Chapter – 6

Advanced Topics

6.1 Multiprocessing Systems

Traditionally, the computer has been viewed as a sequential machine. Most computer programming languages require the programmer to specify algorithms as sequence of instructions. Processor executes programs by executing machine instructions in a sequence and one at a time. This view of the computer has never been entirely true. At the micro operation level, multiple control signals are generated at the same time. Instruction pipelining and the overlapping of fetch & execute instructions from the same program in parallel.

As computer technology has evolved and as the cost of computer hardware has dropped, computer designers have sought more and more opportunities for parallelism, usually to enhance performance and availability. Multiprocessing is an example of parallelism which uses multiple CPUS sharing the common resources such as memory, storage device etc.

Characteristics of Multiprocessing system

- Contains two or more similar general purpose processors of comparable capability.
- All processors share access to common memory.
- Al processors share access to I/o devices either through the same channels that provide paths to the same devices.
- System is controlled by an integrated operating system that provides interaction between processors and their programs.

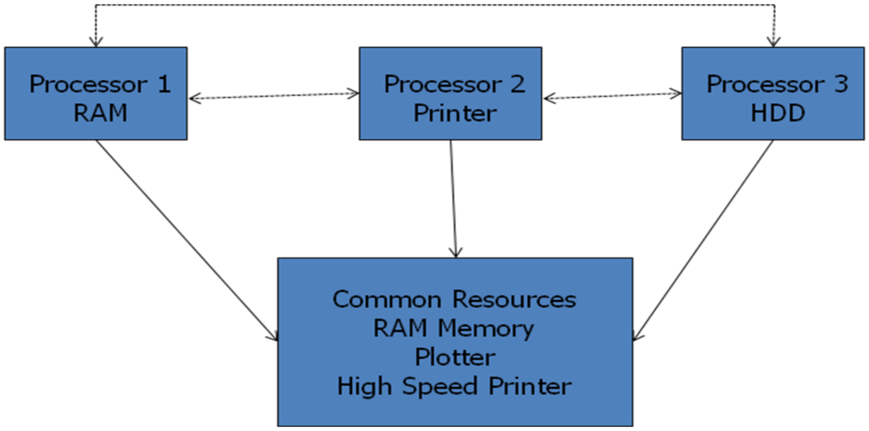

Fig: Organization of Multiprocessing System

Here the processors can communicate with each other through memory. The CPUs can directly exchange signals as indicated by dotted line. The organization of multiprocessor system can be divided into three types.

1. **Time shared or common bus:**

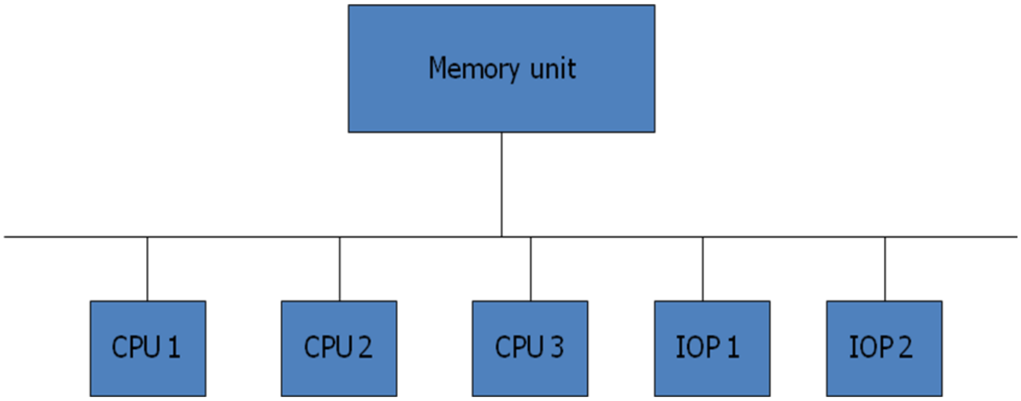

Fig: Time Shared System

There are number of CPUs, I/O modules and memory modules connected to the same bus. So the time shared system must distinguish the modules on the bus to determine the source and destination of the data. Any module in the bus can temporarily act as a master. When one module is controlling the bus, the other should be locked out. The access to each module is divided on the basis of time. The time shared multiprocessing system has the following advantages.

Simplicity:
The physical interface and the addressing time sharing logic of each processor remains the same as in a single processor system, so it is very simplest approach.

Flexibility:
It is easy to expand the system by attaching more CPUs to the bus.

Reliability:
The failure of any attached device should not the failure of the whole system.

Drawback:
The speed of the system is limited by the cycle time because all memory references must pass through the common bus.

2. **Multiport memory:**
   Each processor and /O module has dedicated path to each memory module this system has more performance and complexity than earlier one. For this system, it is possible to configure portions of memory as private to one or more CPUs and or I/O modules. This feature allows increasing security against unauthorized access and the storage of recovery routines in areas of memory not susceptible to modification by other processors.

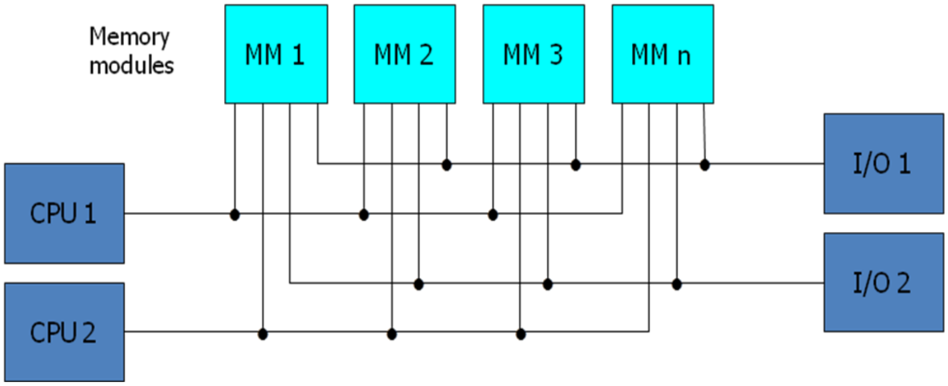

Fig: Multiport Memory System

3. **Central control unit :**
   It manages the transfer of separate data streams back and forth between independent modules like CPU, memory and I/o. the controller can buffer requests and perform arbitration and timing functions. It can also pass status and control messages between CPUS. All the co-ordination is concentrated in the central control unit un-disturbing the modules. It is more flexible and complex as well.

6.1.1 Real and Pseudo-Parallelism

- Traditionally, software has been written for serial computation:
  - To be run on a single computer having a single Central Processing Unit (CPU);
  - A problem is broken into a discrete series of instructions.
  - Instructions are executed one after another.
  - Only one instruction may execute at any moment in time.
- In the simplest sense, parallelism is the simultaneous use of multiple compute resources to solve a computational problem:
  - To be run using multiple CPUs
  - A problem is broken into discrete parts that can be solved concurrently
  - Each part is further broken down to a series of instructions
  - Instructions from each part execute simultaneously on different CPUs
- Real parallelism consists of the parallel modes of physical devices so that each can carry parallel operations to each other. Core parallelism consists of real parallelism. Multiple core processes which are physically different and performs their own operations in parallel.
- Pseudo parallelism consists of the same device carrying the parallel operation. We can logically manage the parallelism for system. Concurrent processing using parallelism is the pseudo parallelism which operates either in time division or using other types of parallel algorithms.

6.1.2 Flynn's Classification

- There are different ways to classify parallel computers. One of the more widely used classifications, in use since 1966, is called Flynn's classification.
- Flynn's classification distinguishes multi-processor computer architectures according to how they can be classified along the two independent dimensions of ***Instruction*** and

Data. Each of these dimensions can have only one of two possible states: ***Single*** or ***Multiple***.

- The matrix below defines the 4 possible classifications according to Flynn:

SISD **Single Instruction, Single Data**	**SIMD** **Single Instruction, Multiple Data**
MISD **Multiple Instruction, Single Data**	**MIMD** **Multiple Instruction, Multiple Data**

1. Single Instruction, Single Data (SISD):

- A serial (non-parallel) computer
- **Single Instruction:** Only one instruction stream is being acted on by the CPU during any one clock cycle
- **Single Data:** Only one data stream is being used as input during any one clock cycle
- Single instruction is performed on a single set of data in a sequential form.
- Deterministic execution
- This is the oldest and even today, the most common type of computer

Examples: older generation mainframes, minicomputers and workstations; most modern day PCs.

2. Single Instruction, Multiple Data (SIMD):

- A type of parallel computer
- **Single Instruction:** All processing units execute the same instruction at any given clock cycle
- **Multiple Data:** Each processing unit can operate on a different data element
- Best suited for specialized problems characterized by a high degree of regularity, such as graphics/image processing.
- Single Instruction is performed on multiple data. A good example is the 'For' loop statement. Over here instruction is the same but the data stream is different. Examples:
    - Processor Arrays: Connection Machine CM-2, MasPar MP-1 & MP-2, ILLIAC IV
    - Vector Pipelines: IBM 9000, Cray X-MP, Y-MP & C90, Fujitsu VP, NEC SX-2, Hitachi S820, ETA10

Most modern computers, particularly those with graphics processor units (GPUs) employ SIMD instructions and execution units.

3. Multiple Instruction, Single Data (MISD):

- A type of parallel computer
- **Multiple Instruction:** Each processing unit operates on the data independently via separate instruction streams.
- **Single Data:** A single data stream is fed into multiple processing units.
- N numbers of processors are working on different set of instruction on the same set of data.
- Few actual examples of this class of parallel computer have ever existed. One is the experimental Carnegie-Mellon C.mmp computer (1971).

- Some conceivable uses might be:
  - multiple frequency filters operating on a single signal stream

Multiple cryptography algorithms attempting to crack a single coded message.

4. Multiple Instruction, Multiple Data (MIMD):

- A type of parallel computer
- **Multiple Instruction:** Every processor may be executing a different instruction stream
- **Multiple Data:** Every processor may be working with a different data stream
- There is an interaction of N numbers of processors on a same data stream shared by all processors.
- Execution can be synchronous or asynchronous, deterministic or non-deterministic
- Currently, the most common type of parallel computer - most modern supercomputers fall into this category.
- Examples: most current supercomputers, networked parallel computer clusters and "grids", multi-processor SMP computers, multi-core PCs.

Note: many MIMD architectures also include SIMD execution sub-components

6.1.3 Instruction Level, Thread Level and Process Level Parallelism

Instruction Level Parallelism

Instruction-level parallelism (**ILP**) is a measure of how many of the operations in a computer program can be performed simultaneously. Consider the following program:

1. e = a + b
2. f = c + d
3. g = e * f

Operation 3 depends on the results of operations 1 and 2, so it cannot be calculated until both of them are completed. However, operations 1 and 2 do not depend on any other operation, so they can be calculated simultaneously. If we assume that each operation can be completed in one unit of time then these three instructions can be completed in a total of two units of time, giving an ILP of 3/2.

A goal of compiler and processor designers is to identify and take advantage of as much ILP as possible. Ordinary programs are typically written under a sequential execution model where instructions execute one after the other and in the order specified by the programmer. ILP allows the compiler and the processor to overlap the execution of multiple instructions or even to change the order in which instructions are executed.

How much ILP exists in programs is very application specific. In certain fields, such as graphics and scientific computing the amount can be very large. However, workloads such as cryptography exhibit much less parallelism.

Micro-architectural techniques that are used to exploit ILP include:

- **Instruction pipelining** where the execution of multiple instructions can be partially overlapped.
- **Superscalar execution**, VLIW, and the closely related Explicitly Parallel Instruction Computing concepts, in which multiple execution units are used to execute multiple instructions in parallel.

Thread Level Parallelism

Thread parallelism (also known as **Task Parallelism**, **function parallelism** and **control parallelism**) is a form of parallelization of computer code across multiple processors in parallel computing environments. Thread parallelism focuses on distributing execution processes (threads) across different parallel computing nodes. It contrasts to data parallelism as another form of parallelism.

It was later recognized that finer-grain parallelism existed with a single program. A single program might have several threads (or functions) that could be executed separately or in parallel. Some of the earliest examples of this technology implemented input/output processing such as direct memory access as a separate thread from the computation thread. A more general approach to this technology was introduced in the 1970s when systems were designed to run multiple computation threads in parallel. This technology is known as multi-threading (MT).

As a simple example, if we are running code on a 2-processor system (CPUs "a" & "b") in a parallel environment and we wish to do tasks "A" and "B" , it is possible to tell CPU "a" to do task "A" and CPU "b" to do task 'B" simultaneously, thereby reducing the run time of the execution.

Thread parallelism emphasizes the distributed (parallelized) nature of the processing (i.e. threads), as opposed to the data (data parallelism). Most real programs fall somewhere on a continuum between Thread parallelism and Data parallelism.

The pseudocode below illustrates task parallelism:

```
program:
...
if CPU="a" then
  do task "A"
else if CPU="b" then
  do task "B"
end if
...
end program
```

The goal of the program is to do some net total task ("A+B"). If we write the code as above and launch it on a 2-processor system, then the runtime environment will execute it as follows.

- In an SPMD system, both CPUs will execute the code.
- In a parallel environment, both will have access to the same data.
- The "if" clause differentiates between the CPU's. CPU "a" will read true on the "if" and CPU "b" will read true on the "else if", thus having their own task.
- Now, both CPU's execute separate code blocks simultaneously, performing different tasks simultaneously.

Code executed by CPU "a":

```
program:
...
do task "A"
...
end program
```

```
Code executed by CPU "b":
program:
...
do task "B"
...
end program
```

This concept can now be generalized to any number of processors.

Data parallelism

Data parallelism is parallelism inherent in program loops, which focuses on distributing the data across different computing nodes to be processed in parallel. "Parallelizing loops often leads to similar (not necessarily identical) operation sequences or functions being performed on elements of a large data structure." Many scientific and engineering applications exhibit data parallelism.
A loop-carried dependency is the dependence of a loop iteration on the output of one or more previous iterations. Loop-carried dependencies prevent the parallelization of loops. For example, consider the following pseudocode that computes the first few Fibonacci numbers:

```
PREV1 := 0
PREV2 := 1
do:
  CUR := PREV1 + PREV2
  PREV1 := PREV2
  PREV2 := CUR
while (CUR < 10)
```

This loop cannot be parallelized because CUR depends on itself (PREV2) and PREV1, which are computed in each loop iteration. Since each iteration depends on the result of the previous one, they cannot be performed in parallel. As the size of a problem gets bigger, the amount of data-parallelism available usually does as well.

Process Level Parallelism

Process-level parallelism is the use of one or more central processing units (CPUs) within a single computer system. The term also refers to the ability of a system to support more than one processor and/or the ability to allocate tasks between them. There are many variations on this basic theme, and the definition of process level parallelism can vary with context, mostly as a function of how CPUs are defined (multiple cores on one die, multiple dies in one package, multiple packages in one system unit, etc.).

This varies, depending upon who you talk to. In the past, a CPU (Central Processing Unit) was a singular execution component for a computer. Then, multiple CPUs were incorporated into a node. Then, individual CPUs were subdivided into multiple "cores", each being a unique execution unit. CPUs with multiple cores are sometimes called "sockets". The result is a node with multiple CPUs, each containing multiple cores.

- During the past 20+ years, the trends indicated by ever faster networks, distributed systems, and multi-processor computer architectures (even at the desktop level) clearly show that **parallelism is the future of computing**.

- In this same time period, there has been a greater than 1000x increase in supercomputer performance, with no end currently in sight.

6.1.4 Inter-process Communication, Resource Allocation and Deadlock

Inter-process communication

In computing, **Inter-process communication** (**IPC**) is a set of methods for the exchange of data among multiple threads in one or more processes. Processes may be running on one or more computers connected by a network. IPC methods are divided into methods for message passing, synchronization, shared memory, and remote procedure calls (RPC). The method of IPC used may vary based on the bandwidth and latency of communication between the threads, and the type of data being communicated.

There are several reasons for providing an environment that allows process cooperation:

- Information sharing
- Speedup
- Modularity
- Convenience
- Privilege separation

IPC may also be referred to as inter-thread communication and inter-application communication.

The combination of IPC with the address space concept is the foundation for address space independence/isolation.

The single operating system controls the use of system resources in a multiprocessing environment. In this system, multiple jobs or process may be active at one time. The responsibility of operating system or system software is to schedule the execution and to allocate resources. The functions of multiprocessor operating system are:

- An interface between users and machine
- Resource management
- Memory management
- Prevent deadlocks
- Abnormal program termination
- Process scheduling
- Managers security

Resource Allocation

In computing, **resource allocation** is necessary for any application to be run on the system. When the user opens any program this will be counted as a process, and therefore requires the computer to allocate certain resources for it to be able to run. Such resources could be access to a section of the computer's memory, data in a device interface buffer, one or more files, or the required amount of processing power.

A computer with a single processor can only perform one process at a time, regardless of the amount of programs loaded by the user (or initiated on start-up). Computers using single processors appear to be running multiple programs at once because the processor quickly alternates between programs, processing what is needed in very small amounts of time. This process is known as multitasking or *time slicing*. The time allocation is automatic, however higher or lower priority may be given to certain processes, essentially giving high priority programs more/bigger **slices** of the processor's time.

On a computer with multiple processors different processes can be allocated to different processors so that the computer can truly multitask. Some programs, such as Adobe Photoshop, which can require intense processing power, have been coded so that they are able to run on more than one processor at once, thus running more quickly and efficiently.

Deadlock

A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting processes may never again change state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.

Processes need access to resources in reasonable order. Suppose a process holds resource A and requests resource B, at same time another process holds B and requests A; both are blocked and remain in deadlock.

A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause. Usually the event is release of a currently held resource. None of the processes can run, release resources and then be awakened.

6.1.5 Features of Typical Operating System

Operating system is a system s/w. which control and manage the hardware through bios (basic i/o system). Bios is the part of operating system which is built in the system and it run application s/w on a hardware. It is an interface b/w user and system/s is a complex s/w. In a simple word operating system is that which is active on a ram before switch of the system. The operating systems key elements are

- A technical layer of software for driving software components.
- A file system for organizing and accessing files logically.
- Simple command language enabling users to run their own programs and manipulating files.

OS Features

1. **Process Management**

Operating system manages the process on hardware level and user level. To create, block, terminate, request for memory, Forking, releasing of memory etc. in multi tasking operating system the multiple processes can be handle and many processes can be create ,block and terminate ,run etc. it allow multiple process to exist at any time and where only one process can execute at a time. The other process may be performing I/O and waiting. The process manager implements the process abstraction and creating the model to use the CPU. It is the major part of operating system and its major goal is scheduling, process synchronization mechanism and deadlock strategy

2. File Managent

Is a computer program and provide interface with file system. it manage the file like creation, deletion, copy, rename etc files typically display in hierarchical and some file
manager provide network connectivity like ftp, nfs, smb or webdav.

3. Memory Management

Is a way to control the computer memory on the logic of data structure? It provides the way to program to allocate in main memory at their request. The main purpose of this
manager is; It allocates the process to main memory; minimize the accessing time and process address allocate in a location of primary memory. The feature of memory manager on multi tasking is following.

- Relocation
- Protection
- Sharing
- Logical organization
- Physical organization

4. Device Management

Allow the user to view its capability and control the device through operating system. Which device may be enabling or disable or install or ignore the functionality of the device. In Microsoft windows operating system the control panel applet is the device manager .it also built on web application server model. Device Manager provides three graphical user interfaces (GUIs). Device manager manage the following:

- Device configuration
- Inventory collection
- S/W distribution
- Initial provisioning

5. Resource management

Is a way to create, manage and allocate the resources? Operating system is a responsible to all activities which is done in computer. Resource manager is the major part of operating system .the main concept of operating system is managing and allocate the resources. The resources of the computer are storage devices, communication and I/O devices etc. these all resources manage and allocate or de allocate by resource manager.

Services Of Operating System

An operating system provides an environment for the execution of programs. It provides certain services to programs and to the users of those programs. The specific services provided, of course, differ from one operating system to another, but we can identify common classes. These operating-system services are provided for the convenience of the programmer, to make the programming task easier.

Program execution: The system must be able to load a program into memory and to run that program. The program must be able to end its execution, either normally or abnormally (indicating error).

I/O operations: A running program may require I/O. This 1/0 may involve a file or an I/O device. For specific devices, special functions may be desired (such as to rewind a tape drive, or to blank a CRT screen). For efficiency and protection, users usually cannot control 1/0 devices directly. Therefore, the
operating system must provide a means to do I/O.

File-system manipulation: The file system is of particular interest. Obviously, programs need to read and write files. Programs also need to create and delete files by name.

Communications: In many circumstances, one process needs to exchange information with another process. Such communication can occur in two major ways. The first takes place between processes that are executing on the same computer; the second takes place between processes that are executing on different computer systems that are tied together by a computer network. Communications may be implemented via shared memory, or by the technique of message passing, in which packets of
information are moved between processes by the operating system.

Error detection: The operating system constantly needs to be aware of possible errors. Errors may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network, or lack of paper in the printer),
and in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too-great use of CPU time). For each type of error, the operating system should take the appropriate action to ensure correct and consistent computing. In addition, another set of operating-system functions exists not for helping the user, but for ensuring the efficient operation of the system itself. Systems with multiple users can gain efficiency by sharing the computer resources among the users.

Resource allocation: When multiple users are logged on the system or multiple jobs are running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system. Some (such as CPU cycles, main memory, and file storage) may have special allocation code, whereas others (such as I/O devices) may have much more general request and release code. For instance, in determining how best to use the CPU, operating systems have CPU-scheduling routines that take into account the speed of the CPU, the jobs that must be executed, the number of registers available, and other factors. There might also be routines to allocate a tape drive for use by a job. One such routine locates an unused tape drive and marks an internal table
to record the drive's new user. Another routine is used to clear that table. These routines may also allocate plotters, modems, and other peripheral devices.

Accounting: We want to keep track of which users use how many and which kinds of computer resources. This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics. Usage statistics may be a valuable tool for researchers who wish to reconfigure the system to improve computing services.

Protection: The owners of information stored in a multiuser computer system may want to control use of that information. When several disjointed processes execute concurrently, it should not be possible for one process to interfere with the others, or with the operating system itself. Protection involves ensuring that all access to system resources is controlled. Security of the system from outsiders is also important. Such security starts with each user having to authenticate himself to the system, usually by means of a password, to be allowed access to the resources. It extends to defending external 1/0 devices, including modems and network adapters, from invalid access attempts, and to recording all such connections for detection of break-ins. If a system is to be protected and secure, precautions must be instituted throughout it. **A** chain is only as strong as its weakest link.

6.2 Different Microprocessor Architectures

6.2.1 Register Based and Accumulator Based Architecture

Accumulator Based Architecture

- Accumulator is a most significant register then compared to other registers and most of the arithmetic and logic operations are performed using the accumulator .performed via the accumulator.
- The internal architecture of 8085 shows that the registers B,C,D,E,H and L are connected with the ALU through the accumulator and temporary register.
- Data can only enter into the ALU from accumulator and the out pot of the ALU can be stared in accumulator through data bus.

Register Based Architecture

- All the arithmetic and logical operation consists of the operands of any registers (ABCD etc). The input/output operations here register A and register B are similar.
- The internal architecture of 8086 shows that the registers A, B, C, D and others directly with the ALU.
- Data can enter into the ALU from any registers.
- For I/O operation register A only can be used.
- The advantage of register based architecture is that extendibility and flexibility in programming.
- The processor will be enhanced in register based architecture.
- The disadvantage is requirement of complex circuitry.

6.2.2 RISC and CISC Architectures

CISC Based Architecture

Complex instruction set computers is the acronym for CISC and machine based on this architecture have complex instructions. Most of the personal computers which are used today are based on CISC architecture. Following are the characteristics of CISC machines.

- CISC machines have complex instructions which need a large cycles for execution.
- The transfer of data among the register is much faster than among memory and processor. In CISC, various instructions based on memory and processer so the processing speed gets reduced.

- Pipelining is the process of fetching one instruction when another instruction is executing in parallel. Due to complex instruction this feature cannot be heavily used in CISC machines.
- Micro-operations form the instruction and instruction form he micro-program which is written in control memory to perform timing and sequencing of the micro-operations implemented in CISC.
- CISC machines have large number of complex instructions based on multiple numbers of addressing modes.
- CISC machines processer does not consist of large number of registers due to large cost. So these machines have to perform various memory read and write operations.
- CIS Machines are preferable where the speed of processer is not the prime issue and where general applications are to be handled. Processers like 8085, 8086, 8086,8086,8086,8086 etc are based on CISC processers and even today's pc.

RISC based architecture:

The term RISC represents reduced instruction set computing / computers. It focused on a small set of instruction which simplifies the hardware design and improves the processer performance. Generally RISC processers include the following feathers.

- The number of instructions is minimized i.e. less than 100 and each can be executed in a single clock cycle the data path cycle time is the time required to fetch the operations from the registers ,run them through ALU and store the result back to register which is very small in RISC.
- the number of addressing modes is relatively low i.e. less than 3 and only few instructions are memory referencing link lode and store so that RISC permits heavy pipelining.
- The processing is register intensive, meaning it includes many more registers and most of the computing is performed using registers range from 32 registers to more than 100 registers.
- There is no micro-programs in RISC machines for interpreting the instructions. The most instructions are directly executed by the hardware.
- Design considerations include support for high level languages through appropriate selection of instruction and optimization of compilers.
- Usually in scientific and research oriented tasks where the speed of processers is apex issue, RISC machines are used and replacing CISC now due to reduce the price of hardware.
- RISC based system are R4400SC from MIPS, PA7100 from HP, Power PC from Apple, IBM and Motorola, super sparc from sun, i860^{TM} from Intel etc.

6.2.3 Digital Signal Processors

The real time signals such as pressure, temperature, voice are continuous time varying signals are known as analog signals. The process of conversion of the analog signals into discrete signal means digital signal which reduces the redundancy and make more immune to noise is called digital signal processing .the digital signal processors take input the digital data for that every analog data is to be converted into digital by using A\D converter. For example microphone converts sound into electrical energy i.e. analog, it is sampled by A\D converter and converted into digital one which is fed to the DSP processor. After performing these data DSP can transfer the discrete data to D\A converter which further to speaker to convert electrical signal to sound.

The whole function is carried out by DSP processers using hardware like microphone, transducer, A\D converter, D\A converter, speaker etc and software like C or MATLAB which carried out FFT (fast Fourier transform). DSP processors have low processing speed due to very curtail signals need to be operated. The micro-processers and computers we used today are based on non Neumann architecture where the instruction defines both the operation and data.

So the DSP processors should be fast processing and for that we need to design best architecture which follows Harvard Architecture where separate buses for instructions and data are used. DSP chips are specially designed for particular application and they are not used for general type of processing like microprocessors do. There are very few manufacturers for DSP chip, one of them is the Texas instruments, USA. Its TMS320C series is worldwide popular and can be used for implementing various types of signal processing application.