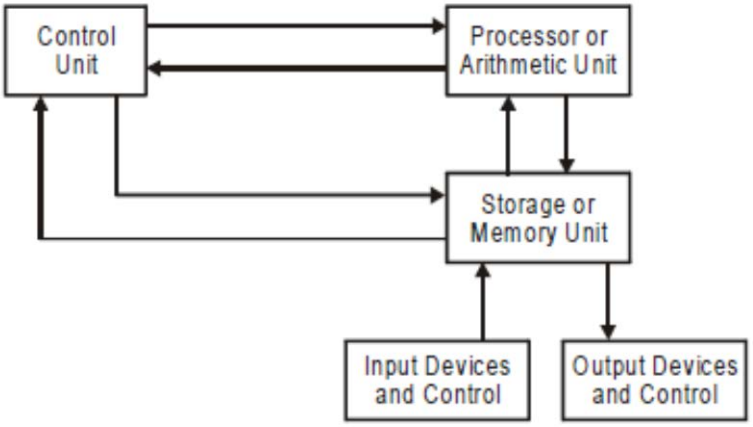
Q2 (a) With the help of diagram explain the different functional units of a Computer?

Answer: A computer consists of five functionally independent units, namely: input, memory, arithmetic and logic (ALU) unit, output and control units as shown in the figure below. The input unit accepts coded information from human operators, form electromechanical devices such as keyboard, or from other computers over digital communication lines. The information received is either stored in computer's memory for later reference or immediately used by the arithmetic and logic circuitry to perform desired operations. The processing steps are determined by a program stored in the memory. Finally, the results are sent back to the outside world through the output unit. All of these actions coordinated by the control unit.



Input Unit**:** Computers accept coded information through input units, which read the data. The most well-known input device is the 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 joysticks, trackballs, and mouse. These are often used as graphics input devices in conjunction with displays.

Memory Unit: The function of the memory unit is to store programs and data. There are two classes of storage, called primary and secondary.
Primary storage is a fast memory that operates at electronic speeds. Programs must be stored in the memory while they are being executed. 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 groups of fixed size called words. The memory is organized so that the contents of one word, containing n bits, can be stored or retrieved in one basic operation. To provide easy access to any word in the memory, a distinct address is associated with each word location. Addresses are numbers that identify successive locations. A given word is accessed by specifying its address and issuing a control command that starts the storage or retrieval process.
Another memory, called secondary storage is used when large amounts of data and many programs have to be stored, particularly for information that is accessed infrequently. A wide selection of secondary storage device is available, including magnetic disks and tapes and optical disks.

Arithmetic And Logic Unit: Most computer operations are executed in the arithmetic and logic unit (ALU) of the processor. 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. Any other arithmetic or logic operations, such as multiplication or comparison of numbers, are initiated by bringing the required operands into the processor, where the operation is performed by the ALU. When operands are brought into the processor, they are stored in high-speed storage elements called registers. Each register can store on word of data. Access times to register are somewhat faster than access times to the fastest cache memory.
The control and the arithmetic and logic units are many times faster than other devices connected to a computer system. This enables a single processor to control a number of external devices such as keyboards, displays, disks, and mechanically controllers.

Output Unit: The output unit is the counterpart of input unit. Its function is to send processed results to the outside world. The most familiar output unit is printer. Some units, such as graphics displays, provide both an output function and an input function. The dual role of such units is the reason for using the single name I/O unit in many cases.

Control Unit: The memory, arithmetic and logic, and input and output units store and process information and perform input and output operations. The operation of these units must be coordinated by the control unit. The control

unit is effectively the nerve center that sends control signals to other units and senses their states.

I/O transfers, consisting of input and output operations, are controlled by the instructions of I/O programs that identify the devices involved and the information to be transferred. However, the actual timing signals that govern the transfers are generated by the control units. Data transfers between the processor and the memory are also controlled by the control unit through timing signals. It is reasonable to think of a control unit as a well-defined, physically separate unit that interacts with other parts of the machine. Much of the control circuitry is physically distributed throughout the machine. A large set of control lines carries the signals used for timing and synchronization of events in all units.

(b) Represent (-23) in:

(i) Sign-and-magnitude representation
(ii) 1's complement representation
(iii) 2's complement representation

Answer: Binary equivalent of $(+23)_{10} = (010111)_2$

(i) In sign-and magnitude representation
(110111)
(ii) 1's complement representation
(101000)
(iii) 2's complement representation
(101001)

(c) Explain the use of following registers:

Answer:

(i) PC: The program counter (PC) keeps track of the execution of a program. It contains the memory address of the next instruction to be fetched and executed. During the execution of an instruction, the contents of the PC are updated to correspond to the address of the next instruction to be executed. Hence, it points to the next instruction that is to be fetched from the memory.

(ii) MAR: The memory address register (MAR) holds the address of the location to be accessed. Execution of the program starts when the PC is set to point to the first instruction of the program. The contents of the PC are transferred to the MAR and a Read control signal is sent to the memory.

(iii) IR: The instruction register (IR) holds the instruction that is currently being executed. Its output is available to the control circuits which generate the timing signals that control the various processing elements involved in executing the instruction.

(iv) MDR: The memory data register (MDR) contains the data to be written into or read out of the addressed location. When the operand has been read from the memory into the MDR, it is transferred from MDR to the ALU.

Q3 (a) Explain the process called subroutine nesting?

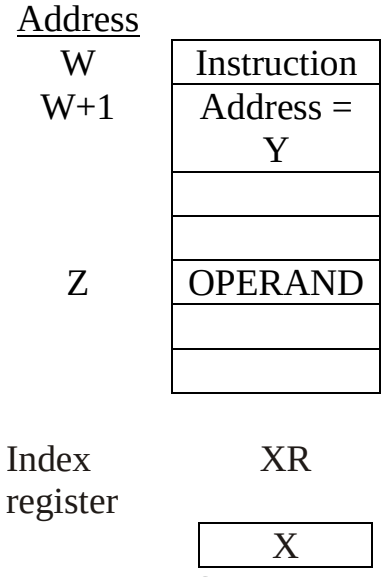
Answer: A common programming practice, called subroutine nesting, is to have one subroutine call another. In this case, the return address of the second call is also stored in the link register, destroying its previous contents. Hence, it is essential to save the contents of the link register in some other location before calling another subroutine. Otherwise, the return address of the first subroutine will be lost.

Subroutine nesting can be carried out to any depth. Eventually, the last subroutine called completes its computations and return to the subroutine that called it. The return address needed for this first return is the last one generated in the nested call sequence. That is return addresses are generated and used in a last-in-first-out order. Thus, the return addresses associated with subroutines calls should be pushed onto a stack. Many processors do this automatically as one of the operations performed by the call instruction. A particular register is designated as the stack pointer, SP to be used in this. The stack pointer points to a stack called the processor stack. The Call instruction pushes the contents of the PC onto the processor stack and then loads the subroutine address into the PC. The Return instruction pops the return address from the processor stack into the PC.

(b) A two word instruction is stored in memory at an address designated by symbol *W*. The address field of the instruction (stored at *W* + 1) is designated by the symbol *Y*. The operand used during the execution of the instruction is stored at an address symbolized by *Z*. An index register contains the value *X*. State how *Z* is calculated from the other addresses if the addressing mode of the instruction is

(i) Direct
(ii) Indirect
(iii) Relative
(iv) Indexed

Answer: According to question, the memory figure is shown in figure below:



Memory Content

(i) In direct addressing mode the address part of the instruction is the address of the operand.
Thus, Z = Y.

(ii) In indirect addressing mode the address part of the instruction gives the memory address, what is the effective address of the operand.
Thus, Z = M[Y]

(iii) In case of relative addressing mode, the effective address is the address part of the instruction (*Y*) plus the content of program counter. The instruction is stored at *W* with address part at *W* + 1. Thus, when instruction is fetched from memory the PC will increment and points to W + 2.

Thus, Z = Y + W + 2.

In indexed addressing mode, the effective address is the address part of instruction plus the content of index register.

Thus, Z = Y + X

(c) Write a program that can evaluate the expression

X = (A-B) * ((C – D * E) / F)

in a single-accumulator processor, Assume that the processor has Load, Store, Multiply, Add instructions, and that all values fit in the accumulator.

Answer:

LOAD	A	AC ← M[A]
SUB	B	AC ← AC – M[B]
STORE	X	M[X] ← AC
LOAD	D	AC ← M[D]
MUL	E	AC ← AC * M[E]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
SUB	T	AC ← AC – M[T]
DIV	F	AC ← AC / M[F]
MUL	X	AC ← AC * M[X]
STORE	X	M[X] ← AC

Q4(a) What are the main advantages of using Input / Output interface? Why interfacing is used in digital computers?

Answer: A hardware unit that plays an important role between the CPU and peripheral devices to supervise the input and output transfers is known as interface.

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

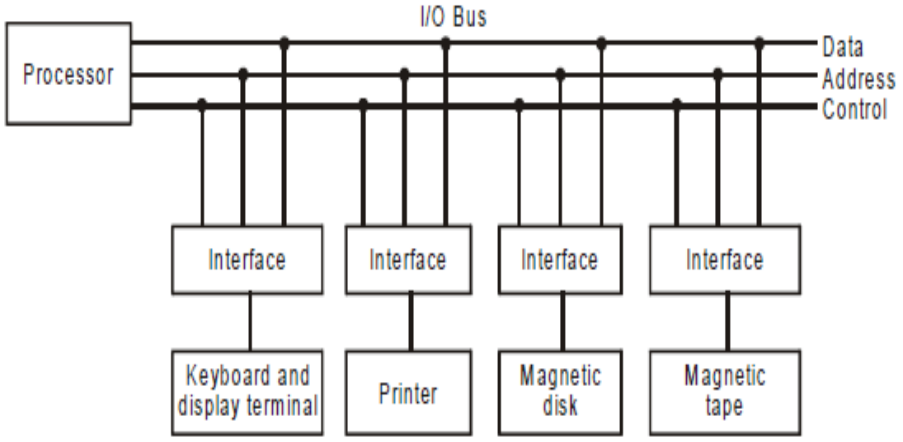
(1) Peripherals are electromechanical and electromagnetic devices and their manners of operations are different from the operation of the CPU and memory, which are electronic devices. Therefore, the conversion of signals is required.

(2) The data transfer rate of peripherals is usually slower than the transfer rate of the CPU and hence, synchronization mechanisms are needed.

(3) Data codes and formats in peripherals are different from the word format in the CPU and memory.

(4) The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware known as “interface units” between the CPU and peripherals to supervise and synchronize all input and output transfers. A typical communication link between the processor and several peripherals are shown in figure. The I/O bus consists of data lines, address lines and control lines.



Each peripheral device has its associated interface unit. Each interface unit decodes the address and control received from I/O bus, interprets them for the peripheral and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripherals and processor. Each peripheral has its own controller that operates the particular electromechanical device. For example, the printer controller controls the paper motion, the printing timing and the selection of printing characters. The I/O bus from the processor is attached to all peripherals interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the devices that it controls. All peripherals whose address does not correspond to the address in bus are disabled by their interfaces. At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface selected responds to the function code and proceeds to execute it. These function codes are referred to as I/O commands. There are four types of commands that the interface may receive. They are:

(i) Control Command: A control command is issued to activate the peripheral and to inform it what to do. The particular control command issued depends on the peripheral. For example, a magnetic tape unit may be instructed to back space the tape by one record, to rewind the tape or to start the tape moving in the forward direction.

(ii) Status: A status command is used to test various status conditions in the interface and the peripherals.

(iii) Output Data: A data output command causes the interface to respond by transferring data from the bus into one of its registers.

(iv) Input Data: The data input command is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register. The processor checks if the data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.

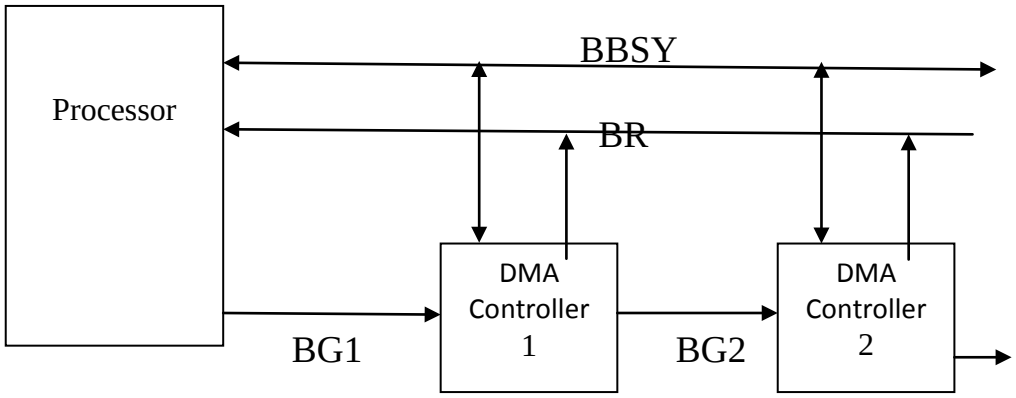
(b) Define Bus arbitration? Explain Centralized and Distributed arbitration process?

Answer: Bus Arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it. The selection of the bus master must take into account the needs of various devices by establishing a priority system for gaining access to the bus.
There are two approaches to bus arbitration: centralized and distributed.

In centralized arbitration a single bus arbiter performs the required arbitration. In distributed arbitration, all devices participate in selection of the next bus master.

Centralized Arbitration

The bus arbiter may be the processor or a separate unit connected to the bus. Figure below illustrates a basic arrangement in which the processor contains the bus arbitration circuitry. In this case, the processor is normally the bus master unless it grants bus mastership to one of the DMA controllers. A DMA controller indicates that it needs to become the bus master by activating the Bus-Request line, $\overline{BR}$. This is an open-drain line. The signal on the Bus-Request line is the logical OR of the bus requests from all the devices connected to it. When Bus-Request is activated, the processor activates the Bus-Grant signal, BG1, indicating to the DMA controllers that they may use the bus when it becomes free. This signal is connected to all DMA controllers using a daisy-chain arrangement. Thus, if DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to other devices. The current bus master indicates to all devices that it is using the bus by activating another open-collector line called Bus-Busy, $\overline{BBSY}$. Hence, after receiving the Bus-Grant signal, a DMA controller waits for Bus-Busy to become inactive, and then assumes mastership of the bus. At this time, it activates Bus-Busy to prevent other devices from using the bus at the same time.

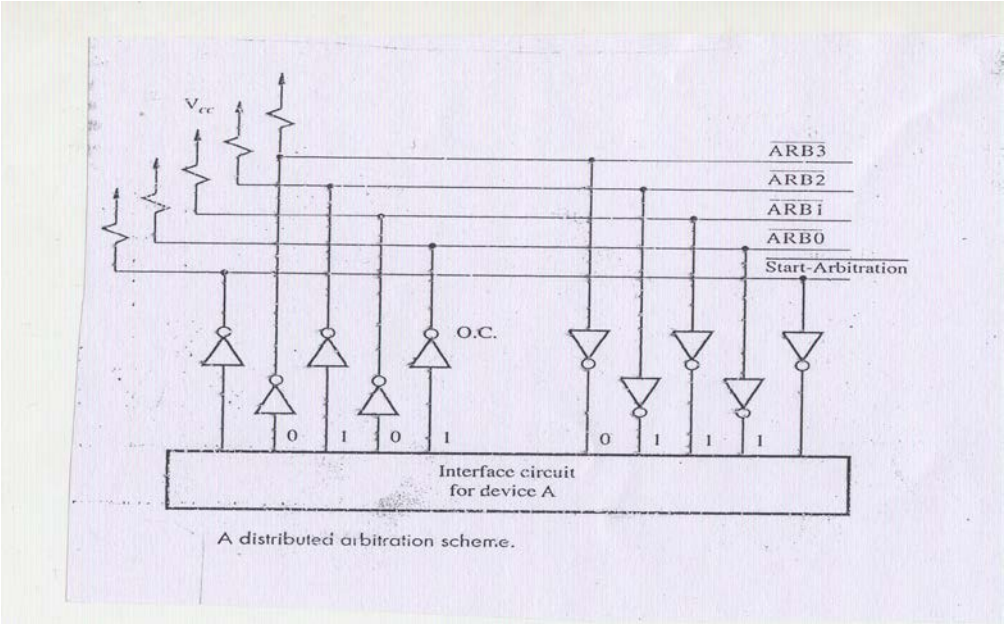


A simple arrangement for bus arbitration using daisy chain

Above figure shows one bus-request line and one bus-grant line forming a daisy chain. Several such pairs may be provided, in an arrangement similar to that used for multiple requests. This arrangement leads to considerable flexibility in determining the order in which requests from different devices are serviced. The arbiter circuit ensures that only one request is granted at any given time, according to a predefined priority scheme. Alternatively, a rotating priority scheme may be used to give all devices an equal chance of being serviced.

Distributed Arbitration

Distributed arbitration means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter. A simple method for distributed arbitration is illustrated in figure below. Each device on the bus is assigned a 4-bit identification number. When one or more devices request the bus, they assert the Start-Arbitration signal and place their 4-bit ID numbers on four open-collector lines, $\overline{ARB0}$ through $\overline{ARB3}$. A winner is selected as a result of the interaction among the signals transmitted over these lines by all contenders. The net outcome is that the code on the four lines represents the request that has the highest ID number.



A distributed arbitration scheme.

Decentralization arbitration has the advantage of offering higher reliability, because operation of the bus is not dependent on any single device.

Q5 (a) Briefly describe the Peripheral Component Interconnect (PCI) Bus Standards?

Answer: The PCI bus is a good example of a system bus that grew out of the need for standardization. It supports the functions found on a processor bus but in a standardized format that is independent of any particular processor. Devices connected to the PCI bus appear to the processor as if they were connected directly to the processor bus. They are assigned addresses in the memory address space of the processor.

The PCI follows a sequence of bus standards that were used primarily in IBM PCs. Early PCs used the 8-bit XT bus, whose signals closely mimicked those of Intel's 80x86 processors. Later, the 16-bit bus used on the PC AT computers became known as the ISA bus. Its extended 32-bit version is known as the EISA bus.

The PCI was developed as a low-cost bus that is truly processor-independent. An important feature that the PCI pioneered is a plug-and-play capability for connecting I/O devices. To connect a new device, the user simply connects the device interface board to the bus. The software takes care of the rest.

(b) What are the sequences of events take place when the processor sends a command to the SCSI controller?

Answer: The following sequence of events takes place when the processor sends a command to the SCSI controller:

- The SCSI controller, acting as an initiator, contends for control of the bus.
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
- The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
- The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
- The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.
- The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator, as before. At the end of this transfer, the logical connection between the two controllers is terminated.
- As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
- The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

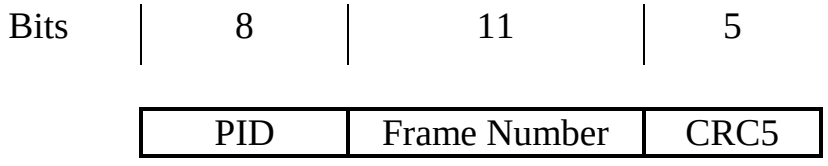
This scenario shows that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus.

(c) Explain how USB support Isochronous data?

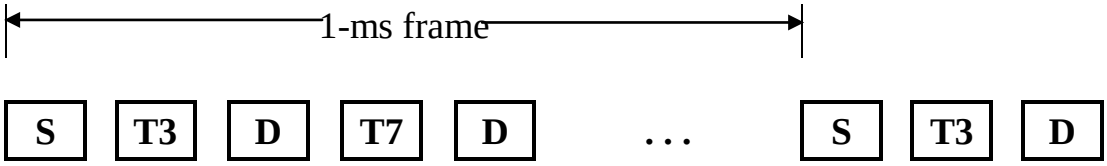
Answer: One of the key objectives of the USB is to support the transfer of isochronous data, such as sampled voice, in a simple manner. Devices that generate or receive isochronous data require a time reference to control the sampling process. To provide this reference transmission over the USB is divided into frames of equal length. A frame is 1 ms long for low – and full-speed data. The root hub generates a Start Of Frame control packet (SOF) precisely once every 1 ms to mark the beginning of a new frame.

The arrival of an SOF packet at any device constitutes a regular clock signal that the device can use for its purpose. To assist devices that may need longer periods of time, the SOF packet carries an 11-bit frame number, as shown in figure (a). Following each SOF packet, the host carries out input and output transfers for isochronous devices. This means that each device will have an opportunity for an input or output transfer once every 1 ms.

The main requirement for isochronous traffic is consistent timing. An occasional error can be tolerated. Hence there is no need to transmit packets that are lost or to send acknowledgements. Figure (b) shows the first two transmission following SOF. A control packet carrying device address 3 is followed by data for that device. This may be input or output data, depending on whether the control packet is an IN or OUT control packet. There is no acknowledgement packet. The next transmission sequence is for device 7.



(a) SOF Packet



S – Start-of-frame packet
Tn – Token packet, address = n
D – Data packet
A – ACK packet

(b) Frame example

As an example, the data packet for device 3 may contain 8 bytes of data. One such packet is sent in each frame, providing a 64-kilobits/s isochronous channel. Such a channel may be used for a voice connection. The transmission of 8 bytes of data requires a 3-byte token packet followed by an 11-byte data packet (including the PID and CRC fields), for a total of 132 bits. A minimum of three more bytes are needed for clock synchronization and to mark the end of packet sequence. At a speed of 12 megabits/s, this takes about 13 μ s. Clearly, there is a room in a frame to support several such devices. Isochronous data are allowed only on full-speed and high-speed links. For high-speed links, the SOF packet is repeated eight times at equal intervals within the 1-ms frame to create eight microframes if 125 μ s each.

Q6 (a) (i) How many 128 × 8 RAM chips are needed to provide a memory capacity of 2048 bytes?

Answer:

$$\text{Number of chips required} = \frac{2048 \times 8}{128 \times 8} = 16 \text{ chips}$$

(ii) How many lines of the address bus must be used to access 2048 bytes of memory? How many of these lines will be common to all chips?

Answer: Since total memory is of 2048 *i.e.* 2^{11} bytes. Hence, 11 lines of the address bus is required to access 2048 bytes.

Also, RAM chip is of 128 i.e. 2^7 bytes. So, 7 lines out of 11 will be common to all chips.

(iii) How many lines must be decoded for chip select? Specify the size of the decoder.

Answer: 4 lines must be decoded for selecting 16 chips. Thus, the size of decoder required is 4 × 16.

(b) A digital computer has a memory unit of 64K × 16 and a cache memory of 1K words. The cache uses direct mapping with a block size of four words.

(i) How many bits are there in the tag, index, block and words fields of the address format?
(ii) How many bits are there in each word of cache, and how are they divided into functions? Include a valid bit.
(iii) How many blocks can the cache accommodate?

Answer: Given, Main memory is of size 64K × 16 words
Therefore, 16 bits is required to address each word of main memory. Also each word is of 16 bits.

$M = 16$

Cache memory is of size 1 K words
Therefore, 10 bits is required to address each word of cache.

$N = 10$

Direct mapping is used and block size is of four words.

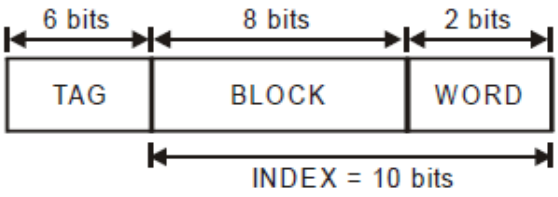
(i) Since cache memory address is of 10 bits
Therefore, Index will be of 10 bits.

Tag = $M - N$
= 16 – 10
= 6 bits

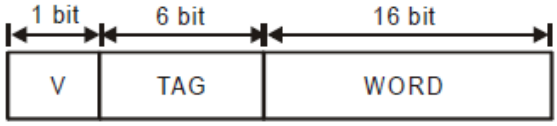
Cache consists of 1 K words and there are 4 words in 1 block

Therefore, Number of blocks = $\frac{1K}{4} = \frac{1024}{4}$
= 256 blocks
= 2^8 blocks

Thus, block will be of 8 bits.
Also, 4 words per block means 2^2 words per block, therefore word will be of 2 bits.
Thus, address format is



(ii) Each word of cache consists of 16-bit word from main memory along with the tag. A valid but will also be included into each cache word. Hence, the word format of cache is

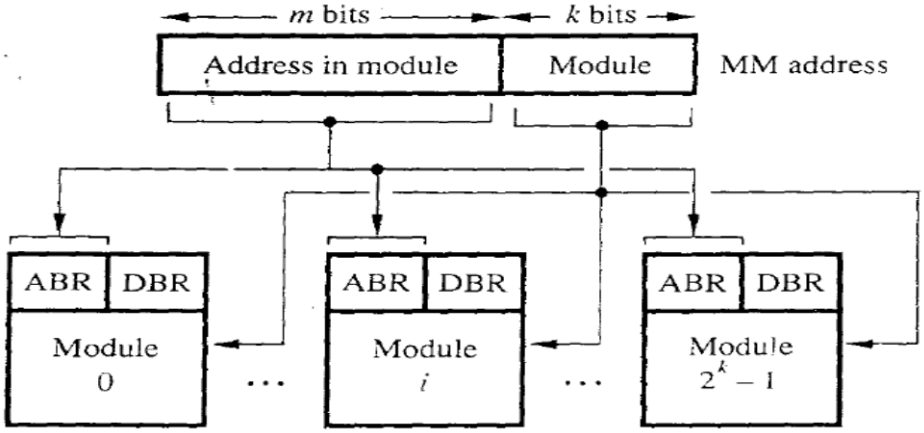


Thus, each word of cache is of 23 words.

(iii) Number of blocks = $\frac{1K}{4} = \frac{1024}{4} = 256\ blocks$
Thus, the cache can accommodate 256 blocks of 4 words each.

(c) What do you mean by Memory – interleaving?

Answer:



The more effective way to address the modules is shown in figure called *memory interleaving.* The low-order k bits of the memory address select a module, and the high-order m bits name a location within that module. In this way, consecutive addresses are located in successive modules. Thus, any component of the system that generates requests for access to consecutive memory locations can keep several modules busy at any one time. This results in both faster access to a block of data and higher average utilization of the memory system as a whole. To implement the interleaved structure, there must be 2^k modules otherwise; there will be gaps of nonexistent locations in the memory address space.

The effect of interleaving is substantial. Consider the time needed to transfer a block of data from the main memory to the cache when a read miss occurs. Suppose that a cache with 8-word blocks is used. On a read miss, the block that contains the desired word must be copied from the memory into the cache. Assume that the hardware has the following properties. It takes one clock cycle to send an address to the main memory. The memory is built with relatively slow DRAM chips that allow the first word to be accessed in 8 cycles, but subsequent words of the block are accessed in 4 clock cycles per word. Also, one clock cycle is needed to send one word to the cache.
If a single memory module is used, then the time needed to load the desired block into the cache is

$$1 + 8 + (7 \times 4) + 1 = 38 \text{ cycles}$$

Suppose now that the memory is constructed as four interleaved modules, using the scheme in figure. When the starting address of the block arrives at the memory, all four modules begin accessing the required data, using the high-order bits of the address. After 8 clock cycles, each modules has one word of data in its DBR. These words are transferred to the cache, one word at a time, during the next 4 clock cycles. During this time, the next word in each module is accessed. Then it takes another 4 cycles to transfer these words to the cache. Therefore, the total time needed to load the block from the interleaved memory is

$$1 + 8 + 4 + 4 = 17 \text{ cycles}$$

Thus, interleaving reduces the block transfer time by more than a factor of 2.

Q7 (a) Design a half adder as a 2 level AND OR circuit? Implement full adder circuit using two half adders.

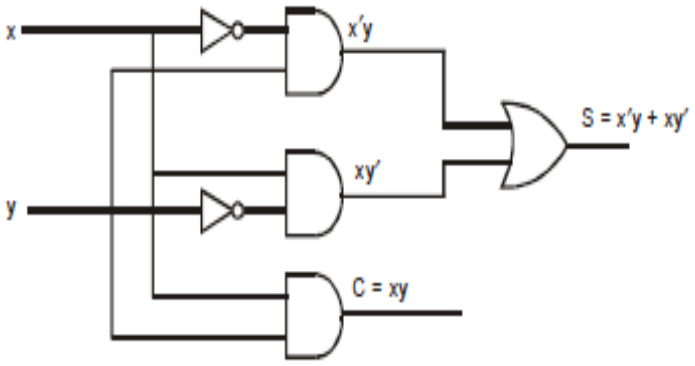
Answer: A combinational circuit that performs the addition of two bits is called a half-adder. The half-adder circuit needs two binary inputs and two binary outputs. The inputs variables are augends and addend bits and the output variables produces sum and carry. The truth-table for half-adder is shown in table below. The two inputs are x and y and the two outputs are S (for sum) and C (for carry).

x	y	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

The simplified functions for S and C outputs can be obtained from the table itself.

$$S = x'y + xy'$$
$$C = xy$$

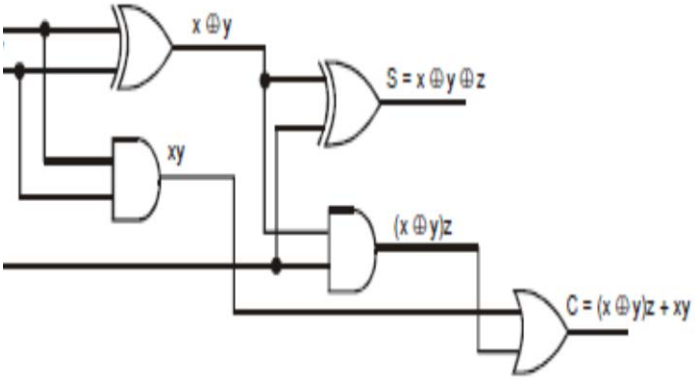
The logic circuit for this implementation is shown in figure below.



Half-Adder Circuit

The sum and carry function for full-adder is as below

$$S = x \oplus y \oplus z$$
$$C = (x \oplus y)z + xy$$



Full-adder circuit using two Half-adders

(b) Explain the purpose of Disk controller?

Answer: Operation of a disk drive is controlled by a disk controller circuit, which also provides an interface between the disk drive and the bus that that connects it to the rest of the computer system. The disk controller may be used to control more than one drive.
A disk controller that is connected directly to the processor system bus, or to an expansion bus such as PCI, contains a number of registers that can be read and written by the operating system. Thus, communication between the OS and the disk controller is achieved. The disk controller uses the DMA scheme to transfer data between the disk and the main memory. Actually, these transfers are from/to the data buffer, which is implemented as a part of disk controller module. The OS initiates the transfers by issuing Read and Write requests, which entail loading the controller's registers with the necessary addressing and control information typically:

- Main memory address – The address of the first main memory location of the block of words involved in the transfer.
- Disk address – The location of the sector containing the beginning of the desired block of words.
- Word count – The number of words in the block to be transferred.

The disk address issued by the OS is logical address. The corresponding physical address on the disk may be different.

(c) A certain magnetic disk has the following specification:

- Number of recording = 10
- Number of tracks/surface = 256
- Number of sectors/track = 32
- Number of bytes/sector = 128
- Disk rotation speed = 2400 rpm

Calculate the following:

(*i*) Disk capacity

Answer: Disk capacity = Number of surfaces × Number of tracks in each surface × Number of sectors in a track × Number of bytes in a sector.

$= 10 \times 256 \times 32 \times 128$
$= 10 \times 2^8 \times 2^5 \times 2^7$
$= 10 \times 2^{20}$
$= 10$ MB

(iii) Transfer rate

Answer: Capacity of one track = Number of sectors in a track × Number of bytes in a sector

$= 32 \times 128$
$= 2^5 \times 2^7 = 2^{12}$ bytes

Also given, disk rotation speed = 2400 rpm
Therefore, 2400 rotation in 1 min *i.e.* 60 sec.

1 rotation in $\frac{60}{2400}\text{sec} = \frac{1}{40}\text{sec}$

In 1/40 sec data transfer = 2^{12} bytes

In 1 sec, data transfer = 2^{12} x 40 bytes
$= 40 \times 4 \times 2^{10}$ bytes
= 160 K bytes

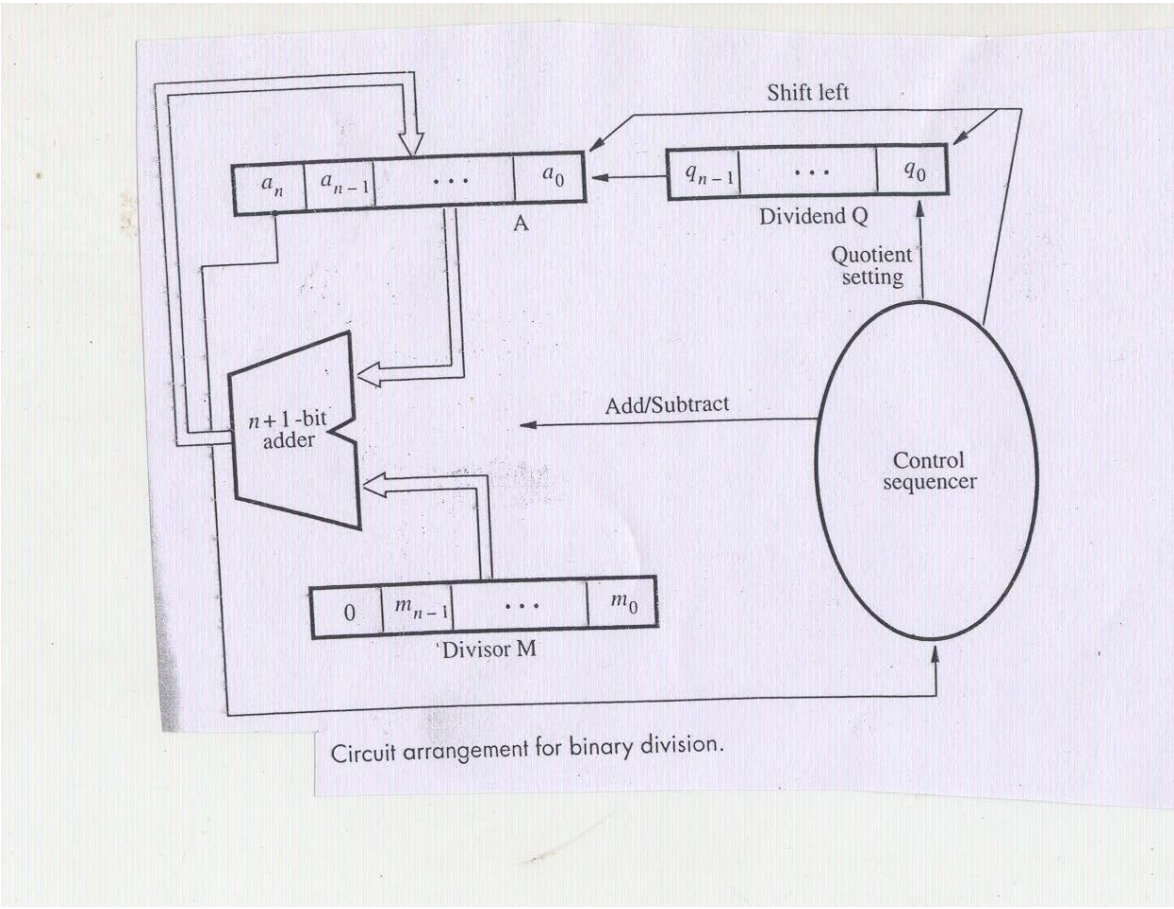
Therefore, transfer rate = 160 K bytes

(*iii*) Latency time

Answer: Latency time = ½ (time in one rotation)
= ½ x 1/40
= 1/80 seconds

Q8 (a) Explain Restoring division and Nonrestoring division process? Also, give algorithms for the two processes?

Answer: The following figure shows a logic circuit arrangement that implements restoring division.



Circuit arrangement for binary division.

Restoring Division Process

In restoring division process, an n-bit positive divisor is loaded into register M and an n-bit positive dividend is loaded into register Q at the start of the operation. Register A is set to 0. After the division is complete, the n-bit quotient is in register Q and the remainder is in register A. the required subtractions are facilitated by using 2's-complement arithmetic. The extra bit position at the left end of both A and M accommodates the sign bit during subtractions. The following algorithm performs restoring division.

1. Shift A and Q left one binary position.
2. Subtract M from A, and place the answer back in A.
3. If the sign of A is 1, set q_0 to 0 and add M back to A (i.e. restore A); otherwise, set q_0 to 1.

Nonrestoring Division Process

The restoring-division algorithm can be improved by avoiding the need for restoring A after an unsuccessful subtraction. Subtraction is said to be unsuccessful if the result is negative. Consider the sequence of operations that takes place after the subtraction operation in Restoring algorithm. If A is positive, we shift left and subtract M, i.e. we perform 2A – M. If A is negative, we restore it by performing A + m, and then we shift it left and subtract M. This is equivalent to performing 2A + M. The q_0 bit is appropriately set to 0 or 1 after the correct operation has been performed. The following algorithm performs nonrestoring division:

Step 1: Do the following n times:

1. If the sign of A is 0, shift A and Q left one bit position and subtract M from A; otherwise, shift A and Q left and add M to A.
2. Now, if the sign of A is 0, set q_0 to 1; otherwise, set q_0 to 0.

Step 2: If the sign of A is 1, add M to A.

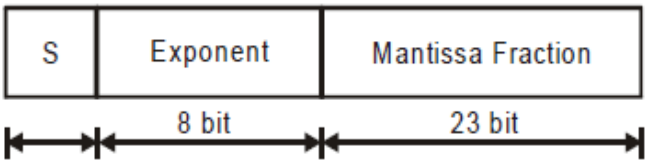
Step 2 is needed to leave the proper positive remainder in A at the end of the n cycles of the step 1.

(b) Briefly discuss IEEE Standard Floating-point representation? Create the 32-bit single-precision IEEE standard representation of the decimal number –(0.625).

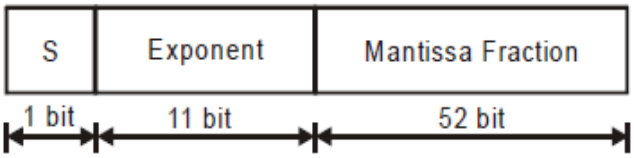
Answer: IEEE Standard specifies two formats for representing floating-point number. One for 32 bits operand (known as single precision) and another for 64 bits operand (known as double precision). The Institute of Electrical and Electronic Engineer (IEEE), U.S.A. formulated a standard, known as IEEE 754 floating point standard, for representing floating point numbers in computers and performing arithmetic operations with the floating point numbers. Such standard is required to ensure portability between two different computers.

Most CPU's these days have floating-points that confirm to the IEEE floating point standard. Both single and double precision formats use radix 2 for fractions and excess notation for exponents.

The format for single and double precision is shown in Fig. 1.39. Both formats starts with a sign bit, 0 for positive sign and 1 for negative sign. Next comes the exponent, using excess-127 representation for single precision and excess-1023 representation for double precision. Finally, we have the mantissa fractions consisting of 23-bits in case of single precision and 52-bits in case of double precision. Thus, a total of 32 bits is needed for single precision and 64 bits is needed for double precision.



Single Precision



Double Precision

A computer must provide at least single-precision representation to conform to the IEEE standard. Double precision representation is optional. The standard also specifies certain optional extended versions of both of these formats. The extended versions are intended to provide increased precision and increased exponent range for the representation.

$$-(0.625)_{10} = -(0.101)_2$$
$$= -(1.01) \times 2^{-1}$$

Since, the given number is negative. The sign bit in IEEE format is 1.

The fraction (value after the radix point and not including the leading 1) is 01. Add 21 zeros to the end to make it 23 bits. Thus, the mantissa fraction, F is

01000000000000000000000

Lastly, the exponent must satisfy the equation

$$E - 127 = -1$$
$$E = 126$$

By, converting $(126)_{10}$ to binary, the 8-bit unsigned binary value is $(01111110)_2$.

Substituting all of these components to IEEE standard we get

Sign bit	Exponent	Mantissa
1	01111110	01000000000000000000000

Therefore, the IEEE representation in 32-bit precision is

10111111001000000000000000000000

Q9 (a) Write the actions required to execute the instructions Move (R1), R2?

Answer: The actions needed to execute the instruction **Move(R1), R2** are:

1. MAR ← [R1]
2. Start a Read operation on the memory bus.
3. Wait for the MFC response from the memory.
4. Load MDR from the memory bus.
5. R2 ← [MDR]

(b) Distinguish between horizontal and vertical microinstruction?

Answer: Horizontal microinstructions have the following characteristics:

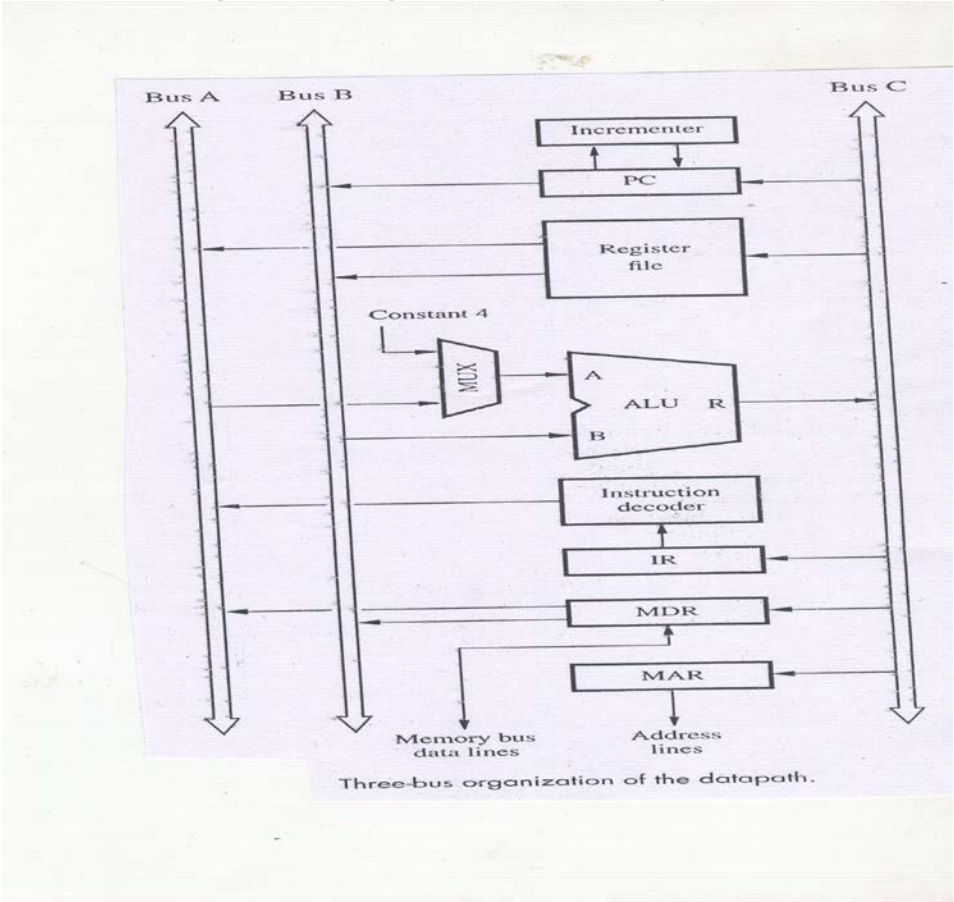
- They usually have long formats.
- They have the ability to express a high degree of parallelism. Hence, they are faster when compared to vertical microinstructions.
- No decoder is used to decode control field.

Vertical microinstructions have the following characteristics:

- They usually have shorts formats, hence are more compact.
- They have limited ability to express parallelism.
- A function code is used for each action to be performed.
- A decoder is used to translate the code into control signals.

(c) With the help of figure, explain multiple-bus organization?

Answer: The following figure depicts a three-bus structure used to connect the registers and the ALU of a processor. All general-purpose registers are combined into a single block called the register file. The register file is said to have three ports. There are two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. The third port allows the data on bus C to be loaded into a third register during the same clock cycle.



Three-bus organization of the datapath.

Buses A and B are used to transfer the source operands to the A and B inputs of the ALU, where an arithmetic or logic operation may be performed. The result is transferred to the destination over bus C. We will call the ALU control signals for such an operation R=A or R=B.

A second feature is the introduction of the Incrementer unit, which is used to increment the PC by 4. Using the Incrementer eliminates the need to add 4 to the PC using the main ALU. The source for the constant 4 at the ALU input multiplexer is still useful. It can be used to increment other addresses, such as memory addresses in LoadMultiple and StoreMultiple instructions.

Consider the three-operand instruction

Add R4, R5, R6

The control sequence for executing this instruction is given below:

Step	Action
1	PC_{out} , R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB} , R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Control sequence for the instruction Add R4, R5, R6
for the three-bus organization

in step 1, the contents of the PC are passed through the ALU, using the R=B control signal, and loaded into the MAR to start a memory read operation. At the same time the PC is incremented by 4. The value loaded into MAR is the original contents of the PC. The incremented value is loaded into the PC at the end of the clock cycle and will not affect the contents of MAR. In step 2, the processor waits for MFC and loads the data received into MDR, then transfers them to IR in step 3. Finally, the execution phase of the instruction requires only one control step to complete, step 4.

TextBook

Computer Organization, Carl Hamacher, Zvonko Vranesic, Safwat Zaky, 5th Edition, TMH, 2002