Department of ISE

Internal Assessment Test 2 – Jan 2024

Evaluation scheme

Sub:	**Digital Design and Computer Organization**					Sub Code:	**BCS302**	Branch:	**ISE**
Date:	**17.01.2024**	Duration:	90 min's	Max Marks:	50	Sem/Sec	03/ A,B,C		OBE

	Answer any FIVE FULL QUESTIONS	MARKS	CO	RBT
1(a)	Design a VERILOG code for 4x1 Multiplexer using dataflow modeling Verilog code-2 Marks Testbench Code-2 Marks Output Waveform-1 Mark	5	CO2	L3
1(a)	Design a VERILOG code for 1x8 De-multiplexer using behavioral modeling Verilog code-2 Marks Testbench Code-2 Marks Output Waveform-1 Mark	5	CO2	L3
2(a)	Construct a 16 x 1 multiplexer with two 8 x 1 and one 2 x 1 multiplexers. Use block diagrams Block Diagram level 1-2 Marks Block Diagram level 2-2 Marks	4	CO2	L3
2(b)	Implement the given Boolean function F (A, B, C) = Σ (1, 2, 6, 7) using 4x1Multiplexer Truth Table-3 Marks Block Diagram-3 Marks	6	CO2	L3
3(a)	Draw the truth table of SR flipflop and JK flipflop Truth table of SR flipflop -2 Marks Truth table of JK flipflop-2 Marks	4	CO2	L2
3(b)	Describe the operation of T Flipflop with a neat sketch T Flipflop Block and logic Diagram-2 Marks Operation Description-2 Marks Truth Table, Characteristic Table, Characteristic Equations-2 Marks	6	CO2	L2
4(a)	Describe the functional units of computer with a neat diagram Block Diagram-1 Marks Input Unit Description-1 Marks CPU Unit Description-1 Marks Memory Unit Description-1 Marks Output Unit Description-1 Marks	5	CO3	L2
4(b)	Describe the memory locations and addresses of a computer in detail Memory locations-3 Marks Memory Addresses-2 Marks	5	CO3	L2

Dr.Ciyamala Kushbu S, AP/ISE,CMRIT

5	Describe in detail the Performance –Processor Clock, Basic Performance Equation, Clock Rate, Performance Measurement of a computer Performance Defintion-2 Marks Processor Clock definition-2 Marks Basic Performance Equation-2 Marks Clock Rate description-2 Marks Performance Measurement of a computer-2 Marks	10	CO3	L2
6	Describe the addressing modes in detail with necessary syntax and example Ten Addressing mode with definition, syntax and example –(Each carries 1 Mark)	10	CO3	L2

CCI **CI** **HOD**

Dr.Ciyamala Kushbu S, AP/ISE,CMRIT

Department of ISE

Internal Assessment Test 2 – Jan. 2024

IAT-2 solutions

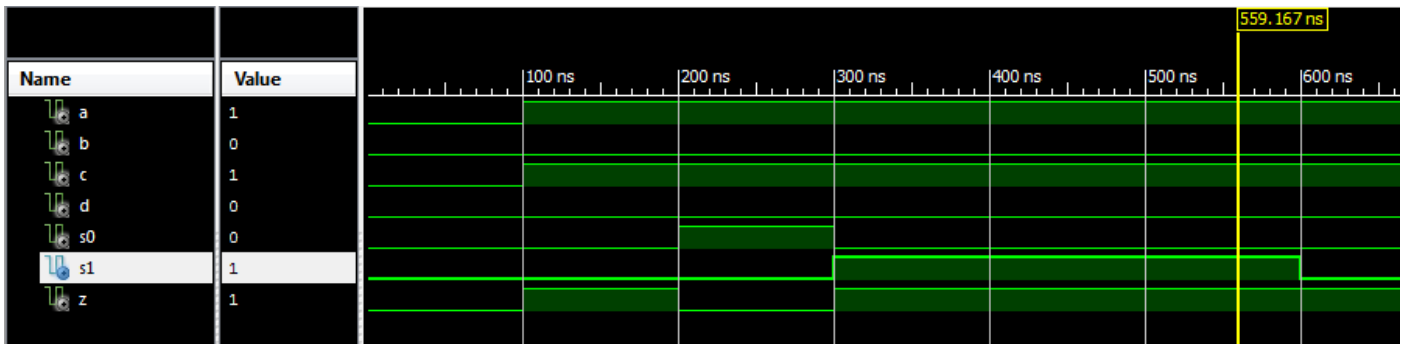
1(a)Design a VERILOG code for 4x1 Multiplexer using dataflow modeling

Verilog code:

```
module m41 ( input a, input b, input c, input d, input s0, s1,output out);
 assign out = s1 ? (s0 ? d : c) : (s0 ? b : a);
endmodule
```

Testbench code:

```
module top;
wire  out;
reg  a;
reg  b;
reg  c;
reg  d;
reg s0, s1;
m41 name(.out(out), .a(a), .b(b), .c(c), .d(d), .s0(s0), .s1(s1));
 initial begin
 a=1'b0; b=1'b0; c=1'b0; d=1'b0;
 s0=1'b0; s1=1'b0;
 #500 $finish;
end
always #40 a=~a;
always #20 b=~b;
always #10 c=~c;
always #5 d=~d;
always #80 s0=~s0;
always #160 s1=~s1;
always@(a or b or c or d or s0 or s1)
$monitor("At time = %t, Output = %d", $time, out);
Endmodule
```



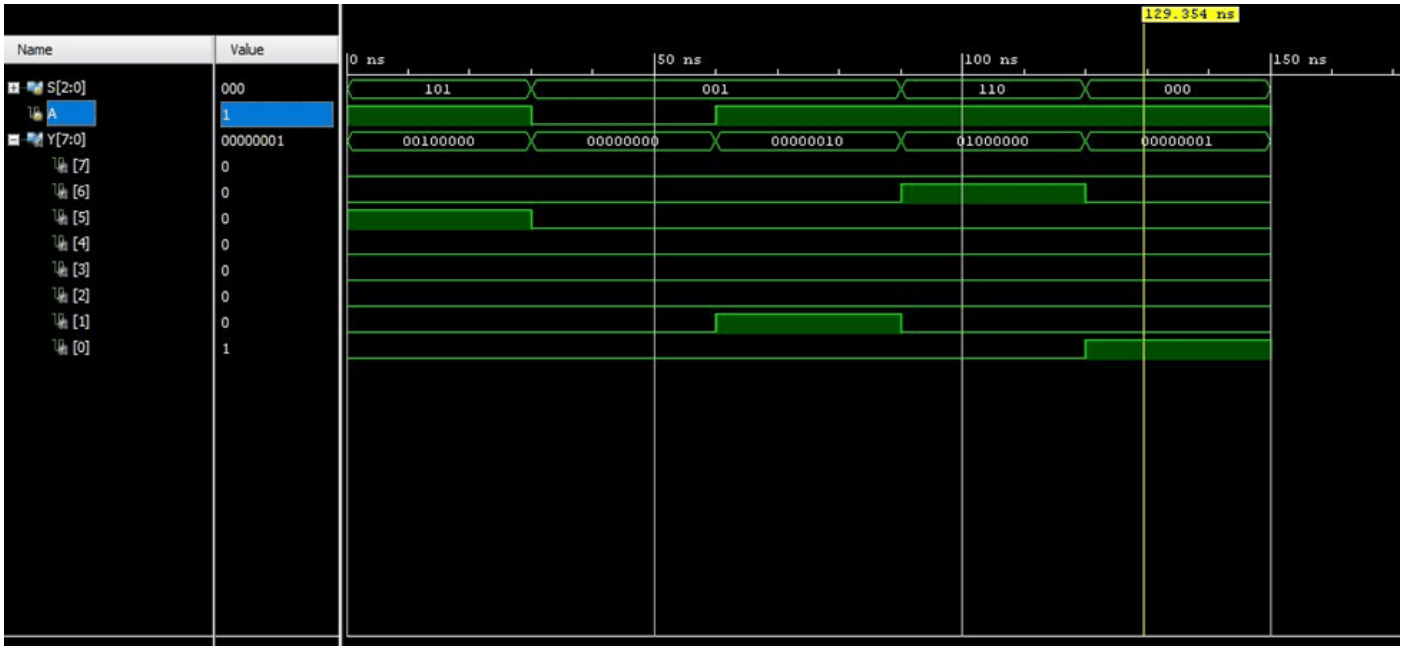
1(b) Design a VERILOG code for 8x1 De-multiplexer using behavioral modeling

Verilog code:

```
module demux_1_8(y,s,a);
output reg [7:0]y;
input [2:0]s;
input a;
always @(*)
begin
y=0;
case(s)
3'd0: y[0]=a;
3'd1: y[1]=a;
3'd2: y[2]=a;
3'd3: y[3]=a;
3'd4: y[4]=a;
3'd5: y[5]=a;
3'd6: y[6]=a;
3'd7: y[7]=a;
endcase
end
endmodule
```

Testbench code:

```
module test_demux;
reg [2:0]S;
reg A;
wire [7:0]Y;
demux_1_8 mydemux(.y(Y), .a(A), .s(S));
initial begin
A=1;
S=3'd5;
#30;
A=0;
S=3'd1;
#30;
A=1;
S=3'd1;
#30;
S=3'd6;
#30;
S=3'd0;
#30;
$finish;
end
endmodule
```

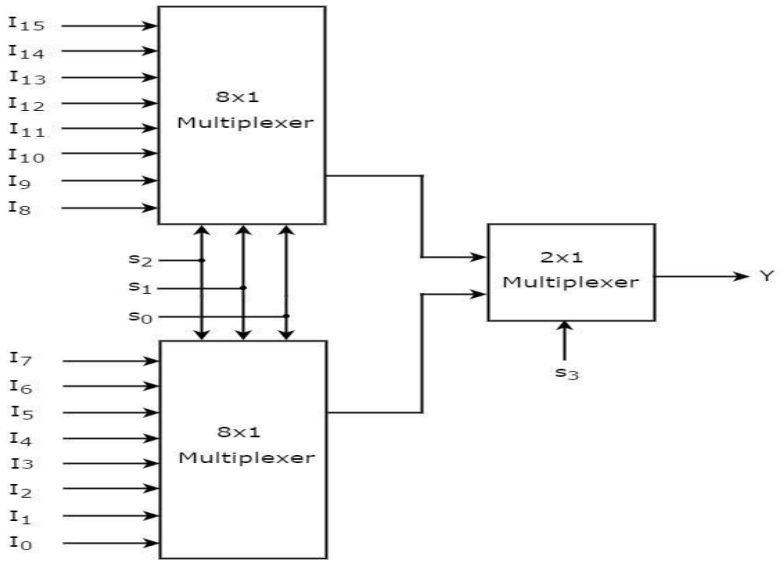


2(a) Construct a 16 x 1 multiplexer with two 8 x 1 and one 2 x 1 multiplexers. Use block diagrams

Let us implement 16x1 Multiplexer using 8x1 Multiplexers and 2x1 Multiplexer. We know that 8x1 Multiplexer has 8 data inputs, 3 selection lines and one output. Whereas, 16x1 Multiplexer has 16 data inputs, 4 selection lines and one output.So, we require two **8x1 Multiplexers** in first stage in order to get the 16 data inputs. Since, each 8x1 Multiplexer produces one output, we require a 2x1 Multiplexer in second stage by considering the outputs of first stage as inputs and to produce the final output.Let the 16x1 Multiplexer has sixteen data inputs I_{15} to I_0, four selection lines s_3 to s_0 and one output Y. The **Truth table** of 16x1 Multiplexer is shown below.

Selection Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	I_0
0	0	0	1	I_1
0	0	1	0	I_2
0	0	1	1	I_3
0	1	0	0	I_4
0	1	0	1	I_5
0	1	1	0	I_6
0	1	1	1	I_7
1	0	0	0	I_8
1	0	0	1	I_9
1	0	1	0	I_{10}
1	0	1	1	I_{11}
1	1	0	0	I_{12}
1	1	0	1	I_{13}
1	1	1	0	I_{14}
1	1	1	1	I_{15}

We can implement 16x1 Multiplexer using lower order Multiplexers easily by considering the above Truth table. The **block diagram** of 16x1 Multiplexer is shown in the following figure.



The **same selection lines, s_2, s_1 & s_0** are applied to both 8x1 Multiplexers. The data inputs of upper 8x1 Multiplexer are I_{15} to I_8 and the data inputs of lower 8x1 Multiplexer are I_7 to I_0. Therefore, each 8x1 Multiplexer produces an output based on the values of selection lines, s_2, s_1 & s_0.

The outputs of first stage 8x1 Multiplexers are applied as inputs of 2x1 Multiplexer that is present in second stage. The other **selection line, s_3** is applied to 2x1 Multiplexer.

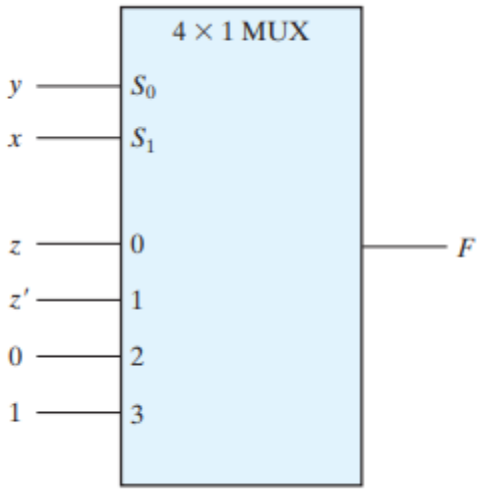
- If s_3 is zero, then the output of 2x1 Multiplexer will be one of the 8 inputs Is_7 to I_0 based on the values of selection lines s_2, s_1 & s_0.
- If s_3 is one, then the output of 2x1 Multiplexer will be one of the 8 inputs I_{15} to I_8 based on the values of selection lines s_2, s_1 & s_0.

Therefore, the overall combination of two 8x1 Multiplexers and one 2x1 Multiplexer performs as one 16x1 Multiplexer.

2(b) Implement the given Boolean function F (A, B, C) = Σ (1, 2, 6, 7) using 4x1Multiplexer

x	y	z	F	
0	0	0	0	$F = z$
0	0	1	1	
0	1	0	1	$F = z'$
0	1	1	0	
1	0	0	0	$F = 0$
1	0	1	0	
1	1	0	1	$F = 1$
1	1	1	1	

(a) Truth table



(b) Multiplexer implementation

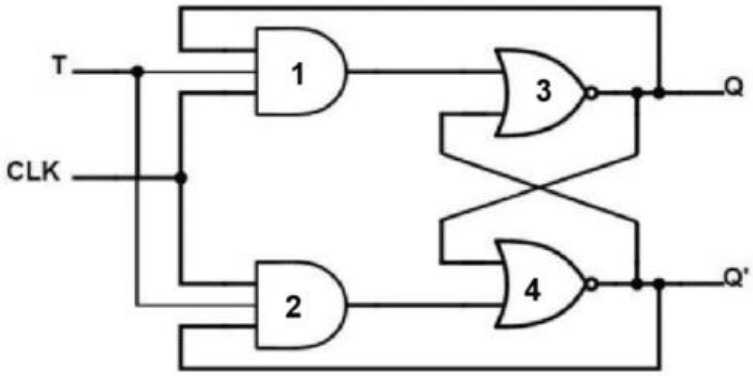
3(a) Draw the truth table of SR flipflop and JK flipflop

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Indeterminate

CLK	J	K	Q_{n+1}
↑	0	0	Q_n
↑	0	1	0
↑	1	0	1
↑	1	1	Q_n'

3(b) Describe the operation of T Flipflop with a neat sketch

T flip flop is similar to JK flip flop. Just tie both J and K inputs together to get a T Flip flop. Just like the D flip flop, it has only one external input along with a clock.



T Flip-Flop Working

Let us take a look at the possible cases and write it down in our truth table. The clock is always 1, so only two cases are possible where T can be high or low.

Case 1: T=0

Gate1 = 0, Gate2 = 0, Gate3/Q(n+1) = Q, Gate4/Q(n+1)' = Q'

Note:

- Since one of the inputs to Gate1 and gate2 is 0 and both are AND gates; the output of gate1 and gate2 will be equal to 0 irrespective of other inputs as per the property of AND gates
- Gate3 = (0+Q')' = (Q')' = Q

- Gate4 = (0+Q)' = (Q)' = Q'

Case 2: T=1

Gate1 = Q, Gate2 = Q', Gate4/Q(n+1)' = 0, Gate3/Q(n+1) = Q'

Note:

- Since one input of both gate1 and gate2 is 0 and both gates are AND gates, the output of both gates will be equal to the third input.
- Gate4 = (Q'+Q)' = 1' = 0
- Gate3 = (Q+0)' = Q'

Now let us write the truth table-

T Flip-Flop Truth Table

CLK	T	Q(n+1)	State
↑	0	Q	NO CHANGE
↑	1	Q'	TOGGLE

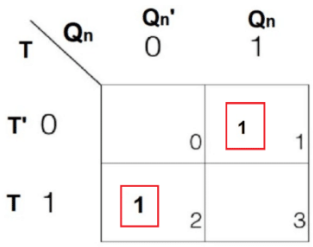
We will use this truth table to write the characteristics table for the T flip flop. In the truth table, you can see there is only one input T and one output Q(n+1). But in the characteristics table, you will see there are two inputs T and Qn, and one output Q(n+1).

From the logic diagram above it is clear that Qn and Qn' are two complementary outputs that also act as inputs for Gate3 and Gate4 hence we will consider Qn i.e the present state of Flip flop as input and Q(n+1) i.e. the next state as output.

After writing the characteristic table, we will draw a 2-variable K-map to derive the characteristic equation.

Characteristic table

T	Qn	Q(n+1)
0	0	0
0	1	1
1	0	1
1	1	0



T Flip-Flop K-Map

From the K-map you get 2 pairs. On solving both we get the following characteristic equation:

Q(n+1) = TQn' + T'Qn = T XOR Qn

Advantages

There are several advantages to using a T flip flop. Some of them are listed below:

- **Single input:** The T flip-flop has a single input that can be used to toggle between two states, which makes it simpler to use and easier to interface with other digital circuits.

- **No invalid states:** The T flip-flop does not have any invalid states, which helps to avoid unpredictable behavior in digital systems.
- **Reduced power consumption:** The T flip-flop consumes less power than other types of flip-flops, making it more energy-efficient.
- **Bi-stable operation:** Like other flip-flops, the T flip-flop has a bi-stable operation, which means that it can hold a state indefinitely until it is changed by an input signal.
- **Easy to implement:** The T flip-flop can be easily implemented using simple logic gates, which makes it a cost-effective option for digital systems.

Limitations

Apart from several advantages, there are some limitations associated with T flip-flops. Some of them are listed below:

- **Inverted output:** The output of a T flip-flop is inverted from the input, which can be confusing and make it difficult to design sequential logic circuits.
- **Limited functionality:** The T flip-flop can only store a single bit of information and cannot perform more complex operations like addition or multiplication.
- **Glitches:** The T flip-flop is vulnerable to glitches and noise on the input signal, which can cause it to toggle unexpectedly and lead to unpredictable behavior in digital systems.
- **Propagation delay:** Like other flip-flops, the T flip-flop has a propagation delay, which can lead to timing issues in digital systems with tight timing constraints.

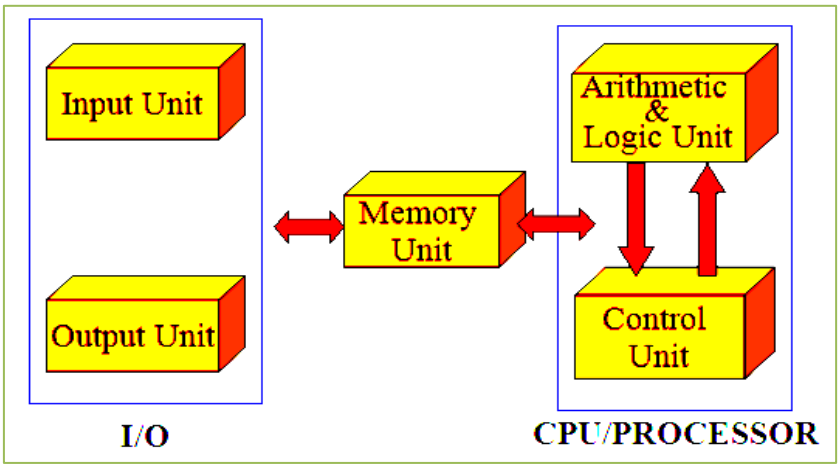
Applications

Some of the applications of T flip flop in real-world includes:

- **Frequency division:** The T flip-flop can be used to divide the frequency of a clock signal by two, making it useful in applications such as digital clocks and frequency synthesizers.
- **Frequency multiplication:** The T flip-flop can also be used to multiply the frequency of a clock signal by two, making it useful in applications such as frequency synthesizers and digital signal processing.
- **Data storage:** The T flip-flop can be used to store a single bit of data, making it useful in applications such as shift registers and memory devices.
- **Counters:** The T flip-flop can be used in conjunction with other digital logic gates to create binary counters that can count up or down depending on the design. This makes them useful in real-time applications such as timers and clocks.

4(a) Describe the functional units of computer with a neat diagram

A computer in its simplest form comprises five functional units namely input unit, output unit memory unit, arithmetic & logic unit and control unit. Figure 3.1 depicts the functional units of a computer system.



Basic Functional Units of a Computer

1.Input Unit:

Computer accepts coded information through input units, which read the data. The primary input device is typically a keyboard. Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor. Many other kinds of input devices are available, including Mouse, Joystick, Tracker ball, Light pen, Digitizer, Scanner etc.

2. Memory Unit:

Memory unit stores the program instructions (Code), data and results of computations etc. Memory unit is classified as: Primary /Main Memory and Secondary /Auxiliary Memory.

Primary/Main memory:

The largest and slowest unit is referred to as main memory. It is a semiconductor memory that operates at electronic speed. Run time program instructions and operands are stored in the main memory. It is faster than secondary storage (like hard drives or SSDs) but loses its content when the power is turned off.

The memory contains a large number of semiconductor storage cells, each capable of storing one bit of information. These cells are rarely read or written as individual cells but instead are processed in group of fixed size called **words.**

The number of bits in each word is often referred as the **word length** of the computer. Typical word lengths range from 16 to 64 bits. Data are usually processed within a machine in units of words, multiple of words, or parts of words. When the memory is accessed, usually only one word of data is read or written.

Addresses are numbers that associates each word location. A given word is accessed by specifying its address and issuing a control command that starts the storage or retrieval process.

The main memory is classified again as ROM and RAM. **ROM** (Read Only Memory) holds system programs and firmware routines such as BIOS, POST, I/O Drivers that are essential to manage the hardware of a computer. Memory in which any location can be reached in a short and fixed amount of time after specifying its address is called **RAM** (Random Access Memory). It is also termed as Read/Write memory or user memory that holds run time program instruction and data. The small, fats RAM units are called caches. While primary storage is essential, it is volatile in nature and expensive.

Secondary /Auxiliary Memory:

Secondary memories are non volatile in nature. It also stores large amount of data and programs particularly for information that is accessed infrequently at cheaper cost. Example: Magnetic disks and tapes, optical disks(CD-ROMs), SD card, Pen drive, Hard disk etc.

3. Arithmetic and logic unit:

ALU consist of necessary logic circuits like adder, comparator etc., to perform computer operations of addition, multiplication, comparison of two numbers etc.

Suppose two numbers located in the memory are to be added, they are brought into the processor, and the actual addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.

When operands are brought into the processor, they are stored in high speed storage elements called registers. Each register can store one word of data. Access times to registers are somewhat faster than access time to the fastest cache unit in the memory hierarchy.

4. Output Unit:

Computer after computation returns the computed results, error messages, etc. via output unit to the outside world. The standard output device is a video monitor, LCD/TFT monitor. Other output devices are printers, plotters etc. Some units such a graphic displays, provide both an output function and an input function.

5. Control Unit:

Control unit co-ordinates activities of all units such as the Memory, ALU, Input and Output units by issuing control signals. Control signals issued by control unit govern the data transfers and then appropriate operations take place.

The actual timing signals that govern the transfers are generated by the control circuits. The **timing signals** are the signals that determines when a given action is to take place. Data transfers between the processor and the memory are also controlled by the control unit through the timing signals.

The operations of a computer can be summarized as follows:

1. The computer accepts information in the form of programs and data through **input unit** and a set of instructions called programs are stored in the **main memory** of computer.
2. The **CPU** fetches those instructions sequentially one-by-one from the main memory, decodes them and performs the specified operation on associated data operands in **ALU.**
3. Processed data and results will be displayed on an **output unit.**
4. All activities pertaining to processing and data movement inside the computer machine are governed by **control unit.**

4(b) Describe the memory locations and addresses of a computer in detail

The memory consists of many millions of storage cells, each of which can store a bit of information having the value 0 or 1. It is organized in such a way that a group of n bits can be stored or retrieved in a single, basic operation. Each group of n bits is referred to as a word of information, and n is called the word length. The memory of a computer can be schematically represented as a collection of words, as shown in

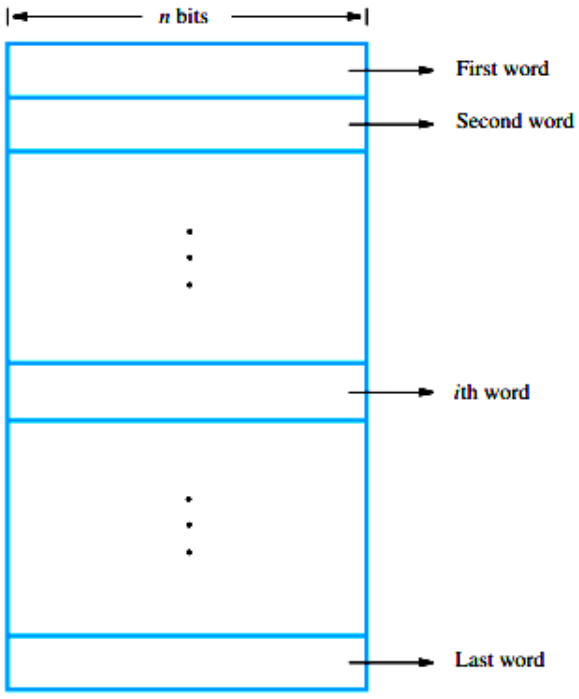


Figure 3.7: Memory words

Modern computers have word lengths that typically range from 16 to 64 bits. If the word length of a computer is 32 bits, a single word can store a 32-bit signed number or four ASCII-encoded characters, each occupying 8 bits, as shown in Figure 3.8. A unit of 8 bits is called a byte. Machine instructions may require one or more words for their representation

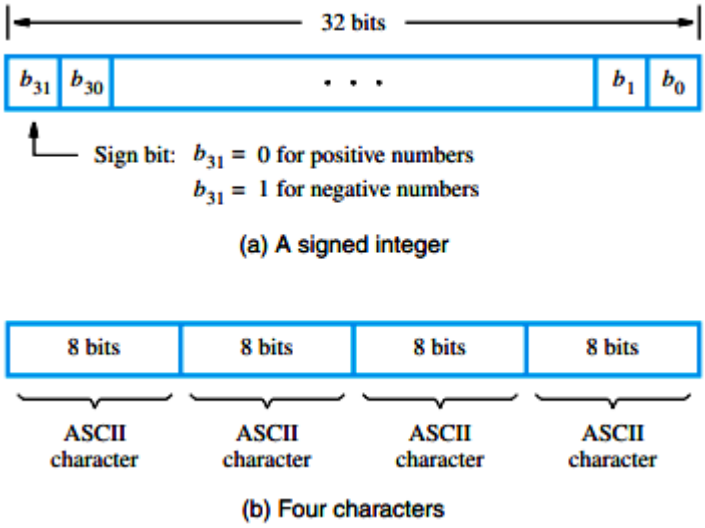


Figure 3.8 Encoded information in a 32-bit word

Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or addresses for each location. It is customary to use numbers from 0 to $2^k - 1$, for some suitable value of k, as the addresses of successive locations in the memory.

Thus, the memory can have up to 2^k addressable locations. The 2^k addresses constitute the address space of the computer. For example, a 24-bit address generates an address space of 2^{24} (16,777,216) locations. This number is usually written as 16M (16 mega), where 1M is the number 2^{20} (1,048,576). A 32-bit address creates an address space of 2^{32} or 4G (4 giga) locations, where 1G is 2^{30}.

Note: Commonly used notations are K (kilo) for the number 2^{10} (1,024), and T (tera) for the number 2^{40}.

3.6.1 Byte Addressability

A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. It is impractical to assign distinct addresses to individual bit locations in the memory. The most practical assignment is to have successive addresses refer to successive byte locations in the memory. This assignment used in most modern computers is known as Byte Addressability.

Byte locations have addresses 0, 1, 2,.... Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8,..., with each word consisting of four bytes

3.6.2 Big-Endian and Little-Endian Assignments

There are two ways that byte addresses can be assigned across words, as shown in Figure 3.9

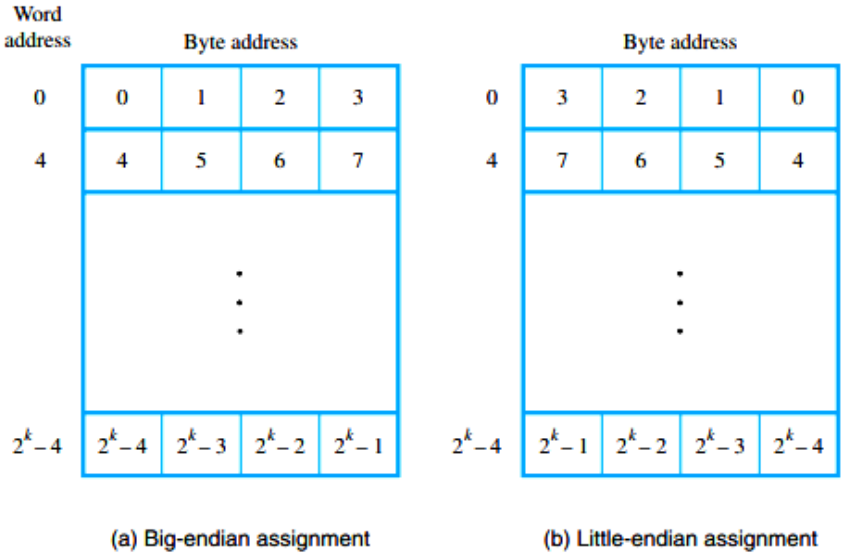


Figure 3.9: Byte and word addressing

The name **big-endian** is used when lower byte addresses are used for the more significant bytes (the leftmost bytes) of the word.

The name **little-endian** is used for the opposite ordering, where the lower byte addresses are used for the less significant bytes (the rightmost bytes) of the word.

In both cases, byte addresses 0, 4, 8,..., are taken as the addresses of successive words in the memory of a computer with a 32-bit word length. These are the addresses used when accessing the memory to store or retrieve a word.

3.6.3 Word Alignment

The word locations have aligned addresses as they begin at a byte address that is a multiple of the number of bytes in a word. Hence, if the word length is 16 (2 bytes), aligned words begin at byte addresses 0, 2, 4,..., and for a word length of 64 (23 bytes), aligned words begin at byte addresses 0, 8, 16,....

3.6.4 Accessing Numbers, Characters and strings

A number usually occupies one word, and can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte address and strings by successive byte addresses.

5 Describe in detail the Performance –Processor Clock, Basic Performance Equation, Clock Rate?

The most important measure of the performance of a computer is how quickly it can execute programs. The speed with which a computer executes programs is affected by the design of its instruction set, its hardware and its software, including the operating system, and the technology in which the hardware is implemented. Because programs are usually written in a high-level language, performance is also affected by the compiler that translates programs into machine language. For best performance, it is necessary to design the compiler, the machine instruction set, and the hardware in a coordinated way.

The total time required to execute the program is elapsed time is a measure of the performance of the entire computer system. It is affected by the speed of the processor, the disk and the printer. The time needed to execute a instruction is called the processor time. In **Figure 3.5**, the elapsed time is $\mathbf{t_5\text{-}t_0}$.

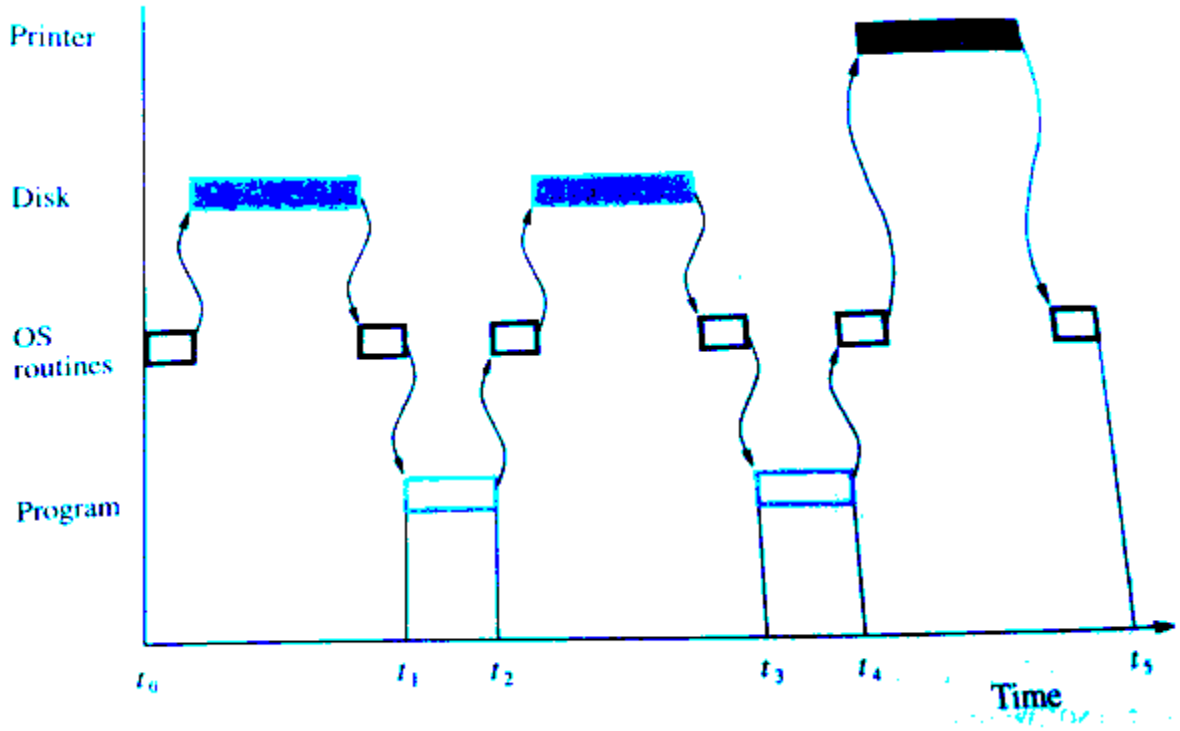


Figure 3.5: User program and OS routine sharing of the processor

Just as the elapsed time for the execution of a program depends on all units in a computer system, the processor time depends on the hardware involved in the execution of individual machine instructions. This hardware comprises the processor, cache memory as a part of the processor and the memory which are usually connected by the bus as shown in the Figure 3.6.

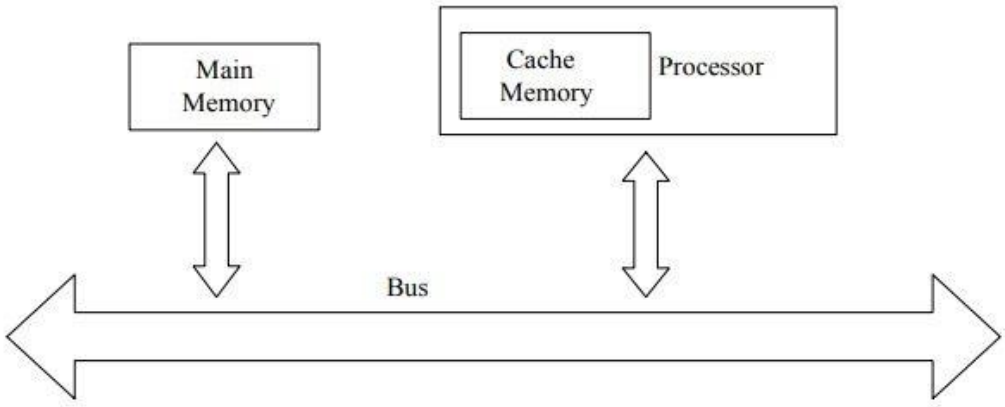


Figure 3.6: Processor Cache

Let us examine the flow of program instructions and data between the memory and the processor. At the start of execution, all program instructions and the required data are stored in the main memory. As the execution proceeds, instructions are fetched one by one over the bus into the processor, and a copy is placed in the cache later if the same instruction or data item is needed a second time, it is read directly from the cache.

The processor and relatively small cache memory can be fabricated on a single IC chip. The internal speed of performing the basic steps of instruction processing on chip is very high and is considerably faster than the speed at which the instruction and data can be fetched from the main memory.

A program will be executed faster if the movement of instructions and data between the main memory and the processor is minimized, which is achieved by using the cache.

For example:- Suppose a number of instructions are executed repeatedly over a short period of time as happens in a program loop. If these instructions are available in the cache, they can be fetched quickly during the period of repeated use. The same applies to the data that are used repeatedly.

3.5.1 Processor Clock:

Processor circuits are controlled by a timing signal called **clock**. The clock defines the regular time intervals called **clock cycles**. To execute a machine instruction the processor divides the action to be performed into a sequence of basic steps that each step can be completed in one clock cycle.

The length P of one clock cycle is an important parameter that affects the processor performance. Processor used in today's personal computer and work station have a clock rates that range from a few hundred million to over a billion cycles per second.

3.5.2 Basic Performance Equation:

Let **'T' be the processor time** required to execute a program that has been prepared in some high-level language.

The compiler generates a machine language object program that corresponds to the source program. Assume that complete execution of the program requires the execution of N machine cycle language instructions.

The number **N is the actual number of instruction execution** and is not necessarily equal to the number of machine cycle instructions in the object program.

Some instruction may be executed more than once, which in the case for instructions inside a program loop others may not be executed all, depending on the input data used.

Suppose that the **average number of basic steps** needed to execute one machine cycle instruction is **S,** where each basic step is completed in one clock cycle. If clock rate is 'R' cycles per second, the program execution time is given by

$$T = \frac{N \times S}{R} \quad \text{----------(1)}$$

Equation 1 is often referred to as the **Basic Performance Equation**. We must emphasize that N, S & R are not independent parameters, Hence, changing one, may affect another.

Introducing a new feature in the design of a processor will lead to improved performance only if the overall result is to reduce the value of T.

3.5.3 Clock Rate:

Clock Rate R is inversely proportional to the Clock performance P where,

$$R = \frac{1}{P} \quad \text{----------(2)}$$

It is measured in Cycles per second.

These are two possibilities for increasing the clock rate 'R'.

1. **Improving the IC technology** makes logical circuit faster, which reduces the time of execution of basic steps. This allows the clock period P, to be reduced and the clock rate R to be increased.

2. **Reducing the amount of processing** done in one basic step also makes it possible to reduce the clock period P. However if the actions that have to be performed by an instructions remain the same, the number of basic steps needed may increase.

Increase in the value 'R' that is entirely caused by improvements in IC technology affects all aspects of the processor's operation equally with the exception of the time it takes to access the main memory. The value of T will be reduced by the same factor as R is increased.

3.5.4 Performance Measurement:

It is very important to be able to access the performance of a computer. Computer designers use performance estimates to evaluate the effectiveness of new features.

Hence measurement of computer performance using bench mark programs is done to make comparisons possible, standardized programs must be used.

The performance measure is the time taken by the computer to execute a given bench mark.

Initially some attempts were made to create artificial programs that could be used as bench mark programs. But synthetic programs do not properly predict the performance obtained when real application programs are run.

A **non-profit organization called SPEC- system** performance Evaluation Corporation selects and publishes bench marks.

The program selected range from game playing, compiler, and data base applications to numerically intensive programs in astrophysics and quantum chemistry. In each case, the program is compiled under test, and the running time on a real computer is measured. The same program is also compiled and run on one computer selected as reference.

The 'SPEC' rating is computed as follows.

$$\textbf{SPEC rating} = \frac{\textbf{Running time on the reference computer}}{\textbf{Running time on the computer under test}} \quad \textbf{---------(3)}$$

If the SPEC rating = 50, it means that the computer under test is 50 times as fast as the UltraSPARC-10. This is repeated for all the programs in the SPEC suit, and the geometric mean of the result is computed.

Let $SPEC_i$ be the rating for program 'i' in the suite. The overall SPEC rating for the computer is given by ,

$$\text{SPEC rating} = \left(\prod_{i=1}^{n} SPEC_i\right)^{\frac{1}{n}}$$

Where 'n' = number of programs in suite. Because the actual execution time is measured, the SPEC rating is a measure of the combined effect of all factors affecting performance, including the compiler, the OS, the processor, the memory of computer being tested.

6 Describe the addressing modes in detail with necessary syntax and example

The different ways for specifying the locations of operands in an instruction are known as **Addressing Modes.**

Name	Assembler syntax	Addressing function
Immediate	# Value	Operand = Value
Register	Ri	EA = Ri
Absolute (Direct)	LOC	EA = LOC
Indirect	(Ri)	EA = [Ri]
	(LOC)	EA = [LOC]
Index	X(Ri)	EA = [Ri] + X
Base with index	(Ri, Rj)	EA = [Ri] + [Rj]
Base with index and offset	X (Ri, Rj)	EA = [Ri] + [Rj] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(Ri)+	EA = [Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA = [Ri]

EA = effective address
Value = a signed number

Figure 3.12: Addressing modes

1.Immediate mode:

The operand is given explicitly in the instruction.

Syntax: #Value

Example : Move #200, R0

This instruction places the value 200 in register R0. Clearly, the Immediate mode is only used to specify the value of a source operand.

2.Register mode:

The operand is the contents of a processor register; the name of the register is given in the instruction.

Syntax: Ri

Example : Move R2, R1

This instruction moves the content of Register R2 to Register R1

3.Absolute or Direct mode:

The operand is in a memory location; the address of this location is given explicitly in the instruction.

Syntax: LOC

Example: Move LOC, R2

This instruction moves the content in memory location LOC to Register R2

4.Indirect mode:

The effective address of the operand is the contents of a register that is specified in the instruction.

Syntax: (Ri)

Example: Add (R1),R0

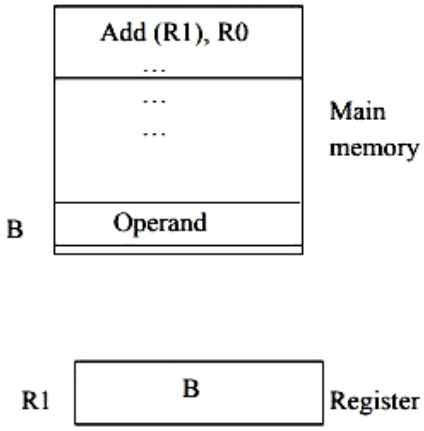


Figure 3.13: Indirect addressing

To execute the Add instruction in Figure 3.13, the processor uses the value which is in **register R1, as the effective address** of the operand.

It requests a read operation from the memory to **read the contents of location B.** the value read is the **desired operand,** which the processor adds to the contents of register R0 and the resultant sum is stored in R0.

5.Index mode:

The effective address of the operand is generated by adding a constant value to the contents of a register.

Syntax: X(Ri)

Where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The **effective address** of the operand is given by **EA = X + [Ri].**

Example: Add 20(R1),R2

Figure 3.14 Indexed addressing

In Figure 3.14, the index register, R1, contains the address of a memory location, and the value 20 defines an offset (also called a displacement) from this address to the location where the operand is found. This operand is added to the content in Register R2 and the resultant sum is placed inside Register R2.

6. Base with Index mode

The effective address of the operand is sum of the contents of registers Ri and Rj.

Syntax: (Ri,Rj)

EA = [Ri]+ [Rj]

Example: Add (R1,R2),R3

Here, the index registers, R1 and R2, contains the address of a memory location, which is summed up to give the address to the location where the operand is found. This operand is added to the content in Register R3 and the resultant sum is placed inside Register R3.

7. Base with Index and Offset mode

The effective address is the sum of the constant X and the contents of registers Ri and Rj.

Syntax: X(Ri,Rj)

EA = X + [Ri+[Rj]

Example: Add 20 (R1,R2),R3

Here, the index registers, R1 and R2, contains the address of a memory location, which is summed up, and the value 20 defines an offset (also called a displacement) from this address to the location to give the address to the location where the operand is found. This operand is added to the content in Register R3 and the resultant sum is placed inside Register R3.

8.Relative mode:

The effective address is determined by the Index mode using the program counter in place of the general-purpose register Ri.

Syntax: X(PC)

EA = X + [PC]

Example: Branch > 0 LOOP

This instruction **c**auses program execution to go to the branch target location identified by the name LOOP if the branch condition is satisfied.

This location can be computed by specifying it as an offset from the current value of the program counter. Since the branch target may be either before or after the branch instruction, the offset is given as a signed number.

9. Auto-increment mode:

The effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically to point to the next item in a list.

Syntax: (Ri)+

EA = [Ri]; Increment Ri

Example: Add (R2)+,R1

10. Auto-decrement mode:

The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand.

Syntax: -(Ri)

Decrement Ri ; EA = [Ri]

Example: Add -(R2),R1