CAO WINTER 18

08 March 2023 02:46 PM

Searched,Sorted & Merged By @official_stranger

Q1 a) Describe the structural overview of computer

ANSWER:-

A computer consists of several components, both hardware and software, that work together to process information. At a high level, the structural overview of a computer can be divided into four main components:

1. Input devices: These are the devices that allow the user to input data and commands into the computer. Examples of input devices include keyboards, mice, touchscreens, and scanners.
2. Output devices: These are the devices that display or output the processed information to the user. Examples of output devices include monitors, printers, and speakers.
3. Central Processing Unit (CPU): The CPU is the "brain" of the computer, responsible for performing calculations and executing instructions. It is composed of several components, including the control unit, the arithmetic logic unit (ALU), and registers.
4. Memory: Memory refers to the storage components that hold data and instructions that the CPU uses to perform its operations. There are two types of memory: primary memory (also known as RAM) and secondary memory (such as hard disk drives, solid-state drives, and flash memory).

Together, these four components work together to process information and execute programs on a computer

b) Define stored program concept and Explain Von Neumann' s Architecture with diagram.

ANSWER:-

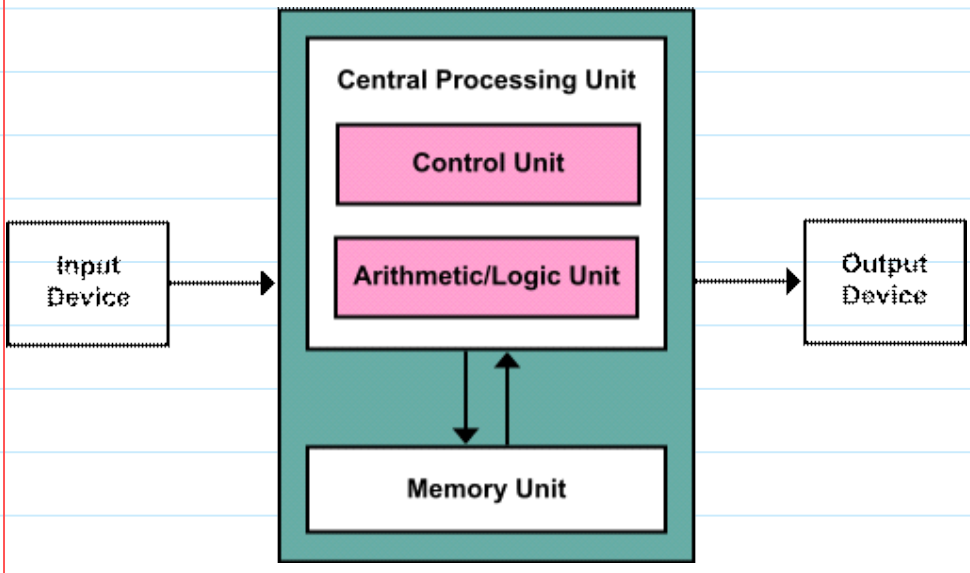
Stored program concept refers to the idea that instructions and data are stored in the same memory, and the CPU can retrieve and execute these instructions in a sequential manner. This concept allows computers to be programmed to perform different tasks by simply changing the instructions stored in memory, making them more versatile and flexible.

Von Neumann's architecture, also known as the Von Neumann model or the Von Neumann bottleneck, is a theoretical framework for building digital computers based on the stored program concept. It was proposed by John Von Neumann in the late 1940s and has since become the basis for most modern computers.

Von Neumann's architecture consists of five main components:

5. Input/output devices: These devices allow the user to input data and commands into the computer and receive output from it.
6. Memory: The memory stores both instructions and data in the same location, allowing the CPU to retrieve and execute them as needed.
7. Arithmetic Logic Unit (ALU): The ALU performs arithmetic and logical operations on data, such as addition, subtraction, and comparison.
8. Control Unit: The control unit fetches instructions from memory and directs the operations of the CPU and other components based on these instructions.
9. Central Processing Unit (CPU): The CPU is responsible for executing instructions

and performing calculations.
Here is a diagram of Von Neumann's architecture:



In this architecture, the CPU fetches instructions from memory, one at a time, and executes them in sequence. The control unit manages this process by decoding the instructions and directing the ALU to perform the required operations. The input/output devices allow the user to interact with the computer and receive output from it. The memory stores both instructions and data, allowing the computer to be programmed to perform different tasks.

Q2 a) List assembler directives? Assuming any assembler: give the necessary directives required for any program.

ANSWER:-

Assembler directives are special instructions that are processed by the assembler, but do not generate machine instructions. Instead, they provide information to the assembler about how to assemble the program, such as where to place code or data in memory, or how to generate symbol table information. The exact set of assembler directives may vary depending on the specific assembler being used, but some common directives include:

10. ORG (origin): This directive specifies the starting address of the program in memory.
11. EQU (equivalent): This directive assigns a value to a symbol or label.
12. DB (define byte): This directive reserves memory space for one or more bytes of data.
13. DW (define word): This directive reserves memory space for one or more 2-byte words of data.
14. DS (define storage): This directive reserves memory space for a specified number of bytes.
15. END (end of program): This directive marks the end of the program.
16. INCLUDE: This directive includes the contents of another source file in the current assembly.
17. GLOBAL: This directive declares a symbol to be visible to other modules or programs.
18. SECTION: This directive defines a section of code or data with a specific set of attributes, such as read-only or execute-only.

To write any program in assembly language, the necessary directives will depend on the specific requirements of the program. However, at a minimum, the program will need directives to specify the starting address of the program, reserve memory

space for data and code, and mark the end of the program.

b) Explain in detail different types of addressing modes

ANSWER:-

Addressing modes are a set of rules or techniques used in computer architecture to determine how the processor or CPU accesses data operands. Different types of addressing modes are used to specify the location or address of the operands, allowing the processor to perform operations on them. Here are some of the most common addressing modes used in computer architecture:

19. Immediate Addressing Mode: In this mode, the operand is part of the instruction itself. This mode is used when the data to be operated on is a constant value. For example, the instruction "MOV AX, #50" would move the constant value 50 into the register AX.
20. Register Addressing Mode: In this mode, the operand is in one of the processor's internal registers. The operand is accessed directly from the register, without any memory access. For example, the instruction "ADD AX, BX" would add the value in register BX to the value in register AX.
21. Direct Addressing Mode: In this mode, the operand is accessed directly from memory using its address. The address is part of the instruction itself, and the processor fetches the operand from memory. For example, the instruction "MOV AX, [2000H]" would move the value stored at memory address 2000H into the register AX.
22. Indirect Addressing Mode: In this mode, the operand is located at an address specified by a register or memory location. The address of the operand is not part of the instruction, but is stored in a register or memory location. For example, the instruction "MOV AX, [BX]" would move the value stored at the memory location pointed to by the register BX into the register AX.
23. Indexed Addressing Mode: In this mode, an index register is used to point to the memory location of the operand. The index register contains an offset value that is added to the base address of the operand to calculate the effective address. For example, the instruction "MOV AX, [SI+200H]" would move the value stored at the memory location pointed to by the sum of the SI register and the constant value 200H into the register AX.
24. Relative Addressing Mode: In this mode, the operand is accessed relative to the current instruction pointer or program counter. The address of the operand is calculated as an offset from the current instruction pointer or program counter. For example, the instruction "JMP LABEL" would jump to the instruction labeled LABEL, which is a fixed offset from the current instruction pointer.
25. Base Addressing Mode: In this mode, the effective address of the operand is calculated as a sum of a base address and an offset. The base address is stored in a register, and the offset is a constant value or a value stored in another register. For example, the instruction "MOV AX, [BX+SI+200H]" would move the value stored at the memory location pointed to by the sum of the BX and SI registers and the constant value 200H into the register AX.

These are some of the most common addressing modes used in computer architecture. Different processors may have additional addressing modes or variations on these modes, but these basic modes cover most use cases. Understanding addressing modes is important for writing efficient and effective assembly language programs.

Q3 a) Convert $(100.125)_{10}$ in IEEE-754 single precision floating point representation.

ANSWER:-

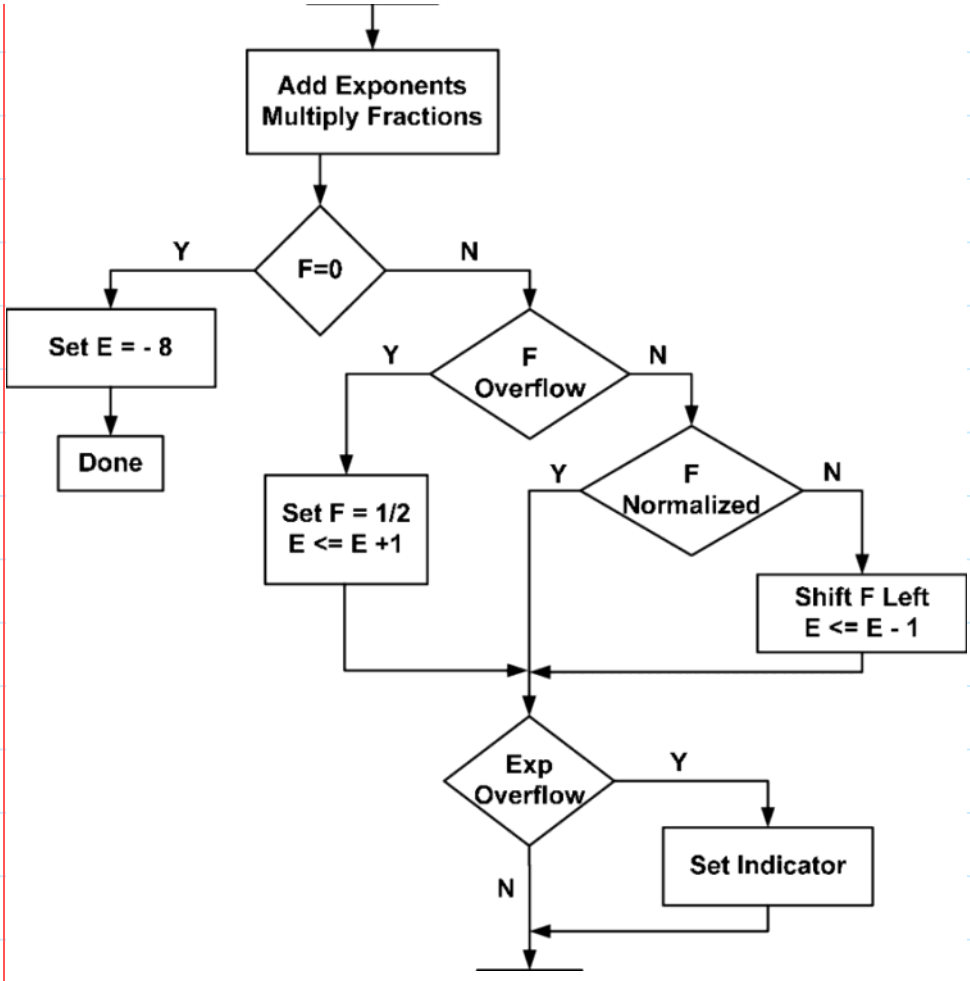
To convert the number (100.125) in base 10 to IEEE-754 single precision floating point representation, we need to first convert it to binary using logarithmic conversion. The steps are as follows:

26. Separate the integer and fractional parts of the number:
- Integer part: 100
- Fractional part: 0.125
27. Convert the integer part to binary:
- 100 = 1100100
28. Convert the fractional part to binary using logarithmic conversion:
- Convert the fractional part to scientific notation: 0.125 = 1.25 x 10^-1
- Compute the binary representation of the exponent:
- log2(10^-1) = -log2(10) = -3.321928094887362
- Round the exponent to the nearest integer: -3
- Add the bias to the exponent: -3 + 127 = 124
- Convert the exponent to binary: 1111100
- Compute the binary representation of the mantissa:
- Multiply the fractional part by 2 until the fractional part becomes 0 or until the desired number of bits for the mantissa is reached:
- 1.25 x 2 = 2.5, integer part is 1, fractional part is 0.5
- 0.5 x 2 = 1.0, integer part is 0, fractional part is 1.0
- The binary representation of the mantissa is 00100000000000000000000.
29. Combine the sign bit, exponent, and mantissa to form the IEEE-754 single precision floating point representation:
- The number is positive, so the sign bit is 0.
- The exponent is 1111100.
- The mantissa is 1100100.00100000000000000000000.
- Combine these bits in the following order: sign bit, exponent, mantissa.
- The final 32-bit representation is: 01000010111110010001000010010000

Therefore, the IEEE-754 single precision floating point representation of (100.125) LOG10 is 01000010111110010001000010010000.

b) Sketch and explain flowchart for multiplication of floating point numbers.

ANSWER:-



A flowchart for the multiplication of floating point numbers typically follows these steps:

30. Convert the two floating point numbers to their binary representations. The binary representations should include a sign bit, an exponent, and a mantissa.
31. Multiply the mantissas of the two numbers.
32. Add the exponents of the two numbers.
33. Normalize the result of the multiplication and exponent addition.
34. Round the result to the desired precision.
35. Convert the result back to the desired format, if necessary.

Here is a more detailed flowchart:

36. Input the two floating point numbers to be multiplied.
37. Convert the two numbers to their binary representations, which should include a sign bit, an exponent, and a mantissa.
38. Multiply the mantissas of the two numbers.
39. Add the exponents of the two numbers.
40. Check if the result is zero or denormalized. If so, skip to step 10.
41. Normalize the result of the multiplication and exponent addition.
42. Check if the result overflows or underflows. If so, return an error or adjust the exponent and mantissa to the maximum or minimum representable values.
43. Round the result to the desired precision, using a rounding mode such as round to nearest, round to zero, round up, or round down.
44. Convert the rounded result back to the desired format, if necessary.
45. Output the final result.

Note that the exact details of the flowchart may depend on the specific format of the floating point numbers and the desired precision and rounding mode. Also, some optimizations can be applied, such as merging steps or using special algorithms for certain cases, to improve the performance or accuracy of the multiplication.

Q4 a) Encode the data 1101 in even parity by using Hamming Code.

ANSWER:-

To encode the data 1101 using Hamming Code with even parity, we can follow these steps:
Step 1: Determine the number of parity bits needed To determine the number of parity bits needed for our data, we need to find the smallest value of r (the number of parity bits) such that 2^r >= m + r + 1, where m is the number of data bits. In this case, m = 4, so we need to find the smallest value of r such that 2^r >= 4 + r + 1.
2^r >= 6 + r
We can try values of r until we find the smallest value that satisfies this inequality:

- r = 1: 2^1 = 2 < 6 + 1 = 7 (not enough)
- r = 2: 2^2 = 4 < 6 + 2 = 8 (not enough)
- r = 3: 2^3 = 8 >= 6 + 3 = 9 (enough)

Therefore, we need 3 parity bits.
Step 2: Determine the positions of the parity bits The positions of the parity bits in the encoded data are the powers of 2. In this case, we have 7 bits (4 data bits and 3 parity bits), so the positions are:

- P1: bit 1 (2^0)
- P2: bit 2 (2^1)
- P3: bit 4 (2^2)

Step 3: Insert the data bits and calculate the parity bits We can now insert the data bits into the encoded data and calculate the parity bits.
The encoded data will have the following layout: P1 D1 P2 D2 D3 P3 D4
We can fill in the data bits: _ _ _ 1 1 0 1
Now we can calculate the parity bits:

- P1 (bit 1): This covers all bits that have a "1" in their binary representation at bit position 1 (i.e., bits 1, 3, 5, 7). We can calculate it by XOR-ing those bits together: P1 = 1 XOR 1 XOR 1 = 1.
- P2 (bit 2): This covers all bits that have a "1" in their binary representation at bit position 2 (i.e., bits 2, 3, 6, 7). We can calculate it by XOR-ing those bits together: P2 = 1 XOR 0 XOR 1 = 0.
- P3 (bit 4): This covers all bits that have a "1" in their binary representation at bit position 4 (i.e., bits 4, 5, 6, 7). We can calculate it by XOR-ing those bits together: P3 = 0 XOR 1 XOR 0 = 1.

We can now fill in the parity bits: 1 _ 0 1 1 0 1
Therefore, the Hamming Code with even parity for the data 1101 is 1011101.

b) Elaborate various types of ROM: Magnetic as well as optical

ANSWER:-

ROM (Read-Only Memory) is a type of computer memory that stores data permanently, meaning the data cannot be modified or erased once it has been programmed. There are several types of ROM, including magnetic and optical ROM.

46. Magnetic ROM: Magnetic ROM is a type of read-only memory that stores data using magnetic fields. It is also known as ferrite-core memory. The memory cells are made up of small ferrite cores that can be magnetized in two directions to represent binary data. The magnetic fields are created using a magnetic write head and read using a magnetic read head. Magnetic ROM was commonly used in early computer systems but has largely been replaced by more advanced memory technologies.
47. PROM (Programmable Read-Only Memory): PROM is a type of ROM that allows data to be programmed into the memory once by the manufacturer or the user. The

data is then permanently stored and cannot be changed. PROM chips are programmed using a device called a programmer, which applies high voltage signals to specific memory cells to program them. PROMs are commonly used in embedded systems and other applications where permanent data storage is required.

48. EPROM (Erasable Programmable Read-Only Memory): EPROM is a type of ROM that can be programmed and erased multiple times. EPROM chips are programmed using a programmer and can be erased using ultraviolet light. The EPROM chip has a small window on top that allows ultraviolet light to penetrate and erase the memory cells. EPROMs are used in applications where the program code may need to be updated or changed frequently.
49. EEPROM (Electrically Erasable Programmable Read-Only Memory): EEPROM is a type of ROM that can be programmed and erased electronically. EEPROM chips can be programmed and erased while they are still in the circuit, making them a popular choice for many embedded systems. EEPROMs are used in applications where the program code may need to be updated or changed frequently, but the system cannot be taken offline for programming.
50. Optical ROM: Optical ROM is a type of read-only memory that uses lasers to read data stored on a disk or other optical media. The data is stored as tiny pits on the surface of the disk, which can be read using a laser beam. Optical ROM is commonly used in CD-ROMs, DVD-ROMs, and other types of optical storage media.

In conclusion, these are the different types of ROM, including magnetic and optical ROM. Each type of ROM has its own advantages and disadvantages, making it suitable for different applications depending on the specific needs of the system.

Q5 a) Discuss Micro operations to execute an instruction MOV RI,R2

ANSWER:-

The instruction MOV RI, R2 is a typical example of a data transfer operation in a computer system. It transfers the contents of register R2 to register RI. To execute this instruction, the processor would need to perform a sequence of micro-operations.

Here is a possible sequence of micro-operations that could be executed to carry out the MOV RI, R2 instruction:

51. Fetch the instruction from memory and place it into the instruction register (IR).
52. Decode the instruction to determine the operands and the operation to be performed. In this case, the operands are RI and R2, and the operation is a data transfer.
53. Fetch the contents of R2 and place them into a temporary register (Temp).
54. Copy the contents of Temp into register RI.
55. Update the program counter to point to the next instruction.

These micro-operations would be performed by the control unit of the processor, which is responsible for coordinating the various components of the processor to execute instructions. The fetch and decode operations would typically involve accessing the instruction cache and the register file, while the data transfer would involve the ALU and the register file. The program counter update would involve incrementing the value stored in the program counter register.

It's worth noting that different processor architectures may implement the MOV instruction in different ways, and may therefore require different micro-operations to execute it. However, the basic idea of fetching operands, performing a data transfer, and updating the program counter is common to most processor designs.

b) Explain Wilki 's design of Micro programmed Control Unit.

ANSWER:-

Wilkes' design of a microprogrammed control unit (MCU) is a technique used to simplify the design of the control unit for a computer's central processing unit (CPU). Rather than creating a complex and inflexible hardwired control unit, Wilkes proposed a design where the control unit's functionality is defined by a microprogram stored in memory.
The microprogram is essentially a set of instructions that define how the control unit should operate. Each microinstruction corresponds to a specific operation of the CPU, such as fetching an instruction from memory or performing an arithmetic operation. These microinstructions are executed sequentially by the control unit to carry out the desired operation.
The advantage of this design is that it allows for greater flexibility in the design of the control unit. Instead of creating a hardwired control unit that is specific to a particular CPU architecture, a microprogrammed control unit can be easily reprogrammed to support different instruction sets or CPU architectures. This makes it easier to adapt the CPU to changing computing needs and to improve performance over time.
Furthermore, the use of a microprogrammed control unit allows for easier debugging and testing of the control unit. Since the microprogram is stored in memory, it can be easily modified and updated as needed. This means that any errors or bugs in the control unit can be quickly identified and corrected without the need to rewire the control unit.
Overall, Wilkes' design of a microprogrammed control unit represents an important innovation in computer architecture, allowing for greater flexibility, adaptability, and ease of use in the design of CPU control units.

Q6 a) Explain Instruction Pipelining.

ANSWER:-

Instruction pipelining is a technique used in computer architecture to improve the efficiency of the CPU's execution of instructions. It involves breaking down the execution of an instruction into a series of smaller steps, and then processing multiple instructions simultaneously by overlapping these steps.
In a pipelined CPU, the execution of an instruction is divided into several stages, with each stage handling a specific operation. These stages typically include instruction fetch, instruction decode, operand fetch, execute, and write-back. Each stage of the pipeline is performed by a dedicated hardware unit, allowing different instructions to be processed simultaneously.
As an instruction is executed, it moves through the pipeline, with each stage processing a different instruction at the same time. For example, while the first instruction is being fetched, the second instruction is being decoded, and the third instruction is being executed.
The benefits of instruction pipelining include increased instruction throughput and faster overall execution of instructions. Since multiple instructions can be processed simultaneously, the overall execution time for a set of instructions is reduced. This makes the CPU more efficient and able to handle more instructions per unit of time.
However, there are also some challenges to implementing instruction pipelining. One of the main challenges is dealing with dependencies between instructions, where the output of one instruction is required as an input for the next instruction. These dependencies can cause delays in the pipeline, reducing the benefits of pipelining.
Overall, instruction pipelining is a key technique used in modern CPU design to improve performance and efficiency, and it has become a standard feature of modern processors.

b) Discuss Interrupt Driven I/O. Compare it with Programmed VO and explain types of Interrupts

ANSWER:-

Interrupt-driven I/O is a method of input/output (I/O) where the processor receives input from a device or sends output to a device without wasting processing time. The interrupt-driven I/O system works on the principle of interrupt signals generated by the input/output device.

Whenever the input/output device has data to transfer or needs attention from the CPU, it sends an interrupt signal to the processor. The processor then interrupts its current operation and handles the I/O request.

The main advantage of interrupt-driven I/O is that it allows the processor to perform other tasks while waiting for input/output operations to complete. This is in contrast to programmed I/O where the processor waits for I/O operations to complete before moving on to the next task. Interrupt-driven I/O is therefore more efficient and allows the processor to perform more tasks in a shorter time.

In contrast, programmed I/O requires the CPU to execute the instructions of the program to transfer data between memory and I/O device. Programmed I/O is suitable for simple systems where the I/O devices operate at low speeds and require minimal CPU time.

Types of Interrupts:

56. Hardware Interrupts: A hardware interrupt is triggered by an external device or peripheral, such as a keyboard or mouse, signaling the processor that it needs attention. The processor then temporarily suspends its current task to respond to the interrupt request.
57. Software Interrupts: Software interrupts are triggered by the program itself when it needs to communicate with the operating system or request a service from the kernel. Software interrupts are used for system calls, which are requests made by programs for services provided by the operating system.
58. Internal Interrupts: Internal interrupts are generated by the processor itself when it encounters an error, such as a divide-by-zero error or memory access violation. The processor generates an internal interrupt to alert the operating system or software that an error has occurred.
59. External Interrupts: External interrupts are generated by external devices such as timers or clocks. These devices generate an interrupt signal at specified intervals to signal the processor that it needs to perform a specific task, such as updating the system clock or triggering an alarm.

In summary, interrupt-driven I/O is a more efficient method of input/output compared to programmed I/O. Interrupt-driven I/O utilizes interrupt signals generated by the input/output device to notify the processor that it needs attention. There are different types of interrupts, including hardware interrupts, software interrupts, internal interrupts, and external interrupts, each with a specific purpose.