UNIT 1

DIGITAL COMPUTERS, REGISTER TRANSFER LANGUAGE AND MICROOPERATIONS, BASIC COMPUTER ORGANIZATION AND DESIGN

PART-A
SHORT QUESTIONS WITH SOLUTIONS

Q1. Define digital computer.

Answer : Model Paper-III, Q1(a)

A digital computer is a fast electronic data processing device, which takes the input in the form of instructions (often referred as programs), process it by referring to its various memory elements and finally produces the result.

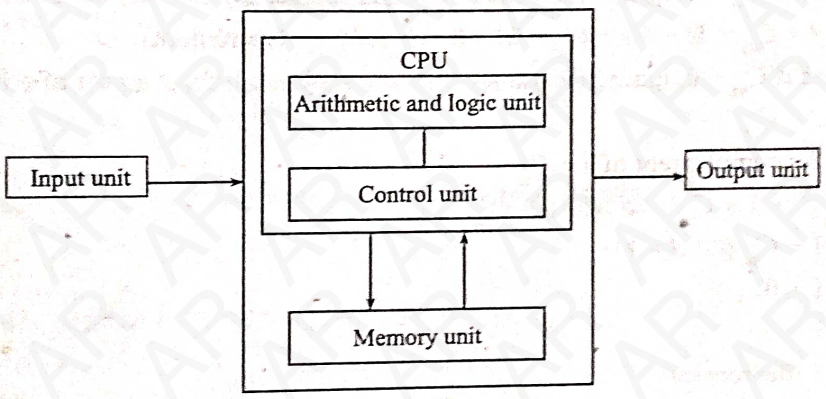


Figure: Block Diagram of a Computer

Q2. Write about memory read and memory write operations.

Answer : Model Paper-I, Q1(a)

Memory Read

A read operation can be performed by transferring data from memory word M to its surroundings.

A address register AR provides address to the memory unit. The data in the address register AR is sent to another register known as data register DR.

Read Operation

$$DR \leftarrow M[AR]$$

This read operation reads the data from the memory address specified in register AR into data register DR.

Memory Write

A write operation is performed to store newly arrived information into the memory.

The data present in the data register DR is transferred to selected address of memory word M by performing write operation.

Write Operation

$$M[AR] \leftarrow R1$$

This write operation writes the data in register $R1$ to address selected by the address register AR into the memory word M.

Q3. Write short notes on three state buffers.

Answer :

A common bus system can be implemented using tri-state or three-state gates instead of multiplexers. A tri-state gate is a digital circuit that can have three output states – HIGH, LOW and high-impedance.

The output states HIGH and LOW corresponds to logic 1 and logic 0 respectively. The output state high-impedance acts as an open circuit which means the output is disconnected.

We know that, bus is a common connection between number of registers and other units. Therefore, the data transfer through tri-state gates may require larger sinking and sourcing of current. The tri-state gate which provides more sinking and sourcing capacities is called tri-state buffer. The following figure shows the logic symbol for tri-state buffer.

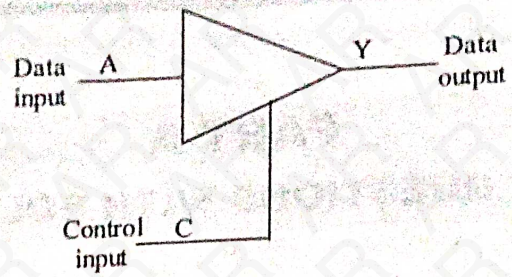


Figure: Logic Symbol for Tri-state Buffer

Q4. Write decrement and increment microoperation of 4-bit arithmetic circuit with the help of function table.

Answer :

When, $S_1S_0 = 10$ and if $C_{in} = 1$, ignore the values of N and replace all the J inputs of a full adder with 0's.

$L = M + J + C_{in} = M + 0\text{'s} + C_{in} = M + 0 + 1 = M + 1$ (increment)

When, $S_1S_0 = 11$ and if $C_{in} = 0$, ignore the values of N and replace all the J inputs of a full adder with 1's

$L = M + J + C_{in}$

$= M +$ 2's complement of $J + C_{in}$

$= M + \bar{J} + 1 + C_{in} \Rightarrow M - J + C_{in}$

$= M - 1 + C_{in}$ $\{\because J = 1\}$

$= M - 1 + 0$

$= M - 1$

$L = M - 1$ (decrement)

Selection Inputs		Carry input	Input of	Output
S_1	S_0	C_{in}	adder, J	$L = M + J + C_{in}$
1	0	1	0	L = M + 1
1	1	0	1	L = M – 1

Q5. Write substraction microoperation of 4-bit arithmetic circuit with the help of function table.

Answer : Model Paper-II, Q1(

When $S_1 S_0 = 01$ and $C_{in} = 0$, a complement of N is applied to the J inputs of a full adder and a subtract with borrow microoperation is carried out.

Output, $L = M + \bar{N} = M +$ 1's complement of N

$= M - N - 1$

When $S_1 S_0 = 01$ and $C_{in} = 1$, a 2's complement of N is applied to the J inputs of a full adder, and a subtract microoperation is carried out. Output, $L = M + \bar{N} = 1 = M +$ 2's complement of N.

Section Inputs		Carry input	Input of	Output
S_1	S_0	C_{in}	adder, J	$L = M + J + C_{in}$
0	1	0	$\bar{N}$	$L = M + \bar{N}$
0	1	1	$\bar{N}$	$L = M + \bar{N} + 1$

Q6. Discuss in brief about microinstruction.

Answer :

Model Paper-III, Q1(b)

A microinstruction is combination of several fields. These fields are assigned their own unique operations to be performed. With these blocks, there are certain bits associated for example the special bits, control bits, address bits etc. Among these bits special bits are designed to decide on the process of evaluation of forthcoming address, control bits are designated to execute several microoperations and finally address bits-look after the branching tasks. In order to reduce the number of control bits that initiate microoperations we divide these bits into fields and in code k bits in each field. Thus, these fields can provide 2t microoperations. Here, each field require special circuitry called a decoder in order to produce control signals. Hence, each field is provided with a separate unit of decoders capable of undertaking 3-bits of data and accordingly producing and control signals to initiate several tasks.

Q7. Draw a diagram that shows the register transfer implementation along with the timing diagram.

Answer :

Model Paper-I, Q1(b)

Block Diagram of Register Transfer Implementation

The figure below shows the block diagram of register transfer from register $R1$ to register $R2$.

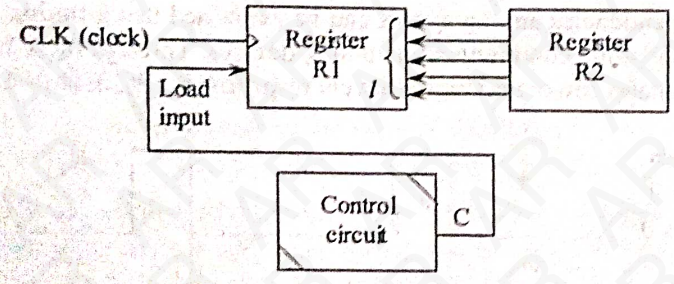


Figure (i): Block Diagram of Register Transfer

Timing Diagram

The timing diagram for the register transfer is shown in figure (ii).

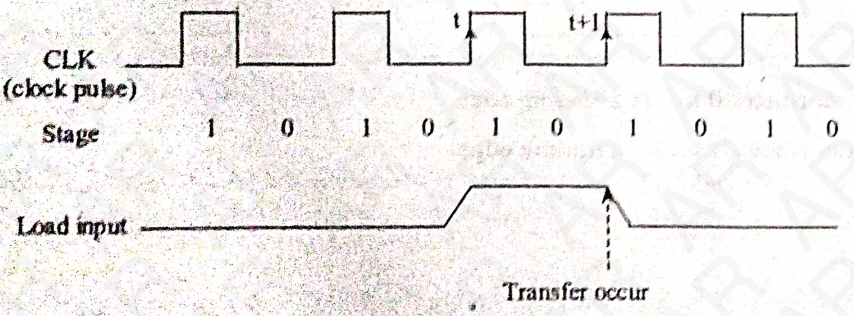


Figure (ii): Timing Diagram of Register Transfer

Q8. Discuss in short about I/O instructions.

Answer :

Input/output instructions are those that access the input/output ports through registers for execution purposes. Memory is not accessed hence the name input/output instructions. The input/output instruction format is shown in the following figure.

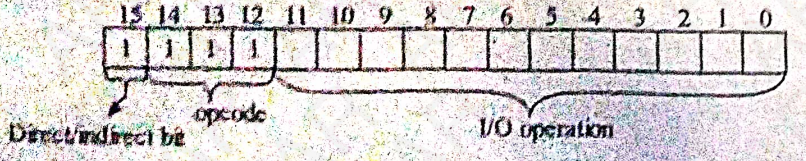


Figure: Input/Output Instruction Format

Q9. Write about decode phase of instruction cycle.

Answer :

In decode phase, the decoding of an instruction is carried out at timing signal 'T_2'.

1. A 3 × 8 Decoder is used to decode the 12-14 bits of the instruction register (IR).
2. The 0-11 bits of instruction register (IR) are loaded into the address register (AR).
3. The 15th bit of instruction register (IR) is loaded into the addressing mode (I).
4. The decode phase microoperations are,

$T_2 : D_0, \ldots\ldots, D_7 \leftarrow$ Decode IR(12-14)

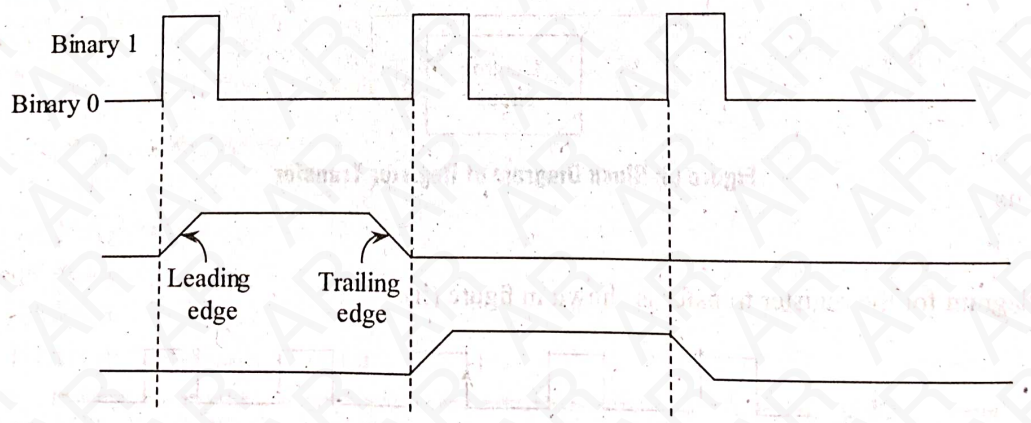
AR $\leftarrow$ IR (0-11)

I $\leftarrow$ IR(15)

Q10. Write about timing signal.

Answer : **Model Paper-II, Q1(b)**

A sequence of events and dependencies among events can be explained using timing signals. These timing signals are set of lines which are capable of establishing communication among devices. Through these lines binary 0 and binary 1 (signal levels) can be transmitted which indicates lower and higher levels respectively. By default binary 0 is represented on the line when no data is transmitted.



A signal transition from Binary 0 to 1 is a leading edge

A signal transition from Binary 1 to 0 is a trailing edge.

PART-B

ESSAY QUESTIONS WITH SOLUTIONS

1.1 DIGITAL COMPUTERS

1.1.1 Introduction, Block Diagram of Digital Computer

Q11. What is a digital computer? Discuss briefly on various types of computers.

Answer :

Digital Computer

A digital computer is a fast electronic data processing device, which takes the input in the form of instructions (often referred as programs), process it by referring to its various memory elements and finally produces the result.

Types of Computers

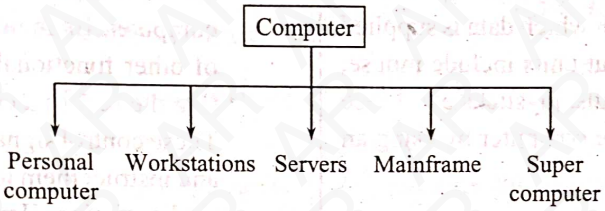


Figure: Different Types of Computers

1. **Personal Computer**

   Personal computers comes in the category of desktop computers. They are majorly used in small scale institutions, banks and homes. It supports various input devices such as keyboard, mouse and joystick using which it fetches data from memory units. It also supports various output devices such as monitors, speakers using which it provides data to the outside world.

2. **Workstations**

   Workstations is another form of computer specially designed for technical and scientific applications. It possess features like high degree of visual effects, exceptional designing capabilities and increased computational speed. Thus because of its versatile behaviour, it has the capability to satisfy all the engineering requirements.

3. **Servers**

   Server is another form of computer device which has been developed to process requests made by clients and deliver data to other clients as needed. It is generally referred to as databases as it maintains large volumes of storage locations.

4. **Mainframes**

   Mainframes are computer devices which are developed to handle large scale needs of business sector. They possess high speed as well as other computational capabilities to support numerous peripheral devices and workstations.

5. **Super Computers**

   Super computers are computer devices developed to handle large databases or very high level of complex calculations. It performs billions of computations in seconds. It usually occupies a room and is operated by many engineers. They are majorly used in advanced military, scientific applications, designing dynamics of machines, animated graphics, supersonic aircrafts.

Major parameters which can differentiate these computers are their size, speed, power consumption requirements.

Q12. Draw the fundamental block diagram of digital computer and explain the function of each unit.

Answer :

Functional units of a computer are fundamental parts that form a computer. Every computer is made up of five independent functional units. These are as follows,

1. Input unit
2. Output unit
3. Memory unit
4. Arithmetic and logic unit
5. Control unit.

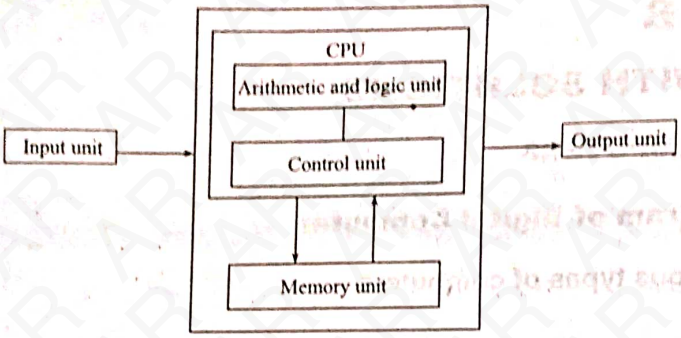


Figure: Block Diagram of a Computer

1. Input Unit

Input units are those units which accepts the data from the outside world. The most commonly used input unit is a keyboard. It is a device through which data is supplied to the computer. Few other input units include mouse, scanners, microphones, track balls, joysticks etc. These input units gets connected to the computer by using an interface.

2. Output Unit

These are the devices through which data is supplied to the outside world. The basic output device is a monitor. It displays text and images. Few other output devices include speaker, printer. The speakers are the devices through which sound is heard. It also acts as an amplifier. Printer is a device which is used for printing data in the form of text or images, it is also most widely used output device.

3. Memory Unit

It is a unit which is responsible for storing large volumes of computer data. Basically, whenever a program is under execution, the processor utilizes these memories to extract required data for execution of the given program. Every computer maintains two types of memories i.e., the primary and the secondary memory.

Primary memories are extremely fast memories. An example of primary memories includes RAMs. This memory is nothing but semiconductor elements capable of storing single bit of information. These elements together store certain length of data called words. A small scale memory stores about few thousands of words where as a medium scale memory can store about few millions of words. An extremely fast memory which is provided to support the speed of processor is referred to as cache memory. A computer consists of a new set of memory referred to as main memory. These type of memories are rather slow, but can store large volumes of data, i.e., data extending in Giga bytes. Secondary memories have less cost and at the same time provide large storage capacity. Few examples of secondary memories are magnetic tapes, CDROMs etc.

4. Arithmetic and Logic Unit

Arithmetic and logic unit is responsible for performing many arithmetic related operations. When user supplies data for either addition, subtraction, multiplication or division etc., the processor initiates the arithmetic and logic unit for further manipulations. For example, if a user wants to perform multiplication of two numbers, the processor initially fetches the required data, i.e., operands and initiates the arithmetic and logic unit. As a result, it considers the data and accordingly performs the required computation and returns the output.

5. Control Unit

The control unit forms the major component of the entire computer. Its main function is to govern the activities of other functional devices. This is done by transmitting the required control signals to various other units. These control signals inhibit the activities of these units and instruct them to perform their operations in the prescribed timing. Hence not granting full independence to other functional units which may lead to degradation of system's performance.

1.1.2 Definition of Computer Organization, Computer Design and Computer Architecture

Q13. Explain the terms computer architecture, computer organization and computer design in detail.

Answer :

Computer Architecture

It specifies the following,

1. Structure of the computer
2. Behavior of the computer.

In addition, computer architecture also specifies how the memory can be addressed with the help of,

(a) Instruction formats

(b) Instruction set.

There are two basic computer architectures. They are,

1. Von Neumann architecture
2. Harvard architecture.

For remaining answer refer Unit-I, Q14.

Computer Organization

It basically describes,

1. How to operate the components of a hardware?
2. How to join these components in order to constitute a computer system?

Computer Design

In computer design, the focus is mainly on two things,

1. The type of hardware to be used in the computer.
2. How to attach various parts of hardware?

Hence, with reference to the above two points, computer design is also known as "Computer Implementation".

Q14. Describe Von Neumann and Harvard architectures.

Answer :

Von Neumann Architecture

The basic features of Von Neumann architecture are,

- A single read/write memory stores data as well as instructions.
- The data in the memory can be accessed using its memory location, without considering its type.
- The instructions are executed in a sequential manner.

A computer with a Von Neumann architecture consists of four basic blocks. They are,

1. Memory unit
2. Input/output unit
3. Data processing unit
4. Program control unit.

1. Memory Unit

This unit stores data and instructions.

2. Input/Output Unit

This unit contains devices to provide input (for example keyboard), and to display output (for example: Monitor).

3. Data Processing Unit

It contains Arithmetic Logic Unit (ALU) to carryout arithmetic and logic micro operations specified by the instructions. In addition, it also contains Accumulator (AC) and Multiplier Quotient (MQ) registers. These registers temporarily store operands and results.

4. Program Control Unit

It fetches and decodes the instructions and provides control signals to all the units to carryout execution process of the instruction. For this purpose, it makes use of Instruction Buffer Register (IBR), Instruction Register (IR), Address Register (AR) and Program Counter (PC) register.

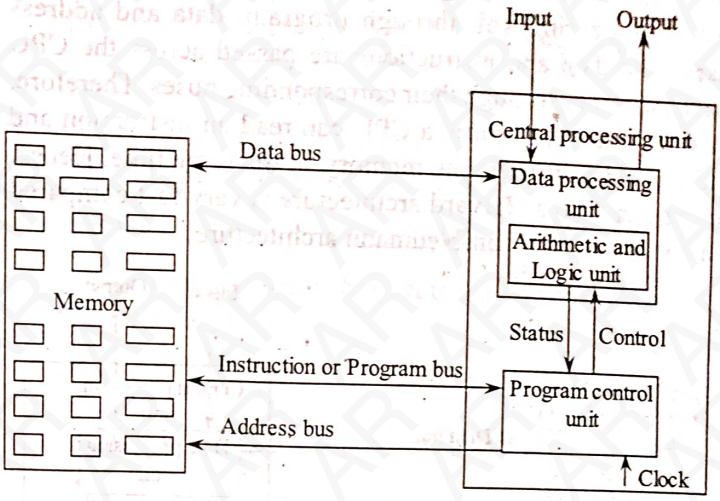


Figure: Von Neumann Architecture

The data and instructions are stored in a single shared memory. There are two bus lines, one for data and the other for address. They allow the processor and the memory to pass data, instructions and addresses between each other.

It has a reduced operation bandwidth as all the instructions and data are fetched in a sequential order. This limitation is referred to as a "Von Neumann Bottleneck".

Harvard Architecture

A computer with a Harvard architecture consists of five different blocks. They are,

1. Program memory unit
2. Data memory unit
3. Input/Output unit
4. Data processing unit
5. Program control unit.

All the instructions are stored in a separate memory unit called a program memory unit. The data is stored in a separate memory unit called data memory unit.

The input-output unit, data processing unit and program control unit have the same functionality as that in Von Neumann architecture.

In a Harvard architecture, instructions and data are stored in separate memories. Both of these memories may have varied word lengths, memory address structures, timings and implementation technologies. Generally, the size of the data memory is smaller compared to that of the program (or instruction) memory. The program memory is a read only memory because the instruction will only be read from the memory. Whereas, the data memory is a read-write memory, because the data may be read or written to the memory.

The data and program memory are connected to the central processing unit through program, data and address buses. The data and instructions are passed across the CPU and the memory through their corresponding buses. Therefore, in a Harvard architecture, a CPU can read an instruction and read/write data from/to the memory, at the same time. Hence, a computer with a Harvard architecture is very fast compared to a computer with Von-Neumann architecture.

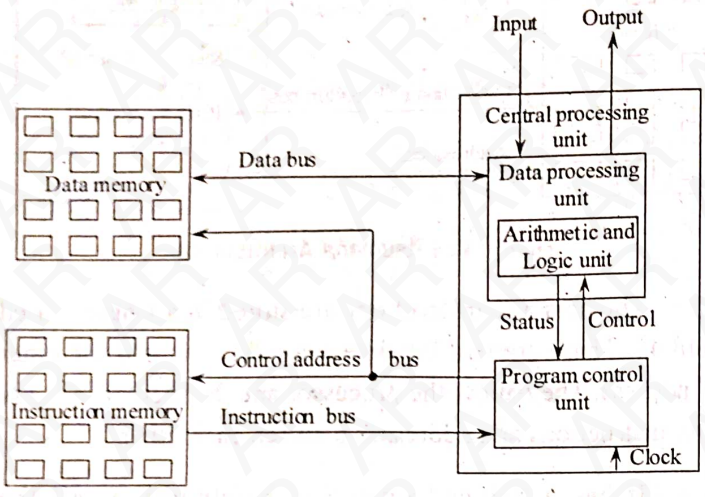


Figure: Harvard Architecture

Q15. Explain the components of the computer system.

Answer : Model Paper-I, Q2(a)

The generic organization of the computer contains three main components. They are,

1. CPU
2. Memory subsystem
3. I/O subsystem.

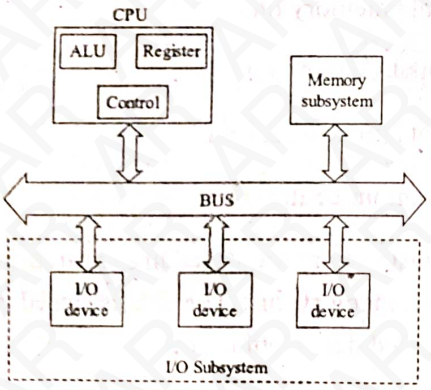


Figure: Generic Computer Organization

1. CPU

Central Processing Unit (CPU) is the brain of the computer that is responsible for controlling it, by interpreting and executing most of the commands from the hardware and software. It fetches, decodes and executes the instructions in order to perform the basic arithmetical, logical and input output (I/O) operations of the system. The three internal sections that are generally present in the CPU are,

(a) Register section

(b) ALU section

(c) Control unit section.

(a) Register Section

It is a collection of a bus (or any other communication mechanism) and registers that are special types of memory units used by the computer in order to transfer the data with high speed. These registers are not considered as the part of main memory, but they do store the data or information temporarily and then pass it based on the directions given by the control unit.

(b) ALU Section

As the name suggest, arithmetic and logic unit is responsible for performing many arithmetic related operations. When user supplies data for either addition, subtraction, multiplication or division etc., the processor initiates the arithmetic and logic unit for further manipulations. For example if a user intends to perform multiplication between two numbers. The processor initially fetches the required data i.e., operands, operators etc., and initiates the arithmetic and logic unit. As a result, the unit considers this data and accordingly performs the required computation by switching these values. Input and output of a special and extremely fast circuitry are referred as registers and returns back the output.

(c) Control Unit Section

The control unit though looks quite simple, but forms the major component of the entire computer. Its main function is to govern the activities of other functional devices. This is done by transmitting the required control signals to various other units. These control signals inhibits the activities of these units and instructs them to perform their operations in the prescribed timing. Hence not granting full independence to other functional units which may lead to degradation of system's performance.

2. Memory Subsystem

Combining two or more memory chips for enhancing the capacity of a memory unit is called memory subsystem.

For example, the 8×2 chips can be combined to form an 8×4 memory. In such a system, the data pins of the first chip are connected to bits 3 and 2 of the data bus, whereas the data pins of second chip are connected to bits and 0 of the data bus. Hence, both chips will receive some number of address inputs and CE and OE signals. Furthermore, when CPU reads data, it places the address on the address bus and both chips read in bits A1, A2 and A0 and decode. Also, the CPU assumes this 8×2 chips to be an 8×4 memory chip.

3. I/O Subsystem

I/O devices are necessary for the process of communication between the computer and external environment. The data and instructions are stored on the computers by using some input devices. After the processing of data, the output generated will be stored on the output devices. These devices are useful for various applications.

Buses are used to connect the components of a computer system. They are used for the transmission of data from one component to the other.

1.2 REGISTER TRANSFER LANGUAGE AND MICROOPERATIONS

1.2.1 Register Transfer Language

Q16. What is a microoperation? Write about register transfer language.

Answer :

Microoperation

Registers are temporary storage devices. Operations are performed on the information stored in these register.

A microoperation is defined as an operation which is performed on the information stored in the registers. It is an elementary operation. Microoperations such as shift, count, clear and load replace or transfer binary information of a register.

A digital system can be described by mentioning its internal hardware organization such as,

(a) The registers available in a digital system and function of each register.

(b) A series of microoperations executed on the information available in the registers.

(c) A control that initiates a series of microoperations.

Register Transfer Language

Microoperations can be explained in terms of words but it is a lengthy procedure, therefore symbols are used to explain information transfer between registers and other digital components.

A microoperation transfer among registers which is described in a symbolic format instead of lengthy descriptive format is called a "Register Transfer Language".

The microoperations applied to the hardware logic circuits such as gates that perform information transfer among registers. This is known as "Register Transfer".

The term language implies programming language that specifies a particular computational process using symbols.

- A register transfer language is an appropriate tool for explaining digital computer's internal organization.
- It is a system that indicates a sequence of microoperations in a symbolic form to carry out transferring of information among registers in a digital system.

Q17. What are register transfer logic languages? Explain few RTL statements for branching with their actual functioning.

Answer :

Register Transfer Logic Language (RTL)

The term "register transfer" refers to a consequence in which, after the execution of a given microinstruction, the data is transferred to certain register(s). Here, language specifies a standard logic using which certain predefined symbols are arranged forming a micro-instruction. Hence, register transfer language refers to a standard logic following which few symbolic sequences (micro-instructions) are written in a given digital circuitry.

RTL Statements for Branching

Few RTL statements for branching are represented below,

1. **BRO X**

   This statement causes branch or jump to the position 'X' if an overflow occurs.

2. **BRN X**

   This statement causes branch to the position 'X' if the result is negative.

3. **BRZ X**

   This statement causes branch to the position 'X' if result = 0

4. **BRP X**

   This statement causes branch to the location 'X' if the result is positive.

5. **BRE R_1, R_2, X**

   This statement causes branch to the location 'X', if the content of register R1 is equal to the content of register R2.

1.2.2 Register Transfer, Bus and Memory Transfers

Q18. Give the block diagram for registers and explain. Also, discuss about memory transfer operations.

Answer : Model Paper-I, Q2(b)

Register Transfer

In any digital system hardware, the storage and data transfers are predominant. For this purpose, registers that perform temporary data storage with the ability to shift/rotate the data bits are employed. The following diagram shows how the registers are rendered on a piece of paper.

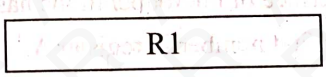


Figure (1): Abstract View of a Register

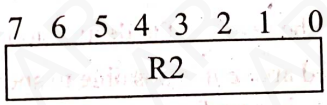


Figure (2): Showing the Bit Positions of an 8-Bit Register

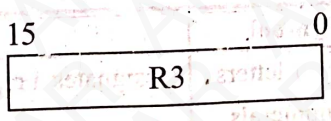


Figure (3): Showing the Range of Bits

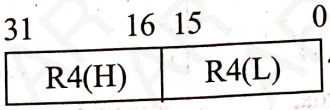


Figure (4): Divided into Two Halves

As shown in the above figures, capital letters followed by numbers are used to designate registers. For example in many systems, accumulator is designated by capital letter A. Bits of an n-bit register are numbered from 0 (right most) to $n-1$ (left most). Figure (1) shows a simple abstract view of a register. In figure (2) the individual bit positions of an 8-bit register can be identified. Figure (3) describes the range of values of 16-bit register. In figure (4) higher order bits (16-31) of register $R4$ is designated by H (for high byte) and the lower order bits (0-15) of register $R4$ is designated by L (for low byte).

Memory Transfer Operations

Memory read and memory write are the two memory transfer operations.

1. Memory Read

A read operation can be performed by transferring data from memory word M to its surroundings.

A address register AR provides address to the memory unit. The data in the address register AR is sent to another register known as data register DR.

Read Operation

$$DR \leftarrow M[AR]$$

This read operation reads the data from the memory address specified in register AR into data register DR.

2. Memory Write

A write operation is performed to store newly arrived information into the memory.

The data present in the data register DR is transferred to selected address of memory word M by performing write operation.

Write Operation

$$M[AR] \leftarrow R1$$

This write operation writes the data in register $R1$ to address selected by the address register AR into the memory word M.

Q19. Tabulate various symbols used for register transfers. With the help of an example, how register transfer is implemented.

Answer : Model Paper-III, Q2

Register Transfer Symbols

In general, the CPU of a computer system is setup with number of general purpose register. And each and every register in it has the property to store words which can be accessed independently. Also the other components of the system can have access to contents of any register.

Microoperations are specified as operations that are executed on data stored in register. This is carried out upon the information stored in multiple registers. The generated output is usually stored in destination register or in other registers.

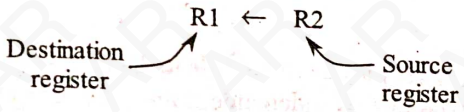
A sequence of microoperations are performed to execute one complete operation. For example, to subtract two numbers the following sequence of microoperations has to be performed,

1. Load first number in register A.
2. Load second number in register B.
3. Perform subtract microoperation.
4. Store the result in the destination register.

As described above it is possible to specify the sequence of microoperations in words, but it involves lengthy descriptive explanation. Hence, it is preferred to use symbolic notations to describe the microoperations. The following table lists the basic symbols used for register transfers.

Symbol	Description	Example
Letter or letters with numerals	Designates a register	PC, MAR, R1
Parentheses ( )	Designates a part of a register	R1(0 – 15) : Bits 0 – 15 of R1 R1(H) : Most significant half of register R1
Arrow ←	Designates transfer of information	R1 ← R2
Comma,	Separates two microoperations that are performed simultaneously	R1 ← R2, R2 ← R1

As shown in the above table, symbol '←' is used to describe the transfer of information from one register to another.



The above statement specifies the replacement of the contents of register *R*1 by the contents of register *R*2. This operation does not change the contents of the source register *R*2.

The term register transfer signifies the availability of hardware logic circuits which supports the transfer of source register output to the input of destination register. In addition to this, it also specifies the ability of destination register to support parallel load. A register transfer language also allows to specify control conditions in the statement. To specify control conditions, it uses control variable along with a colon at the left most side of the statement as shown below.

Here, *C* is used as a control variable. It is basically a boolean variable having a value 1 or 0. A boolean variable is also called a control function. This statement indicates that the operation is performed only when $C = 1$, otherwise transfer operation is not performed. This is just an implementation of if-then statement in the high-level language. This means that if $C = 1$ then $R_1 \leftarrow R_2$.

Register Transfer Implementation

The figure below shows the block diagram of register transfer from register *R*1 to register *R*2.

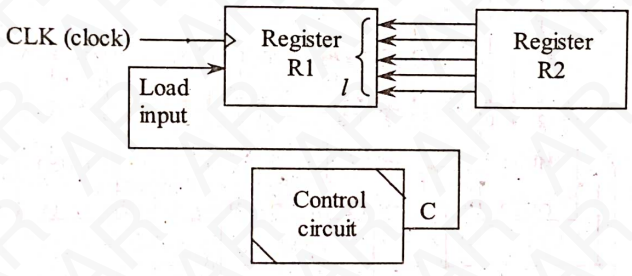


Figure (i): Block Diagram of Register Transfer

In the above figure, register *R*2 is connected to register *R*1 i.e., the outputs of *R*2 are given as inputs to *R*1. Let '*l*' be the length of the register i.e., the number of bits that a register holds. The control circuit has a control variable '*C*' with which the load input associated with the register *R*1 gets enabled. Further, the clock for the control variable must be synchronized with the same clock as applied to the register.

The timing diagram for the register transfer is shown in figure (ii).

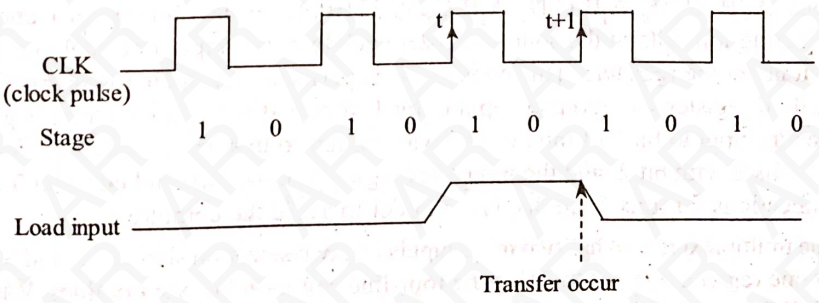


Figure (ii): Timing Diagram of Register Transfer

As shown in the figure, both register (*R*1) and control variable (C) are synchronized with the same clock pulse CLK. Hence, when the register transfer is implemented, at the same time, the control variable *C* is also enabled. In the timing diagram it is shown that at time '*t*', *P* is enabled which is represented by the rising edge of a clock pulse (i.e., 1). And at time $t + 1$, the load input is activated and the inputs of *R*1 are transferred into the register *R*2 simultaneously. Now, at time $t + 1$ the control variable *C* may go back to its original state 0. Otherwise, if *C* is active then the register transfer will occur with every clock pulse transition.

Q20. Describe the transfor of information using common bus.

Answer :

In a digital computer system, the registers play a critical role in data transfer. Every register must contain a data path which transfers information from one register to another. If independent lines are setup between every registers then, it can result in the requirement of excess number of wires, also this may create problems in controlling the wires there by leading to a complex circuit. Incase of multiple-register configuration, the transfer of information between the registers is carried out by a common bus.

A common bus can be specified as a set of common lines, dedicating each line to each bit of a register. These are now responsible for sequential transfer of binary information. Along the same lines, the identification of source register and destination register for a specific task is carried out by control signals.

The implementation of common bus is done in two ways,

1. Using Multiplexers
2. Using Tri-state bus buffers.

1. Using Multiplexers

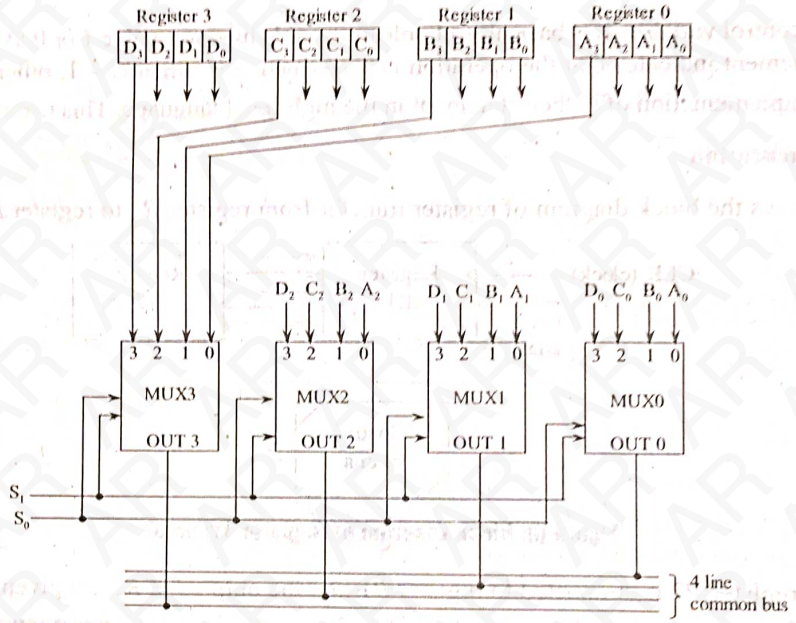


Figure (1): Common Bus System Using Multiplexers

Figure (1) illustrates the implementation of common bus system. It comprises four registers, Register 0, Register 1, Register 2, Register 3, which are connected to multiplexers (mux 0, mux 1, mux 2, mux 3). Each and every register contains four bits from 0 to 3 and each of them is routed to common bus through multiplexers. The implementation has four multiplexers that contains four input lines (A_0 B_0 C_0 D_0, $A_1B_1C_1D_1$, $A_2B_2C_2D_2$, 3210), two select lines (S_1S_0), and one output line. These four multiplexers operate by selecting four bits of the source register such that four input lines belonging to mux 0 are connected to bit 0 outputs belonging to four source registers. This connection is performed in such a way that bit 0 of register 0 is given as input to bit 0 of mux 0, bit 0, of register 1 is given as input to bit 1 of mux 0, bit 0 of register 2 is given as input to bit 2 of mux 0, bit 0 of register 3 is given as input to bit 3 of mux 0. In a way similar to this, the connections for mux 1 can be connected to bit 1 outputs and inputs for mux 2 with bit 2, and the inputs belonging to mux 3 are linked to bit 3 outputs of register 0 through 3. However, only input connections for mux 3 are displays inorder to avoid the complexity.

In addition to this, the multiplexers also has two more inputs i.e., two selection signals (S_1 and S_0). The purpose of these lines is to select four bits of any one register and put them on the four-line common bus via out lines. When both the select signals S_1 S_0 = 00 then the input 0 of all four multiplexers get selected and enforced on the outputs so as to transfer them on common bus. Now, the common bus receives the source register 0 contents. When S_1 S_0 = 01, the contents of register 1 gets transferred to the common bus.

S_1	S_0	Selected Register
0	0	Register 0
0	1	Register 1
1	0	Register 2
1	1	Register 3

Normally, the common bus system comprises n multiplexers, and n - line common bus and K number of inputs to each multiplexers.

Where, n = Number of bits in each register.

K = Total number of registers.

And each multiplexer size becomes $K \times 1$.

By connecting the inputs of all destination registers to bus lines and activating the load control input of specific destination register, the data transfer from a bus to any destination register can be performed. The data transfer between register R1 and R2 using a bus can be specified as BUS ← R1, R2 ← BUS.

It signifies that the content of register R1 is loaded on bus and the content of bus is loaded into the register R2. In most of the data transfers, direct method of representing data transfers between register is adopted, it is easy as bus exit with in the system.

Example

$R2 \leftarrow R1$

2. Using Tri-state Bus Buffers

The implementation of common bus system can be performed not only by multiplexers but the three state (tri-state buffers) also serves as a best tool for this. A tri-state gate can be specified as a digital circuit consisting of three output gates HIGH, LOW, and high impedence. Among them, the output states HIGH, LOW corresponds to logic 1 and logic 0 respectively, where as output state high-impedence acts as output circuit representing that output is disconnected.

As the bus serves a common connection between multiple registers and other units, the data transfer through it requires large sinking and large current source and the tri-state that gives high sinking and sourcing capacities is referred as tri-state buffer. Figure (2) shows the logic symbol for tri-state buffer.

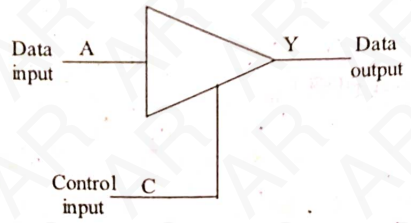


Figure (2): Logic Symbol for Tri-state Buffer

A tri-state buffer has two inputs, a normal data input and a control input. The control input decides the state of the output. When the control input = 1 then the output gets generated i.e., $y = A$. Whereas, when control input = 0 the output reaches high impedence state.

To construct common bus system using tri-state buffer let us assume that four registers are connected to the common bus and four output control signals for each register. For such system, 2 : 4 decoder is required as shown in figure (3). The decoder has 4 outputs which behaves as 4 control signals.

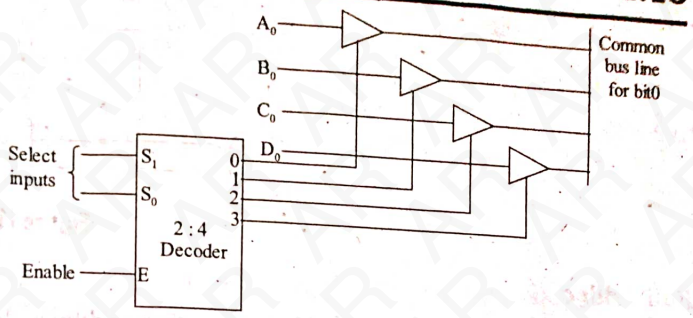


Figure (3): Common Bus Line with Three-state Buffer

When the enable input of the decoder indicates 1, then any one of the four tri-state buffer will be active. This activation of tri-state buffers depends on the binary values of the decoder's select input.

When the enable input of the decoder indicates 0, then all the four outputs of the decoder becomes 0's. This puts the bus into high-impedance state as all the tri-state buffers gets disabled.

1.2.3 Arithmetic Microoperations

Q21. With the help of block diagrams, explain about Half Adder, Full Adder and Parallel Adder.

Answer :

Half-Adder

Consider the following operations,

$0 + 0 = 00$

$0 + 1 = 01$

$1 + 0 = 01$

$1 + 1 = 10$

These operations are executed using a logic circuit called half adder. The half adder takes two binary digits as inputs and generates two binary digits as outputs (i.e sum and carry). The following table shows the truth table for the half adder,

Inputs		Outputs	
A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

From the operation of the half-adder as stated in the above table, expressions can be derived for the sum and the output carry as functions of the inputs. It can be observed that, the output carry (C_{out}) is 1 only when both A and B are 1. Therefore, can be expressed as the AND of the input variables.

$$C_{out} = AB$$

Now, observe that the sum output is 1 only if the input variables A and B are different. The sum can therefore be expressed as the exclusive-OR of the input variables,

$$\text{Sum} = A \oplus B = A\overline{B} + \overline{A}B.$$

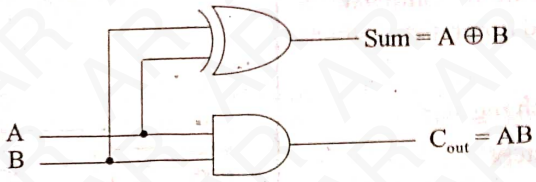


Figure (1): Half-Adder

Full-Adder

The full adder is the second category of the address. It operates by taking two inputs *A*, *B* and input carry and produces an output of sum and output carry. A logic symbol for a full-adder is shown in the figure (2).

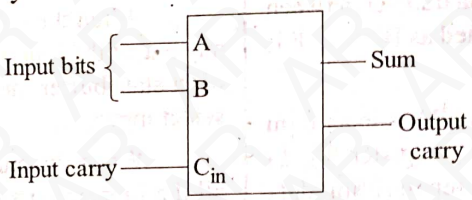


Figure (2): Logic Symbol for Full-Adder

Input			Output	
A	*B*	C_{in}	C_{out}	*S*
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Like a half-adder, the expressions for the sum and output carry of a full-adder are derived as follows,

$$\text{Sum} = (A \oplus B) \oplus C_{in}$$

$$C_{out} = AB + (A \oplus B)C_{in}$$

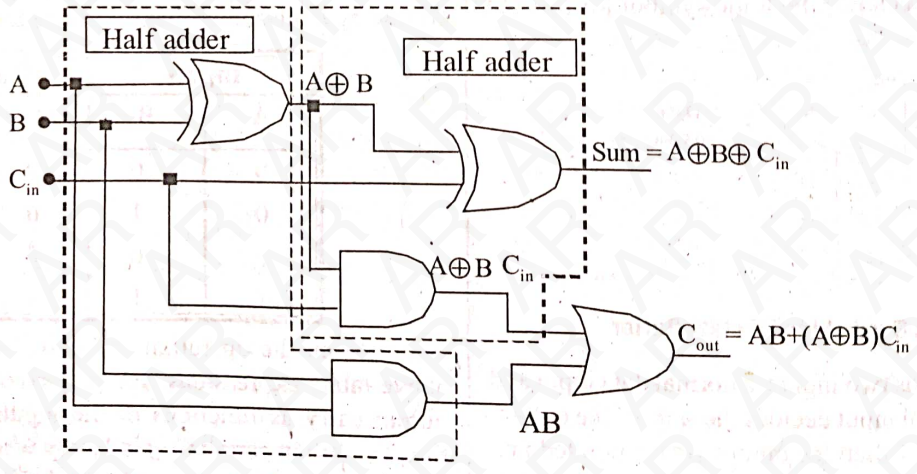
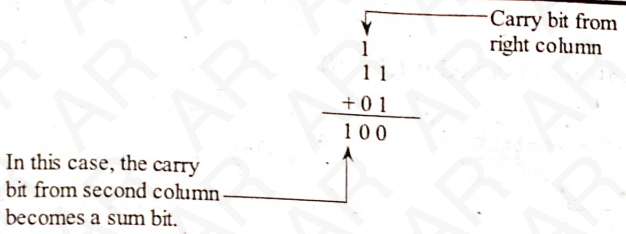


Figure (3): Full-Adder

Parallel Adder

A single full adder is an adder which is effective of adding two one-bit number and an input carry. The addition of binary number with multiple bits can only be performed with the use of full adders. This is called a parallel adder. When one binary number is added to another, each column generates a sum bit and a carry bit (0 or 1) to the next column to the left, as illustrated here with 2-bit numbers.



To add two binary numbers, a full-adder is required for each bit in the numbers. So for 4-bit numbers, four adders are used. The carrry output generated from each adder is connected as input in the form of carry input to the next higher-order adder.

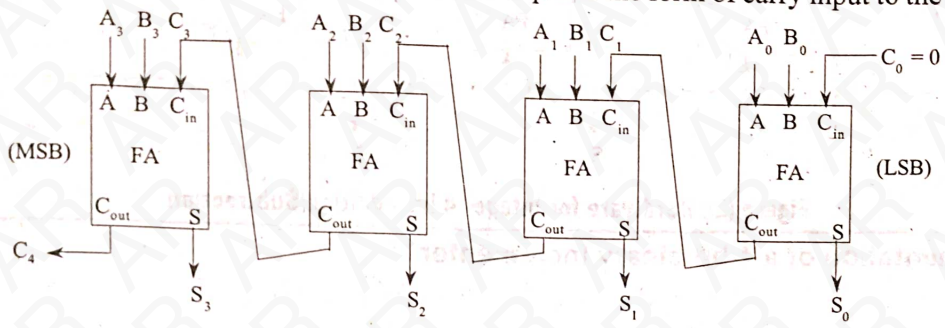


Figure (4): 4-bit Binary Adder

It can be observed that, there is no carry in the least significant bit position due to which the half adder is employed as least significant position or carry input of full adder is made 0.

Q22. With the help of block diagrams, explain the 4-bit binary subtractor.

Answer :

Subtraction is a special case of addition. To subtract two signed numbers, take the 2's complement of the subtrahend and add it with the minuend. Discard any final carry bit.

To get the 2's complement of a number, first find out its 1's complement and then add 1 to the Least Significant Bit (LSB).

The following example illustrates the subtraction of two numbers,

$8 - 3 = 8 + (-3) = 5$

```
          00001000   Minuend (+8)
        + 11111101   2's complement of subtrahend (–3)
Discard ----------
carry ← 100000101   Difference (+5)
```

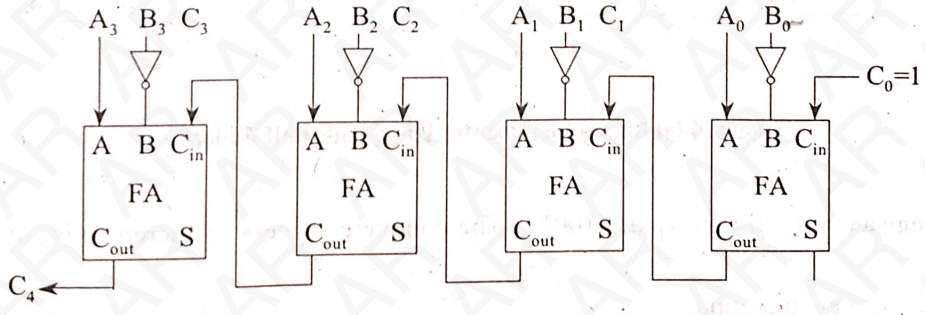


Figure (1): 4-bit Subtractor

Addition/Subtraction Logic Unit

By using a combination of exclusive-OR gates and full adders, both addition and subtraction operations can be performed as shown below. The circuit in figure (1) consists of two inputs A (a_0, a_1, a_2 and a_3) and B (b_0, b_1, b_2, b_3) as well as mode input M which is responsible for controlling the operations. If the value of $M = 0$, then the circuit performs an add operation. And if it is equal to 1 then a subtract operation is performed. Every XOR gate in the circuit has an input of B and an input M associated with it. If $M = 0$ is applied to the XOR gate then the result obtained is $B \oplus 0 = B$ and input carry C_0 is 0. The result is forwarded to all the full address and the circuit performs addition of A and B. Alternatively, if $M = 1$ is applied, then the result obtained is $B \oplus 1 = \overline{B}$ with the input carry $C_0 = 1$. Hence, all the inputs of B are complemented and the input carry is added to it. The circuit performs the subtraction operation by adding A to the 2's complement of B. The result obtained is interpreted as follows,

(i) **Unsigned Numbers**

When the condition $A \geq B$ then the result obtained is $(A - B)$. When $A < B$, then the 2's complement of $(B - A)$ is obtained.

(ii) Signed Numbers

If there is no overflow, then the result obtained is $(A - B)$.

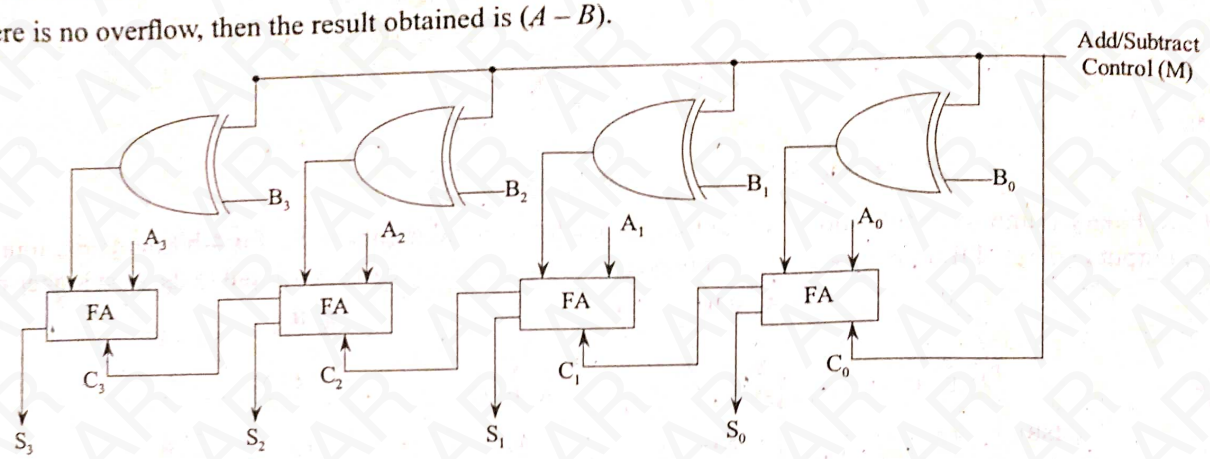


Figure (2): Hardware for Integer 4-bit Addition/Subtraction

Q23. Give the implementation of a 4-bit binary incrementer.

Answer :

The purpose of the binary incrementer unit is to carryout increment microoperation. The operation is done by adding one to the number stored in the register. While the count enable signal is active, the content of the register is incremented by the clock pulse transition. It continues till the count enable signal is deactivated. Implementing the increment microoperation using a binary counter is a register-specific technique. This dependency can be eliminated by using a combinational circuit consisting of cascaded half-adders.

A 4-bit binary incrementer implemented using a combinational circuit of half-adders is shown below. The circuit adds a 4-bit binary number or a nibble. B_0, B_1, B_2 and B_3 indicate the bits in the nibble from LSB (i6.e., B_0) to MSB (i.e., B_3) respectively. The first half-adder takes B_0 and logic-1 as inputs, adds them, stores the sum in S_0 and carry in C. The next half-adder takes the carry from the previous half-adder as one input and B_1 as another. It also adds the two inputs, stores the sum in S_1 and carry in C. The process continues till all the 4-bits are processed. C_4 gives the carry and S_3 is the sum which is generated after incrementing the 4-bit number.

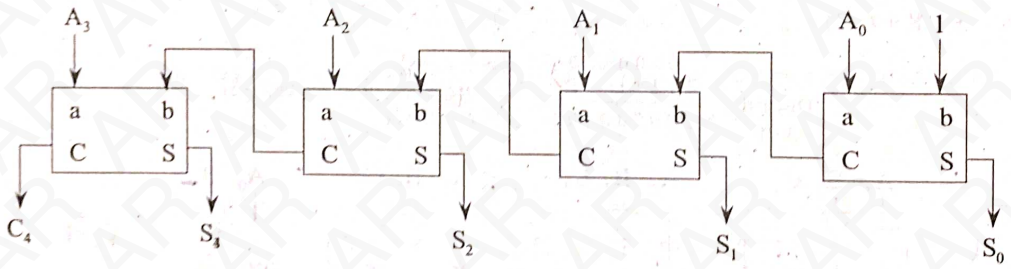


Figure: 4-bit Binary Incrementer Using Four Half-Adders

Example

Consider a 4-bit register has a binary value $(1001)_2$. The complete process of incrementing it using the 4-bit binary incrementer is as follows,

1. $B_0 = 1$ and logic = 1 is another input
   $\therefore$ $C_3 = 1$, $S_0 = 0$
2. $B_1 = 0$ and $C = 1$ (carry generated in previous adder)
   $\therefore$ $C = 0$, $S_1 = 0$
3. $B_2 = 0$ and $C = 0$
   $\therefore$ $C_3 = 0$, $S_2 = 0$
4. $B_3 = 1$ and $C = 0$
   $\therefore$ $C_4 = 0$, $S_3 = 1$

Therefore, after incrementing the binary value 1001 (decimal 9) it will become 1010 (decimal 10). The carry C_4 will be 1 only when the binary number 1111 is incremented and the sum bits S_0, S_1, S_2 and S_3 will contain 0's.

For an n-bit binary incrementer, n half-adders are required. In this case, the flow of carries and generation of sums is same as that of a 4-bit binary incrementer. Thus, a 4-bit binary incrementer can be extended to an n-bit binary incrementer.

Q24. Draw and explain 4-bit arithmetic circuit with the help of a function table.

Answer :

An arithmetic circuit can implement distinct arithmetic operations. It uses the parallel adder as the basic component. By controlling the data inputs of the adder, the arithmetic operations such as addition, subtraction, register transfer, increment and decrement can be generated using a single arithmetic circuit.

Figure shows a 4-bit arithmetic circuit.

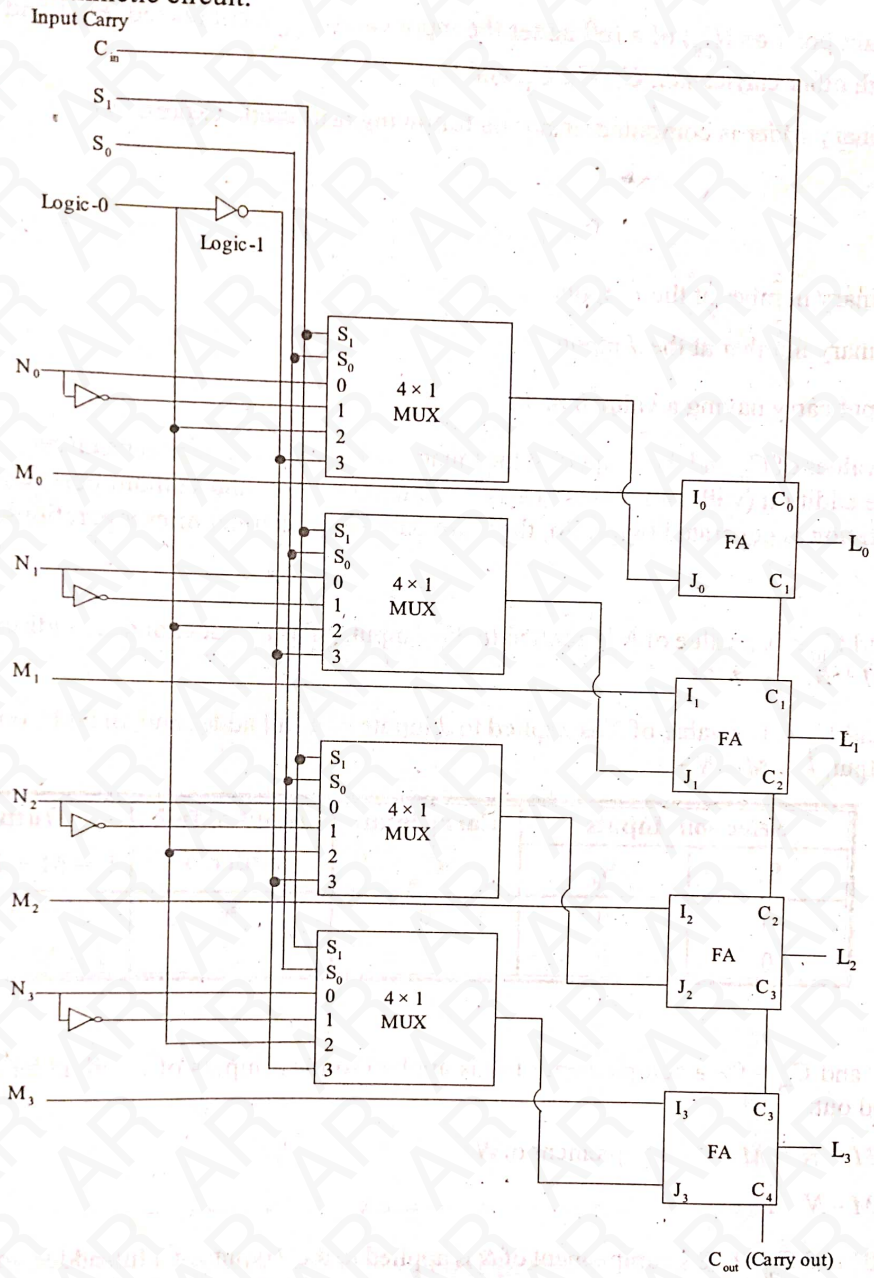


Figure: A 4-bit Arithmetic Circuit

The arithmetic circuit consists of,

- 4 full adders
- 4 multiplexers
- Two 4-bit inputs M and N
- Two selection inputs S_0 and S_1
- 4-bit output 0
- C_{in} (input carry) [C_{in} can be either 0 or 1]
- C_{out} (output carry)
- I and J are the inputs of the full adder.

The four input lines from M are connected to the I inputs of the full adder. From 4-bit input N, each of the 4 inputs go directly to the multiplexer's data inputs. The complements of inputs from N are also connected to the data input of the multiplexer. The remaining two data inputs of multiplexer are connected to logic-0 and logic-1.

Logic-0 is a fixed voltage value. An inverter is used to generate the logic-1 signal. The two selection inputs S_1 and S_0 control the four multiplexers.

To the least significant position (C_0) of a full adder the input carry C_{in} is connected. First and succeeding stages of a full adder are connected through other carries i.e., C_1, C_2, C_3, and C_4.

The output of the binary adder is computed using the following arithmetic expression:

$$L = M + J + C_{in}$$

Where,

M - 4-bit binary number at the I inputs

J - 4-bit binary number at the J inputs

C_{in} - The input carry having a value 0 or 1.

By controlling the values of C_{in} and the J inputs of the binary adder, the 4-bit arithmetic circuit can generate eight arithmetic microoperations. They are addition (with and without carry), subtraction (with and without borrow), increment, decrement and transfer. The transfer operation is generated twice. So, there are only seven distinct microoperations.

(i) Addition

- When $S_1S_0 = 00$ and $C_{in} = 0$, a value of N is applied to the J inputs of a full adder and an addition microoperation is carried out. Output, $L = M + N$.
- When $S_1S_0 = 00$ and $C_{in} = 1$, a value of N is applied to J inputs of a full adder and an addition with carry microoperation is carried out. Output, $L = M + N + 1$

Selection Inputs		Carry input	Input of full adder, J	Output
S_1	S_0	C_{in}		$L = M + J + C_{in}$
0	0	0	N	L = M + N
0	0	1	N	L = M + N + 1

(ii) Subtraction

When $S_1 S_0 = 01$ and $C_{in} = 0$, a complement of N is applied to the J inputs of a full adder and a subtract with borrow microoperation is carried out.

Output, $L = M + \overline{N} = M +$ 1's complement of N

$= M - N - 1$

When $S_1 S_0 = 01$ and $C_{in} = 1$, 2's complement of N is applied to the J inputs of a full adder, and a subtract microoperation is carried out. Output, $L = M + \overline{N} = 1 = M +$ 2's complement of N.

Section Input		Carry input	Input of adder, J	Output
S_1	S_0	C_{in}		$L = M + J + C_{in}$
0	1	0	$\overline{N}$	$L = M + \overline{N}$
0	1	1	$\overline{N}$	$L = M + \overline{N} + 1$

(iii) Transfer

When, $S_1S_0 = 10$ and $C_{in} = 0$, a zero is inserted into J inputs of a full adder by ignoring the input values of N.

Output, $L = M + J + C_{in} = M + 0 + 0$

$L = M$

When $S_1S_0 = 11$ and $C_{in} = 1$, insert all 1's into J inputs of a full adder by ignoring the input values of N

Output, $L = M + J + C_{in}$

$= M +$ 2's complement of $J + C_{in}$

$= M + \bar{J} + 1 + C_{in}$

$= M - J + C_{in}$

$= M - 1 + C_{in}$ $\{\because J = 1\}$

$= M - 1 + 1$

$\Rightarrow \quad L = M$

Section Input		Carry input	Input of	Output
S_1	S_0	C_{in}	adder,z J	$L = M + J + C_{in}$
1	0	0	0	L = M
1	1	1	1	L = M

(iv) **Decrement and Increment**

When, $S_1S_0 = 10$ and if $C_{in} = 1$, ignore the values of N and replace all the J inputs of a full adder with 0's

$L = M + J + C_{in} = M +$ 0's $+ C_{in}$

$= M + 0 + 1 = M + 1$ (increment)

When, $S_1S_0 = 11$ and if $C_{in} = 0$, ignore the values of N and replace all the J inputs of a full adder with 1's

$L = M + J + C_{in}$

$= M +$ 2's complement of $J + C_{in}$

$= M + \bar{J} + 1 + C_{in} \Rightarrow M - J + C_{in}$

$= M - 1 + C_{in}$ $\{\because J = 1\}$

$= M - 1 + 0$

$= M - 1$

$L = M - 1$ (decrement)

Selection Input		Carry input	Input of	Output
S_1	S_0	C_{in}	adder, J	$L = M + J + C_{in}$
1	0	1	0	L = M + 1
1	1	0	1	L = M – 1

1.2.4 Logic Microoperations

Q25. What is a logic microoperation? Discuss in detail various types of logic microoperations. Also give the hardware implementation of logic microoperations.

Answer : Model Paper-II, Q3

Logic Microoperations

The binary operations for bit strings present in a register are specified by the logic microoperations. Binary contents of a register are considered as distinct binary variables.

Consider a statement,

$x : A \leftarrow A \oplus B$

Where, A, B - Two registers

$\oplus$ - Exclusive-OR (XOR) operator

This microoperation is applied on the strings of bits stored in the registers A and B if the control variable $x = 1$.

Consider a numeric example, assume that $A = 0101$ and $B = 0011$

Contents of register A	Contents of register B	Contents of register B after XOR operation When x = 1
0	0	0
1	0	1
0	1	1
1	1	0

Special Symbols

To differentiate boolean functions from logic microoperations having similar functionalities, the logic microoperations use special symbols as shown in the following table.

Boolean Function	Logic Microoperation
(i) OR	(i) $\vee$
$E_0 = (a + b)$	$E_0 \leftarrow (a \vee b)$
(ii) AND	(ii) $\wedge$
$E_0 = (a\ b)$	$E_0 \leftarrow (a \wedge b)$
(iii) Complement	(iiii) Complement
$E_0 = a'$	$E_0 \leftarrow \bar{a}$

Example

$X + Y$: $R1 \leftarrow R3 + R4$,

$R2 \leftarrow R5 \vee R6$

In the above example '+' symbol between R3 and R4 denotes an 'arithmetic addition' operation.

Whereas in boolean function $X + Y$, it denotes an 'OR' operation.

'$\vee$' symbol between R5 and R6 denotes an 'OR' operation.

$R3 + R4$ denotes an ADD operation between binary contents of two registers $R3$ and $R4$.

$R5 \vee R6$ denotes an OR operation between binary contents of two registers $R5$ and $R6$.

Types of Logic Microoperations

There are 16 logic microoperations. They are as follows,

1. Clear Microoperation

Boolean function : F = 0.

Truth Table

x	y	F
0	0	0
0	1	0
1	0	0
1	1	0

2. Set to all 1's : F ← all 1's

Boolean function : F = 1

Truth Table

x	y	F
0	0	1
0	1	1
1	0	1
1	1	1

3. Transfer x : F ← x

Boolean function : F = x

Truth Table

x	y	F
0	0	0
0	1	0
1	0	1
1	1	1

4. Transfer y : F ← y

Boolean function : F = y

Truth Table

x	y	F
0	0	0
0	1	1
1	0	0
1	1	1

5. Complement x : F ← $\overline{x}$

Boolean function : F = x'

Truth Table

x	y	x'	F = x'
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	0

6. Complement y : F ← $\overline{y}$

Boolean function : F = y'

Truth Table

x	y	y'	F = y'
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	0

7. **OR Microoperation : $F \leftarrow x \vee y$**

Boolean function : $F = x + y$

Truth Table

x	y	F
0	0	0
0	1	1
1	0	1
1	1	1

8. **AND Microoperation : $F \leftarrow x \wedge y$**

Boolean function : $F = x\,y$

Truth Table

x	y	F
0	0	0
0	1	0
1	0	0
1	1	1

9. **Exclusive-OR Microoperation (XOR) : $F \leftarrow x \oplus y$**

Boolean function : $F = x \oplus y$

Truth Table

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

10. **NOR Microoperation : $F \leftarrow \overline{x \vee y}$**

Boolean function : $F = (x + y)'$

Truth Table

x	y	(x + y)	F = (x + y)'
0	0	0	1
0	1	1	0
1	0	1	0
0	1	1	0

11. **NAND Microoperation : $F \leftarrow \overline{x \wedge y}$**

Boolean function : $F = (x\,y)'$

Truth Table

x	y	(x y)	F = (x y)'
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

12. **Exclusive-NOR Microoperation (X-NOR): $F \leftarrow \overline{x \oplus y}$**

Boolean function : $F = (x \oplus y)'$

Truth Table

x	y	(x ⊕ y)	F = (x ⊕ y)'
0	0	0	1
0	1	1	0
1	0	1	0
1	1	0	1

13. **Logic Microoperation : $F \leftarrow x \vee \overline{y}$**

Boolean function : $F = (x + y')$

Truth Table

x	y	y'	F = (x + y')
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

14. **Logic Microoperation : $F \leftarrow \overline{x} \vee y$**

Boolean function : $F = (x' + y)$

Truth Table

x	y	x'	F = (x' + y)
0	0	1	1
0	1	1	1
1	0	0	0
1	1	0	1

15. **Logic Microoperation : $F \leftarrow x \wedge \overline{y}$**

Boolean function : $F = x\,y'$

Truth Table

x	y	y'	F = x y'
0	0	1	0
0	1	0	0
1	0	1	1
1	1	0	0

16. **Logic Microoperation : $F \leftarrow \overline{x} \wedge y$**

Boolean function : $F = x'\,y$

Truth Table

x	y'	x'	F = x' y
0	0	1	0
0	1	1	1
1	0	0	0
1	1	0	0

Hardware Implementation of Logic Microoperations

This circuit consists of 16 gates and 16 × 1 multiplexer which generates all the 16 basic logic microoperations by performing logic at a respective gate as shown in figure.

At each gate corresponding microoperation is performed and its result is generated at E_i. The circuit consists of 4 selection variables namely S_0, S_1, S_2, S_3. These selection inputs, select data inputs and passes its value to the output.

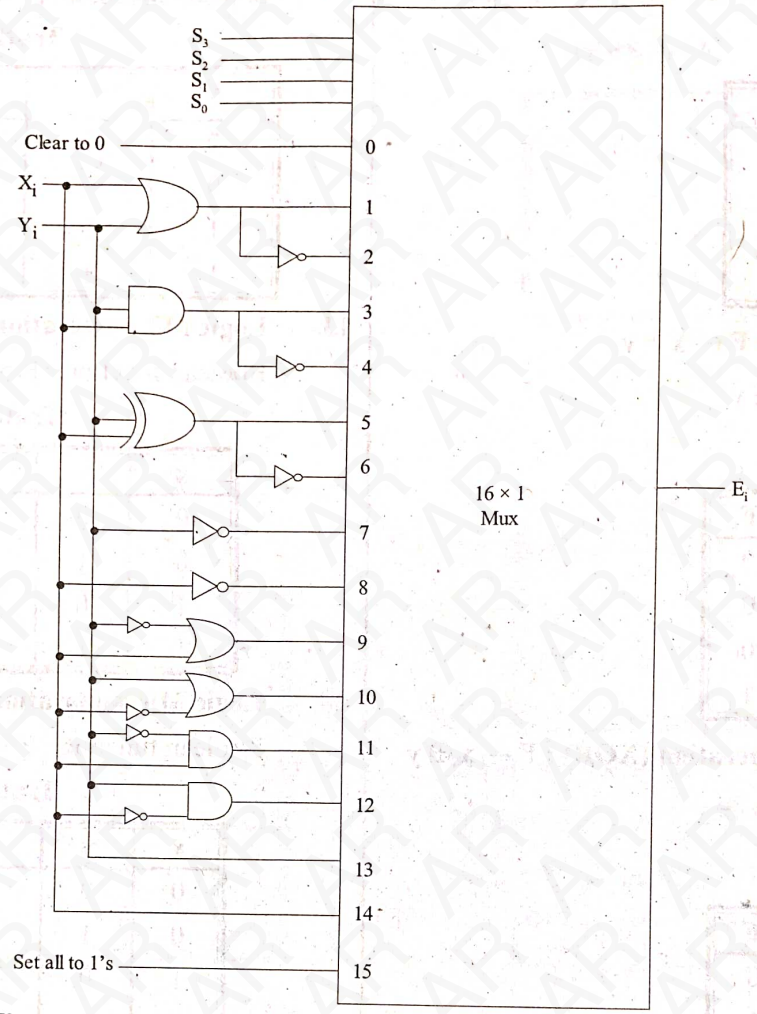


Figure: Logic Diagram of Hardware Associated with Microoperations

S_3	S_2	S_1	S_0	Output	Operation
0	0	0	0	$E = 0$	Clear
0	0	0	1	$E = X \vee Y$	OR
0	0	1	0	$E = \overline{X \vee Y}$	NOR
0	0	1	1	$E = X \wedge Y$	AND
0	1	0	0	$E = \overline{X \wedge Y}$	NAND
0	1	0	1	$E = X \oplus Y$	X-OR
0	1	1	0	$E = \overline{X \oplus Y}$	X-NOR
0	1	1	1	$E = \overline{Y}$	Complement Y
1	0	0	0	$E = \overline{X}$	Complement X
1	0	0	1	$E = X \vee \overline{Y}$	
1	0	1	0	$E = \overline{X} \vee Y$	
1	0	1	1	$E = X \wedge \overline{Y}$	
1	1	0	0	$E = \overline{X} \wedge Y$	
1	1	0	1	$E = Y$	Transfer Y
1	1	1	0	$E = X$	Transfer X
1	1	1	1	$E = 1$	Set all to 1's

Function Table

Q26. What are the applications of logic microoperations?

Answer :

Applications of logic microoperations are,

- In a register, individual bits or a portion of a word can be manipulated.
- Bit values can be changed in a register.
- Group of bits can be deleted from register.
- New bit values can be inserted in a register.

Manipulation of bits present in register *RX* and *RY* can be carried out using the following operations,

(i) Selective-set operation

(ii) Selective-complement operation

(iii) Selective-clear operation

(iv) Mask operation

(v) Insert operation

(vi) Clear operation.

(i) Selective-set Operation

It considers only 1's in register *Y* and sets the corresponding bits in the register *X* to 1 wherever 1's are encountered in register *Y*.

Ignore performing operations on bits of register *X* whose corresponding bits in register *Y* are 0's.

Example

Register X (Before)	Register Y (Set to 1)	Register X (After)
0	0	0
0	1	1 (Bit changed)
1	0	1
1	1	1 (No change, since its value is 1)

Corresponding bit

(ii) Selective-complement Operation

It considers only 1's in register *Y* and complements the corresponding bits in the register *X* wherever 1's are encountered in register *Y*.

Example

Register X (Before)	Register Y (Complement)	Register X (After)
0	0	0
0	1	1 (Complement operation)
1	0	1
1	1	0 (Complement operation)

Corresponding bit

(iii) Selective-clear Operation

It considers only 1's in register *Y* and clears the corresponding bits in the register *X* to 0.

Example

Register X (Before)	Register Y (clear)	Register X (After)
0	0	0
0	1	0 (Cleared to 0)
1	0	1
1	1	0 Cleared to 0

Corresponding bit

(iv) Mask Operation

It considers only 0's in register Y and clears the corresponding bits in register X to 0.

Example

Register X (Before)	Register Y (mask)	Register X (After)
0	0	0 (Mask Operation)
0	1	0
1	0	0 (Mask Operation)
1	1	1

Corresponding bit

The selective-clear and mask operations are similar only the difference is that, here it considers 0's in register y.

(v) Insert Operation

In a group of bits a new value can be inserted by performing mask operation and then OR operation is performed on the result with the required value.

Register X (Before)	Register Y (clear)	Register X (After)
0	0	0 (Mask Operation)
0	1	0
1	0	0 (Mask Operation)
1	1	1

Corresponding bit

Let the bits 1100 be inserted in register X then these bits are ORed with the resulting masked register X.

Masked Register X	Value to be inserted	Register X (After OR)
0	1	1
0	1	1
0	0	0
1	0	1

(vi) Clear Operation

An exclusive-OR (XOR) microoperation works as a clear operation. Whenever two corresponding bits of registers X and Y are similar i.e., both the bits are either 0s or 1s then its value will be assigned as 0 in register X.

Register X (Before)	Register Y (clear)	Register X (After)
Corresponding bit		
0	0	0
0	1	1
1	0	1
1 Corresponding bit	1	0

1.2.5 Shift Microoperations

Q27. Discuss in detail various types of shift microoperations.

Answer : Model Paper-I, Q3

The shift micro-operations are specifically employed for shifting the contents of a register to left or right position, they are used for performing serial transfer of data. During the shifting operation of bits, the binary information from serial input is given to the first flip-flop in the register and the last flip-flop in the register produces the information to the serial output.

The operations that are performed here is shift left operation and shift right operation. In former one, the operation of serial input transfers bit to the right most position where as in latter one, the operation of serial input transfers bit to the left most position.

The shift operations are categorized into three types,

1. Logical shift
2. Arithmetic shift
3. Circular shift or rotate.

1. Logical Shift

The serial input in logical shift operation is taken as zero. Due to the shift operation an empty position is created in the register which is filled with zero that is transferred through serial input. This is illustrated in the following figures. There are two logical shift microoperations i.e., logical shift left (LshL) and logical shift right (LshR).

The following operations illustrates logical shift,

$$R_1 \leftarrow \text{LshL } R_1$$

$$R_2 \leftarrow \text{LshR } R_2$$

The above operations are two microoperations in which R_1 is logically shifted left to 1 place and R_2 is logically shifted right to 1 place respectively.

Figure: Logical Shift Right

Figure: Logical Shift Left

Example of Logical Shift-Right

0 is inserted — This bit is removed

Figure: Before Logical Shift-Right

Figure: After Logical Shift-Right

Example of Logical Shift-Left

This bit is removed — 0 is inserted

Figure: Before Logical Shift-Left

Figure: After Logical Shift-Left

2. Arithmetic Shift

An arithmetic shift is defined as a microoperation which is responsible for shifting a signed binary number towards a right or left position. Hence, the arithmetic shift can either be (i) An arithmetic shift-right (or) (ii) An arithmetic shift-left.

An arithmetic shift-right is equivalent to the division of the binary number by 2 whereas the arithmetic shift-left is equivalent to the multiplication of the binary number by 2. When the number is divided or multiplied by 2, there should be no change in the sign of that number. Thus, in case of arithmetic shift operations, the sign bit of the number remains the same. A register with the left most bit contains the sign bit and the other bits contain the number. If a zero appears in the sign bit then the number is positive, else it is negative. The negative numbers are represented in a 2's complement format.

(i) Arithmetic Shift-Right

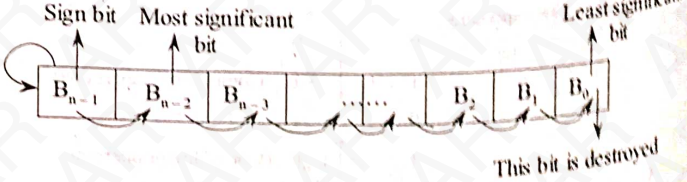


Figure: Arithmetic Shift-Right

The above figure shows a register that holds 'n' bits. The left most bit in the register stores B_{n-1} which is the sign bit. The bits B_{n-2} and B_0 specify the most significant and least significant bits of the number respectively. In the arithmetic shift-right operation, the sign bit must remain the same and the bits along with the sign bit must be shifted from left to right. Hence, the bit B_{n-1} does not change and is passed to B_{n-2}. The received bit in B_{n-2} is again passed to B_{n-3} and so on till B_0 receives a bit from B_1. As a result, the bit in B_0 is removed.

Example

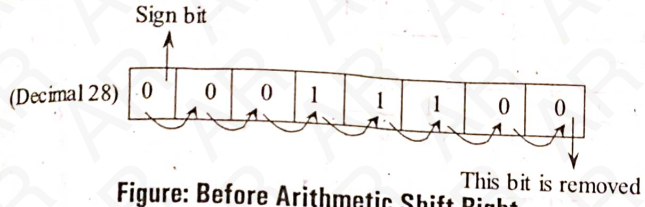


Figure: Before Arithmetic Shift-Right

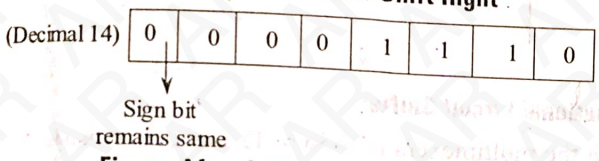


Figure: After Arithmetic Shift-Right

(ii) Arithmetic Shift-Left

The sign bit changes as per the received bit

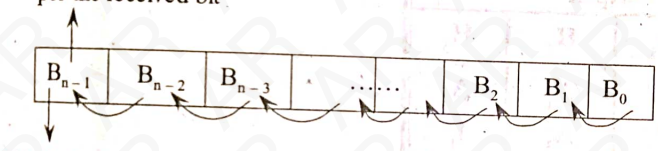


Figure: Arithmetic Shift-Left

During the arithmetic shift-left operation, a zero is placed in the B_0 position and the remaining bits in the register are shifted towards the left. Hence, the value at B_{n-1} bit gets lost and a new value which is received from B_{n-2} bit is stored in B_{n-1}. Depending upon the new value, the sign bit changes after the left shift operation. The sign reversal is performed when an overflow occurs after the multiplication of number by 2.

After an arithmetic shift left operation, an overflow occurs if the value of $B_{n-1} \neq B_{n-2}$ before the shift operation.

In order to identify an arithmetic shift-left overflow, a flip-flop Vs is required such that,

$$Vs = B_{n-1} + B_{n-2}$$

When Vs holds a zero value, no overflow occurs and when Vs holds 1, an overflow occurs and the sign bit is changed. Both the shifting operation and the Vs transfer in the overflow flip-flop must be synchronized with the same clock pulse i.e., they must be performed at the same time.

Example

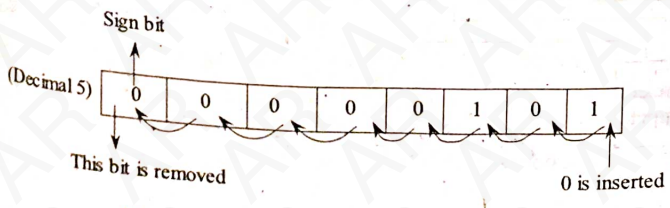


Figure: Before an Arithmetic Shift-Left

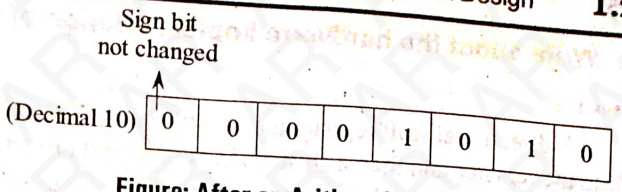


Figure: After an Arithmetic Shift-Left

Symbolic Representation

$R \leftarrow \text{AshL } R$

$R \leftarrow \text{AshR } R$

3. Circular Shift or Rotate

The circular shift or rotate is also called as rotate microoperation. During the shifting of bits in shift microoooperation, the bits which are shifted out of register go off track (lost). However, this bit is maintained by circular shift. It operates by shifting each bit out of one end of register and giving back into the other end of the register. The two types of rotate microoperations are rotate left (RL or CshL) and rotate right (RR or CshR).

Figure: Rotate Left

Figure: Rotate Right

Symbolic Representation

$R \leftarrow \text{CshL } R$

$R \leftarrow \text{CshR } R$

Example of Circular Shift-Left

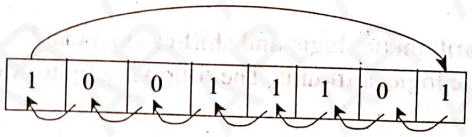


Figure: Before Circular Shift-Left

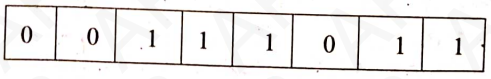


Figure: After Circular Shift-Left

Example of Circular Shift-Right

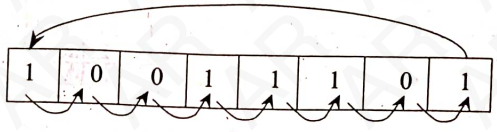


Figure: Before Circular Shift-Right

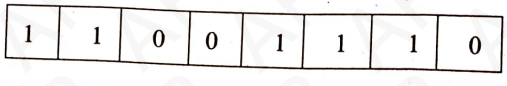


Figure: After Circular Shift-Right

Q28. Write about the hardware implementation of the shift microoperations.

Answer :

A bidirectional shift register along with a parallel load is used for shifting the information. The information can be parallely sent into the register and then a shift-right or shift-left operation can be performed. This system requires two clock pulses, one for storing the information in the register and the other for initiating the shift microoperations. In case of a processor unit containing several registers, a shift operation must be performed using a combinational circuit. Here, the information of a register is applied onto a common bus and the result generated is given to the combinational shifter to produce a shifted value. This value is again stored in the register. Hence, for shifting and storing the value in the register, only a single clock pulse transition is required.

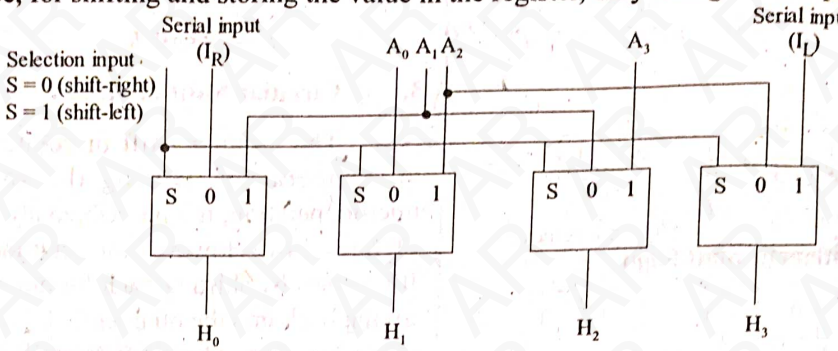


Figure: Representation of a 4-bit Combinational Circuit Shifter

In the above figure, a 4-bit combinational circuit shifter along with the multiplexers is shown. This circuit consists of four data inputs ($A_0 - A_3$) and four data outputs ($H_0 - H_3$). It also contains a selection input(s) and two serial inputs for performing a shift-right (I_R) and shift-left operation (I_L). If $S = 0$ then input data is shifted towards right and if $S = 1$, then it is shifted towards left.

Selection Input	Output			
S	H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

Table: Function Table

The above function table defines which input is connected to which output after the shift microoperation has been performed. Suppose, if a combinational shifter contains *m* data inputs and *m* data outputs then for shifting the data '*m*' multiplexers are required. To handle the two serial inputs another multiplexer is used to provide three possible kinds of shift microoperation.

1.2.6 Arithmetic Logic Shift Unit

Q29. With the help of a diagram explain one stage of arithmetic logic shift unit.

Answer :

Model Paper-II, Q2(a)

The arithmetic, logic and shift circuits can be combined into one composite circuit with the help of a multiplexer to get the arithmetic logic shift unit. The following figure shows one stage of an arithmetic logic shift unit.

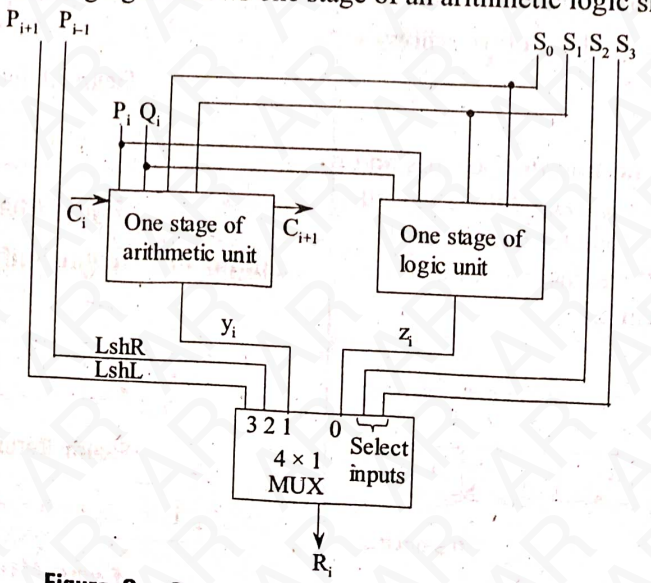


Figure: One Stage of Arithmetic Logic Shift Unit

Both the arithmetic and logic unit receive inputs P_i and Q_i, here the suffix i represents a typical stage i. Inputs S_1 and S_0 are used to select a particular microoperation, while S_2 and S_3 select the data in the multiplexer. The other two inputs to the multiplexer are P_{i-1} and P_{i+1} that perform shift-right and shift-left operations respectively. Actually the purpose of 4×1 multiplexer is to select between an arithmetic output in y_i and logical output in z_i. The diagram shown above is the typical representation of stage1. It must be repeated n times to obtain n-bit ALU. In case of multistage ALU, the output carry C_{i+1} of a given arithmetic stage is connected to input carry C_i of the next stage. The following table gives the list of operations that can be performed using arithmetic logic shift circuit.

Select inputs					Operation	Description
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$R = P$	Transfer P
0	0	0	0	1	$R = P+1$	Increment P
0	0	0	1	0	$R = P+Q$	Add P and Q
0	0	0	1	1	$R = P+Q+1$	Add P and Q with carry
0	0	1	0	0	$R = P+\overline{Q}$	Subtract with borrow
0	0	1	0	1	$R = P+\overline{Q}+1$	Subtract
0	0	1	1	0	$R = P-1$	Decrement P
0	0	1	1	1	$R = P$	Transfer P
0	1	0	0	X	$R = P \wedge Q$	AND
0	1	0	1	X	$R = P \vee Q$	OR
0	1	1	0	X	$R = P \oplus Q$	XOR
0	1	1	1	X	$R = \overline{P}$	Complement of P
1	0	X	X	X	R = LshR P	Logical shift right P
1	1	X	X	X	R = LshL P	Logic shift left P

As shown in the above table, there are 8 arithmetic operations which are selected when $S_3\ S_2 = 00$. When $S_3S_2 = 01$ logical operations are selected. Shift operations are selected when $S_3S_2 = 10$ and $S_3S_2 = 11$.

The input value X denote don't-care condition. The inputs marked with X's have no effect during the particular operation.

1.3 BASIC COMPUTER ORGANIZATION AND DESIGN

1.3.1 Instruction Codes

Q30. Define instruction code and operation code. With the help of examples, explain direct and indirect addressing.

Answer :

Instruction Code

Instruction code is a group of bits that directs the computer to perform operations. It is divided into two parts wherein each part has its own interpretation.

Operation Code

Operation code is a group of bits that define operations like addition, subtraction, multiplication, shift and complement. Number of bits required for operation code is dependent on the number of available operations. There must be at least n bits for 2^n operations. Consider an example of 64 operations with one operation being addition operation. The operation code contains six bits with bit configuration 110010 allocated to the ADD operation. When ADD is decoded in control unit, the computer issues control signals. These control signals read an operand from the memory and adds it to the processor register.

Computer program can be specified as a sequence of instructions. Each and every instruction defines the operation i.e., operation code (opcode) to be performed and the address of an operand. To increase the capability and computing power of an instruction it is necessary to include different types of addressing capabilities in an instruction format. Actually there are two most basic types of addressing modes.

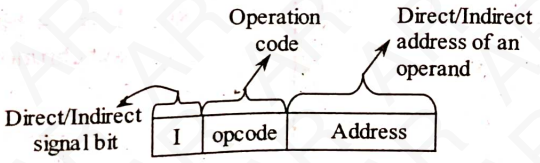
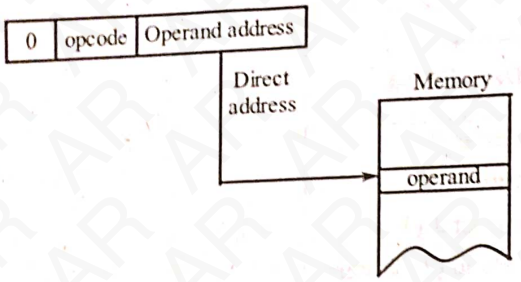


Figure: Instruction Format

1. Direct Addressing

Direct addressing uses direct addresses of operands. i.e., the second part of the instruction code in this scheme is a memory address that stores the data. Direct addressing is indicated to the processor by clearing the I-bit of the instruction code.



Example

Consider the following instruction which uses direct addressing.

O	ADD	3802

This instruction instructs the processor to add the data stored at memory address 3802 to the content of accumulator AC.

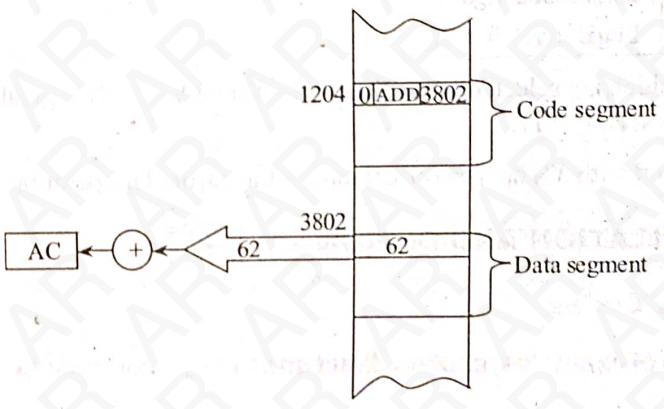
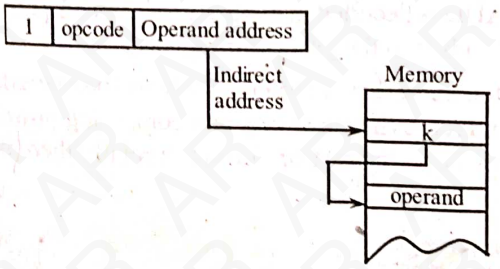


Figure: Direct Addressing

2. Indirect Addressing

Indirect addressing uses indirect address of operands i.e., in this scheme the second part of the instruction code specifies a memory location where the address of the operand is located. This is indicated to the processor by setting the I-bit of the instruction code.



Example

1	ADD	400

This instruction instructs the processor to get the address of the operand from the address 400 and add it to the content of the accumulator AC.

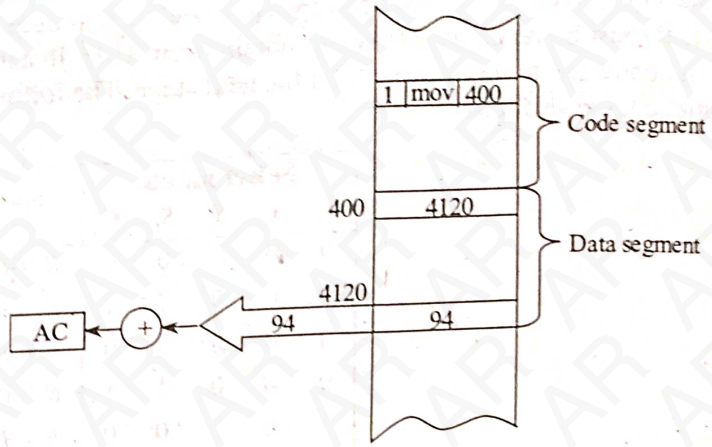


Figure: Indirect Addressing

1.3.2 Computer Registers, Computer Instructions

Q31. List various registers in a computer along with their purpose.

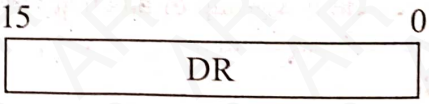
Answer :

Registers are temporary storage devices. These are internally made up of an array of flip-flops connected to each other. Here, each flip-flop stores one bit of information. Hence, an n-bit register contains 'n' flip-flops.

1. Data Register (DR)

It contains operand read from the memory

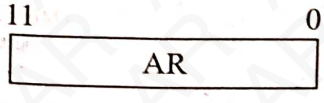
- It consists of 16 bits
- Its control inputs are LD, CLR and INR
- It is a 4-bit binary counter with parallel load and synchronous clear.



2. Address Register (AR)

It is used to specify memory address and it eliminates the need of another bus as it is connected to memory address.

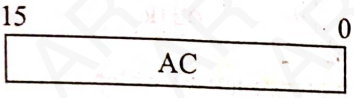
- It contains 12-bit memory address
- Its control inputs are load (LD), increment (INR) and clear (CLR)
- It is a 4-bit binary counter with parallel load and synchronous clear.



3. **Accumulator (AC)**

It holds temporary operands and results of ALU operations.

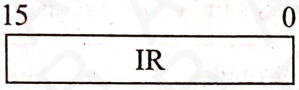
- It is a processing register
- It consists of 16 bits
- Its control inputs are LD, INR, and CLR
- It is a 4-bit binary counter with parallel load and synchronous clear.



4. **Instruction Register (IR)**

It contains 8-bit opcode instruction being executed.

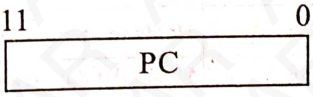
- It consists of 16 bits
- It holds the instruction read from the memory
- LD is the only control input
- It is a 4-bit register with parallel load.



5. **Program Counter (PC)**

It contains the address of the next instruction to be fetched from memory.

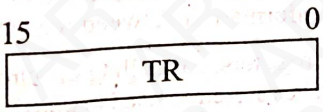
- It consists of 12 bits.
- It allows the computer to read the instructions from the memory in a sequential manner.
- When branch instruction is encountered it reads and executes each instruction word, sequentially.
- It is incremented by 1 for holding the next instruction.
- It's control inputs are LD, INR, CLR.
- It is a 4-bit binary counter with parallel load and synchronous clear.



6. **Temporary Register (TR)**

It contains temporary data.

- While processing it holds the temporary data
- It consists of 16 bits
- Its control inputs are LD, INR, CLR
- It is a 4-bit binary counter with parallel load and synchronous clear.



7. **Input Register (INPR)**

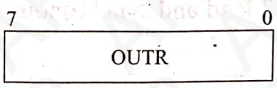
It holds an alphanumeric input information

- It consists of 8 bits
- It provides information for the bus.

8. Output Register (OUTR)

It accepts the coded information

- It contains output character
- It consists of 8 bits
- Its control input is load (LD)
- It receives information from the bus.



The digital computer performs the execution of instructions which are stored consecutively in the memory. These instructions undergo, fetching, decoding operations prior to the sequential execution. A computer uses the registers for storing the instruction code after it is read from memory. The computer employs program counter (PC) and instruction register so as to carry out fetching of instructions. MAR and MDR controls the data transfer between the main memory and the processor.

The MAR holds the address of the main memory to be transferred or from which data is to be transferred. The MDR holds the data which has to be written or read from the addressed word of the main memory. In addition to these registers processor also has general purpose registers like AC (accumulator) and TR (Temporary Register) which are used to hold the operands, the partial results and the memory addresses. For direct input and output operations, registers such as INPR (Input Register) and OUTR (Output Register) are present. The input character is stored in the input register where as the output character is stored and sent by the output register. The following table lists the various registers and their functions.

Register	Symbol Used	Function	Size
Data register or memory data register	DR or MDR	Holds the data read from memory or to be written in the memory.	16 bits
Address register or memory address register	AR or MAR	Holds the address for the memory.	12 bits
Accumulator register	A	Holds the operand for the ALU and can be used as a general -purpose register.	16 bits
Instruction register	IR	Holds the instruction code	16 bits
Program counter	PC	Holds the address of the next instruction to be executed.	12 bits
Temporary register	TR	Holds the temporary data.	16 bits
Input register	INPR	Holds the data read from the input device.	8 bits
Output register	OUTR	Holds the data to be sent to the output device.	8 bits

Table: Basic Computer Registers

Q32. Draw the interconnection structure of commonly used register connected to a common bus.

Answer :

A basic digital computer comprises of eight registers, a memory unit and a control unit. It is necessary to provide data paths between these components so as to transfer information between registers and between register and memory.

When different lines are used in between the registers, controlling the circuit becomes complicated due to the usage of excess wires. To avoid such complicated circuits, a common bus system is used to transfer information between different registers. A common bus can be specified as a set of common lines dedicating each line to each bit of a register. These are now responsible for sequential transfer of binary data. The identification of source register and destination register for a specific task is carried out by control signals.

In this system, the output obtained from seven registers and memory is connected to a common bus. The 16 lines of the common bus establishes a connection with the inputs of seven registers and memory unit. The binary value of the selection variables determines the output that must be selected for these bus lines. The decimal equivalent of the selected binary value is specified along each output line. For example, a number 2 is placed on the output of PC register. The output of PC is applied to the bus line when $S_2, S_1, S_0 = 010$ as it is the binary equivalent of 2. The bus provides its contents to the memory unit only if its write input is in active state. Also, the bus provides the 16-outputs to the memory when its read input is in active state and the values of the selection variables are 111.

As AR and PC are 12-bits registers, the four MSB of these registers are set to zero whenever their contents are provided to the common bus and the 12 LSBs are transferred to the register when AR and PC receivers the content from the bus.

The purpose of 8-bit INPR and OUTR register is to provide content as input to the bus and receive output content from the bus, respectively. An input device provides the input which is in the form of characters to the INPR register which then transfers it to the AC register. This register (AC) inturn transfers the character to OUTR register. After the character has been received it is again transferred to an output device by an OUTR register but not to any other register.

Out of the seven registers, six registers are responsible for providing 16-bit output to the common bus. Five registers consist of three control inputs (LD, INR, CLR) and two registers have LD as their control input. The register can receive information from the bus only if its LD input is enabled. The increment operation can be performed by activating the count input of the counter.

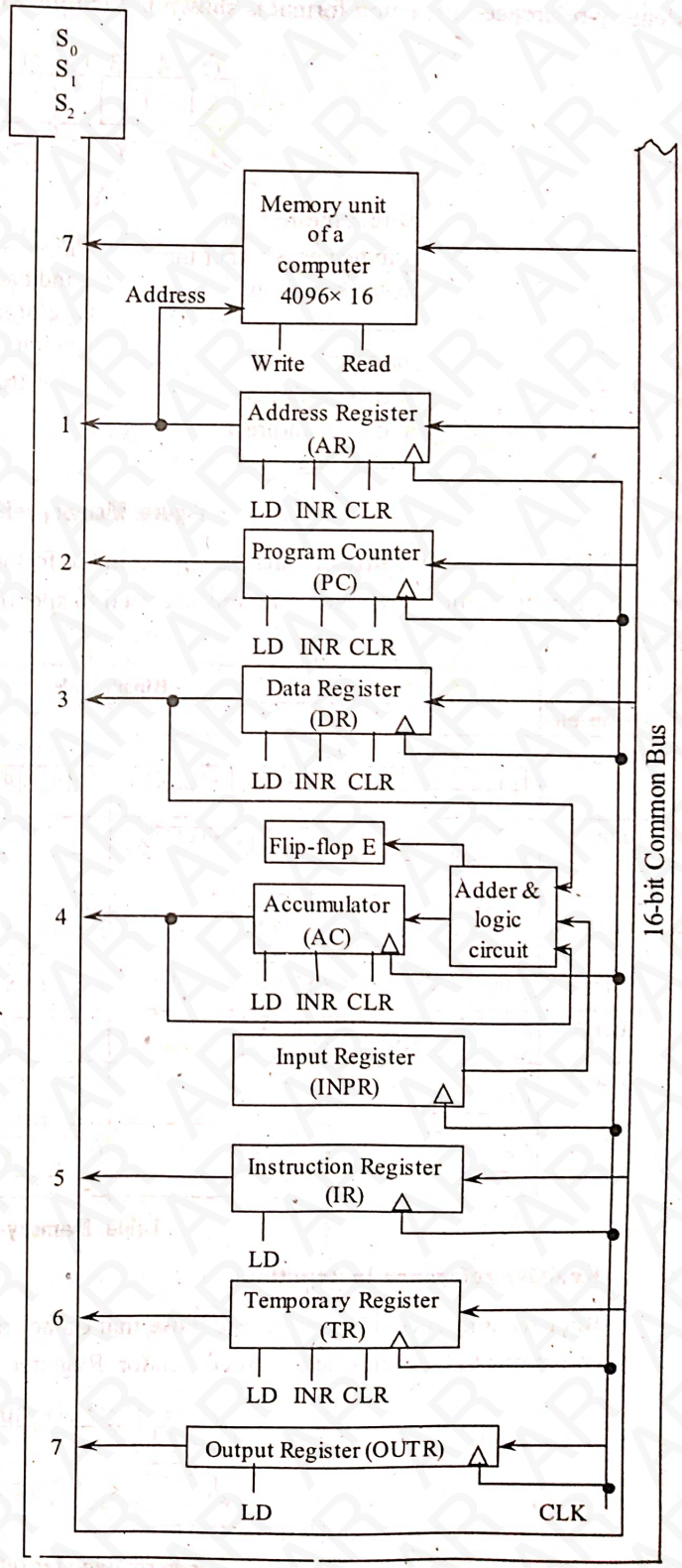
All the data inputs and outputs associated with the memory are connected to the common bus. Also there exists a connection between the memory address and AR, thereby making it mandatory for AR to define the memory address. There is no need for an address bus since only a single register is used for specifying the address. When a write operation is been performed, the register sends the data as inputs to the memory. On the other hand, during the read operation, the register apart from AC receives the data as output from the memory.

An adder and logic circuits are formed by using the three sets of 16 inputs. The outputs of AC send the first set of 16-bits inputs, which are responsible for executing the register microoperations like *shift AC* and *complement AC*. Next, the data register DR sends the other set of 16-bit inputs, which are responsible for implementing the logic and arithmetic microoperations like ADD DR to *AC*. After performing this operation, the result is transmitted to AC and the output 'carry' is passed to the flip-flop *E*. Finally, the input register INPR sends the remaining set of 8-bit inputs.

At the same clock cycle, the operations are executed in the adder and logic circuit, and the content present in any of the registers is placed on the bus. Hence, during the end of the clock cycle, the result obtained from the adder and logic circuit is transferred into AC and the content of the bus is transferred into the desired destination register; consider an example wherein two microoperations such as $DR \leftarrow AC$ and $AC \leftarrow DR$ are implemented during the same lock cycle. This implementation can be done using the following,

1. By applying the content of *AC* on to the bus provided $S_2 S_1 S_0 = 100$
2. By activating the load input (*LD*) associated with *DR*.
3. By transmitting content of *DR* to the register *AC*.
4. By activating load input *LD* associated with *AC*.

Hence, at the end of clock pulse, both the transfers (i.e., $DR \leftarrow AC$ and $AC \leftarrow DR$) occur.



Q33. List and explain different types of computer instructions. Also provide their formats.

Answer :

A basic computer has three types of instructions, each of which are 16-bits long. They are,

1. Memory-reference instructions
2. Register-reference instructions
3. Input/output instructions.

1. Memory-reference Instructions

Memory-reference instructions are those that require the access of memory for storing or loading the data for execution. Memory-reference instruction format is shown in the following figure.

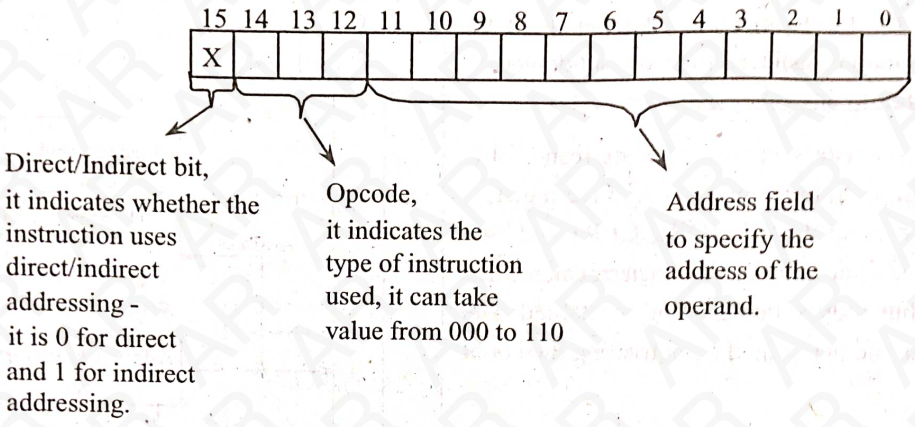


Figure: Memory-reference Instruction Format

Memory-reference instructions are shown in the following table. Along with the description and equivalent hexadecimal code. As shown in the above figure 12-bits are used to specify address of an operand and 1-bit is used to specify the addressing mode.

Mnemonic	Binary code I = 0	Binary code I = 1	Hexa Code I = 0	Hexa Code I = 1	Description
AND	I opcode Address 0 0 0 0 X X X X X X X X X X X X	I opcode Address 1 0 0 0 X X X X X X X X X X X X	0XXX	8XXX	And the contents of the specified memory location with with that the accumulator.
ADD	0 0 0 1 X X X X X X X X X X X X	1 0 0 1 X X X X X X X X X X X X	1XXX	9XXX	Add the contents of the accumulator with that of the specified memory location.
LDA	0010XXXXXXXXXXXX	1010XXXXXXXXXXXX	2XXX	AXXX	Load accumulator with that of specified memory location.
STA	0011XXXXXXXXXXXX	1011XXXXXXXXXXXX	3XXX	BXXX	Store the content of accumulator to the specified memory location.
BUN	0100XXXXXXXXXXXX	1100XXXXXXXXXXXX	4XXX	CXXX	Unconditional branch
BSA	0101XXXXXXXXXXXX	1101XXXXXXXXXXXX	5XXX	DXXX	Jump to the specified memory location and the return address.
ISZ	0110XXXXXXXXXXXX	1110XXXXXXXXXXXX	6XXX	EXXX	Increment the content of the specified memory location and skip if it is zero.

Table: Memory-reference Instructions

2. Register-reference Instructions

Register-reference instructions are those that do not require the access to memory for execution instead, they specify the test or operations to be performed on accumulator. Register-reference instruction format is shown in the following figure.

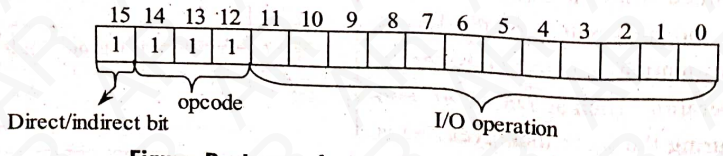


Figure: Register-reference Instruction Format

As shown in the above figure, register-reference instructions are identified by left most 4-bits i.e., 15th bit indicates that the I-bit should be zero and the next 3 bits i.e., 14-12 bits which form the opcode fields should be 111. Some of the register-reference instructions are shown in the following table.

Mnemonic	Description
HLT	Halt the computer (i.e., stop processing)
SZE	Skip next instruction if carry-bit (E) is zero
SZA	Skip next instruction if accumulator (AC) is zero
SNA	Skip next instruction if accumulator (AC) is negative
SPA	Skip next instruction if accumulator (AC) is positive
INC	Increment the accumulator (AC)
CIL	Circulate left accumulator (AC) and carry-bit (E)
CIR	Circulate right accumulator (AC) and the carry-bit (E)
CLA	Clear the contents of Accumulator (AC)
CLE	Clear the carry bit E i.e., make E = 0
CMA	Complement the content of accumulator (AC).
CME	Complement the content of carry-bit (E).

Table: Register-reference Instructions

3. Input/Output Instructions

Input/Output instructions are those that access the input/output ports through registers for execution purposes. The input/output instruction format is shown in the following figure.

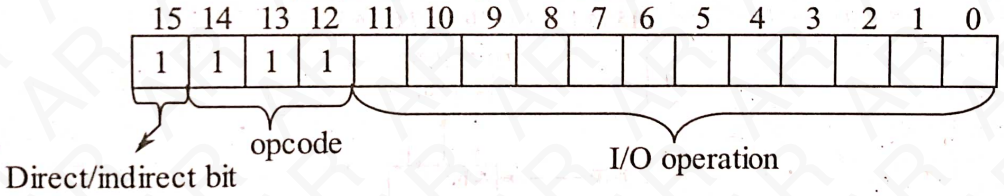


Figure: Input/Output Instruction Format

As shown in the above figure, input/output instructions are also identified by left most 4-bits that is 15th bit indicates that the I-bit should be 1 and the opcode field (14-12 bits) should be 111. Some of the input/output instructions are shown in the following figure.

Mnemonic	Description
INP	Load the accumulator (AC) with input character from input-output port
OUT	Send a character from accumulator (AC) to the INPUT/OUTPUT port
SKO	Skip on output flag
ION	Interrupt on
SKI	Skip on input flag
IOF	Interrupt off

Table: Input/Output Instructions

1.3.3 Timing and Control

Q34. Explain how the computer provides control and timing.

OR

Explain the organization of control unit of basic computer.

Answer :

Control Unit

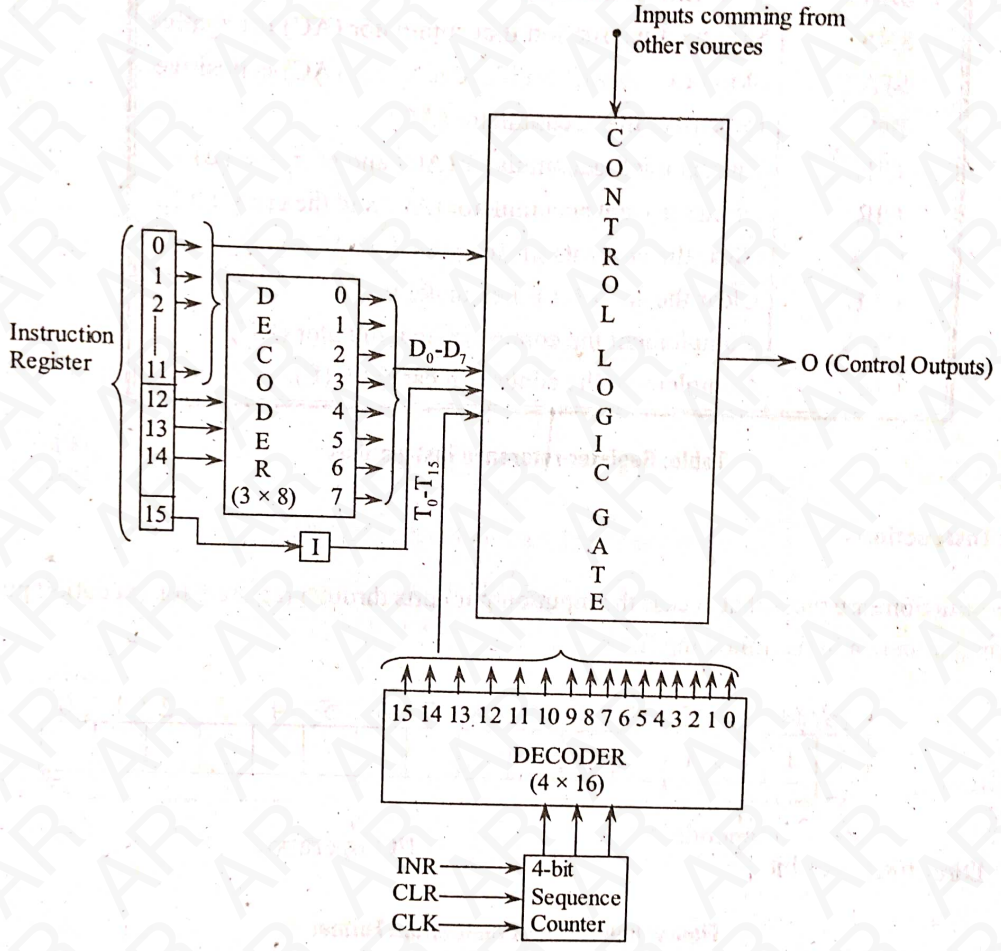


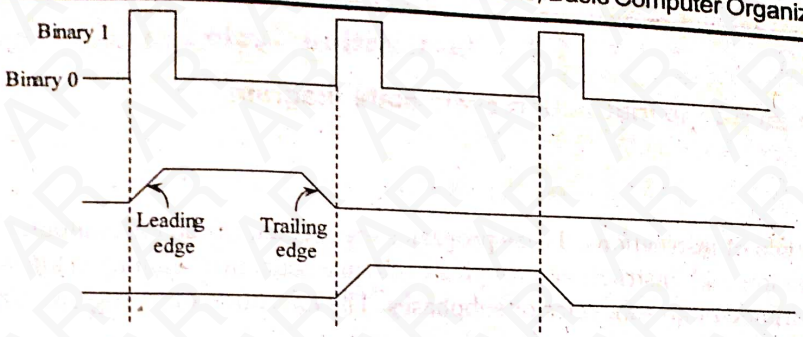
Figure: Control Unit

It can be observed from above figure that the internal circuitry of control unit consists of two decoders, an instruction register, a sequence counter and an array of control logic gates.

The instruction register is capable of storing 16-bits of data acquired from the memory. It is usually divided into three segments i.e., a single bit I, an opcode segment and a segment for storing initial twelve bits of data. The 'I' bit is fed to a flip-flop before being connected to the control logic gates. The 3 × 8 decoder, as the name suggests, takes in three inputs from the opcode segment of the instruction register and decodes it into '8' data signals and then they are fed to the control logic gates. The initial twelve bits are directly fed to the control logic gates. The output of 4-bit sequence counter are converted into 16-bit timing signals by means of a suitable 4 × 16 decoder and then are routed to the logic gate circuit. The operation of sequence counter is controlled by three inputs i.e., increment, clear and clock respectively.

Timing Signal

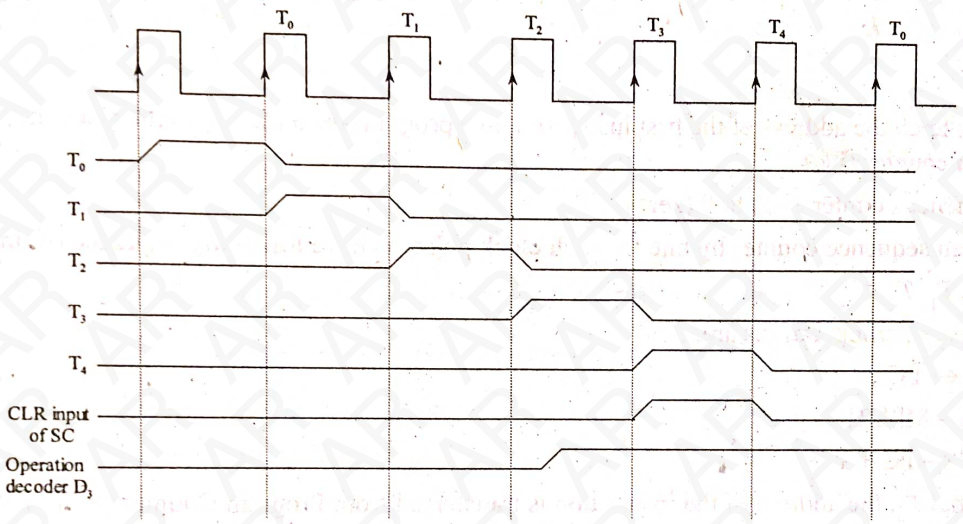
A sequence of events and dependencies among events can be explained using timing signals. These timing signals are set of lines which are capable of establishing communication among devices. Through these lines binary 0 and binary 1 (signal levels) can be transmitted which indicates lower and higher levels respectively. By default binary 0 is represented on the line when no data is transmitted.



A signal transition from Binary 0 to 1 is a leading edge

A signal transition from Binary 1 to 0 is a trailing edge.

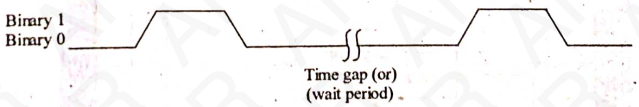
Consider a timing diagram for the basic computer,



From the above figure rising edge of timing signal indicates a memory read or write clock cycles.

In a basic computer, we imagine that the memory clock cycle (i.e., read and write operations) are carried out within the given time limit. We assume memory clock cycle is longer than the processor clock cycle i.e., before the next clock enters its positive transition. During this transition period memory word is loaded into the register.

The systems with long memory clock cycle when compared with a processor clock cycle rarely exist. Therefore time gaps are provided in the processor so that memory word is made available at that time.



- During first positive clock transition SC is cleared to 0 by activating its CLR input.
- At the same time, Timing signal T_0 gets activated.
- At timing signal T_0, the control signals that are connected to T_0 will trigger the corresponding registers.

  T_0 : AR ← PC

  The contents of PC are transferred to address register AR at timing signal T_0.
- As long as the CLR input of SC is active, for every positive transition of a clock SC gets incremented by 1 i.e., the timing signal continues with the sequence of T_0, T_1, T_2, T_3, T_4,...., T_{15}.
- In order to stop timing signal at T_4 and start with T_0, SC must be cleared to 0, i.e., $D_3\ T_4$: SC ← 0.
- Operation Decoder D_3 gets activated when the timing signal T_2 ends.
- The timing signal T_4 gets activated when the control function $D_3\ T_4$ is implemented on the AND gate.
- SC is cleared when the operation decoder D_3 and timing signal T_4 both are active i.e., $D_3\ T_4 = 1$.
- If SC is incremented instead of clearing, timing signal T_5 will get activated.
- Thus SC is cleared again, so that timing signal T_0 will be activated.

1.3.4 Instruction Cycle

Q35. Draw and explain about the instruction cycle state diagram.

Answer : Model Paper-III, Q3

Instruction Cycle

A program is a sequence of instructions. These programs are located inside the computer's memory unit. The process of executing a program by allowing each instruction through a cycle in a sequential manner is known as Instruction cycle. Further, each instruction cycle is partitioned into subcycles or subphases. The execution of a program is accomplished by allowing each instruction to pass through these subphase of a cycle.

Phases of an instruction cycle are,

1. Fetch
2. Decode
3. Execute.

1. Fetch

(i) Initially, fetch the address of the first instruction in a program from memory unit of a computer and place it into the program counter (PC).

(ii) Set sequence counter (SC) to 0 (zero).

(iii) Increment sequence counter by one for each clock pulse to have following sequence of timing signals.

T_0, T_1, T_2

The fetch phase microoperations are,

T_0 : AR ← PC

T_1 : IR ← M[AR]

PC ← PC + 1

At timing signal T_0, the address of the instruction is transferred from Program Counter (PC) to Address Register (AR).

At timing signal T_1, the Instruction Register (IR) is loaded with the instruction read from the memory and the Program Counter is incremented by 1 in order to fetch the address of the next instruction.

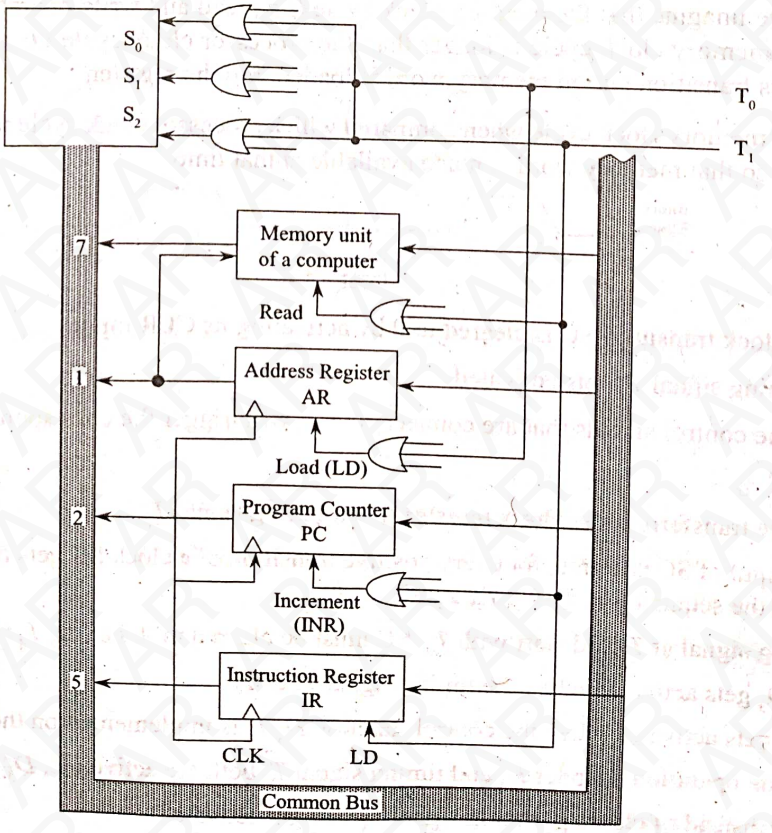


Figure: Fetch Phase Register Transfers Using a Common Bus System

A Common Bus System consists of selection inputs S_0, S_1 and S_2. At timing signal T_0, when $S_0\ S_1\ S_2 = 010$ the data of Program Counter (PC) is loaded into the bus. The control inputs are load (LD) and Increment (INR) read. When control input LD of address register (AR) is enabled then the data present in the bus is loaded into Address Register.

At timing signal T_1, read and increment operations are carried out. In the common bus system connections are made using OR gates. They are,

(a) The control input 'read' of memory unit is enabled.

(b) Make selection inputs $S_0S_1S_2$ equivalent to 111 in order to load memory contents into the bus.

(c) The control input 'load (LD)' of instruction register (IR) is enabled in order to load the data present in the bus into instruction register (IR).

(d) The control input 'increment (INR)' of program counter (PC) is enabled in order to increment the program counter (PC).

2. Decode

During this phase, the decoding of an instruction is carried out at timing signal 'T_2'.

(i) A 3 × 8 Decoder is used to decode the 12-14 bits of the instruction register (IR).

(ii) The 0-11 bits of instruction register (IR) are loaded into the address register (AR).

(iii) The 15th bit of instruction register (IR) is loaded into the addressing mode (I).

(iv) The decode phase microoperations are,

$T_2 : D_0, \ldots\ldots D_7 \leftarrow$ Decode IR(12-14)

AR $\leftarrow$ IR (0-11)

I $\leftarrow$ IR(15)

3. Execute

To carry out the execution phase, the type of the instruction must be determined.

(i) If the 7th bit of the output of the decoder ($D_7 = 1$) then it represents register-reference or input-output instruction depending on the binary value of an opcode (operation code).

(ii) If the output of the decoder ($D_7 = 0$) then it represents memory-reference instruction depending on the binary value of an opcode.

(iii) If $D_7 = 1$ and the addressing mode (I) of Register or Input-output instruction is equal to 1 i.e., $I = 1$ then an I/O instruction is executed when $I = 0$ register-reference instruction is executed.

(iv) If $D_7 = 0$ and the addressing mode (I) of memory-reference instruction is equal to 1 i.e., $I = 1$ then it represents indirection address condition. Therefore, the effective address must be read from memory at timing signal T_3. This is expressed by the register transfer statement as follows: AR $\leftarrow$ M[AR] when $I = 1$ no operation is executed as Address Register (AR) already contains effective address.

(v) The memory-reference instruction is executed after retrieving the effective address either in a direct or indirect mode.

The microoperations for the three instruction types are,

D_7IT_3 : AR $\leftarrow$ M[AR]

$D_7I'T_3$: No operation

D_7IT_3 : Execute on input-output instruction

$D_7I'T_3$: Execute a register - reference instruction.

The figure below shows the flowchart for the instruction cycle.

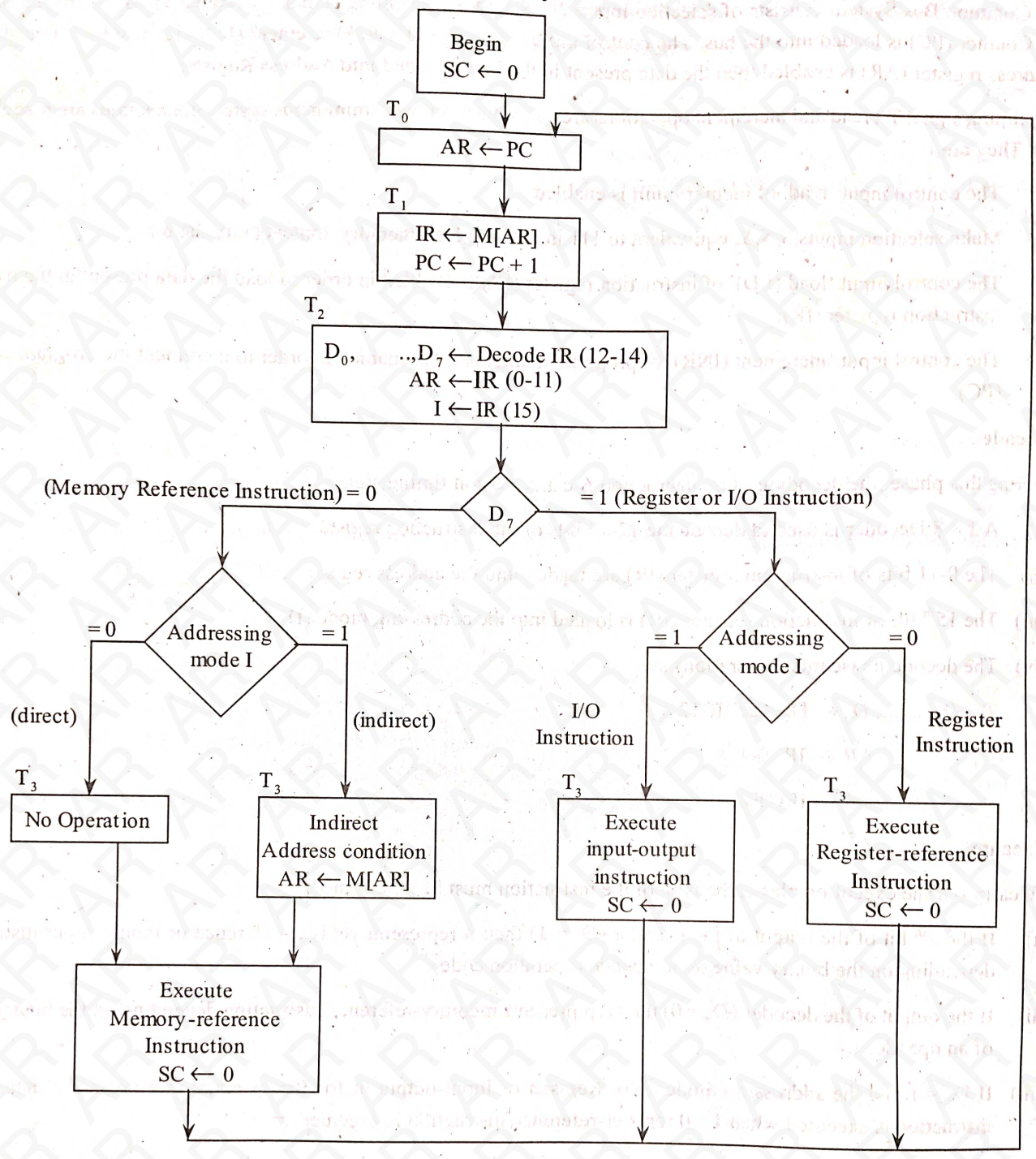


Figure: Instruction Cycle Flowchart

After the execution of an instruction, sequence counter is set to zero and the control goes to the fetch phase of an instruction cycle with $T_0 = 1$. This process is repeated for all instructions in the memory.

Q36. List and explain the register-reference instructions.

Answer :

When the decoder output $D_7 = 1$ and flip-flop $I = 0$, the instruction is identified as register-reference instruction by the control. There are 12 register-reference instructions, each indicated by a bit in the instruction code. IR holds these 12 (0 – 11) bits which are also moved to AR during T_2 time cycle. The execution of these instruction depends upon T_3's clock transition. Each register-reference instruction has a control function $D_7I'T_3$ or Simply 'f' ($f = D_7I'T_3$) and a microoperation associated with it. The table indicating the 12 register-reference instruction, their control functions and microoperations is given below.

Instruction	Operation	Control function	Microoperation
HLT	Half computer	fB_0	$S \leftarrow 0$ (Where S is a start-stop flip flop)
SZE	Skip if E zero	fB_1	if $(E = = 0)$ then $PC \leftarrow PC + 1$
SZA	Skip if AC zero	fB_2	if $(AC = = 0)$ then $PC \leftarrow PC + 1$
SNA	Skip if negative	fB_3	if $(AC\ (15) = = 1)$ then $PC \leftarrow PC + 1$
SPA	Skip if positive	fB_4	if $(AC\ (15) = = 0)$ then $PC \leftarrow PC + 1$
INC	Increment AC	fB_5	$AC * \rightarrow AC + 1$
CIL	Circulate left	fB_6	$AC \leftarrow$ Shl AC $AC(0) \leftarrow E$ $E \leftarrow AC\ (15)$
CIR	Circulate right	fB_7	$AC \leftarrow$ Shr AC $AC\ (15) \leftarrow E$ $E \leftarrow AC\ (0)$
CME	Complement E	fB_8	$E \leftarrow \overline{E}$
CMA	Complement AC	fB_9	$AC \leftarrow \overline{AC}$
CLE	Clear E	fB_{10}	$E \leftarrow 0$
CLA	Clear AC	fB_{11}	$AC \leftarrow 0$

Table: Register-reference Instructions

The 12 bits in IR are indicated by B_0 through B_{11}. Each register reference instruction has the basic instruction $SC \leftarrow 0$ (clear SC) at the end of execution. The control functions are denoted by fB_i $(i = 0, ...11)$. The 12 register reference instructions are,

1. **HLT:** Halts the computer by clearing the start-stop flip-flop 'S' and stopping the sequence counter.
2. **SZE:** Skips one instruction when $E = 0$. Instruction skipping is performed by incrementing the PC (program counter).
3. **SZA:** Skips one instruction when $AC = 0$
4. **SNA:** Skips one instruction when the 15th bit of AC is 1 (positive) i.e., AC (15) = 1.
5. **SPA:** Skips one instruction when $AC\ (15) = 0$ (negative)
6. **INC:** Increments the accumulator (AC).
7. **CIL:** Performs a circular left operation by shifting AC to left by making $AC\ (0) \leftarrow E$ and $E \leftarrow AC\ (15)$.
8. **CIR:** Performs a circular right operation by shifting AC to right by making $AC\ (15) \leftarrow E$ and $E \leftarrow AC(0)$.
9. **CME:** Complements the contents of E.
10. **CMA:** Complements the contents of AC.
11. **CLE:** Clear the contents of E.
12. **CLA:** Clear the contents of AC.

The control function of an instruction is the best representation of the instruction. For example, the hexadecimal code for CLA instruction is 7800 and its binary equivalent is 01111000 0000 0000. The control function of CLA is denoted as fB_{11} or D_7 I' T_3 B_{11} or I' D_7 T_3 B_{11}. Since the first bit in the binary number is 0 it means $I' = 0$ the next three bits (111) represents the opcode and is recognized from the output of the decode D_7. Signifies B_{11} in IR 1.

1.3.5 Memory Reference Instruction

Q37. With the help of examples, explain in detail various types of memory-reference instructions.

Answer :

Memory-reference instructions are those instructions whose execution requires the access of memory. The table shows seven memory-reference instructions along with the corresponding decoded output from the instruction decoder designated by D_i for $i = 0, 1, 2, ..., 6$. When D_7 is selected (i.e., $D_7 = 0$) then it specifies a memory instruction.

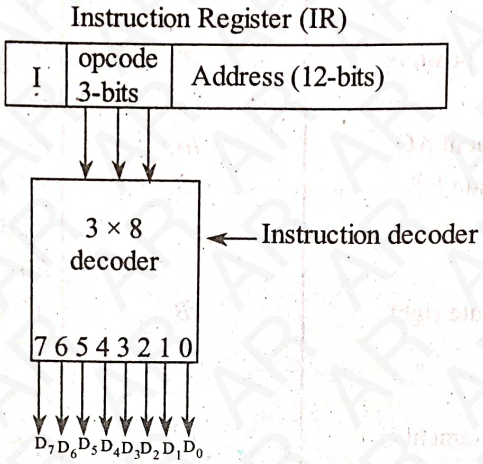


Figure: Decoding of Opcode Field

Depending upon the output generated by the instruction decoder, the action will be performed. For example, as shown in the table below, if the opcode is decoded as D_0 then logical ADN operation is performed. If the opcode is decoded as D_1 then addition operation will be performed. We know that at time T_0, Memory Address Register (MAR) is loaded with the contents of PC, at time T_1 two microoperations are performed, first is to load the instruction from memory pointed to by PC (now AR) and second is to increment the value of PC by 1. Effective address of the operand is in register AR at time T_2 when $I = 0$, or at time T_3 when $I = 1$. No operation is performed at time T_3 when $I = 0$ (i.e., when the operand uses direct addressing). Memory reference instruction execution starts at time T_4.

Mnemonic	Instruction decoder output	Description	Description in register transfer notation
AND	D_0	AND the content of specified memory location with accumulate	AC ← AC ∧ [MAR]
ADD	D_1	ADD the content of specified memory location to accumulator	AC ← AC + [MAR], E ← Cout
LDA	D_2	Load the data from the specified memory location into accumulator	A ← [MAR]
STA	D_3	Store the data to the specified memory location.	M[AR] ← A
BUN	D_4	Unconditional branch to the memory location specified in MAR	PC ← AR
BSA	D_5	Branch and save return address	[MAR] ← PC, PC ← AR + 1
ISZ	D_6	Increment and skip if zero	M[AR] ← M[AR] +1, if [MAR] + 1 = 0 then PC ← PC + 1

AND to AC

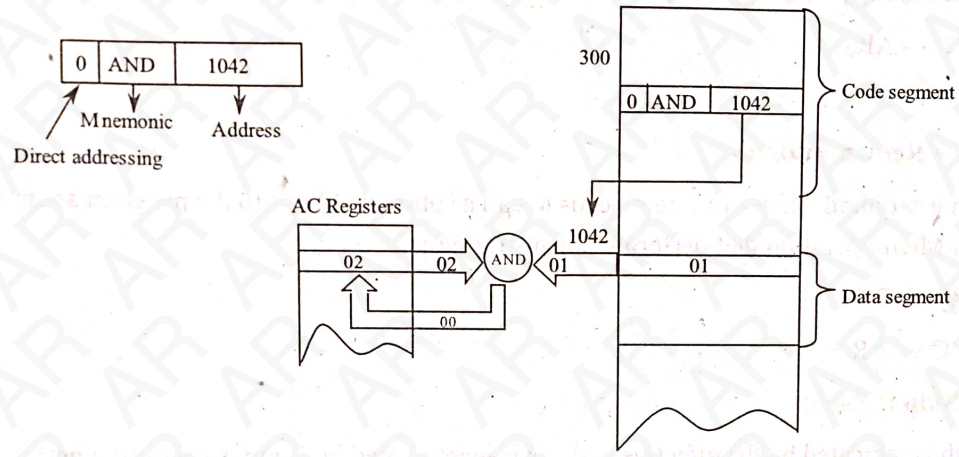
This instruction performs the AND operation on the pairs of bits of the specified memory location and implicit accumulator (we need not specify the second operand explicitly in the instruction). The results of this operation is stored in the accumulator. The following are the two microoperations that perform this operation.

$$\text{At } T_4: \quad DR \leftarrow M[AR]$$

$$\text{At } T_5: \quad AC \leftarrow AC \wedge DR$$

$$SC \leftarrow 0$$

The first microoperation fetches the operand specified in the instruction from memory to the Memory Data Register (MDR). And the second microoperation ANDs the contents of MDR and accumulator and stores the result in accumulator itself. Suppose the accumulator (A) contains 02 and you want to the AND the contents of memory location 1042 with that of accumulator.



After the execution of this instruction accumulator contains 03.

ADD to AC

This instruction adds the contents of specified memory location to the accumulator and stores the result in accumulator itself and if there is a carry out, it is stored in E-flip-flop. The following are the microoperations that perform this operation.

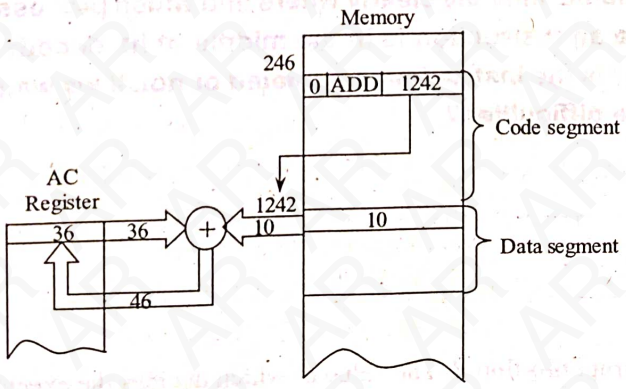
$$\text{At } T_4: \quad DR \leftarrow M[AR]$$

$$\text{At } T_5: \quad AC \leftarrow AC + DR$$

$$E \leftarrow C_{out}$$

$$SC \leftarrow O$$

Suppose the accumulator contains 36 and we want to add the contents of memory location 1242 to the accumulator.



LDA (Load to AC)

This instruction loads the accumulator with the contents of the specified memory location. The microoperations that perform this operation are,

$$\text{At } T_4: \quad MDR \leftarrow M[AR]$$

$$\text{At } T_5: \quad A \leftarrow DR$$

$$SC \leftarrow O$$

STA (Store AC)

The contents of the accumulator is transferred to the specified memory location. The microoperation that performs this operation is,

At T_4: $M[AR] \leftarrow AC$

$SC \leftarrow O$

BUN (Branch Unconditionally)

This instruction branches control unconditionally. That is it breaks the normal flow of execution and transfers the flow of control to the specified memory location.

The microoperation that performs this operation is,

At T_4: $PC \leftarrow AR$,

$SC \leftarrow O$

BSA (Branch and Save Return Address)

This instruction is required when a program needs to call a subroutine back to the program again when the subroutine has finished its operations. Microoperation that performs this operation is,

At T_4: $M[AR] \leftarrow PC$

$PC \leftarrow AR + 1$

ISZ (Increment and Skip if Zero)

The word which is indicated by the effective address is incremented by 1 and is checked whether it is equal to '0'. If yes, then the next instruction present in the program is skipped. To skip the next instruction, PC is incremented. The microoperations that perform this operation are :

At T_4: $DR \leftarrow M[AR]$

At T_5: $DR \leftarrow DR + 1$

At T_6: $M[AR] \leftarrow DR$,

If $(DR = 0)$ then $PC \leftarrow PC + 1$,

$SC \leftarrow 0$

Q38. Explain about instruction, fetch, and decode cycles for a memory reference instruction. Draw a flow chart also to explain the same. Indicate clearly where and which processor registers comes into picture. Now let us assume while an instruction is in the middle of its decode cycle an interrupt has arrived. What is going to happen? Is the instruction completed or not. If we want to stop there itself and handle the interrupt what are the difficulties?

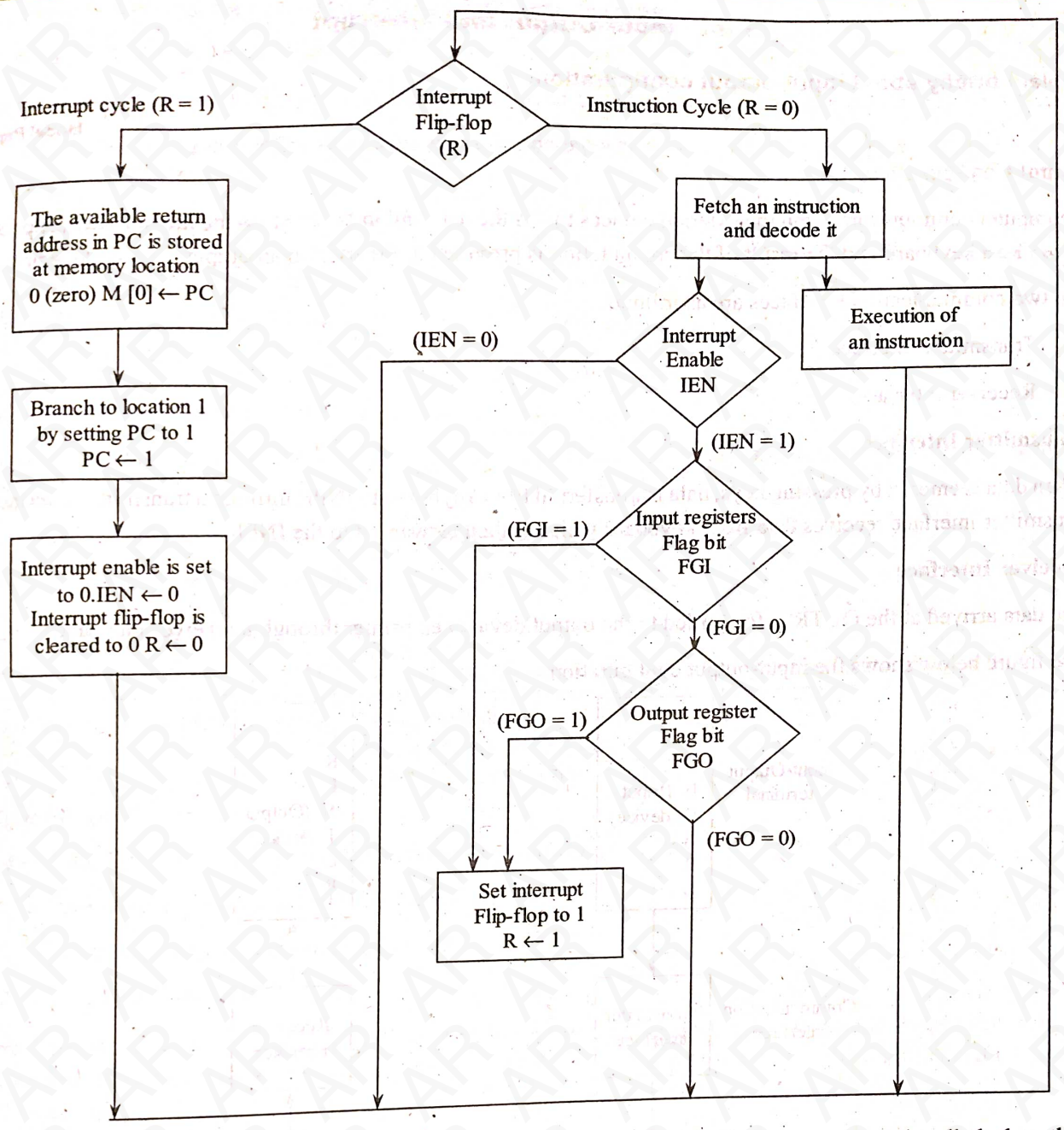
Answer :

Instruction Cycle

For answer refer Unit-I, Q35.

Handling Interrupt

In computer, there is an interrupt flip-flop R. The value of which decides the execution of interrupt cycle. If $R = 0$, then an instruction cycle is executed otherwise the interrupt cycle is executed. When an interrupt occurs in the decode cycle, the control is transferred to the execute phase where the IEN flip-flop is checked. If it is '0', then the interrupt is not handled, but the current instruction is stalled and the next instruction cycle is started. If it is 1, then FGI (i.e., input register flag) and FGO (i.e., output register flag) are checked. If any one of these flags is enabled (i.e., set to 1), then R is set to 1 and the interrupt cycle will be carried out which performs branch and save return address operation (BSA). If both the flags are set to '0', then the current instruction will be stopped and the next instruction cycle will be started, as the input and output registers cannot transfer data.



While executing an instruction, suppose an interrupt has occurred. If that interrupt is to be handled, then the current instruction execution must be stopped and the following steps need to be carried out.

1. Store the return address of the instruction where the control is to be transferred back after the interrupt is handled.

   Suppose, if the return address is to be stored at memory location '9', then the contents of PC are moved into memory location '9'.

   $M\{9\} \leftarrow PC$

2. Branch to a location where the interrupt can be handled.

   Suppose, location 10 holds the address where I/O program is stored that handles interrupts then the content of memory location '10' is moved into PC.

   $PC \leftarrow 10$

3. Clear IEN and *R* flip-flops, so that, no other interrupts occur, while current interrupt is being handled.

   $IEN \leftarrow 0$
   $R \leftarrow 0$

1.3.6 Input-Output and Interrupt

Q39. Explain briefly about input-output configuration.

Answer : Model Paper-II, Q2(b)

Input-Output Configuration

A computer communicates with the external devices that is the data and instructions to the memory are supplied using an input device like a keyboard and the result of the computation is provided to the user on an output device like a printer.

The two communication interfaces are as follows,

1. Transmitter interface
2. Receiver interface.

1. Transmitter Interface

When data is entered by pressing keys, data is transferred from keyboard to INPR through a transmitter interface. Initially, transmitter interface receives data from keyboard which is then forwarded to the INPR.

2. Receiver Interface

The data arrived at the OUTR is forwarded to the output device i.e., printer through a receiver interface.

The figure below shows the input-output configuration.

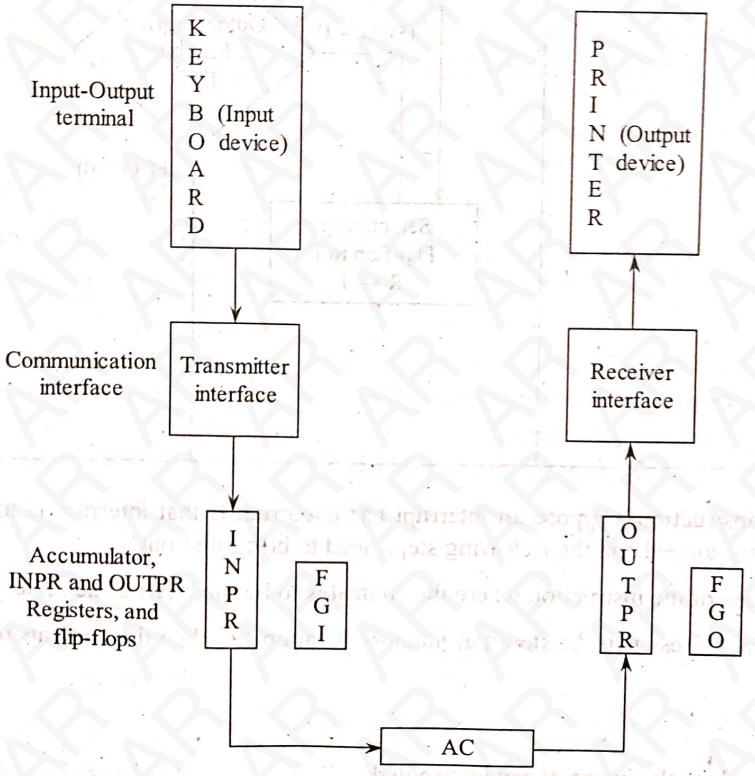


Figure: Input/Output Configuration

Both INPR and OUTPR registers consist of 8-bits. The control flip-flops of INPR and OUTR are FGI and FGO respectively. Each flip-flop can hold 1-bit.

FGI – Input flag

FGO – Output flag

If data is present in the input device then the FGI flag bit is set to 1

If the data is accepted by the computer then the FGI flag bit is cleared to 0.

The data is transferred from keyboard to INPR in the following manner.

1. Set the input flag FGI to zero (FGI = 0) in order to transfer data.
2. The input flag is set to 1 (FGI = 1) so that the data received from keyboard can be transmitted to input register INPR.
3. The data in the input register INPR remains same till the input flag FGI = 1. Even though a user presses any key data remains unchanged.
4. The data gets transferred from INPR to accumulator AC, when the computer encounters input flag bit value as 1.
5. Once the transfer of data is complete the flag bit is cleared to 0 (FGI = 0)
6. Finally, the INPR starts receiving data upon clearing the input flag bit.

The data is transferred from OUTR to printer in the following manner.

1. Set the output flag FGO to 1 (FGO = 1).
2. The data present in the accumulator AC is transferred to OUTR, when the computer encounters output flag bit value as 1 (FGO = 1).
3. Once the data transfer from AC to OUTR is complete clear the output flag bit (FGO = 0).
4. Next, printer accepts the data and prints it on a paper.
5. Again the output flag bit is set to 1 (FGO = 1).

The working of the INPR and OUTR is similar and the only difference is the direction of information flow is opposite.

Q40. List and explain the input-output instructions.

Answer :

Input-Output Instructions

Input and output instructions are used to perform the following:

1. To send the information to and from the AC register.
2. To check the status of the flag bits.
3. To handle the interrupts.

When the condition $D_7 = 1$ and $I = 1$ occurs, the control is used to identify the input and output instructions. These instruction contain an operation code 1111 and the remaining bits associated with the instruction define specific operation. The input and output instructions require the following control functions and microoperations.

1. $\mathbf{D_7 I T_3 = p}$

The clock pulse transition of timing signal T_3 helps in the implementation of input-output instruction. A boolean relation D_7IT_3 designated by the symbol p is used by all the control functions.

2. $\mathbf{I_R (i) = B_i}$

The bits i.e., (6-11 bits) present in IR signify the control function. Bit i of IR is designated by the symbol B_i. Hence, a symbol pB_i for $i = 6$ to 11 bits is used to specify the corresponding control function. When the condition $p = D_7IT_3 = 1$ occurs, the sequence counter SC is set to 0.

i.e., $SC \leftarrow 0$

3. **INP (Input Character)**

Using this instruction, the input information is passed from INPR to the 8 low-order bits of the AC register.

i.e., AC(0 – 7) ← INPR

The INP instruction is also responsible for setting the input flag to 0.

i.e., FGI ← 0.

4. **OUT (Output Character)**

Using this instruction, the 8 LSBs of AC are passed into the output register OUTR.

i.e., OUTR ← AC (0 – 7)

The OUT instruction is also responsible for setting the output flag to 0.

i.e., FGO ← 0

5. **SKI (Skip on Input Flag) and SKO (Skip on Output Flag)**

These two instructions are used to check the status of the flag bits. If the flag *F* holds 1, then the next instruction is skipped. The skipped instruction specifies a branch instruction that goes back to the flag so as to perform the check operation on it. If the flag holds 0 then no skip operation must be performed.

If (FGI = 1) then ($PC \leftarrow PC + 1$)

If (FGO = 1) then ($PC \leftarrow PC + 1$)

From the above two conditions, it has been observed that if flag = 1 then the following operations are performed:

(i) The branch instruction is skipped and

(ii) An input or output instruction is implemented.

6. **ION (Interrupt Enable ON) and IOF (Interrupt Enable OFF)**

An IEN i.e., interrupt enable flip-flop is set and unset by using ION and IOF instruction respectively.

Q41. Discuss about the sequence of steps that occurs when an interrupt occurs.

Answer :

Interrupt

An interrupt is an asynchronous event that halts the normal program execution and diverts the program flow temporarily to an interrupted routine.

Interrupts are caused both by the hardware and software.

On occurrence of an interrupt, the current status of the running program is stored and the control is transferred to ISR (which is responsible for handling interrupts) after its execution, the control switches back to the execution of the suspended program.

Interrupt Processing

A number of events are triggered in the processor hardware and software due to the occurrence of interrupts. Below figure shows the sequence of operations that results when an interrupt occur. When an I/O operation is being completed by any I/O device, it undergoes a number of hardware events as follows.

1. The interrupt issued by the device is signalled to the processor.
2. Prior to interrupt responding, the processor has to complete the currently executing instruction.

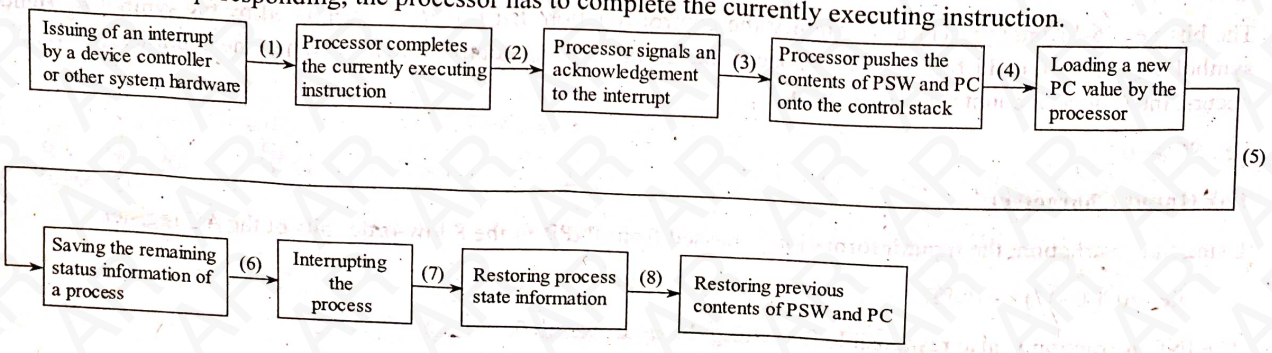


Figure: Simple Interrupt Processing

3. The processor performs a test to determine the presence of any pending requests to handle the interrupts and if so, sends an acknowledgment to the device that issued the interrupt. The acknowledgment causes a device to remove its interrupt signal.

4. The processor now starts preparing to transfer control to the interrupt routine but, prior to transferring the control, it stores the contents of the PSW and PC (location of the next instruction to be executed) on a system control stack, so that after completing the execution of the interrupted routine, the saved information can be retrieved from the stack to resume the execution of the suspended routine.

5. At this instance, the processor loads the starting address of the interrupt handling routine into the PC. Based on the computer architecture and the design of OS, a single program for each type of interrupt or for each device and each type of interrupt, is available. In case of multiple interrupt-handling routines, it is the responsibility of a processor to determine which one to invoke.

   After loading the PC with the address of interrupt-handling routine, the processor initiates the next instruction cycle that starts with the 'fetch' instruction, which can be determined from the contents of PC. Hence, the control is transferred to an interrupt-handling routine.

6. Once, the PC and PSW related to the interrupted routine are saved on the system stack, other status-related information of the executing program must be considered. Among which the information about the processor registers is an important, as these registers are used by the interrupt handler. Hence, the data along with the other state-related information must be stored.

   Usually, an interrupt handler will begin its execution by saving the contents of all the registers on the stack.

7. The interrupt is now being processed by the interrupt handler which includes testing of status information related to the I/O operation or other event that caused an interrupt. It also handles sending of additional commands or acknowledgments to the I/O device.

8. After the completion of interrupt processing, all the register values that were stored on the stack are retrieved and restored back to the registers.

9. Finally, the values of PSW or PC must be restored from the stack. Hence, the next instruction to be executed will be from the previously interrupted program.

Q42. What is a program interrupt? Draw and explain the flowchart of interrupt cycle.

Answer :

Program Interrupt

While a program is running on a computer, to transfer information from computer to input-output device, firstly we have to set the flag bit. By enabling the flag bit, computer initiates the transfer of data between computer and input-output devices. The rate of information transfer between computer and input-output device makes the system inefficient if the information flow among the devices are different i.e., computer has to check the flag bit several times in order to decide whether to transfer or not the information. Thus, the time required to check the flag bit is wasted resulting in inefficiency as no other useful task can be performed at the same time.

To eliminate consumption of time required to check flag bit each time, an external device of a computer should be assigned with a decision making task. This decision making device acts as a program interrupt which decides when information should be transferred or not. It keeps track of flag bit and informs the computer regarding the possibility of transferring information. A computer does not require to check flag bit as program interrupt informs the computer to transfer information whenever IEN = 1. Even though computer is running a program, program interrupt informs computer to halt the program for a while and take care of input-output transfer. Once the transfer is complete, the computer gets back to its position and continues with its work.

Two instructions for Interrupt Enable Flip-flop (IEN) are,

1. ION (Interrupt enable ON)

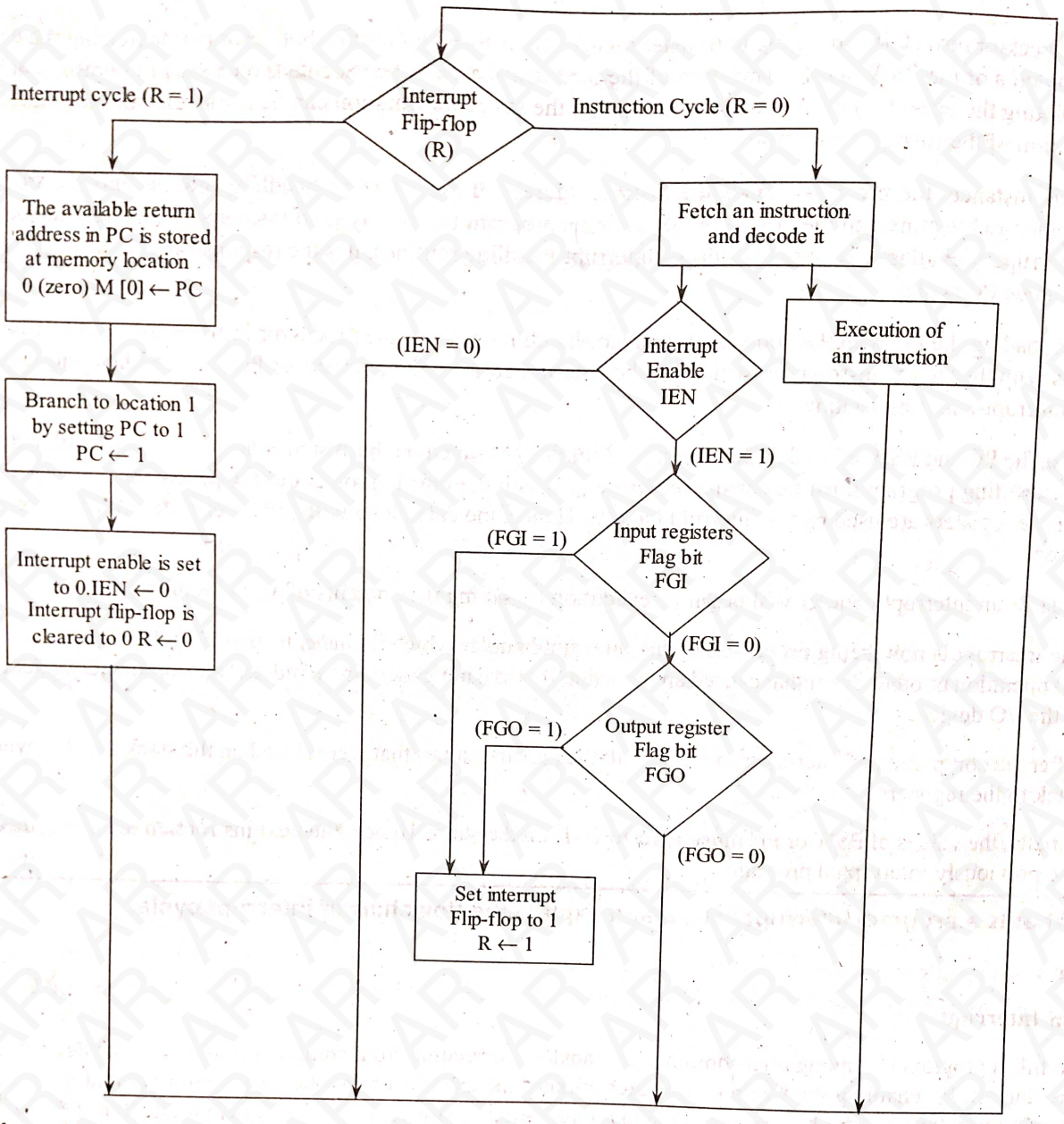
   i.e., IEN $\leftarrow$ 1 (When IEN is set to 1 it interrupts computer)

2. IOF (Interrupt enable OFF)

   i.e., IEN $\leftarrow$ 0 (When IEN is cleared to 0 it will not interrupt computer)

Flowchart of Interrupt Cycle

Flowchart explaining the way interrupts can be handled is as follows,



Consider an interrupt flip-flop R, control checks, whether R = 0 or 1.

- If R = 0, then computer is subjected to instruction cycle.
- If R = 1, then computer is subjected to interrupt cycle.

In an instruction cycle, the control checks whether the Interrupt Enable Flip-flop (IEN) = 0 or 1.

- If IEN = 0, control moves to the next instruction cycle leaving the programmer to continue with his work.
- If IEN = 1, control checks the flag bits of input and output register.
- If flag bits of input and output registers are equal to zero i.e., FGI = 0 and FGO = 0, then no information transfer occurs between the two registers.
- If IEN = 1 and any of the flag bits indicate 1 i.e., (FGI = 0 or FGO = 1, FGI = 1 or FGO = 0), then set the flip-flop R to 1,

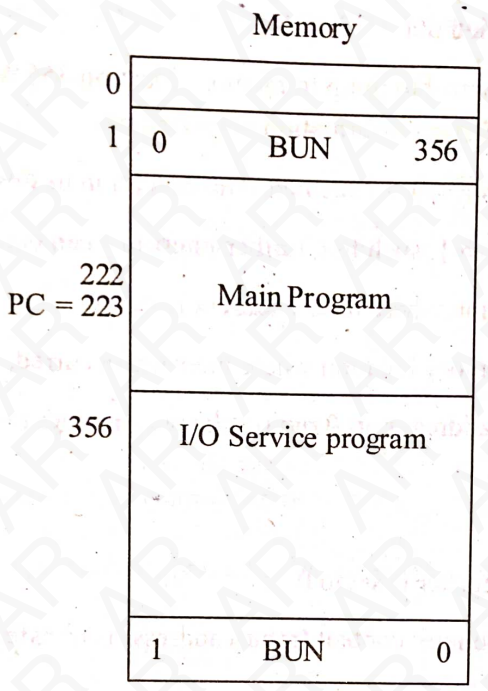
  ⇒ (IEN) (FGI + FGO) : R ← 1

  Symbol (+) indicates OR operation
- Finally, during the execution phase, if the control encounters the value of flip flop (R) as 1 i.e., R = 1 then the control is directed towards interrupt cycle.

In an interrupt cycle, when an interrupt occurs then the programmer stops running a program and switches to the location where information transfer has to be done. During information transfer to avoid further interrupts IEN and R must be cleared. Once the information transfer is over, the programmer has to get back to the location by checking the return address available in PC. PC containing return address is stored at a particular location which provides necessary information to the programmer to return to the instruction and the location at which interrupt occurred.

Example

Let us consider a memory stack before interruption.

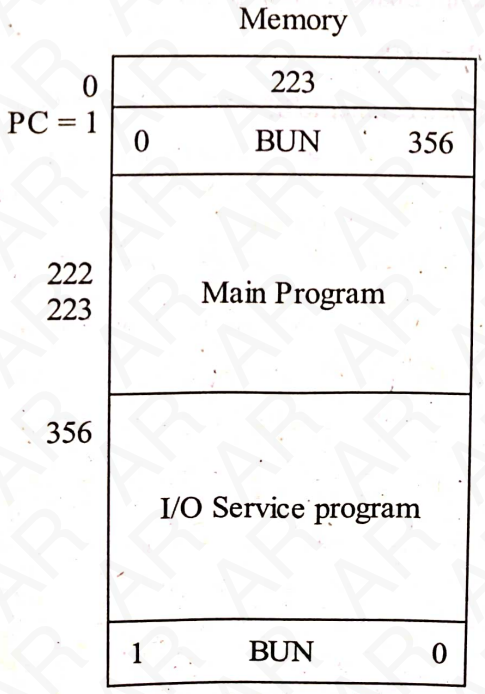


Where (R = 0), indicates that no interrupt occurred.

∴ Control is executing the instruction indicating an address value 222. The PC contains return address 223.

❖ At address 1 BUN 356 instruction is mentioned by the programmer.

Memory Stack After Interruption



- When (R = 1), the value of PC i.e., return address is stored at memory location 0.
- Next PC is set to 1

  PC ← 1
- Then R is set to 0

  R ← 0
- At the starting of the next instruction cycle, PC is at memory location 1 which points to a branch instruction. At this step, the corresponding instruction is carried out.
- The branch instruction causes the control to jump to memory location 356 where input-output services such as checking and determining flag bits, and transferring information is carried out.
- The instruction following the I/O program is executed. The execution of this instruction contains.
  - The value of IEN which is set to 1, such that further interrupts can occur.
  - It also provides memory location where return address is stored.
- Finally the control moves to the memory location where interrupt occurred, and continues with the next instruction.
- Branch indirect instruction with an address port 0 returns the programmer to the original location and computer proceeds with the next instruction.

Microoperations for Interrupt Cycle

T_0 : AR ← 0, // Address register is set to 0

TR ← PC, // Program counter content (return address) is transferred to temporary register

T_1 : M [AR] ← TR

// Return address available in TR is transferred to the address of memory word selected
// by the address register and at that location return address will be stored.

PC ← 0 // Program counter is cleared to 0

T_2 : PC ← PC+1 // Increment PC

IEN ← 0 // Clear Interrupt Enable Flip-flop to 0

R ← 0 // Clear flip-flop to 0

SC ← 0 // Clear sequence counter to 0.