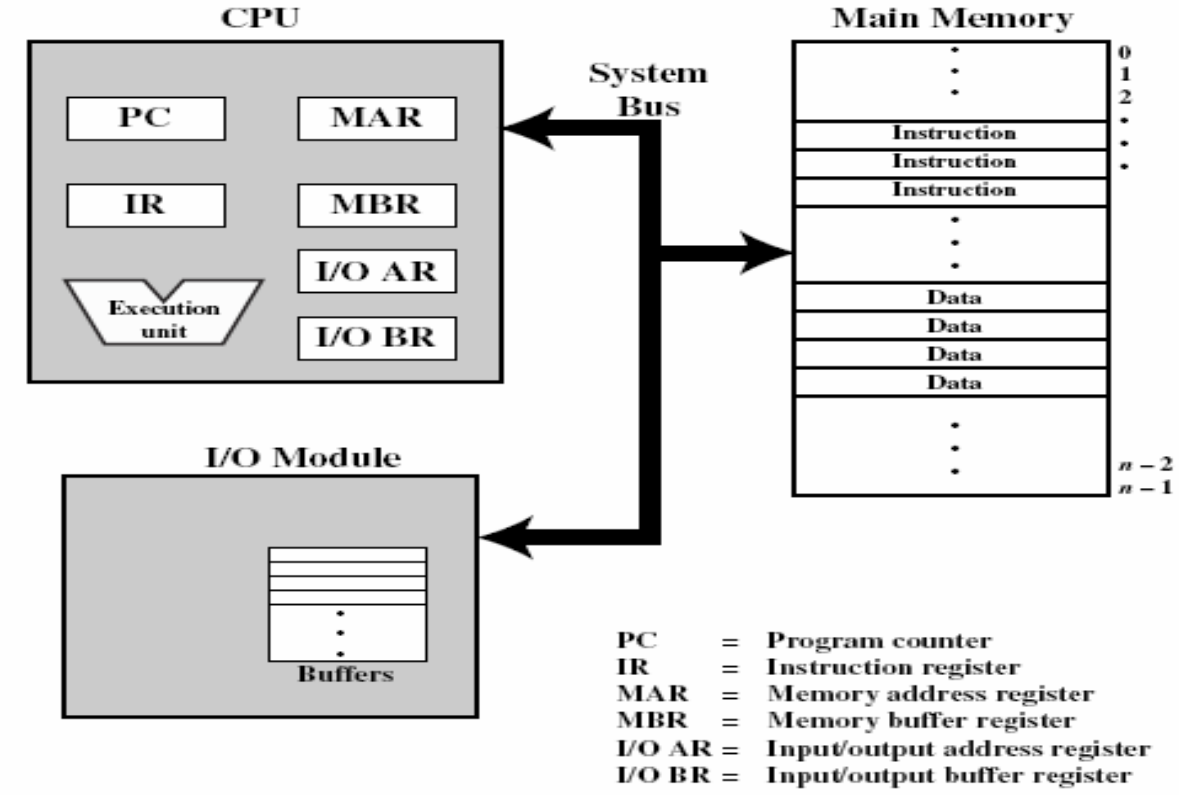
Q.2 a. Explain the basic interaction between the processor and the main memory in a typical computer**.**

Answer:



- CPU – combination of the instruction interpreter (control unit) and the general-purpose arithmetic and logic functions (ALU)
- Input/Output module – used to enter data and instructions and to report results
- Main memory module – used to store both data and instructions

The CPU exchanges data with memory. The CPU typically makes use of two internal registers: a **memory address register (MAR),** which specifies the address in memory for the next read or write, and a **memory buffer register (MBR),** which contains the data to be written into memory or receives the data read from memory. **An I/O address register (I/OAR)** specifies a particular I/O device. An I/O
Buffer register (I/OBR) is used for the exchange of data between an I/O module and the CPU.
A **memory module** consists of a set of locations, defined by sequentially numbered addresses. Each location contains a binary number that can be interpreted as either an instruction or data.
An **I/O module** transfers data from external devices to CPU and memory, and vice versa. It contains internal buffers for temporarily holding these data until they can be sent on.

b. Write the basic performance equation. How performance measurement is done practically?

Answer:

Basic Performance Equation

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate
- Note: these are not independent to each other $T = \frac{N \times S}{R}$

Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.
- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Compile and run (no simulation)
- Reference computer

$$SPEC\ rating = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$$

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

Q.3 a. Explain any four addressing modes giving suitable example.

Answer:

The address field or fields in a typical instruction format are relatively small. We would like to be able to reference a large range of locations in main memory or for some systems, virtual memory. To achieve this objective, a variety of addressing techniques has been employed. They all involve some trade-off between address range and/or addressing flexibility, on the one hand, and the number of memory references and/or the complexity of address calculation, on the other. In this section, we examine the most common addressing techniques:

- Immediate
- Direct
- Indirect
- Register
- Register indirect

Immediate Addressing

The simplest form of addressing is immediate addressing, in which the operand is actually present in the instruction:

- Operand is part of instruction
- Operand = address field
- e.g. ADD 5 ;Add 5 to contents of accumulator ;5 is operand

Direct Addressing

A very simple form of addressing is direct addressing, in which:

- Address field contains address of operand
- Effective address (EA) = address field (A)
- e.g. ADD A ;Add contents of cell A to accumulator

Indirect Addressing

With direct addressing, the length of the address field is usually less than the word length, thus limiting the address range. One solution is to have the address field refer to the address of a word in memory, which in turn contains a full-length address of the operand. This is known as indirect addressing.

Register Addressing

Register addressing is similar to direct addressing. The only difference is that the address field refers to a register rather than a main memory address.

Register Indirect Addressing

Just as register addressing is analogous to direct addressing, register indirect addressing is analogous to indirect addressing. In both cases, the only difference is whether the address field refers to a memory location or a register. Thus, for register indirect address: Operand is in memory cell pointed to by contents of register.

Q.4 a. What are the three basic techniques to perform Input/ Output operations?

Answer:

The CPU executes the I/O instructions and may accept the data temporally, but the ultimate source or destination is the memory unit. Data transfer between the control computer and I/O devices which handled in variety of I/O modes. Some I/O modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit data transfer to and from peripherals may be handled in various possible I/O modes i.e.

(a) Programmed I/O mode

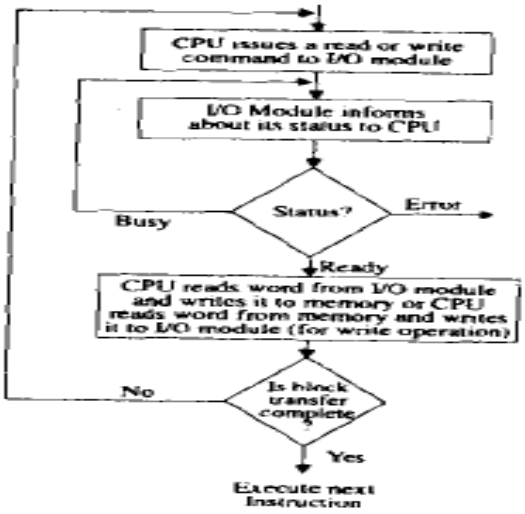
(b) Interrupt initiated I/O mode

(c) Direct Memory Access (DMA).

In programmed I/O, the I/O operations are completely controlled by the CPU. The CPU executes programs that initiate, directs and terminate an I/O operation. It requires a little special I/O hardware, but is quite time consuming for the CPU, since CPU has to wait for slower I/O operations to complete.

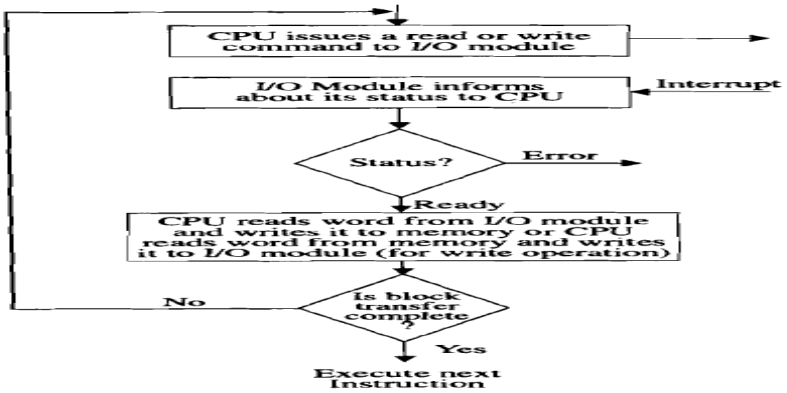
Another technique suggests to reduce the waiting by CPU is interrupt driven I/O. The CPU issues the I/O command to I/O module and starts doing other work, which may be execution of a separate program. When the I/O operation is complete, I/O module interrupts CPU by informing the CPU that I/O has finished. CPU, then, may proceed execution of this program.

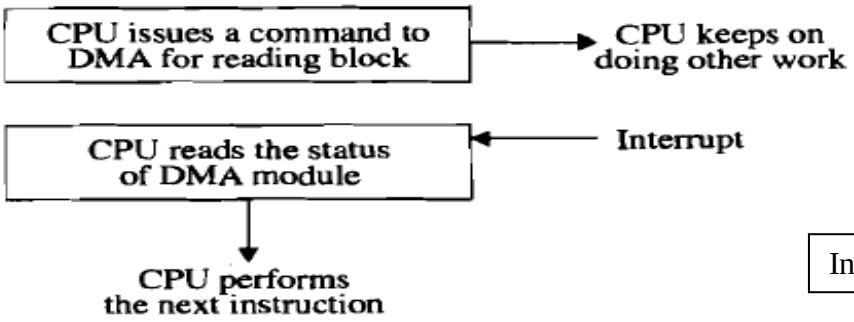
In both programmed I/O and interrupt driven I/OCPU is responsible for extracting data from the memory for Output and storing data in memory for input. Such a requirement does not exist in DMA where the memory can be accessed directly by I/O module. Thus, the I/O module can store or extract data in/from the memory.



(a) Programmed I/O

Programmed I/O





Interrupt driven I/O

Programmed Input/Output

Programmed input/output is useful I/O method for computers where hardware costs need to be minimised. The Input or Output operation in such cases may involve:
Transfer of data from I/O device to the CPU registers.
Transfer of data from CPU registers to memory.
In addition, in a programmed I/O method the responsibility of CPU is to constantly check the status of the I/O device to check whether it has become free (in case output is desired) or it has finished inputting the current series of data (in case input is going on). Thus, Programmed I/O is a very time consuming method where CPU wastes lot of time for checking and verifying the status of an I/0 device.

The interrupt driven I/O

The interrupt driven I/O mechanism for transferring a block of data is shown in figure. Note that after issuing a READ command (for input) the CPU goes off to do other useful work (it may be execution of a different program) while I/O module proceeds for reading of data from associated device. At the completion of an instruction cycle (already discussed in Unit 1 of this block) the CPU checks for interrupts (which will occur when data is in data register of I/O module and it now needs CPU's attention).

Now CPU saves the important register and processor status of the executing program in a stack and request I/O device to provide its data, which is placed on data bus by I/O device. After taking the required action with the data. The CPU can go back to the program it was executing before the Interrupt.

Interrupts are primarily issued on:
\
initiation of Input/Output operation
completion of an Input/Output operation
occurrence of hardware or software errors.

Interrupts can be generated by various sources internal external to the CPU. An interrupt generated internally by CPU is sometimes termed as Traps. The Traps are normally results of programming errors such as division by zero while execution of a program.

The two key issues in Interrupt driven Input/output are: to determine the device which has issued an interrupt in case of occurrence of multiple interrupts which one to be processed first.

DMA

When an I/O is requested, the CPU instructs the DMA module about the operation by providing the information:
Which operation (Read or Write) to be performed.
Address of the I/O device \ to be used.

The starting location on the memory where the information will be read or written to the number to be written or to be read.

The DMA module transfers the requested block byte by byte directly to the memory without Intervening the CPU. On completion of the request DMA module sends an interrupt signal to the CPU.

b. How to ensure that an active interrupt request signal does not lead to successive interruptions, causing the system to enter an infinite loop?

Answer:

The first possibility is to have the processor hardware ignore the interrupt request line until the first instruction of the interrupt-service routine has been completely executed. This can be implemented by using Interrupt-Disable /Interrupt- enable instruction pair

Another most common method is to have processor automatically disable interrupt before it starts executing the interrupt-service routines. This is by using one bit PS register, interrupt mask, indicates whether interrupt are disabled.

The third approach is to arrange the interrupt-handling circuit in the processor so that it responds only to the leading edge of the interrupt-request signal. This type of interrupt request lines are said to be edge-triggered.

c. How can the processor recognize the device requesting an interrupt? Explain briefly.

Answer:

When a device interrupt occurs, basically it is checked through **Daisy chaining priority method** into determine which device issued the interrupt.

The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt. The device with the highest is placed in first position, followed by lower-priority devices upto the device with the lowest priority which is placed last in chain. This method of connection between three devices and CPU is shown in figure. The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in low-level state, the interrupt line goes to low level states and enables the interrupt input in CPU. When no interrupts are pending, interrupt line stays in the high level state and no interrupts are recognized by the CPU. This is equivalent to a negative logic OR operation. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its P1 (priority in) input. The acknowledge signal passes on to next device through P0 (priority out) output only if device 1 is not requesting an interrupt. If devices has a pending interrupt, it blocks the acknowledge signal from next device by placing 0 in P0 output. If then proceeds to insert its own interrupt Vector Address (VAD) into the data bus for the CPU to use during the interrupt cycle.

A device with 0 in its input generates a 0 in its P0 output to inform the next lower priority device that the acknowledge signal has been blocked. A device that is requesting an interrupt and has a 1 in its P1 input will intercept the acknowledge signal by placing 0 in

its P0 output. If the device does not have pending interrupts, it transmits the acknowledge signal to the next device placing interrupts, a L in its P0 output (Thus the device with P1 =1 and PC =0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus. The daisy chain arrangement gives the highest priority to the device that receives the interrupts acknowledge signal from the CPU. The farther the device is form the first position, the lower to its priority.

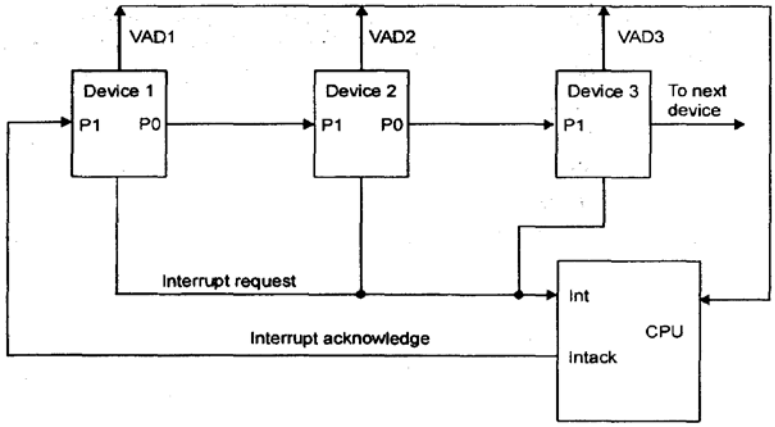


Figure shows the internal logic that must be included within each device when connected in the daisy chaining scheme. The device sets its RF flip-flop when its wants to interrupt the CPU. The Output of the R.F. flip-flop goes through an open-collector in verter, a circuit that provide the wired logic for the common interrupt line. If PT = 0, both PC and the enable line to VAD are equal to 0, irrespective of value of RF. If P1 = I and RF 0, then P0 = 1 and vector address is disabled. This condition passes the acknowledge signal to the next device through P0. The device is active when P1 = I and RF = 1. This condition places 0 in P0 and enables the vector address for the data bus. It is assumed that each device has its own distinct vector address. The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

Q.5 a. How the data transfer happens over the single bus arrangement? Explain the role of interface circuit.

Answer:

-Most modern computer computers use a single bus arrangement

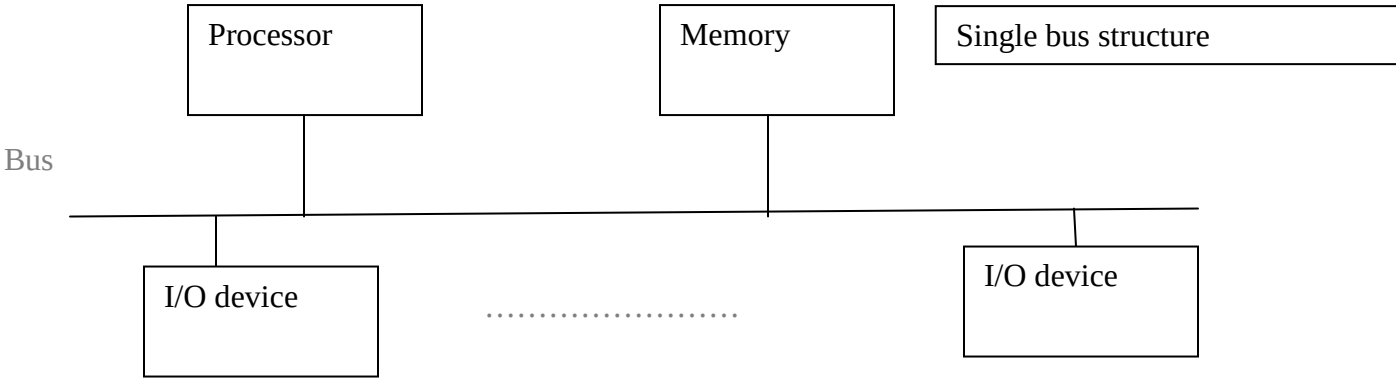
-The processor memory, I/O devices connect to the bus, which consist of lines used to carry address, data and control signals

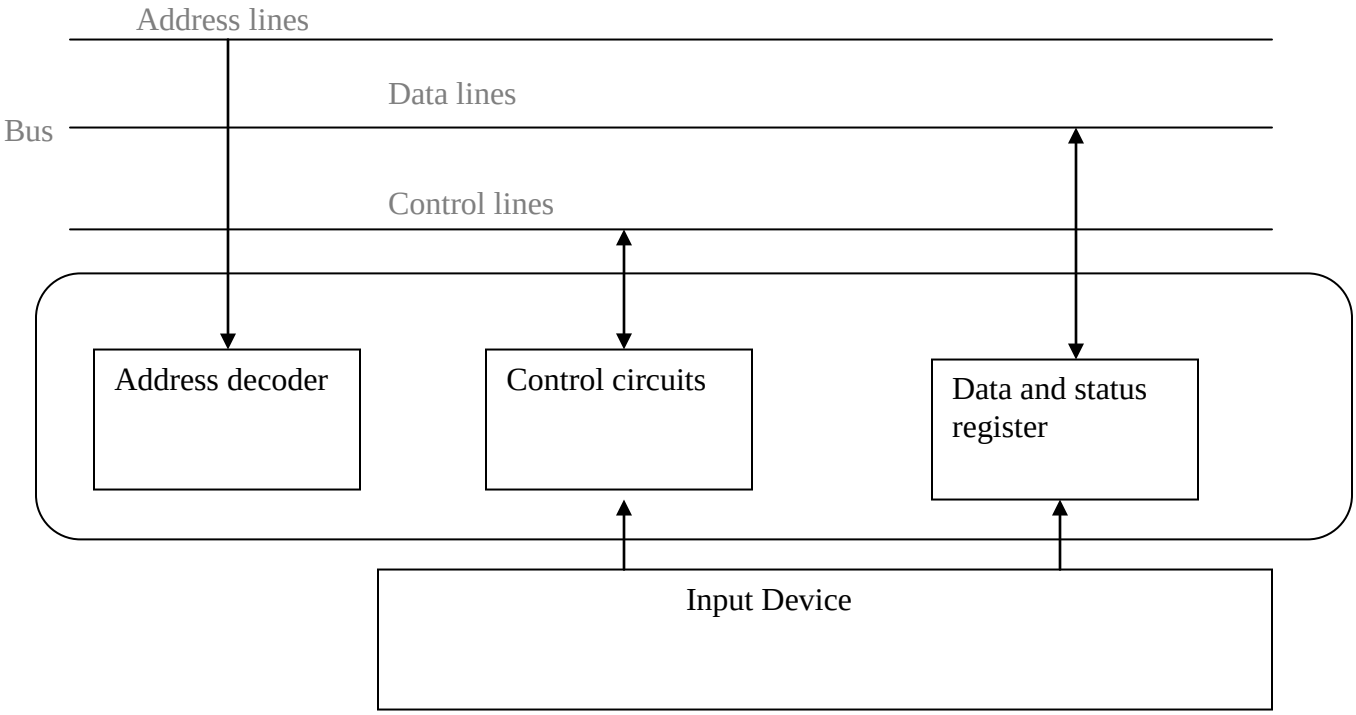
-Each I/O device is assigned a unique set of address

-When the processor places a particular address on the address lines, the device that recognize this address respond to the command issued on the control lines. The data re transferred over data lines

The address decoder enables the device to reconfigure its address when it appears on the address lines. The data register hold data to be transferred to the processor through an input device or receives data from the processor for transfer to an output device. The status register contains information relevant to the operation of the I/O device. Such registers are also connected to the data bus and assigned unique addresses. The

address decoder, the data status registers and control circuitry required to coordinate the I/O transfers constitute the device's interface circuit.





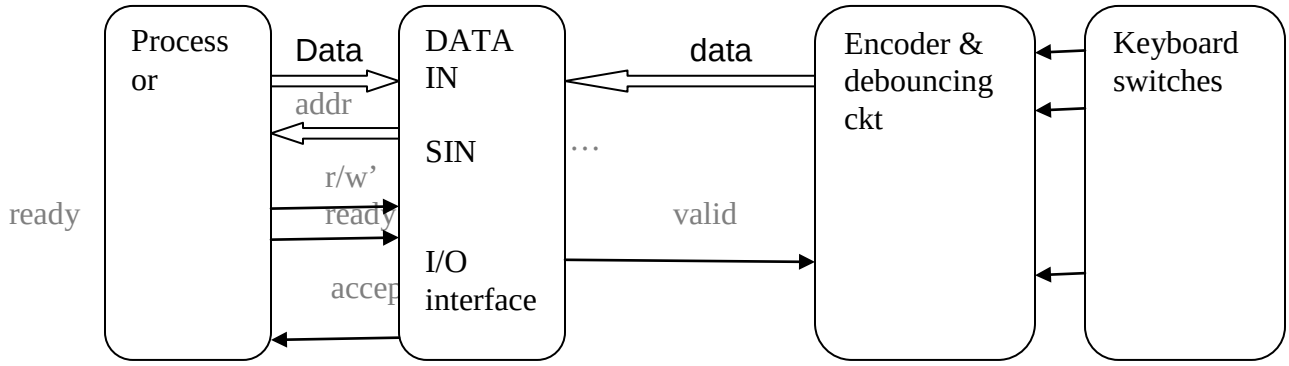
I/O interface for an input device

b. Explain a parallel input interface scheme used to connect the keyboard to the processor.

Answer:

- keyboard consists normally open switch
- when key is closed a connection is established
- an encoder circuit generates the ASCII code from this signal
- debouncing circuit-to eliminate bouncing problem
- DATA IN- data register, SIN status flag(valid signal changes it to 0-1)

- SIN 1 causes ASCII to loaded into DATAIN and SIN set to 0 when the content is read
- R/W' -read write transefer

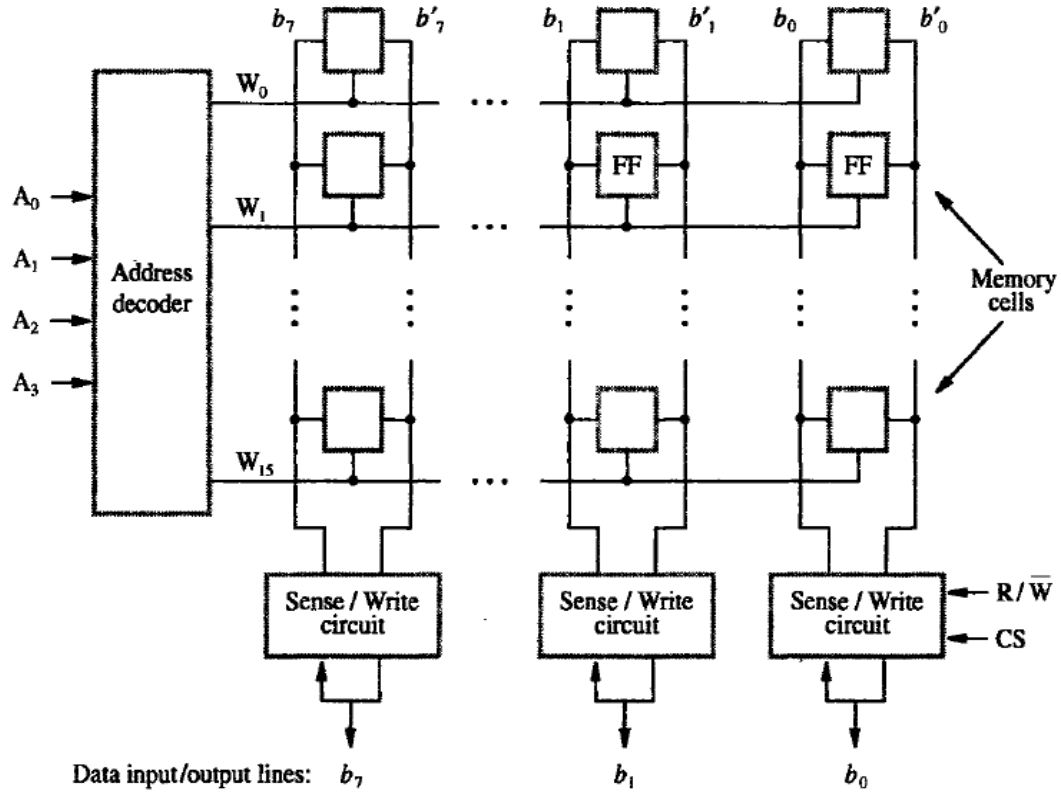


Q.6 a. Explain the internal organization of bit cells in a memory chip which can store 16 words of 8 bit each.

Answer:

- Memory cells are usually organized in the form of an array, in which each cell is capable of storing one bit of information.
- Each row of cells constitutes a memory word, and all cells of a row are connected to a common line called the *word line*, which is driven by the address decoder on the chip.
- The cells in each column are connected to a Sense/Write circuit by two *bit lines.*
- The Sense/Write circuits are connected to the data input/output lines of the chip.
- During a Read operation, these circuits sense, or read, the information stored in the cells selected by a word line and transmit this information to the output data lines.
- During a Write operation, the Sense/Write circuits receive input information and store it in the cells of the selected word.

Two control lines, R/W' and CS (Chip Select) are also provided

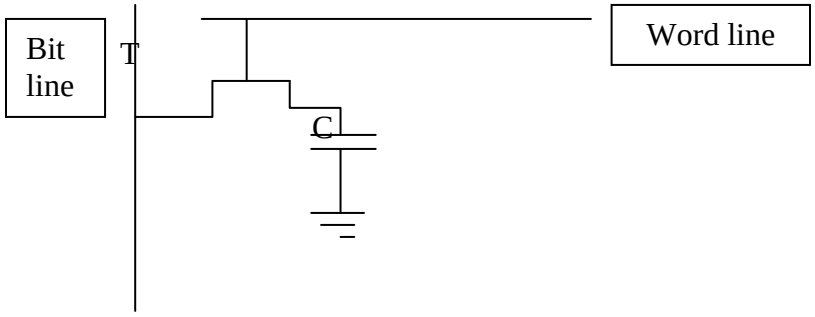


b. Describe the working of a DRAM cell.

Answer:

In *dynamic memory,* information is stored in the form of a charge on a capacitor. A DRAM is capable of storing information for only a few milliseconds. Since each cell is usually required to store information for a much longer time, its contents must be periodically refreshed by restoring the capacitor charge to its full value.

An example of a dynamic memory cell that consists of a capacitor, C, and a transistor, *T*, is shown in Figure In order to store information in this cell, transistor *T* is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored on the capacitor.



After the transistor is turned off, the capacitor begins to discharge. This is caused by the capacitor's own leakage resistance and by the fact that the transistor continues to conduct a tiny amount of current, measured in picoamperes, alter it is turned off. Hence, the information stored in the cell can be retrieved correctly only if it is read before the charge

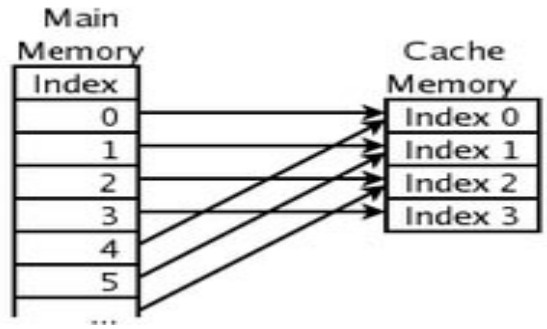
on the capacitor drops below some threshold value. During a Read operation, the bit line is placed in a high-impedance state, and the transistor is turned on. A sense circuit connected to the bit line determines whether the charge on the capacitor is above or below the threshold value. Because this charge is so small, the Read operation is an intricate process whose details are beyond the scope of this text. The Read operation discharges the capacitor in the cell that is being accessed. In order to retain the information stored in the cell, DRAM includes special circuitry that writes back the value that has been read. A memory cell is therefore refreshed every time its contents are read. In fact, all cells connected to a given word line are refreshed whenever this word line is activated.

c What do you mean by direct mapping method to determine the cache location to store memory block?

Answer:

Direct Mapping

If each entry in main memory can go in just one place in the cache, the cache is **direct mapped**. Here each datum can only go in one entry. It doesn't have a replacement policy as such, since there is no choice of which datum to evict. This means that if two locations map to the same entry, they may continually knock each other out. Although simpler, a direct-mapped cache needs to be much larger than an associative one to give comparable performance, and is more unpredictable



More Specifically

The address is broken into three parts: (s-r) MSB bits represent the tag to be stored in a line of the cache corresponding to the block stored in the line; r bits in the middle identifying which line the block is always stored in; and the w LSB bits identifying each word within the block. This means that:

- The number of addressable units = 2^{s+w} words or bytes
- The block size (cache line width not including tag) = 2^{w} words or bytes
- The number of blocks in main memory = 2^{s} (i.e., all the bits that are not in w)
- The number of lines in cache = m = 2^{r}
- The size of the tag stored in each line of the cache = (s - r) bits

Direct mapping is simple and inexpensive to implement, but if a program accesses 2 blocks that map to the same line repeatedly, the cache begins to thrash back and forth reloading the line over and over again meaning misses are very high.

Q.7 a. What do you mean by virtual memory? How this is useful? Explain the basic hardwires required to implement the virtual memory.

Answer:

Virtual Memory

The basic abstraction provided by the OS for memory management VM enables programs to execute without requiring their entire address space to be resident in physical memory

Program can run on machines with less physical RAM than it "needs"

Virtual memory also *isolates* processes from each other

- One process cannot access memory addresses in others
- Each process has its own isolated address space

VM requires both hardware and OS support

- Hardware support: *memory management unit* (MMU) and *translation look aside buffer* (TLB)
- OS support: *virtual memory system* to control the MMU and TLB

A *virtual address* is a memory address that a process uses to access its own memory

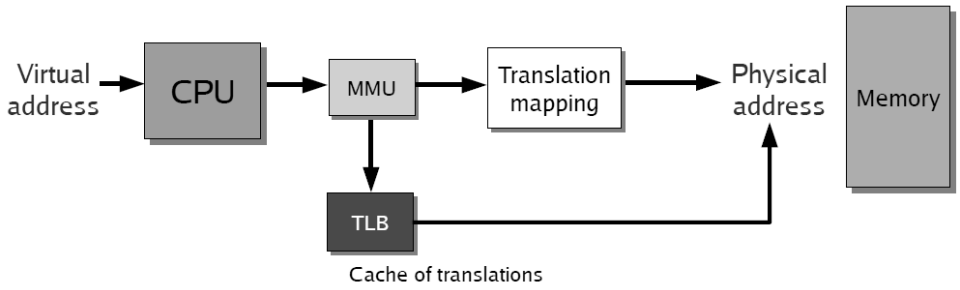
- The virtual address is *not the same* as the physical RAM address in which it is stored
- When a process accesses a virtual address, the MMU hardware *translates* the virtual address into a physical address
- The OS determines the *mapping* from virtual address to physical address

Memory Management Unit (MMU)

- Hardware that translates a virtual address to a physical address
- Each memory reference is passed through the MMU
- Translate a virtual address to a physical address
- *Lots of ways of doing this!*

Translation Lookaside Buffer (TLB)

- **Cache** for MMU virtual-to-physical address translations
- Just an optimization – but an important one!



b. Explain a method for virtual address translation.

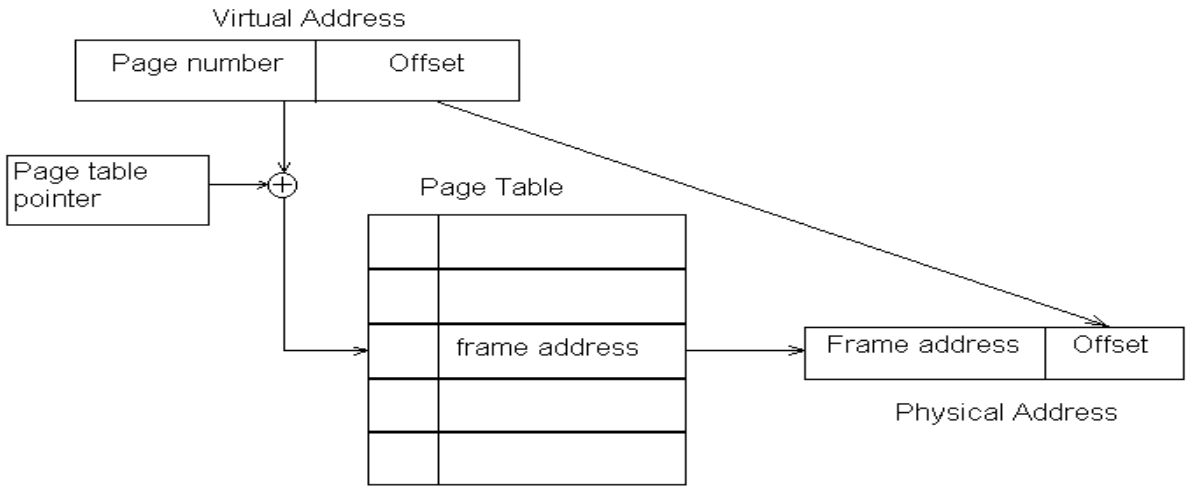
Answer:

Virtual Address Translation

- Virtual-to-physical address translation performed by MMU
- Virtual address is broken into a *virtual page number* and an *offset*
- Mapping from virtual page to physical frame provided by a *page table*
- Various bits accessed by MMU on each page access:
  - Valid bit (V): Whether the corresponding page is in memory
  - Modify bit (M): Indicates whether a page is "*dirty*" (modified)
  - Reference bit (R): Indicates whether a page has been accessed (read or written)
  - Protection bits: Specify if page is readable, writable, or executable
  - Page frame number: Physical location of page in RAM

Virtual address:

- A logical address
- Virtual page number + offset
- Finds Page table entry for virtual page number
- Extract frame number and adds offset
- Fail(MMU raises an exception -page fault):
- bounds error-outside address range
- validation error-non-resident page
- protection error-not permitted access



c. What do you mean by a page fault? Which hardware is responsible for detecting the page fault?

Answer:

A page fault is a trap to the software raised by the hardware when a program accesses a page that is mapped in the virtual address space, but not loaded in physical memory. In the typical case the operating system tries to handle the page fault by making the required page accessible at a location in physical memory or kills the program in the case of an illegal access. The hardware that detects a page fault is the memory management unit in a processor.

Q.8 a. Using booth's multiplication algorithm to multiply the following, showing all the steps:

(i) 3×-4 (ii) -8×2

Answer:

- A = 0011 0000 0
- S = 1101 0000 0
- P = 0000 1100 0

- Perform the loop four times :
  1. P = 0000 110**0 0**. The last two bits are 00.
     - P = 0000 0110 0. A right shift.
  2. P = 0000 011**0 0**. The last two bits are 00.
     - P = 0000 0011 0. A right shift.
  3. P = 0000 001**1 0**. The last two bits are 10.
     - P = 1101 0011 0. P = P + S.
     - P = 1110 1001 1. A right shift.
  4. P = 1110 100**1 1**. The last two bits are 11.
     - P = 1111 0100 1. A right shift.

The product is 1111 0100, which is -12.

multiplying -8 by 2 using 4 bits for the multiplicand and the multiplier:

- A = 1 1000 0000 0
- S = 0 1000 0000 0
- P = 0 0000 0010 0

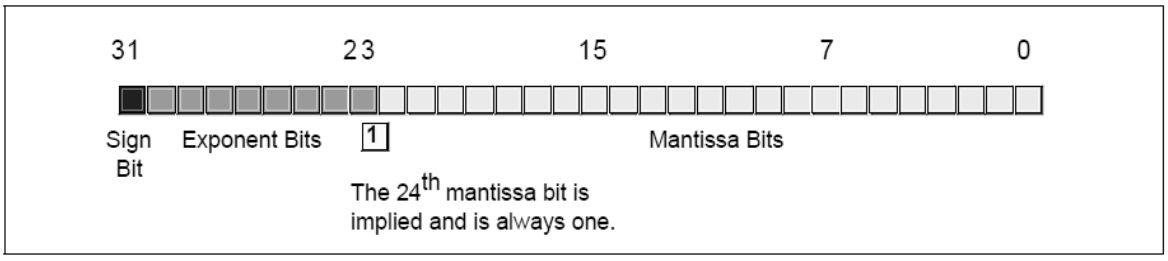
- Perform the loop four times :
  1. P = 0 0000 0010 0. The last two bits are 00.
     - P = 0 0000 0001 0. Right shift.
  2. P = 0 0000 0001 0. The last two bits are 10.
     - P = 0 1000 0001 0. P = P + S.
     - P = 0 0100 0000 1. Right shift.
  3. P = 0 0100 0000 1. The last two bits are 01.
     - P = 1 1100 0000 1. P = P + A.
     - P = 1 1110 0000 0. Right shift.
  4. P = 1 1110 0000 0. The last two bits are 00.
     - P = 1 1111 0000 0. Right shift.

The product is 11110000 (after discarding the first and the last bit) which is -16.

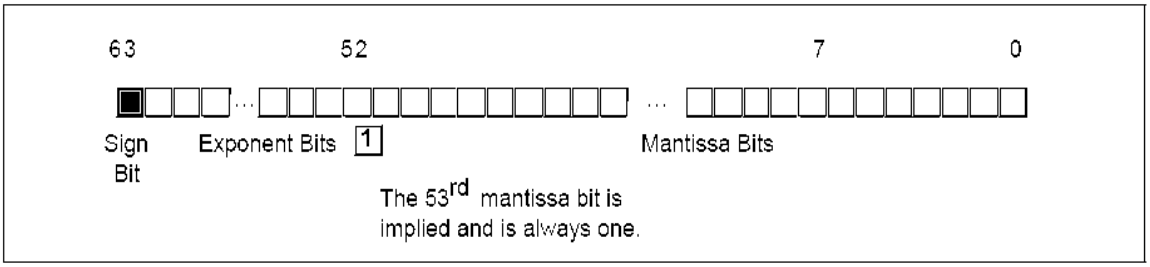
b. Differentiate between single precision and double precision IEEE standard floating point representations.

Answer:

The single precision format uses a one's complement 24 bit mantissa and an eight bit excess-128 exponent. The mantissa usually represents a value between 1.0 to just under 2.0. The H.O. bit of the mantissa is always assumed to be one and represents a value just to the left of the *binary point*2. The remaining 23 mantissa bits appear to the right of the binary point. The double precision format helps overcome the problems of single preicision floating point. Using twice the space, the double precision format has an 11-bit excess-1023 exponent and a 53 bit mantissa (with an implied H.O. bit of one) plus a sign bit. This provides a dynamic range of about 10



Single precision representation



Double precision representation

Q.9 a. Differentiate between hardwired control and micro programmed control.

Answer:

•Control unit (CU) of a processor translates from machine instructions to the control signals for the micro operations that implement them

•Control units are implemented in one of two ways

The control unit generates the signals for sequencing the operations in the data path

– A sequential circuit with states that *dictate the control signals* for the system

– Using status conditions and control inputs, the sequential control unit *Determines the next state* in which additional microoperations are activated.

•***Hardwired*** **Control**

–CU is made up of sequential and combinational circuits to generate the control signals

– The control unit is implemented to provide a particular digital function

•***Micro programmed*** **Control**

–A control memory on the processor contains micro programs that activate the necessary control signals

The control unit's binary control values are stores as words in a micro programmed control (usually ROM).

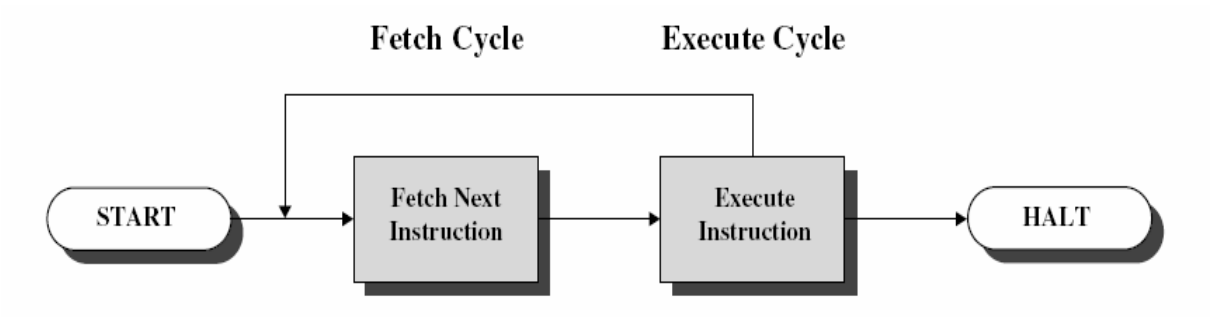
– Each word in the control contains a microinstruction

– A sequence of microinstructions constitutes a micro program

– Firmware!

b. Explain the 'instruction cycle' for the processing of a single instruction in a computer.

Answer:



The processing required for a single instruction is called an **instruction cycle**.

Instruction Fetch and Execute

• The processor fetches an instruction from memory – program counter (PC) register holds the address of the instruction to be fetched next
• The processor increments the PC after each instruction fetch so that it will fetch the next instruction in the sequence – unless told otherwise
• The fetched instruction is loaded into the instruction register (IR) in the processor – the instruction contains bits that specify the action the processor will take
• The processor interprets the instruction and performs the required action

These actions fall into four categories:

• **Processor-memory** – data transferred to or from the processor to memory
• **Processor-I/O** – data transferred to or from a peripheral device by transferring between the processor and an I/O module
• **Data processing** – the processor performs some arithmetic or logic operation on data
• **Control** – an instruction may specify that the sequence of execution be altered
An instruction execution may involve a combination of these actions.

c. Write short notes on the following:
   (i) Control word (CW)
   (ii) Microinstruction
   (iii) Micro-program

Answer:
Control word (CW):
A word with each bit for one of the control signals. Each step of the instruction execution is represented by a control word with all of the bits corresponding to the control signals needed for the step set to one.

Microinstruction:
Each step in a sequence of steps in the execution of a certain machine instruction is considered as a *microinstruction*, and it is represented by a control word. All of the bits corresponding to the control signals that need to be asserted in this step are set to 1, and all others are set to 0 (*horizontal organization*).

Micro-program:

Composed of a sequence of microinstructions corresponding to the sequence of steps in the execution of a given machine instruction.

TEXT BOOK

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