Victoria University of Bangladesh
Name: Md. Ziaul Hoque "Sohel"
Student ID: 2221220031
Course Title: Computer Organization Assembly Programming
Course Code: CSE-233
Batch: 22nd (evening)
Semester: Spring-2024

Ans to the Que No 1(A)

Key steps at Modern Computer Systems:

A computer system works by combining input, storage space, processing, and output. These four are the major components of a computer.

Input: An input is the information that we provide to the computer. We provide the information using the computer's input devices: Keyboard, mouse, microphone, and many more. For example, when we type something using a keyboard, it is known as an Input provided to the computer.

Storage Space: It is the place where our input gets stored. It is known as Computer Memory that keeps the data into it. A computer uses a hard drive for storing files and documents. It uses two types of memory, i.e., internal memory and external memory. Internal memory is known as RAM, which is volatile in nature. It stores data temporarily, i.e., when the data is ready to be processed, is loaded into RAM, and after processing, it moves data for the storage. On the other hand, external memory is used to store data permanently until you remove it or it got crashed.

Processing: The processing of the input is performed by the CPU, which is the Central Processing Unit of the Computer. It is also known as the brain of a computer that is responsible for processing the data provided by the user. The speed of the computer brain is four times faster than the speed of the human brain.

Output: When we type something using a keyboard, the place where we see the typed input is the Computer Monitor or Computer Screen. A computer screen allows seeing the input we provided to the computer. Including this, there are different types of output devices of a computer, such as loudspeakers, projectors, printers, and many more.

Ans to the Que No 1(B)

Difference between low-level language and high-level language:

Parameter	High-Level Language	Low-Level Language
Basic	These are programmer-friendly languages that are manageable, easy to understand, debug, and widely used in today's times.	These are machine-friendly languages that are very difficult to understand by human beings but easy to interpret by machines.
Ease of Execution	These are very easy to execute.	These are very difficult to execute.

Process of Translation	High-level languages require the use of a compiler or an interpreter for their translation into the machine code.	Low-level language requires an assembler for directly translating the instructions of the machine language.
Efficiency of Memory	These languages have a very low memory efficiency. It means that they consume more memory than any low-level language.	These languages have a very high memory efficiency. It means that they consume less energy as compared to any high-level language.
Portability	These are portable from any one device to another.	A user cannot port these from one device to another.
Comprehensibility	High-level languages are human-friendly. They are, thus, very easy to understand and learn by any programmer.	Low-level languages are machine-friendly. They are, thus, very difficult to understand and learn by any human.
Dependency on Machines	High-level languages do not depend on machines.	Low-level languages are machine-dependent and thus very difficult to understand by a normal user.
Debugging	It is very easy to debug these languages.	A programmer cannot easily debug these languages.
Maintenance	High-level languages have a simple and comprehensive maintenance technique.	It is quite complex to maintain any low-level language.
Usage	High-level languages are very common and widely used for programming in today's times.	Low-level languages are not very common nowadays for programming.
Speed of Execution	High-level languages take more time for execution as compared to low-level languages because these require a translation program.	The translation speed of low-level languages is very high.
Abstraction	High-level languages allow a higher abstraction.	Low-level languages allow very little abstraction or no abstraction at all.
Need of Hardware	One does not require a knowledge of hardware for writing programs.	Having knowledge of hardware is a prerequisite to writing programs.
Facilities Provided	High-level languages do not provide various facilities at the hardware level.	Low-level languages are very close to the hardware. They help in writing various programs at the hardware level.
Ease of Modification	The process of modifying programs is very difficult with high-level programs. It is because every single statement in it may execute a bunch of instructions.	The process of modifying programs is very easy in low-level programs. Here, it can directly map the statements to the processor instructions.
Examples	Some examples of high-level languages include Perl, BASIC, COBOL, Pascal, Ruby, etc.	Some examples of low-level languages include the Machine language and Assembly language.

Ans to the Que No 2 (A)

Graphics 7 inches * 5 inches with 600dpi. The amount of memory required to store the graphic:

To calculate the amount of memory required to store a graphic, you first need to find out how many pixels the graphic contains.

Given that the graphic is 7 inches by 5 inches with a resolution of 600 dots per inch (dpi), you can calculate the number of pixels in each dimension:

Width in pixels = 7 inches * 600 dpi = 4200 pixels
Height in pixels = 5 inches * 600 dpi = 3000 pixels

Now,
to find out the total number of pixels in the graphic, you multiply the width by the height:

Total pixels = Width in pixels * Height in pixels
= 4200 pixels * 3000 pixels = 12,600,000 pixels

Since,

each pixel stores color information, you also need to consider the color depth. Let's assume the graphic is using 24 bits per pixel (which is common for images),

where each pixel is represented by 3 bytes (1 byte for red, 1 byte for green, and 1 byte for blue).

Memory required = Total pixels * Color depth
= 12,600,000 pixels * 24 bits/pixel = 302,400,000 bits

To convert bits to bytes, divide by 8:

Memory required in bytes = 302,400,000 bits / 8 = 37,800,000 bytes

To convert bytes to megabytes (MB), divide by 1024*1024:

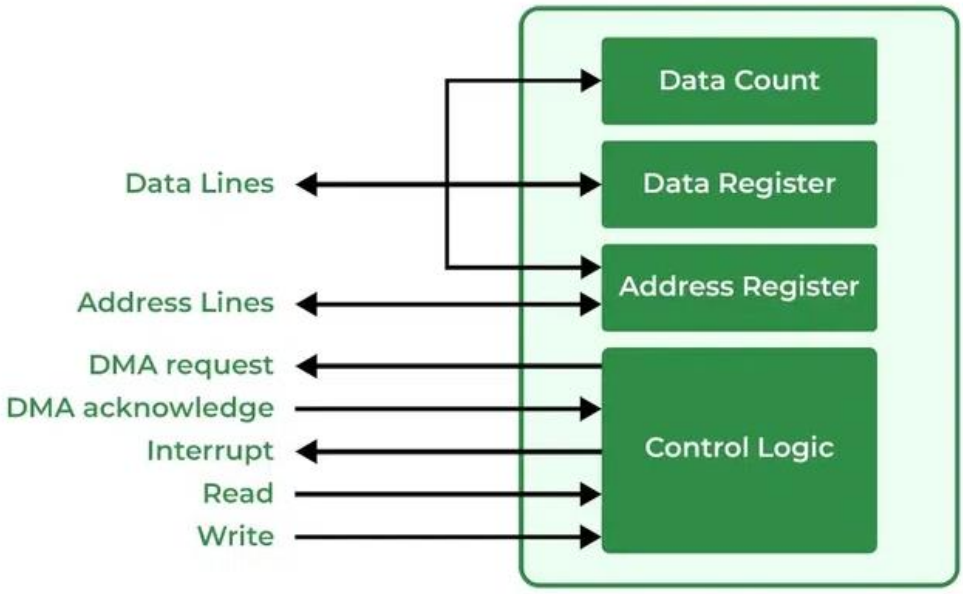
Memory required in MB ≈ 37,800,000 bytes / (1024 * 1024) ≈ 36.06 MB

So, approximately 36.06 megabytes of memory would be required to store the graphic.

Ans to the Que No 2 (A)

DMA Controllers:

DMA Controller is a type of control unit that works as an interface for the data bus and the I/O Devices. As mentioned, DMA Controller has the work of transferring the data without the intervention of the processors, processors can control the data transfer. DMA Controller also contains an address unit, which generates the address and selects an I/O device for the transfer of data. Here we are showing the block diagram of the DMA Controller.



Ans to the Que No 2 (B)

Operand

Computer instruction is a binary code that determines the micro-operations in a sequence for a computer. They are saved in the memory along with the information. Each computer has its specific group of instructions. They can be categorized into two elements as Operation codes (Opcodes) and Address. Opcodes specify the operation for specific instructions, and an address determines the registers or the areas used for that operation.

Operands are definite elements of computer instruction that show what information is to be operated on. The most important general categories of data are

1. Addresses
2. Numbers
3. Characters
4. Logical data

Functional representation of ALU:

The Arithmetic Logic Unit (ALU) is a crucial component of a CPU responsible for performing arithmetic and logical operations on data. Representing the ALU functionally involves describing its inputs, outputs, and the operations it can perform. Here's a simplified functional representation:

Inputs:
Operand A (n bits)**:** The first operand for arithmetic/logical operations.
Operand B (n bits)**:** The second operand for arithmetic/logical operations.
Control Signals (various)**:** Signals that determine which operation the ALU performs.

Outputs:
Result (n bits)**:** The output of the arithmetic or logical operation.
Flags (various)**:** Flags indicating conditions such as zero, carry, overflow, etc., resulting from the operation.

Functional Description:

The ALU takes two input operands (Operand A and Operand B) and a set of control signals that specify the operation to be performed. The control signals typically include:

Operation Select**:** Determines the type of operation to be performed (e.g., addition, subtraction, AND, OR, etc.).
Carry In**:** Indicates whether there is a carry from a previous operation, relevant for multi-precision arithmetic.
Shift Amount**:** Specifies the amount by which to shift the operands (for shift operations).
Based on the control signals, the ALU performs the specified operation on the input operands. The result is then produced as the output, along with any relevant flags indicating the status of the operation (e.g., overflow, carry, zero).

Example Operations:

Addition**:** Operand A + Operand B
Subtraction**:** Operand A - Operand B
AND, OR, XOR: Bitwise logical operations between Operand A and Operand B.
Shifts**:** Left shift, right shift of Operand A or Operand B by a specified number of bits.
Comparison**:** Operand A compared to Operand B, setting flags such as zero, sign, and carry accordingly.

Control Signal Examples:

Operation Select**:**
0000: Addition
0001: Subtraction
0010: Bitwise AND
0011: Bitwise OR
...

Carry In: 0 or 1
Shift Amount: Integer value indicating the number of bits to shift.
This functional representation provides a high-level overview of how an ALU operates without delving into the specific circuitry or implementation details.

Ans to the Que No 2 (C)

Convert 9G.AE216 to binary number:

To convert a hexadecimal number like 9G.AE216 into binary, you first need to convert each hexadecimal digit into its binary representation.

Here's the conversion:

9 -> 1001
G -> 0111 (since G represents 6 in hexadecimal)
. -> (decimal point, ignored in binary conversion)
A -> 1010
E -> 1110
2 -> 0010
1 -> 0001
6 -> 0110

So, 9G.AE216 in binary would be:
1001 0111.1010 1110 0010 0001 0110

- END -