instructions,
• and produces the resulting output information.

Computer is an electronic device which performs tasks given by user with extremely fast speed and accuracy. Like any other device or machine, a computer system has also a number of parts. A computer system can be blocked into mainly three parts:

1. Input Unit
2. Central Processing Unit
3. Output Unit

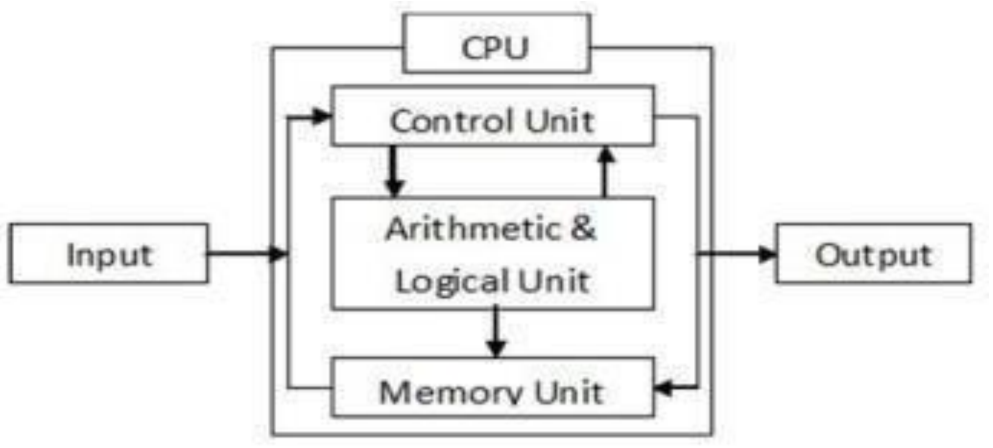


Fig. Block Diagram of Computer

1. **Input unit** – Input unit is a unit that accepts any input device. The input device is used to input data into the computer system. Function of input unit.

1. It converts inputted data into binary codes.
2. It sends data to main memory of computer.

Examples of Input devices :

2. **Central Processing Unit (CPU)** – CPU is called the brain of a computer. An electronic circuitry that carries out the instruction given by a computer program.

Functions of CPU:

1. It controls all the parts and software and data flow of computer.
2. It performs all operations.
3. It accepts data from input device.
4. It sends information to output device.

5. Executing programs stored in memory.
6. It stores data either temporarily or permanent basis.
7. It performs arithmetical and logical operations.

CPU can be sub classified into three parts.

i. Control unit (CU).
ii. Arithmetic & Logic unit (ALU).
iii. Memory Unit (MU).

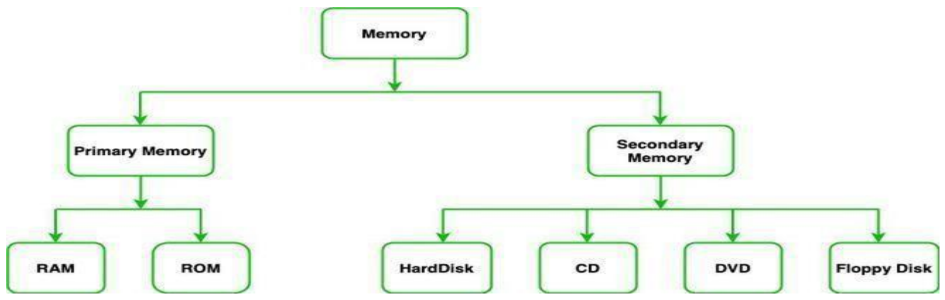
i. Control unit (CU)- the control unit manages the various components of the computer. It reads instructions from memory and interpretation and changes in a series of signals to activate other parts of the computer. It controls and co-ordinate is input output memory and all other units.

ii. Arithmetic & Logic unit (ALU) – The arithmetic logic unit (ALU), which performs simple arithmetic operation such as +,-, *, / and logical operation such as >, <, =<, <= etc.

iii. Memory Unit (MU)- Memory is used to store data and instructions before and after processing.

It is classified into:

1. Primary memory or Main memory.
2. Secondary memory



Primary memory: also called as internal memory or main memory or volatile memory. It is a computer's memory that is accessed first or directly by a processor or computer. Here the data is stored temporarily.

Ex: RAM (Random Access Memory)

Secondary memory : also called nonvolatile memory.It is a type of computer memory that stores data and programs permanently, even when the computer is turned off. It's also known as auxiliary, external, or non-volatile memory.

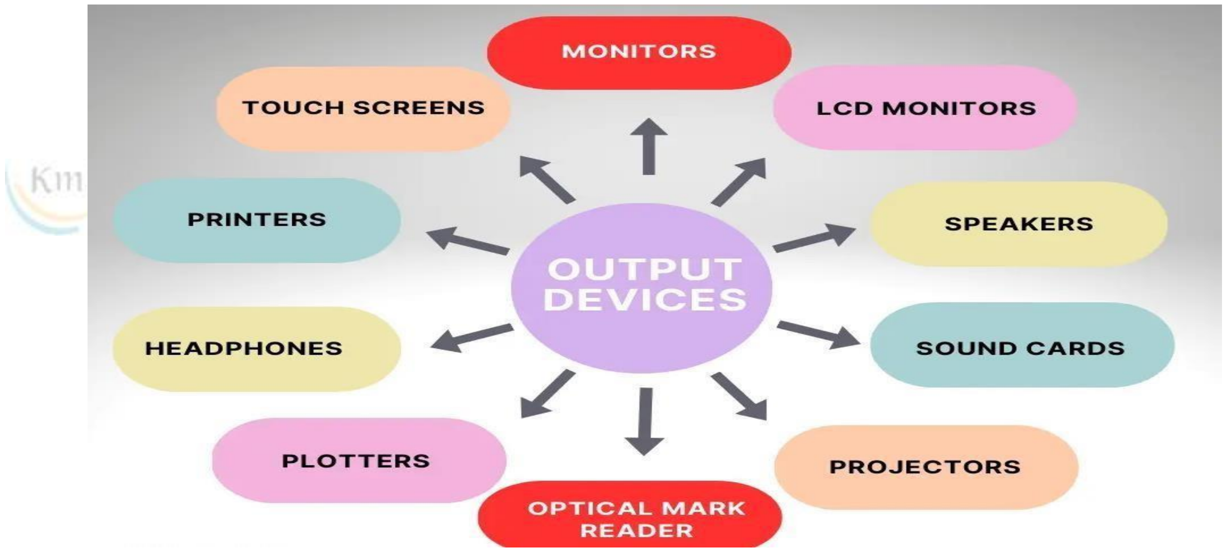
Ex: Hard disk, Compact disc, Flash drive, etc.

1. **Output Unit –**Output unit is a unit that constituents a number of output device. An output device is used to show the result of processing.

Function of Output unit:

1. It accepts data or information sends from main memory of computer
2. It converts binary coded information into HLL or inputted languages.

Examples of Output devices:



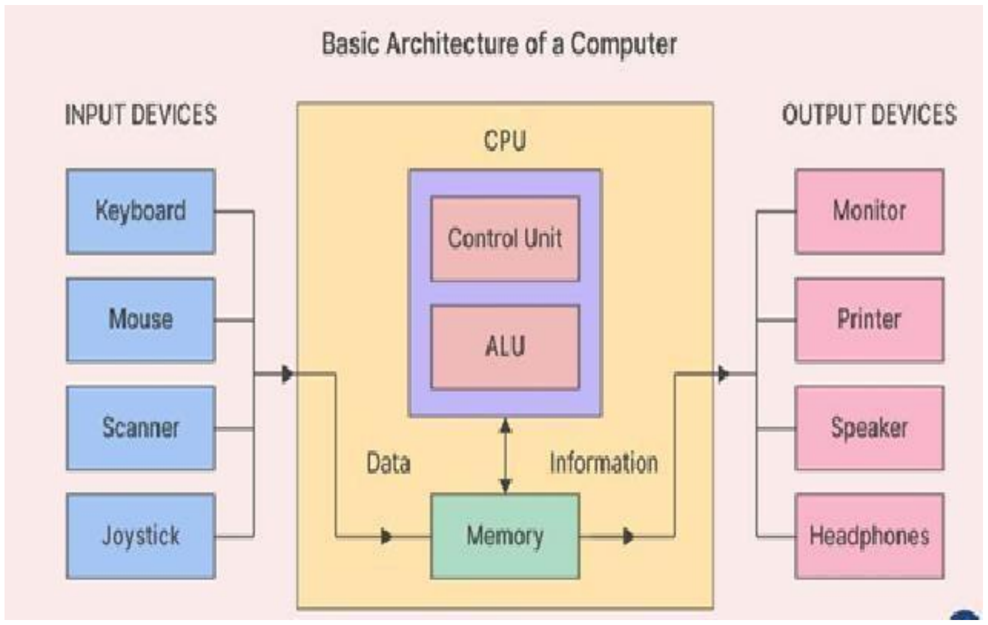
TOPIC- DEFINITION OF COMPUTER ORGANIZATION AND COMPUTER ARCHITECTURE

Definition of Computer Organization and Computer Architecture

What is meant by Computer Architecture?

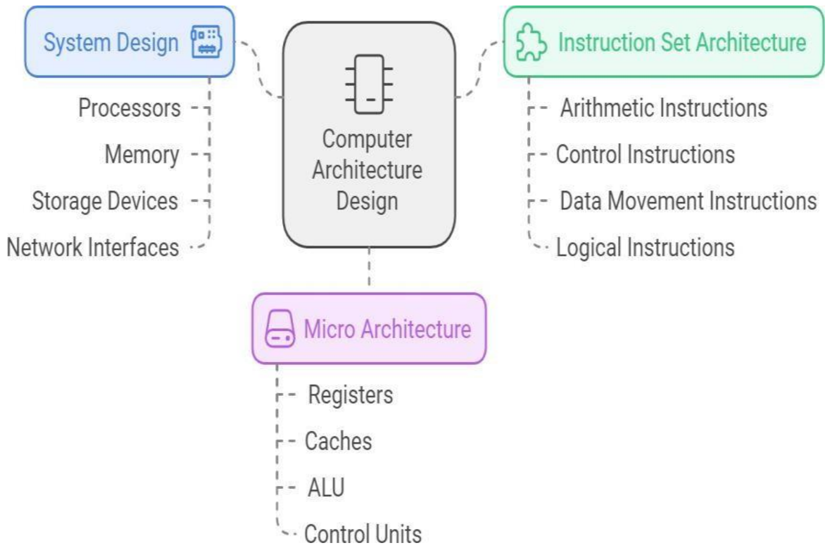
Computer architecture refers to the design and basic structure of a computer system, including its hardware structures, their interconnections, and the principles that guide their organization.

It encompasses the higher-level aspects of computer design, such as the instruction set architecture (ISA), memory hierarchy, and the organization of the central processing unit (CPU).



The following 3 main categories are considered while considering the design of architecture:

- **System Design** (contains hardware components that are used for building the system)
- **Instruction Set Architecture** (includes all the instructions provided to the computer system)
- **Micro Architecture**(give minute detail about storage element)



Computer architecture plays a critical role in determining the overall performance, power efficiency, and scalability of a computer system.

What is meant by Computer Organization?

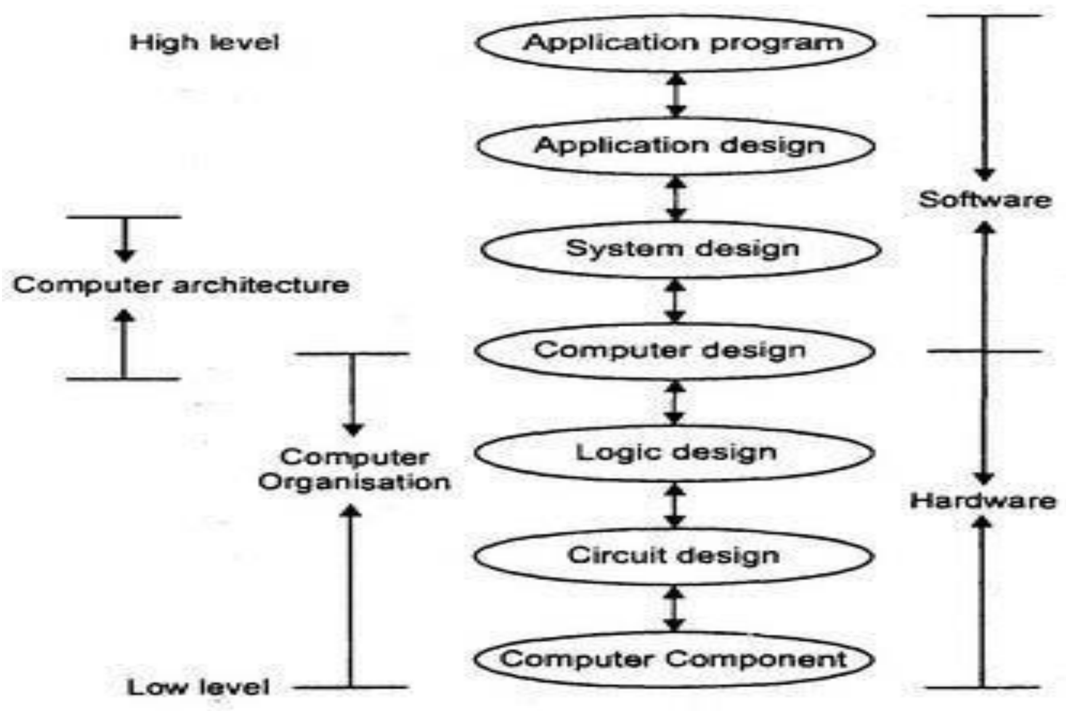
Computer organization deals with the physical components of a computer. It refers to the way the various components of a computer system are interconnected and work together to perform tasks and execute programs.

It encompasses the design and arrangement of hardware components, including the central processing unit (CPU), memory, input/output devices, and control units, to ensure efficient and effective operation of the computer.

Computer organization includes the physical connection component, like circuits with adder subtractor. CPU organization is of three types:

- Single Accumulator Organization
- General Register Organization
- Stack Organization

Computer organization plays a crucial role in determining the overall performance and efficiency of a computer system.



Difference between Computer Architecture and Computer Organization

Computer Architecture	**Computer Organization**
Computer Architecture is concerned with the way hardware components are connected to form a computer system.	Computer organization is concerned with the structure and behavior of the computer system as seen by the user.
It is a blueprint for design. So, while designing a computer system, architecture is decided first.	Computer organization is decided after the architecture.
It involves logical components such as Instruction Set, Addressing Modes etc.	It involves physical units such as circuit design, adders, signals, peripherals, etc.
It describes how a computer system is designed.	It describes how a computer system works.
It acts as an interface between hardware and software.	It deals with the components of a computer and the interconnection of components.
Computer architecture deals with high-level design issues.	Computer organization deals with low- level design issues.
Computer architecture defines the logical aspects of a computer system.	Computer organization defines the physical aspects of the computer system.
It deals with the functional behavior of the computer system.	It deals with the organizational structure of the computer and the various structural relationships.
It is also called an instruction set architecture.	It is also called microarchitecture.
Concerned with - What to do? (Instruction Set)	Concerned with - How to do?(implementation of the architecture)

Computer architecture and Computer organization in simple words

"Computer architecture is a blueprint for the design of a computer system. It describes how the computer system is designed."

On the other hand,

"Computer Organization is how operational parts of a computer system are linked together. It explains how the computer works and provides a structural relationship between parts of the computer system.

COMPUTER ORGANIZATION AND ARCHITECTURE

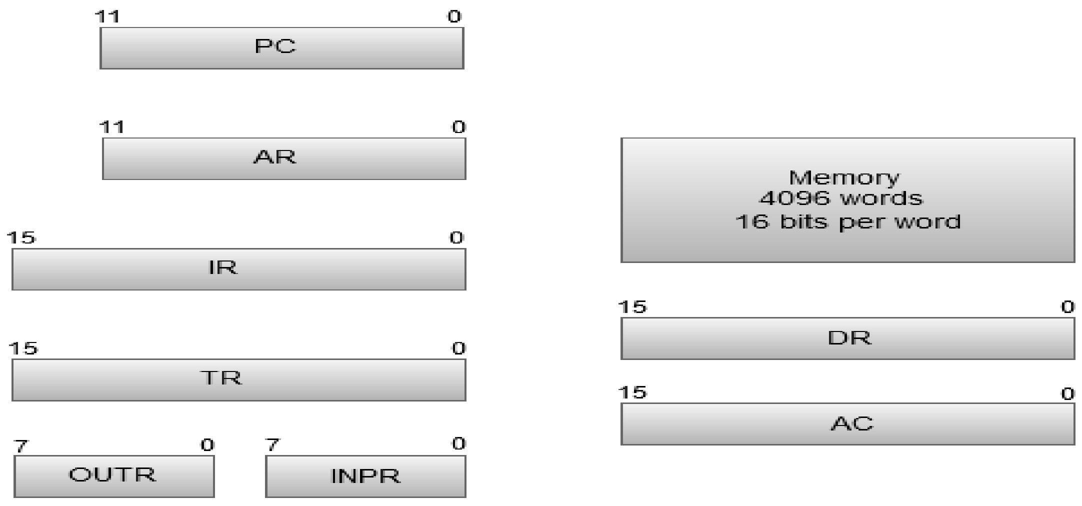
TOPIC- COMPUTER REGISTERS

Computer Registers

A register in a computer is a temporary storage area for data and instructions that are used by the central processing unit (CPU) for quick processing. Registers are a critical component of computer memory that allow for the efficient access and manipulation of information. The registers used by the CPU are often termed as Processor registers.

A processor register may hold an instruction, a storage address, or any data (such as bit sequence or individual characters).

Register and Memory Configuration of a basic computer:



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

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

- The Memory unit has a capacity of 4096 words, and each word contains 16 bits.

- The Data Register (DR) contains 16 bits which hold the operand read from the memory location.
- The Memory Address Register (MAR) contains 12 bits which hold the address for the memory location.
- The Program Counter (PC) also contains 12 bits which hold the address of the next instruction to be read from memory after the current instruction is executed.
- 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 the temporary data during the processing.

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

The Output Registers (OR) holds the output after processing the input data

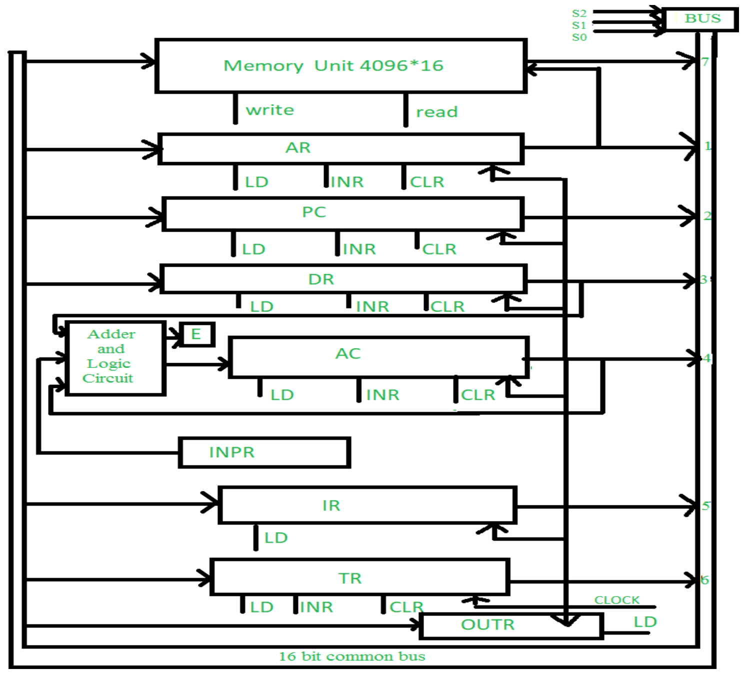
TOPIC- COMMON BUS SYSTEM, BUS AND MEMORY TRANSFER

Common Bus System

A bus structure, is more efficient for transferring information between registers in a multi- register configuration system. A bus consists of a set of common lines, one for each bit of register, through which binary information is transferred one at a time. A basic computer has 8 registers, memory unit and a control unit. The diagram of the common bus system is as shown below.

Connections:

The outputs of all the registers except the OUTR (output register) are connected to the common bus. The output selected depends upon the binary value of variables S2, S1 and S0. The lines from common bus are connected to the inputs of the registers and memory.



A register receives the information from the bus when its LD (load) input is activated while in case of memory the Write input must be enabled to receive the information. The contents of memory are placed onto the bus when its Read input is activated.

Various Registers:

4 registers DR, AC, IR and TR have 16 bits and 2 registers AR and PC have 12 bits. The INPR and OUTR have 8 bits each. The INPR receives character from input device and delivers it to the AC while the OUTR receives character from AC and transfers it to the output device. 5 registers have 3 control inputs LD (load), INR (increment) and CLR (clear). These types of registers are similar to a binary counter.

Adder and logic circuit:

The adder and logic circuit provides the 16 inputs of AC. This circuit has 3 sets of inputs. One set comes from the outputs of AC which implements register micro operations. The other set comes from the DR (data register) which are used to perform arithmetic and logic micro operations. The result of these operations is sent to AC while the end around carry is stored in E.

Bus and Memory Transfer

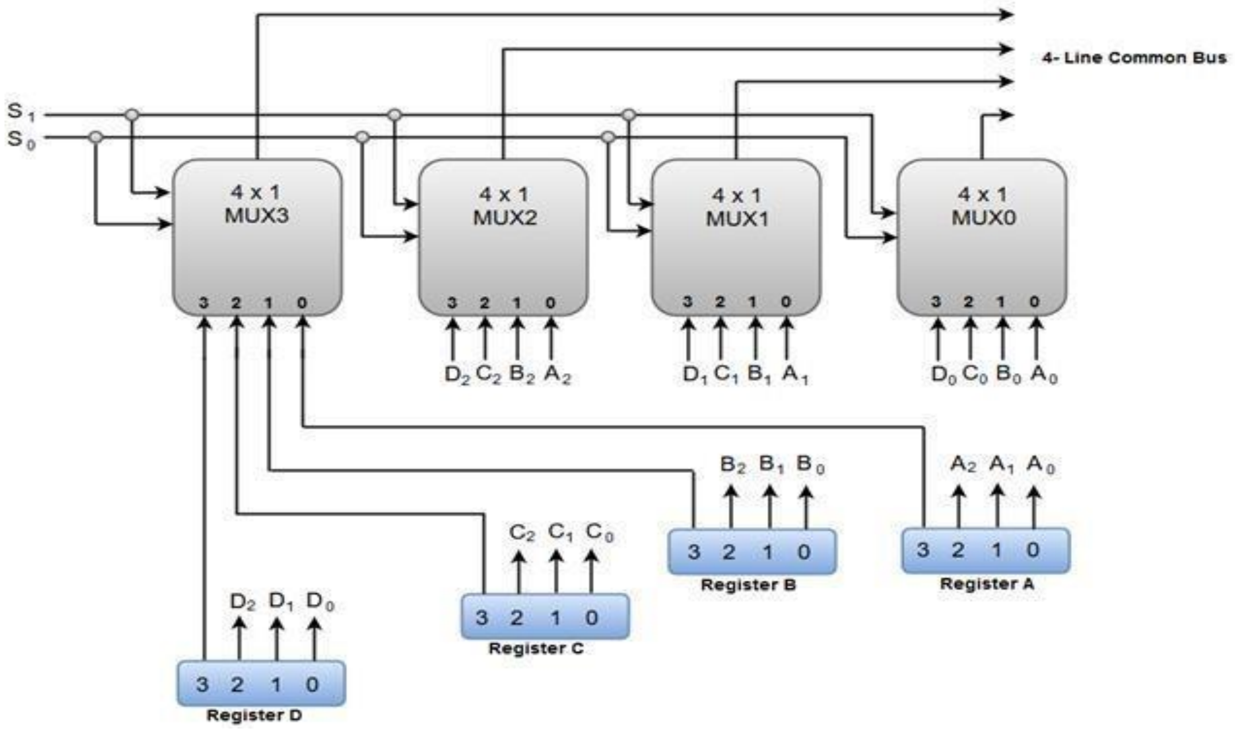
A digital system composed of many registers, and paths must be provided to transfer information from one register to another. The number of wires connecting all of the registers will be excessive if separate lines are used between each register and all other registers in the system.

A bus structure, on the other hand, is more efficient for transferring information between registers in a multi-register configuration system.

A bus consists of a set of common lines, one for each bit of register, through which binary information is transferred one at a time. Control signals determine which register is selected by the bus during a particular register transfer.

The following block diagram shows a Bus system for four registers. It is constructed with the help of four 4 * 1 Multiplexers each having four data inputs (0 through 3) and two selection inputs (S1 and S2).

Bus System for 4 Registers:



The two selection lines S1 and S2 are connected to the selection inputs of all four multiplexers. The selection lines choose the four bits of one register and transfer them into the four-line common bus.

When both of the select lines are at low logic, i.e. S1S0 = 00, the 0 data inputs of all four multiplexers are selected and applied to the outputs that forms the bus. This, in turn, causes the bus lines to receive the content of register A since the outputs of this register are connected to the 0 data inputs of the multiplexers.

Similarly, when S1S0 = 01, register B is selected, and the bus lines will receive the content provided by register B.

The following function table shows the register that is selected by the bus for each of the four possible binary values of the Selection lines.

S1	**S0**	**Register Selected**
0	0	A
0	1	B
1	0	C
1	1	D

Memory Transfer

The standard notations used for specifying operations on memory transfer are stated below:

- The transfer of information from a memory unit to the user end is called a **Read** operation.
- The transfer of new information to be stored in the memory is called a **Write** operation.
- A memory word is designated by the letter **M**.
- We must specify the address of memory word while writing the memory transfer operations.
- The **address register** is designated by **AR** and the **data register** by **DR**.
- Thus, a read operation can be stated as:

Read: DR ← M [AR]

- The **Read** statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR).
- And the corresponding write operation can be stated as:

Write: M [AR] ← R1

- The Write statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).

TOPIC- INSTRUCTION CODE, INTRODUCTION TO COMPUTER INSTRUCTIONS

Instruction codes

The organization of the computer is defined by its internal registers, the timing and control structure, and the set of instructions that it uses. Internal organization of a computer is defined by the sequence of micro-operations it performs on data stored in its registers. Computer can be instructed about the specific sequence of operations it must perform. User controls this process by means of a Program.

Program: set of instructions that specify the operations, operands, and the sequence by which processing has to occur.

Instruction: a binary code that specifies a sequence of micro-operations for the computer.

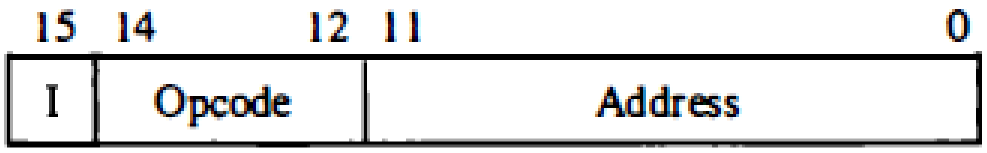
The computer reads each instruction from memory and places it in a control register. The control then interprets the binary code of the instruction and proceeds to execute it by issuing a sequence of micro-operations. – Instruction Cycle

Instruction Code: group of bits that instruct the computer to perform specific operation. Instruction code is usually divided into two parts: Opcode and address(operand)

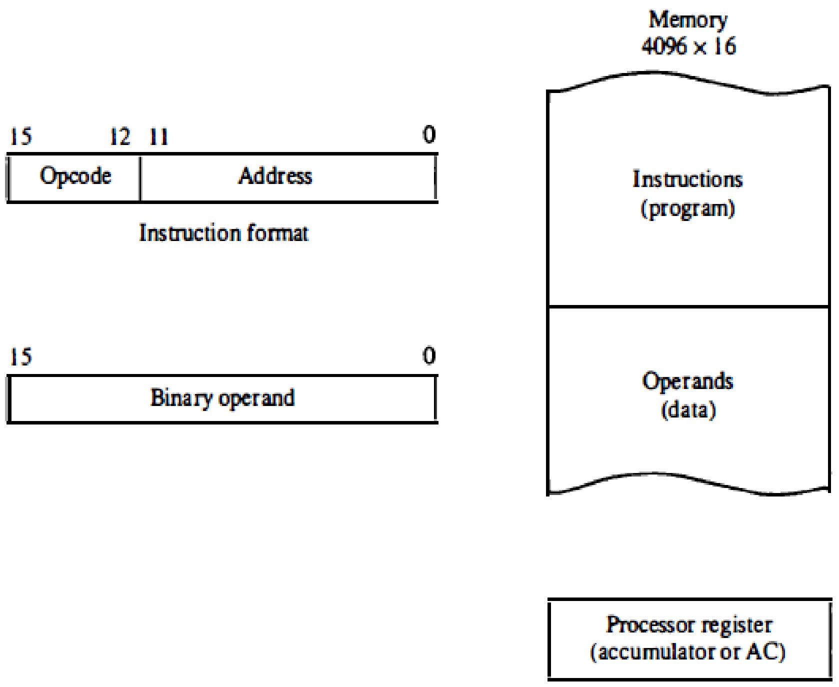
The **operation code** of an instruction is a group of bits that define such operations as add, subtract, multiply, shift, and complement. The **operation part** of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory.

The bit I specifies the addressing mode. If I=0, means direct addressing.If I=1, means indirect addressing.

An **effective address** is the address of the operand.



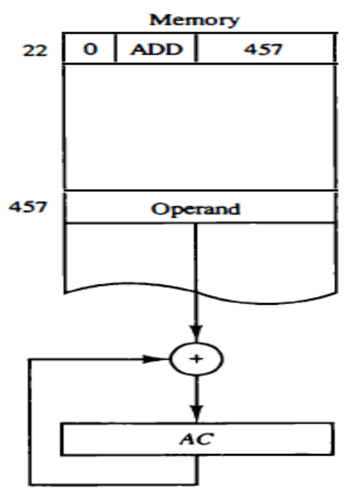
(a) Instruction format



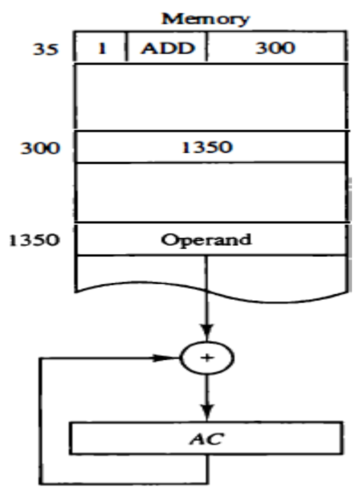
Stored program organization

Direct and Indirect Address

When the second part of an instruction code specifies an operand, the instruction is said to have an immediate operand. When the second part specifies the address of an operand, the instruction is said to have a direct address. When the bits in the second part of the instruction designate an address of a memory word in which the address of the operand is found, the instruction is said to an indirect address. One bit of the instruction code can be used to distinguish between a direct and an indirect address.



(b) Direct address



(c) Indirect address

Demonstration of direct and indirect address.

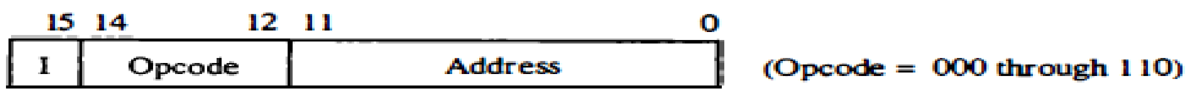
Introduction to Computer Instructions

A computer performs tasks based on certain instructions provided to it. Instructions provided to a computer are divided into different fields or groups. Each field provides some specific information. The contents of a field are written in binary language. The instruction register of a basic computer is 16 bits. Instructions comprise the following fields:

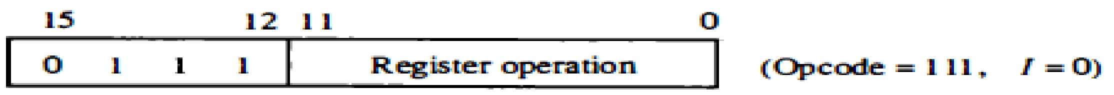
- **Operation field:** This field indicates what operation is to be performed, for example, addition, subtraction, multiplication, etc.
- **Address field:** This field indicates the memory location of the operand.
- **Mode field:** This field indicates how the memory address of the operand is determined.

A computer instruction can be in one of the following formats or has three types of instruction code formats:

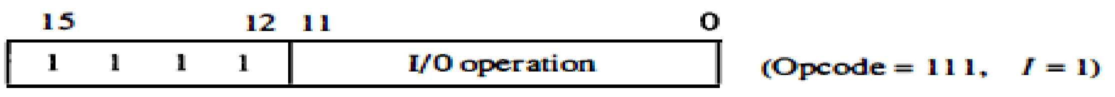
- Memory reference instructions
- Register reference instructions
- Input-output instructions



(a) Memory – reference instruction



(b) Register – reference instruction



(c) Input – output instruction

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.

Basic computer instruction formats

If the three opcode bits in positions 12 to 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.

A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I = 0 for direct address and I = 1 for indirect address.

If the 3-bit opcode = 111, control then inspects the bit in position 15. If this bit = 0, the instruction is a **register-reference type**. These instructions use 16 bits to specify an operation.

If the bit I = 1, the instruction is **an input-output type**. These instructions also use all 16 bits to specify an operation.

TOPIC- MEMORY REFERENCE INSTRUCTIONS

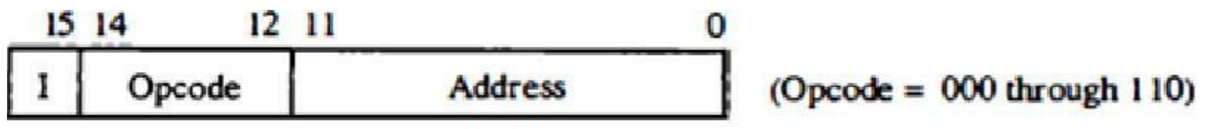
Memory Reference Instructions

A **memory reference instruction is** a type of machine language instruction used in computer programming that involves the accessing of memory locations to retrieve or store data. Memory reference instructions are critical for data processing in computer programs, as they allow the program to access and manipulate data stored in memory.

This type of instruction is divided into three parts-mode, opcode and address. The first 12 bits of memory (0-11) specify an operation address. The next three bits (12-14) specify an opcode, and the last bit (I) specifies the addressing mode. If I is 0, it specifies a direct addressing mode, and if I is 1, it specifies an indirect addressing mode.

In an Instruction format of a specific instruction, if the three opcode bits in positions 12 to 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.

A memory-reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I = 0 for direct address and I = 1 for indirect address.



(a) Memory – reference instruction

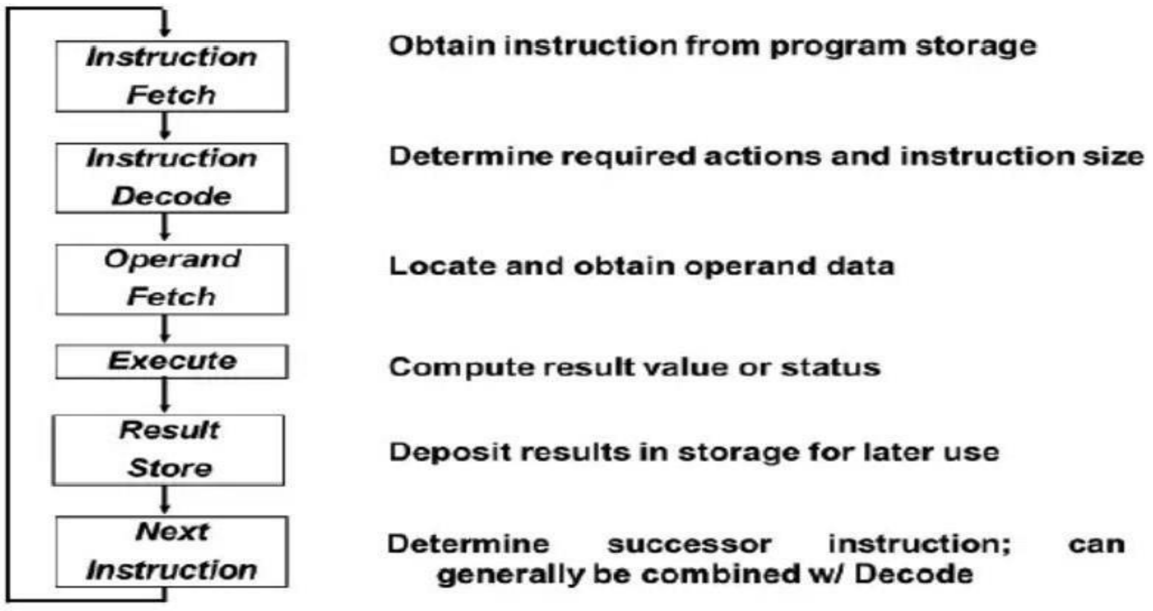
List of Memory-reference instructions

	Hexadecimal code		
Symbol	$I = 0$	$I = 1$	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

- The effective address of the instruction is in AR and was placed there during timing signal T2 when I = 0, or during timing signal T3when I = 1.
- Data from memory must be accessed to processor during T4.

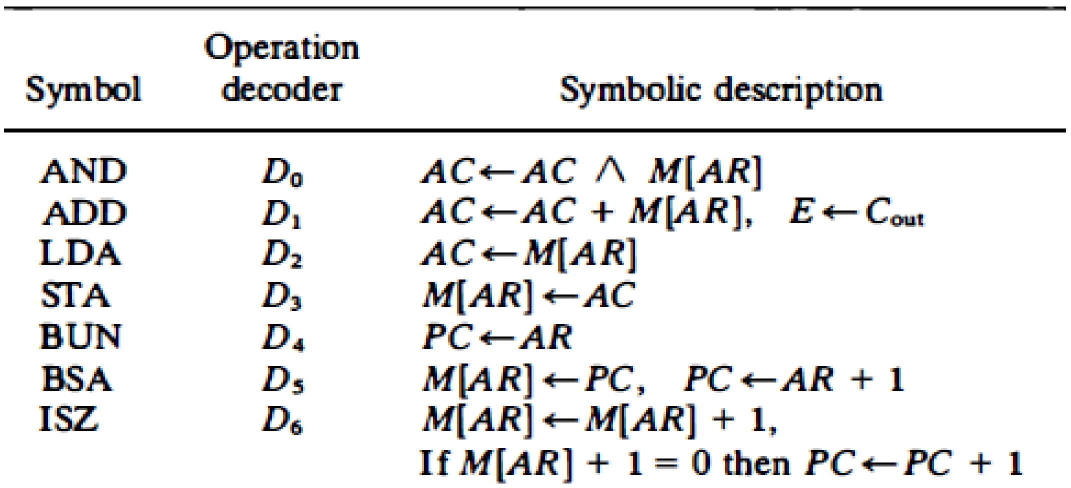
- The execution of MR instruction starts with T4. The complete execution of this type of instructions will require a sequence of micro-operations during T4 and T5 and likely T6.

The operands specified by memory reference instructions are


Instruction Fetch
Obtain instruction from program storage
Instruction Decode
Determine required actions and instruction size
Operand Fetch
Locate and obtain operand data
Execute
Compute result value or status
Result Store
Deposit results in storage for later use
Next Instruction
Determine successor instruction; can generally be combined w/ Decode

A 3 by 8 decoder is used to decode the 3 bits to 8 lines D0 to D7, although D7 not used here. The effective address of the instruction is in AR and was placed there during timing signal T2 when I = 0, or during timing signal T3when I = 1. Data from memory must be accessed to processor during

T4. The execution of MR instruction starts with T4. The complete execution of this type of instructions will require a sequence of micro-operations during T4 and T5 and likely T6.

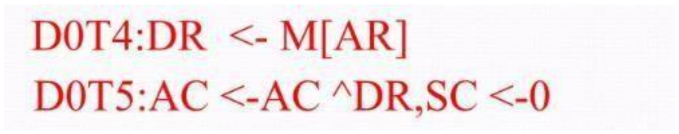


Symbol	Operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], \quad E \leftarrow C_{out}$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC, \quad PC \leftarrow AR + 1$
ISZ	D_6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

List of all memory reference

Instructions: AND: 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 micro operations that execute this instruction are:

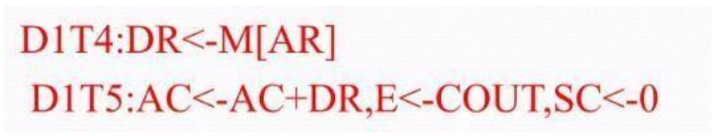


D0T4:DR <- M[AR]

D0T5:AC <-AC ^DR,SC <-0

ADD : 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 is transferred to the E (extended accumulator) flip-flop. The micro operations needed to execute this instruction are:



D1T4:DR<-M[AR]

D1T5:AC<-AC+DR,E<-COUT,SC<-0

LDA: Load to AC

This instruction transfers the memory word specified by the effective address to AC. The micro operations needed to execute this instruction are:

D2T4:DR<-M[AR]

D2T5:AC<-DR,SC<-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 micro operation:

D3T4:M[AR]<-AC,SC<-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 micro operation:

D4 T4: PC ← AR, SC ← 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.

M[AR]<-PC, PC<-AR+1.

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.

M[135]<-21,PC<-135+1=136.

This is shown in part (a) of the figure 1.23. The BSA instruction performs the following numerical operation:

The result of this operation is shown in part (b) of the figure. The return address 21 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.

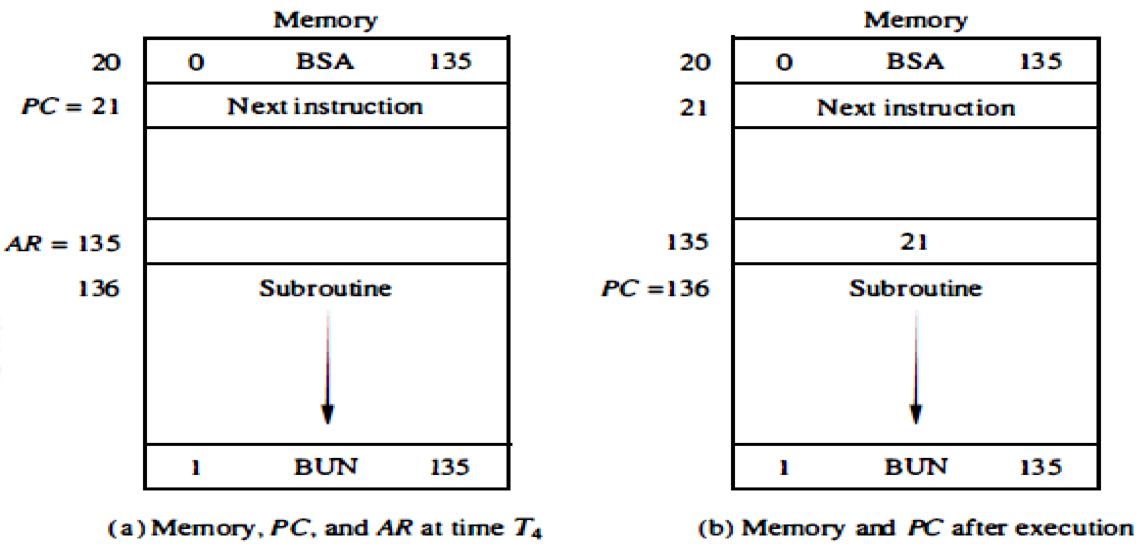


Figure1.23: Branch and save return address

BSA instruction must be executed with a sequence of two microoperations:

D5T4 :M[AR]<-PC,AR <-+1

D5T5 :PC<-AR,SC<-0

Timing signal T4 initiates a memory write operation, places the content of PC onto the bus, and enables the INR input of AR . The memory write operation is completed and AR is incremented by the time the next clock transition occurs. The bus is used at T5 to transfer the content of AR to PC.

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. The programmer usually stores a negative number (in 2's complement) in the memory word. As this negative number is repeatedly incremented by one, it eventually reaches the value of zero. At that time PC is incremented by one in order to skip the next instruction in the program.

D6T4:DR<-M[AR].

D6T5: DR<-DR+1.

D6T6:M[AR]<-DR ,if(DR=0)then (PC<-PC+1),SC<-0.

To execute the ISZ instruction, three timing signals are needed:

1. The clock transition associated with timing signal T4 read the memory into DR.
2. The clock transition associated with timing signal T5 increments DR.
3. The clock transition associated with timing signal T6 store the word back into memory.

In the same clock transition, SC is cleared to 0 which transfers the control to timing signal T0 to start a new instruction cycle.

A flowchart of all micro-operations for the execution of the seven memory- reference instructions is shown below:

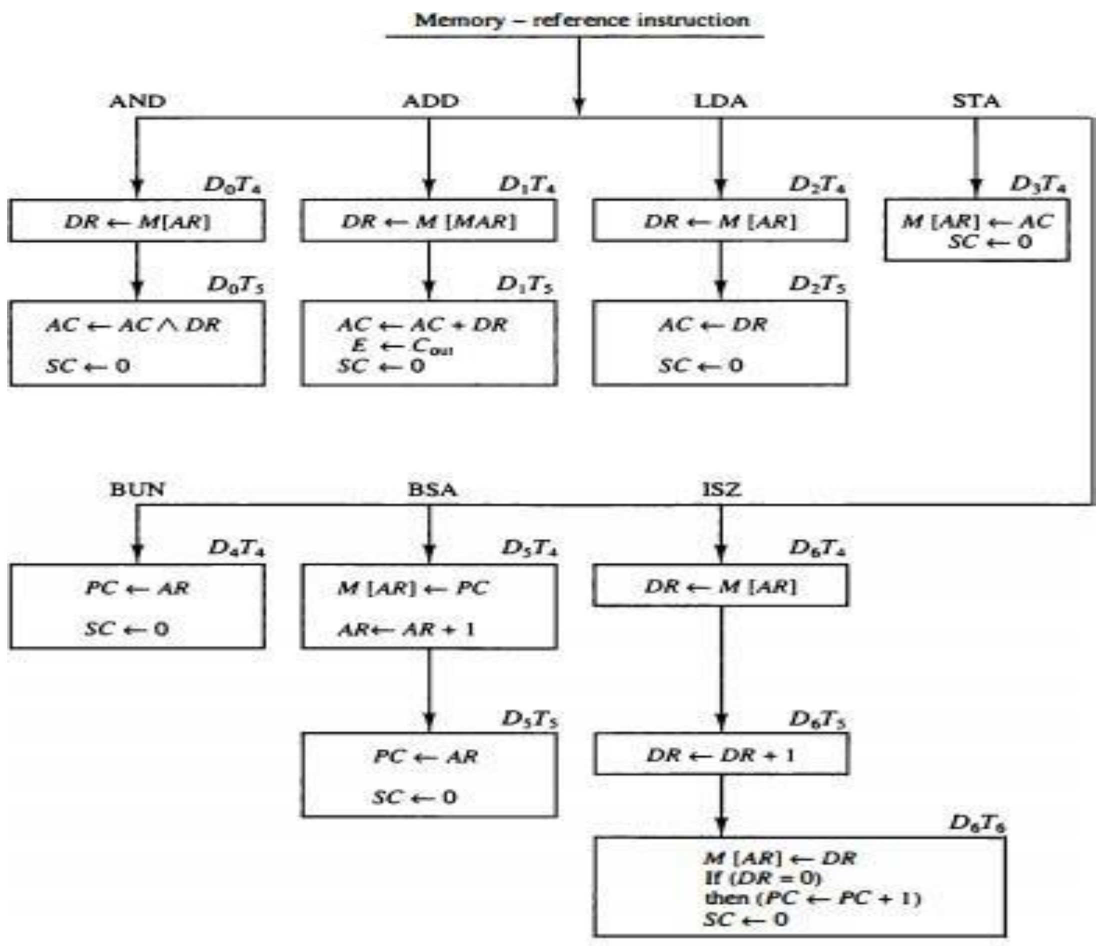


Figure Flowchart for memory-reference instructions.

TOPIC- REGISTER REFERENCE INSTRUCTIONS

Register Reference Instruction

Register reference instruction is a computer instruction that operates on the data stored in the registers instead of memory addresses. Hence, the register reference instructions are primarily used for manipulation of data stored in the registers of the processors of a computer.

Similar to memory reference instructions, register reference instructions can also perform operation like data manipulation, arithmetic operations, logical operations, etc. on the data stored in registers. In comparison to memory reference instructions, the register reference instructions are faster in execution.

Register reference instruction consists of two parts namely: opcode, and register operation. The opcode (operational code) represents the operation to be performed.

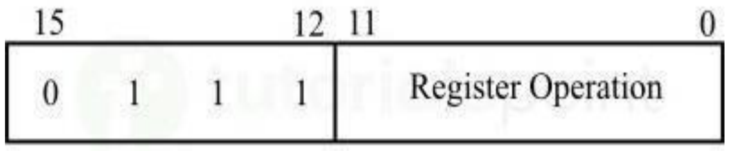


Figure - Register Reference Instruction

In computer organization, the register reference instructions are determined by their opcode 111 with a 0 at the left most of the instruction. In the case of register reference instructions, there is no need of an operand from memory because it uses additional 12 bits to identify the operation to be implemented.

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.

These instructions are executed with the clock transition associated with timing variable T3.Control function needs the Boolean relation D7I'T3, which we designate for convenience by the symbol r. By assigning the symbol Bi to bit i of IR, all control functions can be simply denoted by rBi.

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 D7.

Bit 11 in IR is 1 and is recognized from B11. The control function that initiates the microoperation for this instruction is D7I'T3 B11 = rB11. The execution of a register-reference instruction is completed at time T3.

TABLE Execution of Register-Reference Instructions

$D_7I'T_3 = r$ (common to all register-reference instructions)
$IR(i) = B_i$ [bit in $IR(0\text{–}11)$ 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

The sequence counter SC is cleared to 0 and the control goes back to fetch the next instruction with timing signal T0. 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.

INPUT-OUTPUT INSTRUCTIONS

Input-Output Instructions

Those computer instructions that provide a mean for communication between a computer system and its I/O peripherals are called input-output instructions.

Therefore, the input-output instructions enable the data transfer from and to input and output devices like keyboard, memory disks, monitor, etc.

With the help of input-output instructions, the computer system receives data input from users and sends output to users. Hence, input-output instructions act as the communication interface between CPU and peripheral devices of the computer system.

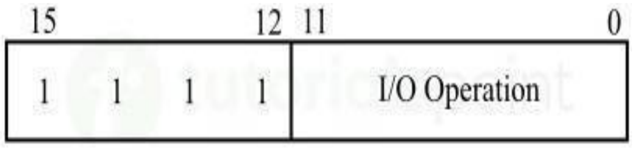


Figure - Input/ Output Instruction

Similar to register reference instructions, the input-output instructions also do not require a memory reference. It is identified by the opcode 111 with a 1 at the leftmost of the instruction. The rest 12-bits are used to represent the type of input/output operation to be performed.

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 micro operations for the input-output instructions are listed in Table These

instructions are executed with the clock transition associated with timing signal T3.

Each control function needs a Boolean relation D7IT3, which we designate for convenience by the symbol p. The control function is distinguished by one of the bits in IR (6-11).

By assigning the symbol Bi to bit i of IR, all control functions can be denoted by pBi for i = 6 though 11.

The sequence counter SC is cleared to 0 when p = D7IT3 = 1.

The last two instructions set and clear an interrupt enable flip-flop IEN according to two instructions as follows:

1. With the IOF instruction, the IEN is cleared to 0 and the input or output flags cannot interrupt the computer.

2. With the ION instruction, the IEN is set to 1, and the computer can be interrupted.

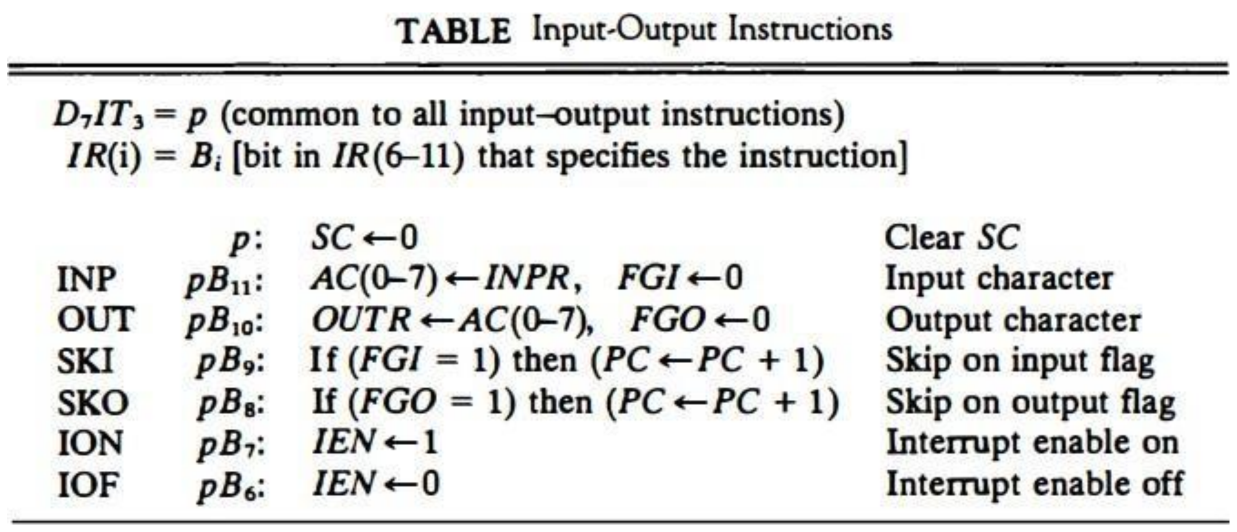


TABLE Input-Output Instructions

$D_7IT_3 = p$ (common to all input–output instructions)

$IR(i) = B_i$ [bit in $IR(6–11)$ that specifies the instruction]

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

The different types of I/O operations are as follows:

Symbol	**Description**	**Hexadecimal code**
IOF	Interrupt off	F040
ION	Interrupt on	F080
SKO	Skip on flag output	F100
SKI	Skip on flag input	F200
OUT	Output the contents from an accumulator	F400
INP	Input a character to accumulator	F800

TIMING AND CONTROL

Timing and Control

CPU is partitioned into Arithmetic Logic Unit (ALU) and Control Unit (CU). The function of control unit is to generate relevant timing and control signals to all operations in the computer. It controls the flow of data between the processor and memory and peripherals.

The control unit directs the entire computer system to carry out stored program instructions. The control unit must communicate with both the arithmetic logic unit (ALU) and main memory. The control unit instructs the arithmetic logic unit that which logical or arithmetic operation is to be performed. The control unit co-ordinates the activities of the other two units as well as all peripherals and auxiliary storage devices linked to the computer.

Design of Control unit:

Control unit generates control signals using one of the two organizations:

Hardwired Control Unit

It is implemented as logic circuits (gates, flip-flops, decoders etc.) in the hardware. This organization is very complicated if we have a large control unit. In this organization, if the design has to be modified or changed, requires changes in the wiring among the various components. Thus the modification of all the combinational circuits may be very difficult.

Advantages:

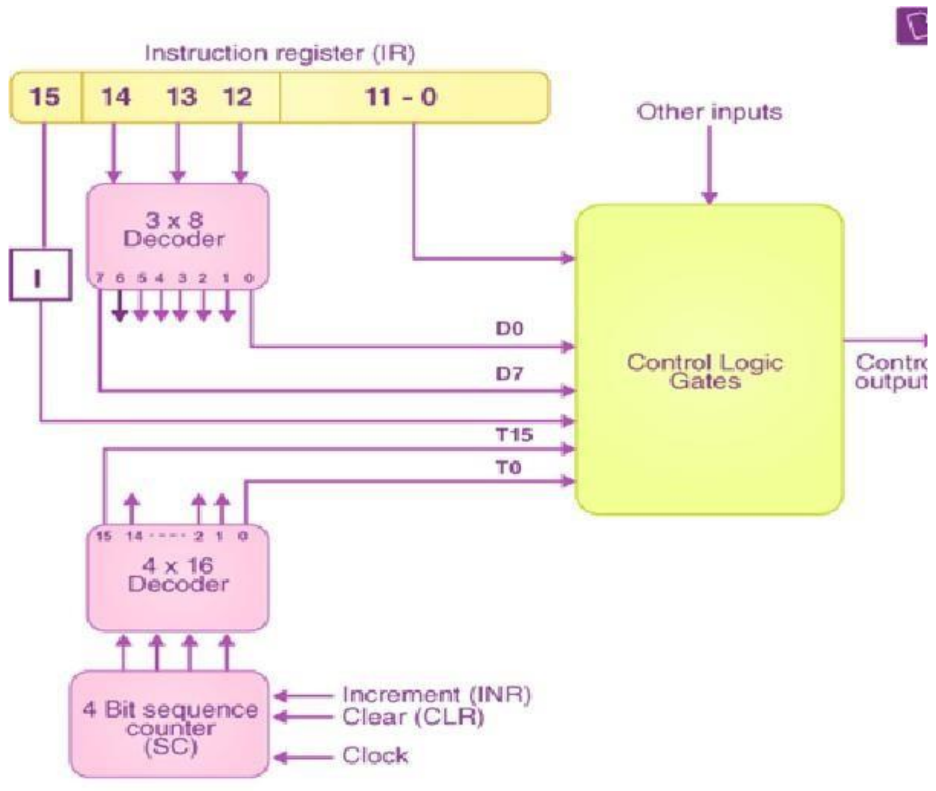
- Hardwired Control Unit is fast because control signals are generated by combinational circuits.
- The delay in generation of control signals depends upon the number of gates.

Disadvantages:

- More is the control signals required by CPU; more complex will be the design of control unit.
- Modifications in control signal are very difficult. That means it requires rearranging of wires in the hardware circuit.
- It is difficult to correct mistake in original design or adding new feature in existing design of control unit.

Characteristics of Hardwired Control Unit

- Two decoders, sequence counter and logic gates make up a Hardwired Control.
- The instruction register stores an instruction retrieved from the memory unit (IR).
- An instruction register consists of the operation code, the I bit, and bits 0 through 11.
- A 3 x 8 decoder is used to encode the operation code in bits 12 through 14.
- The decoder's outputs are denoted by the letters D0 through D7.
- The bit 15 operation code is transferred to a flip-flop with the symbol I.
- The control logic gates are programmed with operation codes from bits 0 to 11.
- The sequence counter (or SC) can count from 0 to 15 in binary.



Control Unit of a Basic Computer

An instruction read from memory is placed in the instruction register (IR). The instruction register is divided into three parts: the I bit, operation code, and address part.

First 12-bits (0-11) to specify an address, next 3-bits specify the operation code (opcode) field of the instruction and last left most bit specify the addressing mode I. I = 0 for direct address I = 1 for indirect address.

First 12-bits (0-11) are applied to the control logic gates. The operation code bits (12 – 14) are decoded with a 3 x 8 decoder. The eight outputs (D0 through D7) from a decoder goes to the control logic gates to perform specific operation. Last bit 15 is transferred to a I flip-flop designated by symbol I.

The 4-bit sequence counter SC can count in binary from 0 through 15. The counter output is decoded into 16 timing pulses T0 through T15. The sequence counter can be incremented by INR input or clear by CLR input synchronously.

For example: Consider the case where SC is incremented to provide timing signalsT0, T1, T2 , T3, and T4 in sequence. At time T4 , SC is cleared to 0 if decoder output D3 is active. This is expressed symbolically by the statement: $D_3 T_4 : SC \leftarrow 0$ The timing diagram shows the time relationship of the control signals.

Timing Diagram

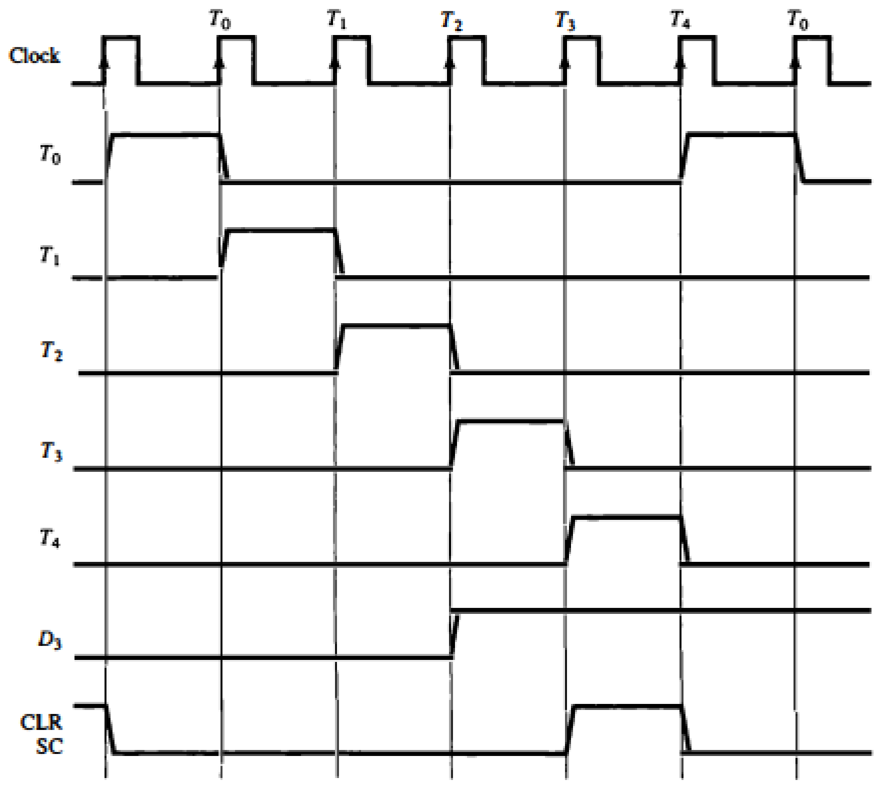


Figure Example of control timing signals.

Micro-programmed Control Unit

A micro-programmed control unit is implemented using programming approach. A sequence of micro-operations is carried out by executing a program consisting of micro- instructions. Micro-program, consisting of micro-instructions is stored in the control memory of the control unit. Execution of a micro-instruction is responsible for generation of a set of control signals.

A micro-instruction consists of:

- One or more micro-operations to be executed.
- Address of next microinstruction to be executed.

Micro-Operations: The operations performed on the data stored inside the registers are called micro-operations.

Micro-Programs: Microprogramming is the concept for generating control signals using programs. These programs are called micro-programs.

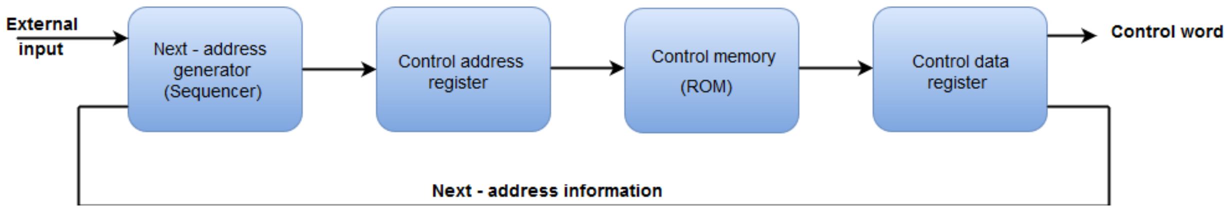
Advantages:

- The design of micro-program control unit is less complex because micro-programs are implemented using software routines.
- The micro-programmed control unit is more flexible because design modifications, correction and enhancement is easily possible.
- The new or modified instruction set of CPU can be easily implemented by simply rewriting or modifying the contents of control memory.
- The fault can be easily diagnosed in the micro-program control unit using diagnostics tools by maintaining the contents of flags, registers and counters.

Disadvantages:

- The micro-program control unit is slower than hardwired control unit. That means to execute an instruction in micro-program control unit requires more time.
- The micro-program control unit is expensive than hardwired control unit in case of limited hardware resources.
- The design duration of micro-program control unit is more than hardwired control unit for smaller CPU.

Architecture of Micro-Programmed Control Unit



The Control memory address register specifies the address of the micro-instruction. The Control memory is assumed to be a ROM, within which all control information is permanently stored. The control register holds the microinstruction fetched from the memory.

The micro-instruction contains a control word that specifies one or more micro-operations for the data processor. While the micro-operations are being executed, the next address is computed in the next address generator circuit and then transferred into the control address register to read the next microinstruction. The next address generator is often referred to as a micro-program sequencer, as it determines the address sequence that is read from control memory.

Hardwired vs Micro-Programmed Control Unit

	Hardwired Control Unit	**Micro-programmed Control Unit**
Implementation	Fixed set of logic gates and circuits	Microcode stored in memory
Flexibility	Less flexible, difficult to modify	More flexible, easier to modify
Instruction Set	Supports limited instruction sets	Supports complex instruction sets
Complexity of Design	Simple design, easy to implement	Complex design, more difficult to implement
Speed	Fast operation	Slower operation due to microcode decoding
Debugging and Testing	Difficult to debug and test	Easier to debug and test
Size and Cost	Smaller size, lower cost	Larger size, higher cost
Maintenance and Upgradability	Difficult to upgrade and maintain	Easier to upgrade and maintain

INSTRUCTION CYCLE

Instruction Cycle

A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of the following phases:

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.

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 T0. After each clock pulse, SC is incremented by one, so that the timing signals go through a sequence T0, T1, T2, and so on. The micro operations for the fetch and decode phases can be specified by the following register transfer statements

T0: AR ← PC ($S_0S_1S_2$=010, T0=1)
T1: IR ← M [AR], PC ← PC + 1 (S0S1S2=111, T1=1)
T2: D0, . . . , D7 ← 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 T0. The instruction read from memory is then placed in the instruction register IR with the clock transition associated with timing signal T1.

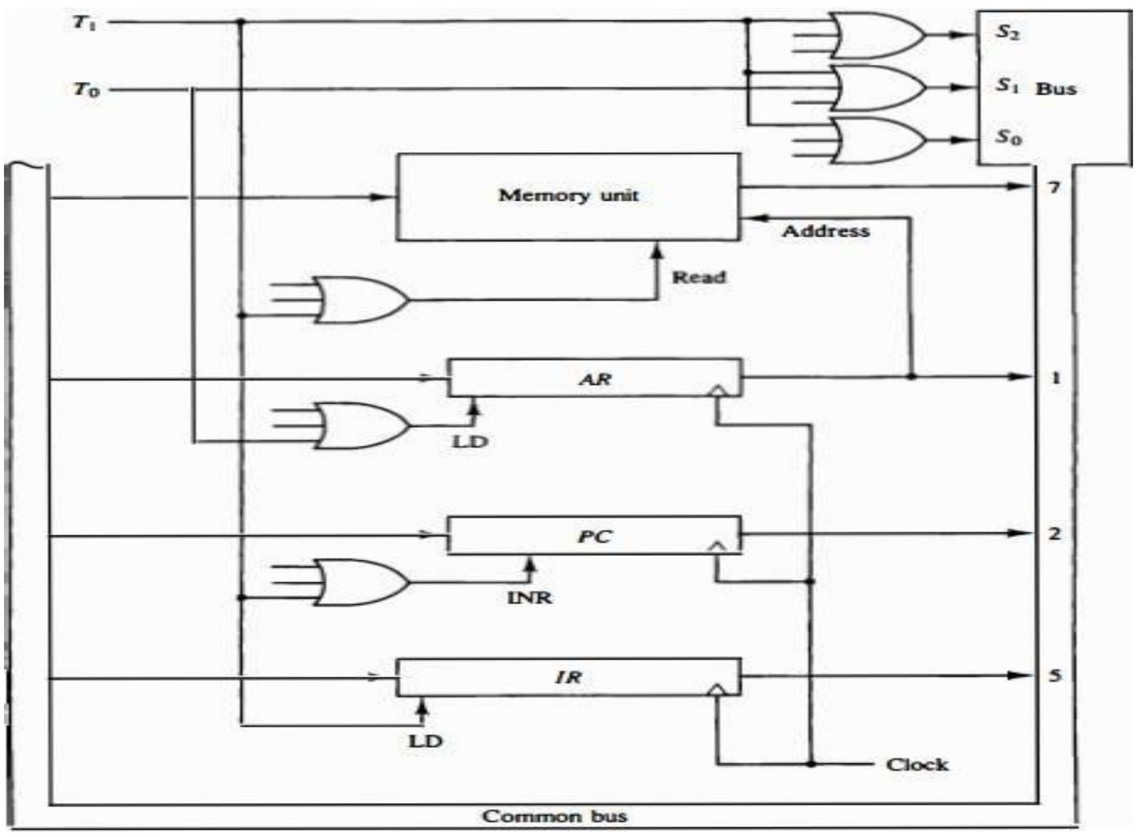


Figure Register transfers for the fetch phase.

At the same time, PC is incremented by one to prepare it for the address of the next instruction in the program.

At time T2, 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 that SC is incremented after each clock pulse to produce the sequence T0, T1, and T2.

Figure above shows how the first two register transfer statements are implemented in the bus system.

To provide the data path for the transfer of PC to AR we must apply timing signal T0 to achieve the following connection:

1. Place the content of PC onto the bus by making the bus selection inputs S2S1S0 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 T0 = 1.

In order to implement the second statement :

T1: IR $\leftarrow$ M[AR], PC $\leftarrow$ 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 S2S1S0 = 111.
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.

The next clock transition initiates the read and increment operations since T1 = 1.

Figure above duplicates a portion of the bus system and shows how T0 and T1 are connected to the control inputs of the registers, the memory, and the bus selection inputs.

Determine the Type of Instruction

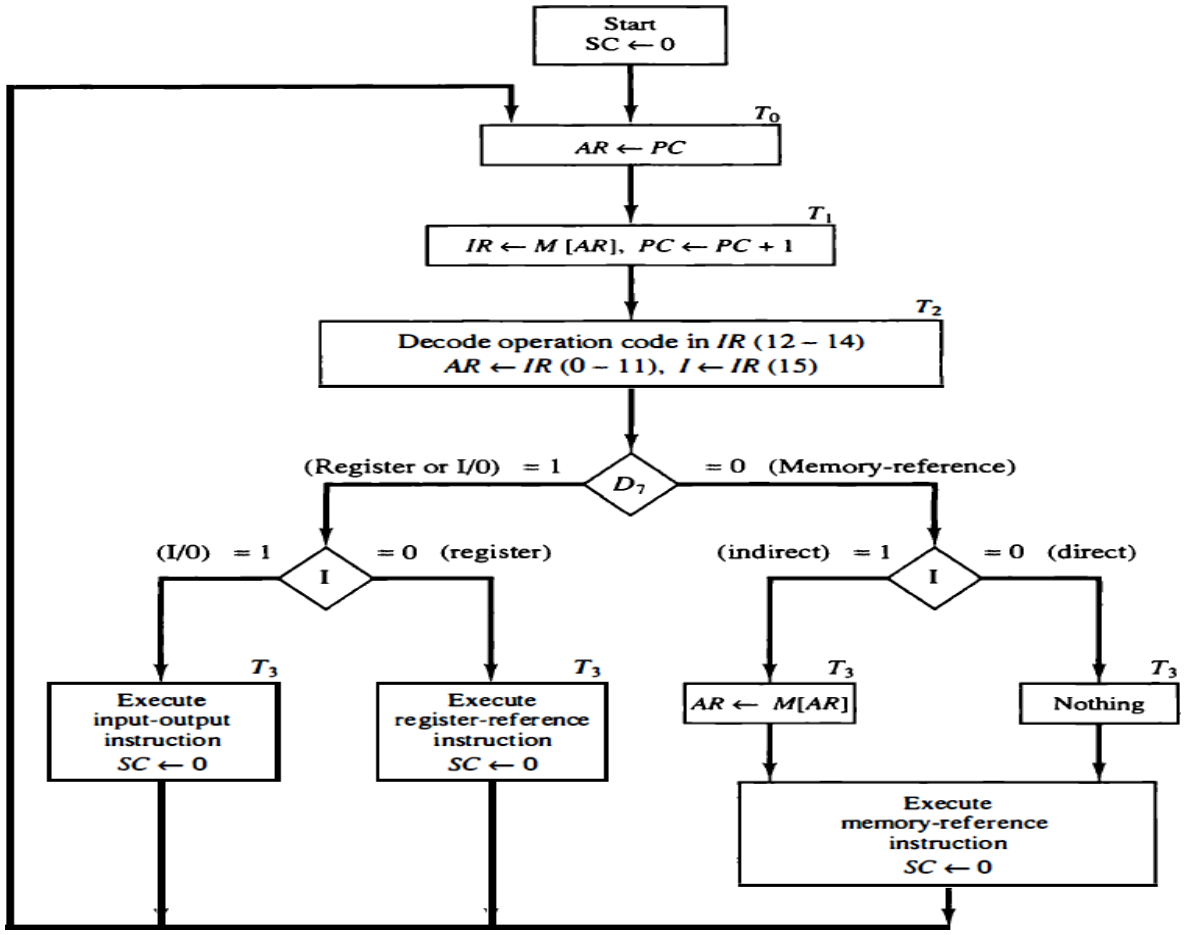
The timing signal that is active after the decoding is T_3.
During time T_3 the control unit determines the type of instruction that was just read from memory.
Decoder output D_7 is equal to 1 if the operation code is equal to binary 111.

- If $D_7 = 1$, the instruction must be a register-reference or input-0utput type.
- If $D_7 = 0$, the operation code must be one of the other seven values 000 through 110, specifying a memory-reference instruction.

$D_7' I T_3$: $AR \leftarrow M[AR]$
$D_7' I' T_3$: Nothing
$D_7 I' T_3$: Execute a register-reference instruction
$D_7 I T_3$: Execute an input–output instruction

When a memory-reference instruction with I = 0 is encountered, it is not necessary to do anything since the effective address is already in AR. However, the sequence counter SC must be incremented when $D_7'T_3 = 1$, so that the execution of the memory-reference instruction can be continued with timing variable T4.

A register-reference or input-output instruction can be executed with the clock associated with timing signal T_3. After the instruction is executed, SC is cleared to 0 and control returns to the fetch phase with $T_0 = 1$.



Start
SC ← 0
T_0
$AR \leftarrow PC$
T_1
$IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$
T_2
Decode operation code in IR (12 – 14)
$AR \leftarrow IR$ (0 – 11), $I \leftarrow IR$ (15)
(Register or I/O) = 1
D_7
= 0 (Memory-reference)
(I/O) = 1
I
= 0 (register)
(indirect) = 1
I
= 0 (direct)
T_3
Execute input-output instruction $SC \leftarrow 0$
T_3
Execute register-reference instruction $SC \leftarrow 0$
T_3
$AR \leftarrow M[AR]$
T_3
Nothing
Execute memory-reference instruction $SC \leftarrow 0$

Figure: Flowchart of Instruction cycle

INTERRUPT CYCLE

Interrupt cycle

The difference of information flow rate between the computer and the input-output device makes this type of transfer inefficient.

Suppose that the processor is transferring data to a I/O device (let's say a printer) using the instruction cycle. However, most external devices are much slower than the processor.

So after each write operation, the processor has to pause and remain idle until the printer catches up. The length of this pause may be equivalent to hundreds or even thousands of instruction cycles that could have occurred. Clearly, this is a very wasteful use of the processor.

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.

An interrupt is just that: **an interruption of the normal sequence of execution**. When the interrupt processing is completed, execution resumes.With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.

While the computer is running a program, it does not check the flags. However, when a flag is set ,i.e When the external device becomes ready to accept more data from the processor, the I/O module for that external device sends an **interrupt request signal** to the processor.

The processor responds by suspending the current program, branching off to a program to accept data from that I/O device (known as an **interrupt handler**), It then returns to the current program to continue what it was doing before the interrupt.

The interrupt enable flip-flop lEN can be set and cleared with two instructions (IOF and ION instructions).

To accommodate interrupts, an interrupt cycle is added to the instruction cycle.
The way that the interrupt is handled by the computer can be explained by means of the flowchart of Figure.

- ❑ 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 lEN 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 lEN 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 lEN = 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.

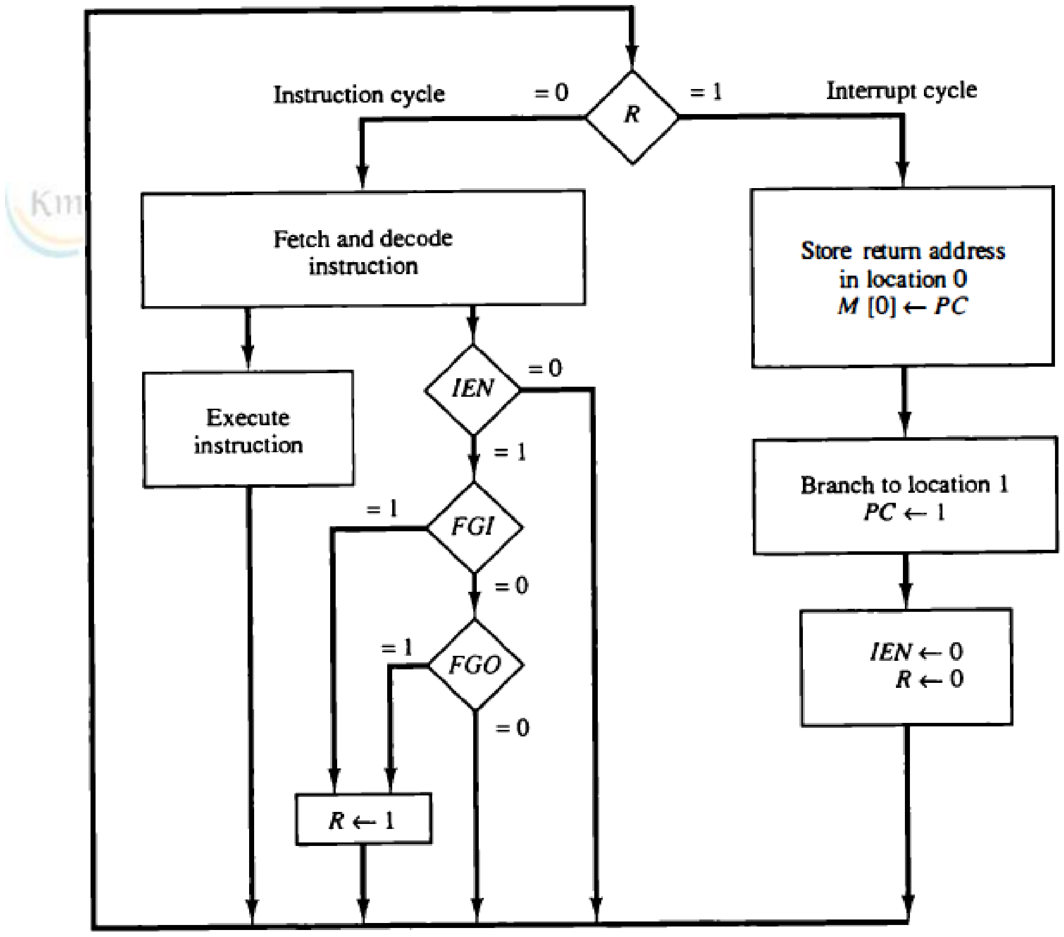
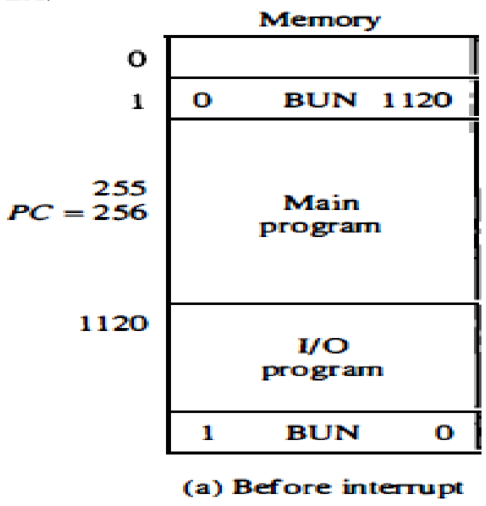


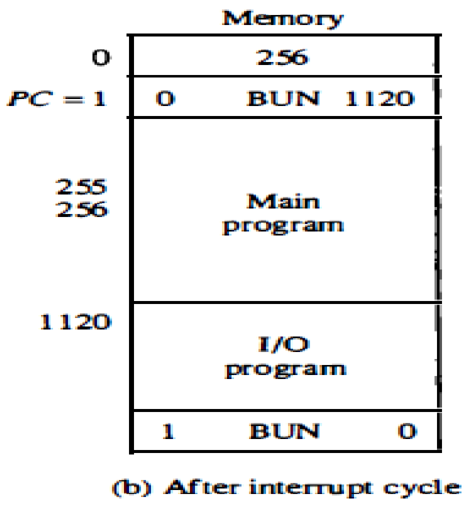
Figure: Flowchart for interrupt cycle

The interrupt cycle is a hardware implementation of a **branch and save return address(BSA)**operation.

EX:



(a) Before interrupt



(b) After interrupt cycle

Figure : Demonstration of Interrupt Cycle

An example that shows what happens during the interrupt cycle is shown in Figure (t). Suppose that an interrupt occurs and R is set to 1 while the control is executing the instruction at address 255. At this time, the return 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 Figure (a).

When control reaches timing signal T0 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. At the beginning of the next instruction cycle, the instruction that is read from memory is in address 1 since this is the content of PC. 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 program returns to the location where it was interrupted. This is shown in Figure (b).

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 lEN = 1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T0, T1 or T2 are active. The condition for setting flip- flop R to 1 can be expressed with the following register transfer statement:

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

During the first timing signal AR is cleared to 0, and the content of PC is transferred to the temporary register TR. With the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0. The third timing signal increments PC to 1, clears lEN and R, and control goes back to T0 by clearing SC to 0. The beginning of the next instruction cycle has the condition R' T0 and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

$$\begin{aligned} &RT_0: && AR \leftarrow 0, \quad TR \leftarrow PC \\ &RT_1: && M[AR] \leftarrow TR, \quad PC \leftarrow 0 \\ &RT_2: && PC \leftarrow PC + 1, \quad IEN \leftarrow 0, \quad R \leftarrow 0, \quad SC \leftarrow 0 \end{aligned}$$