BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)
COLLEGE OF ENGINEERING, PUNE
DEPARTMENT OF COMPUTER ENGINEERING

Input/output Organization

Module Function

The major functions or requirements for an I/O module fall into the following categories:

- Control and timing
- Processor communication
- Device communication
- Data buffering
- Error detection

Control and timing

- Processor has to establish the interaction with one or more external devices in an unpredictable pattern
- I/O module includes the a **control and timing requirement**, to coordinate the flow of traffic between internal resources and external devices
- Example for control and timing: control of the transfer of data from an external device to the processor might involve the following sequence of steps:

1. The processor interrogates the I/O module to check the status of the attached device.

2. The I/O module returns the device status.

3. If the device is operational and ready to transmit, the processor requests the transfer of data, by means of a command to the I/O module.

4. The I/O module obtains a unit of data (e.g., 8 or 16 bits) from the external device.

5. The data are transferred from the I/O module to the processor.

Processor Communication

Processor communication involves the following:

- **Command Decoding:** The I/O module accepts commands from the processor, typically sent as signals on the control bus. For example, an I/O module for a disk drive might accept the following commands: READ SECTOR, WRITE SECTOR, SEEK track number, and SCAN record ID.
- **Data:** Data are exchanged between the processor and the I/O module over the data bus.
- **Status reporting:** Because peripherals are so slow, it is important to know the status of the I/O module. For example, if an I/O module is asked to send data to the processor (read), it may not be ready to do so because it is still working on the previous I/O command. This fact can be reported with a status signal. Common status signals are BUSY and READY. There may also be signals to report various error conditions.
- Address Recognition: **:** Just as each word of memory has an address, each I/O device has address. I/O module must recognize one unique address for each peripheral it controls.

Data Buffering

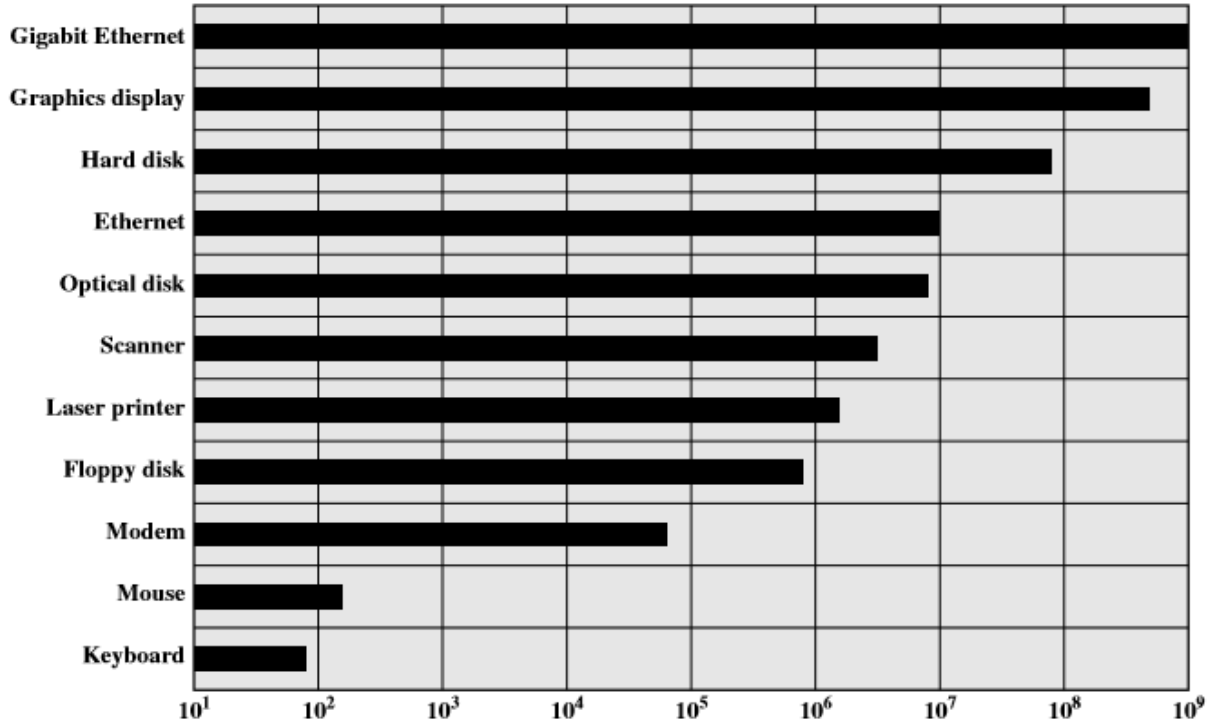


figure: Typical I/O Device Data rates

The transfer rate into and out of main memory or the processor is quite high, the rate is orders of magnitude lower for many peripheral devices and covers a wide range. Data coming from main memory are sent to an I/O module in a rapid burst. The data are buffered in the I/O module and then sent to the peripheral device at its data rate.

if the I/O device operates at a rate higher than the memory access rate, then the I/O module performs the needed buffering operation.

Error Detection

- I/O module is often responsible for **error detection** and for subsequently reporting errors to the processor.
- One class of errors includes mechanical and electrical malfunctions reported by the device (e.g., paper jam, bad disk track).
- Another class consists of unintentional changes to the bit pattern as it is transmitted from device to I/O module.
- A simple example is the use of a parity bit on each character of data. For example, the IRA character code occupies 7 bits of a byte. The eighth bit is set so that the total number of 1s in the byte is even (even parity) or odd (odd parity). When a byte is received, the I/O module checks the parity to determine whether an error has occurred.

I/O Module Structure

- I/O modules vary considerably in complexity and the number of external devices that they control.

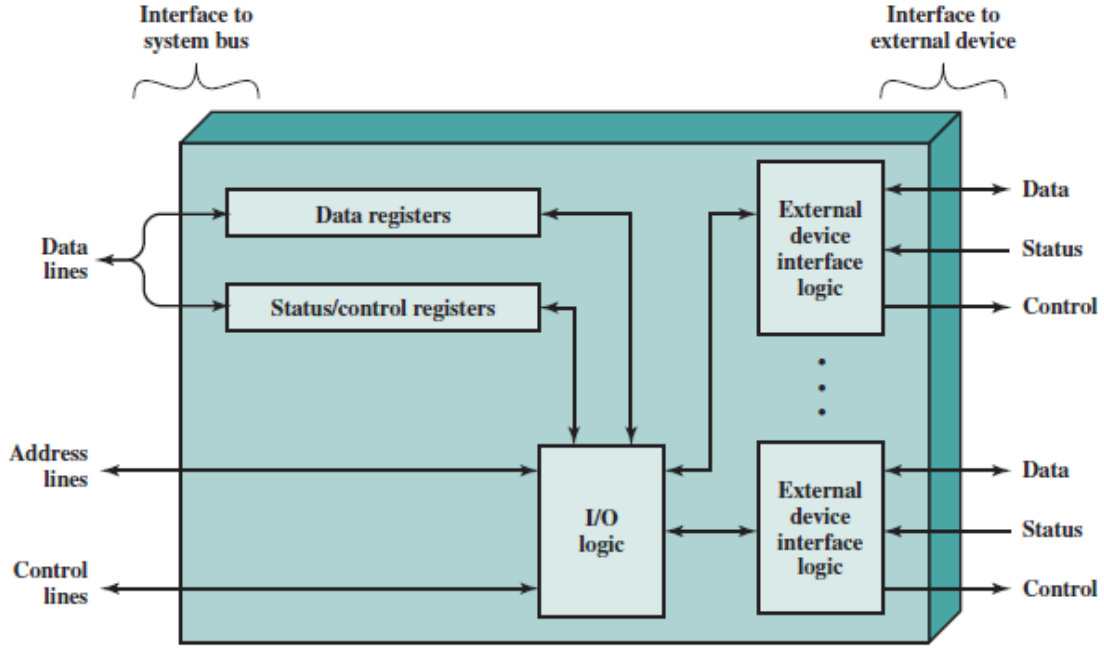
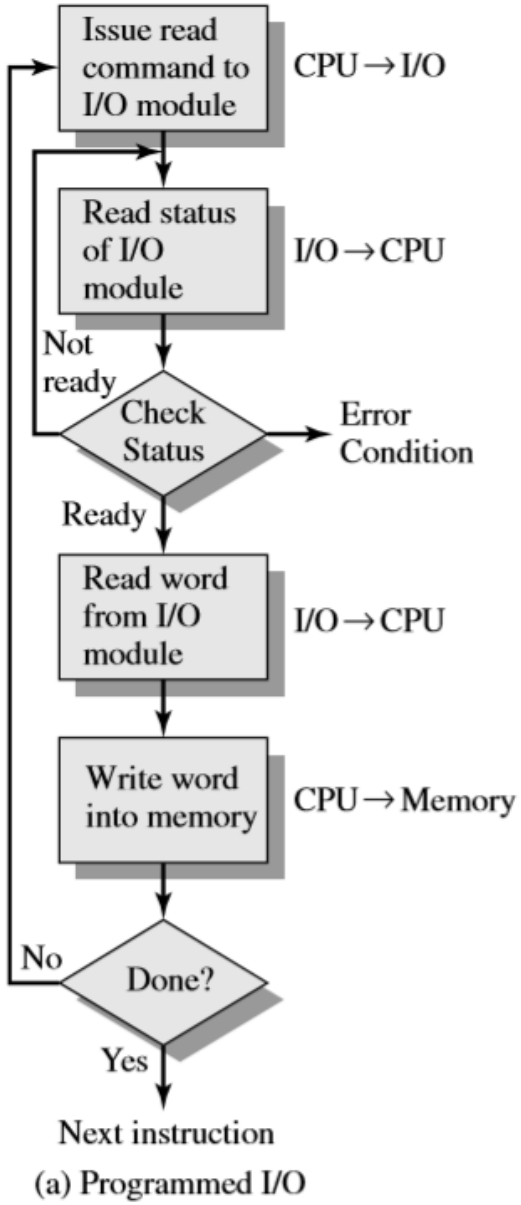


Figure 7.3 Block Diagram of an I/O Module

- Figure 7.3 provides a general block diagram of an I/O module. The module connects to the rest of the computer through a set of signal lines (e.g., system bus lines).

Programmed I/O



(a) Programmed I/O

Programmed I/O(contd.)

- With programmed I/O, the I/O module will perform the requested action and then set the appropriate bits in the I/O status register.
- The I/O module takes no further action to alert the processor.
- In particular, it does not interrupt the processor.
- Thus, it is the responsibility of the processor periodically to check the status of the I/O module until it finds that the operation is complete.

I/O Commands

To execute an I/O-related instruction, the processor issues an address, specifying the particular I/O module and external device, and an I/O command. There are four types of I/O commands that an I/O module may receive when it is addressed by a processor:

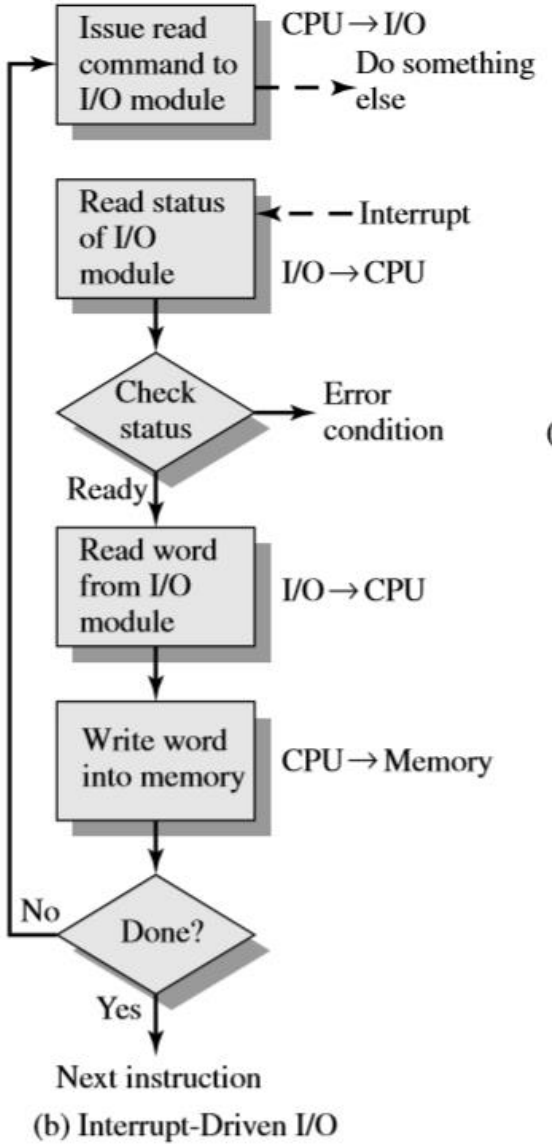
Control: Used to activate a peripheral and tell it what to do.

Test: Used to test various status conditions associated with an I/O module and its peripherals.

Read: Causes the I/O module to obtain an item of data from the peripheral and place it in an internal buffer.

Write: Causes the I/O module to take an item of data (byte or word) from the data bus and subsequently transmit that data item to the peripheral.

Interrupt-Driven I/O



(b) Interrupt-Driven I/O

Interrupt-Driven I/O(Contd.)

- The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.
- The processor, while waiting, must repeatedly interrogate the status of the I/O module.
- This type of I/O operation, where the CPU constantly tests a part to see if data is available, is polling, that is, the CPU Polls (asks) the port if it has data available or if it is capable of accepting data. Polled I/O is inherently inefficient.
- The solution to this problem is to provide an interrupt mechanism.
- In this approach the processor issues an I/O command to a module and then go on to do some other useful work.

Interrupt-Driven I/O(Contd.)

Let us consider how it works.

a) From the point of view of the I/O module:

- For input, the I/O module services a READ command from the processor.
- The I/O module then proceeds to read data from an associated peripheral device.
- Once the data are in the modules data register, the module issues an interrupt to the processor over a control line.
- The module then waits until its data are requested by the processor.
- When the request is made, the module places its data on the data bus and is then ready for another I/O operation.

b) From the processor point of view; the action for an input is as follows:

- The processor issues a READ command.
- It then does something else(e.g. the processor may be working on several different programs at the same time)
- At the end of each instruction cycle, the processor checks for interrupts
- When the interrupt from an I/O module occurs, the processor saves the context (e.g. program counter & processor registers) of the current program and processes the interrupt.
- In this case, the processor reads the word of data from the I/O module and stores it in memory.
- If then restores the context of the program it was working on and resumes execution.

Interrupt Processing

- The occurrence of an interrupt triggers a number of events, both in the processor hardware and in software. When an I/O device completes an I/O operation, the following sequences of hardware events occurs:

1. The device issues an interrupt signal to the processor.
2. The processor finishes execution of the current instruction before responding to the interrupt.
3. The processor test for the interrupt; if there is one interrupt pending, and then the processor sends as acknowledgement signal to the device which issued the interrupt. After getting acknowledgement, the device removes its interrupt signals.
4. The processor now needs to prepare to transfer control to the interrupt routine. It needs to save the information needed to resume the current program at the point of interrupt. The minimum information required to save is the processor status word (PSW) and the location of the next instruction to be executed which is nothing but the contents of program counter. These can be pushed into the system control stack.
5. The processor now loads the program counter with the entry location of the interrupt handling program that will respond to the interrupt. Once the program counter has been loaded, the processor proceeds to the next instruction cycle, which begins with an interrupt fetch. The control will transfer to interrupt handler routine for the current interrupt. The following operations are performed at this point.

At the point, the program counter and PSW relating to the interrupted program have been saved on the system stack. In addition to that some move information must be because

- The interrupt handles next processes the interrupt. This includes an examination of status information relating to the I/O operation or, other event that caused an interrupt.
- When interrupt processing is complete, the saved register values are retrieved from the stack and restored to the registers.
- The final act is to register the PSW and program counter values from the stack. As a result, the next instruction to be executed will be from the previously interrupted program.

I/O Channels

- A channel is an independent hardware component that co-ordinate all I/O to a set of controllers. Computer systems that use I/O channel have special hardware components that handle all I/O operations.
- The I/O channel represents an extension of the DMA concept.
- An I/O channel has the ability to execute I/O instructions, which gives it complete control over I/O operations.
- In a computer system with such devices, the CPU does not execute I/O instructions. Such instructions are stored in main memory to be executed by a special-purpose processor in the I/O channel itself. Thus, the CPU initiates an I/O transfer by instructing the I/O channel to execute a program in memory. The program will specify the device or devices, the area or areas of memory for storage, priority, and actions to be taken for certain error conditions.

I/O Channels(Contd.)

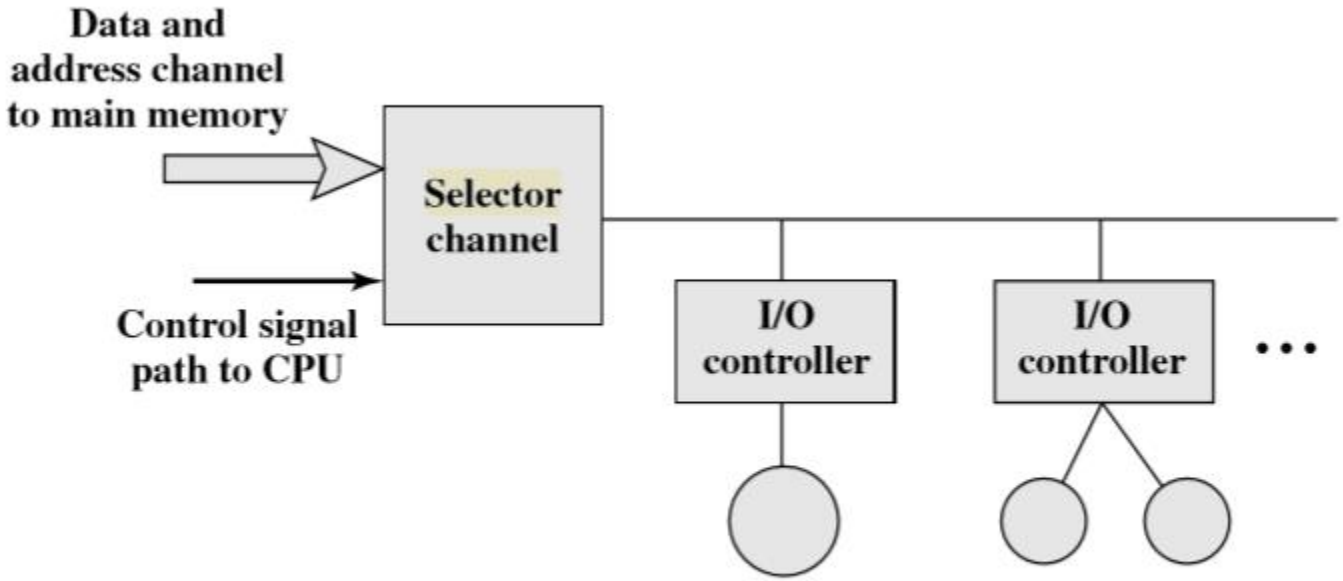
- Two types of I/O channels are common:

1. Selector
2. Multiplexer

1.Selector

A *selector channel* controls multiple high-speed devices and, at any one time, is dedicated to the transfer of data with one of those devices. Thus, the I/O channel selects one device and effects the data transfer. Each device, or a small set of devices, is handled by a *controller,* or I/O module, that is much like the I/O modules we have been discussing. Thus, the I/O channel serves in place of the CPU in controlling these I/O controllers.

I/O channel: Selector

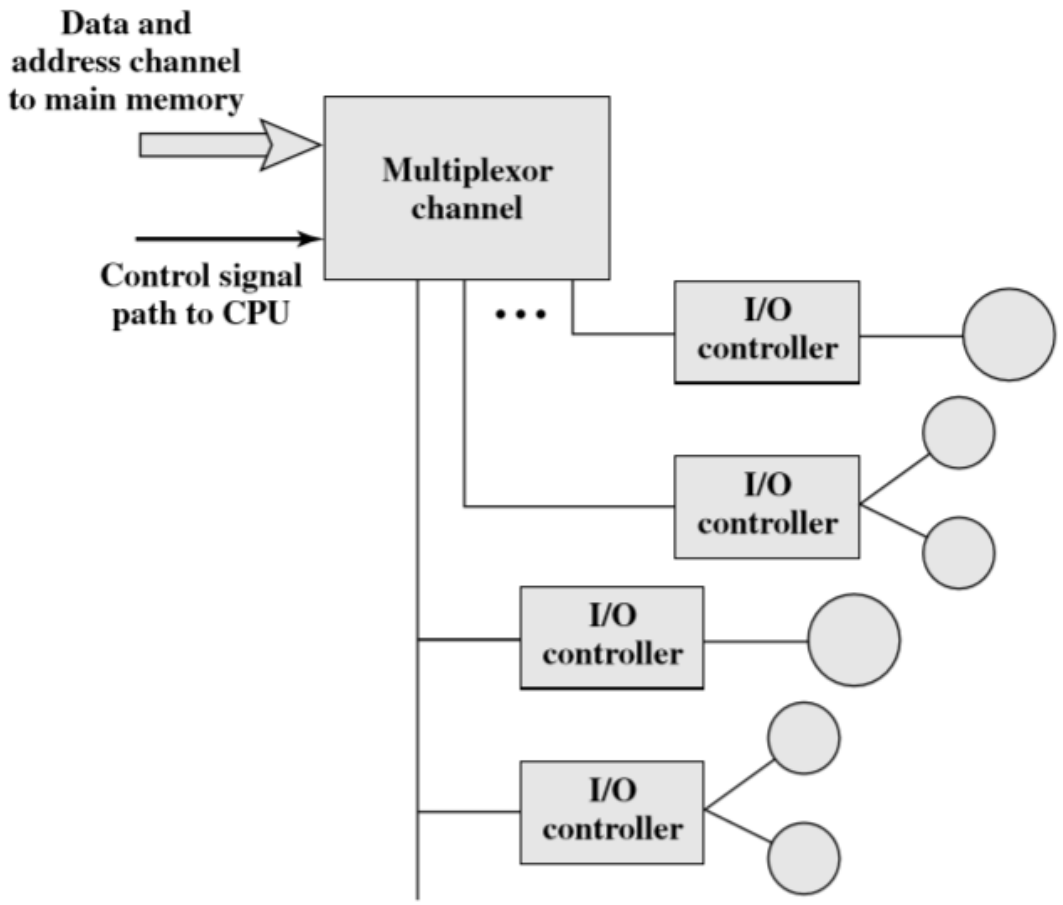


(a) Selector

Multiplexer

- A *multiplexor channel* can handle I/O with multiple devices at the same time. For low-speed devices, a *byte multiplexor* accepts or transmits characters as fast as possible to multiple devices. For example, the resultant character stream from three devices with different rates and individual streams A1A2A3A4 …, B1B2B3B4 …, and C1C2C3C4 … might be A1B1C1A2C2A3B2C3A4, and so on. For high-speed devices, a *block multiplexor* interleaves blocks of data from several devices

I/O Channel: Multiplexer(Contd.)



(b) Multiplexor

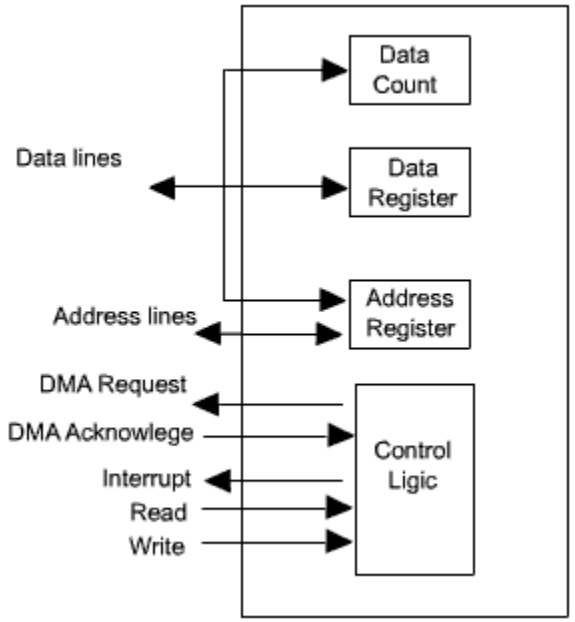
DMA: Direct Memory Access

- programmed I/O and Interrupt-driven I/O both the methods require the active intervention of the processor to transfer data between memory and the I/O module, and any data transfer must transverse a path through the processor. Thus both these forms of I/O suffer from two inherent drawbacks.
    - The I/O transfer rate is limited by the speed with which the processor can test and service a device.
    - The processor is tied up in managing an I/O transfer; a number of instructions must be executed for each I/O transfer.
- To transfer large block of data at high speed, a special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called direct memory access or DMA.

DMA: Direct Memory Access(contd.)

- DMA transfers are performed by a control circuit associated with the I/O device and this circuit is referred as DMA controller. The DMA controller allows direct data transfer between the device and the main memory without involving the processor.
- To transfer data between memory and I/O devices, DMA controller takes over the control of the system from the processor and transfer of data take place over the system bus. For this purpose, the DMA controller must use the bus only when the processor does not need it, or it must force the processor to suspend operation temporarily. The later technique is more common and is referred to as cycle stealing, because the DMA module in effect steals a bus cycle.

DMA: Direct Memory Access(contd.)



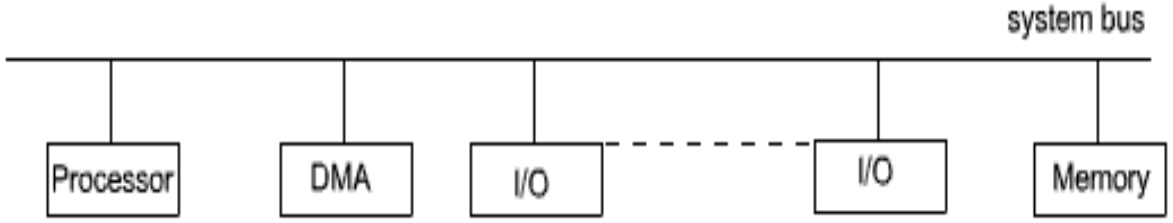
DMA: Direct Memory Access(contd.)

The DMA mechanism can be configured in different ways. The most common amongst them are:

- **Single bus, detached DMA - I/O configuration.**
- **Single bus, Integrated DMA - I/O configuration.**
- **Using separate I/O bus.**

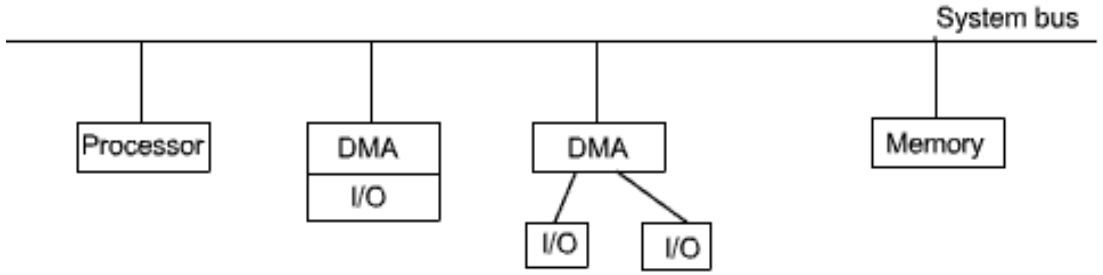
Single bus, detached DMA - I/O configuration

- In this organization all modules share the same system bus. The DMA module here acts as a surrogate processor. This method uses programmed I/O to exchange data between memory and an I/O module through the DMA module.
- For each transfer it uses the bus twice. The first one is when transferring the data between I/O and DMA and the second one is when transferring the data between DMA and memory. Since the bus is used twice while transferring data, so the bus will be suspended twice. The transfer consumes two bus cycle.
- The interconnection organization is shown in the Figure



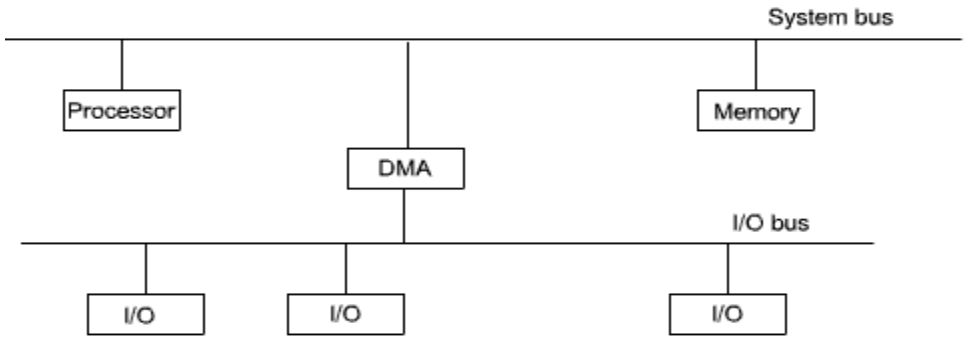
Single bus, Integrated DMA - I/O configuration

- By integrating the DMA and I/O function the number of required bus cycle can be reduced. In this configuration, the DMA module and one or more I/O modules are integrated together in such a way that the system bus is not involved. In this case DMA logic may actually be a part of an I/O module, or it may be a separate module that controls one or more I/O modules.
- The DMA module, processor and the memory module are connected through the system bus. In this configuration each transfer will use the system bus only once and so the processor is suspended only once.
- The system bus is not involved when transferring data between DMA and I/O device, so processor is not suspended. Processor is suspended when data is transferred between DMA and memory.
  The configuration is shown in the Figure



Using separate I/O bus

- In this configuration the I/O modules are connected to the DMA through another I/O bus. In this case the DMA module is reduced to one.
- Transfer of data between I/O module and DMA module is carried out through this I/O bus. In this transfer, system bus is not in use and so it is not needed to suspend the processor.
- There is another transfer phase between DMA module and memory. In this time system bus is needed for transfer and processor will be suspended for one bus cycle. The configuration is shown in the Figure



BUS INTERCONNECTION

- Bus Structure
- Multiple-Bus Hierarchies
- Elements of Bus Design

BUS INTERCONNECTION(Contd.)

- A bus is a communication pathway connecting two or more devices.
- A key characteristic of a bus is that it is a shared transmission medium.
- Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, their signals will overlap and become garbled. Thus, only one device at a time can successfully transmit. Typically, a bus consists of multiple communication pathways, or lines. Each line is capable of transmitting signals representing binary 1 and binary 0.
- Over time, a sequence of binary digits can be transmitted across a single line. Taken together, several lines of a bus can be used to transmit binary digits simultaneously (in parallel). For example, an 8-bit unit of data can be transmitted over eight bus lines.
- Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy.
- A bus that connects major computer components (processor, memory, I/O) is called a **system bus**.

Bus Structure

- A system bus consists, typically, of from about fifty to hundreds of separate lines. Each line is assigned a particular meaning or function. Although there are many different bus designs, on any bus the lines can be classified into three functional groups : data, address, and control lines.
- In addition, there may be power distribution lines that supply power to the attached modules.
- The **data lines** provide a path for moving data among system modules. These lines, collectively, are called the **data bus**. The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the *width* of the data bus.
- The **address lines** are used to designate the source or destination of the data on the data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines.
- The **control lines** are used to control the access to and the use of the data and address lines. Because the data and address lines are shared by all components, there must be a means of controlling their use. Control signals transmit both command and timing information among system modules.

Bus Structure(Contd.)

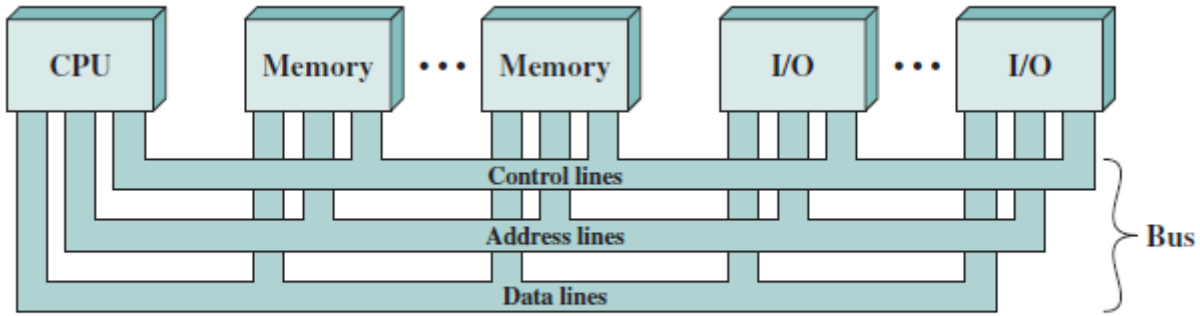


Figure 3.16 Bus Interconnection Scheme

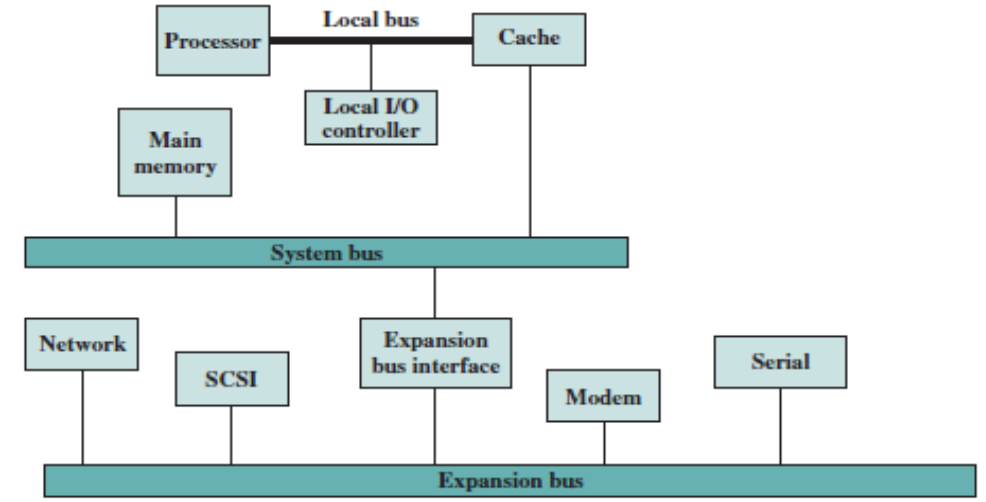
Multiple-Bus Hierarchies

- If a great number of devices are connected to the bus, performance will suffer. There are two main causes:

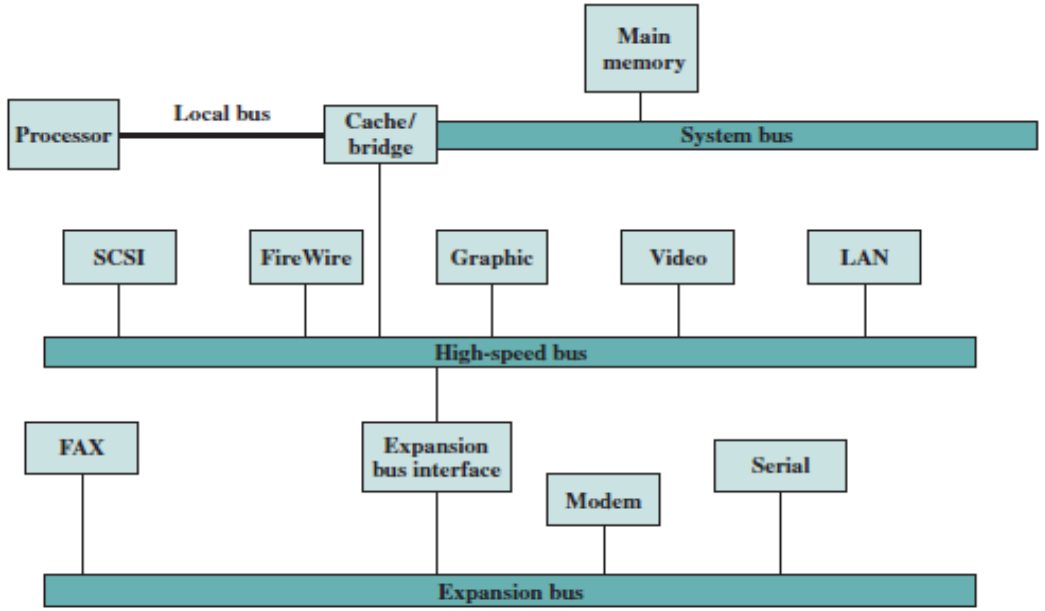
1. In general, the more devices attached to the bus, the greater the bus length and hence the greater the propagation delay. This delay determines the time it takes for devices to coordinate the use of the bus. When control of the bus passes from one device to another frequently, these propagation delays can noticeably affect performance.
2. The bus may become a bottleneck as the aggregate data transfer demand approaches the capacity of the bus. This problem can be countered to some extent by increasing the data rate that the bus can carry and by using wider buses (e.g., increasing the data bus from 32 to 64 bits). However, because the data rates generated by attached devices (e.g., graphics and video controllers, network interfaces) are growing rapidly, this is a race that a single bus is ultimately destined to lose.

Accordingly, most bus-based computer systems use multiple buses, generally laid out in a hierarchy.

Multiple-Bus Hierarchies(Contd.)



(a) Traditional bus architecture



(b) High-performance architecture

Figure 3.17 Example Bus Configurations

Elements of Bus Design

- Although a variety of different bus implementations exist, there are a few basic parameters or design elements that serve to classify and differentiate buses. Table lists key elements.

Type	**Bus Width**
Dedicated	Address
Multiplexed	Data
Method of Arbitration	**Data Transfer Type**
Centralized	Read
Distributed	Write
Timing	Read-modify-write
Synchronous	Read-after-write
Asynchronous	Block

Table: Elements of Bus design

Bus Arbitration

- **Bus Arbitration** refers to the process by which the current bus master accesses and then leaves the control of the bus and passes it to the another bus requesting processor unit. The controller that has access to a bus at an instance is known as **Bus master**.
- A conflict may arise if the number of DMA controllers or other controllers or processors try to access the common bus at the same time, but access can be given to only one of those. Only one processor or controller can be Bus master at the same point of time. To resolve these conflicts, Bus Arbitration procedure is implemented to coordinate the activities of all devices requesting memory transfers.
- 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. The **Bus Arbiter** decides who would become current bus master.

Bus Arbitration(Contd.)

Approaches of bus arbitration

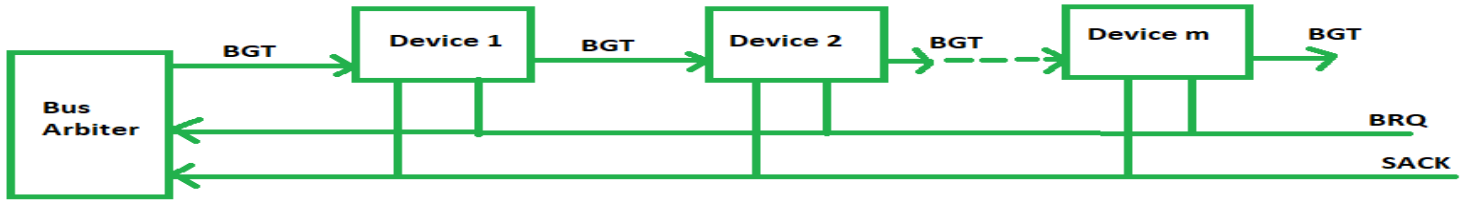
- **Centralized bus arbitration –** A single bus arbiter performs the required arbitration.
- **Distributed bus arbitration –** All devices participate in the selection of the next bus master.

Methods of BUS Arbitration

- **Daisy Chaining method**
- **Polling or Rotating Priority method**
- **Fixed priority or Independent Request method**

Daisy Chaining method

- It is a centralized bus arbitration method. During any bus cycle, the bus master may be any device – the processor or any DMA controller unit, connected to the bus.



Daisy chained bus arbitration

Advantages –

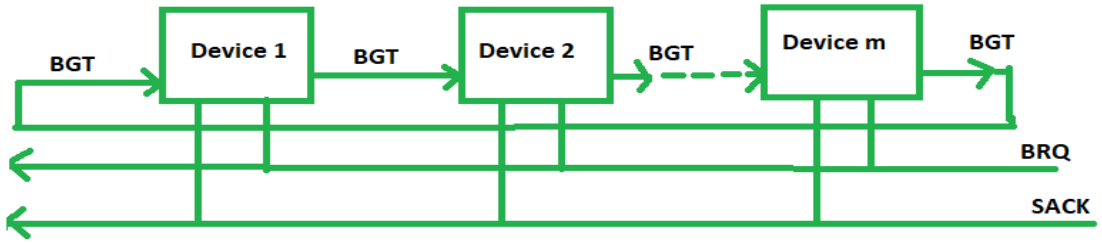
- Simplicity and Scalability.
- The user can add more devices anywhere along the chain, up to a certain maximum value.

Disadvantages –

- The value of priority assigned to a device is depends on the position of master bus.
- Propagation delay is arises in this method.
- If one device fails then entire system will stop working.

Polling or Rotating Priority method

In this method, the devices are assigned unique priorities and complete to access the bus, but the priorities are dynamically changed to give every device an opportunity to access the bus.



Rotating priority bus arbitration

Advantages –

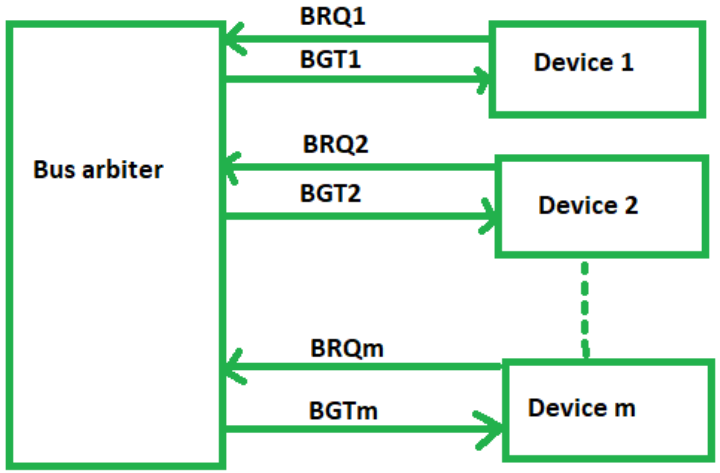
- This method does not favor any particular device and processor.
- The method is also quite simple.
- If one device fails then entire system will not stop working.

Disadvantages –

- Adding bus masters is different as increases the number of address lines of the circuit.

Fixed priority or Independent Request method

- In this method, the bus control passes from one device to another only through the centralized bus arbiter.



Fixed priority bus arbitration method

- **Advantages –**
- This method generates fast response.
- **Disadvantages –**
- Hardware cost is high as large no. of control lines are required.

PCI Bus

- The **peripheral component interconnect (PCI)** is a popular high-bandwidth, processor independent bus that can function as a mezzanine or peripheral bus.
- Compared with other common bus specifications, PCI delivers better system performance for highspeed I/O subsystems (e.g., graphic display adapters, network interface controllers, and disk controllers).
- Intel began work on PCI in 1990 for its Pentium-based systems. Intel soon released all the patents to the public domain and promoted the creation of an industry association, the PCI Special Interest Group (SIG), to develop further and maintain the compatibility of the PCI specifications. The result is that PCI has been widely adopted and is finding increasing use in personal computer, workstation, and server systems.

PCI Bus(Contd.)

- **Switch:** The switch manages multiple PCIe streams.
- **PCIe endpoint:** An I/O device or controller that implements PCIe, such as a Gigabit Ethernet switch, a graphics or video controller, disk interface, or a communications controller.
- **Legacy endpoint:** Legacy endpoint category is intended for existing designs that have been migrated to PCI Express, and it allows legacy behaviors such as use of I/O space and locked transactions.
- **PCIe/PCI bridge:** Allows older PCI devices to be connected to PCIe-based systems.

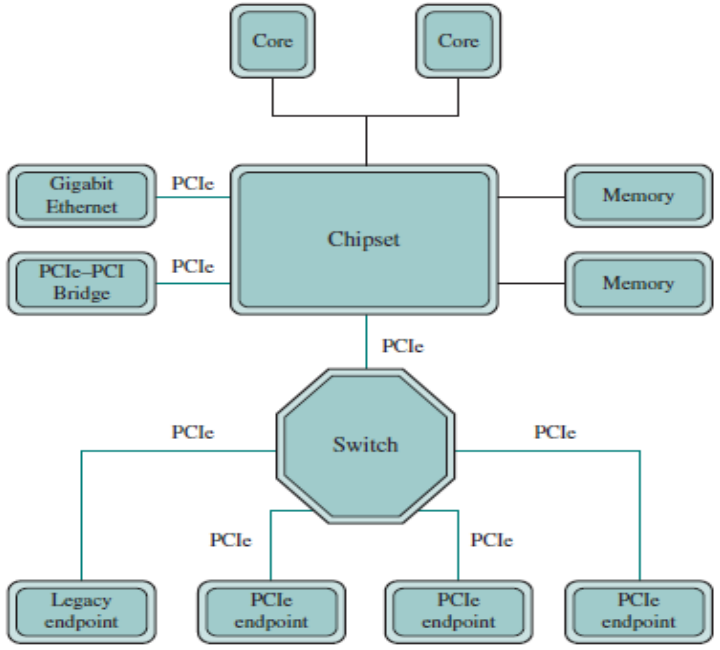


Figure 3.24 Typical Configuration Using PCIe

SCSI Bus

- The Small Computer Systems Interface (SCSI) bus is an ANSI standard for the interconnection of computers with each other and with disks, floppies, tapes, printers, optical disks, and scanners.
- The SCSI standard includes all the mechanical, electrical, and Data transfer rates are individually negotiated with each device attached to a given SCSI bus.
- For example, a 4 MB per second device and a 10 MB per second device may share a fast narrow bus. When the 4 MB per second device is using the bus, the transfer rate is 4 MB per second. When the 10 MB per second device is using the bus, the transfer rate is 10 MB per second.


THANK
You

BHARATI VIDYAPEETH (DEEMED TO BE UNIVERSITY)
COLLEGE OF ENGINEERING, PUNE
DEPARTMENT OF COMPUTER ENGINEERING

Memory Organization

Internal Memory

- Within internal memory we will consider the following points:

1. Semiconductor Main Memory
2. Error Correction
3. Advanced DRAM Organization

Semiconductor Main Memory

- Organization
- DRAM and SRAM
- Types of ROM
- Chip Logic
- Chip Packaging
- Module Organization
- Interleaved Memory

Semiconductor Main Memory: Organization

The basic element of a **semiconductor memory** is the memory cell. Although a variety of electronic technologies are used, all semiconductor memory cells share certain properties:

- They exhibit two stable (or semistable) states, which can be used to represent binary 1 and 0.
- They are capable of being written into (at least once), to set the state.
- They are capable of being read to sense the state.

Figure 5.1 depict the operation of memory cell

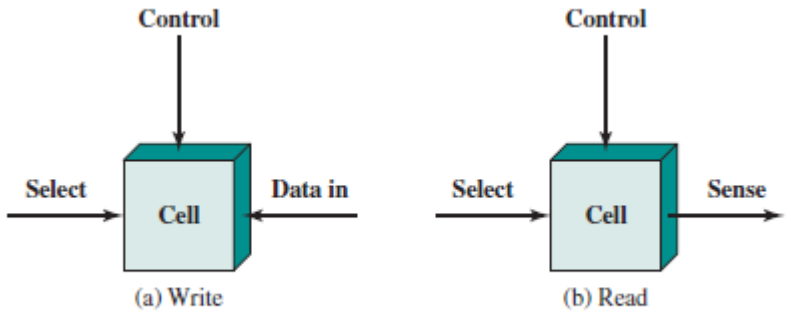


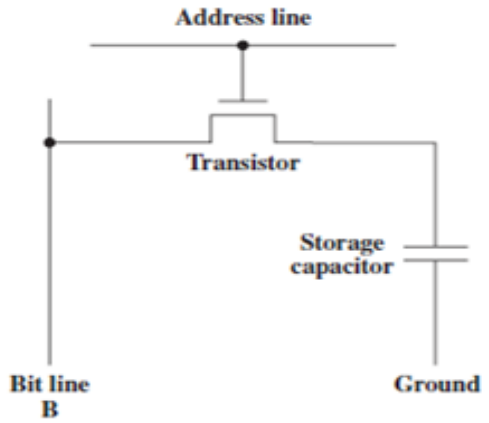
Figure 5.1 Memory Cell Operation

Semiconductor Main Memory: DRAM and SRAM

- One distinguishing characteristic of memory that is designated as RAM is that it is possible both to read data from the memory and to write new data into the memory easily and rapidly.
- Both the reading and writing are accomplished through the use of electrical signals.
- Other distinguishing characteristic of RAM is that it is volatile.
- A RAM must be provided with a constant power supply. If the power is interrupted, then the data are lost. Thus, RAM can be used only as temporary storage.
- The two traditional forms of RAM used in computers are DRAM and SRAM.

DYNAMIC RAM

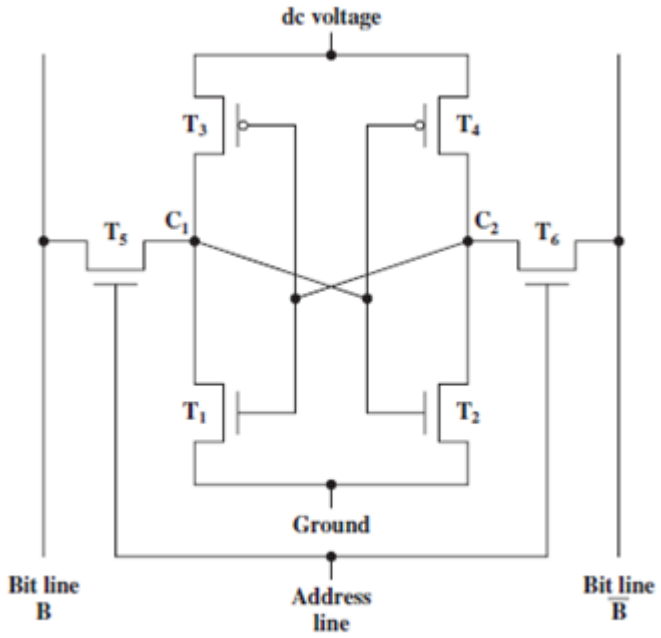
- A dynamic RAM (DRAM) is made with cells that store data as charge on capacitors.
- The presence or absence of charge in a capacitor is interpreted as a binary 1 or 0.
- Because capacitors have a natural tendency to discharge, dynamic RAMs require periodic charge refreshing to maintain data storage.
- The term dynamic refers to this tendency of the stored charge to leak away, even with power continuously applied.



(a) Dynamic RAM (DRAM) cell

STATIC RAM

- A static RAM (SRAM) is a digital device that uses the same logic elements used in the processor.
- In a SRAM, binary values are stored using traditional flip-flop logic-gate configurations. A static RAM will hold its data as long as power is supplied to it.



(b) Static RAM (SRAM) cell

SRAM VERSUS DRAM

- Both static and dynamic RAMs are volatile; that is, power must be continuously supplied to the memory to preserve the bit values.
- A dynamic memory cell is simpler and smaller than a static memory cell. Thus, a DRAM is more dense (smaller cells = more cells per unit area) and less expensive than a corresponding SRAM. On the other hand, a DRAM requires the supporting refresh circuitry.
- For larger memories, the fixed cost of the refresh circuitry is more than compensated for by the smaller variable cost of DRAM cells. Thus, DRAMs tend to be favored for large memory requirements.
- A final point is that SRAMs are somewhat faster than DRAMs. Because of these relative characteristics, SRAM is used for cache memory (both on and off chip), and DRAM is used for main memory.

Types of ROM

- A read-only memory(ROM) contains a permanent pattern of data that cannot be changed.
- A ROM is nonvolatile; that is, no power source is required to maintain the bit values in memory. While it is possible to read a ROM, it is not possible to write new data into it.
- A ROM is created like any other integrated circuit chip, with the data actually wired into the chip as part of the fabrication process. This presents two problems:

1. The data insertion step includes a relatively large fixed cost, whether one or thousands of copies of a particular ROM are fabricated.

2. There is no room for error. If one bit is wrong, the whole batch of ROMs must be thrown out.

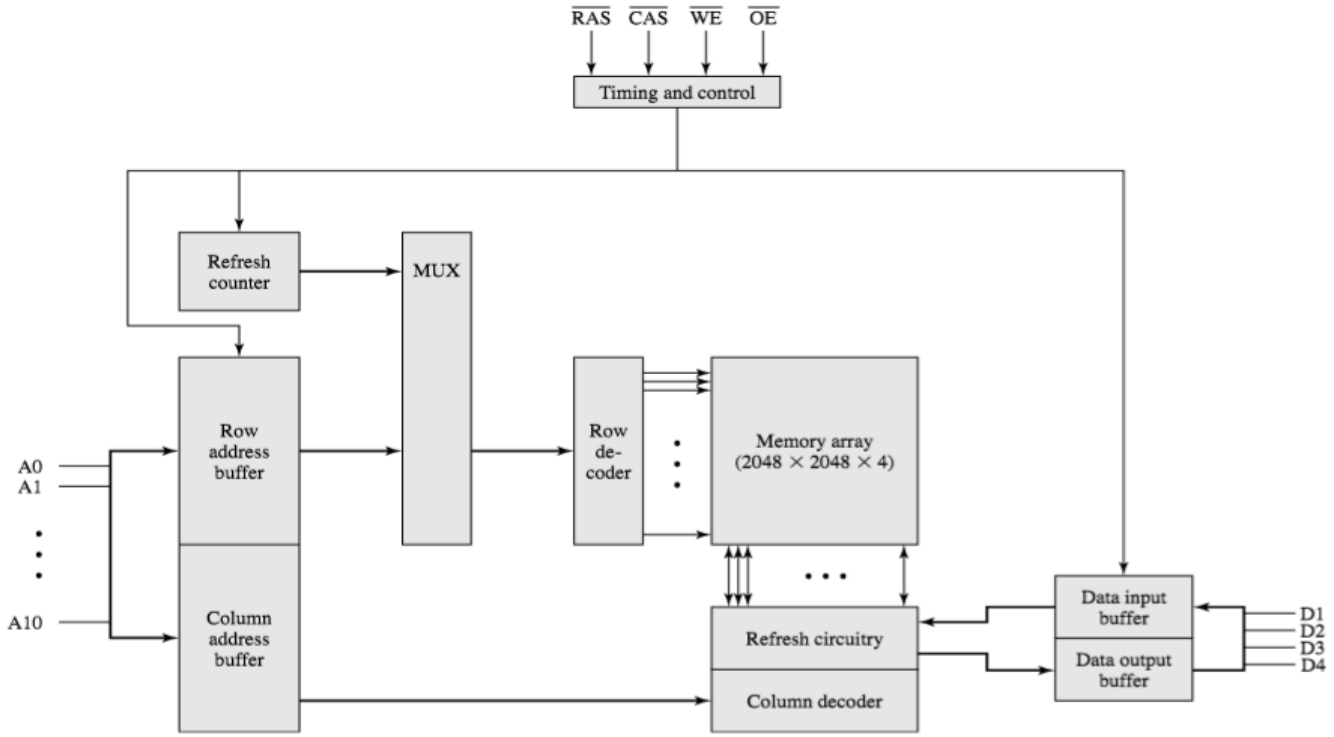
- When only a small number of ROMs with a particular memory content is needed, a less expensive alternative is the **programmable ROM (PROM)**. Like the ROM, the PROM is **nonvolatile** and may be written into only once.

Types of ROM(Contd.)

- For the PROM, the writing process is performed electrically and may be performed by a supplier or customer at a time later than the original chip fabrication. Special equipment is required for the writing or “programming” process. PROMs provide flexibility and convenience. The ROM remains attractive for high-volume production runs.
- Another variation on read-only memory is the **read-mostly memory**, which is useful for applications in which read operations are far more frequent than write operations but for which nonvolatile storage is required. There are three common forms of read-mostly memory: **EPROM, EEPROM,** and **flash memory.**

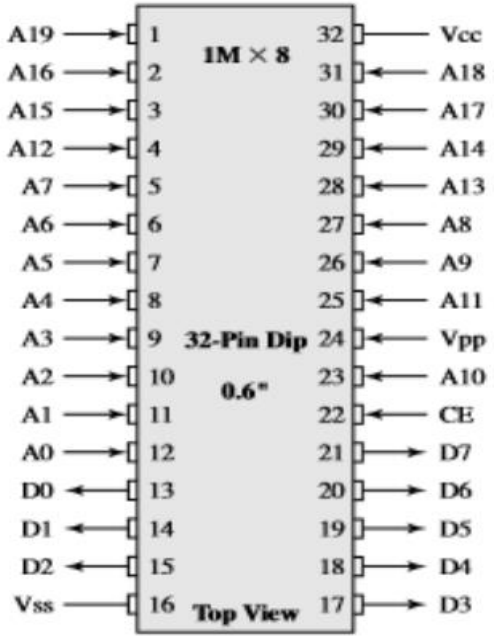
Chip Logic

- As with other integrated circuit products, semiconductor memory comes in packaged chips.
- Each chip contains an array of memory cells.

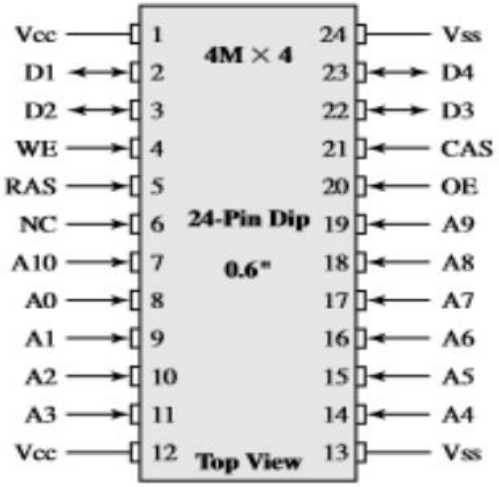


Chip Packaging

- An integrated circuit is mounted on a package that contains pins for connection to the outside world.
- Figure shows an example EPROM package, which is an 8-Mbit chip organized as 1M 8. In this case, the organization is treated as a one-word-per-chip package. The package includes 32 pins, which is one of the standard chip package sizes.



(a) 8-Mbit EPROM



(b) 16-Mbit DRAM

Module Organization

- If a RAM chip contains only 1 bit per word, then clearly we will need at least a number of chips equal to the number of bits per word. As an example, Figure shows how a memory module consisting of 256K 8-bit words could be organized. For 256K words, an 18-bit address is needed and is supplied to the module from some external source (e.g.,the address lines of a bus to which the module is attached).The address is presented to 8 256K 1-bit chips, each of which provides the input/output of 1 bit.

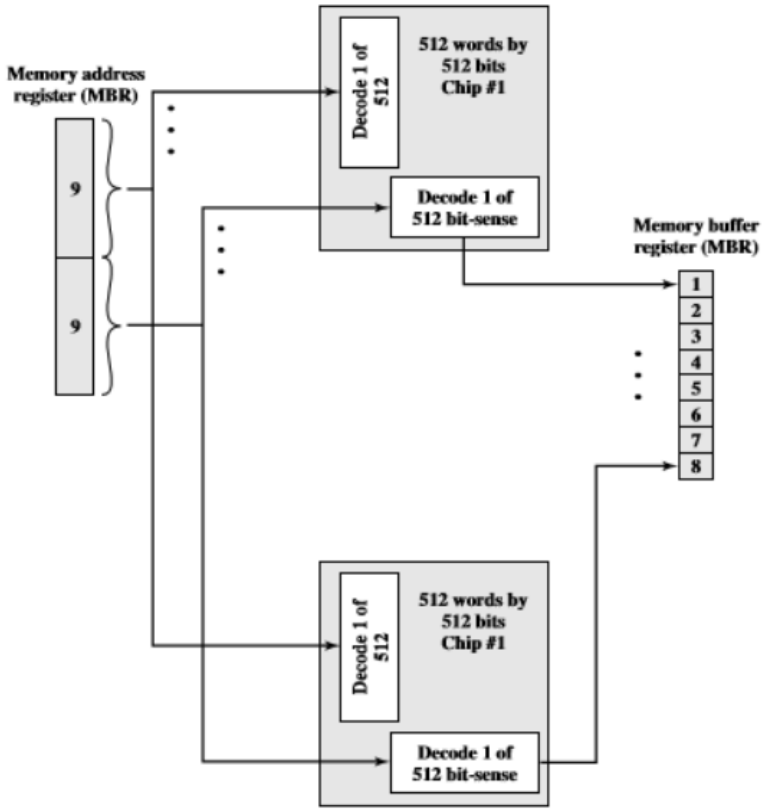


Figure 5.5 256-KByte Memory Organization

Interleaved Memory

- Main memory is composed of a collection of DRAM memory chips.
- A number of chips can be grouped together to form a memory bank.
- It is possible to organize the memory banks in a way known as interleaved memory.
- Each bank is independently able to service a memory read or write request, so that a system with K banks can service K requests simultaneously, increasing memory read or write rates by a factor of K. If consecutive words of memory are stored in different banks,then the transfer of a block of memory is speeded up.

ERROR CORRECTION

- A semiconductor memory system is subject to errors.
- These can be categorized as hard failures and soft errors.
- A **hard failure** is a permanent physical defect so that the memory cell or cells affected cannot reliably store data but become stuck at 0 or 1 or switch erratically between 0 and 1. Hard errors can be caused by harsh environmental abuse, manufacturing defects, and wear.
- A **soft error** is a random, nondestructive event that alters the contents of one or more memory cells without damaging the memory. Soft errors can be caused by power supply problems or alpha particles.
- Both hard and soft errors are clearly undesirable, and most modern main memory systems include logic for both detecting and correcting errors

ERROR CORRECTION(Contd.)

- Figure illustrates in general terms how the process is carried out.

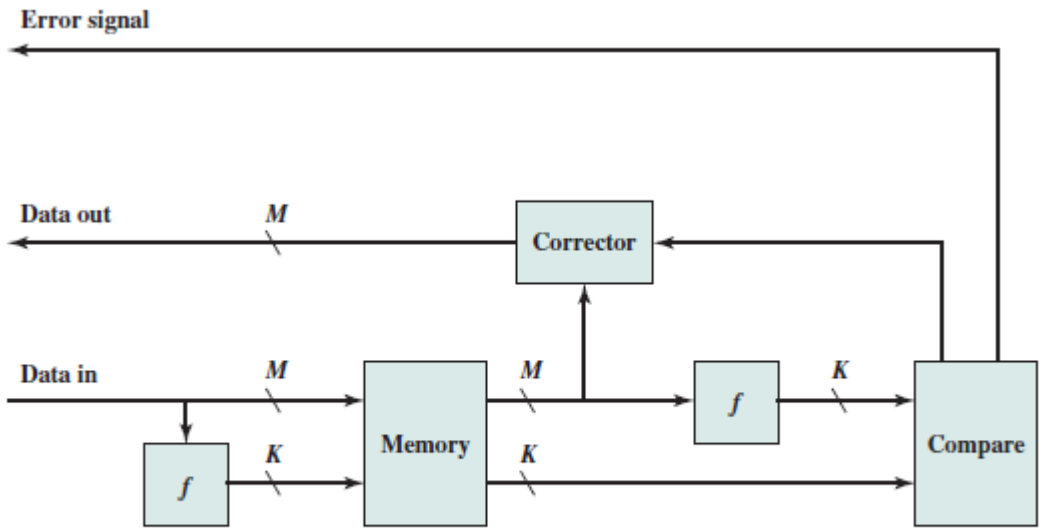


Figure 5.7 Error-Correcting Code Function

Codes that operate in the correction and detection are referred to as **error-correcting codes**. A code is characterized by the number of bit errors in a word that it can correct and detect.

ERROR CORRECTION(Contd.)

- The simplest of the error-correcting codes is the **Hamming code** devised by Richard Hamming at Bell Laboratories. Figure uses Venn diagrams to illustrate the use of this code on 4-bit words (*M* = 4). With three intersecting circles, there are seven compartments.

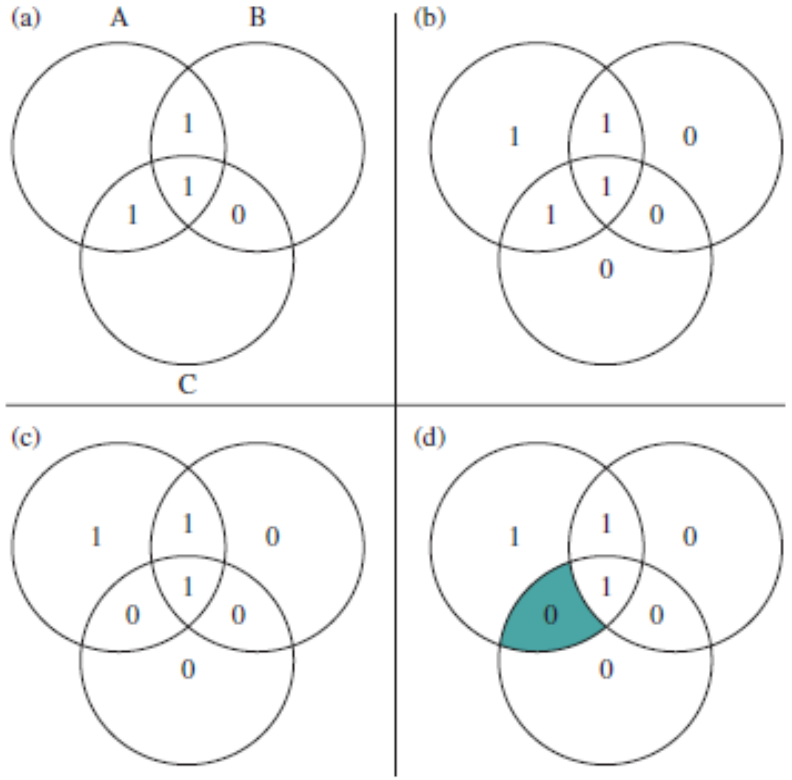


Figure 5.8 Hamming Error-Correcting Code

Characteristics of memory system

1) **Location**

CPU registers

Cache

Internal Memory(main)

External (secondary)

2) **Capacity**

The capacity of any memory device is expressed in terms of: i)word size ii)Number of words

Word size : Typically equal to the number of bits used to represent a number and to the instruction length.

For addresses of length A (in bits), the number of addressable units is **2^A.**

3) **Unit of Transfer**

- **Word**
- **Block**

4) **Access Method**

Sequential Access

- Access must be made in a specific linear sequence.
- Time to access an arbitrary record is highly variable.

Direct access

- Individual blocks or record have an address based on physical location.
- Access is by direct access to general vicinity of desired information, then some search.
- Access time is still variable, but not as much as sequential access.

Characteristics of memory system(Contd.)

*** Random access**

- Each addressable location has a unique, physical location.
- Access is by direct access to desired location,
- Access time is constant and independent of prior accesses.

*** Associative**

- Desired units of information are retrieved by comparing a sub-part of unit.
- Location is needed.
- Most useful for searching.

5) **Performance**

- *** Access Time (Latency)**

For random access memory, latency is the time it takes to perform. A read or write operation that is the time from the instant that the address is presented to the memory to the instant that the data have been stored available for use.

*** Memory Cycle Time**

- Access time + additional time required before a second access can begin(refresh time, for example).

*** Transfer Rate**

- Generally measured in bit/second.

The Memory Hierarchy

A variety of technologies are used to implement memory systems, and across this spectrum of technologies, the following relationships hold:

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access time

The dilemma facing the designer is clear.

- The way out of this dilemma is not to rely on a single memory component or technology, but to employ a **memory hierarchy**. As one goes down the hierarchy, the following occur:

a. Decreasing cost per bit

b. Increasing capacity

c. Increasing access time

d. Decreasing frequency of access of the memory by the processor

The Memory Hierarchy(Contd.)

Figure shows memory hierarchy

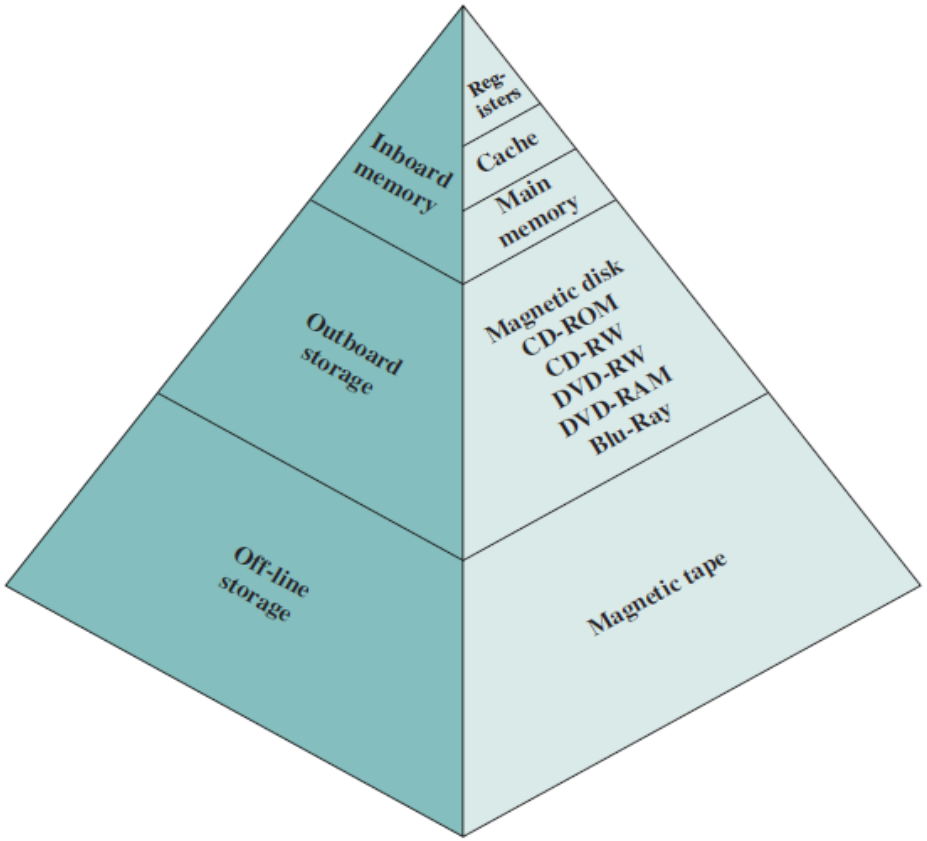
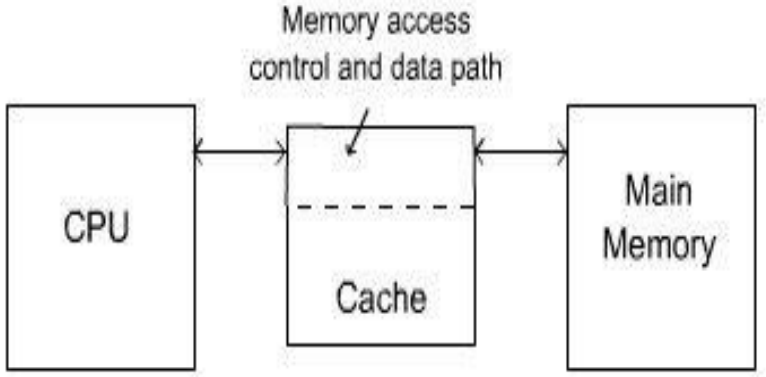


Figure 4.1 The Memory Hierarchy

Cache memory organization

- It is the fact that CPU is a faster device and memory is a relatively slower device.
- Memory access is the main bottleneck for the performance efficiency. If a faster memory device can be inserted between main memory and CPU, the efficiency can be increased. The faster memory that is inserted between CPU and Main Memory is termed as **Cache memory**.



Mapping

- The mapping functions are used to map a particular block of main memory to a particular block of cache. This mapping function is used to transfer the block from main memory to cache memory.

Direct mapping:

A particular block of main memory can be brought to a particular block of cache memory. So, it is not flexible.

Associative mapping:

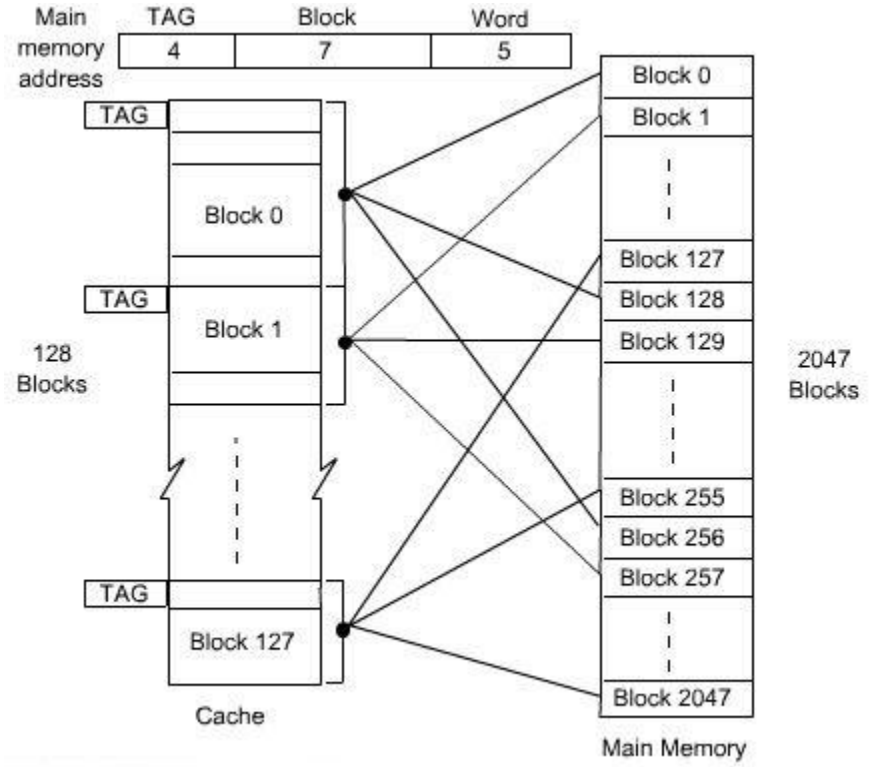
In this mapping function, any block of Main memory can potentially reside in any cache block position. This is much more flexible mapping method.

Block-set-associative mapping:

In this method, blocks of cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. From the flexibility point of view, it is in between to the other two methods.

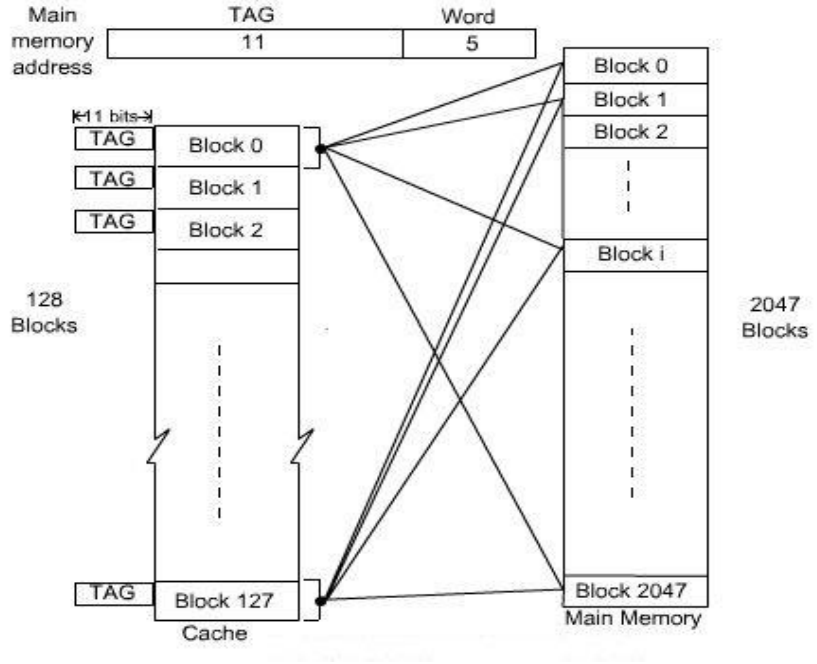
Direct Mapping

- The main memory address is divided into three fields. The field size depends on the memory capacity and the block size of cache. In this example, the lower 5 bits of address is used to identify a word within a block. Next 7 bits are used to select a block out of 128 blocks (which is the capacity of the cache). The remaining 4 bits are used as a TAG to identify the proper block of main memory that is mapped to cache.



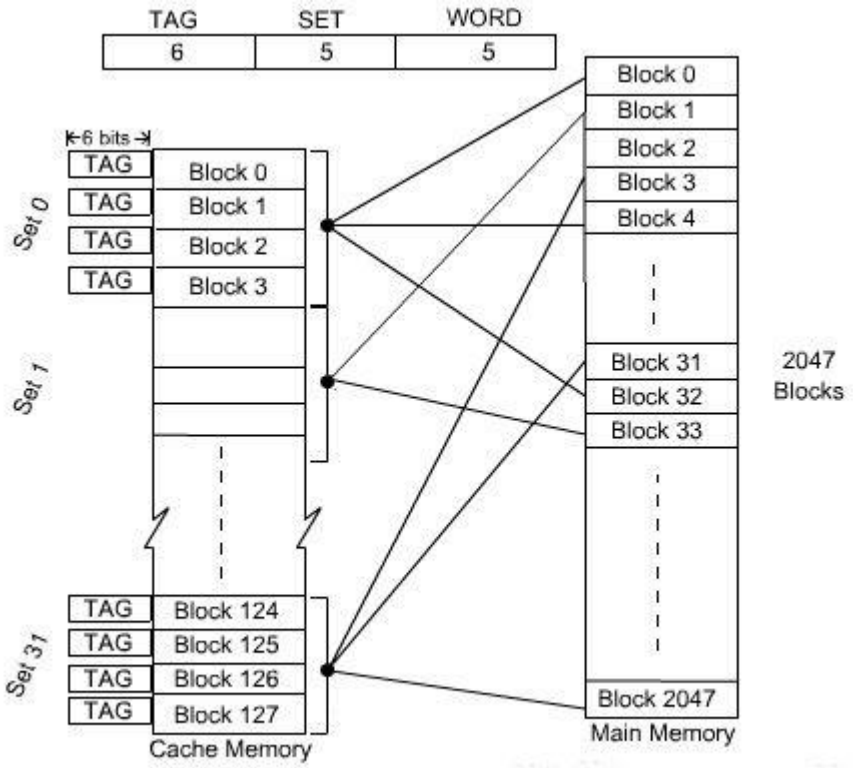
Associated Mapping Technique

- In the associative mapping technique, a main memory block can potentially reside in any cache block position. In this case, the main memory address is divided into two groups, low-order bits identifies the location of a word within a block and high-order bits identifies the block.
- In the example here, 11 bits are required to identify a main memory block when it is resident in the cache , high-order 11 bits are used as TAG bits and low-order 5 bits are used to identify a word within a block. The TAG bits of an address received from the CPU must be compared to the TAG bits of each block of the cache to see if the desired block is present.



Block-Set-Associative Mapping

- This mapping technique is intermediate to the previous two techniques. Blocks of the cache are grouped into sets, and the mapping allows a block of main memory to reside in any block of a specific set. Therefore, the flexibility of associative mapping is reduced from full freedom to a set of specific blocks.



Replacement Algorithms

- In an operating system that uses paging for memory management, a page replacement algorithm is needed to decide which page needs to be replaced when new page comes in.
- **Page Fault –** A page fault happens when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.
- Since actual physical memory is much smaller than virtual memory, page faults happen. In case of page fault, Operating System might have to replace one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.
- **Page Frame**: The main memory is divided into fixed sized portions called page frame

Replacement Algorithms(Contd.)

- **First In First Out (FIFO) –**
  This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.
- **Optimal Page replacement –**
  In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.
- **Least Recently Used –**
  In this algorithm page will be replaced which is least recently used.
- **Belady's anomaly –** Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm.

Pentium cache organization

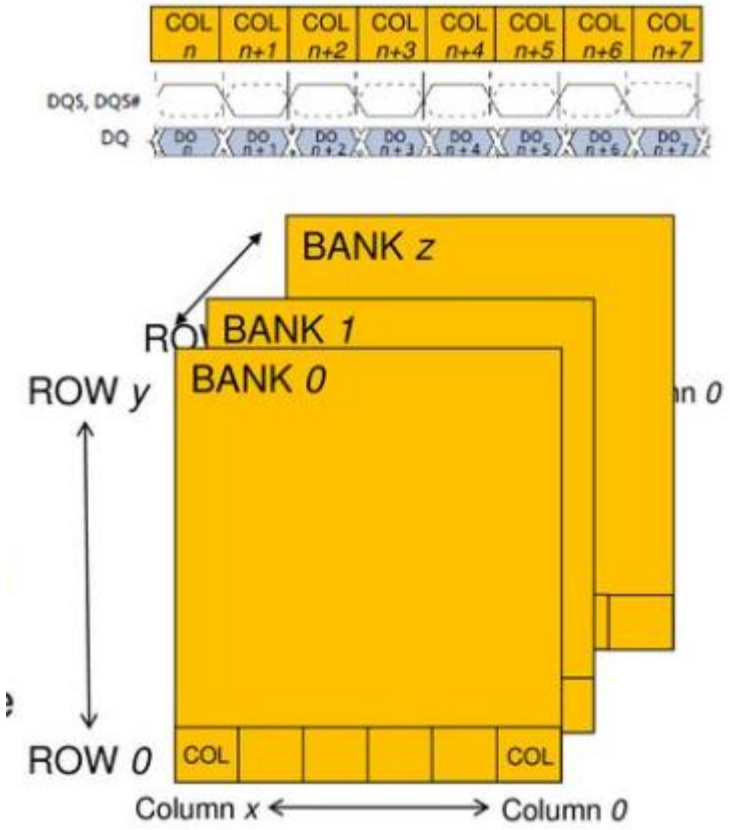
- L1 cache on the Pentium processor is 2-way set-associative in structure.
- In a set-associative structure the cache is divided into equal sections called cache ways.
- The cache page size is equal to the size of the cache way and each cache way is treated like a small direct mapped cache. In a 2-way scheme, two lines of memory may be stored at any time.
- The Pentium processor's cache line size is 32 bytes and is filled by a burst of four reads on the processor's 64-bit data bus. Each cache way contains 128 cache lines and the cache page size is 4K, or 128 lines.

DDR3 memory Organization

- DDR3 stands for **Double Data Rate Type 3**; it is kind of DRAM.
- Predecessor for DDR3 are DDR2 and DDR.
- It was released to the market in 2007 and today almost all computers and laptops in the market uses DDR3 as the RAM.
- The voltage specification for DDR is 1.5 V and, therefore, it consumes very less power when compared to its predecessors. DDR3 standard allows chips up to capacity 8 GB.

DDR3 memory Organization(Contd.)

- Each column is used to store one data word: Each read/write transfer consists of 8 adjacent words.
- Each row consists of multiple column: Active row is called page.
- Each bank consists of multiple rows: Each component consists of multiple banks.



Multiprocessors and Multicomputers

We will discuss two types of parallel computers −

- Multiprocessors
- Multicomputers

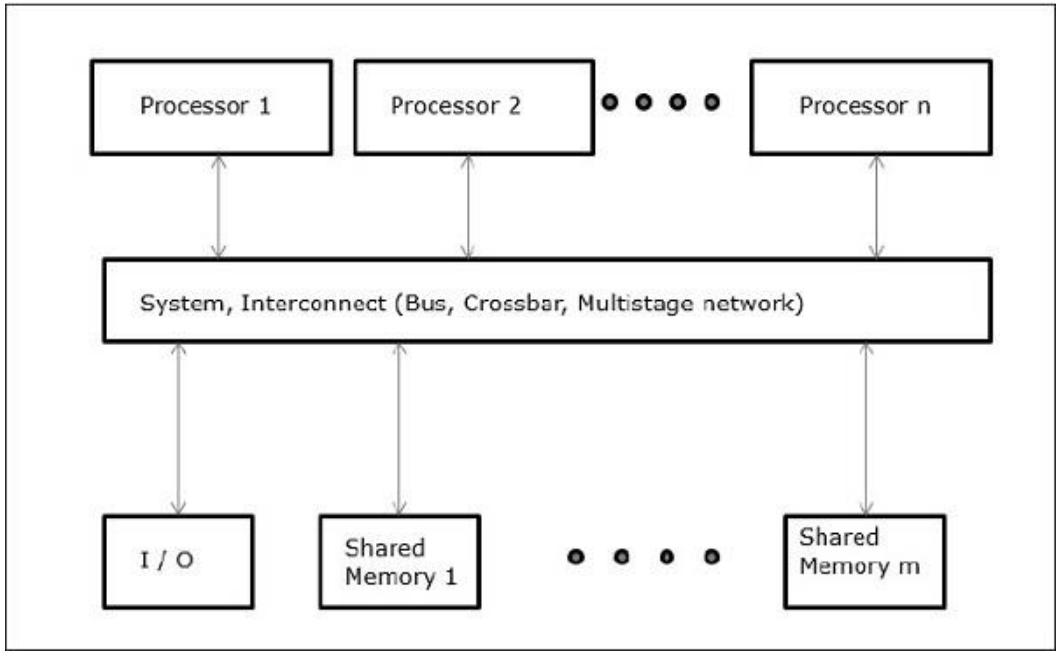
Shared-Memory Multicomputers

- Three most common shared memory multiprocessors models are

1. Uniform Memory Access (UMA)
2. Non-uniform Memory Access (NUMA)
3. Cache Only Memory Architecture (COMA)

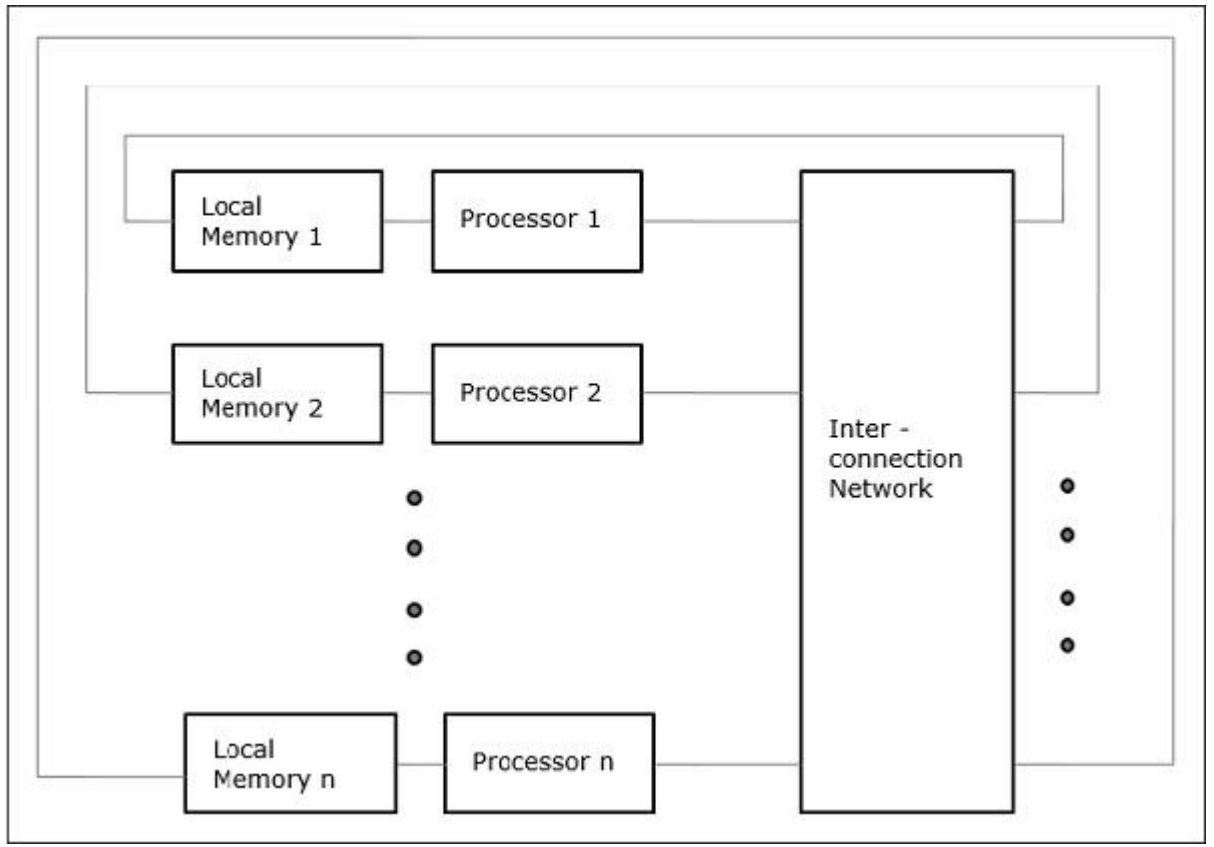
Uniform Memory Access (UMA)

- In this model, all the processors share the physical memory uniformly. All the processors have equal access time to all the memory words. Each processor may have a private cache memory. Same rule is followed for peripheral devices.
- When all the processors have equal access to all the peripheral devices, the system is called a **symmetric multiprocessor**. When only one or a few processors can access the peripheral devices, the system is called an **asymmetric multiprocessor**.



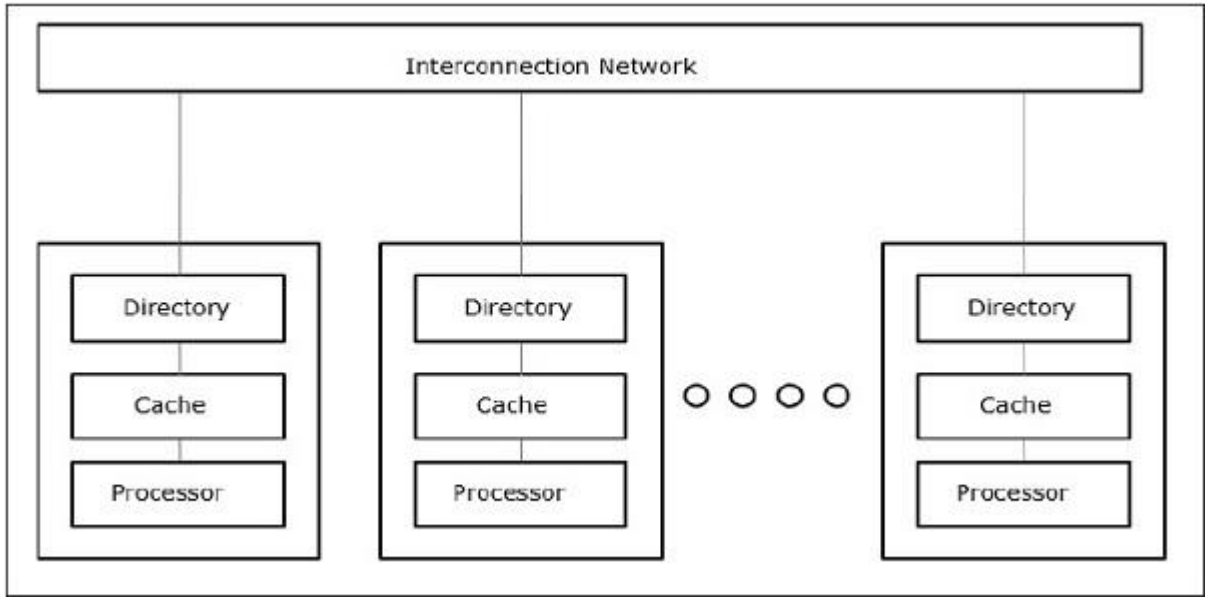
Non-uniform Memory Access (NUMA)

- In NUMA multiprocessor model, the access time varies with the location of the memory word. Here, the shared memory is physically distributed among all the processors, called local memories. The collection of all local memories forms a global address space which can be accessed by all the processors.



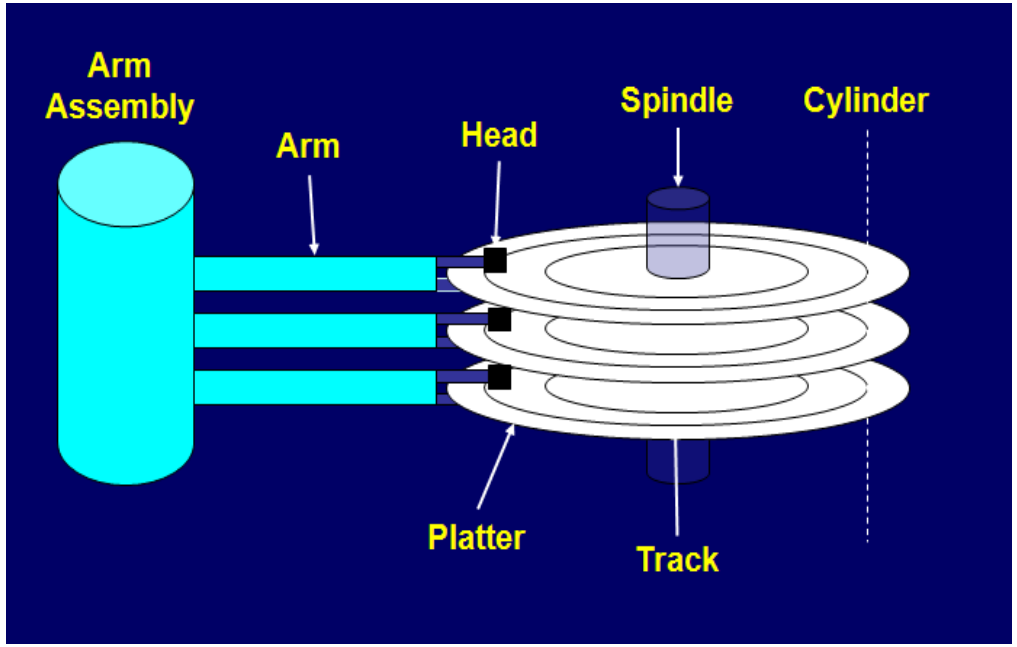
Cache Only Memory Architecture (COMA)

The COMA model is a special case of the NUMA model. Here, all the distributed main memories are converted to cache memories.



Hard Disk Drive

- Hard disk drive is made up of a series of circular disks called **platters** arranged one over the other almost ½ inches apart around a **spindle**. Disks are made of non-magnetic material like aluminum alloy and coated with 10-20 nm of magnetic material.
- Standard diameter of these disks is 14 inches and they rotate with speeds varying from 4200 rpm (rotations per minute) for personal computers to 15000 rpm for servers. Data is stored by magnetizing or demagnetizing the magnetic coating. A magnetic reader arm is used to read data from and write data to the disks. A typical modern HDD has capacity in terabytes (TB).

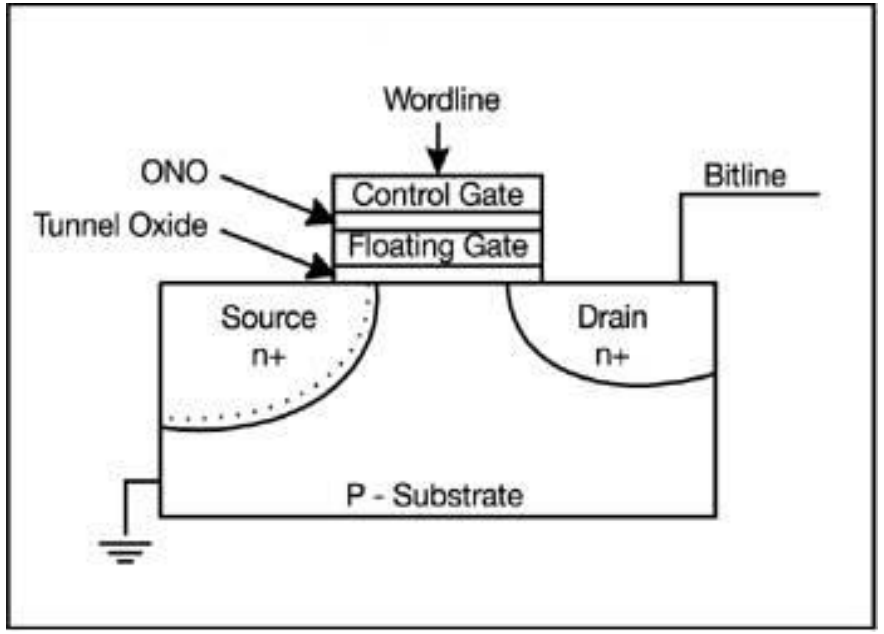


Flash Drives

- Flash memory is a form of EEPROM (Electrically Erasable Programmable Read-Only Memory) that allows multiple memory locations to be erased or written in one programming operation.
- Flash memory is non-volatile, that no power needed to maintain the information stored in the chip.
- The information is stored in Flash memory as an array of floating gate transistors, called "cells", each of which traditionally stores one bit of information. Newer flash memory devices, sometimes referred to as multi-level cell devices, can store more than 1 bit per cell, by varying the number of electrons placed on the floating gate of a cell.

Structure and Operation of Flash Memory

- Flash memory uses memory cells similar to an EPROM, but with a much thinner, precisely grown oxide between the floating gate and the source. The flash memory cell functions by storing charge in the floating gate.
- The presence of charge will then determine whether the channel will conduct or not. During the read cycle a "1" at the output corresponds to the channel being in its low resistance or ON state. The Control gate is used to charge up the gate capacitance during the write cycle.



RAID levels

- RAID or **R**edundant **A**rray of **I**ndependent **D**isks, is a technology to connect multiple secondary storage devices and use them as a single storage media.
- RAID consists of an array of disks in which multiple disks are connected together to achieve different goals. RAID levels define the use of disk arrays.
- The term was coined by David Patterson, Garth A. Gibson, and Randy Katz at the University of California, Berkeley in 1987.

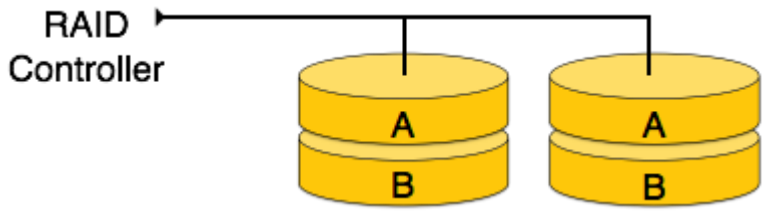
RAID 0

- In this level, a striped array of disks is implemented. The data is broken down into blocks and the blocks are distributed among disks.

RAID levels(Contd.)

RAID 1

- RAID 1 uses mirroring techniques. When data is sent to a RAID controller, it sends a copy of data to all the disks in the array. RAID level 1 is also called mirroring and provides 100% redundancy in case of a failure.



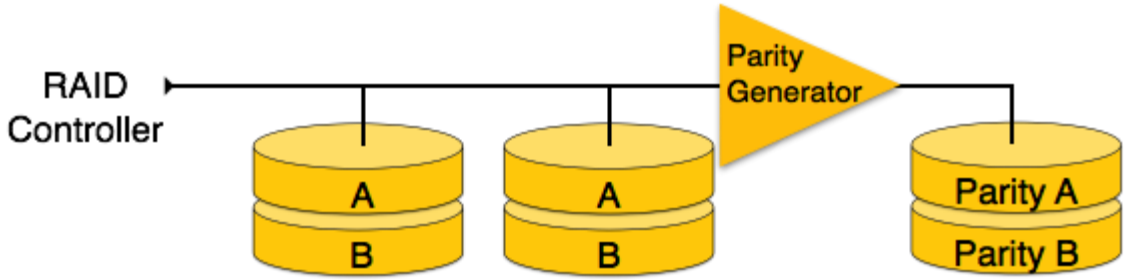
RAID 2

- RAID 2 records Error Correction Code using Hamming distance for its data, striped on different disks. Like level 0, each data bit in a word is recorded on a separate disk and ECC codes of the data words are stored on a different set disks. Due to its complex structure and high cost, RAID 2 is not commercially available.

RAID levels(Contd.)

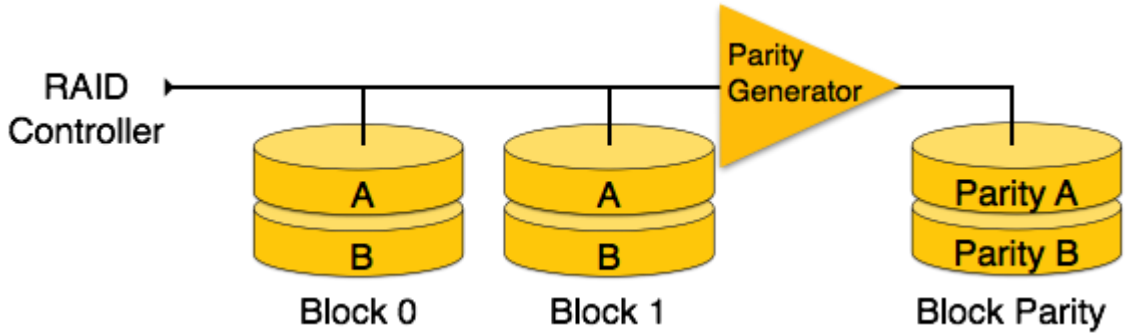
RAID 3

- RAID 3 stripes the data onto multiple disks. The parity bit generated for data word is stored on a different disk. This technique makes it to overcome single disk failures.



RAID 4

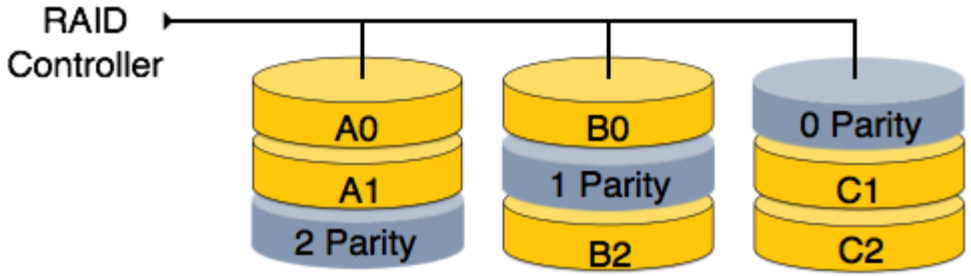
- In this level, an entire block of data is written onto data disks and then the parity is generated and stored on a different disk. Note that level 3 uses byte-level striping, whereas level 4 uses block-level striping. Both level 3 and level 4 require at least three disks to implement RAID.



RAID levels(Contd.)

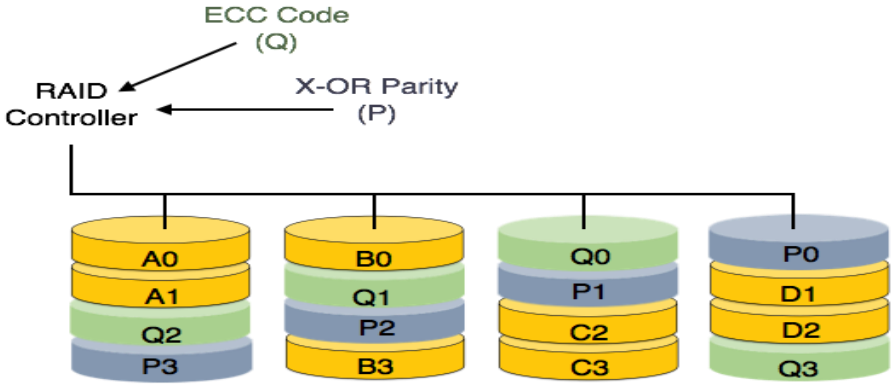
RAID 5

- RAID 5 writes whole data blocks onto different disks, but the parity bits generated for data block stripe are distributed among all the data disks rather than storing them on a different dedicated disk.



RAID 6

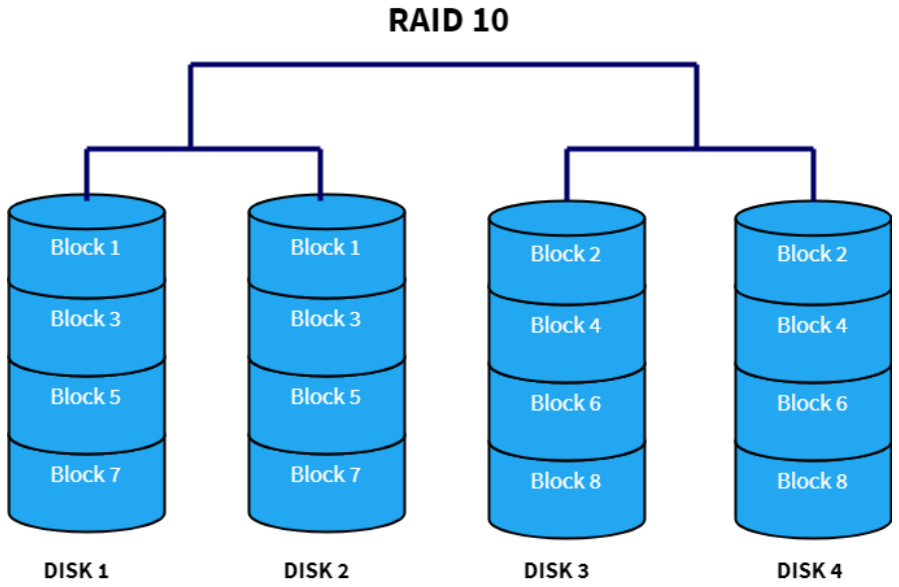
- RAID 6 is an extension of level 5. In this level, two independent parities are generated and stored in distributed fashion among multiple disks. Two parities provide additional fault tolerance. This level requires at least four disk drives to implement RAID.



RAID levels(Contd.)

RAID 10 (RAID 1+0)

- RAID 10 combines both RAID 1 and RAID 0 by layering them in opposite order. Sometimes, it is also called as “nested” or “hybrid” RAID. This is a “best of both worlds approach”, because it has the fast performance of RAID 0 and the redundancy of RAID 1. In this setup, multiple RAID 1 blocks are connected with each other to make it like RAID 0. It is used in cases where huge disk performance (greater than RAID 5 or 6) along with redundancy is required.



- A USB flash drive -- also known as a USB stick, USB thumb drive or pen drive -- is a plug-and-play portable storage device that uses flash memory and is lightweight enough to attach to a keychain.
- A USB flash drive can be used in place of a compact disc. When a user plugs the flash memory device into the USB port, the computer's operating system (OS) recognizes the device as a removable drive and assigns it a drive letter.


THANK
YOU

UNIT-I

CPU structure and function:

Components and functions of computer system, CPU architecture, Processor organization, Register Organization, Instruction Cycle, instruction pipeline. RISC and CISC architecture, The Pentium Processor, Power PC., Superscalar processors.

Components and functions of computer system

Virtually all contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton. Such a design is referred to as the *von Neumann architecture* and is based on three key concepts:

- Data and instructions are stored in a single read–write memory.
- The contents of this memory are addressable by location, without regard to the type of data contained there.
- Execution occurs in a sequential fashion (unless explicitly modified) from one instruction to the next.

There is a small set of basic logic components that can be combined in various ways to store binary data and to perform arithmetic and logical operations on that data. If there is a particular computation to be performed, a configuration of logic components designed specifically for that computation could be constructed. We can think of the process of connecting the various components in the desired configuration as a form of programming. The resulting "program" is in the form of hardware and is termed a *hardwired program.* Now consider this alternative. Suppose we construct a general-purpose configuration of arithmetic and logic functions. This set of hardware will perform various functions on data depending on control signals applied to the hardware. In the original case of customized hardware, the system accepts data and produces results (Figure 3.1a).With general-purpose hardware, the system accepts data and control signals and produces results. Thus, instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals. How shall control signals be supplied? The answer is simple but subtle. The entire program is actually a sequence of steps. At each step, some arithmetic or logical operation is performed on some data. For each step, a new set of control signals is needed. Let us provide a unique code for each possible set of control signals, and let
us add to the general-purpose hardware a segment that can accept a code and generate control signals (Figure 3.1b). Programming is now much easier. Instead of rewiring the hardware for each new program, all we need to do is provide a new sequence of codes. Each code is, in effect, an instruction, and part of the hardware interprets each instruction and generates control signals. To distinguish this new method of programming, a sequence of codes or instructions is called *software.* Figure 3.1b indicates two major components of the system: an instruction interpreter and a module of general-purpose arithmetic and logic functions. These two constitute the CPU. Several other components are needed to yield a functioning computer. Data and instructions must be put into the system. For this we need some sort of input module. This module contains basic components for accepting data and instructions in some form and converting them into an internal form of signals usable by the system. A means of reporting results is needed, and this is

in the form of an output module. Taken together, these are referred to as *I/O components.* One more component is needed. An input device will bring instructions and data in sequentially. But a program is not invariably executed sequentially; it may jump around (e.g., the IAS jump instruction). Similarly, operations on data may require access to more than just one element at a time in a predetermined sequence. Thus, there must be a place to store temporarily both instructions and data. That module is called *memory,* or *main memory* to distinguish it from external storage or peripheral devices. Von Neumann pointed out that the same memory could be used to store both instructions and data. Figure 3.2 illustrates these top-level components and suggests the interactions among them. The CPU exchanges data with memory. For this purpose, it typically makes use of two internal (to the CPU) 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. Similarly, an I/O address register (I/OAR) specifies a particular I/O device. An I/O buffer (I/OBR) register 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.

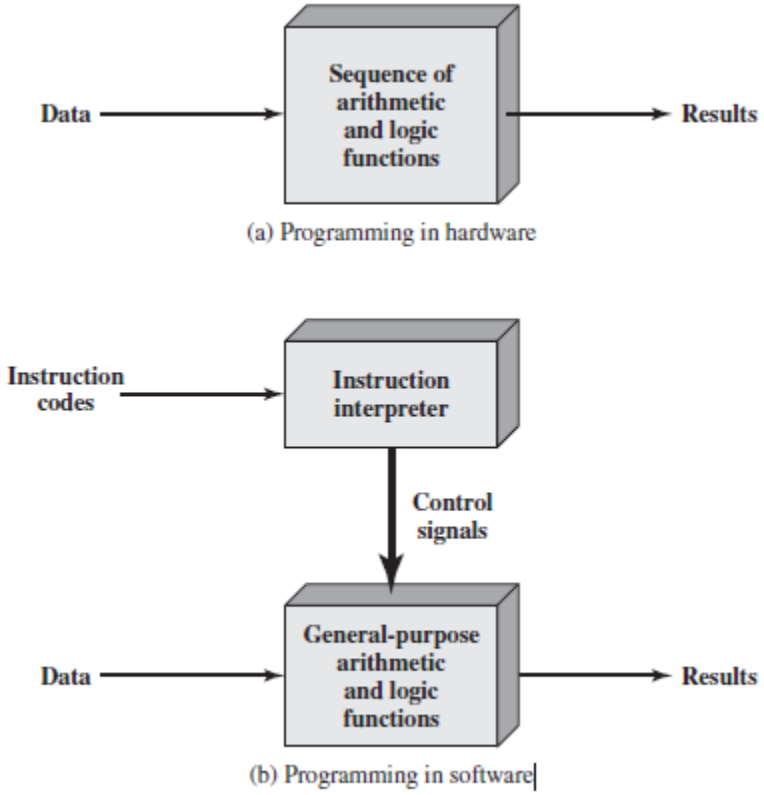


Figure 3.1 Hardware and Software Approaches

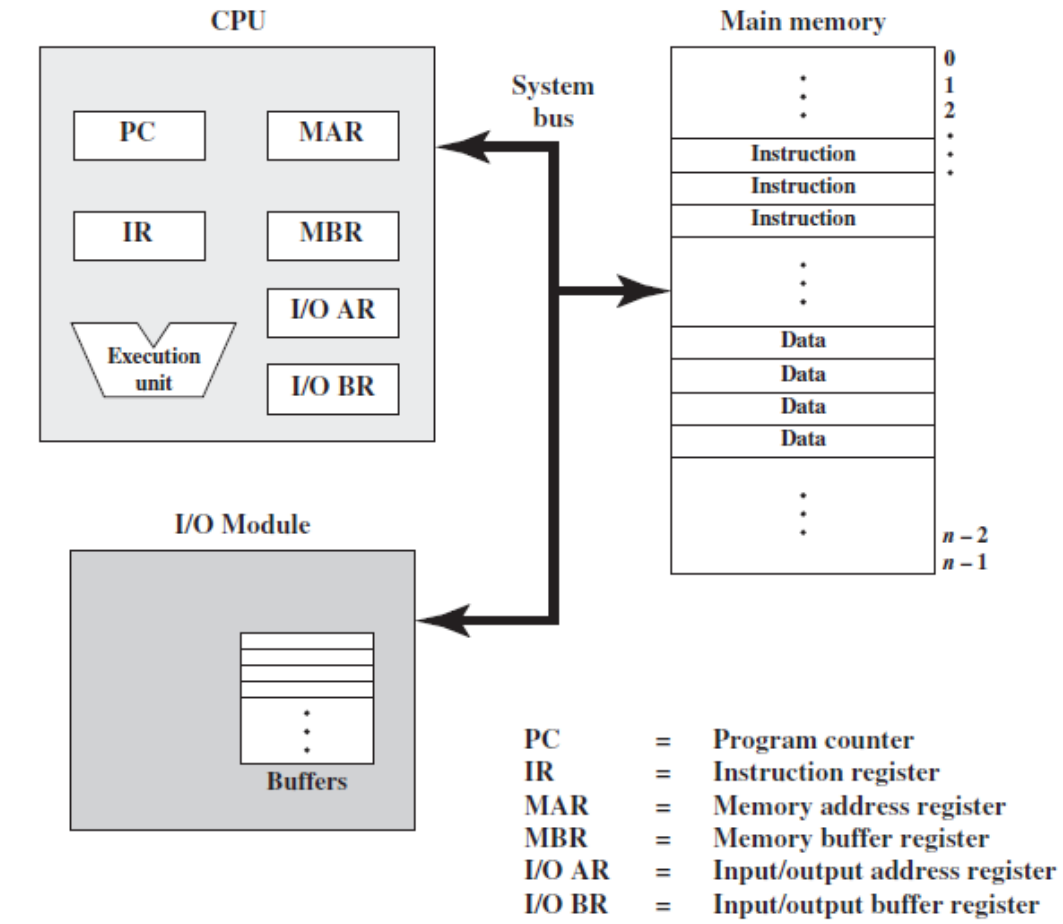


Figure 3.2 Computer Components:Top-Level View

Computer Functions

The basic function performed by a computer is execution of a program, which consists of a set of instructions stored in memory. The processor does the actual work by executing instructions specified in the program.

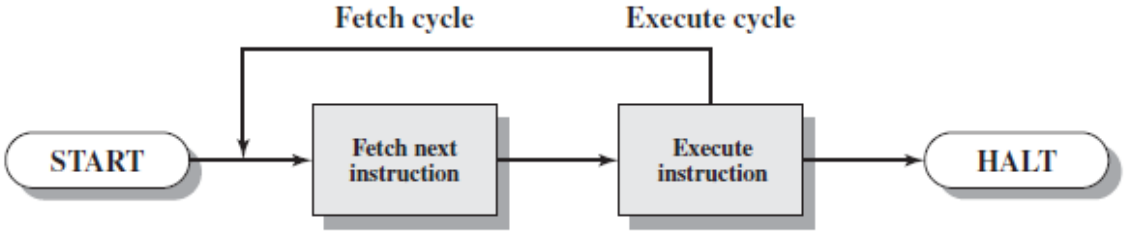


Figure 3.3 Basic Instruction Cycle

At the beginning of each instruction cycle, the processor fetches an instruction from memory. In a typical processor, a register called the program counter (PC) holds the address of the instruction to be fetched next. Unless told otherwise, the processor always increments the PC after each instruction fetch so that it will fetch the next instruction in sequence (i.e., the instruction located at the next higher memory address).

So, for example, consider a computer in which each instruction occupies one 16-bit word of memory. Assume that the program counter is set to location 300.The processor will next fetch the instruction at location 300. On succeeding instruction cycles, it will fetch instructions from locations 301, 302, 303, and so on. This sequence may be altered, as explained presently. The fetched instruction is loaded into a register in the processor known as the instruction register (IR). The instruction contains bits that specify the action the processor is to take. The processor interprets the instruction and performs the required
action. In general, these actions fall into four categories:

- **Processor-memory:** Data may be transferred from processor to memory or from memory to processor.
- **Processor-I/O:** Data may be transferred to or from a peripheral device by transferring between the processor and an I/O module.
- **Data processing:** The processor may perform some arithmetic or logic operation on data.
- **Control:** An instruction may specify that the sequence of execution be altered. For example, the processor may fetch an instruction from location 149, which specifies that the next instruction be from location 182. The processor will remember this fact by setting the program counter to 182.Thus, on the next fetch cycle, the instruction will be fetched from location 182 rather than 150.

Processor Organization

To understand the organization of the processor, let us consider the requirements placed on the processor, the things that it must do:

- **Fetch instruction:** The processor reads an instruction from memory (register, cache, main memory).
- **Interpret instruction:** The instruction is decoded to determine what action is required.
- **Fetch data:** The execution of an instruction may require reading data from memory or an I/O module.
- **Process data:** The execution of an instruction may require performing some arithmetic or logical operation on data.
- **Write data:** The results of an execution may require writing data to memory or an I/O module.

To do these things, it should be clear that the processor needs to store some data temporarily. It must remember the location of the last instruction so that it can know where to get the next instruction. It needs to store instructions and data temporarily while an instruction is being executed. In other words, the processor needs a small internal memory. Figure 12.1 is a simplified view of a processor, indicating its connection to the rest of the system via the system bus. A similar interface would be needed for any of
the interconnection structures described in Chapter 3.The reader will recall that the major components of the processor are an *arithmetic and logic unit* (ALU) and a *control unit* (CU).The ALU does the actual computation or processing of data. The control unit controls the movement of data and instructions into and out of the processor and controls the operation of the ALU. In addition, the figure shows a minimal internal memory, consisting of a set of storage locations, called *registers.* Figure 12.2 is a slightly more detailed view of the processor. The data transfer and logic control paths are indicated, including an element labeled *internal processor bus.* This

element is needed to transfer data between the various registers and the ALU because the ALU in fact operates only on data in the internal processor
memory. The figure also shows typical basic elements of the ALU. Note the similarity between the internal structure of the computer as a whole and the internal structure of the processor. In both cases, there is a small collection of major elements (computer: processor, I/O, memory; processor: control unit ,ALU, registers) connected by data paths.

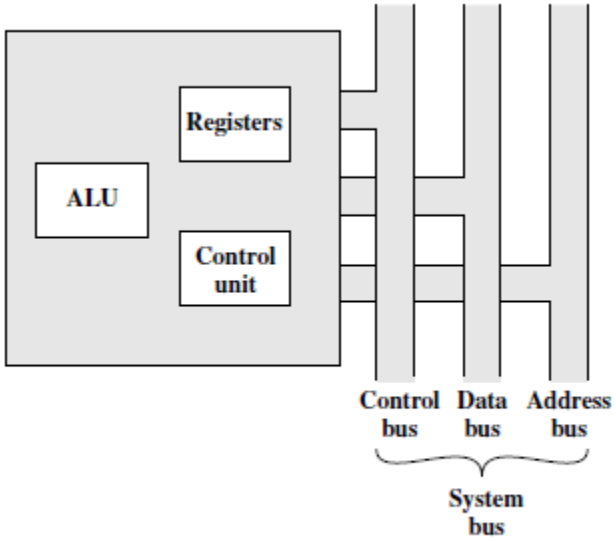


Figure 12.1 The CPU with the System Bus

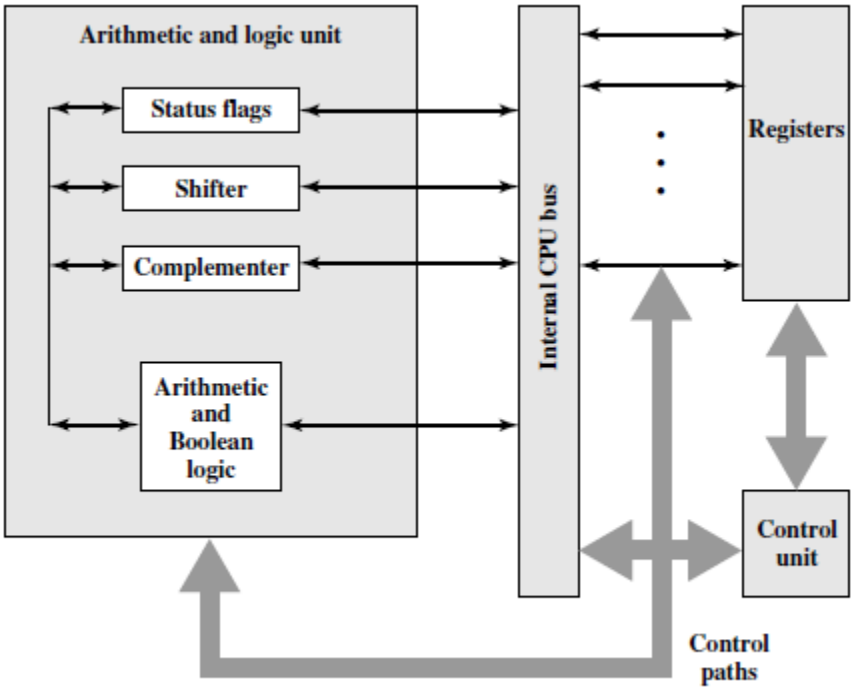


Figure 12.2 Internal Structure of the CPU

Register Organization

Within the processor, there is a set of registers that function as a level of memory above main memory and cache in the hierarchy. The registers in the processor perform two roles:

- **User-visible registers:** Enable the machine- or assembly language programmer to minimize main memory references by optimizing use of registers.
- **Control and status registers:** Used by the control unit to control the operation of the processor and by privileged, operating system programs to control the execution of programs.

There is not a clean separation of registers into these two categories. For example, on some machines the program counter is user visible (e.g., x86), but on many it is not. For purposes of the following discussion, however, we will use these categories.

User-Visible Registers

A user-visible register is one that may be referenced by means of the machine language that the processor executes. We can characterize these in the following categories:

• General purpose
• Data
• Address
• Condition codes

General-purpose registers can be assigned to a variety of functions by the programmer. Sometimes their use within the instruction set is orthogonal to the operation. That is, any general-purpose register can contain the operand for any opcode. This provides true general-purpose register use. Often, however, there are restrictions. For example, there may be dedicated registers for floating-point and stack operations. In some cases, general-purpose registers can be used for addressing functions (e.g., register indirect, displacement). In other cases, there is a partial or clean separation between data registers and address registers.

- **Data registers** may be used only to hold data and cannot be employed in the calculation of an operand address.
- **Address registers** may themselves be somewhat general purpose, or they may be devoted to a particular addressing mode. Examples include the following:
- **Segment pointers:** In a machine with segmented addressing ,a segment register holds the address of the base of the segment. There may be multiple registers: for example, one for the operating system and one for the current process.
- **Index registers:** These are used for indexed addressing and may be autoindexed.
- **Stack pointer:** If there is user-visible stack addressing, then typically there is a dedicated register that points to the top of the stack. This allows implicit addressing; that is, push, pop, and other stack instructions need not contain an explicit stack operand.

A final category of registers, which is at least partially visible to the user, holds **condition codes** (also referred to as *flags*). Condition codes are bits set by the processor hardware as the result of operations. For example, an arithmetic operation may produce a positive, negative, zero, or

overflow result. In addition to the result itself being stored in a register or memory, a condition code is also set.The code may subsequently be tested as part of a conditional branch operation. Condition code bits are collected into one or more registers. Usually, they form part of a control register. Generally, machine instructions allow these bits to be read by implicit reference, but the programmer cannot alter them. Many processors, including those based on the IA-64 architecture and the MIPS processors, do not use condition codes at all. Rather, conditional branch instructions specify a comparison to be made and act on the result of the comparison, without storing a condition code.

In some machines, a subroutine call will result in the automatic saving of all user-visible registers, to be restored on return. The processor performs the saving and restoring as part of the execution of call and return instructions. This allows each subroutine to use the user-visible registers independently. On other machines, it is the responsibility of the programmer to save the contents of the relevant uservisible
registers prior to a subroutine call, by including instructions for this purpose in the program.

Table 12.1 Condition Codes

Advantages	Disadvantages
1. Because condition codes are set by normal arithmetic and data movement instructions, they should reduce the number of COMPARE and TEST instructions needed. 2. Conditional instructions, such as BRANCH are simplified relative to composite instructions, such as TEST AND BRANCH. 3. Condition codes facilitate multiway branches. For example, a TEST instruction can be followed by two branches, one on less than or equal to zero and one on greater than zero.	1. Condition codes add complexity, both to the hardware and software. Condition code bits are often modified in different ways by different instructions, making life more difficult for both the microprogrammer and compiler writer. 2. Condition codes are irregular; they are typically not part of the main data path, so they require extra hardware connections. 3. Often condition code machines must add special non-condition-code instructions for special situations anyway, such as bit checking, loop control, and atomic semaphore operations. 4. In a pipelined implementation, condition codes require special synchronization to avoid conflicts.

Control and Status Registers

There are a variety of processor registers that are employed to control the operation of the processor. Most of these, on most machines, are not visible to the user. Some of them may be visible to machine instructions executed in a control or operating system mode. Of course, different machines will have different register organizations and use different terminology.

Four registers are essential to instruction execution:

- **Program counter (PC):** Contains the address of an instruction to be fetched
- **Instruction register (IR):** Contains the instruction most recently fetched
- **Memory address register (MAR):** Contains the address of a location in memory
- **Memory buffer register (MBR):** Contains a word of data to be written to memory or the word most recently read.

Not all processors have internal registers designated as MAR and MBR, but some equivalent buffering mechanism is needed whereby the bits to be transferred to the system bus are staged and the bits to be read from the data bus are temporarily stored. Typically, the processor updates the PC after each instruction fetch so that the PC always points to the next instruction to be executed. A branch or skip instruction will also modify the contents of the PC.The fetched instruction is loaded into an IR, where the opcode and operand specifiers are analyzed. Data are exchanged with memory using the MAR and MBR. In a bus-organized system, the MAR connects directly to the address bus, and the MBR connects directly to the data bus. Uservisible registers, in turn, exchange data with the MBR. The four registers just mentioned are used for the movement of data between the processor and memory. Within the processor, data must be presented to the ALU for processing. The ALU may have direct access to the MBR and user-visible registers. Alternatively, there may be additional buffering registers at the boundary to the ALU; these registers serve as input and output registers for the ALU and exchange data with the MBR and user-visible registers. Many processor designs include a register or set of registers, often known as the *program status word* (PSW), that contain status information. The PSW typically contains condition codes plus other status information. Common fields or flags include the following:

- **Sign:** Contains the sign bit of the result of the last arithmetic operation.
- **Zero:** Set when the result is 0.
- **Carry:** Set if an operation resulted in a carry (addition) into or borrow (subtraction) out of a high-order bit. Used for multiword arithmetic operations.
- **Equal:** Set if a logical compare result is equality.
- **Overflow:** Used to indicate arithmetic overflow.
- **Interrupt Enable/Disable:** Used to enable or disable interrupts.
- **Supervisor:** Indicates whether the processor is executing in supervisor or user mode. Certain privileged instructions can be executed only in supervisor mode, and certain areas of memory can be accessed only in supervisor mode.

A number of other registers related to status and control might be found in a particular processor design. There may be a pointer to a block of memory containing additional status information (e.g., process control blocks). In machines using vectored interrupts, an interrupt vector register may be provided. If a stack is used to implement certain functions (e.g. subroutine call), then a system stack pointer is needed. A page table pointer is used with a virtual memory system. Finally, registers may be used in the control of I/O operations. A number of factors go into the design of the control and status register organization. One key issue is operating system support. Certain types of control information are of specific utility to the operating system. If the processor designer has a functional understanding of the operating system to be used, then the register organization can to some extent be tailored to the operating system. Another key design decision is the allocation of control information between registers and memory. It is common to dedicate the first (lowest) few hundred or thousand words of memory for control purposes. The designer must decide how much control information should be in registers and how much in memory. The usual trade-off of cost versus speed arises.

Instruction Cycle

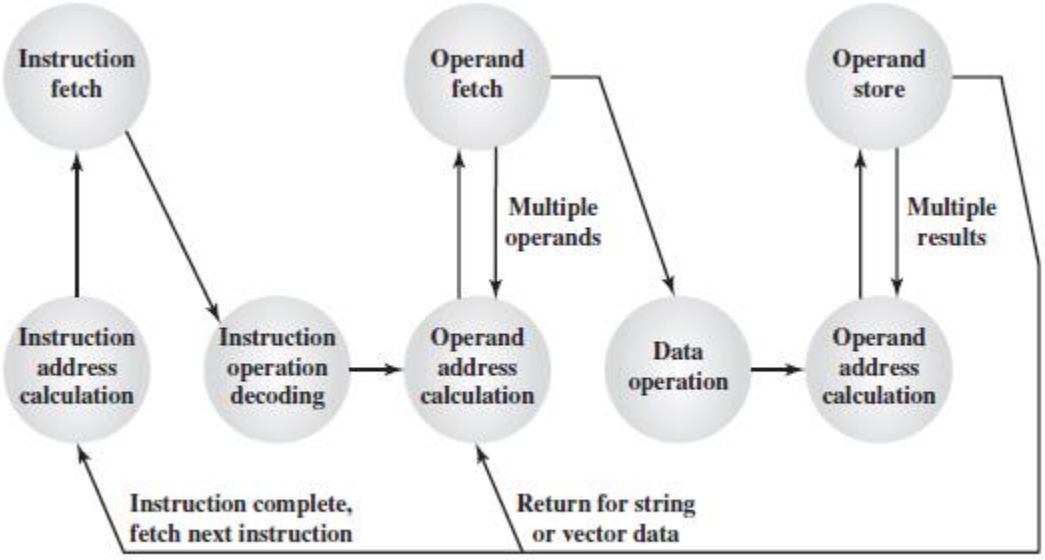


Figure 3.6 Instruction Cycle State Diagram

The states can be described as follows:

- **Instruction address calculation (iac):** Determine the address of the next instruction to be executed. Usually, this involves adding a fixed number to the address of the previous instruction. For example, if each instruction is 16 bits long and memory is organized into 16-bit words, then add 1 to the previous address. If, instead, memory is organized as individually addressable 8-bit bytes, then add 2 to the previous address.
- **Instruction fetch (if):** Read instruction from its memory location into the processor.
- **Instruction operation decoding (iod):** Analyze instruction to determine type of operation to be performed and operand(s) to be used.
- **Operand address calculation (oac):** If the operation involves reference to an operand in memory or available via I/O, then determine the address of the operand.
- **Operand fetch (of):** Fetch the operand from memory or read it in from I/O.
- **Data operation (do):** Perform the operation indicated in the instruction.
- **Operand store (os):** Write the result into memory or out to I/O.

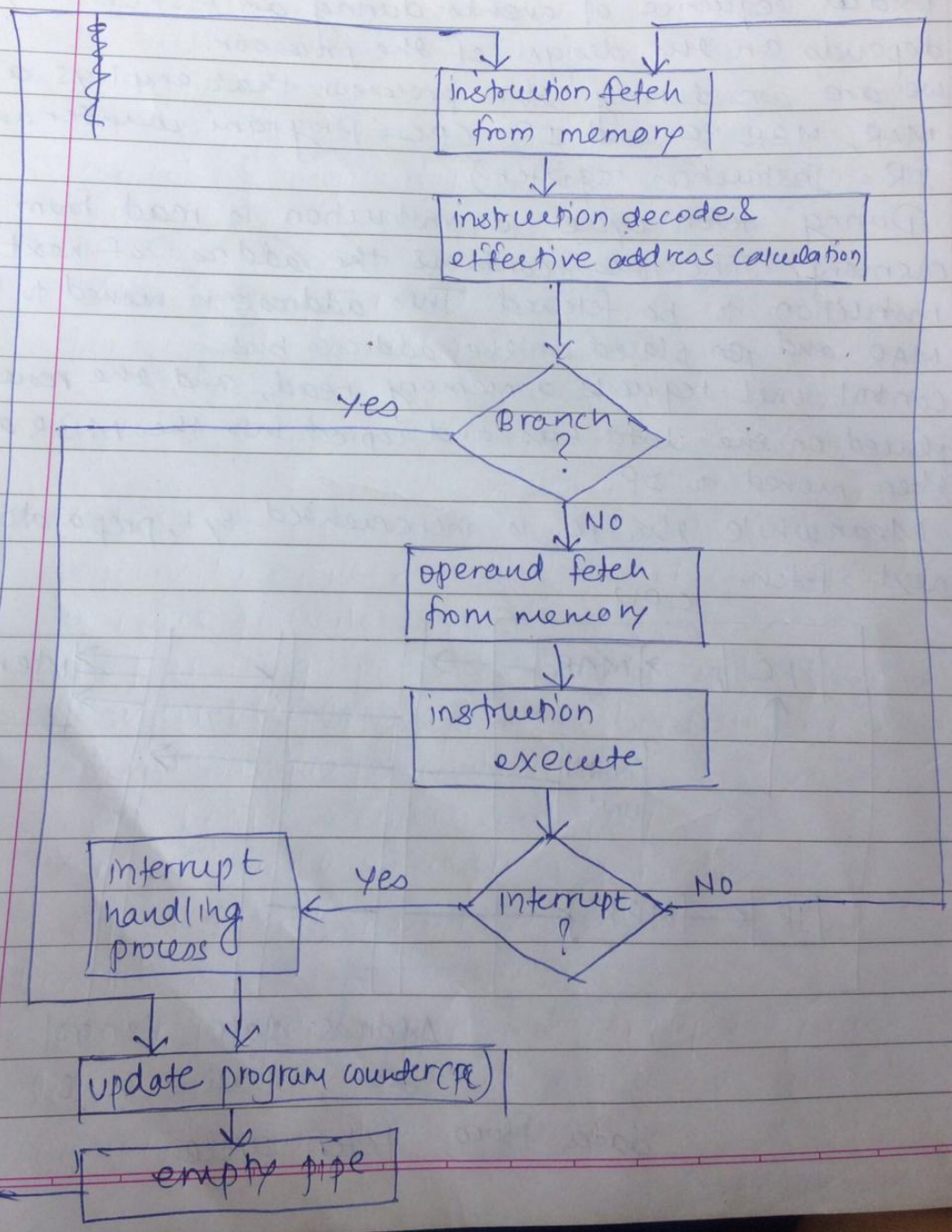
States in the upper part of Figure 3.6 involve an exchange between the processor and either memory or an I/O module. States in the lower part of the diagram involve only internal processor operations. The oac state appears twice, because an instruction may involve a read, a write, or both. However, the action performed during that state is fundamentally the same in both cases, and so only a single state identifier is needed. Also note that the diagram allows for multiple operands and multiple results, because some instructions on some machines require this. For example, the PDP-11 instruction ADD A,B results in the following sequence of states: iac, if, iod, oac, of, oac, of, do, oac, os. Finally, on some machines, a single instruction can specify an operation to be performed on a vector (one-dimensional array) of numbers or a string (one-dimensional array) of characters. As Figure 3.6 indicates, this would involve repetitive operand fetch and/or store operations.

Instruction Pipeline

risc architecture

instruction pipeline

IF = Instruction fetch
ID = Instruction decode
OF = Operand fetch
Ex = execute



branch instruction :- instructions like jump, go to which hampers the normal execution.

steps	1	2	3	4	5	6	7	8	9	10	11	12	13
instruction 1	IF	ID	OF	EX									
2		IF	ID	OF	EX								
branch 3			IF	ID	OF	EX							
4				IF	-	-	IF	ID	OF	EX			
5					-	-	-	IF	ID	OF	EX		
6						-	-	-	IF	ID	OF	EX	
7							-	-	-	IF	ID	OF	EX

↑ ↑ → time stamps

timing diagram of instruction pipeline

- first instruction has fetched and then it will be decode, meanwhile ~~2nd~~ 1st instruction is decoded 2nd instruction will be fetched
- and in third cycle 1st instruction will go for "operand fetch" phase and meanwhile second instruction has decoded and so on.
- third instruction is branch instruction (eg. jump go to) so linear or sequential execution of instruction get hampered.
- As third instruction is having the branch instruction, the next (4th) instruction will not get execution (depends on condition) and control transfers to the target memory location.

But as shown in timing diagram, meanwhile 3rd instruction has decoded, 4th instruction has already fetched but due to presence of branch instruction the execution of 4th instruction get blocked. and ~~next~~ cycles next to 4th instruction remains unutilized.

- and after execution of 3rd instruction 4th instruction will get executed.

such mechanism of stopping instruction due to branch instruction is known as pipeline ~~halt~~ stall.

RISC and CISC architecture

Central Processing Unit Architecture operates the capacity to work from "Instruction Set Architecture" to where it was designed. The architectural designs of CPU are RISC (Reduced instruction set computing) and CISC (Complex instruction set computing). CISC has the ability to execute addressing modes or multi-step operations within one instruction set. It is the design of the CPU where one instruction performs many low-level operations. For example, memory storage, an arithmetic operation and loading from memory. RISC is a CPU design strategy based on the insight that simplified instruction set gives higher performance when combined with a microprocessor architecture which has the ability to execute the instructions by using some microprocessor cycles per instruction.

This article discusses about the RISC and CISC architecture with suitable diagrams.

- Hardware of the Intel is termed as Complex Instruction Set Computer (CISC)
- Apple hardware is Reduced Instruction Set Computer (RISC).

What is RISC and CISC Architectures

Hardware designers invent numerous technologies & tools to implement the desired architecture in order to fulfill these needs. Hardware architecture may be implemented to be either hardware specific or software specific, but according to the application both are used in the required quantity. As far as the processor hardware is concerned, there are 2 types of concepts to implement the processor hardware architecture. First one is RISC and other is CISC.

CISC Architecture

The CISC approach attempts to minimize the number of instructions per program, sacrificing the number of cycles per instruction. Computers based on the CISC architecture are designed to decrease the memory cost. Because, the large programs need more storage, thus increasing the memory cost and large memory becomes more expensive. To solve these problems, the number of instructions per program can be reduced by embedding the number of operations in a single instruction, thereby making the instructions more complex.

- MUL loads two values from the memory into separate registers in CISC.
- CISC uses minimum possible instructions by implementing hardware and executes operations.
- Instruction Set Architecture is a medium to permit communication between the programmer and the hardware. Data execution part, copying of data, deleting or editing is the user commands used in the microprocessor and with this microprocessor the Instruction set architecture is operated.
- The main keywords used in the above Instruction Set Architecture are as below

Instruction Set: Group of instructions given to execute the program and they direct the computer by manipulating the data. Instructions are in the form – Opcode (operational code) and

Operand. Where, opcode is the instruction applied to load and store data, etc. The operand is a memory register where instruction applied.

Addressing Modes: Addressing modes are the manner in the data is accessed. Depending upon the type of instruction applied, addressing modes are of various types such as direct mode where straight data is accessed or indirect mode where the location of the data is accessed. Processors having identical ISA may be very different in organization. Processors with identical ISA and nearly identical organization is still not nearly identical.

CPU performance is given by the fundamental law

Thus, CPU performance is dependent upon Instruction Count, CPI (Cycles per instruction) and Clock cycle time. And all three are affected by the instruction set architecture.

Examples of CISC PROCESSORS

IBM 370/168 – It was introduced in the year 1970. CISC design is a 32 bit processor and four 64-bit floating point registers.
VAX 11/780 – CISC design is a 32-bit processor and it supports many numbers of addressing modes and machine instructions which is from Digital Equipment Corporation.
Intel 80486 – It was launched in the year 1989 and it is a CISC processor, which has instructions varying lengths from 1 to 11 and it will have 235 instructions.

CHARACTERISTICS OF CISC ARCHITECTURE

- Instruction-decoding logic will be Complex.
- One instruction is required to support multiple addressing modes.
- Less chip space is enough for general purpose registers for the instructions that are 0operated directly on memory.
- Various CISC designs are set up two special registers for the stack pointer, handling interrupts, etc.
- MUL is referred to as a “complex instruction” and requires the programmer for storing functions.

RISC Architecture

RISC (Reduced Instruction Set Computer) is used in portable devices due to its power efficiency. For Example, Apple iPod and Nintendo DS. RISC is a type of microprocessor architecture that uses highly-optimized set of instructions. RISC does the opposite, reducing the cycles per instruction at the cost of the number of instructions per program Pipelining is one of the unique feature of RISC. It is performed by overlapping the execution of several instructions in a pipeline fashion. It has a high performance advantage over CISC.

RISC processors take simple instructions and are executed within a clock cycle

RISC ARCHITECTURE CHARACTERISTICS

- Simple Instructions are used in RISC architecture.
- RISC helps and supports few simple data types and synthesize complex data types.
- RISC utilizes simple addressing modes and fixed length instructions for pipelining.
- RISC permits any register to use in any context.
- One Cycle Execution Time
- The amount of work that a computer can perform is reduced by separating "LOAD" and "STORE" instructions.
- RISC contains Large Number of Registers in order to prevent various number of interactions with memory.
- In RISC, Pipelining is easy as the execution of all instructions will be done in a uniform interval of time i.e. one click.
- In RISC, more RAM is required to store assembly level instructions.
- Reduced instructions need a less number of transistors in RISC.
- RISC uses Harvard memory model means it is Harvard Architecture.
- A compiler is used to perform the conversion operation means to convert a high-level language statement into the code of its form.

RISC & CISC Comparison

MUL instruction is divided into three instructions
"LOAD" – moves data from the memory bank to a register
"PROD" – finds product of two operands located within the registers
"STORE" – moves data from a register to the memory banks
The main difference between RISC and CISC is the number of instructions and its complexity.

The Advantages and Disadvantages of RISC and CISC

The Advantages of RISC architecture

- RISC(Reduced instruction set computing)architecture has a set of instructions, so high-level language compilers can produce more efficient code
- It allows freedom of using the space on microprocessors because of its simplicity.
- Many RISC processors use the registers for passing arguments and holding the local variables.
- RISC functions use only a few parameters, and the RISC processors cannot use the call instructions, and therefore, use a fixed length instruction which is easy to pipeline.
- The speed of the operation can be maximized and the execution time can be minimized. Very less number of instructional formats, a few numbers of instructions and a few addressing modes are needed.

The Disadvantages of RISC architecture

- Mostly, the performance of the RISC processors depends on the programmer or compiler as the knowledge of the compiler plays a vital role while changing the CISC code to a RISC code
- While rearranging the CISC code to a RISC code, termed as a code expansion, will increase the size. And, the quality of this code expansion will again depend on the compiler, and also on the machine's instruction set.
- The first level cache of the RISC processors is also a disadvantage of the RISC, in which these processors have large memory caches on the chip itself. For feeding the instructions, they require very fast memory systems.

Advantages of CISC architecture

- Microprogramming is easy assembly language to implement, and less expensive than hard wiring a control unit.
- The ease of microcoding new instructions allowed designers to make CISC machines upwardly compatible:
- As each instruction became more accomplished, fewer instructions could be used to implement a given task.

Disadvantages of CISC architecture

- The performance of the machine slows down due to the amount of clock time taken by different instructions will be dissimilar
- Only 20% of the existing instructions is used in a typical programming event, even though there are various specialized instructions in reality which are not even used frequently.
- The conditional codes are set by the CISC instructions as a side effect of each instruction which takes time for this setting – and, as the subsequent instruction changes the condition code bits – so, the compiler has to examine the condition code bits before this happens.

Pentium Architecture

The Pentium family of processors originated from the 80486 microprocessor. The term "Pentium processor" refers to a family of microprocessors that share a common architecture and instruction set. The first Pentium processors were introduced in 1993. It runs at a clock frequency of either 60 or 66 MHz and has 3.1 million transistors. Some of the features of Pentium architecture are

- Complex Instruction Set Computer (CISC) architecture with Reduced Instruction Set Computer (RISC) performance.
- 64-Bit Bus
- Upward code compatibility.
- Pentium processor uses Superscalar architecture and hence can issue multiple instructions per cycle.
- Multiple Instruction Issue (MII) capability.
- Pentium processor executes instructions in five stages. This staging, or pipelining, allows the processor to overlap multiple instructions so that it takes less time to execute two instructions in a row.

- The Pentium processor fetches the branch target instruction before it executes the branch instruction.
- The Pentium processor has two separate 8-kilobyte (KB) caches on chip, one for instructions and one for data. It allows the Pentium processor to fetch data and instructions from the cache simultaneously.
- When data is modified, only the data in the cache is changed. Memory data is changed only when the Pentium processor replaces the modified data in the cache with a different set of data
- The Pentium processor has been optimized to run critical instructions in fewer clock cycles than the 80486 processor.

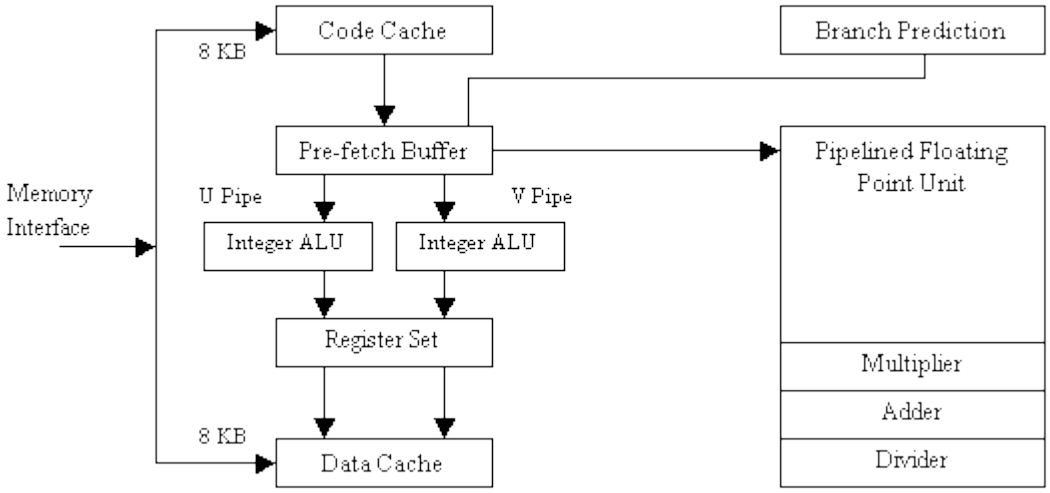


Fig 35.1 Superscalar Architecture of Pentium

The Pentium processor has two primary operating modes -

1. **Protected Mode** - In this mode all instructions and architectural features are available, providing the highest performance and capability. This is the recommended mode that all new applications and operating systems should target.
2. **Real-Address Mode** - This mode provides the programming environment of the Intel 8086 processor, with a few extensions. Reset initialization places the processor in real mode where, with a single instruction, it can switch to protected mode

The Pentium's basic integer pipeline is five stages long, with the stages broken down as follows:

1. **Pre-fetch/Fetch** : Instructions are fetched from the instruction cache and aligned in pre-fetch buffers for decoding.
2. **Decode1** : Instructions are decoded into the Pentium's internal instruction format. Branch prediction also takes place at this stage.
3. **Decode2** : Same as above, and microcode ROM kicks in here, if necessary. Also, address computations take place at this stage.
4. **Execute** : The integer hardware executes the instruction.
5. **Write-back** : The results of the computation are written back to the register file.

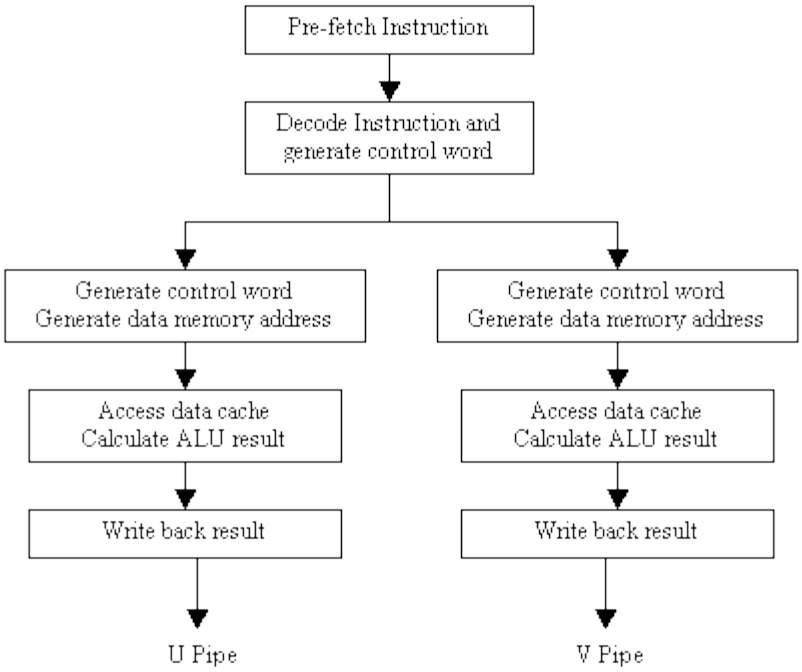


Fig 35.2 Pentium pipeline stages

Floating Point Unit :

There are 8 general-purpose 80-bit Floating point registers. Floating point unit has 8 stages of pipelining. First five are similar to integer unit. Since the possibility of error is more in Floating Point unit (FPU) than in integer unit, additional error checking stage is there in FPU. The floating point unit is shown as below

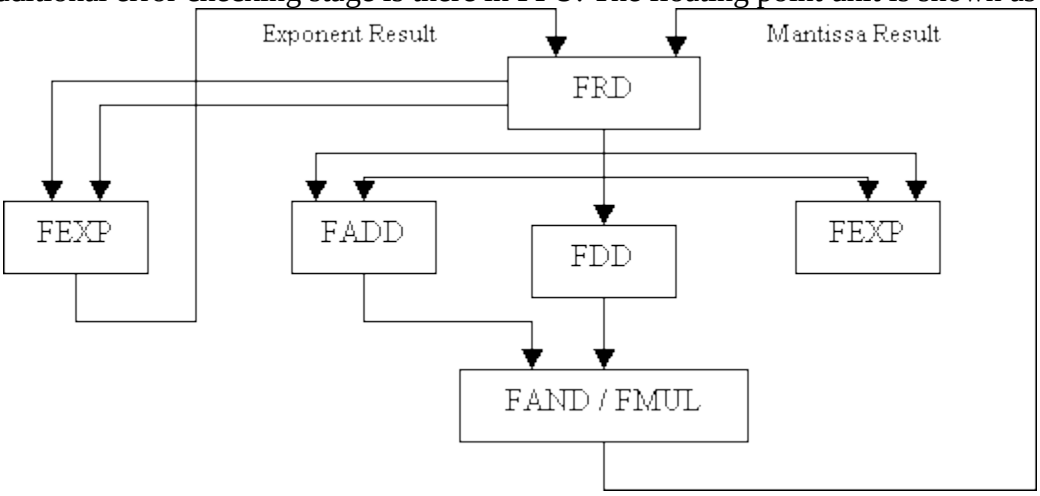


Fig 35.3 Floating Point Unit

FRD - Floating Point Rounding
FDD - Floating Point Division
FADD - Floating Point Addition
FEXP - Floating Point Exponent
FAND - Floating Point And
FMUL - Floating Point Multiply

Power PC

In February 1990, IBM introduced RS/6000 microprocessor based on POWER architecture with UNIX operating system. PowerPC was second generation POWER architecture. It has Reduced Instruction Set Computer (RISC) architecture. RISC architecture tries to keep the processor as busy as possible. Salient features of RISC architecture are -

- Fixed length instructions (4 byte instructions). This allows single decoding mechanism
- Mostly single cycle instruction execution
- Less number of instructions

PowerPC was created in 1991 by Apple-IBM-Motorola alliance. Originally intended for personal computers , PowerPC CPUs have since become popular embedded and high-performance processors as well. It is largely based and compatible with POWER microprocessor. Design features of PowerPC are as follows -

- Broad range implementation
- Simple processor design
- Superscalar architecture
- Multiprocessor features
- 64-bit architecture
- Support for operation in both big-endian and little-endian mode. PowerPC can switch from one mode to another at run time.
- Separate set of floating point instructions for
- Separate set of Floating Point Registers (FPRs) for floating-point instructions

Motorola PowerPC 601 was the first PowerPC. Few of its features were -

1. 64-bit microprocessor
2. 32-bit address lines
3. Can handle integer data of 8, 16 and 32 bits
4. RISC architecture with 4 byte instruction length
5. PC 601 has virtual memory addressing of 4 penta byte.

Apart from the changes to the instruction set, the most significant changes in PowerPC were in the memory model and the memory management definition. In the POWER Architecture, the processor did not maintain data memory consistent with either I/O accesses or instruction fetches. Software had to manage memory consistency for both these areas. Before copying an area of memory to disk, software had to ensure that any modified copies of the memory area that were in the data cache had been written to main memory. Before starting a read from disk, software had to ensure that the data cache did not contain a copy of any part of the memory area, and software had to invalidate any copy of the memory area in the instruction cache before restarting the program that requested the operation. POWER processors always accessed main memory through the caches.

PowerPC memory model, however, provides greater flexibility. It implements processor-enforced data memory consistency, relieving software of the responsibility for the consistency of memory with respect to

I/O operations. The model allows speculative access to any page unless it has an attribute indicating that it contains I/O or it exhibits other volatile characteristics. It also makes it possible to map I/O into the main memory space.
As in the POWER memory model, the PowerPC memory model requires software to maintain instruction memory consistent with data memory. Programs that modify or generate instructions must ensure that cached copies of a memory area containing the new instructions are consistent with the main memory before attempting to execute those instructions.

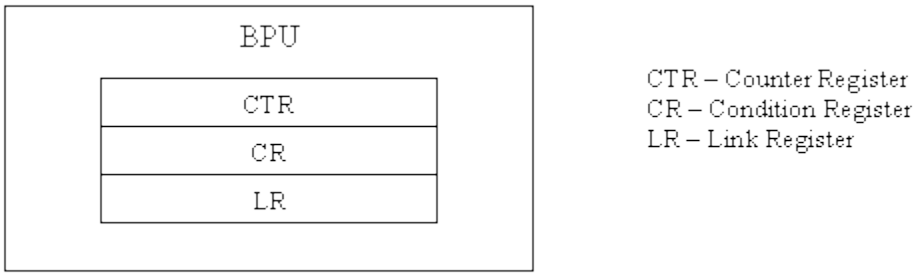


Fig 36.1 Branch Processing Unit of PowerPC

The Branch Processing Unit (BPU) looks at lower four instructions in instruction queue to bring the branch instruction in advance. The jump instruction is analyzed and the next instruction is brought and executed till the write-back stage. With this the branch takes single cycle. A branch instruction has a Jump Prediction Bit associated with which tells whether there is likelihood of jump or not. In case a jump is predicted new instructions may be brought in for the entire instruction queue. Later, if the prediction comes out to be true then the execution continues normally and we have considerable amount of performance gain. However, if branch prediction turns out to be false then we have something called Branch Folding. In branch folding all instructions executed after the prediction are discarded and the execution resumes just after branch instruction. We have loss of instruction cycles in this case.
The PowerPC Architecture permits a range of implementations from low-cost controllers through high-performance processors. It allows the implementation of processors targeted for desktop and notebook systems, yet it contains features to support the efficient implementation of processors for use in a range of multiprocessor systems.

Superscalar processor

A **superscalar processor** is one in which multiple independent instruction pipelines are used. Each pipeline consists of multiple stages, so that each pipeline can handle multiple instructions at a time. Multiple pipelines introduce a new level of parallelism, enabling multiple streams of instructions to be processed at a time. A superscalar processor exploits what is known as **instruction-level parallelism**, which refers to the degree to which the instructions of a program can be executed in parallel.

◆ A superscalar processor typically fetches multiple instructions at a time and then attempts to find nearby instructions that are independent of one another and can therefore be executed in parallel. If the input to one instruction depends on the output of a preceding instruction, then the latter instruction cannot complete execution at the same time or before the former instruction. Once such dependencies have been identified, the processor may issue and complete instructions in an order that differs from that of the original machine code.

The term *superscalar*, first coined in 1987 refers to a machine that is designed to improve the performance of the execution of scalar instructions. In most applications, the bulk of the operations are on scalar quantities. Accordingly, the superscalar approach represents the next step in the evolution of high-performance general-purpose processors. The essence of the superscalar approach is the ability to execute instructions independently and concurrently in different pipelines. The concept can be further exploited by allowing instructions to be executed in an order different from the program order. Figure 14.1 shows, in general terms, the superscalar approach. There are multiple functional units, each of which is implemented as a pipeline, which support parallel execution of several instructions. In this example, two integer, two floating-point, and one memory (either load or store) operations can be executing at the same time. Many researchers have investigated superscalar-like processors, and their research indicates that some degree of performance improvement is possible.

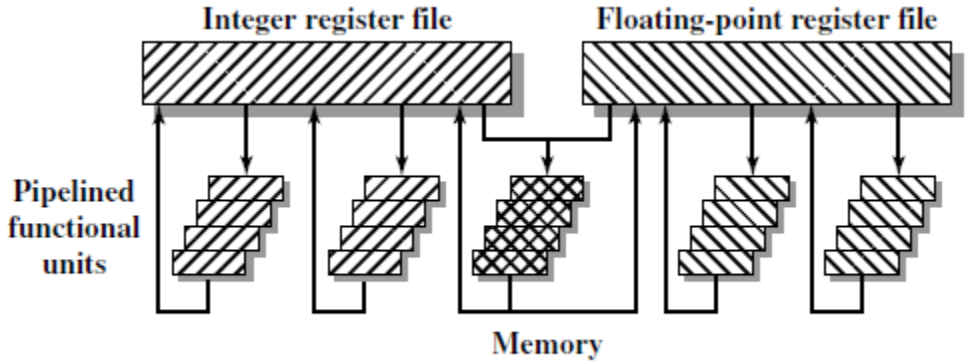
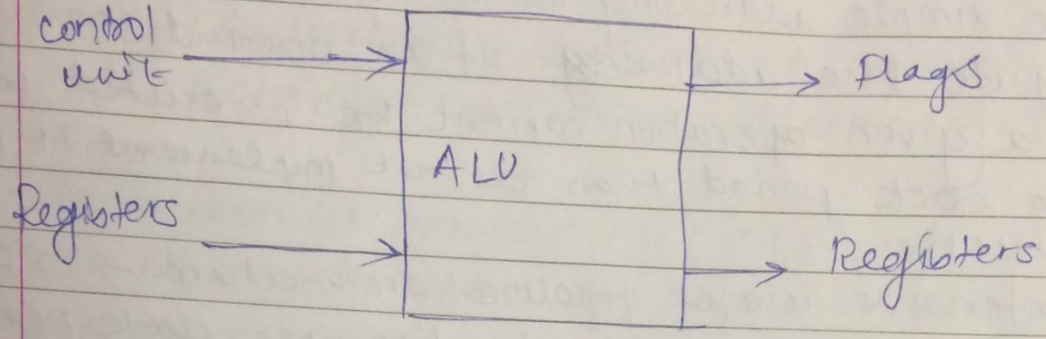


Figure 14.1 General Superscalar Organization

UNIT 2

Computer Arithmetic

- Arithmetic logic unit (ALU) is a combination digital electronic circuits that performs arithmet and bitwise operations on integer binary numbers.

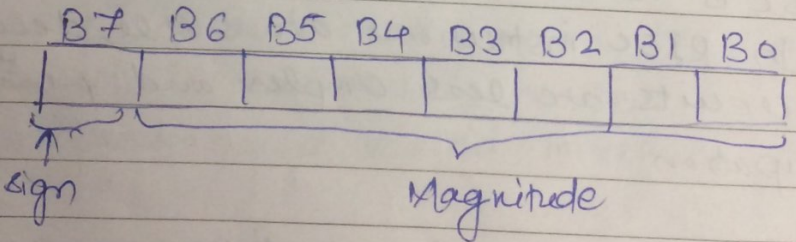


Integer Representation

1) Sign-Magnitude Representation
2) Two's complement Representation
3) Fixed point representation.

Sign-magnitude representation:

- In signed number it is necessary to represent negative as well as positive numbers.
- In sign-magnitude representation, the most significant bit (MSB) is used to represent the sign of number.
- If MSB is 0, then number is positive, and if MSB is 1, then number is negative. and remaining bits of number represent the magnitude



for ex. '16 ⇒ Binary no = 10000

+16	00010000
-16	10010000

In general, if an n-bit sequence of binary digits $a_{n-1} a_{n-2} \cdots a_1 a_0$ is interpreted as an unsigned integer A, its value is

$$A = \sum_{i=0}^{n-1} 2^i a_i$$

for. $N = A_n 2^n + A_{n-1} 2^{n-1} + \cdots A_1 2^1 + A_0 2^0$

$$N = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$
$$= 8 + 4 + 0 + 1$$
$$= 13$$

There are several drawbacks to sign-magnitude representation.

- addition and substraction require a consideration of both the signs of numbers and their relative magnitudes to carry out the required operation
- Another drawback is that there are two representations of 0. $+0_{10} = 00000000$
  $-0_{10} = 10000000$

this is inconvenient because it is slightly more difficult to test 0 (an operation performed frequently on computers) than if there were a single representation.

Representations of +0 and -0 would be 0000 and 1000 respectively, which can cause big complications for computers and digital system.

Two's complement Representation

Two's complement of binary number is obtained by adding 1 to the least significant bit (LSB) of 1's complement of the number.

2's complement : 1's complement +1

for eg. 1001

1st complement 0110

$+ \quad 1$

0111

These representations ~~are~~ referred to as fixed-point representation.

Arithmetic Arithmetic

1) Negation:
- In sign-magnitude representation, the rule for forming the negation of an integer is simple. ∴ invert the sign bit.
- In twos complement notation, the negation of an integer can be formed with the following rules:

1. Take the boolean complement of each bit of integer (including sign bit) ie. set 1 to 0 and 0 to 1
2. Treating the result as an unsigned binary integer, add 1.

$$
\begin{array}{rl}
+18 & = 00010010 \\
 & + \quad 11101101 \text{ (1s complement)} \\
 & \quad\quad\quad 1 \\
\hline
 & 11101110 \quad = -18
\end{array}
$$

$$
\begin{array}{rl}
-18 & 11101110 \\
 & 00010001 \text{ (1s complement)} \\
 & + \quad\quad 1 \\
\hline
 & 00010010 \quad = (+18)
\end{array}
$$

$b_3b_2b_1b_0$	sign & magnitude	1s complement	2s complement
0111	+7	+7	+7
0110	+6	+6	+6
0101	+5	+5	+5
0100	+4	+4	+4
0011	+3	+3	+3
0010	+2	+2	+2
0001	+1	+1	+1
0000	+0	+0	+0
1000	-0	-7	-8
1001	-1	-6	-7
1010	-2	-5	-6
1011	-3	-4	-5
1100	-4	-3	-4
1101	-5	-2	-3
1110	-6	-1	-2
1111	-7	-0	-1

- in sign magnitude, if MSB is 1 then no. is -ve
  if MSB is 0 then no. is +ve.
- There are distinct representation for +0 and -0 in both the sign-magnitude and 1s complement systems, but the 2's complement system has only one representation for 0.

Addition of positive numbers:

$$\begin{array}{cccc} {\scriptstyle 0} & {\scriptstyle 1} & {\scriptstyle 0} & {\scriptstyle 1} \\ +\,0 & +\,0 & +\,1 & +\,1 \\ \hline 0 & 1 & 1 & 10 \end{array}$$

carry

2's complement addition & subtraction

a) 0010 (+2) + 0011 (+3) = 0101 (+5)

b) 0100 (+4) + 1010 (-6) = 1110 (-2)

c) 1011 (-5) + 1110 (-2) = 1001 (-7)

d) 0111 (+7) + 1101 (-3) = 0100

e) 1101 (-3) − 1001 (-7) ⇒ 1101 (-3) + 0111 (7) = 0100 (+4)

f) 0010 (+2) − 0100 (+4) ⇒ 0010 (+2) + 1100 (-4) = 1110 (-2)

g) 0110 (6) − 0011 (3) ⇒ 0110 (6) + 1101 (-3) = 0011 (+3)

i) 1001 (-7) − 0001 (+1) ⇒ 1001 (-7) + 1111 (-1) = 1000 (-8)

j) 0010 (+2) − 1101 (-3) ⇒ 0010 (2) + 0011 (3) = 0101 (+5)

2's complement method is used for computing in modern computers.

Overflow in integer Arithmetic

- In two's complement number representation system, n-bits can represent values in range -2^{n-1} to $+2^{n-1}-1$ for ex. using four bits, range of numbers that can be represented is -8 through +7.
- when the result of arithmetic operation is outside the representable range, an arithmetic overflow has occured.
  1) Overflow can occur only when adding two numbers that have same sign.
  2) if two numbers are added, and they are both positive or both negative, then overflow occurs if and only if the result has opposite sign.

ex.
$$\begin{array}{r} 0011\ (3) \\ 0110\ (6) \\ \hline 1001\ (-7) \end{array}$$

(correct answer would be 9, but 9 cannot be represented in 4-bit 2's complement.

Add (-25) & (-37)

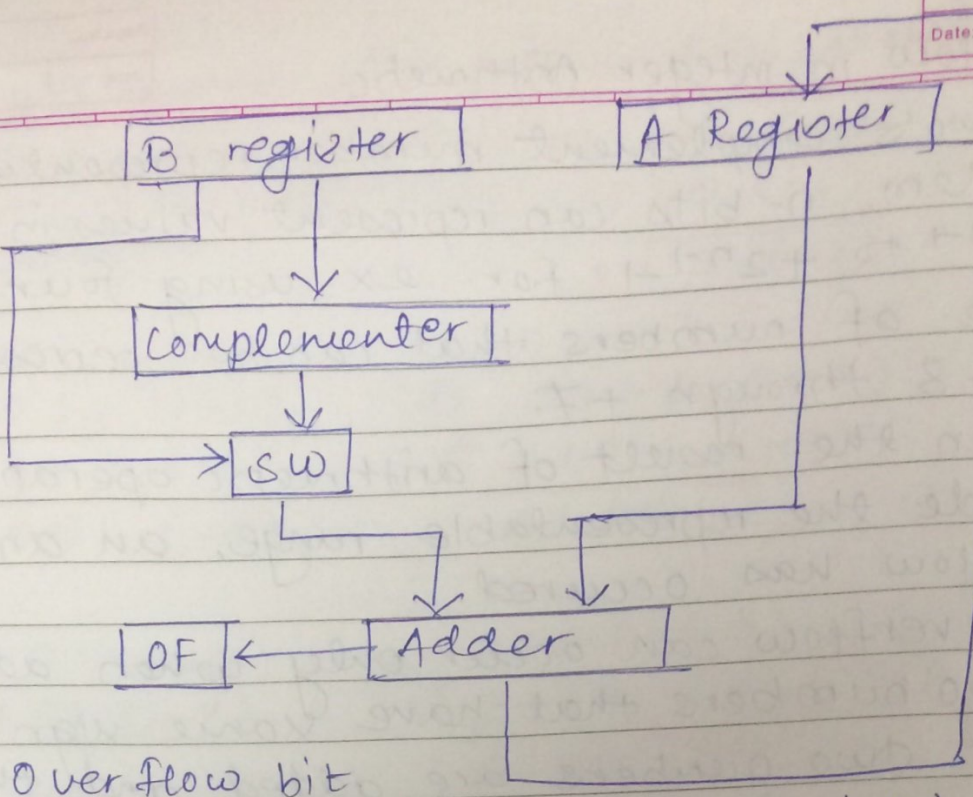
- $+25 + 00011001$
  $-37 \quad 11011011$
  $\overline{11110100} \Leftarrow (-12)$

in 2's complement no. system, 4-bit can be represent numbers from (-8 to 7)

5 bit can represent numbers from (-16 to 15)

overflow: eg.
$$\begin{array}{r} 0001\ (1) \\ 0111\ (7) \\ \hline 1000\ (-8) \end{array}$$

7+1 = 8 but (8) cannot be represented with 4 bit 2's complement number as it is out of range. (And largest +ve no in 8-bits is



OF = Overflow bit

SW = Select addition or subtraction (Switch)

Block diagram of hardware for addition and subtraction.

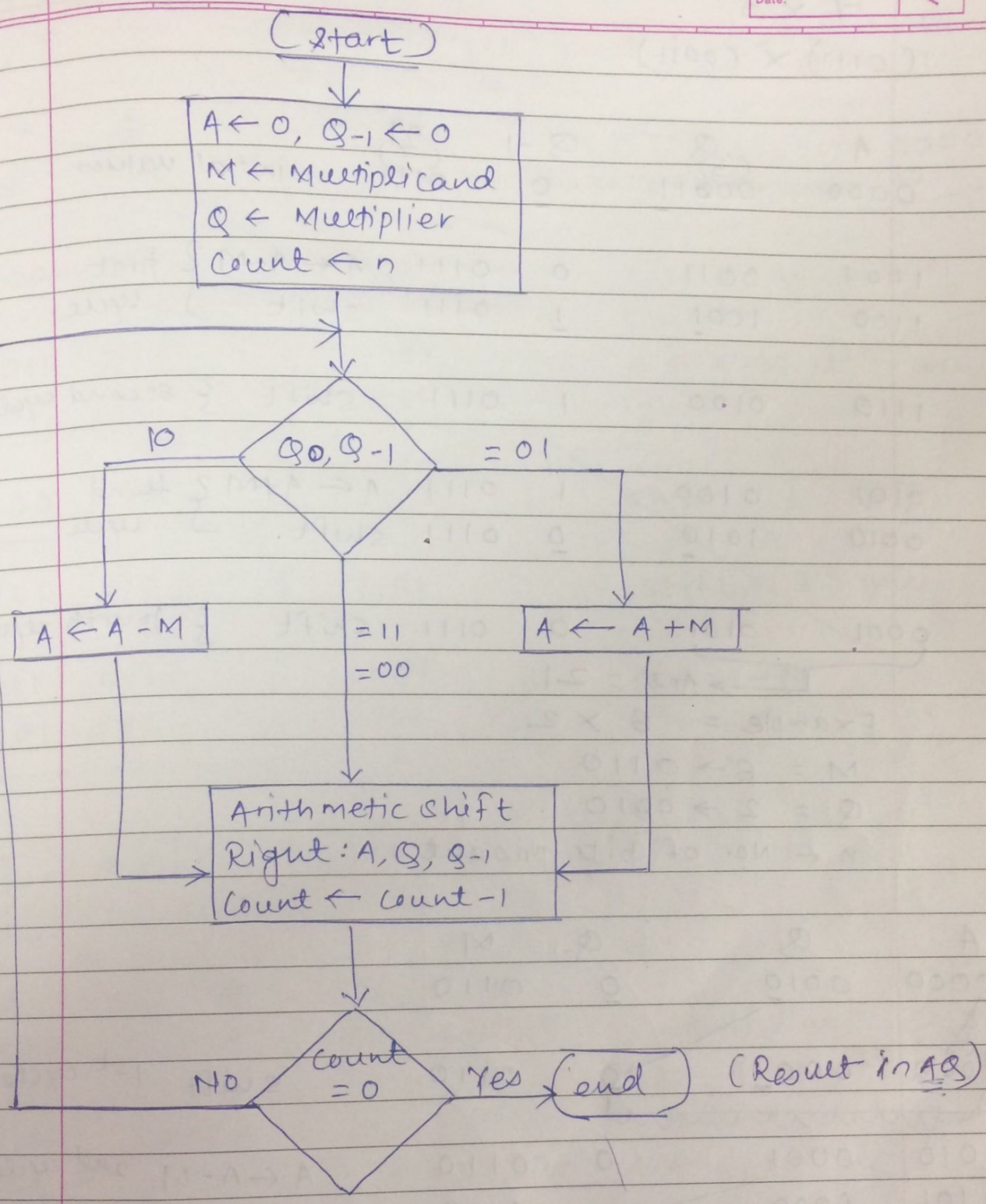
- two numbers are presented to adder from two registers, designated in this case A and B registers. the result may be stored in one of these registers or third.
- Overflow indication is stored in a 1-bit overflow flag.
- for subtraction, the subtrahend is passed through a twos complementer so that its twos complementer is presented to the adder.

$(-9) \times (-9) = 18$

$-9 \times 9 = -18$

Multiplication

Boath's algorithm for multiplication



Booth's algorithm for multiplication

Consider example (7 × 3)

7 × 3

(0111) × 0011

multiplicand multiplier

7 x 3

(0111) x (0011)

A	Q (3)	Q-1	M (7)	
0000	0011	0	0111	initial values
1001	0011	0	0111	A ← A-M } first
1100	1001	1	0111	shift } cycle
1110	0100	1	0111	shift } second cycle
0101	0100	1	0111	A ← A+M } third
0010	1010	0	0111	shift } cycle
0001	0101	0	0111	shift } fourth cycle

→ Ans = 21

Example = 6 x 2

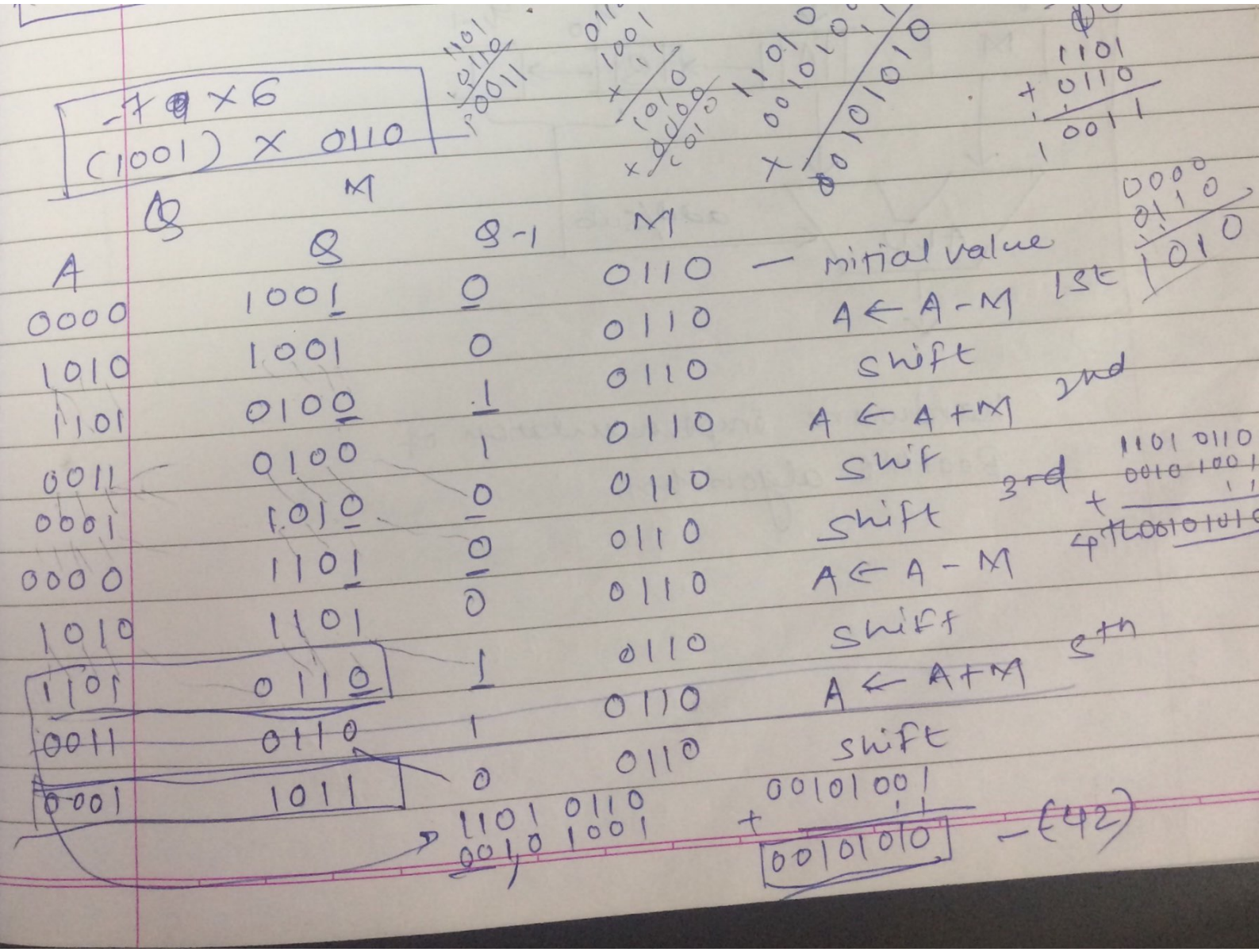
M = 6 → 0110

Q = 2 → 0010

n ← No. of bits present

A	Q	Q-1	M	
0000	0010	0	0110	
0000	0001	0	0110	shift 1st cycle
1010	0001	0	0110	A ← A-M 2nd cycle
1101	0000	1	0110	shift
0011	0000	1	0110	A ← A+M 3rd cycle
0001	1000	0	0110	shift
0000	1100	0	0110	shift 4th cycle

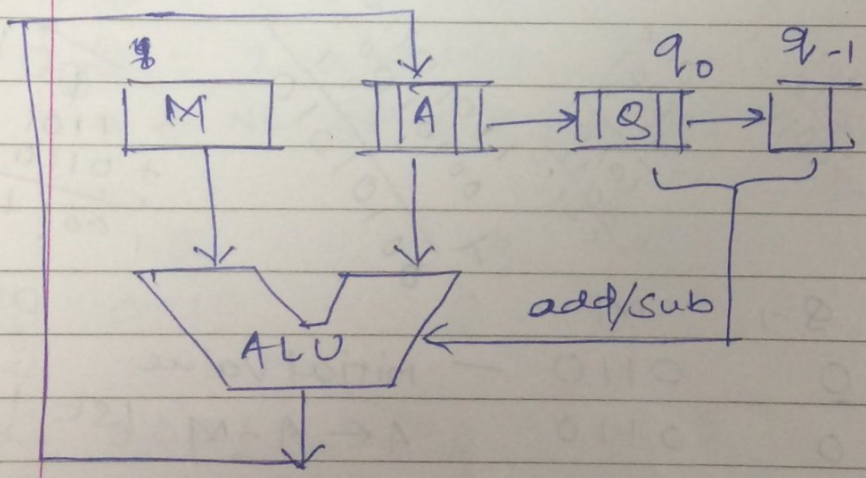
Ans = 00001100 ⇒ 12


-7 × 6
(1001) × 0110
A
Q
M
Q-1
A
Q
Q-1
M
0000
1001
0
0110 — initial value
1010
1001
0
0110
A ← A-M
1st
1101
0100
1
0110
shift
0011
0100
1
0110
A ← A+M
2nd
0001
1010
0
0110
shift
0000
1101
0
0110
shift
3rd
1010
1101
1
0110
A ← A-M
1101
0011
0110
shift
4th
0001
1011
0
0110
A ← A+M
1101 0110
0010 1001
00101001
00101010
-(42)
Scanned with CamScanner

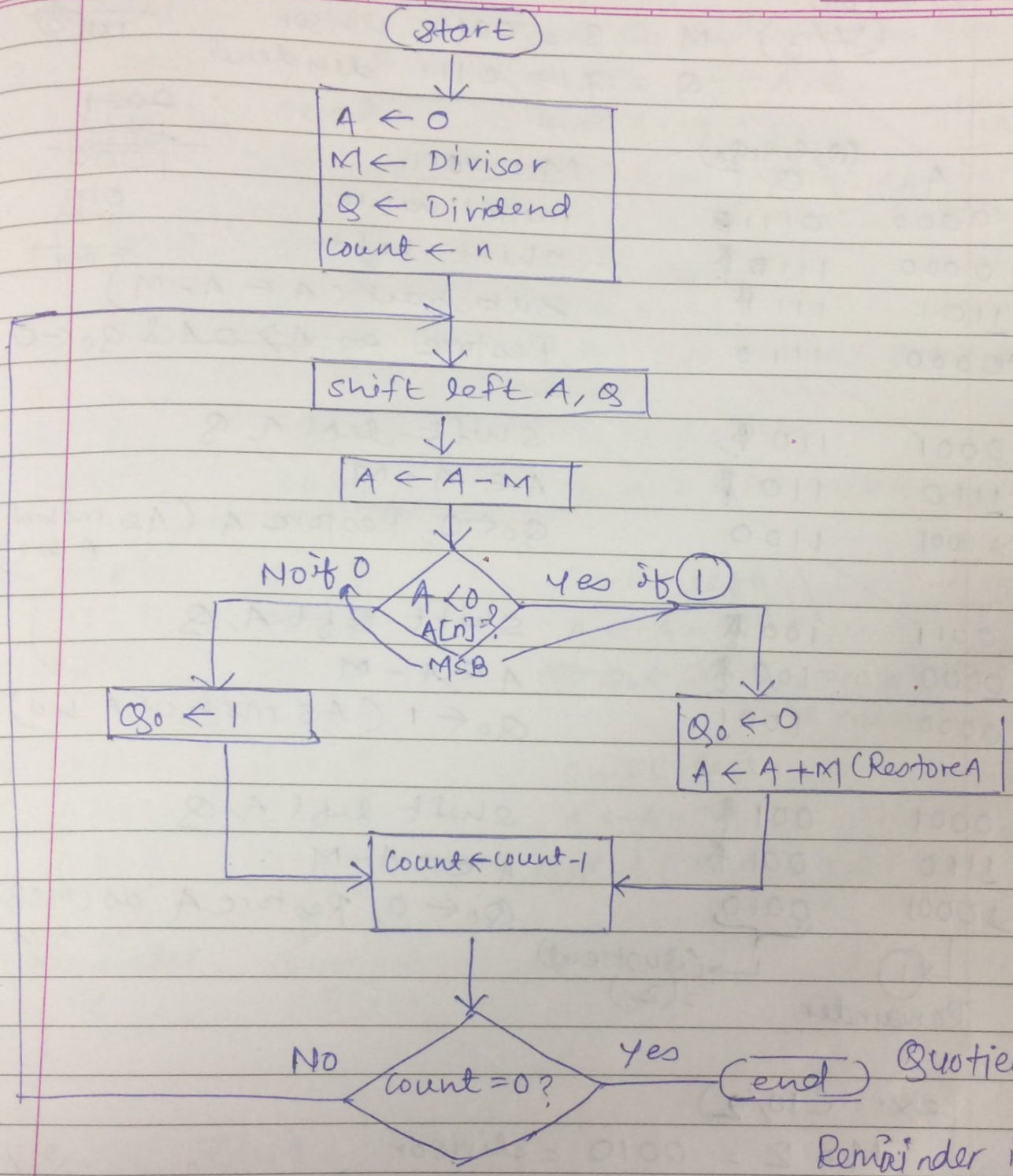
- In two's complement notation, a non-negative number is represented by its ordinary binary representation, in this case, the MSB is 0,
- Though, the range of numbers represented is not the same as with unsigned binary numbers.
- for ex, 8-bit unsigned number can represent the values 0 to 255 (11111111), However a two's complement 8-bit number can only represent positive integers from 0 to 127 (11111111)

Booth's algorithm is for the two's complement multiplication.

1001 (9)	1001 (-7)
0011 (3)	0011 (3)
27	-21



hardware implementation of Booth's algorithm.



Flowchart for Restoring division algorithm.
(unsigned binary division)

example:

(7/3) M = 3 = 0011 Divisor

Q = 7 = 0111 dividend

	A	Q ($Q_3Q_2Q_1Q_0$)	M = 0011
1	0000	0111	initial value
	0000	1110	shift left.
	1101	1110	substract ($A \leftarrow A-M$)
	0000	1110	Restore ~~as A→0~~ A & $Q_0 \leftarrow 0$
2.	0001	1100	shift left A, Q
	1110	1100	$A \leftarrow A-M$
	0001	1100	$Q_0 \leftarrow 0$, Restore A (As msb of A is 1)
3.	0011	1000	shift left A, Q
	0000	1000	$A \leftarrow A-M$
	0000	1001	$Q_0 \leftarrow 1$ (As msb of A is 0)
4.	0001	0010	shift left A, Q
	1110	0010	$A \leftarrow A-M$
	0001	0010	$Q_0 \leftarrow 0$, Restore A as (MSB of A is 1)

(1) → Remainder

(Quotient) = (2)

ex. (10/2)

M = 2 = 0010 = divisor

Q = 10 = 1010 ← dividend.

	A	Q	M = 0010
1	0000	1010	initial value
	0001	0100	shift left A, Q
	1111	0100	$A \leftarrow A-M$
	0001	0100	$Q_0 \leftarrow 0$, Restore A
2.	0010	1000	shift left A, Q
	0000	1000	$A \leftarrow A-M$

	A	Q	M = 0010
	0000	1001	$Q_0 \leftarrow 1$ (as msb of A is 0)
3.	00001	0011	shift left
	1111	0011	$A \leftarrow A - M$
	0001	0010	$Q_0 \leftarrow 0$, (Restore A) (As msb of A is 1)
4.	0010	0101	shift left.
	0000	0101	$A \leftarrow A - M$
	0000	0101	$Q_0 \leftarrow 1$ (As msb of A is 0)

$$\begin{array}{r} 0010 \\ +\ 1101 \\ \hline 11 \\ \hline 1110 \\ +\ 0001 \\ \hline 1111 \end{array}$$

Remainder ⇓ (0)

Quotient (5)

ex. (7/2)

M ⇒ 2 = divisor = 0010

Q ⇒ 7 = dividend = 0111

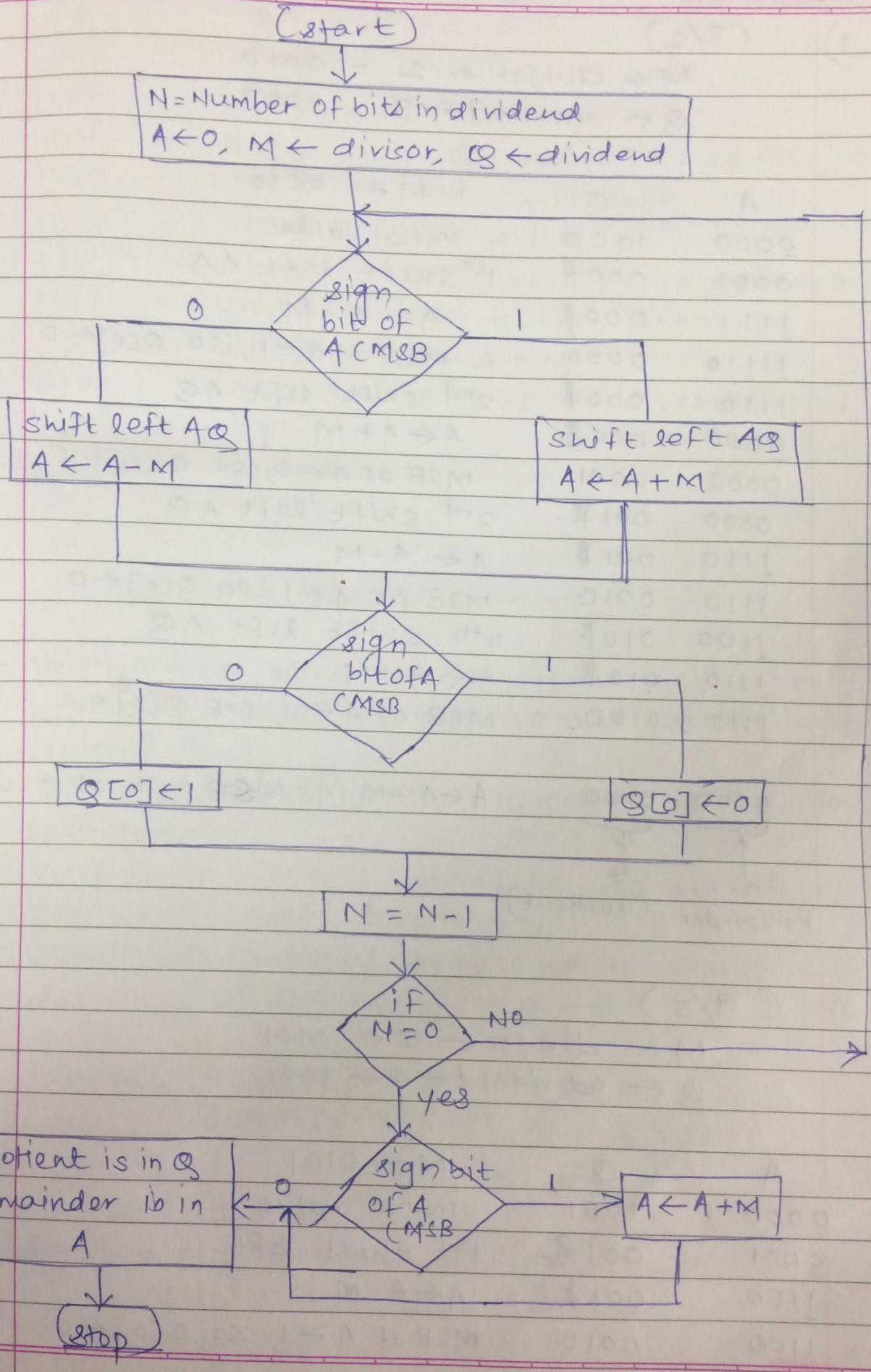
	A	Q	M = 0010
1	0000	0111	initialization
	0000	1110	shift left A, Q
	1110	1110	$A \leftarrow A - M$
	0000	1110	$Q \leftarrow 0$, Restore A (As msb of A is 1)

$M = 0010$

	A	Q	
2	0001	1100	shift left
	1111	1100	$A \leftarrow A - M$
	0001	1100	$Q_0 \rightarrow 0$, Restore A (As msb of A is
3	0011	1000	shift left
	0001	1000	$A \leftarrow A - M$
	0001	1001	$Q \leftarrow 1$ (As msb of A is 0)
4.	0011	0010	shift left
	0001	0010	$A \leftarrow A - M$
	0001	0011	$Q_0 \leftarrow 1$ (As msb of A is 0)

Remainder (1) — Quotient (3)

Non-restoring division algorithm



example:

1) (8/2)

M ← divisor ← 2 = 0010

Q ← dividend ← 8 = 1000

A	Q	M ← 0010
0000	1000	initial value
0001	000✗	1st shift left A,Q
1111	000✗	A ← A - M
1111	0000	MSB of A ← 1, so Q[0] ← 0
1110	000✗	2nd shift left AQ
0000	000✗	A ← A + M
0000	0001	MSB of A ← 0, so Q[0] ← 1
0000	001✗	3rd shift left AQ
1110	001✗	A ← A - M
1110	0010	MSB of A ← 1, so Q[0] ← 0
1100	010✗	4th shift left AQ
1110	010✗	A ← A + M
1110	0100	MSB of A ← 1, so Q[0] ← 0

0000 0100 A ← A + M, (As ~~MSB~~ msb of A is 1)

0 4

Remainder (Quotient)

2) (9/5)

M ← divisor ← 5 ← 0101

Q ← dividend ← 9 ← 1001

A	Q	M ← 0101
0000	1001	initial value
0001	001✗	1st shift left
1100	001✗	A ← A - M
1100	0010	MSB of A ← 1, so Q[0] ← 0

A	Q	
1000	010 ✱	2nd shift left
1101	010 ✱	A ← A+M
1101	0100	MSB of A ← 1 so Q[0] ← 0
1010	100 ✱	3rd shift left
1111	100 ✱	A ← A+M
1111	1000	MSB of A ← 1 so Q[0] ← 0
1111	000 ✱	4th shift left
0100	000 ✱	A ← A+M
0100	0001	MSB of A ← 0 so Q[0] ← 1

Remainder (4) — Quotient (1)

Floating point Representation:

- With a fixed point notation, it is possible to represent a range of negative and positive integers centered on 0.
- But fixed-point number representation has the limitations.
- very large numbers cannot be represented nor can very small fractions.
- Sometimes, fractional part of quotient in a division of two large numbers could be lost.

example $8000000000000 \Rightarrow 8 \times 10^{12}$

$0.0000000000008 \Rightarrow 0.8 \times 10^{-12}$

$\pm E$

$\Rightarrow \pm S \times B$

⇒ plus or minus ∈ sign

Significand S

Exponent E

Base B is implicit and need not be stored because it is same for all numbers.

- To represent very tiny or huge (big) numbers using fewer no. of bits in the memory of computer, floating point number representation is used.
- To have fixed and single representation of a number in floating point representation we have to normalize the number.
- Normalization rules for normalizing the numbers:

1) Integer part should be zero

2) if $0.d_1d_2d_3 \ldots d_n \times B^{\pm E}$ then $d_1 > 0$ and all $d_{i=2 \text{ to } n} \geq 0$

So for example considering three cases

1) 0.245×10^4

2) 0.0245×10^5

3) 2.45×10^3

Among above three example '2' example no 3) is having integer part as non-zero (2) and then in second example

0.0245×10^5

$0.d_1d_2d_3d_4 \times 10^5$

$d_1 \Rightarrow 0$

So that's why example no. number 2 and 3 are not normalized

But in case of 1) example

it is $(0.245 \times 10^4) \Rightarrow$ which has fulfilled all the rules which are implied for normalized

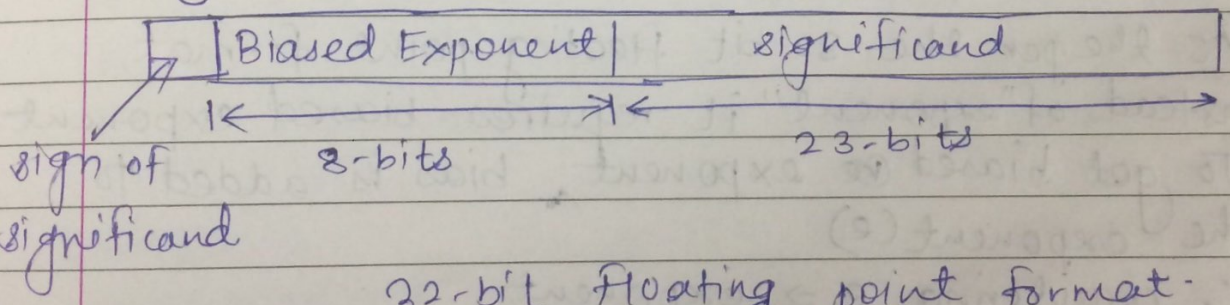
so 1) 0.245×10^4 is normalized number

floating point representation techniques

1) single precision (32-bit)
2) double precision (64-bit)

float and double data type in c are naming the 4 bytes and 8 bytes respectively
(32-bit) (64-bits)

1) single precision (32-bit)



32-bit floating point format.

- figure shows the 32-bit floating point representation
- Left most bit stored sign of number (0 = positive, 1 = negative)
- Exponent is value stored in the next 8 bits. The representation used is known as a biased representation.
- A fixed value called as bias, is substracted from the field to get true exponent value.
- A bias equals $(2^{K-1} - 1)$, where K is the number

of bits in binary exponent.

- example 1

$1.1010001 \times 2^{10100}$

0 10010011 10100010000 00000000000

1.6328125×2^{20}

Typically bias equals to $(2^{k-1}-1)$ (for 8-bits)

k = no. of bits in binary exponent
= 8-bits (As mentioned in figure)

so $(2^{k-1}-1)$

$= (2^{(8-1)}-1) = (2^7-1) = (128-1) = 127$..

so bias = 127

- As the per the 32-bit floating point format, instead of "exponent" it requires biased exponent
- To get biased exponent, bias is added to the exponent (e)

∴ considering e ⇒ exponent
E ⇒ biased exponent

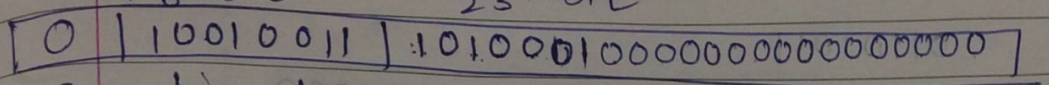
for example 1

E (biased exponent) = e + 127
= 20 + 127
= 147

so biased exponent = 147
= $147_{(10)} \Rightarrow 10010011_{(2)}$

↑
binary conversion fi
$= (147)$

and hence
32-bit representation (floating point representation)
for example 1 ie
$1.6328125 \times 2^{20} \Rightarrow 1.1010001 \times 2^{10100}$



for this kind of floating point representation normalized rules are as follows;

$\Rightarrow \pm 1.bbb\ldots b \times 2^{\pm E}$

where b is either binary digit (0 or 1).
Because the most significant bit is always one, it is unnecessary to store this bit.; rather it is implicit. Thus 23-bit is used to represent 23-bit significand (can be called as mantissa)

example 2
-1.6328125×2^{20}
$\Downarrow$
$-1.1010001 \times 2^{10100}$
$\Downarrow$
1 10010011 10100010000000000000000
S biased exponent significand

example 3. 1.6328125×2^{-20}
Here in this example exponent is -20 so

So, to calculate biased exponent, we will calculate as follows:

$E = e + 127$

$E = -20 + 127$

$= 107$

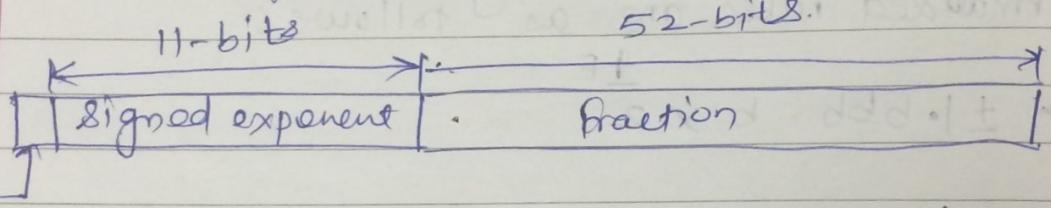
$\downarrow$

01101011

$\therefore$ 0 01101011 10100010000000000000000

S biased exponent — significand

double precision format



double precision floating point format:

64-bit

and value of bias

$(2^{k-1} - 1)$ =

k = no. of bit = 11

$(2^{11-1} - 1) = (2^{10} - 1) = (1024 - 1)$

$= (1023)$

bias value = (1023)

Requirement of biased exponent (bias)

With the exponent size (8 bit)

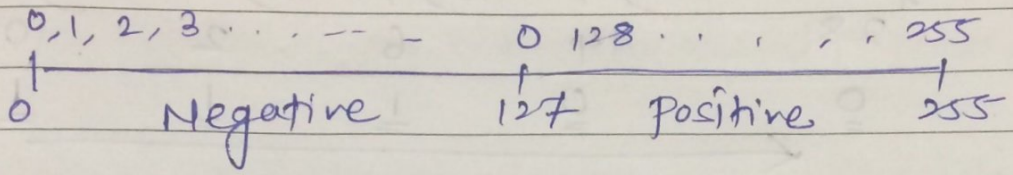
we can get numbers as $2^8 = 2^{56}$

$= [0, 1, 2 \ldots 255]$

$2^8 = 256 = [0, 1, 2, 3, 4, 5, \ldots 127, \ldots 255]$

unsigned : $[-127, -126, -125 \ldots -0, 0, 1, 2, \ldots 127]$

128 (+ve) | 128 (-ve)



- Exponent can be positive or negative
- If it is positive, then there is not a big issue, But whenever there is negative number then there can be issue in restoring.
- So to restore the exponent in computer hardware with a common representation (irrespective of sign) we consider the bias (middle value) that will be act as a factor for storing negative and positive no exponent in similar (predefined standard) format.

in case of 64-bit double precision format the exponent

$(2^{(11-1)}-1) = (2^{10}-1)$

$= (1024-1)$

$= (1023) \Rightarrow$ bias for 64-bit double precision format.

Example :-

Represent 28.125 in single and double precision format

Solution :-

Represent 28.125 in single and double precision format.

Solution: 28.125

- First convert 28 into binary

∴

14	7	3	1	0
2|28	2|14	2|7	2|3	2|1
28	14	6	2	0
0	0	1	1	1

So binary representation for $28 = (11100)_2$

- then we have convert 0.125 into binary

$0.125 \times 2 = 0.25$

$0.25 \times 2 = 0.5$

$0.5 \times 2 = 1.0$

$0.0 \times 2 = 0.0$

So binary representation (0.125)

$\Rightarrow .001$

∴ binary representation of 28.125

$= 11100.001$ —— (1)

So to normalize expression (1) we will do following procedure:-

$11100.001 \Rightarrow 1.1100001 \times 2^4$

So $\boxed{1.1100001} \times 2^4$ ← Exponent (e)

Significand or Mantissa

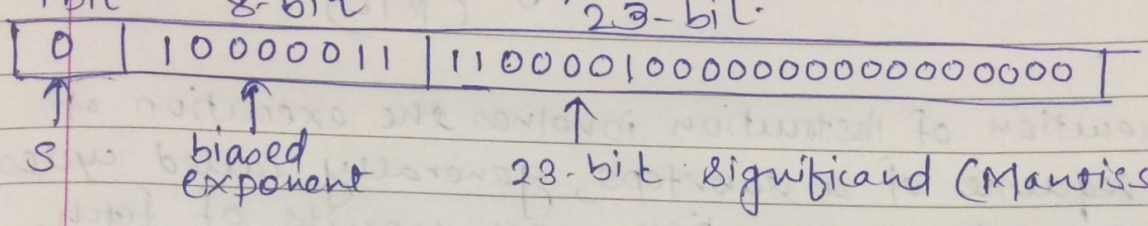
Base

and hence we will calculate biased exponent as

$e + 127 = 4 + 127 \Rightarrow 131$

∴ so binary representation of 131 = (10000011)

∴ single precision representation is as follows



example 2 Represent 245.125 in single precision format

solution :- 1) convert 245 into binary

$(245)_{10} \Rightarrow 11110101$

2) Convert (0.125) into binary

$(0.125)_{10} \Rightarrow (0.001)$

3) $(245.125)_{10} \Rightarrow (11110101.001)$

so to normalize

$\Downarrow$

$1.1110101001 \times 2^{7}$

and so

single precision format is as follows

~~0 11101001~~ we have calculate biased exponent

$= e + 127$

$= 7 + 127 = (134)_{10} \Rightarrow (10000110)_2$