CS1252 – COMPUTER ORGANIZATION AND ARCHITECTURE
(Common to CSE and IT)

L T P C
3 1 0 4

UNIT I BASIC STRUCTURE OF COMPUTERS 9

Functional units – Basic operational concepts – Bus structures – Performance and metrics – Instructions and instruction sequencing – Hardware – Software interface – Instruction set architecture – Addressing modes – RISC – CISC – ALU design – Fixed point and floating point operations.

UNIT II BASIC PROCESSING UNIT 9

Fundamental concepts – Execution of a complete instruction – Multiple bus organization – Hardwired control – Micro programmed control – Nano programming.

UNIT III PIPELINING 9

Basic concepts – Data hazards – Instruction hazards – Influence on instruction sets – Data path and control considerations – Performance considerations – Exception handling.

UNIT IV MEMORY SYSTEM 9

Basic concepts – Semiconductor RAM – ROM – Speed – Size and cost – Cache memories – Improving cache performance – Virtual memory – Memory management requirements – Associative memories – Secondary storage devices.

UNIT V I/O ORGANIZATION 9

Accessing I/O devices – Programmed I/O – Interrupts – Direct memory access – Buses – Interface Circuits – Standard I/O interfaces (PCI, SCSI, and USB) – I/O Devices and processors.

L: 45 T: 15 Total: 60

TEXT BOOKS
1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, 5th Edition, Tata Mc-Graw Hill, 2002.
2. Heuring, V.P. and Jordan, H.F., “Computer Systems Design and Architecture”, 2nd Edition, Pearson Education, 2004.

REFERENCES
1. Patterson, D. A., and Hennessy, J.L., “Computer Organization and Design:The Hardware/Software Interface”, 3rd Edition, Elsevier, 2005.
2. William Stallings, “Computer Organization and Architecture – Designing for Performance”, 6th Edition, Pearson Education, 2003.
3. Hayes, J.P., “Computer Architecture and Organization”, 3rd Edition, Tata Mc-Graw Hill, 1998.

UNIT I

BASIC STRUCTURE OF COMPUTERS

- Functional units
- Basic operational concepts
- Bus structures
- Performance and metrics
- Instructions and instruction sequencing
- Hardware
- Software interface
- Instruction set architecture
- Addressing modes
- RISC
- CISC
- ALU design
- Fixed point and floating point operations

BASIC STRUCTURE OF COMPUTERS:

Computer Organization:

It refers to the operational units and their interconnections that realize the architectural specifications.

It describes the function of and design of the various units of digital computer that store and process information.

Computer hardware:

Consists of electronic circuits, displays, magnetic and optical storage media, electromechanical equipment and communication facilities.

Computer Architecture:

- It is concerned with the structure and behaviour of the computer.
- It includes the information formats, the instruction set and techniques for addressing memory.

Functional Units

A computer consists of 5 main parts.

- Input
- Memory
- Arithmetic and logic
- Output
- Control Units

Functional units of a Computer

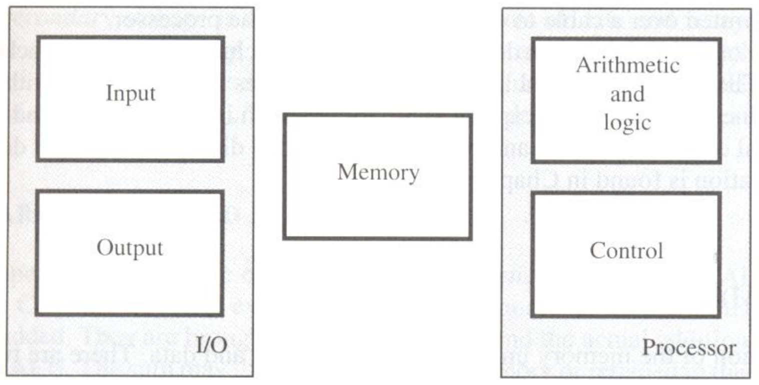


Figure 1.1 Basic functional units of a computer.

- Input unit accepts coded information from human operators, from electromechanical devices such as keyboards, or from other computers over digital communication lines.
- The information received is either stored in the computers memory for later reference or immediately used by the arithmetic and logic circuitry to perform the desired operations.
- The processing steps are determined by a program stored in the memory.
- Finally the results are sent back to the outside world through the output unit.
- All of these actions are coordinated by the control unit.
- The list of instructions that performs a task is called a program.
- Usually the program is stored in the memory.
- The processor then fetches the instruction that make up the program from the memory one after another and performs the desire operations.

1.1 Input Unit:

- Computers accept coded information through input units, which read the data.
- Whenever a key is pressed, the corresponding letter or digit is automatically translated into its corresponding binary code and transmitted over a cable to either the memory or the processor.

Some input devices are

- ✓ Joysticks
- ✓ Trackballs
- ✓ Mouses
- ✓ Microphones (Capture audio input and it is sampled & it is converted into digital codes for storage and processing).

1.2.Memory Unit:

It stores the programs and data.

There are 2 types of storage classes

- ✓ Primary
- ✓ Secondary

Primary Storage:

- It is a fast memory that operates at electronic speeds.
- Programs must be stored in the memory while they are being executed.
- The memory contains large no of semiconductor storage cells.
- Each cell carries 1 bit of information.
- The Cells are processed in a group of fixed size called Words.
- To provide easy access to any word in a memory,a distinct address is associated with each word location.

- Addresses are numbers that identify successive locations.
- The number of bits in each word is called the word length.
- The word length ranges from 16 to 64 bits.
- There are 3 types of memory.They are

  - ✓ RAM(Random Access Memory)
  - ✓ Cache memory
  - ✓ Main Memory

RAM:

Memory in which any location can be reached in short and fixed amount of time after specifying its address is called RAM.

Time required to access 1 word is called Memory Access Time.

Cache Memory:

The small,fast,RAM units are called Cache. They are tightly coupled with processor to achieve high performance.

Main Memory:

The largest and the slowest unit is called the main memory.

1.3. ALU:

Most computer operations are executed in ALU.

Consider a example,

Suppose 2 numbers located in memory are to be added. They are brought into the processor and the actual addition is carried out by the ALU. The sum may then be stored in the memory or retained in the processor for immediate use.

Access time to registers is faster than access time to the fastest cache unit in memory.

1.4. Output Unit:

Its function is to send the processed results to the outside world. eg.Printer

Printers are capable of printing 10000 lines per minute but its speed is comparatively slower than the processor.

1.5. Control Unit:

The operations of Input unit, output unit, ALU are co-ordinate by the control unit.

The control unit is the Nerve centre that sends control signals to other units and senses their states.

Data transfers between the processor and the memory are also controlled by the control unit through timing signals.

The operation of computers are,

- ▪ The computer accepts information in the form of programs and data through an input unit and stores it in the memory.

- Information stored in the memory is fetched, under program control into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities inside the machine are directed by the control unit.

BASIC OPERATIONAL CONCEPTS:

The data/operands are stored in memory.

The individual instruction are brought from the memory to the processor, which executes the specified operation.

Eg:1 Add LOC A ,R1

Instructions are fetched from memory and the operand at LOC A is fetched. It is then added to the contents of R0, the resulting sum is stored in Register R0.

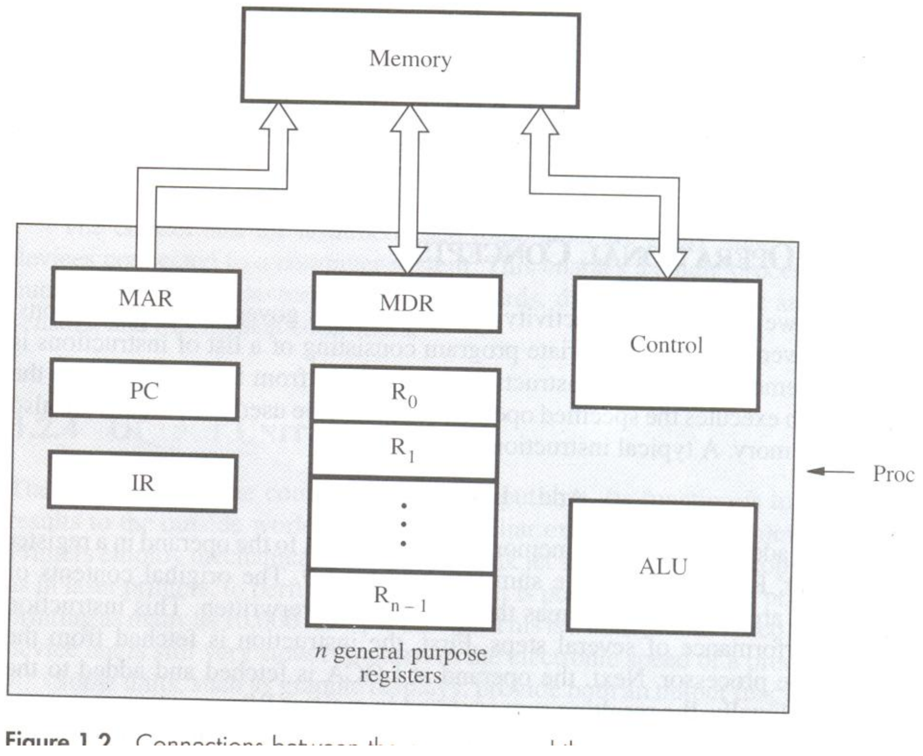
Eg:2 Load LOC A, R1

Transfer the contents of memory location A to the register R1.

Eg:3 Add R1 ,R0

Add the contents of Register R1 & R0 and places the sum into R0.

Fig:Connection between Processor and Main Memory


Memory
MAR
MDR
Control
PC
R_0
R_1
IR
Proc
ALU
R_{n-1}
n general purpose registers
Figure 1.2 Connections between th

- Memory Address Register(MAR)
- Memory Data Register(MDR)

Instruction Register (IR):

- ➢ It holds the instruction that is currently being executed.
- ➢ It generates the timing signals.

Program Counter (PC):

It contains the memory address of the next instruction to be fetched for execution.

Memory Address Register (MAR):

It holds the address of the location to be accessed.

Memory Data Register (MDR):

- ➢ It contains the data to written into or read out of the address location.
- ➢ MAR and MDR facilitates the communication with memory.

Operation Steps:

- ➢ The program resides in memory. The execution starts when PC is point to the first instruction of the program.
- ➢ MAR read the control signal.
- ➢ The Memory loads the address word into MDR.The contents are transferred to Instruction register. The instruction is ready to be decoded & executed.

Interrupt:

- Normal execution of the program may be pre-empted if some device requires urgent servicing.
- Eg...Monitoring Device in a computer controlled industrial process may detect a dangerous condition.
- In order to deal with the situation immediately, the normal execution of the current program may be interrupted & the device raises an interrupt signal.
- The processor provides the requested service called the Interrupt Service Routine(ISR).

- ISR save the internal state of the processor in memory before servicing the interrupt because interrupt may alter the state of the processor.
- When ISR is completed, the state of the processor is restored and the interrupted program may continue its execution.

BUS STRUCTURES:

A group of lines that serves as the connection path to several devices is called a Bus.
A Bus may be lines or wires or one bit per line.
The lines carry data or address or control signal.

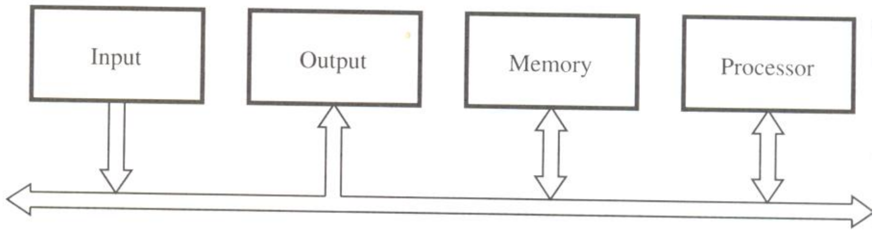


Figure 1.3 Single-bus structure.

There are 2 types of Bus structures. They are
Single Bus Structure
Multiple Bus Structure

3.1.Single Bus Structure:

It allows only one transfer at a time.
It costs low.
It is flexible for attaching peripheral devices.
Its Performance is low.

3.2.Multiple Bus Structure:

It allows two or more transfer at a time.
It costs high.
It provides concurrency in operation.
Its Performance is high.

Devices Connected with Bus	**Speed**
Electro-mechanical decvices (Keyboard,printer)	Slow
Magnetic / optical disk	High
Memory & processing units	Very High

The Buffer Register when connected with the bus, carries the information during transfer.
The Buffer Register prevents the high speed processor from being locked to a slow I/O device during a sequence of data transfer.

SOFTWARE:

System Software is a collection of programs that are executed as needed to perform function such as,

- Receiving & Interpreting user commands.
- Entering & editing application program and storing them as files in secondary Storage devices.
- Managing the storage and retrieval of files in Secondary Storage devices.
- Running the standard application such as word processor, games, and spreadsheets with data supplied by the user.
- Controlling I/O units to receive input information and produce output results.
- Translating programs from source form prepared by the user into object form.
- Linking and running user-written application programs with existing standard library routines.

Software is of 2 types.They are

- Application program
- System program

Application Program:

It is written in high level programming language(C,C++,Java,Fortran)
The programmer using high level language need not know the details of machine program instruction.

System Program:(Compiler,Text Editor,File)
Compiler:

It translates the high level language program into the machine language program.

Text Editor:

It is used for entering & editing the application program.

System software Component ->OS(OPERATING SYSTEM)
Operating System :

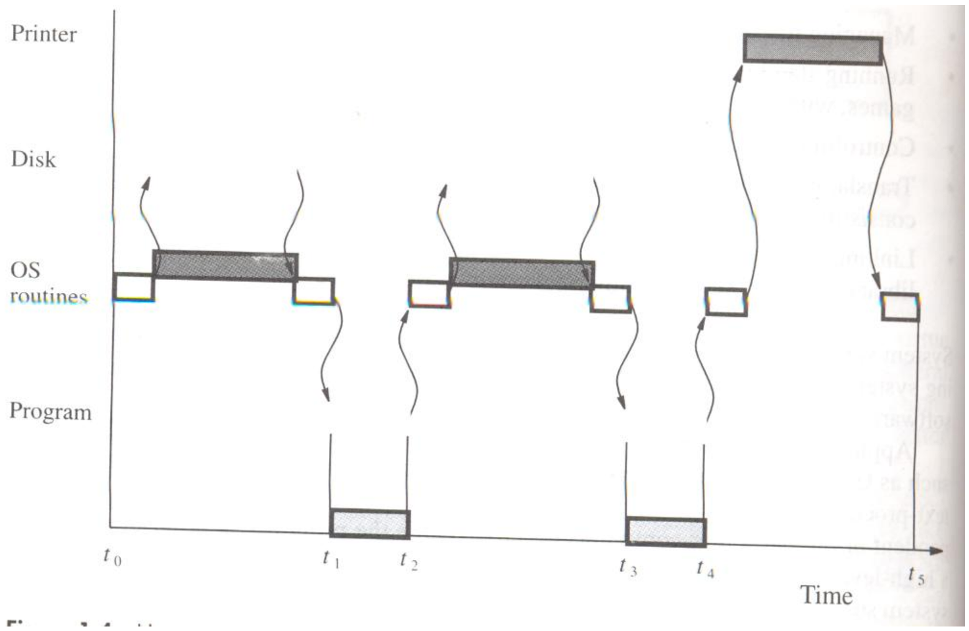
It is a large program or a collection of routines that is used to control the sharing of and interaction among various computer units.

Functions of OS:

- Assign resources to individual application program.
- Assign memory and magnetic disk space to program and data files.

- Move the data between the memory and disk units.
- Handles I/O operation.

Fig:User Program and OS routine sharing of the process



Steps:

1. The first step is to transfer the file into memory.
2. When the transfer is completed, the execution of the program starts.
3. During time period ‘t0’ to ‘t1’ , an OS routine initiates loading the application program from disk to memory, wait until the transfer is complete and then passes the execution control to the application program & print the results.
4. Similar action takes place during ‘t2’ to ‘t3’ and ‘t4’ to ‘t5’.
5. At ‘t5’, Operating System may load and execute another application program.
6. Similarly during ‘t0’ to ‘t1’ , the Operating System can arrange to print the previous program’s results while the current program is being executed.
7. The pattern of managing the concurrent execution of the several application programs to make the best possible use of computer resources is called the multi-programming or multi-tasking.

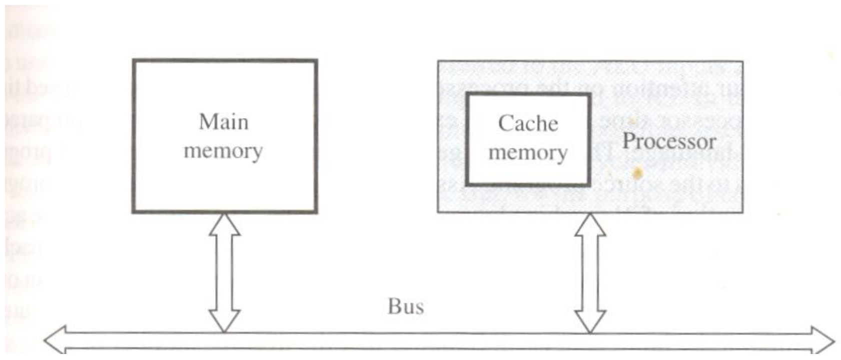
PERFORMANCE:

For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinate way.

Elapsed Time→the total time required to execute the program is called the elapsed time. It depends on all the units in computer system.

Processor Time→The period in which the processor is active is called the processor time. It depends on hardware involved in the execution of the instruction.

Fig: The Processor Cache



A Program will be executed faster if the movement of instruction and data between the main memory and the processor is minimized, which is achieved by using the Cache.

Processor clock:

Clock→The Processor circuits are controlled by a timing signal called a clock.
Clock Cycle→The cycle defines a regular time interval called clock cycle.

$$\boxed{\text{Clock Rate}, R = 1/P}$$

Where, P→Length of one clock cycle.

Basic Performance Equation:

$$\boxed{T = (N*S)/R}$$

Where, T→Performance Parameter
R→Clock Rate in cycles/sec

N→Actual number of instruction execution
S→Average number of basic steps needed to execute one machine instruction.
To achieve high performance,

N,S<R

Pipelining and Superscalar operation:

Pipelining→A Substantial improvement in performance can be achieved by overlapping the execution of successive instruction using a technique called pipelining.

Superscalar Execution →It is possible to start the execution of several instruction in everey clock cycles (ie)several instruction can be executed in parallel by creating parallel paths.This mode of operation is called the Superscalar execution.

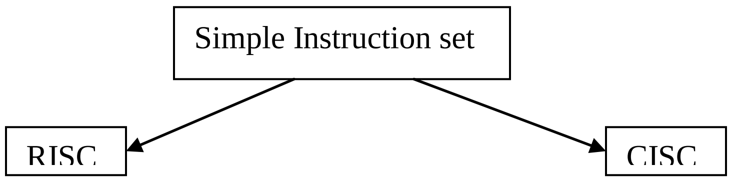
Clock Rate:

There are 2 possibilities to increase the clock rate(R).They are,

- Improving the integrated Chip(IC) technology makes logical circuits faster.
- Reduce the amount of processing done in one basic step also helps to reduce the clock period P.

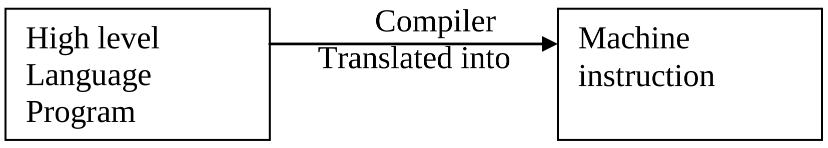
Instruction Set:CISC AND RISC:

The Complex instruction combined with pipelining would achieve the best performance.
It is much easier to implement the efficient pipelining in processor with simple instruction set.



(Reduced Instruction Set Computer)
It is the design of the instruction set of a processor with simple instruction

(Complex Instruction Set Computer)
It is the design of the instruction set of a processor with complex instruction.



Functions of Compiler:

- The compiler re-arranges the program instruction to achieve better performance.
- The high quality compiler must be closely linked to the processor architecture to reduce the total number of clock cycles.

Performance Measurement:

- The Performance Measure is the time it takes a computer to execute a given benchmark.
- A non-profit organization called SPEC(System Performance Evaluation Corporation) selects and publishes representative application program.

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

The Overall SPEC rating for the computer is given by,

$$\boxed{\text{SPEC rating} = \left(\prod_{i=1}^{n} \text{SPEC}i \right)^{1/n}}$$

<u>INSTRUCTION AND INSTRUCTION SEQUENCING</u>

A computer must have instruction capable of performing the following operations. They are,

- ➢ Data transfer between memory and processor register.
- ➢ Arithmetic and logical operations on data.
- ➢ Program sequencing and control.
- ➢ I/O transfer.

Register Transfer Notation:

The possible locations in which transfer of information occurs are,

- Memory Location
- Processor register
- Registers in I/O sub-system.

Location	**Hardware Binary Address**	**Eg**	**Description**
Memory	LOC,PLACE,A,VAR2	R1←[LOC]	The contents of memory location are transferred to. the processor register.
Processor	R0,R1,….	[R3]←[R1]+[R2]	Add the contents of register R1 &R2 and places .their sum into register R3.It is .called Register Transfer Notation.
I/O Registers	DATAIN,DATAOUT		Provides Status information

Assembly Language Notation:

Assembly Language Format	**Description**
Move LOC,R1	Transfers the contents of memory location to the processor register R1.
Add R1,R2,R3	Add the contents of register R1 & R2 and places their sum into register R3.

Basic Instruction Types:

Instruction Type	**Syntax**	**Eg**	**Description**
Three Address	Operation Source1,Source2,Destination	Add A,B,C	Add values of variable A ,B & place the result into c.
Two Address	Operation Source,Destination	Add A,B	Add the values of A,B & place the result into B.
One Address	Operation Operand	Add B	Content of accumulator add with content of B.

Instruction Execution and Straight–line Sequencing:

Instruction Execution:

There are 2 phases for Instruction Execution. They are,

- Instruction Fetch
- Instruction Execution

Instruction Fetch:

The instruction is fetched from the memory location whose address is in PC.This is placed in IR.

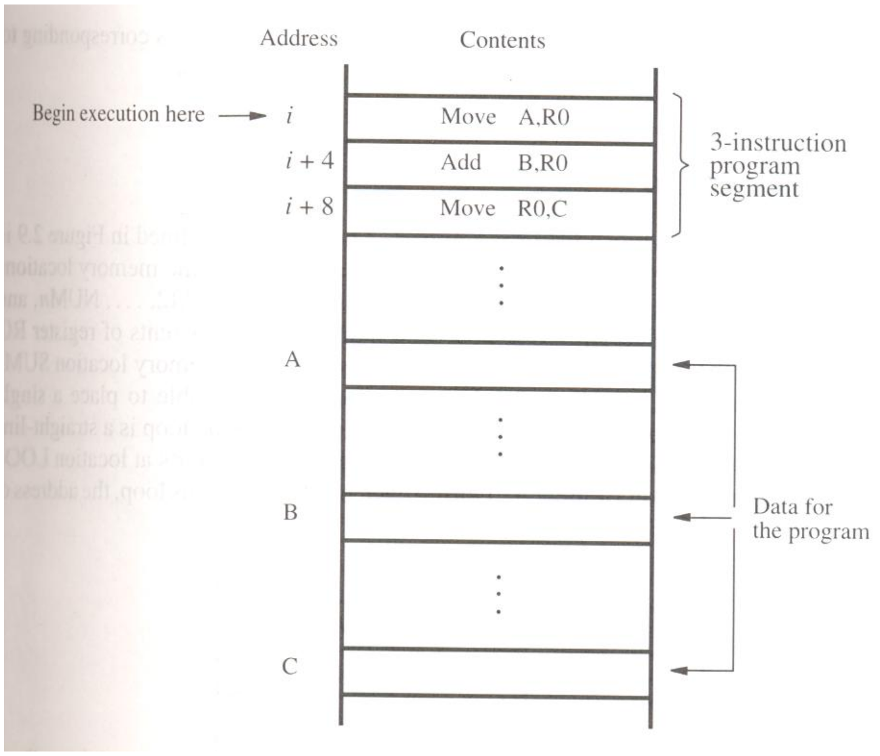
Instruction Execution:

Instruction in IR is examined to determine whose operation is to be performed.

Program execution Steps:

- To begin executing a program, the address of first instruction must be placed in PC.
- The processor control circuits use the information in the PC to fetch & execute instructions one at a time in the order of increasing order.
- This is called Straight line sequencing.During the execution of each instruction,the PC is incremented by 4 to point the address of next instruction.

Fig: Program Execution

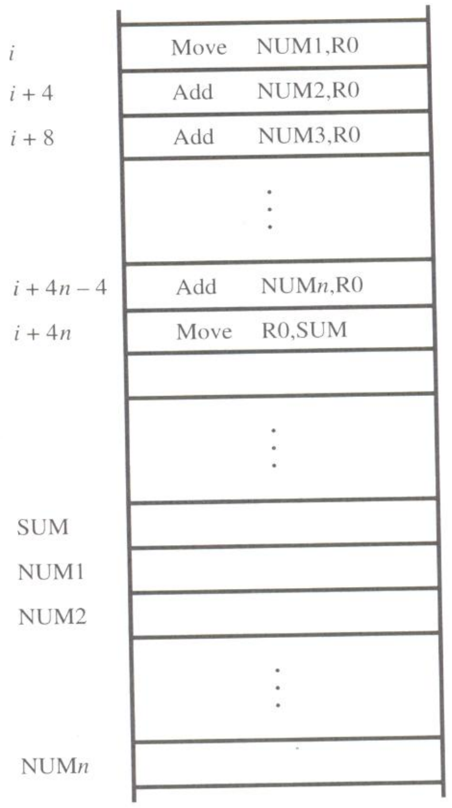


Branching:

- The Address of the memory locations containing the n numbers are symbolically given as NUM1,NUM2…..NUMn.

- Separate Add instruction is used to add each number to the contents of register R0.
- After all the numbers have been added,the result is placed in memory location SUM.

Fig:Straight Line Sequencing Program for adding ‘n’ numbers



Using loop to add ‘n’ numbers:

- Number of enteries in the list ‘n’ is stored in memory location M.Register R1 is used as a counter to determine the number of times the loop is executed.
- Content location M are loaded into register R1 at the beginning of the program.
- It starts at location Loop and ends at the instruction.Branch>0.During each pass,the address of the next list entry is determined and the entry is fetched and added to R0.

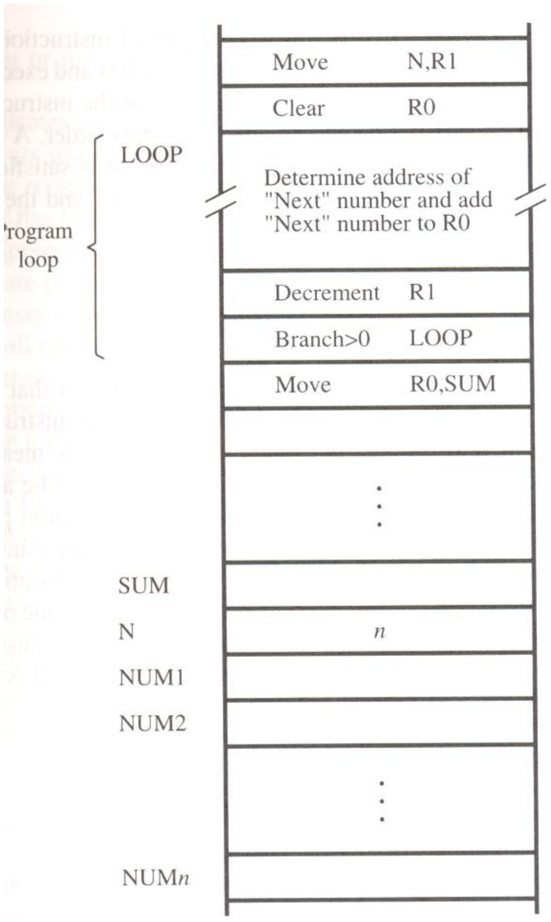
Decrement R1

- It reduces the contents of R1 by 1 each time through the loop.

Branch >0 Loop

- A conditional branch instruction causes a branch only if a specified condition is satisfied.

Fig:Using loop to add 'n' numbers:



Conditional Codes:

Result of various operation for user by subsequent conditional branch instruction is accomplished by recording the required information in individual bits often called **Condition code Flags.**

Commonly used flags:

- **N(Negative)**→set to 1 if the result is –ve ,otherwise cleared to 0.
- **Z(Zero)**→ set to 1 if the result is 0 ,otherwise cleared to 0.
- **V(Overflow)**→ set to 1 if arithmetic overflow occurs ,otherwise cleared to 0.
- **C(Carry)**set to 1 if carry and results from the operation ,otherwise cleared to 0.

ADDRESSING MODES

The different ways in which the location of an operand is specified in an instruction is called as Addressing mode.

Generic Addressing Modes:

- Immediate mode
- Register mode
- Absolute mode
- Indirect mode
- Index mode
- Base with index
- Base with index and offset
- Relative mode
- Auto-increment mode
- Auto-decrement mode

Implementation of Variables and Constants:

Variables:

The value can be changed as needed using the appropriate instructions.
There are 2 accessing modes to access the variables. They are

- Register Mode
- Absolute Mode

Register Mode:

The operand is the contents of the processor register.
The name(address) of the register is given in the instruction.

Absolute Mode(Direct Mode):

- The operand is in new location.
- The address of this location is given explicitly in the instruction.

Eg: MOVE LOC,R2

The above instruction uses the register and absolute mode.
The processor register is the temporary storage where the data in the register are accessed using register mode.
The absolute mode can represent global variables in the program.

Mode	**Assembler Syntax**	**Addressing Function**
Register mode	Ri	EA=Ri
Absolute mode	LOC	EA=LOC

Where **EA**-Effective Address

Constants:

Address and data constants can be represented in assembly language using Immediate Mode.

<u>Immediate Mode.</u>

The operand is given explicitly in the instruction.

Eg: Move 200 immediate ,R0

It places the value 200 in the register R0.The immediate mode used to specify the value of source operand.

In assembly language, the immediate subscript is not appropriate so # symbol is used.
It can be re-written as

Move #200,R0

Assembly Syntax:	**Addressing Function**
Immediate #value	Operand =value

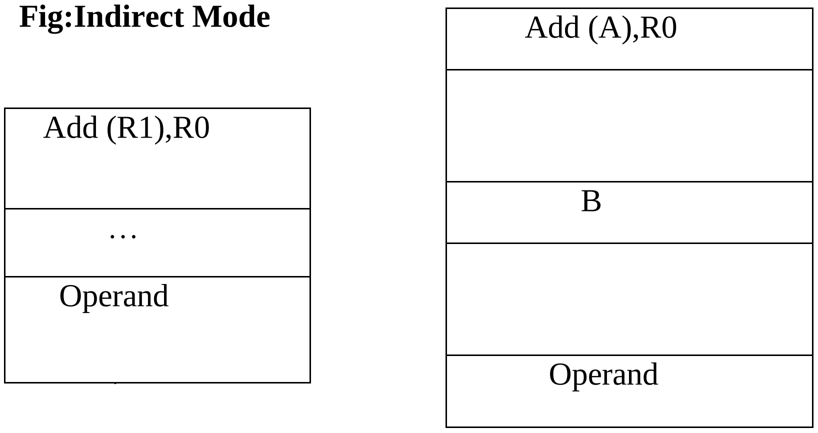
<u>Indirection and Pointers:</u>

Instruction does not give the operand or its address explicitly.Instead it provides information from which the new address of the operand can be determined.This address is called effective Address(EA) of the operand.

Indirect Mode:

- The effective address of the operand is the contents of a register .
- We denote the indirection by the name of the register or new address given in the instruction.

Fig:Indirect Mode



Address of an operand(B) is stored into R1 register.If we want this operand,we can get it through register R1(indirection).

The register or new location that contains the address of an operand is called the **pointer.**

Mode	**Assembler Syntax**	**Addressing Function**
Indirect	Ri , LOC	EA=[Ri] or EA=[LOC]

<u>Indexing and Arrays:</u>

Index Mode:

- The effective address of an operand is generated by adding a constant value to the contents of a register.
- The constant value uses either special purpose or general purpose register.
- We indicate the index mode symbolically as,

$$\mathbf{X(R_i)}$$

Where **X** – denotes the constant value contained in the instruction
R_i – It is the name of the register involved.

The Effective Address of the operand is,

$$\mathbf{EA=X + [Ri]}$$

The index register R1 contains the address of a new location and the value of X defines an offset(also called a displacement).

To find operand,

- First go to Reg R1 (using address)-read the content from R1-1000

- Add the content 1000 with offset 20 get the result.
- 1000+20=1020

- Here the constant X refers to the new address and the contents of index register define the offset to the operand.
- The sum of two values is given explicitly in the instruction and the other is stored in register.

Eg: Add 20(R1) , R2 (or) EA=>1000+20=1020

Index Mode	**Assembler Syntax**	**Addressing Function**
Index	X(Ri)	EA=[Ri]+X
Base with Index	(Ri,Rj)	EA=[Ri]+[Rj]
Base with Index and offset	X(Ri,Rj)	EA=[Ri]+[Rj] +X

Relative Addressing:

It is same as index mode. The difference is, instead of general purpose register, here we can use program counter(PC).

Relative Mode:

- The Effective Address is determined by the Index mode using the PC in place of the general purpose register (gpr).
- This mode can be used to access the data operand. But its most common use is to specify the target address in branch instruction.Eg. Branch>0 Loop
- It causes the program execution to goto the branch target location. It is identified by the name loop if the branch condition is satisfied.

Mode	**Assembler Syntax**	**Addressing Function**
Relative	X(PC)	EA=[PC]+X

Additional Modes:

There are two additional modes. They are

- Auto-increment mode
- Auto-decrement mode

Auto-increment mode:

- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically incremented to point to the next item in the list.

Mode	**Assembler syntax**	**Addressing Function**
Auto-increment	(Ri)+	EA=[Ri]; Increment Ri

Auto-decrement mode:

- The Effective Address of the operand is the contents of a register in the instruction.
- After accessing the operand, the contents of this register is automatically decremented to point to the next item in the list.

Mode	**Assembler Syntax**	**Addressing Function**
Auto-decrement	-(Ri)	EA=[Ri]; Decrement Ri

<u>CISC</u>

Pronounced *sisk*, and stands for **C**omplex **I**nstruction **S**et **C**omputer. Most PC's use CPU based on this architecture. For instance Intel and AMD CPU's are based on CISC architectures.
Typically CISC chips have a large amount of different and complex instructions. The philosophy behind it is that hardware is always faster than software, therefore one should make a powerful instruction set, which provides programmers with assembly instructions to do a lot with short programs.
In common CISC chips are relatively slow (compared to RISC chips) per instruction, but use little (less than RISC) instructions.

<u>RISC</u>

Pronounced *risk*, and stands for **R**educed **I**nstruction **S**et **C**omputer. RISC chips evolved around the mid-1980 as a reaction at CISC chips. The philosophy behind it is that almost no one uses complex assembly language instructions as used by CISC, and people mostly use compilers which never use complex instructions. Apple for instance uses RISC chips. Therefore fewer, simpler and faster instructions would be better, than the large, complex and slower CISC instructions. However, more instructions are needed to accomplish a task.
An other advantage of RISC is that - in theory - because of the more simple instructions,

RISC chips require fewer transistors, which makes them easier to design and cheaper to produce.
Finally, it's easier to write powerful optimised compilers, since fewer instructions exist.

RISC vs CISC

There is still considerable controversy among experts about which architecture is better. Some say that RISC is cheaper and faster and therefor the architecture of the future. Others note that by making the hardware simpler, RISC puts a greater burden on the software. Software needs to become more complex. Software developers need to write more lines for the same tasks.
Therefore they argue that RISC is not the architecture of the future, since conventional CISC chips are becoming faster and cheaper anyway.
RISC has now existed more than 10 years and hasn't been able to kick CISC out of the market. If we forget about the embedded market and mainly look at the market for PC's, workstations and servers I guess a least 75% of the processors are based on the CISC architecture. Most of them the x86 standard (Intel, AMD, etc.), but even in the mainframe territory CISC is dominant via the IBM/390 chip. Looks like CISC is here to stay …
Is RISC than really not better? The answer isn't quite that simple. RISC and CISC architectures are becoming more and more alike. Many of today's RISC chips support just as many instructions as yesterday's CISC chips. The PowerPC 601, for example, supports *more* instructions than the Pentium. Yet the 601 is considered a RISC chip, while the Pentium is definitely CISC. Further more today's CISC chips use many techniques formerly associated with RISC chips.

ALU Design

In computing an **arithmetic logic unit** (**ALU**) is a digital circuit that performs arithmetic and logical operations. The ALU is a fundamental building block of the central processing unit (CPU) of a computer, and even the simplest microprocessors contain one for purposes such as maintaining timers. The processors found inside modern CPUs and graphics processing units (GPUs) accommodate very powerful and very complex ALUs; a single component may contain a number of ALUs.

Mathematician John von Neumann proposed the ALU concept in 1945, when he wrote a report on the foundations for a new computer called the EDVAC. Research into ALUs remains an important part of computer science, falling under **Arithmetic and logic structures** in the ACM Computing Classification System

Numerical systems

An ALU must process numbers using the same format as the rest of the digital circuit. The format of modern processors is almost always the two's complement binary number representation. Early computers used a wide variety of number systems, including ones' complement, Two's complement sign-magnitude format, and even true decimal systems, with ten tubes per digit.

ALUs for each one of these numeric systems had different designs, and that influenced the current preference for two's complement, as this is the representation that makes it easier for the ALUs to calculate additions and subtractions.

The ones' complement and Two's complement number systems allow for subtraction to be accomplished by adding the negative of a number in a very simple way which negates the need for specialized circuits to do subtraction; however, calculating the negative in Two's complement requires adding a one to the low order bit and propagating the carry. An alternative way to do Two's complement subtraction of A-B is present a 1 to the carry input of the adder and use ~B rather than B as the second input.

Practical overview

Most of a processor's operations are performed by one or more ALUs. An ALU loads data from input registers, an external Control Unit then tells the ALU what operation to perform on that data, and then the ALU stores its result into an output register. Other mechanisms move data between these registers and memory.

Simple operations

A simple example arithmetic logic unit (2-bit ALU) that does AND, OR, XOR, and addition

Most ALUs can perform the following operations:

- Integer arithmetic operations (addition, subtraction, and sometimes multiplication and division, though this is more expensive)
- Bitwise logic operations (AND, NOT, OR, XOR)
- Bit-shifting operations (shifting or rotating a word by a specified number of bits to the left or right, with or without sign extension). Shifts can be interpreted as multiplications by 2 and divisions by 2.

Complex operations

Engineers can design an Arithmetic Logic Unit to calculate any operation. The more complex the operation, the more expensive the ALU is, the more space it uses in the processor, the more power it dissipates. Therefore, engineers compromise. They make the ALU powerful enough to make the processor fast, but yet not so complex as to become prohibitive. For example, computing the square root of a number might use:

1. **Calculation in a single clock** Design an extraordinarily complex ALU that calculates the square root of any number in a single step.
2. **Calculation pipeline** Design a very complex ALU that calculates the square root of any number in several steps. The intermediate results go through a series of circuits arranged like a factory production line. The ALU can accept new numbers to calculate even before having finished the previous ones. The ALU can now

produce numbers as fast as a single-clock ALU, although the results start to flow out of the ALU only after an initial delay.

3. **interactive calculation** Design a complex ALU that calculates the square root through several steps. This usually relies on control from a complex control unit with built-in microcode
4. **Co-processor** Design a simple ALU in the processor, and sell a separate specialized and costly processor that the customer can install just beside this one, and implements one of the options above.
5. **Software libraries** Tell the programmers that there is no co-processor and there is no emulation, so they will have to write their own algorithms to calculate square roots by software.
6. **Software emulation** Emulate the existence of the co-processor, that is, whenever a program attempts to perform the square root calculation, make the processor check if there is a co-processor present and use it if there is one; if there isn't one, interrupt the processing of the program and invoke the operating system to perform the square root calculation through some software algorithm.

The options above go from the fastest and most expensive one to the slowest and least expensive one. Therefore, while even the simplest computer can calculate the most complicated formula, the simplest computers will usually take a long time doing that because of the several steps for calculating the formula.

Powerful processors like the Intel Core and AMD64 implement option #1 for several simple operations, #2 for the most common complex operations and #3 for the extremely complex operations.

Inputs and outputs

The inputs to the ALU are the data to be operated on (called operands) and a code from the control unit indicating which operation to perform. Its output is the result of the computation.

In many designs the ALU also takes or generates as inputs or outputs a set of condition codes from or to a status register. These codes are used to indicate cases such as carry-in or carry-out, overflow, divide-by-zero, etc.

ALUs vs. FPUs

A Floating Point Unit also performs arithmetic operations between two values, but they do so for numbers in floating point representation, which is much more complicated than the two's complement representation used in a typical ALU. In order to do these calculations, a FPU has several complex circuits built-in, including some internal ALUs.

In modern practice, engineers typically refer to the ALU as the circuit that performs integer arithmetic operations (like two's complement and BCD). Circuits that calculate more complex formats like floating point, complex numbers, etc. usually receive a more specific name such as FPU.

FIXED POINT NUMBER AND OPERATION

In computing, a **fixed-point number** representation is a real data type for a number that has a fixed number of digits after (and sometimes also before) the radix point (*e.g.*, after the decimal point '.' in English decimal notation). Fixed-point number representation can be compared to the more complicated (and more computationally demanding) floating point number representation.

Fixed-point numbers are useful for representing fractional values, usually in base 2 or base 10, when the executing processor has no floating point unit (FPU) or if fixed-point provides improved performance or accuracy for the application at hand. Most low-cost embedded microprocessors and microcontrollers do not have an FPU.

Representation

A value of a fixed-point data type is essentially an integer that is scaled by a specific factor determined by the type. For example, the value 1.23 can be represented as 1230 in a fixed-point data type with scaling factor of 1/1000, and the value 1230000 can be represented as 1230 with a scaling factor of 1000. Unlike floating-point data types, the scaling factor is the same for all values of the same type, and does not change during the entire computation.

The scaling factor is usually a power of 10 (for human convenience) or a power of 2 (for computational efficiency). However, other scaling factors may be used occasionally, e.g. a time value in hours may be represented as a fixed-point type with a scale factor of 1/3600 to obtain values with one-second accuracy.

The maximum value of a fixed-point type is simply the largest value that can be represented in the underlying integer type, multiplied by the scaling factor; and similarly for the minimum value. For example, consider a fixed-point type represented as a binary integer with b bits in two's complement format, with a scaling factor of $1/2^f$ (that is, the last f bits are fraction bits): the minimum representable value is $-2^{b-1}/2^f$ and the maximum value is $(2^{b-1}-1)/2^f$.

Operations

To convert a number from a fixed point type with scaling factor R to another type with scaling factor S, the underlying integer must be multiplied by R and divided by S; that is, multiplied by the ratio R/S. Thus, for example, to convert the value 1.23 = 123/100 from a type with scaling factor R=1/100 to one with scaling factor S=1/1000, the underlying integer 123 must be multiplied by (1/100)/(1/1000) = 10, yielding the representation 1230/1000. If S does not divide R (in particular, if the new scaling factor R is less than the original *S*), the new integer will have to be rounded. The rounding rules and methods are usually part of the language's specification.

To add or subtract two values the same fixed-point type, it is sufficient to add or subtract the underlying integers, and keep their common scaling factor. The result can be exactly represented in the same type, as long as no overflow occurs (i.e. provided that the sum of the two integers fits in the underlying integer type.) If the numbers have different fixed-point types, with different scaling factors, then one of them must be converted to the other before the sum.

To multiply two fixed-point numbers, it suffices to multiply the two underlying integers, and assume that the scaling factor of the result is the product of their scaling factors. This operation involves no rounding. For example, multiplying the numbers 123 scaled by 1/1000 (0.123) and 25 scaled by 1/10 (2.5) yields the integer 123×25 = 3075 scaled by (1/1000)×(1/10) = 1/10000, that is 3075/10000 = 0.3075. If the two operands belong to the same fixed-point type, and the result is also to be represented in that type, then the product of the two integers must be explicitly multiplied by the common scaling factor; in this case the result may have to be rounded, and overflow may occur. For example, if the common scaling factor is 1/100, multiplying 1.23 by 0.25 entails multiplying 123 by 25 to yield 3075 with an intermediate scaling factor if 1/10000. This then must be multiplied by 1/100 to yield either 31 (0.31) or 30 (0.30), depending on the rounding method used, to result in a final scale factor of 1/100.

To divide two fixed-point numbers, one takes the integer quotient of their underlying integers, and assumes that the scaling factor is the quotient of their scaling factors. The first division involves rounding in general. For example, division of 3456 scaled by 1/100 (34.56) by 1234 scaled by 1/1000 (1.234) yields the integer 3456÷1234 = 3 (rounded) with scale factor (1/100)/(1/1000) = 10, that is, 30. One can obtain a more accurate result by first converting the dividend to a more precise type: in the same example, converting 3456 scaled by 1/100 (34.56) to 3456000 scaled by 1/100000, before dividing by 1234 scaled by 1/1000 (1.234), would yield 3456000÷1234 = 2801 (rounded) with scaling factor (1/100000)/(1/1000) = 1/100, that is 28.01 (instead of 290). If both operands and the desired result are represented in the same fixed-point type, then the quotient of the two integers must be explicitly divided by the common scaling factor.

Precision loss and overflow

Because fixed point operations can produce results that have more bits than the operands there is possibility for information loss. For instance, the result of fixed point multiplication could potentially have as many bits as the sum of the number of bits in the two operands. In order to fit the result into the same number of bits as the operands, the answer must be rounded or truncated. If this is the case, the choice of which bits to keep is very important. When multiplying two fixed point numbers with the same format, for instance with I integer bits, and Q fractional bits, the answer could have up to $2I$ integer bits, and $2Q$ fractional bits.

For simplicity, fixed-point multiply procedures use the same result format as the operands. This has the effect of keeping the middle bits; the *I*-number of least significant integer bits, and the *Q*-number of most significant fractional bits. Fractional bits lost below this value represent a precision loss which is common in fractional multiplication. If any integer bits are lost, however, the value will be radically inaccurate.

Some operations, like divide, often have built-in result limiting so that any positive overflow results in the largest possible number that can be represented by the current format. Likewise, negative overflow results in the largest negative number represented by the current format. This built in limiting is often referred to as *saturation*.

Some processors support a hardware overflow flag that can generate an exception on the occurrence of an overflow, but it is usually too late to salvage the proper result at this point.

<u>FLOATING POINT NUMBERS & OPERATIONS</u>

Floating point Representation:

To represent the fractional binary numbers, it is necessary to consider binary point.If binary point is assumed to the right of the sign bit ,we can represent the fractional binary numbers as given below,

$$\boxed{B= (b_0 * 2^0 + b_{-1} * 2^{-1} + b_{-2} * 2^{-2} + \ldots + b_{-(n-1)} * 2^{-(n-1)})}$$

With this fractional number system,we can represent the fractional numbers in the following range,

$$-1 < F < 1 - 2^{-(n-1)}$$

- The binary point is said to be float and the numbers are called floating point numbers.
- The position of binary point in floating point numbers is variable and hence numbers must be represented in the specific manner is referred to as floating point representation.
- The floating point representation has three fields.They are,
  - Sign
  - Significant digits and
  - Exponent.

Eg: 111101.1000110$\rightarrow$1.111101100110 $*2^5$

Where ,

2^5 $\rightarrow$Exponent and scaling factor

UNIT I

PART A

1.Difference between Computer architecture and computer organization.
2.Discusss the various functions of computer.
3.List out the various instruction type.
4.What is addressing mode.
5.Difference between Stack and Queue.
6.What are the different types of Assembler directive?
7.A computer executes 3 instructions with the length of 6cycles/sec which performs its instruction in 4 steps.Calculate the performance.
8.What is pipelining?
9.What is program Counter?
10.What is bus and explain its types.
11. Give an example each of zero-address, one-address, two-address, and three-address instructions.
12. Which data structures can be best supported using (a) indirect addressing mode (b) indexed addressing mode?

PART B

1.Explain the functional Units of a Computer.
2.What is Addressing Mode?Explain the different types of addressing mode.
3.What is Instruction Type?Write short notes on various instruction formats.
4.What is byte addressability?Explain the various types.
5.Write short notes on:

a. IR	b..MAR
c.MDR	d.PC
e.Interrupt	f.Pipelining and Superscalar operation

6. Explain in detail the different types of instructions that are supported in a typical processor.

UNIT II

BASIC PROCESSING UNIT

- Fundamental concepts
- Execution of a complete instruction
- Multiple bus organization
- Hardwired control
- Micro programmed control
- Nano programming.

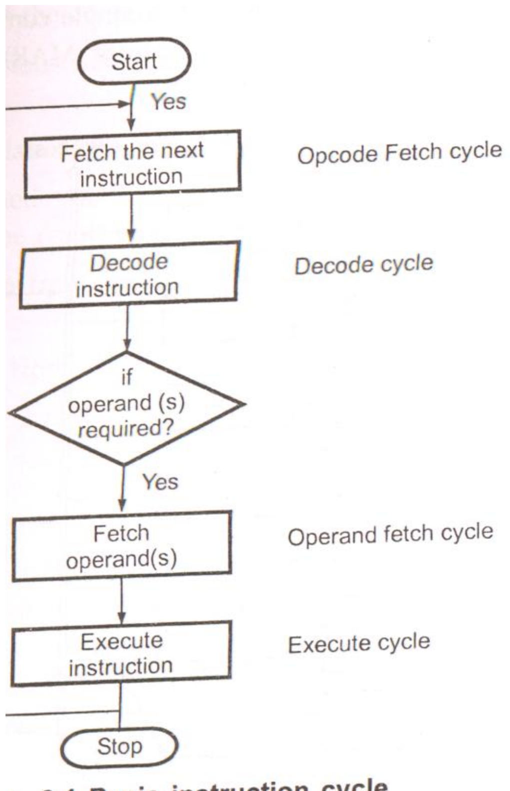
Basic fundamental concepts

Some fundamental concepts

- The primary function of a processor unit is to execute sequence of instructions stored in a memory, which is external to the processor unit.
- The sequence of operations involved in processing an instruction constitutes an instruction cycle,which can be subdivided into 3 major phases:

  1. Fetch cycle
  2. Decode cycle
  3. Execute cycle

Basic instruction cycle



g. 3.1 Basic instruction cycle

- To perform fetch, decode and execute cycles the processor unit has to perform set of operations called micro-operations.
- Single bus organization of processor unit shows how the building blocks of processor unit are organised and how they are interconnected.

- They can be organised in a variety of ways, in which the arithmetic and logic unit and all processor registers are connected through a single common bus.
- It also shows the external memory bus connected to memory address(MAR) and data register(MDR).

Single Bus Organisation of processor

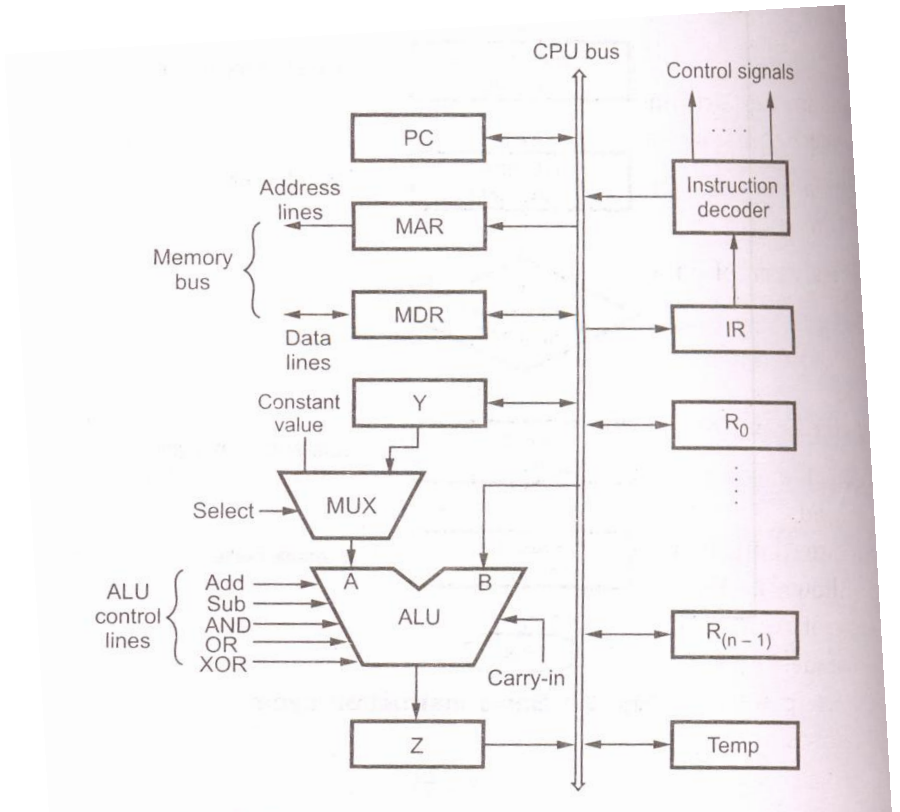


Fig. 3.2 Single bus organisation of processor

- The registers Y,Z and Temp are used only by the processor unit for temporary storage during the execution of some instructions.
- These registers are never used for storing data generated by one instruction for later use by another instruction.
- The programmer cannot access these registers.
- The IR and the instruction decoder are integral parts of the control circuitry in the processing unit.
- All other registers and the ALU are used for storing and manipulating data.
- The data registers, ALU and the interconnecting bus is referred to as data path.
- Register R_0 through $R_{(n-1)}$ are the processor registers.
- The number and use of these register vary considerably from processor to processor.

- These registers include general purpose registers and special purpose registers such as stack pointer, index registers and pointers.
- These are 2 options provided for A input of the ALU.
- The multiplexer(MUX) is used to select one of the two inputs.
- It selects either output of Y register or a constant number as an A input for the ALU according to the status of the select input.
- It selects output of Y when select input is 1 (select Y) and it selects a constant number when select input is 0(select C) as an input A for the multiplier.
- The constant number is used to increment the contents of program counter.
- For the execution of various instructions processor has to perform one or more of the following basic operations:

  a) Transfer a word of data from one processor register to the another or to the ALU.
  b) perform the arithmetic or logic operations on the data from the processor registers and store the result in a processor register.
  c) Fetch a word of data from specified memory location and load them into a processor register.
  d) Store a word of data from a processor register into a specified memory location.

1. Register Transfers

Each register has input and output gating and these gates are controlled by corresponding control signals.

Fig: Input and Output Gating for the Registers

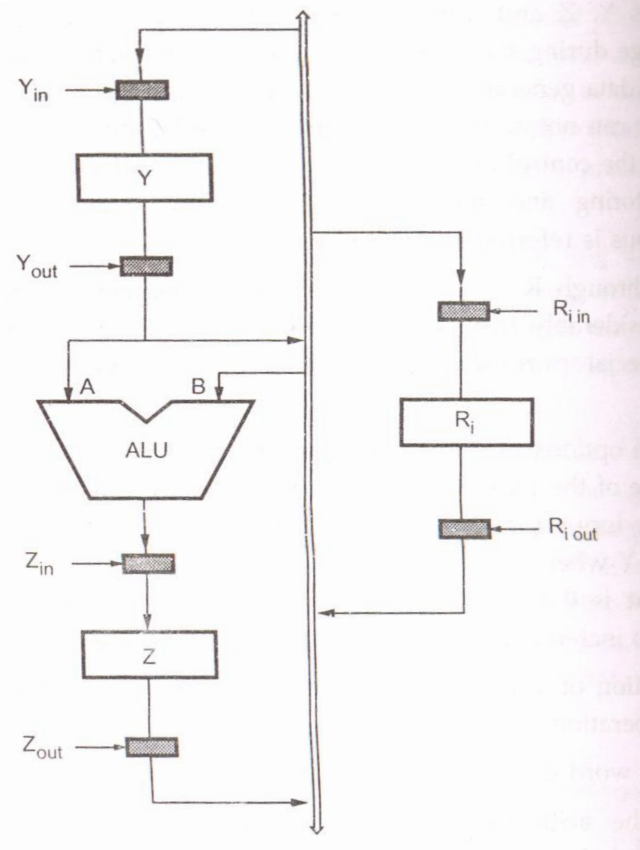
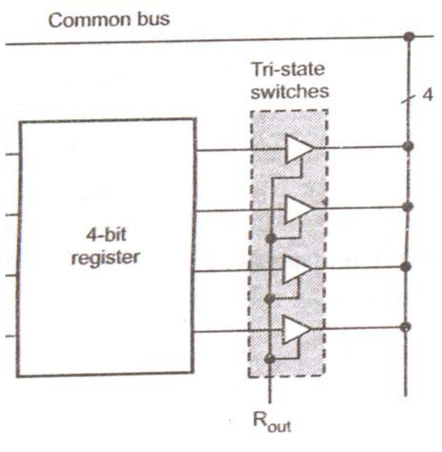


Fig. 3.3 Input and output gating for the register

- The input and output gates are nothing but the electronic switches which can be controlled by the control signals.
- When signal is 1, the switch is ON and when the signal is 0, the switch is OFF.

Implementation of input and output gates of a 4 bit register



Consider that we have transfer data from register R_1 to R_2

It can be done by,

a. Activate the output enable signal of R_1,R_1 out=1. It places the contents of R_1 on the common bus.
b. Activate the input enable signal of R_2, R_2 in=1. It loads data from the common bus into the register R_2.

One-bit register

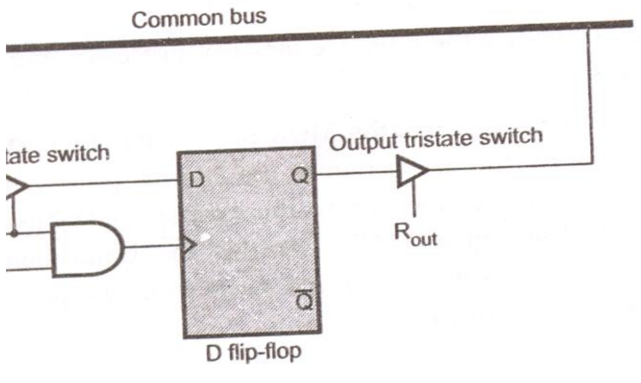


Fig. 3.5 One-bit register

- The edge triggered D flip-flop which stores the one-bit data is connected to the common bus through tri-state switches.
- Input D is connected through input tri-state switch and output Q is connected through output tri-state switch.
- The control signal R_{in} enables the input tri-state switch and the data from common bus is loaded into the D flip-flop in synchronisation with clock input when R_{in} is active.
- It is implemented using AND gate .
- The control signal R_{out} is activated to load data from Q output of the D flip-flop on to the common bus by enabling the output tri-state switch.

2. Performing an arithmetic or logic operation

- ALU performs arithmetic and logic operations.
- It is a combinational circuit that has no internal memory.
- It has 2 inputs A and B and one output.
- It's A input gets the operand from the output of the multiplexer and its B input gets the operand directly from the bus.
- The result produced by the ALU is stored temporarily in register Z.

Let us find the sequence of operations required to subtract the contents of register R_2 from register R_1 and store the result in register $R_{3.}$

This sequence can be followed as:

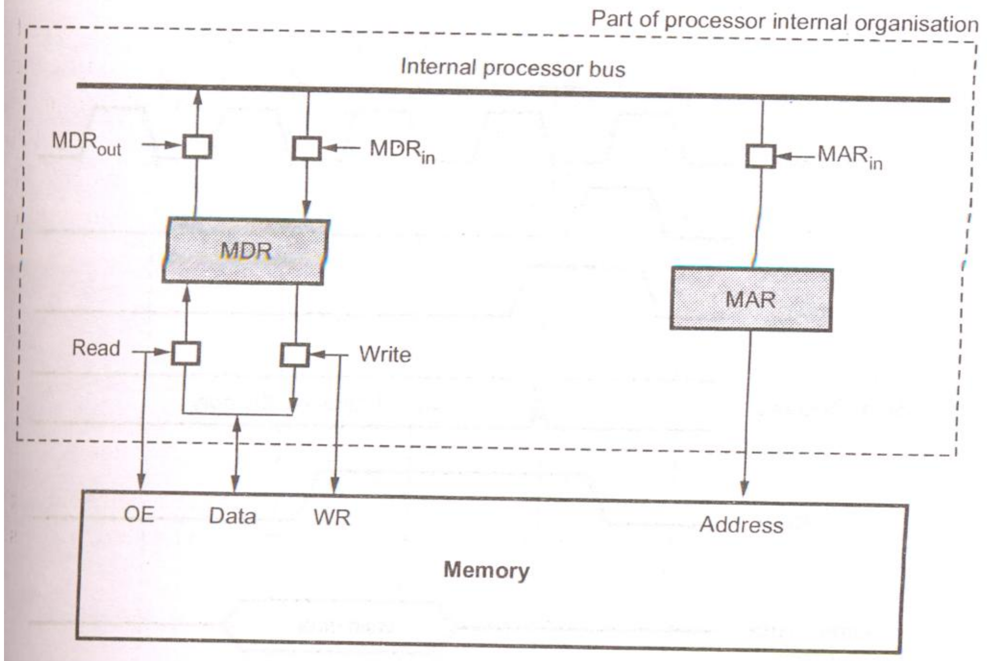
a) R_1,Y_{in}
b) R_{2out},Select Y, sub,Z_{in}
c) Z_{out},R_{3in}

Step 1: contents from register R_1 are loaded into register Y.
Step 2: contents from Y and from register R_2 are applied to the A and B inputs of ALU, subtraction is performed and result is stored in the Z register.
Step 3: The contents of Z register is stored in the R_3 register.

3. Fetching a word from memory

- To fetch a word of data from memory the processor gives the address of the memory location where the data is stored on the address bus and activates the Read operation.
- The processor loads the required address in MAR, whose output is connected to the address lines of the memory bus.
- At the same time processor sends the Read signal of memory control bus to indicate the Read operation.
- When the requested data is received from the memory its stored into the MDR, from where it can be transferred to other processor registers.

3 - 7 Basic Processing Unit



3.6 Data, address and control signals for data transf

4. Storing a word in memory

- To write a word in memory location processor has to load the address of the desired memory location in the MAR, load the data to be written in memory, in MDR and activate write operation.
- Assume that we have to execute instruction Move(R_2), R_1.
- This instruction copies the contents of register R_1 into the memory whose location is specified by the contents of register R_2.

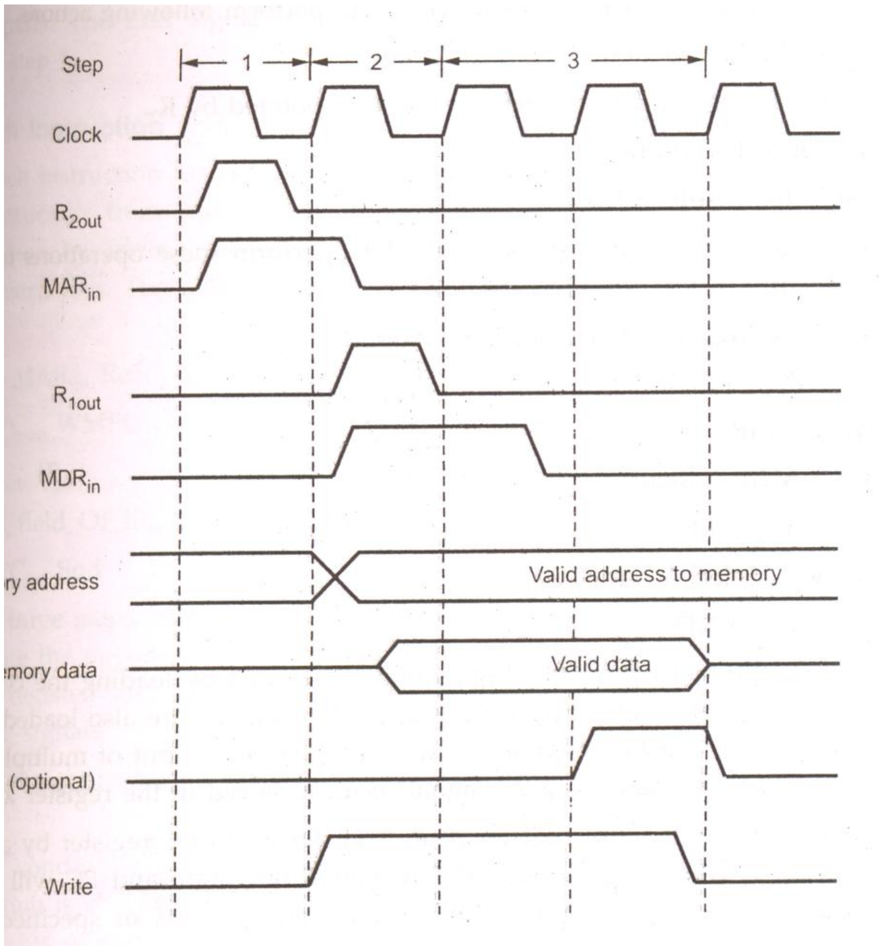
The actions needed to execute this instruction are as follows**:**

a) MAR$\leftarrow$ [R_2]
b) MDR$\leftarrow$ [R_1]
c) Activate the control signal to perform the write operation.

The various control signals which are necessary to activate to perform given actions in each step.

a) R_{2out}, MAR_{in}
b) R_{1out}, MDR_{inP}
c) MAR_{out}, MDR_{outM},Write

Timing diagram for MOVE(R_2), R_1 instruction (Memory write operation)



Timing diagram for MOVE (R_2), R_1 instruction (Memory write operation)

- The MDR register has 4 control signals:

  MDR_{inP} & MDR_{outP} control the connection to the internal processor data bus and signals MDR_{inM} & MDR_{outM} control the connection to the memory Data bus.
- MAR register has 2 control signals.

  Signal MAR_{in} controls the connection to the internal processor address bus and signal MAR_{out} controls the connection to the memory address bus.
- Control signals read and write from the processor controls the operation Read and Write respectively.

- The address of the memory word to be read word from that location to the register R_3,.
- It can be indicated by instruction MOVE R_3,(R_2).

The actions needed to execute this instruction are as follows:

a) MAR $\longleftarrow$ [R_2]
b) Activate the control signal to perform the Read operation
c) Load MDR from the memory bus
d) $R_3 \longleftarrow$ [MDR]

Various control signals which are necessary to activate to perform given actions in each step:

a) R_{2out}, MAR_{in}
b) MAR_{out}, MDR_{inM}, Read
c) MDR_{outP},R_{3in}

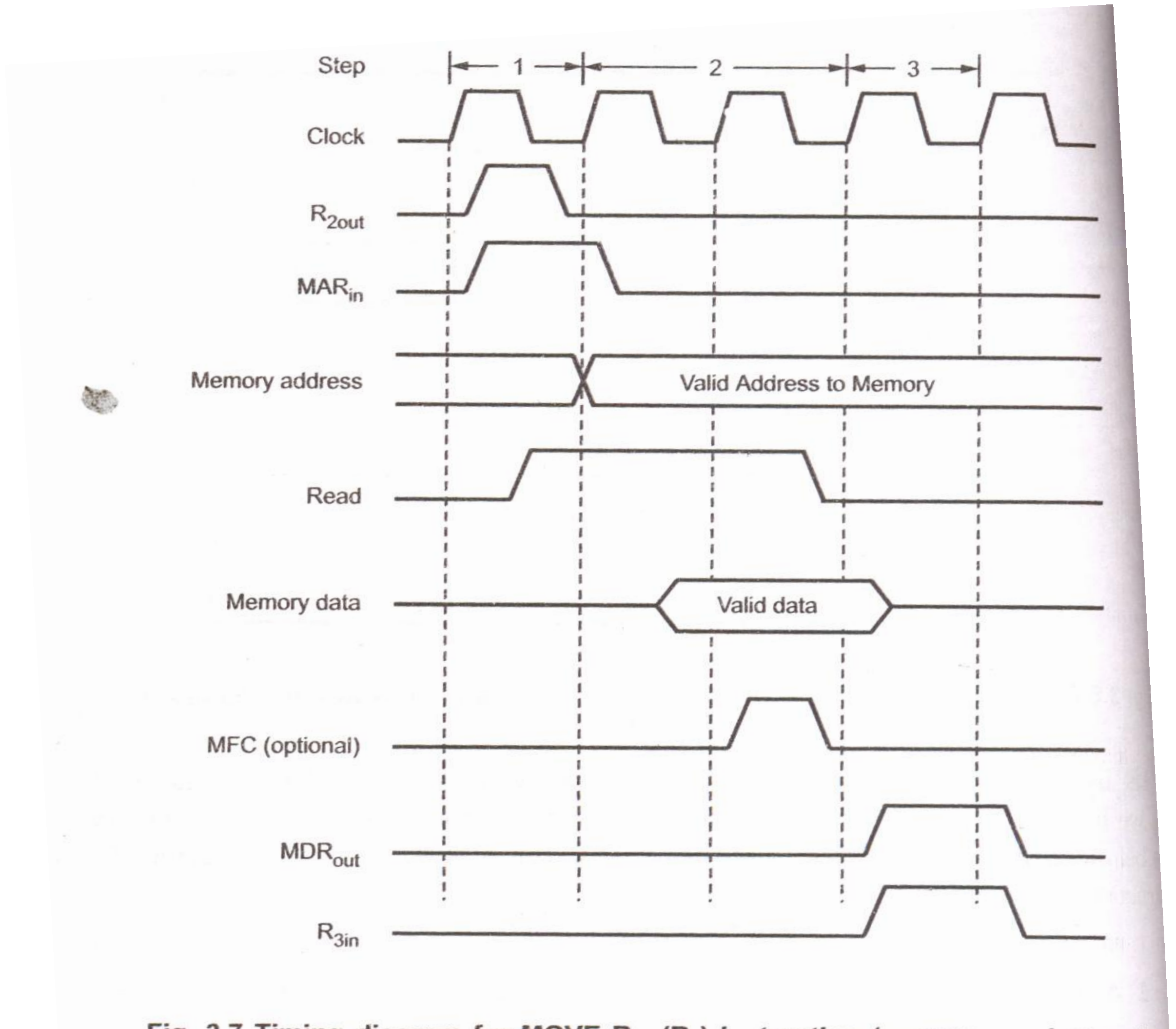


Fig. 3.7 Timing diagram for MOVE R_3, (R_2) instruction (memory read operation)

Note : 1. In above ti

EXECUTION OF A COMPLETE INSTRUCTION

Let us find the complete control sequence for execution of the instruction Add R_1,(R_2) for the single bus processor.

- This instruction adds the contents of register R_1 and the contents of memory location specified by register R_2 and stores results in the register R_1.
- To execute bus instruction it is necessary to perform following actions:
  1. Fetch the instruction
  2. Fetch the operand from memory location pointed by R_2.
  3. Perform the addition
  4. Store the results in R_1.

The sequence of control steps required to perform these operations for the single bus architecture are as follows;

1. PC_{out}, MAR_{in} Y_{in}, select C, Add, Z_{in}
2. Z_{out}, PC_{in}, MAR_{out} , $MAR_{inM,}$ Read
3. MDR_{out} P,MAR_{in}
4. R_{2out} , MAR_{in}
5. R_{2out} , Y_{in},MAR_{out} , $MAR_{inM,}$ Read
6. MDR_{out} P, select Y, Add, Z_{in}
7. Z_{out}, R_{1in}

(i) Step1, the instruction fetch operation is initiated by loading the controls of the PC into the MAR.

- PC contents are also loaded into register Y and added constant number by activating select C input of multiplexer and add input of the ALU.
- By activating Z_{in} signal result is stored in the register Z

(ii) Step2 , the contents of register Z are transferred to pc register by activating Z_{out} and pc_{in} signal.

- This completes the PC increment operation and PC will now point to next instruction,
- In the same step (step2), MARout , MDR $_{inM}$ and Read signals are activated.
- Due to MARout signal , memory gets the address and after receiving read signal and activation of MDR in M Signal ,it loads the contents of specified location into MDR register.

(iii) Step 3 contents of MDR register are transferred to the instruction register(IR) of the processor.

- The step 1 through 3 constitute the instruction fetch phase.
- At the beginning of step 4, the instruction decoder interprets the contents of the IR.
- This enables the control circuitry to activate the control signals for steps 4 through 7, which constitute the execution phase.

(iv) Step 4, the contents of register R_2 are transferred to register MAR by activating R_{2out} and MAR in signals.

(v) Step 5, the contents of register R_1 are transferred to register Y by activating R_{1out} and Y_{in} signals. In the same step, MAR_{out}, MDR_{inM} and Read signals are activated.

- Due to MAR_{out} signal, memory gets the address and after receiving read signal and activation of MDR_{inM} signal it loads the contents of specified location into MDR register.

(vi) Step 6 MDR_{outP}, select Y, Add and Z_{in} signals are activated to perform addition of contents of register Y and the contents of MDR. The result is stored in the register Z.
(vii) Step 7, the contents of register Z are transferred to register R_1 by activating Z_{out} and R_{1in} signals.

Branch Instruction

The branch instruction loads the branch target address in PC so that PC will fetch the next instruction from the branch target address.

The branch target address is usually obtained by adding the offset in the contents of PC. The offset is specified within the instruction.

The control sequence for unconditional branch instruction is as follows:

1. PC_{out}, MAR_{in}, Y_{in}, SelectC, Add, Z_{in}
2. Z_{out}, PC_{in}, MAR_{out}, MDR_{inM}, Read
3. MDR_{outP},IR_{in}
4. PC_{out},Y_{in}
5. Offset_field_Of_IR_{out},SelectY,Add,Z_{in}
6. Z_{out},PC_{in}

First 3 steps are same as in the previous example.
Step 4: The contents of PC are transferred to register Y by activating PC_{out} and Y_{in} signals.
Step 5: The contents of PC and the offset field of IR register are added and result is saved in register Z by activating corresponding signals.
Step 6: The contents of register Z are transferred to PC by activating Z_{out} and PC in signals.

Multiple Bus Organisation:

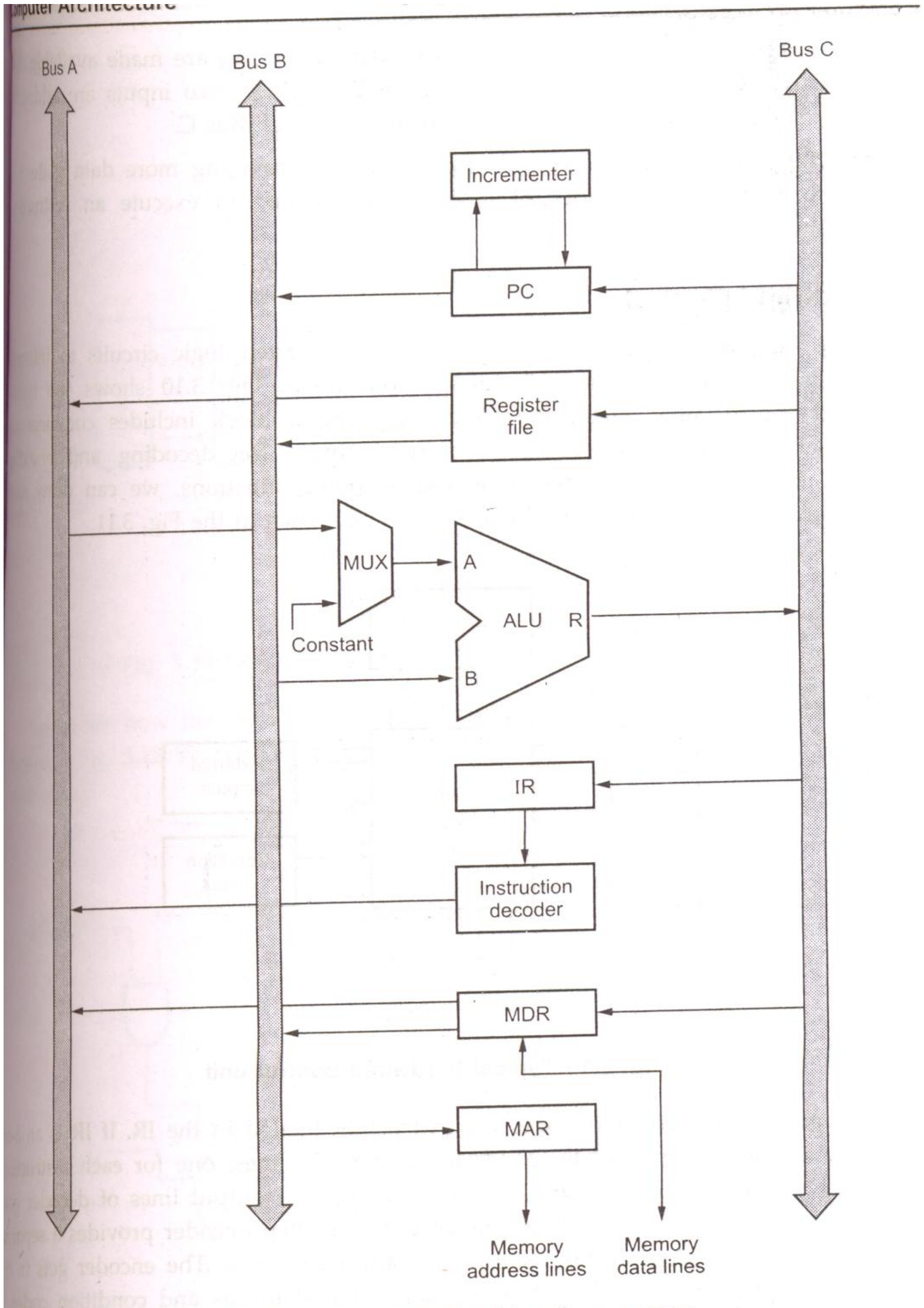


Fig. 3.9 Three-bus organisation of processor

- Single bus only one data word can be transferred over the bus in a clock cycle.
- This increases the steps required to complete the execution of the instruction.
- To reduce the number of steps needed to execute instructions, most commercial process provide multiple internal paths that enable several transfer to take place in parallel.
- 3 buses are used to connect registers and the ALU of the processor.
- All general purpose registers are shown by a single block called register file.
- There are 3 ports, one input port and two output ports.
- So it is possible to access data of three register in one clock cycle, the value can be loaded in one register from bus C and data from two register can be accessed to bus A and bus B.
- Buses A and B are used to transfer the source operands to the A and B inputs of the ALU.
- After performing arithmetic or logic operation result is transferred to the destination operand over bus C.
- To increment the contents of PC after execution of each instruction to fetch the next instruction, separate unit is provided. This unit is known as incrementer.
- Incrementer increments the contents of PC accordingly to the length of the instruction so that it can point to next instruction in the sequence.
- The incrementer eliminates the need of multiplexer connected at the A input of ALU.
- Let us consider the execution of instruction, Add,R_1,R_2,R_3.
- This instruction adds the contents of registers R_2 and the contents of register R_3 and stores the result in R_1.
- With 3 bus organization control steps required for execution of instruction Add R_1,R_2,R_3 are as follows:
    1. PC_{out}, MAR_{in}
    2. MAR_{out}, MDR_{inM}, Read
    3. MDR_{outP},IRin
    4. R_{2out},R_{3out},Add,R_{1in}

  Step 1: The contents of PC are transferred to MAR through bus B.

  Step 2: The instruction code from the addressed memory location is read into MDR.

  Step 3: The instruction code is transferred from MDR to IR register. At the beginning of step 4, the instruction decoder interprets the contents of the IR.

  This enables the control circuitry to activate the control signals for step 4, which constitute the execution phase.

  Step 4: two operands from register R_2 and register R_3 are made available at A and B inputs of ALU through bus A and bus B.

  These two inputs are added by activation of Add signal and result is stored in R_1 through bus C.

Hardwird Control

The control units use fixed logic circuits to interpret instructions and generate control signals from them.

The fixed logic circuit block includes combinational circuit that generates the required control outputs for decoding and encoding functions.

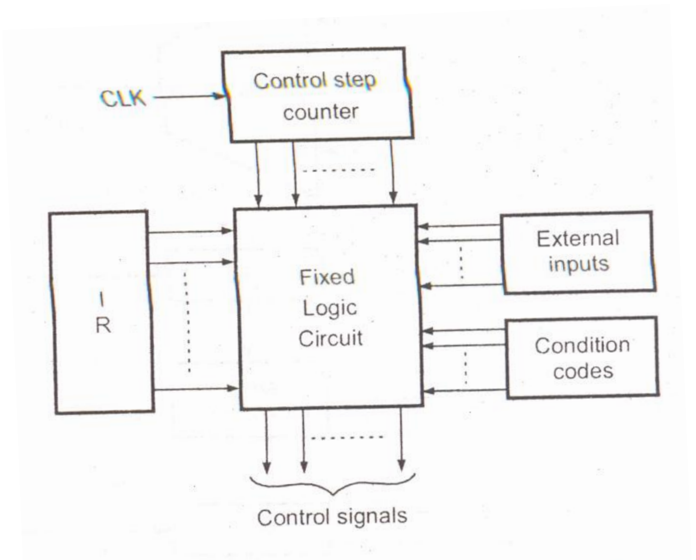


Fig. 3.10 Typical hardwired control unit

n decoder decodes the instruction loaded in the IF

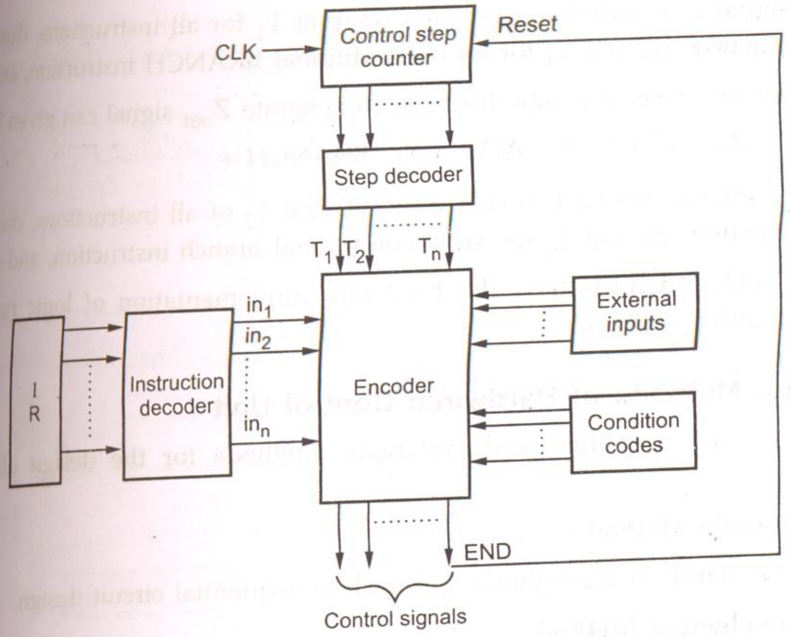
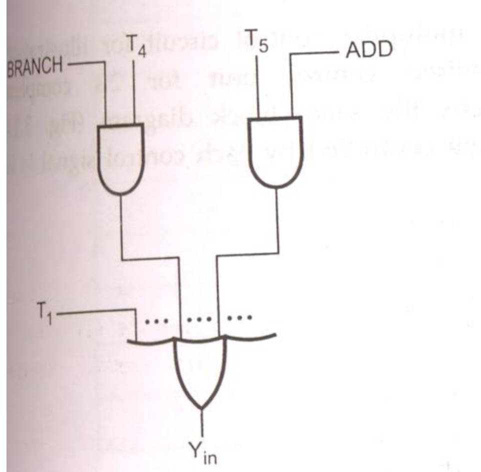


Fig. 3.11 Detail block diagram for hardwired control unit

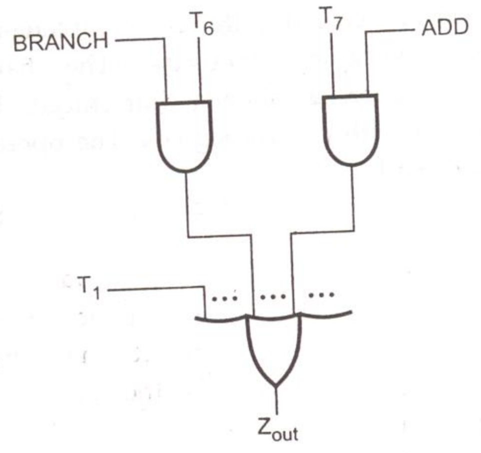
Let us see how the encoder generates signal for single bus processor organisation wn in Fig. 3.12 Y_{in}. The encoder circuit implements the following logic function to erate Y_{in}.

$$Y_{in} = T_1 + T_5 \cdot ADD + T_4 \cdot BRANCH + \ldots$$



Generation of the Y_{in} control signal

Fig. 3.12



Generation of the Z_{out} control signal

Fig. 3.13

Instruction decoder

It decodes the instruction loaded in the IR.

If IR is an 8 bit register then instruction decoder generates 2^8(256 lines); one for each instruction.

According to code in the IR, only one line amongst all output lines of decoder goes high (set to 1 and all other lines are set to 0).

Step decoder

It provides a separate signal line for each step, or time slot, in a control sequence.

Encoder

It gets in the input from instruction decoder, step decoder, external inputs and condition codes.

It uses all these inputs to generate the individual control signals.

After execution of each instruction end signal is generated this resets control step counter and make it ready for generation of control step for next instruction.

The encoder circuit implements the following logic function to generate Y_{in}

$\mathbf{Y_{in} = T_1 + T_5 . Add + T . BRANCH+\ldots}$

The Y_{in} signal is asserted during time interval T_1 for all instructions, during T_5 for an ADD instruction, during T_4 for an unconditional branch instruction, and so on.

As another example, the logic function to generate Z_{out} signal can given by

$\mathbf{Z_{out} = T_2 + T_7 . ADD + T_6 . BRANCH +\ldots.}$

The Z_{out} signal is asserted during time interval T_2 of all instructions, during T_7 for an ADD instruction, during T_6 for an unconditional branch instruction, and so on.

A Complete processor

It consists of

- Instruction unit
- Integer unit
- Floating-point unit
- Instruction cache
- Data cache
- Bus interface unit
- Main memory module
- Input/Output module.

Instruction unit- It fetches instructions from an instruction cache or from the main memory when the desired instructions are not available in the cache.

Integer unit – To process integer data

Floating unit – To process floating –point data

Data cache – The integer and floating unit gets data from data cache

The 80486 processor has 8-kbytes single cache for both instruction and data whereas the Pentium processor has two separate 8 kbytes caches for instruction and data.

The processor provides bus interface unit to control the interface of processor to system bus, main memory module and input/output module.

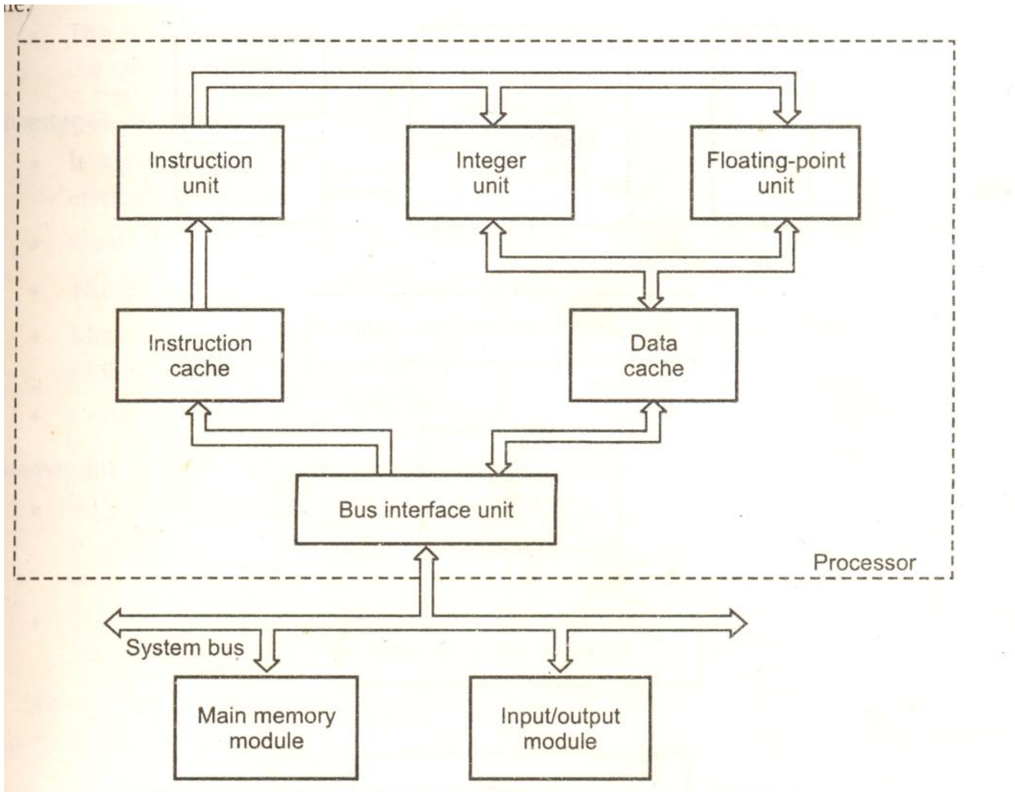


Fig. 3.14 Block diagram of a complete processor

Microprogrammed Control

Every instruction in a processor is implemented by a sequence of one or more sets of concurrent micro operations.

Each micro operation is associated with a specific set of control lines which, when activated, causes that micro operation to take place.

Since the number of instructions and control lines is often in the hundreds, the complexity of hardwired control unit is very high.

Thus, it is costly and difficult to design. The hardwired control unit is relatively inflexible because it is difficult to change the design, if one wishes to correct design error or modify the instruction set.

Microprogramming is a method of control unit design in which the control signal memory CM.

The control signals to be activated at any time are specified by a microinstruction, which is fetched from CM.

A sequence of one or more micro operations designed to control specific operation, such as addition, multiplication is called a micro program.

The micro programs for all instructions are stored in the control memory.

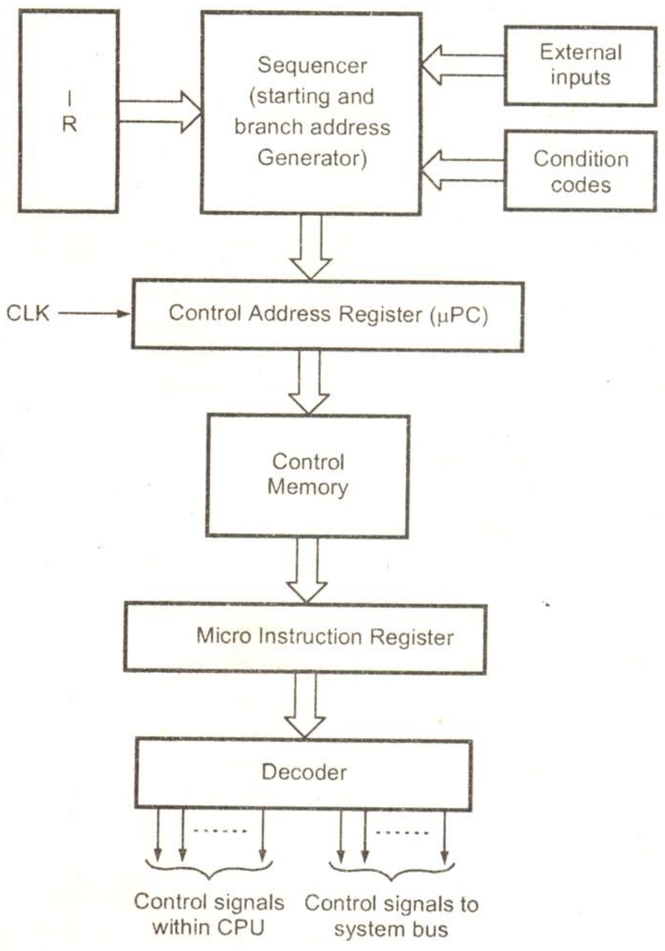

I R
Sequencer (starting and branch address Generator)
External inputs
Condition codes
CLK
Control Address Register (μPC)
Control Memory
Micro Instruction Register
Decoder
Control signals within CPU
Control signals to system bus

Fig. 3.15 Microprogrammed control unit

The address where these microinstructions are stored in CM is generated by microprogram sequencer/microprogram controller.

The microprogram sequencer generates the address for microinstruction according to the instruction stored in the IR.

The microprogrammed control unit,

- control memory
- control address register
- micro instruction register
- microprogram sequencer

The components of control unit work together as follows:

- ✓ The control address register holds the address of the next microinstruction to be read.
- ✓ When address is available in control address register, the sequencer issues READ command to the control memory.
- ✓ After issue of READ command, the word from the addressed location is read into the microinstruction register.
- ✓ Now the content of the micro instruction register generates control signals and next address information for the sequencer.
- ✓ The sequencer loads a new address into the control address register based on the next address information.

Advantages of Microprogrammed control

- ➢ It simplifies the design of control unit. Thus it is both, cheaper and less error phrone implement.
- ➢ Control functions are implemented in software rather than hardware.
- ➢ The design process is orderly and systematic
- ➢ More flexible, can be changed to accommodate new system specifications or to correct the design errors quickly and cheaply.
- ➢ Complex function such as floating point arithmetic can be realized efficiently.

Disadvantages

- ➢ A microprogrammed control unit is somewhat slower than the hardwired control unit, because time is required to access the microinstructions from CM.
- ➢ The flexibility is achieved at some extra hardware cost due to the control memory and its access circuitry.

Microinstruction

A simple way to structure microinstructions is to assign one bit position to each control signal required in the CPU.

Grouping of control signals

Grouping technique is used to reduce the number of bits in the microinstruction.
Gating signals: IN and OUT signals
Control signals: Read,Write, clear A, Set carry in, continue operation, end, etc.
ALU signals: Add, Sub,etc;

There are 46 signals and hence each microinstruction will have 46 bits.
It is not at all necessary to use all 46 bits for every microinstruction because by using grouping of control signals we minimize number of bits for microinstruction.

Way to reduce number of bits microinstruction:

- Most signals are not needed simultaneously.
- Many signals are mutually exclusive
  e.g. only one function of ALU can be activated at a time.
- A source for data transfers must be unique which means that it should not be possible to get the contents of two different registers on to the bus at the same time.
- Read and Write signals to the memory cannot be activated simultaneously.

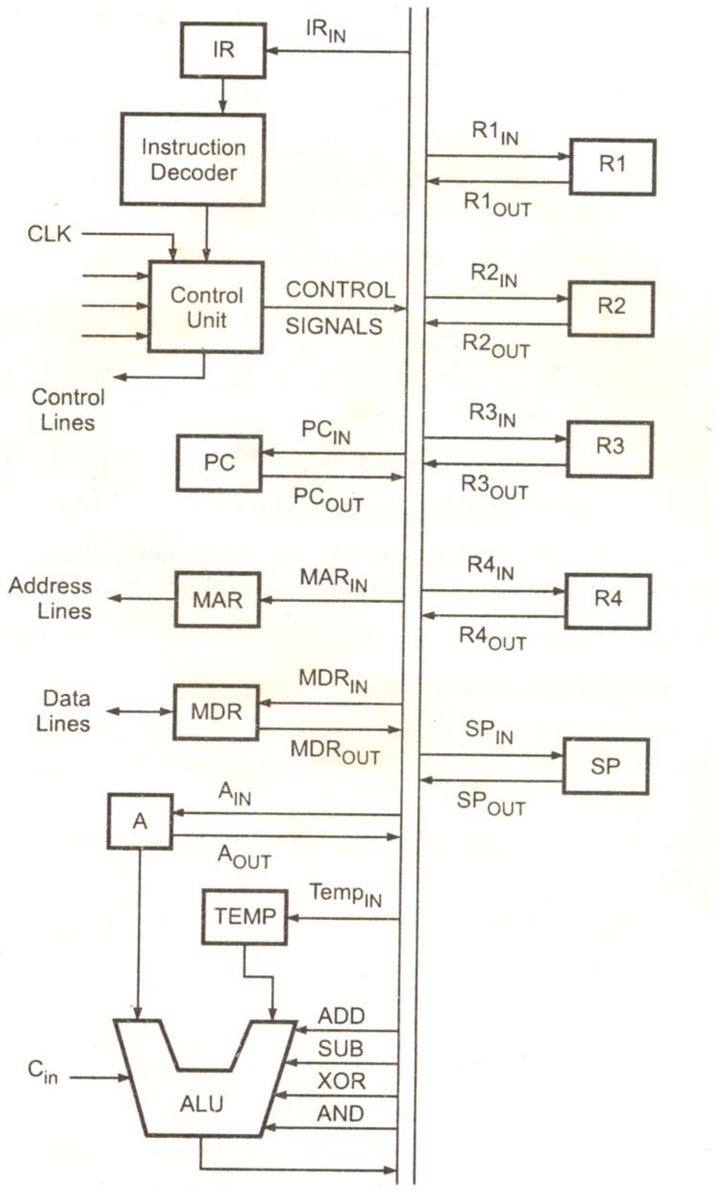
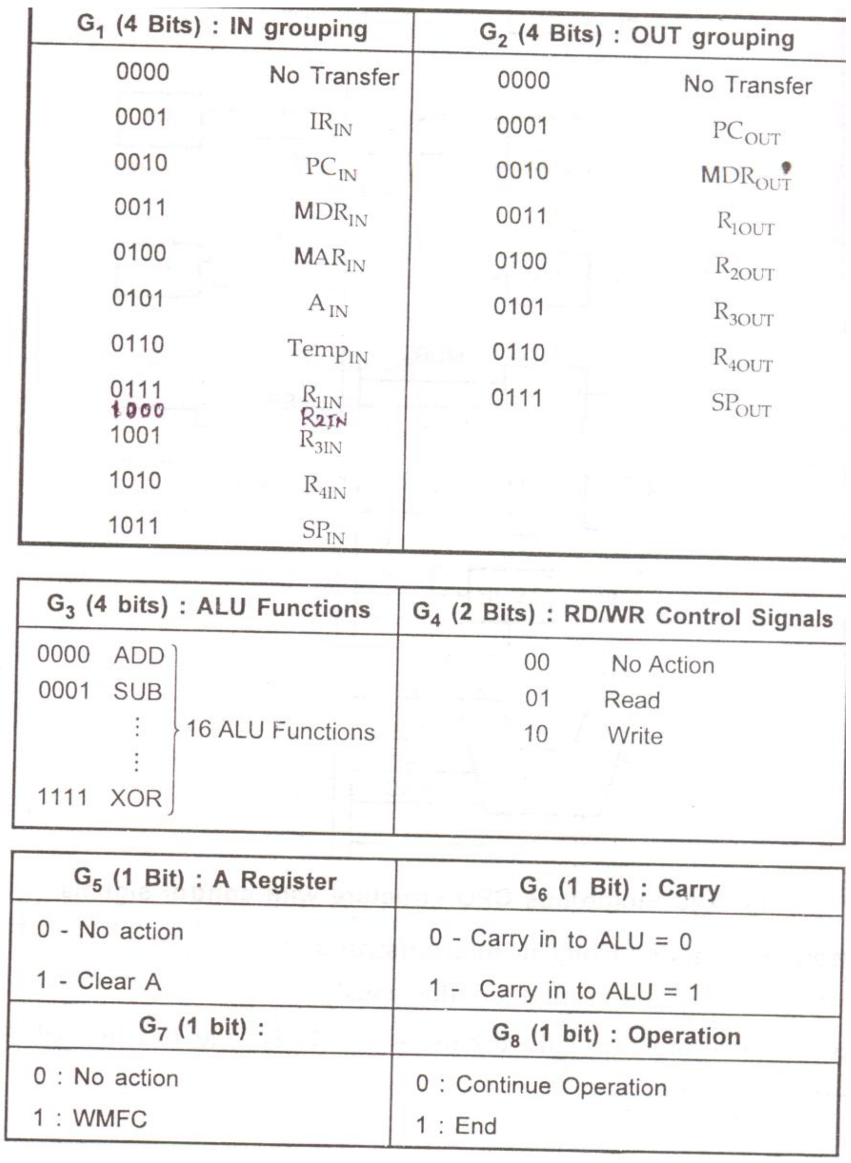


Fig. 3.16 Single bus CPU structure with control signals

46 control signals can be grouped in 7 different groups.



G_1 (4 Bits) : IN grouping		G_2 (4 Bits) : OUT grouping	
0000	No Transfer	0000	No Transfer
0001	IR_{IN}	0001	PC_{OUT}
0010	PC_{IN}	0010	MDR_{OUT}
0011	MDR_{IN}	0011	R_{1OUT}
0100	MAR_{IN}	0100	R_{2OUT}
0101	A_{IN}	0101	R_{3OUT}
0110	$Temp_{IN}$	0110	R_{4OUT}
0111	R_{1IN}	0111	SP_{OUT}
1000	R_{2IN}		
1001	R_{3IN}		
1010	R_{4IN}		
1011	SP_{IN}		

G_3 (4 bits) : ALU Functions			G_4 (2 Bits) : RD/WR Control Signals	
0000	ADD	16 ALU Functions	00	No Action
0001	SUB		01	Read
	⋮		10	Write
	⋮			
1111	XOR			

G_5 (1 Bit) : A Register	G_6 (1 Bit) : Carry
0 - No action	0 - Carry in to ALU = 0
1 - Clear A	1 - Carry in to ALU = 1

G_7 (1 bit) :	G_8 (1 bit) : Operation
0 : No action	0 : Continue Operation
1 : WMFC	1 : End

The total number of grouping bits are 17. Therefore, we minimized 46 bits microinstruction to 17 bit microinstruction.

Techniques of grouping of control signals

The grouping of control signal can be done either by using technique called vertical organisation or by using technique called vertical organisation or by using technique called horizontal organisation.

Vertical organisation

Highly encoded scheme that can be compact codes to specify only a small number of control functions in each microinstruction are referred to as a vertical organisation.

Horizontal organisation

The minimally encoded scheme, in which resources can be controlled with a single instruction is called a horizontal organisation.

Comparison between horizontal and vertical organisation

S.No	**Horizontal**	**Vertical**
1	Long formats	Short formats
2	Ability to express a high degree of parallelism	Limited ability to express parallel microoperations
3	Little encoding of the control information	Considerable encoding of the control information
4	Useful when higher operating speed is desired	Slower operating speeds

Advantages of vertical and horizontal organization

1. Vertical approach is the significant factor,it is used to reduce the requirement for the parallel hardware required to handle the execution of microinstructions.
2. Less bits are required in the microinstruction.
3. The horizontal organisation approach is suitable when operating speed of computer is a critical factor and where the machine structure allows parallel usage of a number of resources.

Disadvantages

Vertical approach results in slower operations speed.

Microprogram sequencing

The task of microprogram sequencing is done by microprogram sequencer.

2 important factors must be considered while designing the microprogram sequencer:

a) The size of the microinstruction
b) The address generation time.

The size of the microinstruction should be minimum so that the size of control memory required to store microinstructions is also less.
This reduces the cost of control memory.
With less address generation time, microinstruction can be executed in less time resulting better throughout.
During execution of a microprogram the address of the next microinstruction to be executed has 3 sources:

i. Determined by instruction register
ii. Next sequential address
iii. Branch

- Microinstructions can be shared using microinstruction branching.

Consider instruction ADD src, Rdst.

- The instruction adds the source operand to the contents of register Rdst and places the sum in Rdst, the destination register.

Let us assume that the source operand can be specified in the following addressing modes:

a) Indexed
b) Autoincrement
c) Autodecrement
d) Register indirect
e) Register direct

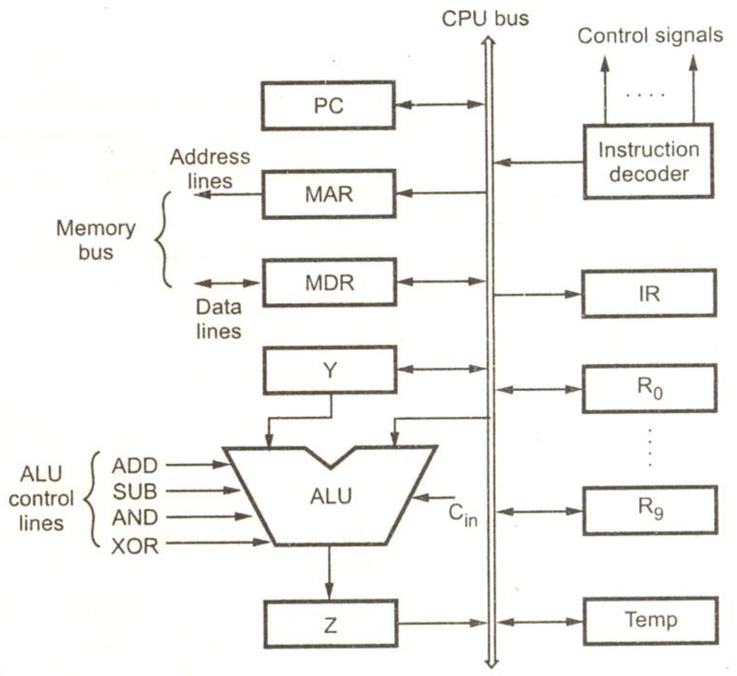


Fig. 3.17 CPU structure

Each box in the flowchart corresponds to a microinstruction that controls the transfers and operations indicated within the box.

The microinstruction is located at the address indicated by the number above the upper right-hand corner of the box.

During the execution of the microinstruction, the branching takes place at point A.

The branching address is determined by the addressing mode used in the instruction.

Techniques for modification or generation of branch addresses

i. Bit-ORing

The branch address is determined by ORing particular bit or bits with the current address of microinstruction.

Eg: If the current address is 170 and branch address is 172 then the branch address can be generated by ORing 02(bit 1), with the current address.

i. **Using condition variables**

It is used to modify the contents CM address register directly, thus eliminating whole or in part the need for branch addresses in µµµµµµmicroinstructions.

Eg: Let the condition variable CY indicate occurance of CY = 1, and no carry when CY = 0.

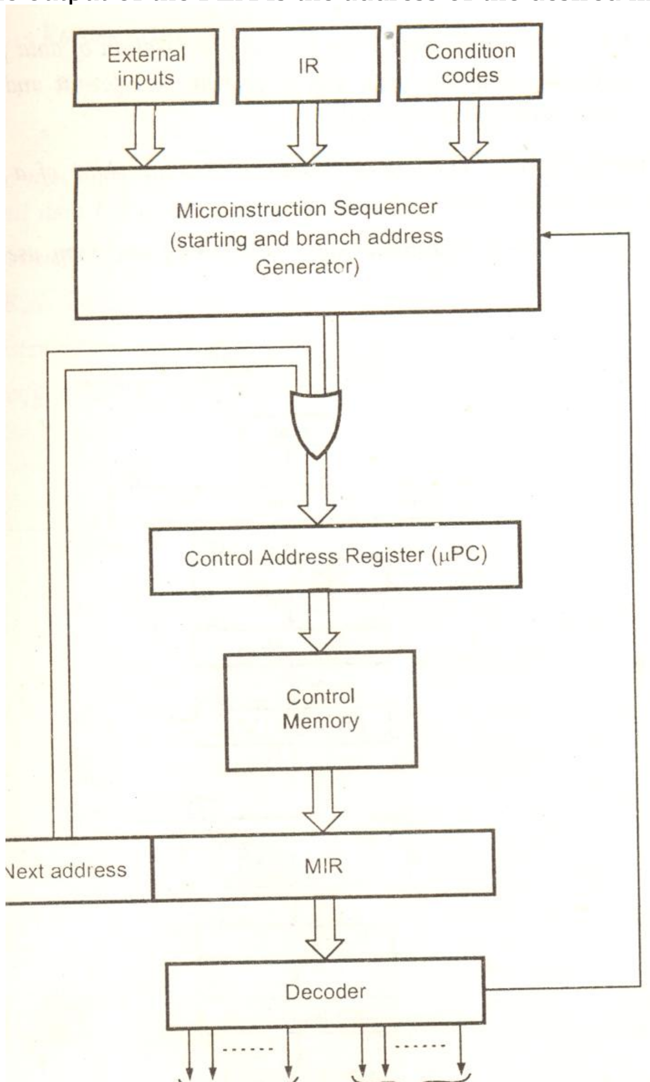
Suppose that we want to execute a SKIP_ON_CARRY microinstruction.

It can be done by logically connecting CY to the count enable input of µpc at on appropriate point in the microinstruction cycle.

It allows the overflow condition increment µpc an extra time, thus performing the desired skip operation.

iii. **Wide-Brance Addressing**

Generating branch addresses becomes more difficult as the number of branches increases.In such situations programmable logic array can be used to generate the required branch addresses.The simple and inexpensive way of generating branch addresses is known as wide-branch addressing.The opcode of a machine instruction is translated into the starting address of the corresponding micro-routine.This is achieved by connecting the opcode bits of the instruction register as inputs to the PLA , which acts as a decoder.The output of the PLA is the address of the desired microroutine.


External inputs
IR
Condition codes
Microinstruction Sequencer (starting and branch address Generator)
Control Address Register (µPC)
Control Memory
Next address
MIR
Decoder

Comparison between Hardwired and Microprogrammed Control

Attribute	**Hardwired Control**	**Microprogrammed Control**
Speed	Fast	Slow
Control functions	Implemented in hardware	Implemented in software
Flexibility	Not flexible to accommodate new system specifications or new instructions	More flexible, to accommodate new system specification or new instructions redesign is required
Ability to handle large/complex instruction sets	Difficult	Easier
Ability to support operating systems and diagnostic features	Very difficult	Easy
Design process	Complicated	Orderly and systematic
Applications	Mostly RISC microprocessors	Mainframes, some microprocessors
Instructionset size	Usually under 100 instructions	Usually over 100 instructions
ROM size	**-**	2K to 10K by 20-400 bit microinstructions
Chip area efficiency	Uses least area	Uses more area

UNIT III

PIPELINING

- ➢ Basic concepts
- ➢ Data hazards
- ➢ Instruction hazards
- ➢ Influence on instruction sets
- ➢ Data path and control considerations
- ➢ Performance considerations
- ➢ Exception handling

Basic Concepts

It is a particularly effective way of organizing concurrent activity in a computer system.

Sequencial execution

Hardware organization

An inter-stage storage buffer, B1, is needed to hold the information being passed from one stage to the next.

New information is loaded into this buffer at the end of each clock cycle.

F	Fetch: read the instruction from the memory
D	Decode: decode the instruction and fetch the source operand(s)
E	Execute: perform the operation specified by the instruction
W	Write: store the result in the destination location

Pipelined execution

- In the first clock cycle, the fetch unit fetches an instruction I_1 (step F_1) and stores it in buffer B1 at the end of the clock cycle.
- In the second clock cycle the instruction fetch unit proceeds with the fetch operation for instruction I_2 (step F_2).
- Meanwhile, the execution unit performs the operation specified by instruction I_1,which is available to it in buffer B1 (step E_1).
- By the end of the second clock cycle, the execution of instruction I_1 is completed and instruction I_2 is available.
- Instruction I_2 is stored in B1, replacing I_1, which is no longer needed.
- Step E_2 is performed by the execution unit during the third clock cycle, while instruction I_3 is being fetched by the fetch unit.

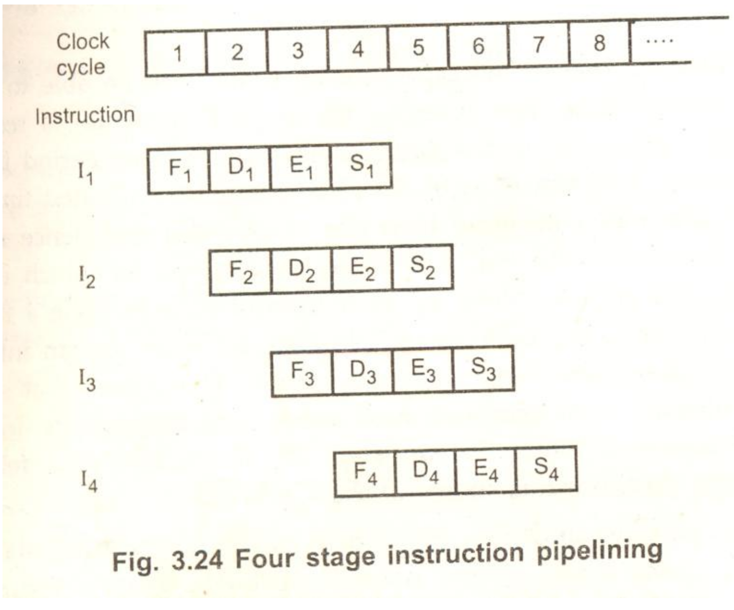


Fig. 3.24 Four stage instruction pipelining

- Four instructions are in progress at any given time.
- So it needs four distinct hardware units.

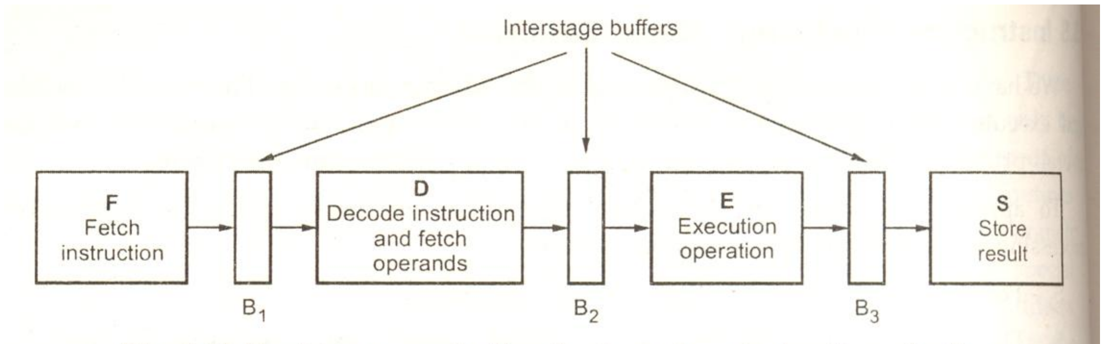


Fig. 3.25 Hardware organisation for four-stage instruction pipeline

3.8.1 Hazards in Instruction Pipelining

These units must be capable of performing their tasks simultaneously and without interfering with one another.
Information is passed from one unit to the next through a storage buffer.
During clock cycle 4, the information in the buffers is as follows:

- Buffer B1 holds instruction I_3, which was fetched in cycle 3 and is being decoded by the instruction-decoding unit.
- Buffer B2 holds both the source operands for instruction I_2 and the specifications of the operation to be performed. This is the information produced by the decoding hardware in cycle 3.
    - The buffer also holds the information needed for the write step of instruction I_2(step W_2).
    - Even though it is not needed by stage E, this information must be passed on to stage W in the following clock cycle to enable that stage to perform the required write operation.
- Buffer B3 holds the results produced by the execution unit and the destination information for instruction I_1.

Pipeline performance

- Pipelining is proportional to the number of pipeline stages.
- For variety of reasons, one of the pipeline stages may not be able to complete its processing task for a given instruction in the time allotted.
- For eg, stage E in the four-stage pipeline is responsible for arithmetic and logic operations and one clock cycle is assigned for this task.
- Although this may be sufficient for most operations some operations such as divide may require more time to complete.
- Instruction I_2 requires 3 cycles to complete from cycle 4 through cycle 6.
- Thus in cycles 5 and 6 the write stage must be told to do nothing, because it has no data to work with.
- Meanwhile, the information in buffer B2 must remain intact until the execute stage has completed its operation.

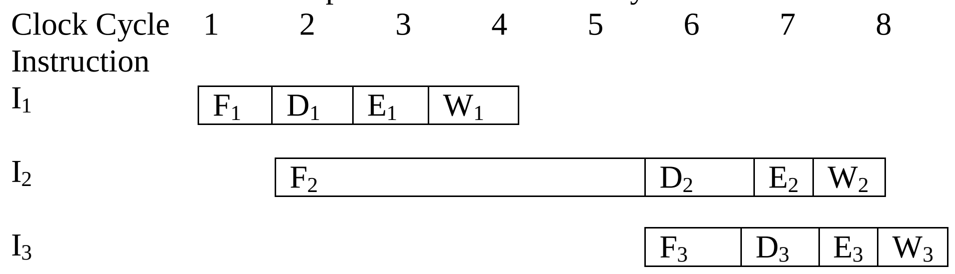
- This means that stage 2 and in turn stage1 are blocked from accepting new instructions because the information in B1 cannot be overwritten.
- Thus steps D_4 and F_5 must be postponded.

❖ The pipeline may also be stalled because of a delay in the availability of an instruction.

❖ For example, this may be a result of a miss in the cache, requiring the instruction to be fetched from the main memory.

❖ Such hazards are often called control hazards or instruction hazards.

Instruction execution steps in successive clock cycles:



Function performed by each process stage in successive clock cycles

Clock Cycle	1	2	3	4	5	6	7	8	9
Stage									
F: Fetch	F_1	F_2	F_2	F_2	F_2				
D: Decode		D_1	idle	idle	idle	D_2	D_3		
E: Execute			E_1	idle	idle	idle	E_2	E_3	
W : Write				W_1	idle	idle	idle	W_2	W_3

- Instruction I_1 is fetched from the cache in $cycle_1$,and its execution proceeds normally.
- Thus in cycles 5 and 6, the write stage must be told to do nothing, because it has no data to work with.
- Meanwhile ,the information in buffer B2 must remain intact until the execute stage has completed its operation.
- This means that stage 2 and in turn stage1 are blocked from accepting now instructions because the information in B1 cannot be overwritten.
- Thus steps D4 and F4 must be postponed.
- Pipelined operation is said to have been stacked for two clock cycles.
- Normal pipelined operation resumes in cycle 7.

Hazard:

Any location that causes the pipeline to stall is called hazard.

Data Hazard

A data hazard is any conditions in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result some operation has to be delayed ,and the pipeline stalls.

b)Pipeline performance

For a variety of reasons, one of the pipeline stages may not be able to complete its processing task for a given instruction I the time allotted.

For Ex. Stage E in the 4 stage pipeline is responsible for arithmetic and logic operations and one clock cycle is assigned for this task.

Although this may be sufficient for most operations, same operations such as divide may require more time to complete.

Effect of an execution operation talking more than one clock cycle:

- The operation specified in instruction I2 requires three cycles to complete ,from cycle 4 through cycle 6.

Pipe Lining:

1. **Basic Concepts:**
   Pipelining is a particularly effective way of organization concurrent activity in a computer system.

Instruction pipeline:

The fetch, decode and execute cycles for several instructions are performed simultaneously to reduce overall processing time. This process is referred to as instruction pipelining.

Consider 4 stage process

F → Fetch : read the instruction from the memory
D→ Decode: Decode the instruction and fetch the some operands
E→ Execute: perform the operation specified by the instruction.
W→ Write: Store the results in the destination location.

a) Instruction execution divided into 4 steps:

During

Influence on Instruction sets

1. Addressing modes

Many processors provide various combinations of addressing modes such as index,indirect,auto increment, auto decrement and so on.

We can classify these addressing modes as simple addressing modes and complex addressing modes.

A complex address mode may require several accesses to the memory to reach the named operand while simple addressing mode requires only one access to the memory to reach the named operand.

Consider the instruction Load R1,(X(R0)).

This instruction has a complex addressing mode because this instruction requires two memory accesses one to read location X+[R0] and then to read location [X+[R0]].

If R1 is a source operand in the next instruction that instruction has to be stalled for two cycles.

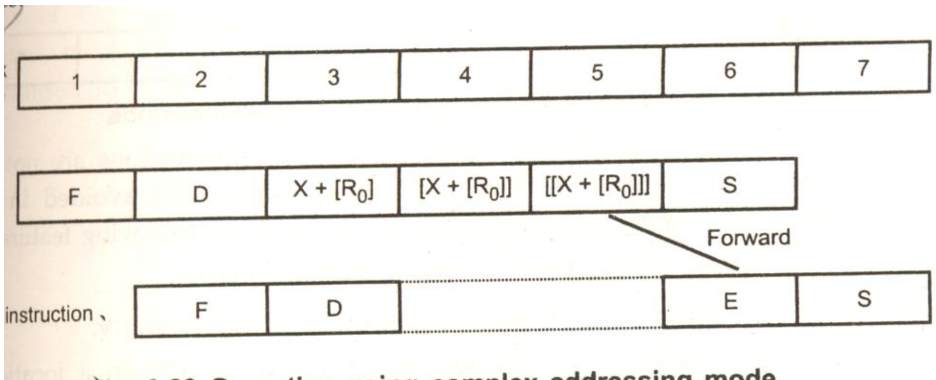


Fig. 3.29 Operation using complex addressing mode

If we want to perform the same operation using simple addressing mode instructions we require to execute three instructions:

ADD R1,R0,#X
LOAD R1,(R2)
LOAD R1,(R1)

The ADD instruction performs the operation R1 $\leftarrow$ [R0]+X.

The two load instructions fetch the address and then the operand from the memory.

The two load instructions fetch the address and then the operand from the memory.

This sequence of instructions takes exactly the same number of clock cycles as the single load instruction having a complex addressing mode.

The above example indicates that in a pipeline processor, complex addressing mode do not necessarily lead to faster execution.

But it reduce the number of instructions needed to perform particular task and hence reduce the program space needed in the main memory.

Disadvantages

a) Long execution times of complex addressing mode instructions may cause pipeline to stall.
b) They require more complex hardware to decode and execute them.
c) They are not convenient for compilers to work with.

Due to above reasons the complex addressing mode instructions are not suitable for pipelined execution.
These addressing modes are avoided in modern processors.

Features of modern processor addressing modes

a) They access operand from the memory in only one cycle.
b) Only load and store instructions are provided to access memory.
c) The addressing modes used do not have side effects.(when a location other than one explicitly named in an instruction as a destination operand is affected, the instruction is said to have a side effect).

2. Condition codes

Most of the processors store condition code flags in their status register.
The conditional branch instruction checks the condition code flag and the flow of program execution is decided.
When the data dependency or branch exists, the pipeline may stall.
A special optimizing compiler is designed which reorders the instructions in the program which is to be executed.
This avoids the stalling the pipeline, the compiler takes the care to obtain the correct output while reordering the instructions.
To handle condition codes,

a) The condition code flags should be affected by as few instructions as possible.
It provides flexibility in reordering instructions.
Otherwise, the dependency produced by the condition code flags reduces the flexibility available for the compiler reorder instructions.
b) It should be known to compiler that in which instructions of a program the condition codes are affected and in which they remain unaffected.

Datapath and control considerations

When single bus is used in a processor only one data word can be transferred over the bus in a clock cycle.
This increases the steps required to complete the execution of the instruction.
To reduce the number of steps needed to execute instructions most commercial processors provide multiple internal paths that enable several transfer to take place in parallel.

Modified 3 bus structure of the processor for pipelined execution

3 buses are used to connect registers and the ALU of the processor.
All general purpose registers are connected by a single block called register file.

It has 3 ports:

- One input port
- Two output ports

It is possible to access data of 3 register in one clock cycle, the value can be loaded in one register from bus C and data from two register can be accessed to bus A and bus B respectively.

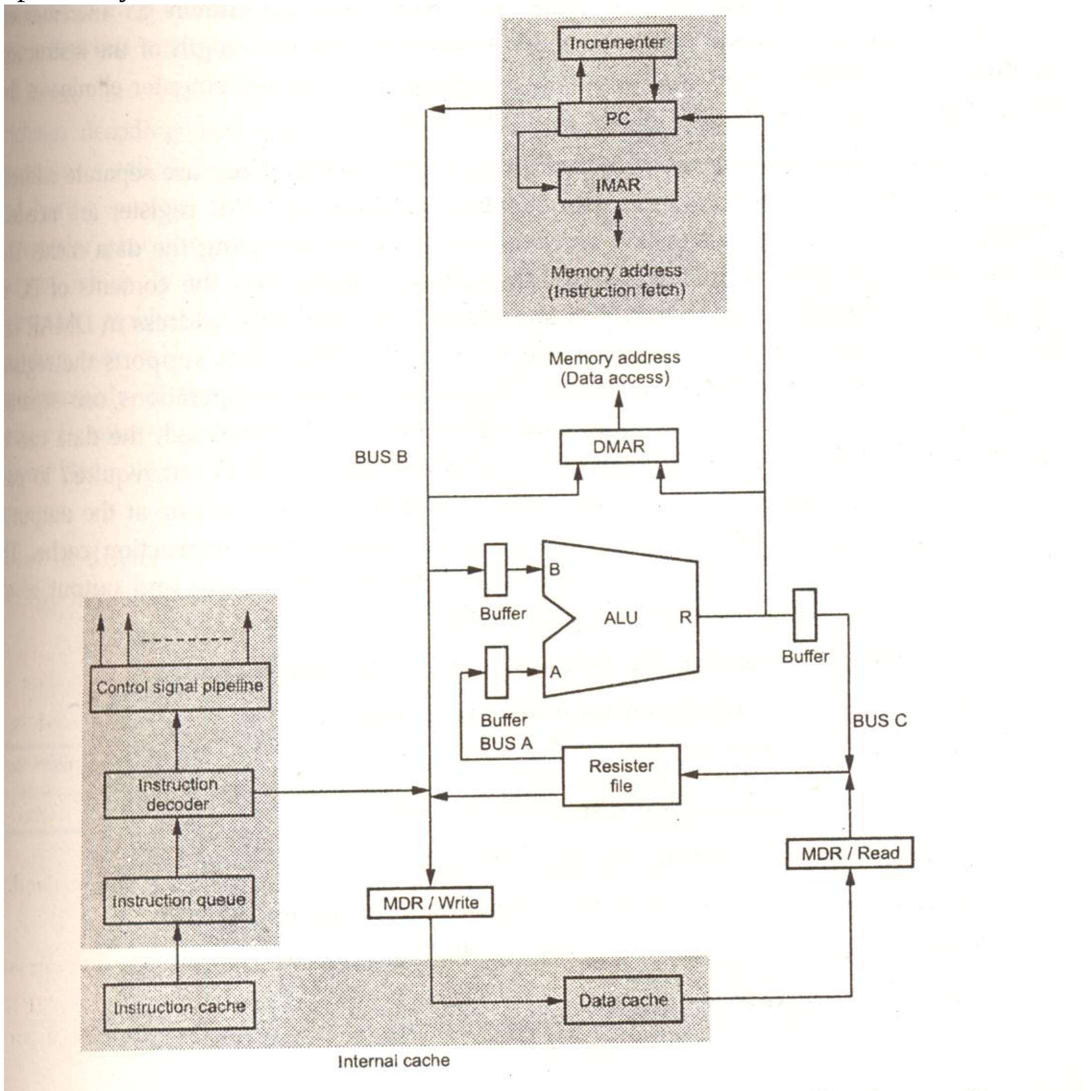


Fig. 3.31 Modified three-bus structure of the processor for pipelined execution

Buses A and B are used to transfer the source operands to the A and B inputs of the ALU.

After performing arithmetic or logic operation result is transferred to the destination operand over bus C.

To increment the contents of PC after execution of each instruction to fetch the next instruction separate unit is provided, it is known as incrementer.

Incrementer increments the contents of PC accordingly to the length of the instruction so that it can point to next instruction in the sequence.

The processor uses separate instruction and data caches.

They use separate address and data connections to the processor.

Two versions of MAR register are needed IMAR for accessing the instruction cache and DMAR for accessing the data cache.

The PC is connected directly to the IMAR.

This allows transferring the contents of PC to IMAR and performing ALU operation simultaneously.

The data address in DMAR can be obtained directly from the register file or from the ALU.

It supports the register indirect and indexed addressing modes.

The read and write operations use separate MDR registers.

When load and store operations are to be performed the data can be transferred directly between these registers file.

It is not required to pass this data through the ALU.

Two buffer registers at the input and one at the output of the ALU are used.

The instruction queue gets loaded from instruction cache.

The output of the queue is connected to the instruction decoder input and output of the decoder is connected to the control signal pipeline.

The processor can perform the following operations independently,

1. Reading an instruction from the instruction cache.
2. Incrementing the PC.
3. Decoding an instruction by instruction decoder.
4. Reading from and writing into the data cache.
5. Reading the contents of up to two registers from the register file.
6. Writing into one register in the register file.
7. Performing an ALU operation.

Superscalar Operation

- Several instructions can be executed concurrently because of pipelining. However, these instructions in the pipeline are in different stages of execution such as fetch, decode, ALU operation.
- When one instruction is fetch phase, one instruction is completing its execution in the absence of hazards.
- Thus at the most, one instruction is executed in each clock cycle.
- This means the maximum throughput of the processor is one instruction per clock cycle when pipelining is used.
- Processors reach performance levels greater than one instruction per cycle by fetching, decoding and executing several instructions concurrently.

- This mode of operation is called superscalar.
- A processor capable of parallel instruction per cycle is known as superscalar processor.
- A superscalar processor has multiple execution units (E-units), each of which is usually pipelined, so that they constitute a set of independent instruction pipelines.
- Its program control unit (PCU) is capable of fetching and decoding several instructions concurrently.
- It can issue multiple instructions simultaneously.
- If the processor can issue up to k instructions simultaneously to the various E-units, then k is called instruction issue degree.
- It can be six or more using current technology.

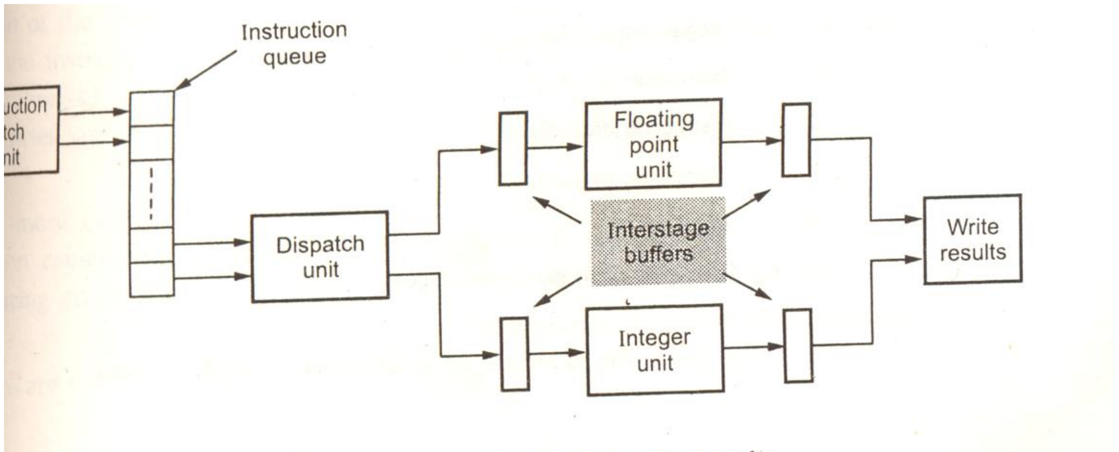
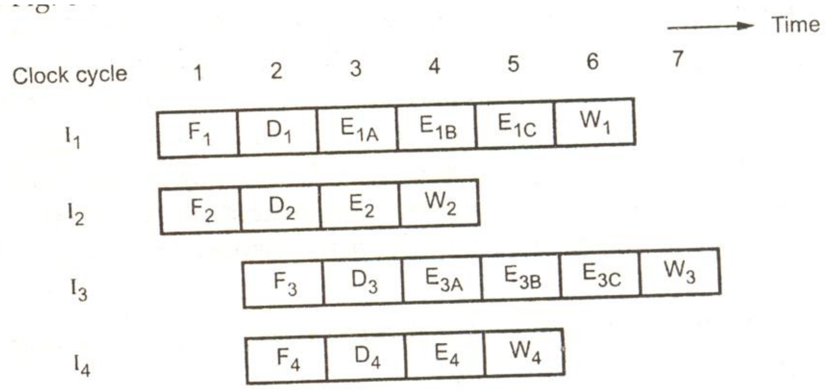


Fig. 3.32 A processor with two execution units

- A processor has two execution units,
  - i) Integer unit (used for integer operations)
  - ii) Floating point unit (used for floating point operations)

- The instruction fetch unit can read two instructions simultaneously and stores them temporarily in the instruction queue.
- The queue operates on the principle first in first out (FIFO).
- In each clock cycle, the dispatch unit can take two instructions from the instruction queue and decodes them.
- If these instructions are such that one is integer and another is floating-point with no hazards, then both instructions are dispatched in the same clock cycle.
- The compiler is designed to avoid hazards by proper selection and ordering of the instructions.


Time
Clock cycle 1 2 3 4 5 6 7
I_1 F_1 D_1 E_{1A} E_{1B} E_{1C} W_1
I_2 F_2 D_2 E_2 W_2
I_3 F_3 D_3 E_{3A} E_{3B} E_{3C} W_3
I_4 F_4 D_4 E_4 W_4

Note : I_1, I_3	: Floating point Instruction
I_2 : I_4	: Integer instructions
F	: Instruction fetching
D	: Instruction decoding
E	: Instruction execution
W	: Write results
E_A, E_B, E_C	: Instruction execution stages

It is assumed that no hazards are encountered.

Fig. 3.33. An example of instruction execution flow

- It is [illegible] t takes one clock cycle to complete execution.
  1. The instruction fetch unit fetches instructions I1 and I2 in clock cycle.
  2. I1 and I2 enter the dispatch unit in cycle 2. The fetch unit fetches next two instructions, I3 and I4 during the same cycle.
  3. The floating point unit takes three clock cycles to complete the floating point operation(Execution: EA,EB,EC) specified in I1. The integer unit completes execution of I2 in one clock cycle(clock cycle 3). Also, instructions I3 and I4 enter the dispatch unit in cycle 3.
  4. We have assumed the floating point unit as a three-stage pipeline. So it can accept a new instruction for execution in each clock cycle. So during clock cycle 4, though the execution of I1 is in progress, it accepts I3 for the execution. The integer unit can accept a new instruction for execution because instruction I2 has entered to the write stage. Thus, instructions I3 and I4 are dispatched in cycle 4. Instruction I1 is still in the execution phase, but in the second stage of the internal pipeline in floating-point unit.
  5. The instructions complete execution with no hazards as shown in further clock cycles.

Out-of-order Execution

- The dispatch unit dispatches the instructions in the order in which they appear in the program. But their execution may be completed in the different order.
- For example, execution of I2 is completed before the complete execution of I1.
- Thus the execution of the instructions is completed out of order.
- If there is a data dependency among the instructions, the execution of the instructions is delayed.
- For example, if the execution of I2 needs the results of execution of I1, I2 will be delayed.
- If such dependencies are handled correctly, the execution of the instructions will not be delayed.
- There are two causes of exceptions,
    - a bus error (during an operand fetch)
    - illegal operation(eg. Divide by zero)
    -

2 types of exceptions,

- Imprecise exceptions
- Precise exceptions

Imprecise exception

Consider the pipeline timing, the result of operation of I2 is written into the register file in cycle 4.

If instruction I1 causes an exception and succeeding instructions are permitted to complete execution, then the processor is said to have imprecise exceptions. Because of the exception by I1, program execution is in an inconsistent state.

Precise exception

In the imprecise exception, consistent state is not guaranteed when an exception occurs.
If the exception occurred then to guarantee a consistant state, the result of the execution of instructions must be written into the destination locations strictly in the program order.
The result of execution of I2 is written to the destination in cycle 4(W2).
But the result of I1 is written to the destination in cycle 6(W1).
So step W2 is to be delayed until cycle 6.
The integer execution unit has to retain the result of execution of I2.
So it cannot accept instruction I4 until cycle 6.
Thus in precise exception, if the exception occurs during an instruction, the succeeding instructions, may have been partially executed are discarded.
If an external interrupt is received, the dispatch unit stops reading new instructions from the instruction queue.
The instructions which are placed in the instruction queue are discarded.
The processor first completes the pending execution of the instructions to completion.
The consistent state of the processor and all its registers is achieved.

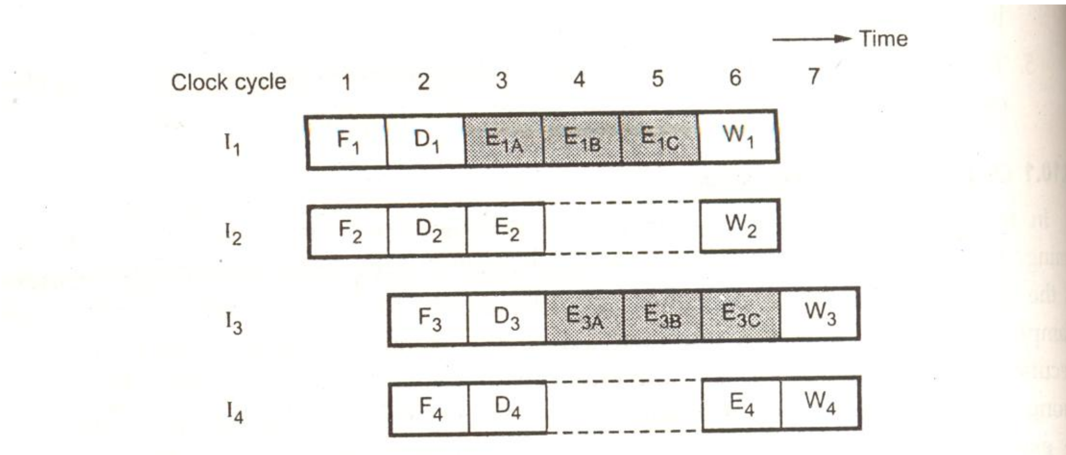


Fig. 3.34 Instruction execution completion in program order with delayed write

Then the processor starts the interrupt handling process.

Execution completion

The decoding instructions are stored temporarily into the temporary registers and later they are transferred to the permanent registers in correct program order.

Thus 2 write operations TW and W respectively are carried out.

The step W is called the commitment step because the final result gets stored into the permanent register during this step.

Eventhough, exception occurs, the result of succeeding instructions would be in temporary registers and hence can be safely discarded.

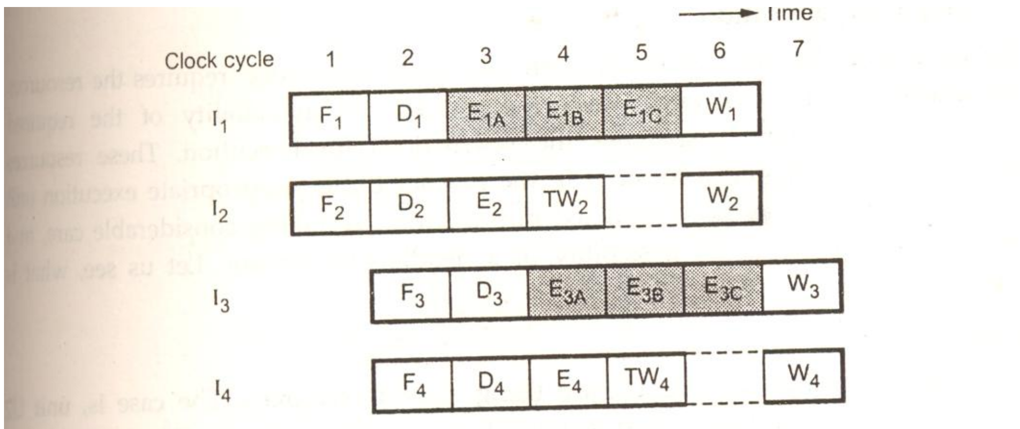


Fig. 3.35 Instruction execution completion in program order using temporary registers

lote : TW : Write into a temporary register

W : Transfer contents of TW into permanent register

Register renaming

- When temporary register holds the contents of the permanent register, the name of permanent register is given to that temporary register is called as register renaming.
- For example, if I2 uses R4 as a destination register, then the temporary register used in step TW2 is also reffered as R4 during cycles 6 and 7, that temporary register used only for instructions that follow I2 in program order.
- For example, if I1 needs to read R4 in cycle 6 or 7, it has to access R4 though it contains unmodified data be I2.

Commitment Unit

It is a special control unit needed to guarantee in-order commitment when out-of-order execution is allowed.

It uses a special queue called the reorder buffer which stores the instruction committed next.

Retired instructions

- The instructions entered in the reorder buffer(queue)of the commitment unit strictly in program order.
- When the instruction from this queue is executed completely, the results obtained from it are copied from temporary registers to the permanent registers and instruction is removed from the queue.
- The resources which were assigned to the instruction are released.
- At this stage, the instruction is said to have been retired.
- The instructions are retired in program order though they may complete execution out of order.
- That is, all instructions that were dispatched before it must also have been retired.

Dispatch operation

Each instruction requires the resources for its execution.
The dispatch unit first checks the availability of the required resources and then only dispatches the instructions for execution.
These resources include temporary register, a location in the order buffer, appropriate execution unit etc.

Deadlock

Consider 2 units U1 and U2 are using shared resources.
U2 needs completion of the task assign to unit U1 to complete its task.
If unit U2 is using a resource which is also required to unit U1, both units cannot complete the tasks assigned to them.
Both the units remain waiting for the need resource.
Also, unit U2 is waiting for the completion of task by unit U1 before it can release that resource. Such a situation is called a deadlock.

UNIT IV

MEMORY SYSTEM

- Basic concepts
- Semiconductor RAM
- ROM
- Speed ,Size and cost
- Cache memories
- <u>Improving cache performance</u>
- Virtual memory
- Memory management requirements
- <u>Associative memories</u>
- Secondary storage devices

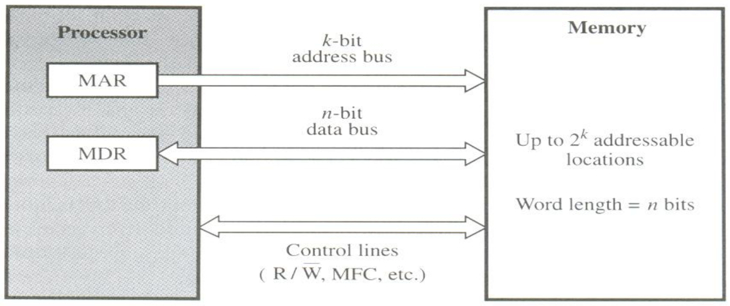
Performance consideration is here

BASIC CONCEPTS

The maximum size of the memory that can be used in any computer is determined by the addressing scheme.

Address	Memory Locations
16 Bit	2^{16} = 64 K
32 Bit	2^{32} = 4G (Giga)
40 Bit	2^{40} = IT (Tera)

Fig: Connection of Memory to Processor:



- If MAR is k bits long and MDR is n bits long, then the memory may contain upto 2^K addressable locations and the n-bits of data are transferred between the memory and processor.
- This transfer takes place over the processor bus.
- The processor bus has,
  - Address Line
  - Data Line
  - Control Line (R/W, MFC – Memory Function Completed)
- The control line is used for co-ordinating data transfer.
- The processor reads the data from the memory by loading the address of the required memory location into MAR and setting the R/W line to 1.
- The memory responds by placing the data from the addressed location onto the data lines and confirms this action by asserting MFC signal.
- Upon receipt of MFC signal, the processor loads the data onto the data lines into MDR register.
- The processor writes the data into the memory location by loading the address of this location into MAR and loading the data into MDR sets the R/W line to 0.

Memory Access Time → It is the time that elapses between the intiation of an

Operation and the completion of that operation.

Memory Cycle Time → It is the minimum time delay that required between the initiation of the two successive memory operations.

RAM (Random Access Memory):

In RAM, if any location that can be accessed for a Read/Write operation in fixed amount of time, it is independent of the location's address.

Cache Memory:

- It is a small, fast memory that is inserted between the larger slower main memory and the processor.
- It holds the currently active segments of a pgm and their data.

Virtual memory:

- The address generated by the processor does not directly specify the physical locations in the memory.
- The address generated by the processor is referred to as a virtual / logical address.
- The virtual address space is mapped onto the physical memory where data are actually stored.
- The mapping function is implemented by a special memory control circuit is often called the memory management unit.
- Only the active portion of the address space is mapped into locations in the physical memory.
- The remaining virtual addresses are mapped onto the bulk storage devices used, which are usually magnetic disk.
- As the active portion of the virtual address space changes during program execution, the memory management unit changes the mapping function and transfers the data between disk and memory.
- Thus, during every memory cycle, an address processing mechanism determines whether the addressed in function is in the physical memory unit.
- If it is, then the proper word is accessed and execution proceeds.
- If it is not, a page of words containing the desired word is transferred from disk to memory.
- This page displaces some page in the memory that is currently inactive.

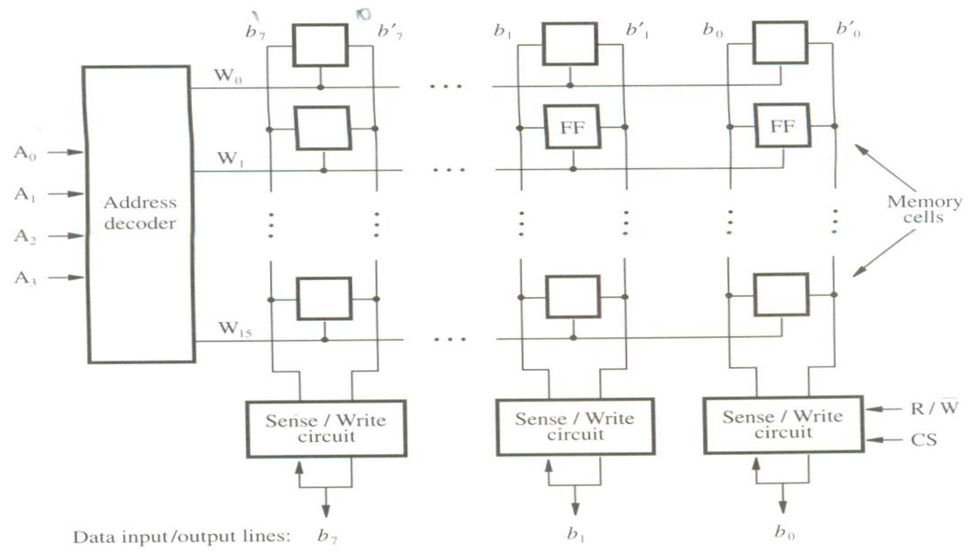
SEMI CONDUCTOR RAM MEMORIES:

- Semi-Conductor memories are available is a wide range of speeds.
- Their cycle time ranges from 100ns to 10ns

INTERNAL ORGANIZATION OF MEMORY CHIPS:

- Memory cells are usually organized in the form of array, in which each cell is capable of storing one bit of in formation.
- Each row of cells constitute a memory word and all cells of a row are connected to a common line called as **word line**.
- The cells in each column are connected to Sense / Write circuit by two bit lines.
- The Sense / Write circuits are connected to data input or output lines of the chip.
- During a write operation, the sense / write circuit receive input information and store it in the cells of the selected word.

Fig: Organization of bit cells in a memory chip



- The data input and data output of each senses / write ckt are connected to a single bidirectional data line that can be connected to a data bus of the cptr.

R / W →Specifies the required operation.

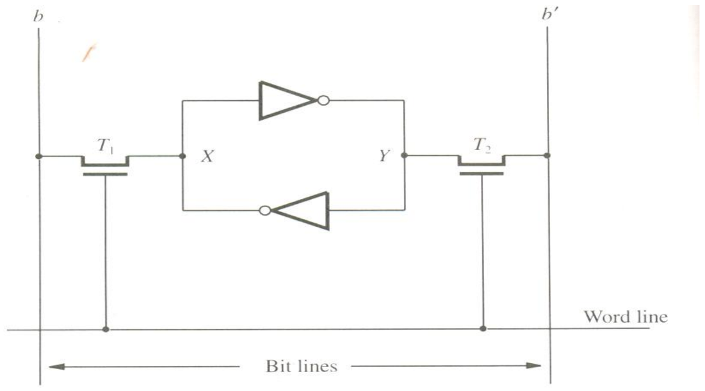
CS →**Chip Select** input selects a given chip in the multi-chip memory system

Bit Organization	**Requirement of external connection for address, data and control lines**
128 (16x8)	14
(1024) 128x8(1k)	19

Static Memories:

Memories that consists of circuits capable of retaining their state as long as power is applied are known as **static memory.**

Fig:Static RAM cell



- Two inverters are cross connected to form a batch
- The batch is connected to two bit lines by transistors T_1 and T_2.
- These transistors act as switches that can be opened / closed under the control of the word line.
- When the wordline is at ground level, the transistors are turned off and the latch retain its state.

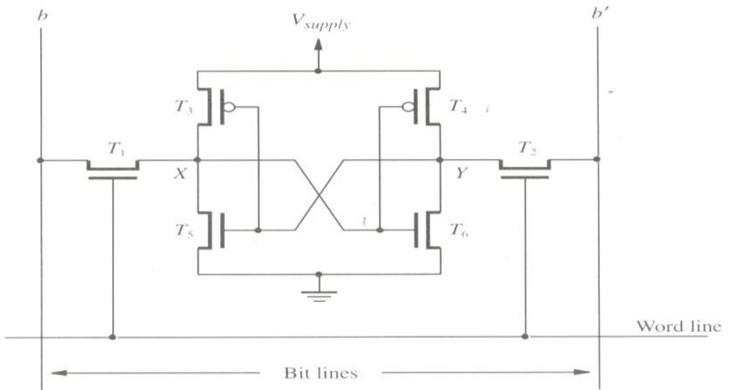
Read Operation:

- In order to read the state of the SRAM cell, the word line is activated to close switches T_1 and T_2.
- If the cell is in state 1, the signal on bit line b is high and the signal on the bit line b is low.Thus b and b are complement of each other.
- Sense / write circuit at the end of the bit line monitors the state of b and b' and set the output accordingly.

Write Operation:

- The state of the cell is set by placing the appropriate value on bit line b and its complement on b and then activating the word line. This forces the cell into the corresponding state.
- The required signal on the bit lines are generated by Sense / Write circuit.

Fig:CMOS cell (Complementary Metal oxide Semi Conductor):



- Transistor pairs (T_3, T_5) and (T_4, T_6) form the inverters in the latch.
- In state 1, the voltage at point X is high by having T_5, T_6 on and T_4, T_5 are OFF.
- Thus T_1, and T_2 returned ON (Closed), bit line b and b will have high and low signals respectively.
- The CMOS requires 5V (in older version) or 3.3.V (in new version) of power supply voltage.
- The continuous power is needed for the cell to retain its state

Merit :

- It has low power consumption because the current flows in the cell only when the cell is being activated accessed.
- Static RAM's can be accessed quickly. It access time is few nanoseconds.

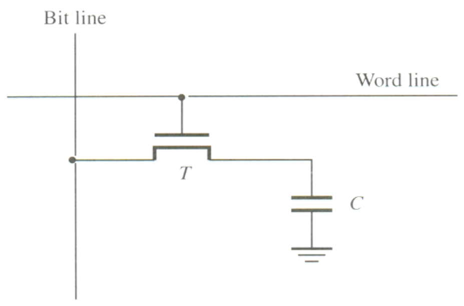
Demerit:

- SRAM's are said to be volatile memories because their contents are lost when the power is interrupted.

Asynchronous DRAMS:-

- Less expensive RAM's can be implemented if simplex calls are used such cells cannot retain their state indefinitely. Hence they are called **Dynamic RAM's (DRAM).**
- The information stored in a dynamic memory cell in the form of a charge on a capacitor and this charge can be maintained only for tens of Milliseconds.
- The contents must be periodically refreshed by restoring by restoring this capacitor charge to its full value.
-

Fig:A single transistor dynamic Memory cell

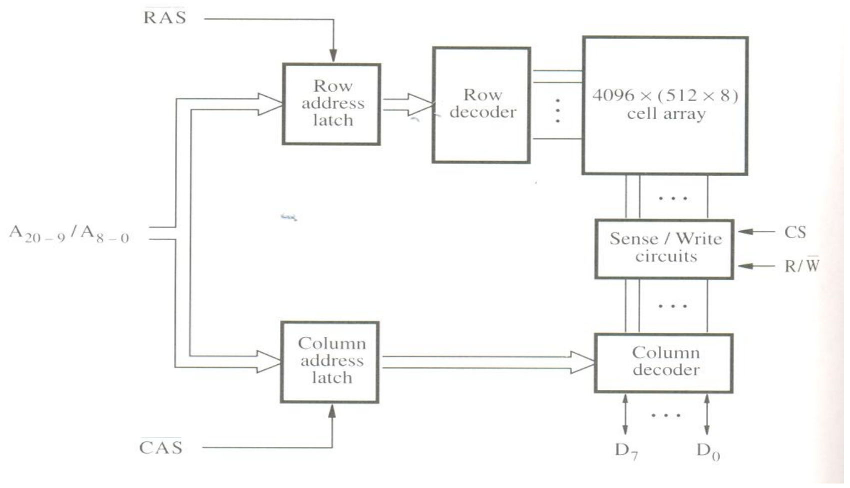


- In order to store information in the cell, the transistor T is turned 'on' & the appropriate voltage is applied to the bit line, which charges the capacitor.
- After the transistor is turned off, the capacitor begins to discharge which is caused by the capacitor's own leakage resistance.
- Hence the information stored in the cell can be retrieved correctly before the threshold value of the capacitor drops down.

- During a read operation, the transistor is turned ‘on’ & a sense amplifier connected to the bit line detects whether the charge on the capacitor is above the threshold value.

  If charge on capacitor > threshold value -> Bit line will have logic value ‘1’.
  If charge on capacitor < threshold value -> Bit line will set to logic value ‘0’.

Fig:Internal organization of a 2M X 8 dynamic Memory chip.



DESCRIPTION:

- The 4 bit cells in each row are divided into 512 groups of 8.
- 21 bit address is needed to access a byte in the memory(12 bit→To select a row,9 bit→Specify the group of 8 bits in the selected row).

  A_{8-0} →Row address of a byte.
  A_{20-9} →Column address of a byte.

- During Read/ Write operation ,the row address is applied first. It is loaded into the row address latch in response to a signal pulse on **Row Address Strobe(RAS)** input of the chip.
- When a Read operation is initiated, all cells on the selected row are read and refreshed.
- Shortly after the row address is loaded,the column address is applied to the address pins & loaded into **Column Address Strobe(CAS).**
- The information in this latch is decoded and the appropriate group of 8 Sense/Write circuits are selected.
- R/W =1(read operation)→The output values of the selected circuits are transferred to the data lines D0 - D7.
- R/W =0(write operation)→The information on D0 - D7 are transferred to the selected circuits.

- RAS and CAS are active low so that they cause the latching of address when they change from high to low. This is because they are indicated by RAS & CAS.
- To ensure that the contents of a DRAM 's are maintained, each row of cells must be accessed periodically.
- Refresh operation usually perform this function automatically.
- A specialized memory controller circuit provides the necessary control signals RAS & CAS, that govern the timing.
- The processor must take into account the delay in the response of the memory. Such memories are referred to as **Asynchronous DRAM's.**

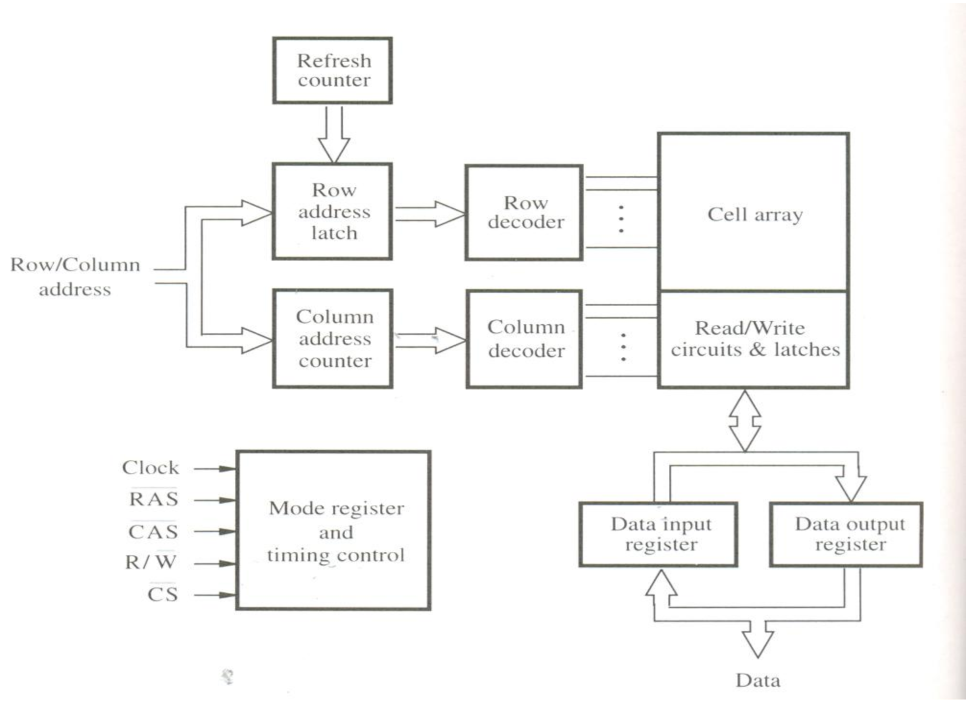
Fast Page Mode:

- Transferring the bytes in sequential order is achieved by applying the consecutive sequence of column address under the control of successive CAS signals.
- This scheme allows transferring a block of data at a faster rate. The block of transfer capability is called as **Fast Page Mode.**

Synchronous DRAM:

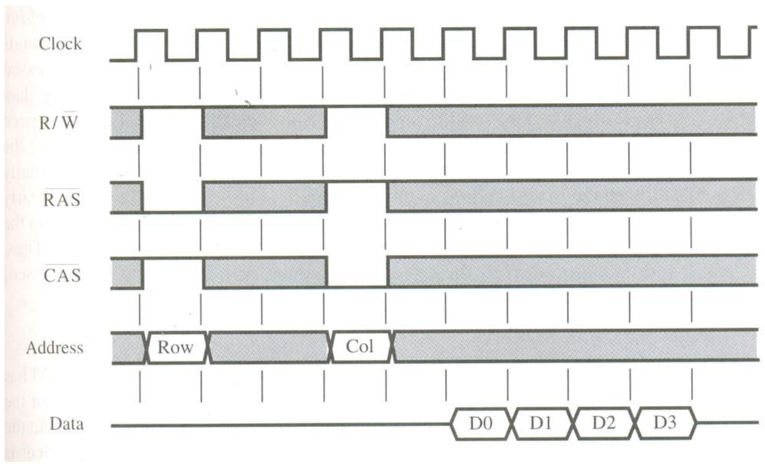
- Here the operations e directly synchronized with clock signal.
- The address and data connections are buffered by means of registers.
- The output of each sense amplifier is connected to a latch.
- A Read operation causes the contents of all cells in the selected row to be loaded in these latches.

Fig:Synchronous DRAM



- Data held in the latches that correspond to the selected columns are transferred into the data output register, thus becoming available on the data output pins.

Fig:Timing Diagram →Burst Read of Length 4 in an SDRAM



- First ,the row address is latched under control of RAS signal.
- The memory typically takes 2 or 3 clock cycles to activate the selected row.
- Then the column address is latched under the control of CAS signal.
- After a delay of one clock cycle,the first set of data bits is placed on the data lines.
- The SDRAM automatically increments the column address to access the next 3 sets of bits in the selected row, which are placed on the data lines in the next 3 clock cycles.

Latency & Bandwidth:

- A good indication of performance is given by two parameters.They are,
  - Latency
  - Bandwidth

Latency:

- It refers to the amount of time it takes to transfer a word of data to or from the memory.
- For a transfer of single word,the latency provides the complete indication of memory performance.
- For a block transfer,the latency denote the time it takes to transfer the first word of data.

Bandwidth:

- It is defined as the number of bits or bytes that can be transferred in one second.

- Bandwidth mainly depends upon the speed of access to the stored data & on the number of bits that can be accessed in parallel.

Double Data Rate SDRAM(DDR-SDRAM):

- The standard SDRAM performs all actions on the rising edge of the clock signal.
- The double data rate SDRAM transfer data on both the edges(loading edge, trailing edge).
- The Bandwidth of DDR-SDRAM is doubled for long burst transfer.
- To make it possible to access the data at high rate , the cell array is organized into two banks.
- Each bank can be accessed separately.
- Consecutive words of a given block are stored in different banks.
- Such interleaving of words allows simultaneous access to two words that are transferred on successive edge of the clock.

Larger Memories:

Dynamic Memory System:

- The physical implementation is done in the form of Memory Modules.
- If a large memory is built by placing DRAM chips directly on the main system printed circuit board that contains the processor ,often referred to as Motherboard;it will occupy large amount of space on the board.
- These packaging consideration have led to the development of larger memory units known as SIMM's & DIMM's .
  SIMM-Single Inline memory Module
  DIMM-Dual Inline memory Module

- SIMM & DIMM consists of several memory chips on a separate small board that plugs vertically into single socket on the motherboard.

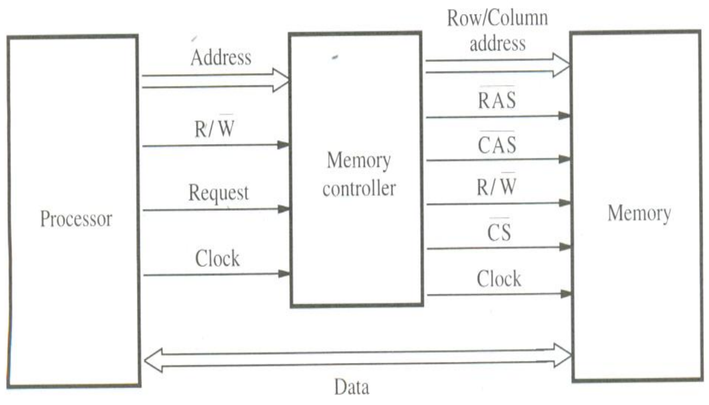
MEMORY SYSTEM CONSIDERATION:

- To reduce the number of pins, the dynamic memory chips use multiplexed address inputs.
- The address is divided into two parts.They are,

  - **High Order Address Bit**(Select a row in cell array & it is provided first and latched into memory chips under the control of RAS signal).
  - **Low Order Address Bit**(Selects a column and they are provided on same address pins and latched using CAS signals).

- The Multiplexing of address bit is usually done by **Memory Controller Circuit.**

Fig:Use of Memory Controller



- The Controller accepts a complete address & R/W signal from the processor, under the control of a Request signal which indicates that a memory access operation is needed.
- The Controller then forwards the row & column portions of the address to the memory and generates RAS &CAS signals.
- It also sends R/W &CS signals to the memory.
- The CS signal is usually active low, hence it is shown as CS.

Refresh Overhead:

- All dynamic memories have to be refreshed.
- In DRAM ,the period for refreshing all rows is 16ms whereas 64ms in SDRAM.

Eg:Given a cell array of 8K(8192).

Clock cycle=4
Clock Rate=133MHZ
No of cycles to refresh all rows =8192*4
=32,768
Time needed to refresh all rows=32768/133*10^6
=246*10^{-6} sec
=0.246sec
Refresh Overhead =0.246/64
Refresh Overhead =0.0038

Rambus Memory:

- The usage of wide bus is expensive.
- Rambus developed the implementation of narrow bus.
- Rambus technology is a fast signaling method used to transfer information between chips.

- Instead of using signals that have voltage levels of either 0 or V_{supply} to represent the logical values, the signals consists of much smaller voltage swings around a reference voltage $V_{ref.}$
- .The reference Voltage is about 2V and the two logical values are represented by 0.3V swings above and below $V_{ref..}$
- This type of signaling is generally is known as **Differential Signalling**.
- Rambus provides a complete specification for the design of communication links(**Special Interface circuits**) called as **Rambus Channel**.
- Rambus memory has a clock frequency of 400MHZ.
- The data are transmitted on both the edges of the clock so that the effective data transfer rate is 800MHZ.
- The circuitry needed to interface to the Rambus channel is included on the chip.Such chips are known as Rambus DRAM's(RDRAM).
- Rambus channel has,

  - 9 Data lines(1-8→Transfer the data,9th line→Parity checking).
  - Control line
  - Power line

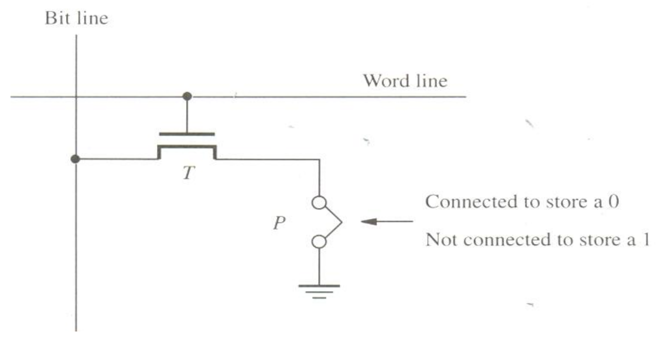
- A two channel rambus has 18 data lines which has no separate address lines.It is also called as **Direct RDRAM's.**
- Communication between processor or some other device that can serves as a master and RDRAM modules are serves as slaves ,is carried out by means of packets transmitted on the data lines.
- There are 3 types of packets.They are,

  - Request
  - Acknowledge
  - Data

<u>READ ONLY MEMORY:</u>

- Both SRAM and DRAM chips are volatile,which means that they lose the stored information if power is turned off.
- Many application requires Non-volatile memory (which retain the stored information if power is turned off).
- Eg:Operating System software has to be loaded from disk to memory which requires the program that boots the Operating System ie. It requires non-volatile memory.
- Non-volatile memory is used in embedded system.
- Since the normal operation involves only reading of stored data ,a memory of this type is called ROM.

Fig:ROM cell



At Logic value '0' → Transistor(T) is connected to the ground point(P).
Transistor switch is closed & voltage on bitline nearly drops to zero.
At Logic value '1' → Transistor switch is open.
The bitline remains at high voltage.

- To read the state of the cell,the word line is activated.
- A Sense circuit at the end of the bitline generates the proper output value.

Types of ROM:

- Different types of non-volatile memory are,
  - PROM
  - EPROM
  - EEPROM
  - Flash Memory

PROM:-Programmable ROM:

- PROM allows the data to be loaded by the user.
- Programmability is achieved by inserting a 'fuse' at point P in a ROM cell.
- Before it is programmed, the memory contains all 0's
- The user can insert 1's at the required location by burning out the fuse at these locations using high-current pulse.
- This process is irreversible.

Merit:

- It provides flexibility.
- It is faster.
- It is less expensive because they can be programmed directly by the user.

EPROM:-Erasable reprogrammable ROM:

- EPROM allows the stored data to be erased and new data to be loaded.

- In an EPROM cell, a connection to ground is always made at ‘P’ and a special transistor is used, which has the ability to function either as a normal transistor or as a disabled transistor that is always turned ‘off’.
- This transistor can be programmed to behave as a permanently open switch, by injecting charge into it that becomes trapped inside.
- Erasure requires dissipating the charges trapped in the transistor of memory cells. This can be done by exposing the chip to ultra-violet light, so that EPROM chips are mounted in packages that have transparent windows.

Merits:

- It provides flexibility during the development phase of digital system.
- It is capable of retaining the stored information for a long time.

Demerits:

- The chip must be physically removed from the circuit for reprogramming and its entire contents are erased by UV light.

EEPROM:-Electrically Erasable ROM:

Merits:

- It can be both programmed and erased electrically.
- It allows the erasing of all cell contents selectively.

Demerits:

- It requires different voltage for erasing ,writing and reading the stored data.

Flash Memory:

- In EEPROM, it is possible to read & write the contents of a single cell.
- In Flash device, it is possible to read the contents of a single cell but it is only possible to write the entire contents of a block.
- Prior to writing,the previous contents of the block are erased.
- Eg.In MP3 player,the flash memory stores the data that represents sound.
- Single flash chips cannot provide sufficient storage capacity for embedded system application.
- There are 2 methods for implementing larger memory modules consisting of number of chips.They are,
  - Flash Cards
  - Flash Drives.

Merits:

- Flash drives have greater density which leads to higher capacity & low cost per bit.
- It requires single power supply voltage & consumes less power in their operation.

Flash Cards:

- One way of constructing larger module is to mount flash chips on a small card.
- Such flash card have standard interface.

- The card is simply plugged into a conveniently accessible slot.
- Its memory size are of 8,32,64MB.
- Eg:A minute of music can be stored in 1MB of memory. Hence 64MB flash cards can store an hour of music.

Flash Drives:

- Larger flash memory module can be developed by replacing the hard disk drive.
- The flash drives are designed to fully emulate the hard disk.
- The flash drives are solid state electronic devices that have no movable parts.

Merits:

- They have shorter seek and access time which results in faster response.
- They have low power consumption which makes them attractive for battery driven application.
- They are insensitive to vibration.

Demerit:

- The capacity of flash drive (<1GB) is less than hard disk(>1GB).
- It leads to higher cost perbit.
- Flash memory will deteriorate after it has been written a number of times(typically atleast 1 million times.)

SPEED,SIZE COST:

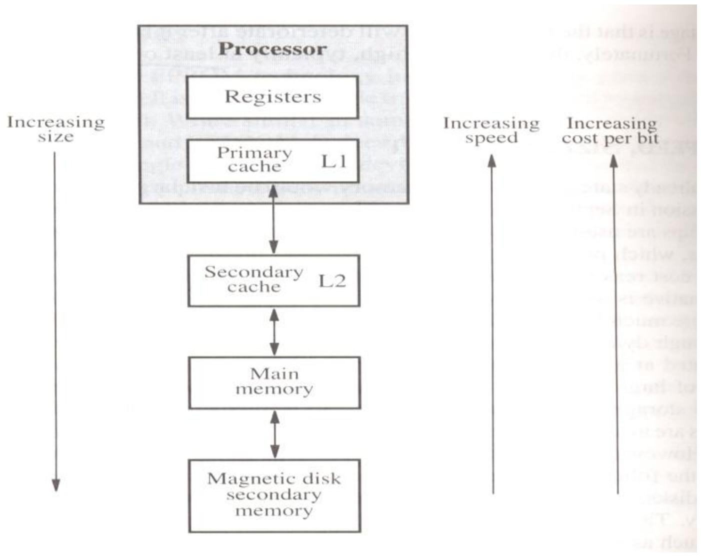
Characteristics	**SRAM**	**DRAM**	**Magnetis Disk**
Speed	Very Fast	Slower	Much slower than DRAM
Size	Large	Small	Small
Cost	Expensive	Less Expensive	Low price

Magnetic Disk:

- A huge amount of cost effective storage can be provided by magnetic disk;The main memory can be built with DRAM which leaves SRAM's to be used in smaller units where speed is of essence.

Memory	**Speed**	**Size**	**Cost**
Registers	Very high	Lower	Very Lower
Primary cache	High	Lower	Low
Secondary cache	Low	Low	Low
Main memory	Lower than Seconadry cache	High	High
Secondary Memory	Very low	Very High	Very High

Fig:Memory Hierarchy



Types of Cache Memory:

- The Cache memory is of 2 types.They are,
    - Primary /Processor Cache(Level1 or L1 cache)
    - Secondary Cache(Level2 or L2 cache)

Primary Cache → It is always located on the processor chip.
Secondary Cache→It is placed between the primary cache and the rest of the memory.

- The main memory is implemented using the dynamic components(**SIMM,RIMM,DIMM**).
- The access time for main memory is about 10 times longer than the access time for L1 cache.

<u>CACHE MEMORIES</u>

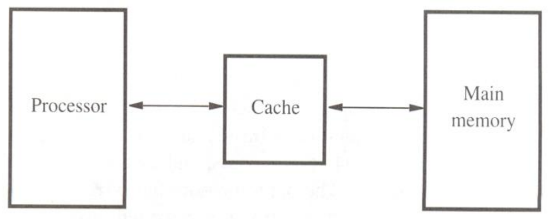
- The effectiveness of cache mechanism is based on the property of '**Locality of reference'.**

Locality of Reference:

- Many instructions in the localized areas of the program are executed repeatedly during some time period and remainder of the program is accessed relatively infrequently.
- It manifests itself in 2 ways.They are,
    - **Temporal**(The recently executed instruction are likely to be executed again very soon.)
    - **Spatial(**The instructions in close proximity to recently executed instruction are also likely to be executed soon.)
- If the active segment of the program is placed in cache memory, then the total execution time can be reduced significantly.

- The term Block refers to the set of contiguous address locations of some size.
- The cache line is used to refer to the cache block.

Fig:Use of Cache Memory



- The Cache memory stores a reasonable number of blocks at a given time but this number is small compared to the total number of blocks available in Main Memory.
- The correspondence between main memory block and the block in cache memory is specified by a mapping function.
- The Cache control hardware decide that which block should be removed to create space for the new block that contains the referenced word.
- The collection of rule for making this decision is called the **replacement algorithm.**
- The cache control circuit determines whether the requested word currently exists in the cache.
- If it exists, then Read/Write operation will take place on appropriate cache location. In this case **Read/Write hit** will occur.
- In a Read operation, the memory will not involve.
- The write operation is proceed in 2 ways.They are,

  - Write-through protocol
  - Write-back protocol

Write-through protocol:

- Here the cache location and the main memory locations are updated simultaneously.

Write-back protocol:

- This technique is to update only the cache location and to mark it as with associated flag bit called **dirty/modified bit.**
- The word in the main memory will be updated later,when the block containing this marked word is to be removed from the cache to make room for a new block.
- If the requested word currently not exists in the cache during read operation,then **read miss** will occur.
- To overcome the read miss **Load –through / Early restart protocol** is used.

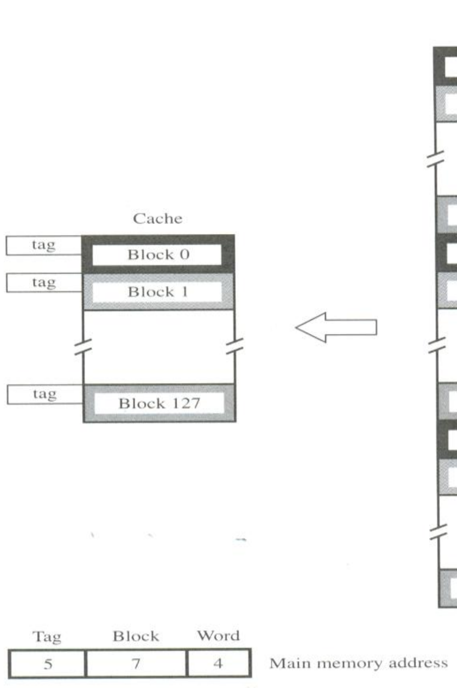
Read Miss:
The block of words that contains the requested word is copied from the main memory into cache.
Load –through:

- After the entire block is loaded into cache,the particular word requested is forwarded to the processor.
- If the requested word not exists in the cache during write operation,then **Write Miss** will occur.
- If Write through protocol is used,the information is written directly into main memory.
- If Write back protocol is used then block containing the addressed word is first brought intothe cache and then the desired word in the cache is over-written with the new information.

Mapping Function:
Direct Mapping:

- It is the simplest technique in which block j of the main memory maps onto block 'j' modulo 128 of the cache.
- Thus whenever one of the main memory blocks 0,128,256 is loaded in the cache,it is stored in block 0.
- Block 1,129,257 are stored in cache block 1 and so on.
- The contention may arise when,
    - ➢ When the cache is full
    - ➢ When more than one memory block is mapped onto a given cache block position.
- The contention is resolved by allowing the new blocks to overwrite the currently resident block.
- Placement of block in the cache is determined from memory address.

Fig: Direct Mapped Cache



- The memory address is divided into 3 fields.They are,

  Low Order 4 bit field(word)→Selects one of 16 words in a block.
  7 bit cache block field→When new block enters cache,7 bit determines the cache position in which this block must be stored.
  5 bit Tag field→The high order 5 bits of the memory address of the block is stored in 5 tag bits associated with its location in the cache.
- As execution proceeds, the high order 5 bits of the address is compared with tag bits associated with that cache location.
- If they match,then the desired word is in that block of the cache.
- If there is no match,then the block containing the required word must be first read from the main memory and loaded into the cache.

Merit:

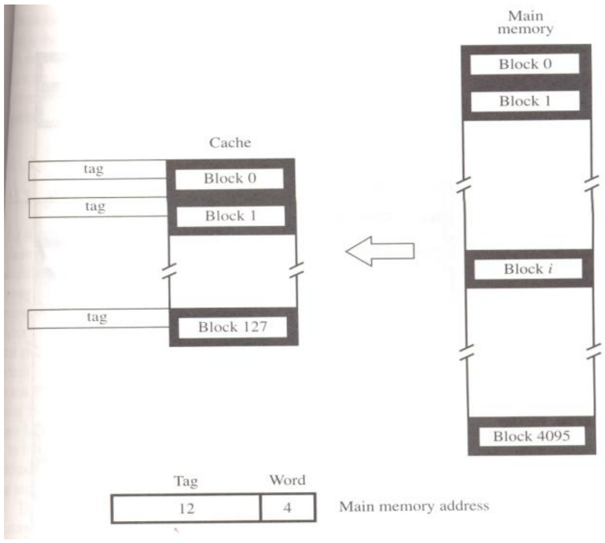
- It is easy to implement.

Demerit:

- It is not very flexible.

Associative Mapping:
In this method, the main memory block can be placed into any cache block position.

Fig:Associative Mapped Cache.



- 12 tag bits will identify a memory block when it is resolved in the cache.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is persent.This is called **associative mapping.**
- **I**t gives complete freedom in choosing the cache location.
- A new block that has to be brought into the cache has to replace(eject)an existing block if the cache is full.

- In this method,the memory has to determine whether a given block is in the cache.
- A search of this kind is called an **associative Search.**

Merit:

- It is more flexible than direct mapping technique.

Demerit:

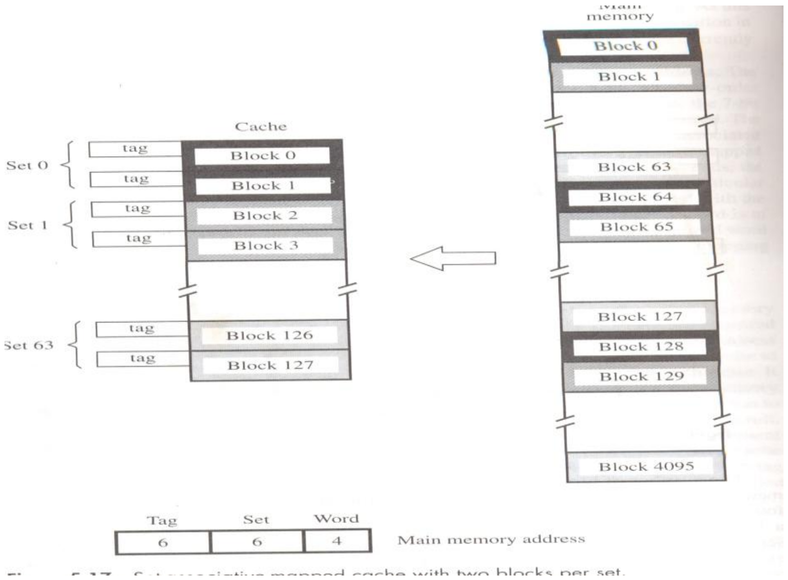
- Its cost is high.

Set-Associative Mapping:

- It is the combination of direct and associative mapping.
- The blocks of the cache are grouped into sets and the mapping allows a block of the main memory to reside in any block of the specified set.
- In this case,the cache has two blocks per set,so the memory blocks 0,64,128……..4032 maps into cache set '0' and they can occupy either of the two block position within the set.
-

6 bit set field→Determines which set of cache contains the desired block .
6 bit tag field→The tag field of the address is compared to the tags of the two blocks of the set to clock if the desired block is present.

Fig: Set-Associative Mapping:



<u>**No of blocks per set**</u>	<u>**no of set field**</u>
2	6
3	5
8	4
128	no set field

- The cache which contains 1 block per set is called **direct Mapping.**
- A cache that has ‘k’ blocks per set is called as ‘**k-way set associative cache**’.
- Each block contains a control bit called a **valid bit**.
- The Valid bit indicates that whether the block contains valid data.
- The dirty bit indicates that whether the block has been modified during its cache residency.

  Valid bit=0→When power is initially applied to system

  Valid bit =1→When the block is loaded from main memory at first time.
- If the main memory block is updated by a source & if the block in the source is already exists in the cache,then the valid bit will be cleared to ‘0’.
- If Processor & DMA uses the same copies of data then it is called as the **Cache Coherence Problem.**

Merit:

- The Contention problem of direct mapping is solved by having few choices for block placement.
- The hardware cost is decreased by reducing the size of associative search.

Replacement Algorithm:

- In direct mapping, the position of each block is pre-determined and there is no need of replacement strategy.
- In associative & set associative method,the block position is not pre-determined;ie..when the cache is full and if new blocks are brought into the cache, then the cache controller must decide which of the old blocks has to be replaced.
- Therefore,when a block is to be over-written,it is sensible to over-write the one that has gone the longest time without being referenced.This block is called **Least recently Used(LRU) block** & the technique is called **LRU algorithm**.
- The cache controller track the references to all blocks with the help of block counter.

Eg:

Consider 4 blocks/set in set associative cache,

- 2 bit counter can be used for each block.
- When a **‘hit’** occurs,then block counter=0;The counter with values originally lower than the referenced one are incremented by 1 & all others remain unchanged.
- When a **‘miss’** occurs & if the set is full,the blocks with the counter value 3 is removed,the new block is put in its place & its counter is set to ‘0’ and other block counters are incremented by 1.

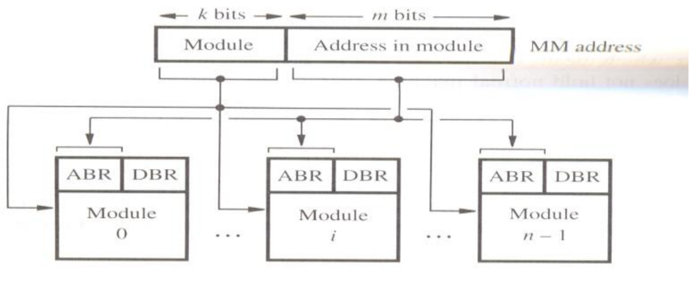
Merit:

- The performance of LRU algorithm is improved by randomness in deciding which block is to be over-written.

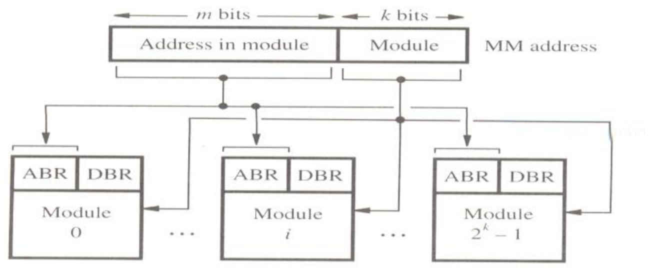
PERFORMANCE CONSIDERATION:

- Two Key factors in the commercial success are the performance & cost ie the best possible performance at low cost.
- A common measure of success is called the **Pricel Performance ratio**.
- Performance depends on how fast the machine instruction are brought to the processor and how fast they are executed.
- To achieve parallelism(ie. Both the slow and fast units are accessed in the same manner),**interleaving** is used.

Interleaving:
Fig:Consecutive words in a Module



(a) Consecutive words in a module



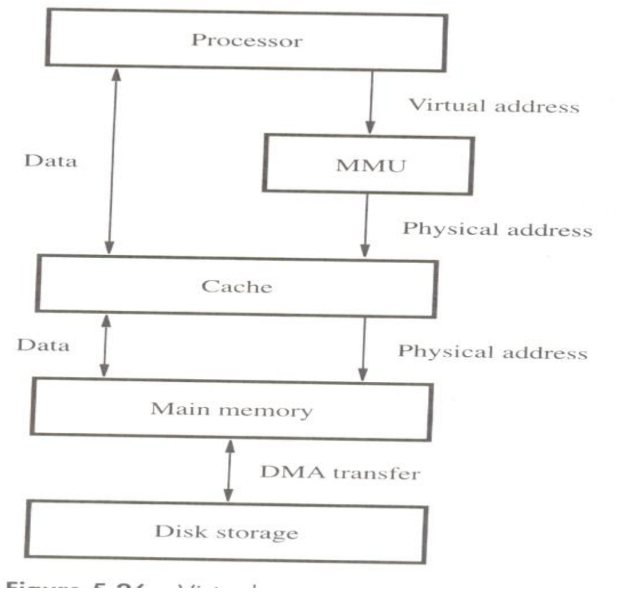
(b) Consecutive words in consecutive modules

VIRTUAL MEMORY:

- Techniques that automatically move program and data blocks into the physical main memory when they are required for execution is called the **Virtual Memory.**
- The binary address that the processor issues either for instruction or data are called the **virtual / Logical address**.
- The virtual address is translated into physical address by a combination of hardware and software components.This kind of address translation is done by **MMU**(Memory Management Unit).
- When the desired data are in the main memory ,these data are fetched /accessed immediately.
- If the data are not in the main memory,the MMU causes the Operating system to bring the data into memory from the disk.

- Transfer of data between disk and main memory is performed using DMA scheme.

Fig:Virtual Memory Organisation



Address Translation:

- In address translation,all programs and data are composed of fixed length units called **Pages.**
- The Page consists of a block of words that occupy contiguous locations in the main memory.
- The pages are commonly range from 2K to 16K bytes in length.
- The **cache bridge** speed up the gap between main memory and secondary storage and it is implemented in software techniques.
- Each virtual address generated by the processor contains **virtual Page number**(Low order bit) and **offset**(High order bit)

Virtual Page number+ Offset→Specifies the location of a particular byte (or word) within a page.

Page Table:

- It contains the information about the main memory address where the page is stored & the current status of the page.

Page Frame:

- An area in the main memory that holds one page is called the page frame.

Page Table Base Register:

- It contains the starting address of the page table.

Virtual Page Number+Page Table Base register→Gives the address of the corresponding entry in the page table.ie)it gives the starting address of the page if that page currently resides in memory.

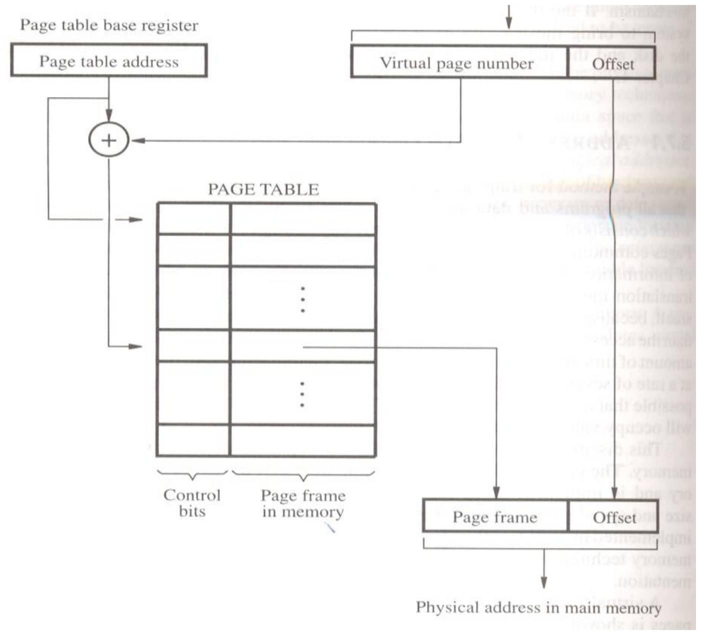
Control Bits in Page Table:

- The Control bits specifies the status of the page while it is in main memory.

Function:

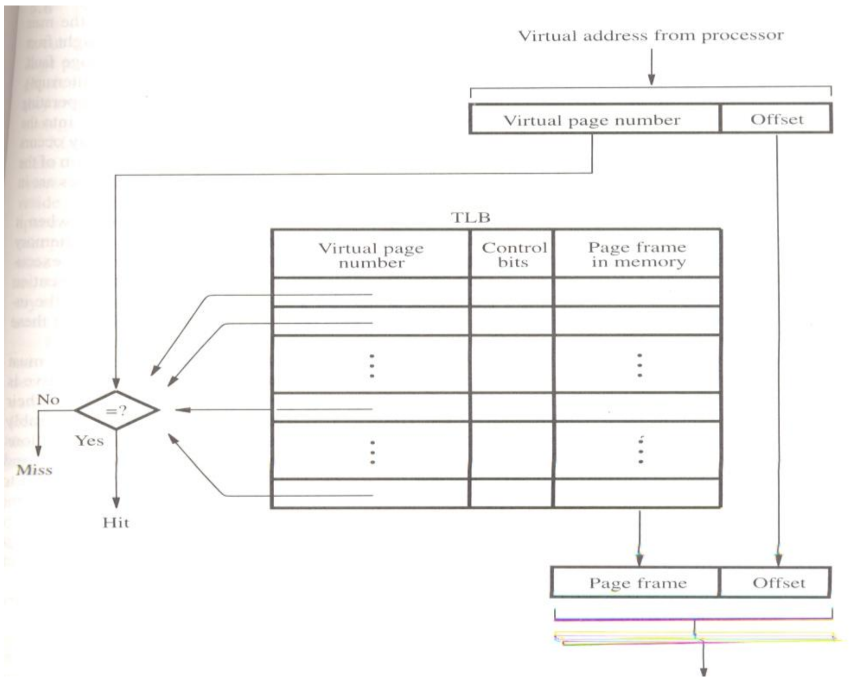
- The control bit indicates the validity of the page ie)it checks whether the page is actually loaded in the main memory.
- It also indicates that whether the page has been modified during its residency in the memory;this information is needed to determine whether the page should be written back to the disk before it is removed from the main memory to make room for another page.

Fig:Virtual Memory Address Translation



- The Page table information is used by MMU for every read & write access.
- The Page table is placed in the main memory but a copy of the small portion of the page table is located within MMU.
- This small portion or small cache is called Translation **LookAside Buffer(TLB).**
- This portion consists of the page table enteries that corresponds to the most recently accessed pages and also contains the virtual address of the entry.

Fig:Use of Associative Mapped TLB



- When the operating system changes the contents of page table ,the control bit in TLB will invalidate the corresponding entry in the TLB.
- Given a virtual address,the MMU looks in TLB for the referenced page.
- If the page table entry for this page is found in TLB,the physical address is obtained immediately.
- If there is a miss in TLB,then the required entry is obtained from the page table in the main memory & TLB is updated.
- When a program generates an access request to a page that is not in the main memory ,then Page Fault will occur.
- The whole page must be broght from disk into memry before an access can proceed.
- When it detects a page fault,the MMU asks the operating system to generate an interrupt.
- The operating System suspend the execution of the task that caused the page fault and begin execution of another task whose pages are in main memory because the long delay occurs while page transfer takes place.
- When the task resumes,either the interrupted instruction must continue from the point of interruption or the instruction must be restarted.
- If a new page is brought from the disk when the main memory is full,it must replace one of the resident pages.In that case,it uses LRU algorithm which removes the least referenced Page.

- A modified page has to be written back to the disk before it is removed from the main memory. In that case,write –through protocol is used.

MEMORY MANAGEMENT REQUIREMENTS:

- Management routines are part of the Operating system.
- Assembling the OS routine into virtual address space is called '**System Space**'.
- The virtual space in which the user application program reside is called the '**User Space'**.
- Each user space has a separate page table.
- The MMU uses the page table to determine the address of the table to be used in the translation process.
- Hence by changing the contents of this register, the OS can switch from one space to another.
- The process has two stages. They are,
    - User State
    - Supervisor state.

User State:

- In this state,the processor executes the user program.

Supervisor State:

- When the processor executes the operating system routines,the processor will be in supervisor state.

Privileged Instruction:

- In user state,the machine instructions cannot be executed.Hence a user program is prevented from accessing the page table of other user spaces or system spaces.
- The control bits in each entry can be set to control the access privileges granted to each program.
- Ie)One program may be allowed to read/write a given page,while the other programs may be given only red access.

SECONDARY STORAGE:

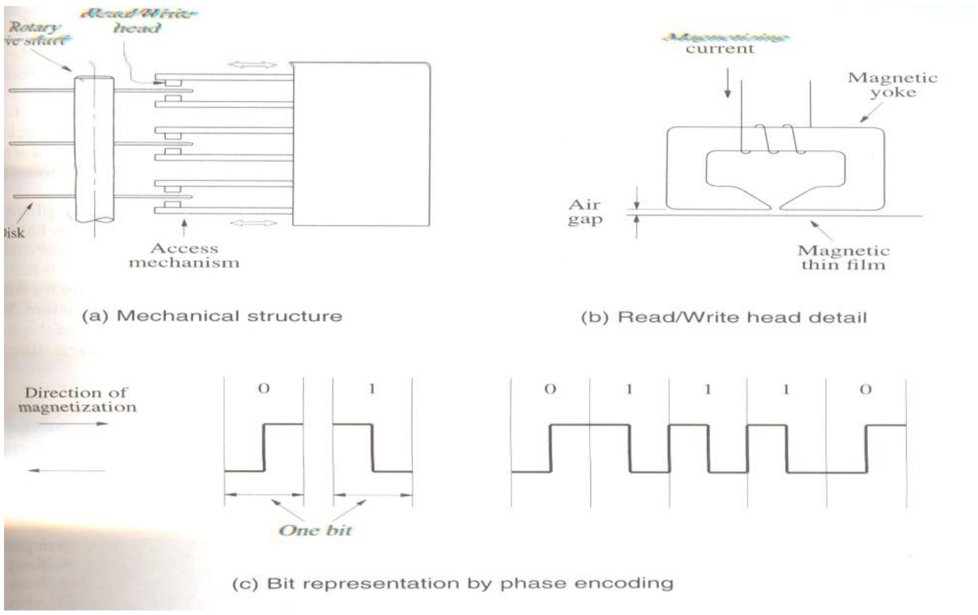
- The Semi-conductor memories donot provide all the storage capability.
- The Secondary storage devices provide larger storage requirements.
- Some of the Secondary Storage devices are,
    - Magnetic Disk
    - Optical Disk
    - Magnetic Tapes.

Magnetic Disk:

- Magnetic Disk system consists o one or more disk mounted on a common spindle.
- A thin magnetic film is deposited on each disk, usually on both sides.

- The disk are placed in a rotary drive so that the magnetized surfaces move in close proximity to read /write heads.
- Each head consists of **magnetic yoke & magnetizing coil**.
- Digital information can be stored on the magnetic film by applying the current pulse of suitable polarity to the magnetizing coil.
- Only changes in the magnetic field under the head can be sensed during the Read operation.
- Therefore if the binary states 0 & 1 are represented by two opposite states of magnetization, a voltage is induced in the head only at 0-1 and at 1-0 transition in the bit stream.
- A consecutive (long string) of 0's & 1's are determined by using the clock which is mainly used for synchronization.
- Phase Encoding or Manchester Encoding is the technique to combine the clocking information with data.
- The Manchester Encoding describes that how the self-clocking scheme is implemented.

Fig:Mechanical Structure



- The Read/Write heads must be maintained at a very small distance from the moving disk surfaces in order to achieve high bit densities.
- When the disk are moving at their steady state, the air pressure develops between the disk surfaces & the head & it forces the head away from the surface.
- The flexible spring connection between head and its arm mounting permits the head to fly at the desired distance away from the surface.

Wanchester Technology:

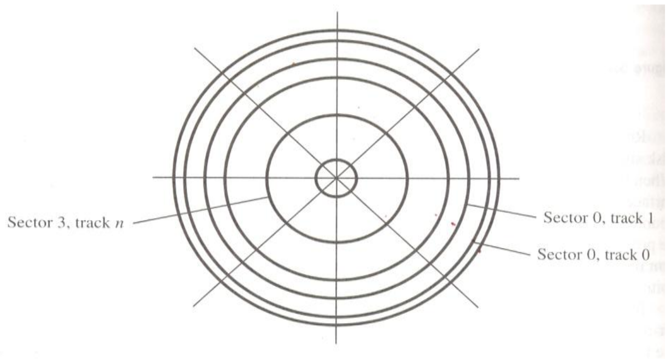
- Read/Write heads are placed in a sealed, air –filtered enclosure called the Wanchester Technology.

- In such units, the read/write heads can operate closure to magnetic track surfaces because the dust particles which are a problem in unsealed assemblies are absent.

Merits:

- It have a larger capacity for a given physical size.
- The data intensity is high because the storage medium is not exposed to contaminating elements.
- The read/write heads of a disk system are movable.
- The disk system has 3 parts.They are,
    - **Disk Platter**(Usually called Disk)
    - **Disk Drive(**spins the disk & moves Read/write heads)
    - **Disk Controller**(controls the operation of the system.)

Fig:Organizing & Accessing the data on disk



- Each surface is divided into concentric **tracks**.
- Each track is divided into **sectors**.
- The set of corresponding tracks on all surfaces of a stack of disk form a **logical cylinder.**
- The data are accessed by specifying **the surface number,track number and the sector number.**
- The Read/Write operation start at sector boundaries.
- Data bits are stored serially on each track.
- Each sector usually contains 512 bytes.

Sector header -> contains identification information.
It helps to find the desired sector on the selected track.
ECC (Error checking code)- used to detect and correct errors.

- An unformatted disk has no information on its tracks.
- The formatting process divides the disk physically into tracks and sectors and this process may discover some defective sectors on all tracks.
- The disk controller keeps a record of such defects.

- The disk is divided into logical partitions. They are,
    - ➢ Primary partition
    - ➢ Secondary partition
- In the diag, Each track has same number of sectors.
- So all tracks have same storage capacity.
- Thus the stored information is packed more densely on inner track than on outer track.

Access time

- There are 2 components involved in the time delay between receiving an address and the beginning of the actual data transfer. They are,
    - ➢ Seek time
    - ➢ Rotational delay / Latency

Seek time – Time required to move the read/write head to the proper track.
Latency – The amount of time that elapses after the head is positioned over the correct track until the starting position of the addressed sector passes under the read/write head.

Seek time + Latency = Disk access time

Typical disk

One inch disk- weight=1 ounce, size -> comparable to match book
Capacity -> 1GB
Inch disk has the following parameter
Recording surface=20
Tracks=15000 tracks/surface
Sectors=400.
Each sector stores 512 bytes of data
Capacity of formatted disk=20x15000x400x512=60x10^9 =60GB
Seek time=3ms
Platter rotation=10000 rev/min
Latency=3ms
Internet transfer rate=34MB/s

Data Buffer / cache

- A disk drive that incorporates the required SCSI circuit is referred as SCSI drive.
- The SCSI can transfer data at higher rate than the disk tracks.
- An efficient method to deal with the possible difference in transfer rate between disk and SCSI bus is accomplished by including a data buffer.
- This buffer is a semiconductor memory.
- The data buffer can also provide cache mechanism for the disk (ie) when a read request arrives at the disk, then controller first check if the data is available in the cache(buffer).
- If the data is available in the cache, it can be accessed and placed on SCSI bus . If it is not available then the data will be retrieved from the disk.

Disk Controller

- The disk controller acts as interface between disk drive and system bus.
- The disk controller uses DMA scheme to transfer data between disk and main memory.

- When the OS initiates the transfer by issuing Read/Write request, the controllers register will load the following information. They are,
- Main memory address(address of first main memory location of the block of words involved in the transfer)
- Disk address(The location of the sector containing the beginning of the desired block of words)

(number of words in the block to be transferred).

Sector header -> contains identification information.
It helps to find the desired sector on the selected track.
ECC (Error checking code)- used to detect and correct errors.
An unformatted disk has no information on its tracks.
The formatting process divides the disk physically into tracks and sectors and this process may discover some defective sectors on all tracks.
The disk controller keeps a record of such defects.
The disk is divided into logical partitions. They are,
Primary partition
Secondary partition
In the diag, Each track has same number of sectors.
So all tracks have same storage capacity.
Thus the stored information is packed more densely on inner track than on outer track.

Access time

There are 2 components involved in the time delay between receiving an address and the beginning of the actual data transfer. They are,
Seek time
Rotational delay / Latency
Seek time – Time required to move the read/write head to the proper track.
Latency – The amount of time that elapses after the head is positioned over the correct track until the starting position of the addressed sector passes under the read/write head.
Seek time + Latency = Disk access time

Typical disk

One inch disk- weight=1 ounce, size -> comparable to match book
Capacity -> 1GB
3.5 inch disk has the following parameter
Recording surface=20
Tracks=15000 tracks/surface
Sectors=400.
Each sector stores 512 bytes of data
Capacity of formatted disk=20x15000x400x512=60x10^9 =60GB
Seek time=3ms
Platter rotation=10000 rev/min
Latency=3ms
Internet transfer rate=34MB/s

Data Buffer / cache

A disk drive that incorporates the required SCSI circuit is referred as SCSI drive.

The SCSI can transfer data at higher rate than the disk tracks.
An efficient method to deal with the possible difference in transfer rate between disk and SCSI bus is accomplished by including a data buffer.
This buffer is a semiconductor memory.
The data buffer can also provide cache mechanism for the disk (ie) when a read request arrives at the disk, then controller first check if the data is available in the cache(buffer). If the data is available in the cache, it can be accessed and placed on SCSI bus . If it is not available then the data will be retrieved from the disk.

Disk Controller

The disk controller acts as interface between disk drive and system bus.
The disk controller uses DMA scheme to transfer data between disk and main memory.
When the OS initiates the transfer by issuing Read/Write request, the controllers register will load the following information. They are,
Main memory address(address of first main memory location of the block of words involved in the transfer)
Disk address(The location of the sector containing the beginning of the desired block of words)
(number of words in the block to be transferred).

UNIT V
MEMORY SYSTEM

PART-A

1) Write about the functions of memory organization?
2) Draw the block diagram for memory unit?
3) What re the key characteristics of memories?
4) What are the types of access method?
5) What is meant by memory cycle time?
6) Define transfer rate
7) What are the memory operations?
8) Define throughput
9) What is memory bus?
10) What are the types of memory?
11) Write about RAM
12) What is static RAM?
13) Write short notes on ROM
14) What is error detection and correction code?
15) Explain parity bit?
16) What is hamming error-correcting codes?
17) What is cache memory?
18) What is fully associative mapping?
19) What is block replacement?
20) What is meant by LRU?
21) Explain write strategy?
22) Define bit ratio

23) Explain cache update policies?
24) What is direct mapped cache?
25) What is associative mapping?
26) Explain cache organization?
27) What is cache miss rate?
28) Write about cache directories?
29) What is a write-protect bit?
30) What is multilevel cache?
31) What is virtual memory?
32) What is segmented memory?
33) Write about block access time?
34) What is page mode?
35) Write about I/O-address strobe (RA)?
36) What is column-address strobe (CA)?
37) What is access time myth?
38) How can hits or misses myth?
39) How are memory blocks mapped into cache lines?
40) What is secondary storages?

PART – B

1) Explain the memory organization?
2) Write about the following
   a. Memory bus
   b. Byte storage methods
3) Write about memory hierarchy
4) Explain semi conductor RAM?
5) Write about semi conductor ROM?
6) Explain error detection and correction codes?

UNIT V

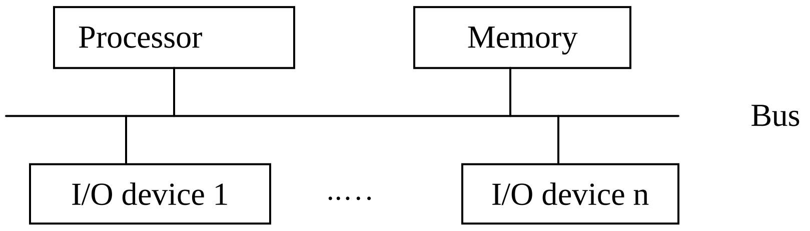
I/O ORGANIZATION

- Accessing I/O devices
- Programmed I/O
- Interrupts
- Direct memory access
- Buses
- Interface Circuits
- Standard I/O interfaces (PCI, SCSI, and USB)
- I/O Devices and processors

ACCESSING I/O DEVICES

- A simple arrangement to connect I/O devices to a computer is to use a single bus structure. It consists of three sets of lines to carry
  - ❖ Address
  - ❖ Data
  - ❖ Control Signals.
- When the processor places a particular address on address lines, the devices that recognize this address responds to the command issued on the control lines.
- The processor request either a read or write operation and the requested data are transferred over the data lines.
- When I/O devices & memory share the same address space, the arrangement is called **memory mapped I/O**.

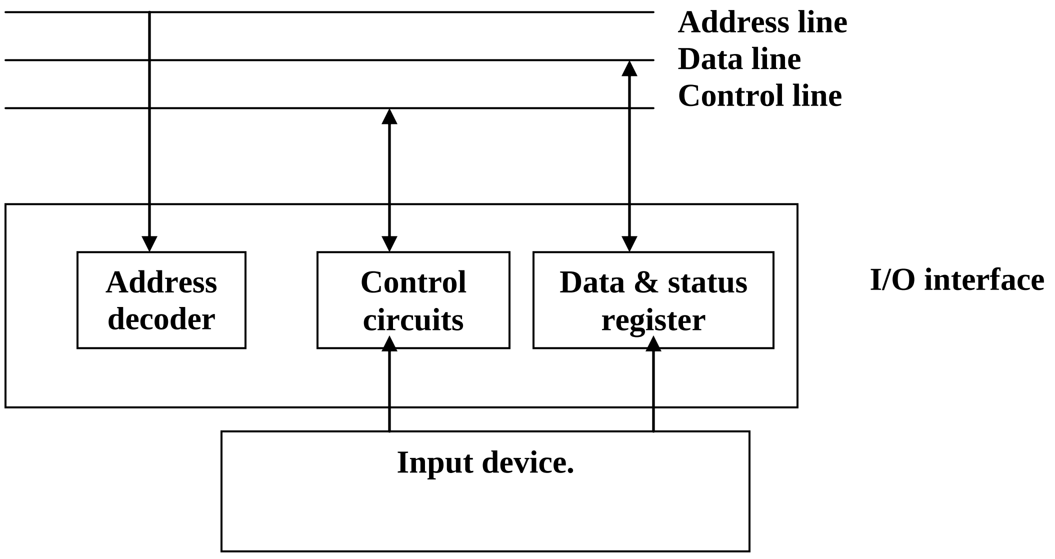
Single Bus Structure



Eg:-

Move DATAIN, R_o → Reads the data from DATAIN then into processor register $R_{o.}$
Move R_o, DATAOUT → Send the contents of register Ro to location DATAOUT.
DATAIN → Input buffer associated with keyboard.
DATAOUT → Output data buffer of a display unit / printer.

Fig: I/O Interface for an Input Device



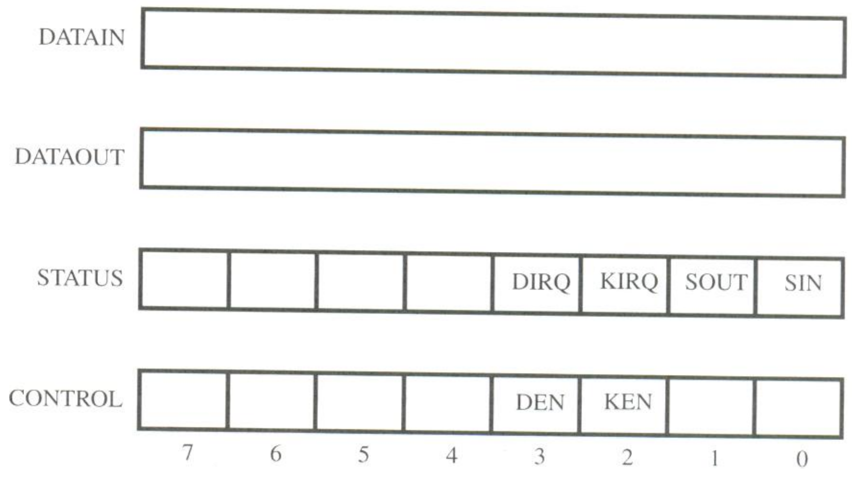
Address Decoder:

- It enables the device to recognize its address when the address appears on address lines.

Data register → It holds the data being transferred to or from the processor.
Status register → It contains infn/. Relevant to the operation of the I/O devices.

- The address decoder, data & status registers and the control circuitry required to co-ordinate I/O transfers constitute the device's I/F circuit.
- For an input device, SIN status flag in used SIN = 1, when a character is entered at the keyboard.
- For an output device, SOUT status flag is used SIN = 0, once the char is read by processor.

Eg



DIR Q → Interrupt Request for display.
KIR Q →Interrupt Request for keyboard.
KEN → keyboard enable.
DEN → Display Enable.
SIN, SOUT → status flags.

The data from the keyboard are made available in the DATAIN register & the data sent to the display are stored in DATAOUT register.

Program:

```
WAIT K    Move          # Line, Ro
          Test Bit      #0, STATUS
          Branch = 0    WAIT K
          Move          DATAIN, R1
WAIT D    Test Bit      #1, STATUS
          Branch = 0    WAIT D
          Move          R1, DATAOUT
          Move          R1, (Ro)+
          Compare       #OD, R1
          Branch = 0    WAIT K
          Move          #DOA, DATAOUT
          Call          PROCESS
```

EXPLANATION:

- This program, reads a line of characters from the keyboard & stores it in a memory buffer starting at locations LINE.
- Then it calls the subroutine "PROCESS" to process the input line.
- As each character is read, it is echoed back to the display.
- Register Ro is used as a updated using Auto – increment mode so that successive characters are stored in successive memory location.
- Each character is checked to see if there is carriage return (CR), char, which has the ASCII code 0D(hex).
- If it is, a line feed character (on) is sent to more the cursor one line down on the display & subroutine PROCESS is called. Otherwise, the program loops back to wait for another character from the keyboard.

PROGRAM CONTROLLED I/O

Here the processor repeatedly checks a status flag to achieve the required synchronization between Processor & I/O device.(ie) the processor polls the device.

There are 2 mechanisms to handle I/o operations. They are,

➔ Interrupt, -
➔ DMA (Synchronization is achieved by having I/O device send special over the bus where is ready for data transfer operation)

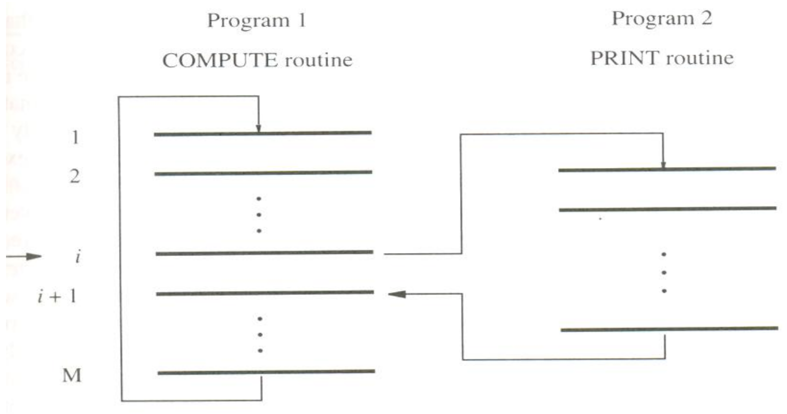
DMA:

- Synchronization is achieved by having I/O device send special over the bus where is ready for data transfer operation)
- It is a technique used for high speed I/O device.
- Here, the input device transfer data directly to or from the memory without continuous involvement by the processor.

INTERRUPTS

- When a program enters a wait loop, it will repeatedly check the device status. During this period, the processor will not perform any function.
- The Interrupt request line will send a hardware signal called the interrupt signal to the processor.
- On receiving this signal, the processor will perform the useful function during the waiting period.
- The routine executed in response to an interrupt request is called **Interrupt Service Routine.**
- The interrupt resembles the subroutine calls.

Fig:Transfer of control through the use of interrupts

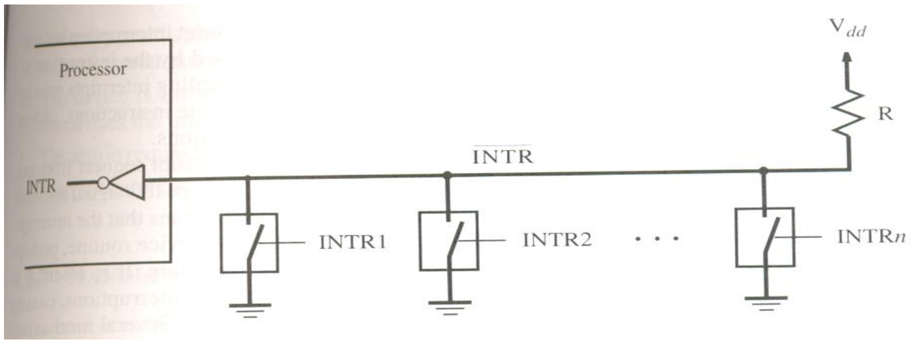


- The processor first completes the execution of instruction i Then it loads the PC(Program Counter) with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction i + 1.
- Therefore, when an interrupt occurs, the current contents of PC which point to i +1 is put in temporary storage in a known location.
- A return from interrupt instruction at the end of ISR reloads the PC from that temporary storage location, causing the execution to resume at instruction i+1.
- When the processor is handling the interrupts, it must inform the device that its request has been recognized so that it remove its interrupt requests signal.
- This may be accomplished by a special control signal called the **interrupt acknowledge signal.**
- The task of saving and restoring the information can be done automatically by the processor.
- The processor saves only the contents of **program counter & status register** (ie) it saves only the minimal amount of information to maintain the integrity of the program execution.

- Saving registers also increases the delay between the time an interrupt request is received and the start of the execution of the ISR. This delay is called the **Interrupt Latency.**
- Generally, the long interrupt latency in unacceptable.
- The concept of interrupts is used in Operating System and in Control Applications, where processing of certain routines must be accurately timed relative to external events. This application is also called as **real-time processing**.

Interrupt Hardware:

Fig:An equivalent circuit for an open drain bus used to implement a common interrupt request line



- A single interrupt request line may be used to serve 'n' devices. All devices are connected to the line via switches to ground.
- To request an interrupt, a device closes its associated switch, the voltage on INTR line drops to 0(zero).
- If all the interrupt request signals (INTR1 to INTRn) are inactive, all switches are open and the voltage on INTR line is equal to Vdd.
- When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices.

(ie) **INTR = INTR1+…………+INTRn**

$\overline{\text{INTR}}$ → It is used to name the INTR signal on common line it is active in the low voltage state.

- **Open collector** (bipolar ckt) or **Open drain** (MOS circuits) is used to drive $\overline{\text{INTR}}$ line.
- The Output of the Open collector (or) Open drain control is equal to a switch to the ground that is open when gates input is in '0' state and closed when the gates input is in '1' state.
- Resistor 'R' is called a **pull-up resistor** because it pulls the line voltage upto the high voltage state when the switches are open.

Enabling and Disabling Interrupts:

- The arrival of an interrupt request from an external device causes the processor to suspend the execution of one program & start the execution of another because the interrupt may alter the sequence of events to be executed.
- INTR is active during the execution of **Interrupt Service Routine**.
- There are 3 mechanisms to solve the problem of infinite loop which occurs due to successive interruptions of active INTR signals.
- The following are the typical scenario.

  - ➔ The device raises an interrupt request.
  - ➔ The processor interrupts the program currently being executed.
  - ➔ Interrupts are disabled by changing the control bits is PS (Processor Status register)
  - ➔ The device is informed that its request has been recognized & in response, it deactivates the INTR signal.
  - ➔ The actions are enabled & execution of the interrupted program is resumed.

Edge-triggered:

The processor has a special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line said to be edge-triggered.

Handling Multiple Devices:

- When several devices requests interrupt at the same time, it raises some questions. They are.

  - ➢ How can the processor recognize the device requesting an interrupt?
  - ➢ Given that the different devices are likely to require different ISR, how can the processor obtain the starting address of the appropriate routines in each case?
  - ➢ Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  - ➢ How should two or more simultaneous interrupt requests be handled?

Polling Scheme:

- If two devices have activated the interrupt request line, the ISR for the selected device (first device) will be completed & then the second request can be serviced.
- The simplest way to identify the interrupting device is to have the ISR polls all the encountered with the IRQ bit set is the device to be serviced
- IRQ (Interrupt Request) -> when a device raises an interrupt requests, the status register IRQ is set to 1.

Merit:

It is easy to implement.

Demerit:

The time spent for interrogating the IRQ bits of all the devices that may not be requesting any service.

Vectored Interrupt:

- Here the device requesting an interrupt may identify itself to the processor by sending a special code over the bus & then the processor start executing the ISR.
- The code supplied by the processor indicates the starting address of the ISR for the device.
- The code length ranges from 4 to 8 bits.
- The location pointed to by the interrupting device is used to store the staring address to ISR.
- The processor reads this address, called the interrupt vector & loads into PC.
- The interrupt vector also includes a new value for the Processor Status Register.
- When the processor is ready to receive the interrupt vector code, it activate the interrupt acknowledge (INTA) line.

Interrupt Nesting:
Multiple Priority Scheme:

- In multiple level priority scheme, we assign a priority level to the processor that can be changed under program control.
- The priority level of the processor is the priority of the program that is currently being executed.
- The processor accepts interrupts only from devices that have priorities higher than its own.
- At the time the execution of an ISR for some device is started, the priority of the processor is raised to that of the device.
- The action disables interrupts from devices at the same level of priority or lower.

Privileged Instruction:

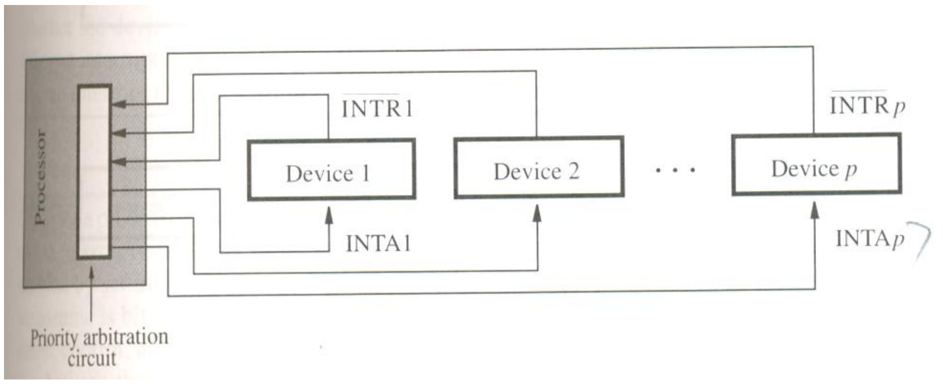
- The processor priority is usually encoded in a few bits of the Processor Status word. It can also be changed by program instruction & then it is write into PS. These instructions are called **privileged instruction.** This can be executed only when the processor is in supervisor mode.
- The processor is in supervisor mode only when executing OS routines.
- It switches to the user mode before beginning to execute application program.

Privileged Exception:

- User program cannot accidently or intentionally change the priority of the processor & disrupts the system operation.

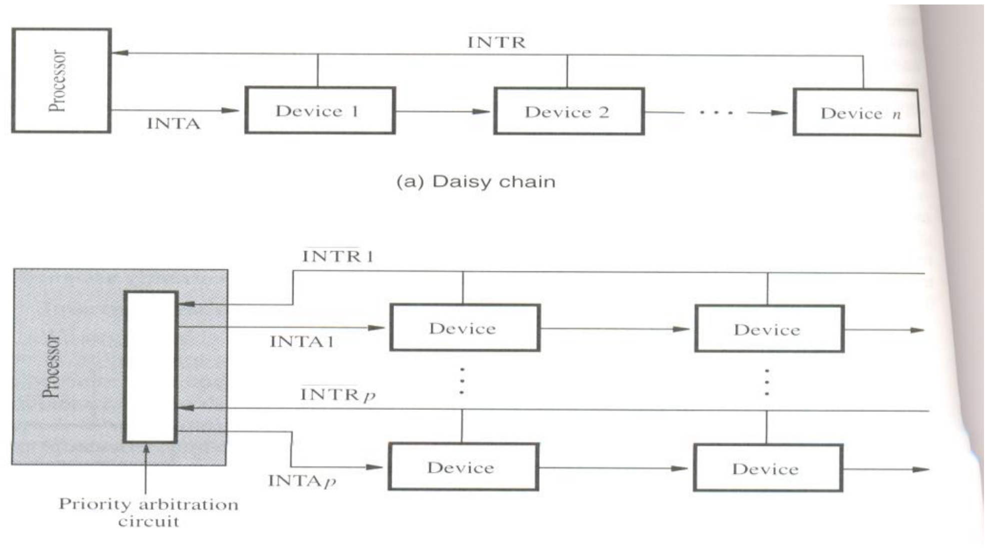
- An attempt to execute a privileged instruction while in user mode, leads to a special type of interrupt called the privileged exception.

Fig: Implementation of Interrupt Priority using individual Interrupt request acknowledge lines


Processor
$\overline{\text{INTR}}1$
$\overline{\text{INTR}}p$
Device 1
Device 2
Device p
INTA1
INTAp
Priority arbitration circuit

- Each of the interrupt request line is assigned a different priority level.
- Interrupt request received over these lines are sent to a priority arbitration circuit in the processor.
- A request is accepted only if it has a higher priority level than that currently assigned to the processor,

Simultaneous Requests:
Daisy Chain:


Processor
INTR
INTA
Device 1
Device 2
Device n
(a) Daisy chain
$\overline{\text{INTR}}1$
INTA1
$\overline{\text{INTR}}p$
INTAp
Device
Priority arbitration circuit

(b) Arrangement of priority groups

- The interrupt request line INTR is common to all devices. The interrupt acknowledge line INTA is connected in a daisy chain fashion such that INTA signal propagates serially through the devices.
- When several devices raise an interrupt request, the INTR is activated & the processor responds by setting INTA line to 1. this signal is received by device.

- Device1 passes the signal on to device2 only if it does not require any service.
- If devices1 has a pending request for interrupt blocks that INTA signal & proceeds to put its identification code on the data lines.
- Therefore, the device that is electrically closest to the processor has the highest priority.

Merits:

It requires fewer wires than the individual connections.

Arrangement of Priority Groups:

- Here the devices are organized in groups & each group is connected at a different priority level.
- Within a group, devices are connected in a daisy chain.

Controlling Device Requests:

KEN	→	Keyboard Interrupt Enable
DEN	→	Display Interrupt Enable
KIRQ / DIRQ	→	Keyboard / Display unit requesting an interrupt.

- There are two mechanism for controlling interrupt requests.
- At the devices end, an interrupt enable bit in a control register determines whether the device is allowed to generate an interrupt requests.
- At the processor end, either an interrupt enable bit in the PS (Processor Status) or a priority structure determines whether a given interrupt requests will be accepted.

Initiating the Interrupt Process:

- ➢ Load the starting address of ISR in location INTVEC (vectored interrupt).
- ➢ Load the address LINE in a memory location PNTR. The ISR will use this location as a pointer to store the i/p characters in the memory.
- ➢ Enable the keyboard interrupts by setting bit 2 in register CONTROL to 1.
- ➢ Enable interrupts in the processor by setting to 1, the IE bit in the processor status register PS.

Exception of ISR:

- ➢ Read the input characters from the keyboard input data register. This will cause the interface circuits to remove its interrupt requests.
- ➢ Store the characters in a memory location pointed to by PNTR & increment PNTR.
- ➢ When the end of line is reached, disable keyboard interrupt & inform program main.
- ➢ Return from interrupt.

Exceptions:

- An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin.
- The Exception is used to refer to any event that causes an interruption.

Kinds of exception:

- ❖ Recovery from errors
- ❖ Debugging
- ❖ Privileged Exception

Recovery From Errors:

- Computers have error-checking code in Main Memory , which allows detection of errors in the stored data.
- If an error occurs, the control hardware detects it informs the processor by raising an interrupt.
- The processor also interrupts the program, if it detects an error or an unusual condition while executing the instance (ie) it suspends the program being executed and starts an execution service routine.
- This routine takes appropriate action to recover from the error.

Debugging:

- System software has a program called debugger, which helps to find errors in a program.
- The debugger uses exceptions to provide two important facilities
- They are
    - ❖ Trace
    - ❖ Breakpoint

Trace Mode:

- When processor is in trace mode , an exception occurs after execution of every instance using the debugging program as the exception service routine.
- The debugging program examine the contents of registers, memory location etc.
- On return from the debugging program the next instance in the program being debugged is executed
- The trace exception is disabled during the execution of the debugging program.

Break point:

- Here the program being debugged is interrupted only at specific points selected by the user.

- An instance called the Trap (or) software interrupt is usually provided for this purpose.
- While debugging the user may interrupt the program execution after instance 'I'
- When the program is executed and reaches that point it examine the memory and register contents.

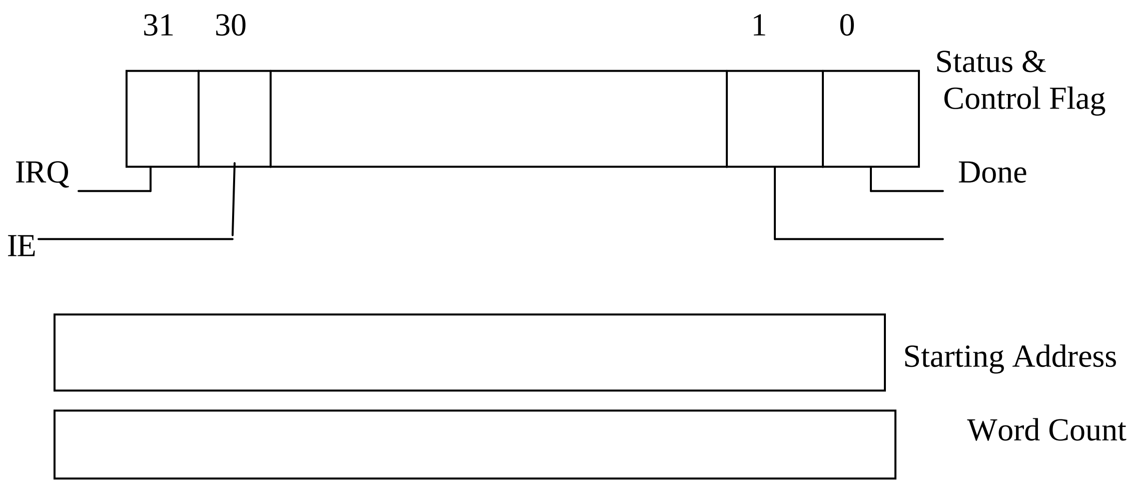
Privileged Exception:

- To protect the OS of a computer from being corrupted by user program certain instance can be executed only when the processor is in supervisor mode. These are called privileged exceptions.
- When the processor is in user mode, it will not execute instance (ie) when the processor is in supervisor mode , it will execute instance.

DIRECT MEMORY ACCESS

- A special control unit may be provided to allow the transfer of large block of data at high speed directly between the external device and main memory , without continous intervention by the processor. This approach is called **DMA.**
- DMA transfers are performed by a control circuit called the **DMA Controller**.
- To initiate the transfer of a block of words , the processor sends,
    - Starting address
    - Number of words in the block
    - Direction of transfer.
- When a block of data is transferred , the DMA controller increment the memory address for successive words and keep track of number of words and it also informs the processor by raising an interrupt signal.
- While DMA control is taking place, the program requested the transfer cannot continue and the processor can be used to execute another program.
- After DMA transfer is completed, the processor returns to the program that requested the transfer.

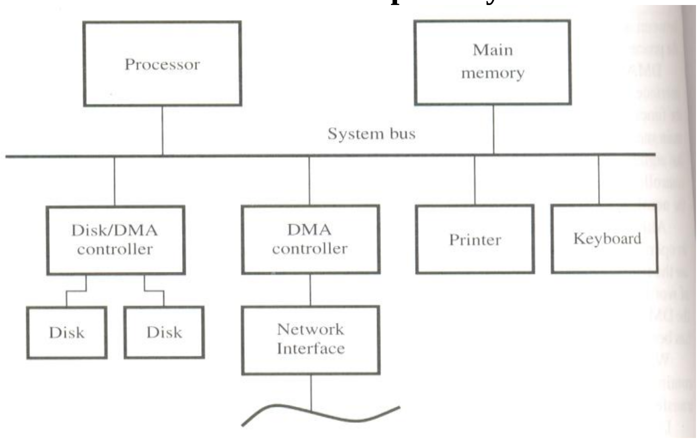
Fig:Registes in a DMA Interface



R/W → Determines the direction of transfer .
When

R/W =1, DMA controller read data from memory to I/O device.
R/W =0, DMA controller perform write operation.
Done Flag=1, the controller has completed transferring a block of data and is ready to receive another command.
IE=1, it causes the controller to raise an interrupt (interrupt Enabled) after it has completed transferring the block of data.
IRQ=1, it indicates that the controller has requested an interrupt.

Fig: Use of DMA controllers in a computer system



- A DMA controller connects a high speed network to the computer bus . The disk controller two disks, also has DMA capability and it provides two DMA channels.
- To start a DMA transfer of a block of data from main memory to one of the disks, the program write s the address and the word count inf. Into the registers of the corresponding channel of the disk controller.
- When DMA transfer is completed, it will be recorded in status and control registers of the DMA channel (ie) **Done bit=IRQ=IE=1**.

Cycle Stealing:

- Requests by DMA devices for using the bus are having higher priority than processor requests .
- Top priority is given to high speed peripherals such as ,
  - Disk
  - High speed Network Interface and Graphics display device.

- Since the processor originates most memory access cycles, the DMA controller can be said to steal the memory cycles from the processor.
- This interviewing technique is called **Cycle stealing.**

Burst Mode:

The DMA controller may be given exclusive access to the main memory to transfer a block of data without interruption. This is known as **Burst/Block Mode**

Bus Master:

The device that is allowed to initiate data transfers on the bus at any given time is called the bus master.

Bus Arbitration:

It is the process by which the next device to become the bus master is selected and the bus mastership is transferred to it.

Types:

There are 2 approaches to bus arbitration. They are,

- Centralized arbitration (A single bus arbiter performs arbitration)
- Distributed arbitration (all devices participate in the selection of next bus master).

Centralized Arbitration:

- Here the processor is the bus master and it may grants bus mastership to one of its DMA controller.
- A DMA controller indicates that it needs to become the bus master by activating the Bus Request line (BR) which is an open drain line.
- The signal on BR is the logical OR of the bus request from all devices connected to it.
- When BR is activated the processor activates the Bus Grant Signal (BGI) and indicated the DMA controller that they may use the bus when it becomes free.
- This signal is connected to all devices using a daisy chain arrangement.
- If DMA requests the bus, it blocks the propagation of Grant Signal to other devices and it indicates to all devices that it is using the bus by activating open collector line, Bus Busy (BBSY).

Fig:A simple arrangement for bus arbitration using a daisy chain

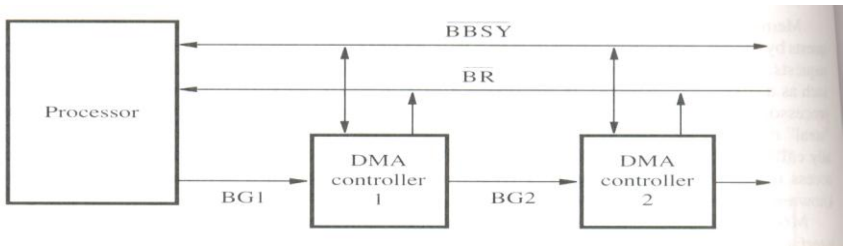
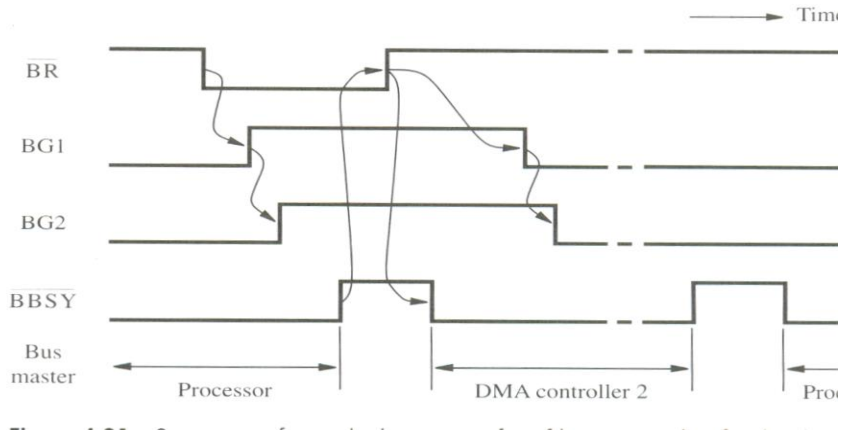


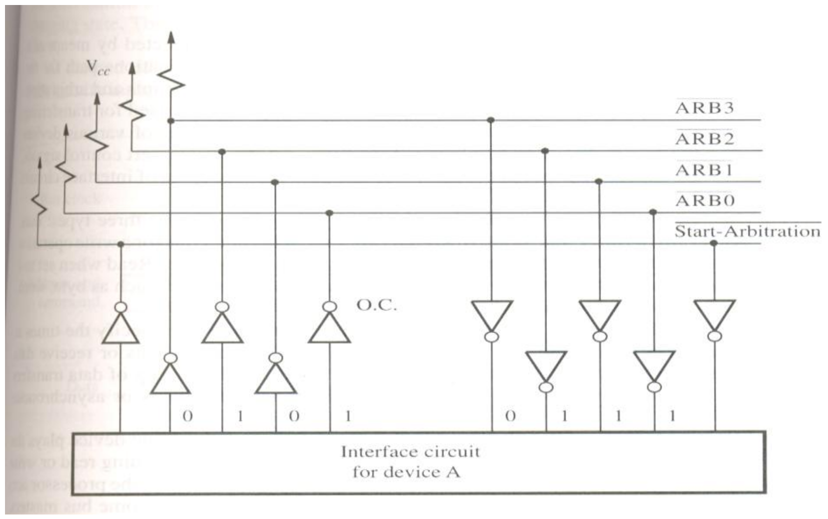
Fig: Sequence of signals during transfer of bus mastership for the devices



- The timing diagram shows the sequence of events for the devices connected to the processor is shown.
- DMA controller 2 requests and acquires bus mastership and later releases the bus.
- During its tenture as bus master, it may perform one or more data transfer.
- After it releases the bus, the processor resources bus mastership

Distributed Arbitration:
It means that all devices waiting to use the bus have equal responsibility in carrying out the arbitration process.

Fig:A distributed arbitration scheme



- Each device on the bus is assigned a 4 bit id.
- When one or more devices request the bus, they assert the Start-Arbitration signal & place their 4 bit ID number on four open collector lines, ARB0 to ARB3.
- A winner is selected as a result of the interaction among the signals transmitted over these lines.

- The net outcome is that the code on the four lines represents the request that has the highest ID number.
- The drivers are of open collector type. Hence, if the i/p to one driver is equal to 1, the i/p to another driver connected to the same bus line is equal to ‘0’(ie. bus the is in low-voltage state).

Eg:

- Assume two devices A & B have their ID 5 (0101), 6(0110) and their code is 0111.
- Each devices compares the pattern on the arbitration line to its own ID starting from MSB.
- If it detects a difference at any bit position, it disables the drivers at that bit position. It does this by placing ‘0’ at the i/p of these drivers.
- In our eg. ‘A’ detects a difference in line ARB1, hence it disables the drivers on lines ARB1 & ARB0.
- This causes the pattern on the arbitration line to change to 0110 which means that ‘B’ has won the contention.

Buses

- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus ie, when to place information in the bus, assert control signals etc.
- The bus lines used for transferring data is grouped into 3 types. They are,
  - ➢ Address line
  - ➢ Data line
  - ➢ Control line.

Control signals→Specifies that whether read / write operation has to performed.
It also carries timing infn/. (ie) they specify the time at which the processor & I/O devices place the data on the bus & receive the data from the bus.

- During data transfer operation, one device plays the role of a ‘**Master**’.
- **Master** device initiates the data transfer by issuing read / write command on the bus. Hence it is also called as ‘**Initiator**’.
- The device addressed by the master is called as **Slave / Target**.

Types of Buses:

There are 2 types of buses. They are,

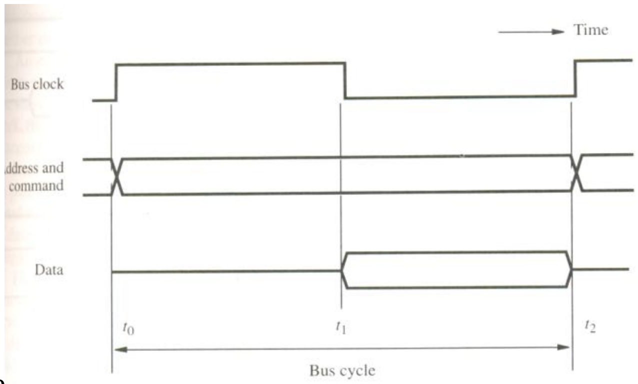
- ➢ Synchronous Bus
- ➢ Asynchronous Bus.

Synchronous Bus:-

- In synchronous bus, all devices derive timing information from a common clock line.
- Equally spaced pulses on this line define equal time.

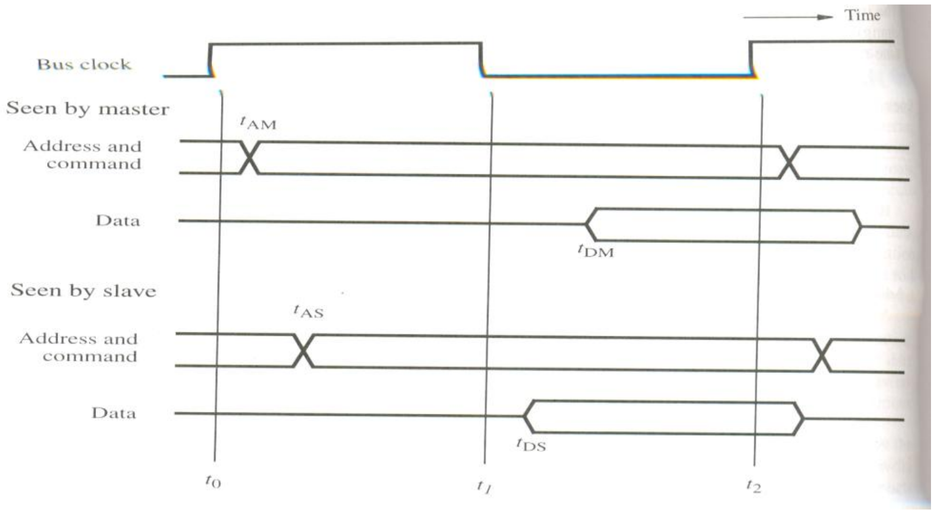
- During a ‘**bus cycle**’, one data transfer on take place.
- The ‘**crossing points**’ indicate the tone at which the patterns change.
- A ‘**signal line’** in an indeterminate / high impedance state is represented by an intermediate half way between the low to high signal levels.

Fig:Timing of an input transfer of a synchronous bus.



- At time t0, the master places the device address on the address lines & sends an appropriate command on the control lines.
- In this case, the command will indicate an input operation & specify the length of the operand to be read.
- The clock pulse width t1 – t0 must be longer than the maximum delay between devices connected to the bus.
- The clock pulse width should be long to allow the devices to decode the address & control signals so that the addressed device can respond at time t_1.
- The slaves take no action or place any data on the bus before t_1.

Fig:A detailed timing diagram for the input transfer



- The picture shows two views of the signal except the clock.
- One view shows the signal seen by the master & the other is seen by the salve.
- The master sends the address & command signals on the rising edge at the beginning of clock period (t_0). These signals do not actually appear on the bus until t_{am}.

- Some times later, at t_{AS} the signals reach the slave.
- The slave decodes the address & at t1, it sends the requested data.
- At $t_{2,}$ the master loads the data into its i/p buffer.
- Hence the period $t_{2,}$ t_{DM} is the setup time for the masters i/p buffer.
- The data must be continued to be valid after $t_{2,}$ for a period equal to the hold time of that buffers.

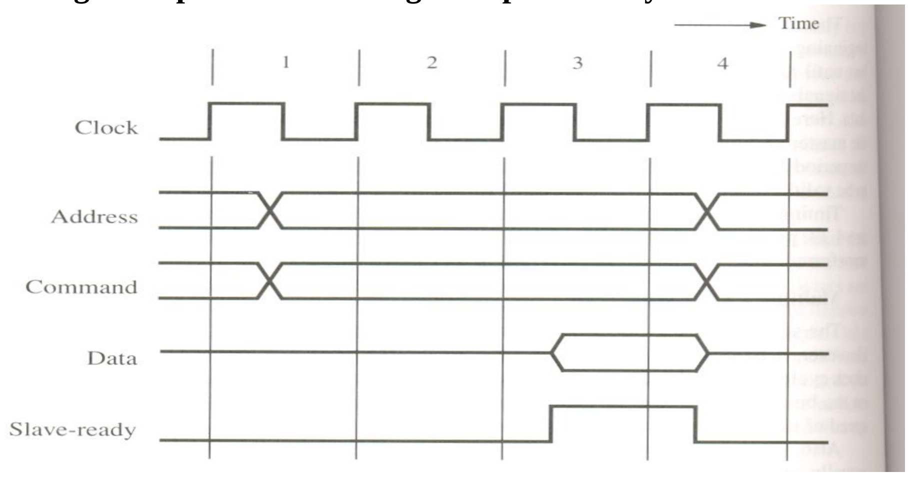
Demerits:

- ➢ The device does not respond.
- ➢ The error will not be detected.

Multiple Cycle Transfer:-

- During, clock cycle1, the master sends address & cmd infn/. On the bus' requesting a 'read' operation.
- The slave receives this information & decodes it.
- At the active edge of the clock (ie) the beginning of clock cycel2, it makes accession to respond immediately.
- The data become ready & are placed in the bus at clock cycle3.
- At the same times, the slave asserts a control signal called '**slave-ready**'.
- The master which has been waiting for this signal, strobes, the data to its i/p buffer at the end of clock cycle3.
- The bus transfer operation is now complete & the master sends a new address to start a new transfer in clock cycle4.
- The '**slave-ready**' signal is an acknowledgement form the slave to the master confirming that valid data has been sent.

Fig:An input transfer using multiple clock cycles

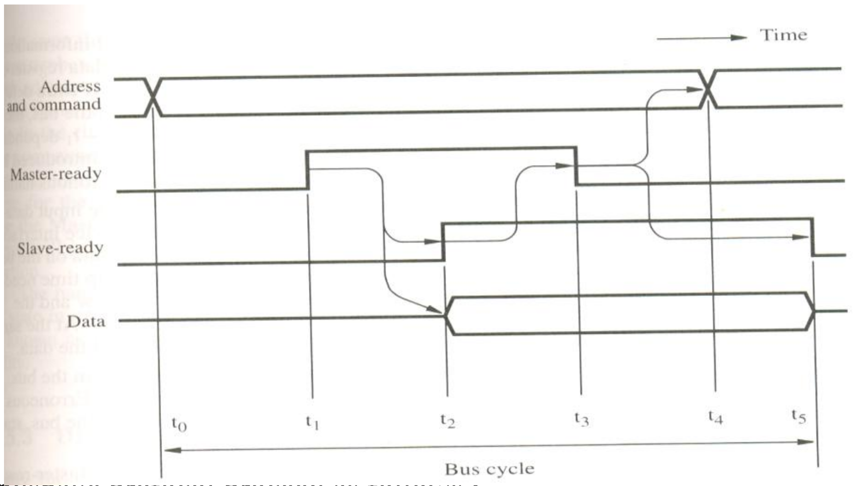


Asynchronous Bus:-

- An alternate scheme for controlling data transfer on. The bus is based on the use of '**handshake**' between **Master** & the **Slave**. The common clock is replaced by two timing control lines.
- They are

- Master–ready
- Slave ready.

Fig:Handshake control of data transfer during an input operation



The handshake protocol proceed as follows :

At $\mathbf{t_0}$ → The master places the address and command information on the bus and all devices on the bus begin to decode the information

At $\mathbf{t_1}$ → The master sets the Master ready line to 1 to inform the I/O devices that the address and command information is ready.

- The delay $t_1 – t_0$ is intended to allow for any skew that may occurs on the bus.
- The skew occurs when two signals simultaneously transmitted from one source arrive at the destination at different time.
- Thus to guarantee that the Master ready signal does not arrive at any device a head of the address and command information the delay $t_1 – t_0$ should be larger than the maximum possible bus skew.

At $\mathbf{t_2}$ → The selected slave having decoded the address and command information performs the required i/p operation by placing the data from its data register on the data lines. At the same time, it sets the “slave – Ready” signal to 1.

At $\mathbf{t_3}$ → The slave ready signal arrives at the master indicating that the i/p data are available on the bus.

At $\mathbf{t_4}$ → The master removes the address and command information on the bus. The delay between t_3 and t_4 is again intended to allow for bus skew. Errorneous addressing may take place if the address, as seen by some device on the bus, starts to change while the master – ready signal is still equal to 1.

At $\mathbf{t_5}$ → When the device interface receives the 1 to 0 tranitions of the Master – ready signal. It removes the data and the slave – ready signal from the bus. This completes the i/p transfer.

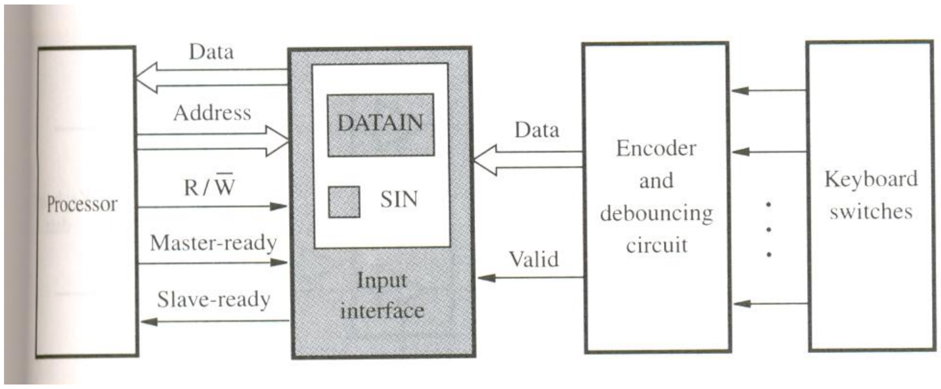
- In this diagram, the master place the output data on the data lines and at the same time it transmits the address and command information.
- The selected slave strobes the data to its o/p buffer when it receives the Master-ready signal and it indicates this by setting the slave – ready signal to 1.
- At time t_0 to t_1 and from t_3 to t_4, the Master compensates for bus.
- A change of state is one signal is followed by a change is the other signal. Hence this scheme is called as **Full Handshake**.
- It provides the higher degree of flexibility and reliability.

INTERFACE CIRCUITS:

The interface circuits are of two types.They are

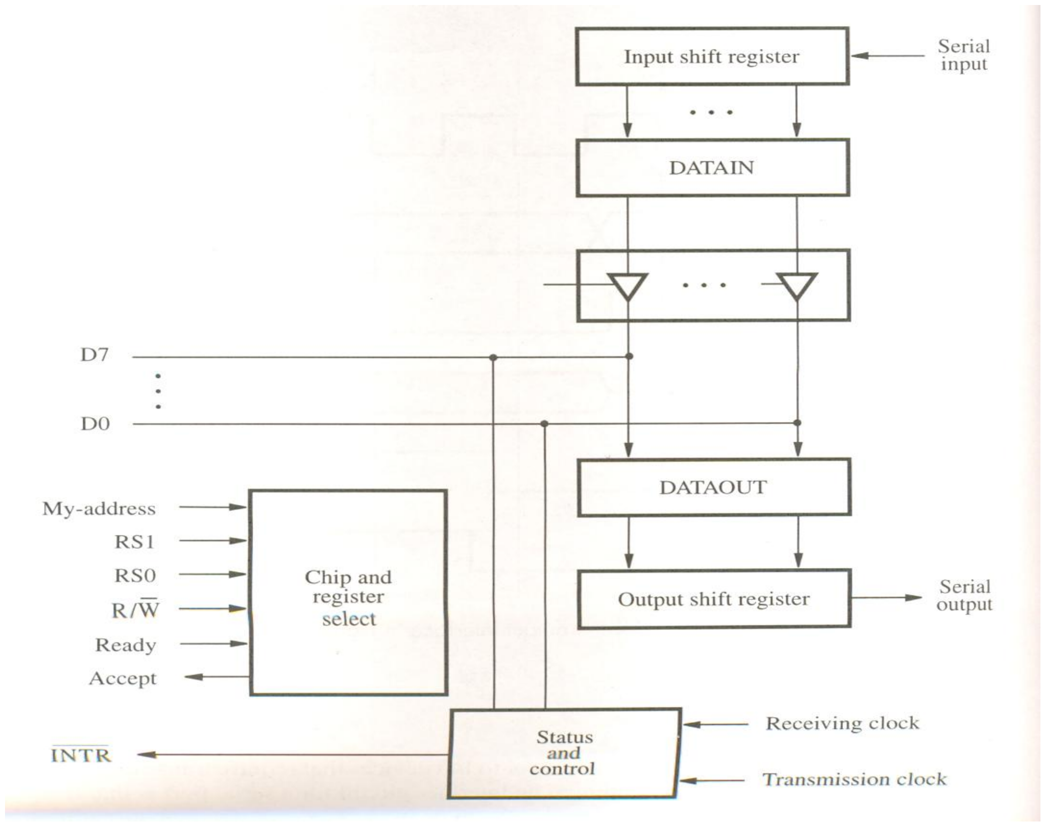
- Parallel Port
- Serial Port

Parallel Port:



- The output of the encoder consists of the bits that represent the encoded character and one signal called **vali**d,whi**c**h indicates the key is pressed.
- The information is sent to the interface circuits,which contains a data register,DATAIN and a status flag SIN.
- When a key is pressed, the Valid signal changes from 0 to1,causing the ASCII code to be loaded into DATAIN and SIN set to 1.
- The status flag SIN set to 0 when the processor reads the contents of the DATAIN register.
- The interface circuit is connected to the asynchronous bus on which transfers are controlled using the Handshake signals Master ready and Slave-ready.

Serial Port:

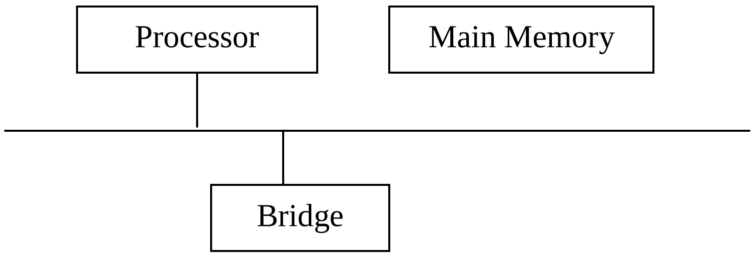


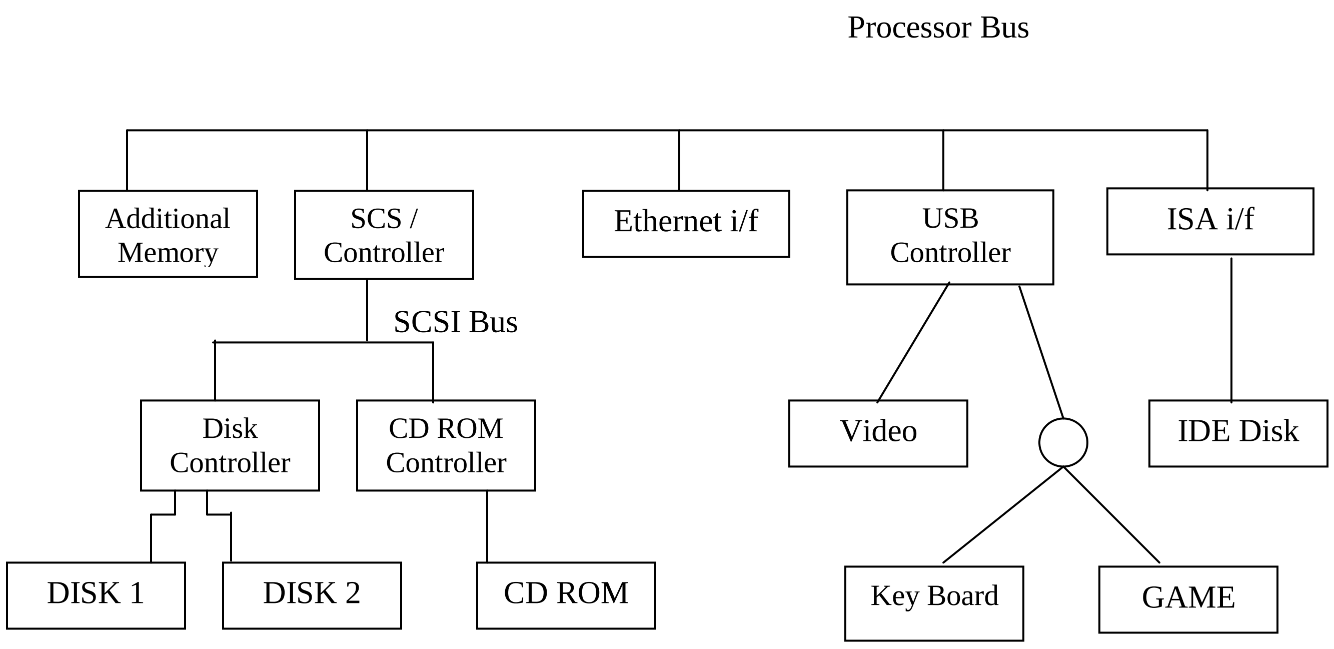
A serial port used to connect the processor to I/O device that requires transmission one bit at a time.
It is capable of communicating in a bit serial fashion on the device side and in a bit parallel fashion on the bus side.

STANDARD I/O INTERFACE

- A standard I/O Interface is required to fit the I/O device with an Interface circuit.
- The processor bus is the bus defined by the signals on the processor chip itself.
- The devices that require a very high speed connection to the processor such as the main memory, may be connected directly to this bus.
- **The bridge** connects two buses, which translates the signals and protocols of one bus into another.
- The bridge circuit introduces a small delay in data transfer between processor and the devices.

Fig:Example of a Computer System using different Interface Standards





We have 3 Bus standards.They are,

- **PCI** **(**Peripheral Component Inter Connect)
- **SCSI** (Small Computer System Interface)
- **USB** (Universal Serial Bus)

- **PCI** defines an expansion bus on the motherboard.
- **SCSI** and **USB** are used for connecting additional devices both inside and outside the computer box.
- **SCSI** bus is a high speed parallel bus intended for devices such as disk and video display.
- **USB** uses a serial transmission to suit the needs of equipment ranging from keyboard keyboard to game control to internal connection.
- **IDE (Integrated Device Electronics)** disk is compatible with ISA which shows the connection to an Ethernet.

PCI:

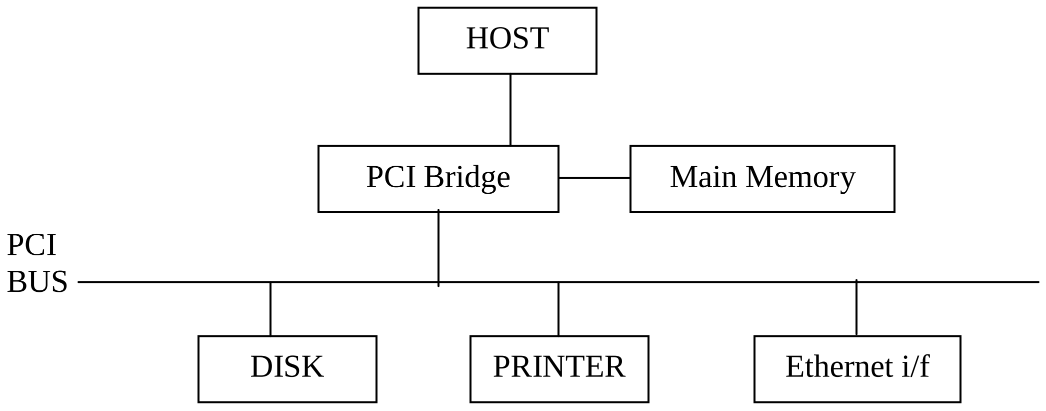
- PCI is developed as a low cost bus that is truly processor independent.
- It supports high speed disk, graphics and video devices.
- PCI has plug and play capability for connecting I/O devices.
- To connect new devices, the user simply connects the device interface board to the bus.

Data Transfer:

- The data are transferred between cache and main memory is the bursts of several words and they are stored in successive memory locations.
- When the processor specifies an address and request a 'read' operation from memory, the memory responds by sending a sequence of data words starting at that address.
- During write operation, the processor sends the address followed by sequence of data words to be written in successive memory locations.
- PCI supports read and write operation.
- A read / write operation involving a single word is treated as a burst of length one.
- PCI has three address spaces. They are
  - ➢ Memory address space
  - ➢ I/O address space
  - ➢ Configuration address space

- I/O address space → It is intended for use with processor
- Configuration space → It is intended to give PCI, its plug and play capability.
- PCI Bridge provides a separate physical connection to main memory.
- The master maintains the address information on the bus until data transfer is completed.
- At any time, only one device acts as **bus master**.
- A master is called 'initiator' in PCI which is either processor or **DMA.**
- The addressed device that responds to read and write commands is called a **target.**
- A complete transfer operation on the bus, involving an address and bust of data is called a '**transaction'.**

Fig:Use of a PCI bus in a Computer system

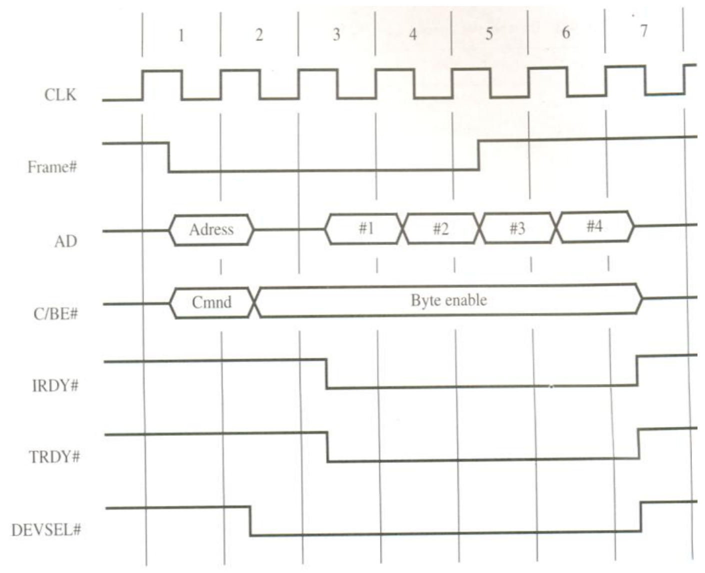


Data Transfer Signals on PCI Bus:

Name	**Function**
CLK	→ 33 MHZ / 66 MHZ clock
FRAME #	→ Sent by the indicator to indicate the duration of transaction
AD	→ 32 address / data line
C/BE #	→ 4 command / byte Enable Lines
IRDY, TRDYA	→ Initiator Ready, Target Ready Signals
DEVSEL #	→ A response from the device indicating that it has recognized its address and is ready for data transfer transaction.
IDSEL #	→ Initialization Device Select

Individual word transfers are called '**phases**'.

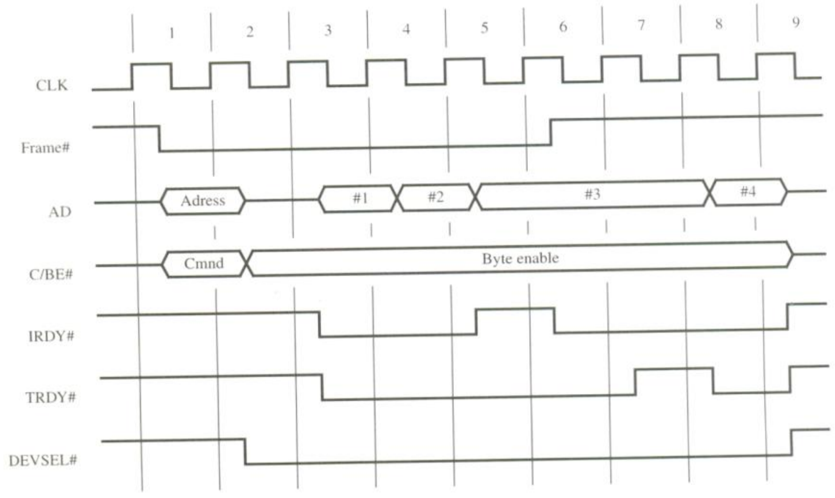
Fig :Read operation an PCI Bus



- In Clock cycle1, the processor asserts FRAME # to indicate the beginning of a transaction ; it sends the address on AD lines and command on C/BE # Lines.
- Clock cycle2 is used to turn the AD Bus lines around ; the processor ; The processor removes the address and disconnects its drives from AD lines.
- The selected target enable its drivers on AD lines and fetches the requested data to be placed on the bus.
- It asserts DEVSEL # and maintains it in asserted state until the end of the transaction.
- C/BE # is used to send a bus command in clock cycle and it is used for different purpose during the rest of the transaction.

- During clock cycle 3, the initiator asserts IRDY #, to indicate that it is ready to receive data.
- If the target has data ready to send then it asserts TRDY #. In our eg, the target sends 3 more words of data in clock cycle 4 to 6.
- The indicator uses FRAME # to indicate the duration of the burst, since it read 4 words, the initiator negates FRAME # during clock cycle 5.
- After sending the 4th word, the target disconnects its drivers and negates DEVSEL # during clockcycle 7.

Fig: A read operation showing the role of IRDY# / TRY#



- It indicates the pause in the middle of the transaction.
- The first and words are transferred and the target sends the 3rd word in cycle 5.
- But the indicator is not able to receive it. Hence it negates IRDY#.
- In response the target maintains 3rd data on AD line until IRDY is asserted again.
- In cycle 6, the indicator asserts IRDY. But the target is not ready to transfer the fourth word immediately, hence it negates TRDY in cycle 7. Hence it sends the 4th word and asserts TRDY# at cycle 8.

Device Configuration:

- The PCI has a configuration ROM memory that stores information about that device.
- The configuration ROM's of all devices are accessible in the configuration address space.
- The initialization s/w read these ROM's whenever the S/M is powered up or reset
- In each case, it determines whether the device is a printer, keyboard, Ethernet interface or disk controller.
- Devices are assigned address during initialization process and each device has an w/p signal called IDSEL # (Initialization device select) which has 21 address lines (AD) (AD to AD_{31}).
- During configuration operation, the address is applied to AD i/p of the device and the corresponding AD line is set to and all other lines are set to 0.

-
  AD11 - AD31 →**Upper address line**
  A00 - A10 →**Lower address line** → Specify the type of the operation and to access the content of device configuration ROM.
- The configuration software scans all 21 locations.
- PCI bus has interrupt request lines.
- Each device may requests an address in the I/O space or memory space

Electrical Characteristics:

- The connectors can be plugged only in compatible motherboards PCI bus can operate with either 5 – 33V power supply.
- The motherboard can operate with signaling system.

SCSI Bus:- (Small Computer System Interface)

- SCSI refers to the standard bus which is defined by ANSI (American National Standard Institute).
- SCSI bus the several options. It may be,

Narrow bus	→ It has 8 data lines & transfers 1 byte at a time.
Wide bus	→ It has 16 data lines & transfer 2 byte at a time.
Single-Ended Transmission	→ Each signal uses separate wire.
HVD (High Voltage Differential)	→ It was 5v (TTL cells)
LVD (Low Voltage Differential)	→ It uses 3.3v

- Because of these various options, SCSI connector may have 50, 68 or 80 pins.
- The data transfer rate ranges from 5MB/s to 160MB/s 320Mb/s, 640MB/s.
- The transfer rate depends on,
  - Length of the cable
  - Number of devices connected.
- To achieve high transfer rat, the bus length should be 1.6m for SE signaling and 12m for LVD signaling.
- The SCSI bus us connected to the processor bus through the SCSI controller.
- The data are stored on a disk in blocks called sectors.
- Each sector contains several hundreds of bytes. These data will not be stored in contiguous memory location.
- SCSI protocol is designed to retrieve the data in the first sector or any other selected sectors.
- Using SCSI protocol, the burst of data are transferred at high speed.
- The controller connected to SCSI bus is of 2 types. They are,
  - Initiator
  - Target

Initiator:

- It has the ability to select a particular target & to send commands specifying the operation to be performed.
- They are the controllers on the processor side.

Target:

- The disk controller operates as a target.
- It carries out the commands it receive from the initiator. The initiator establishes a logical connection with the intended target.

Steps:

Consider the disk read operation, it has the following sequence of events.

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

Bus Signals:-

- The bus has no address lines.
- Instead, it has data lines to identify the bus controllers involved in the selection / reselection / arbitration process.
- For narrow bus, there are 8 possible controllers numbered from 0 to 7.
- For a wide bus, there are 16 controllers.
- Once a connection is established b/w two controllers, these is no further need for addressing & the datalines are used to carry the data.

SCSI bus signals:

Category	Name	Function
Data	- DB (0) to DB (7) - DB(P)	Datalines Parity bit for data bus.
Phases	- BSY - SEL	Busy Selection
Information type	- C/D - MSG	Control / Data Message
Handshake	- REQ - ACK	Request Acknowledge
Direction of transfer	I/O	Input / Output
Other	- ATN - RST	Attention Reset.

- All signal names are proceeded by minus sign.
- This indicates that the signals are active or that the dataline is equal to 1, when they are in the low voltage state.

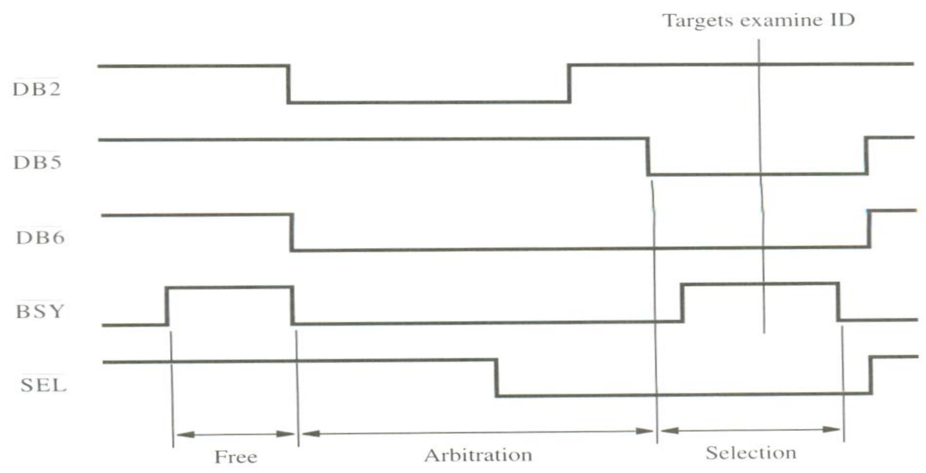
Phases in SCSI Bus:-

- The phases in SCSI bus operation are,
  - Arbitration
  - Selection
  - Information transfer
  - Reselection

Arbitration:-

- When the –BSY signal is in inactive state, the bus will he free & any controller can request the use of the bus.
- Since each controller may generate requests at the same time, SCSI uses distributed arbitration scheme.
- Each controller on the bus is assigned a fixed priority with controller 7 having the highest priority.
- When –BSY becomes active, all controllers that are requesting the bus examines the data lines & determine whether the highest priority device is requesting the bus at the same time.
- The controller using the highest numbered line realizes that it has won the arbitration process.
- At that time, all other controllers disconnect from the bus & wait for –BSY to become inactive again.

Fig:Arbitration and selection on the SCSI bus.Device 6 wins arbitration and select device 2



Selection:

- Here Device wons arbitration and it asserts –BSY and –DB6 signals.
- The Select Target Controller responds by asserting –BSY.
- This informs that the connection that it requested is established.

Reselection:

- The connection between the two controllers has been reestablished,with the target in control the bus as required for data transfer to proceed.

USB – Universal Serial Bus

- USB supports 3 speed of operation. They are,
  - ➢ Low speed (1.5Mb/s)
  - ➢ Full speed (12mb/s)
  - ➢ High speed (480mb/s)
- The USB has been designed to meet the key objectives. They are,

❖ It provide a simple, low cost & easy to use interconnection s/m that overcomes the difficulties due to the limited number of I/O ports available on a computer.
❖ It accommodate a wide range of data transfer characteristics for I/O devices including telephone & Internet connections.
❖ Enhance user convenience through **‘Plug & Play’** mode of operation.

Port Limitation:-

- Normally the system has a few limited ports.
- To add new ports, the user must open the computer box to gain access to the internal expansion bus & install a new interface card.
- The user may also need to know to configure the device & the s/w.

Merits of USB:-

USB helps to add many devices to a computer system at any time without opening the computer box.

Device Characteristics:-

- The kinds of devices that may be connected to a cptr cover a wide range of functionality.
- The speed, volume & timing constrains associated with data transfer to & from devices varies significantly.

Eg:1 Keyboard → Since the event of pressing a key is not synchronized to any other event in a computer system, the data generated by keyboard are called asynchronous. The data generated from keyboard depends upon the speed of the human operator which is about 100bytes/sec.

Eg:2 Microphone attached in a cptr s/m internally / externally

- The sound picked up by the microphone produces an analog electric signal, which must be converted into digital form before it can be handled by the cptr.
- This is accomplished by sampling the analog signal periodically.
- The sampling process yields a continuous stream of digitized samples that arrive at regular intervals, synchronized with the sampling clock. Such a stream is called isochronous (ie) successive events are separated by equal period of time.
- If the sampling rate in 'S' samples/sec then the maximum frequency captured by sampling process is s/2.
- A standard rate for digital sound is 44.1 KHz.

Requirements for sampled Voice:-

- It is important to maintain precise time (delay) in the sampling & replay process.
- A high degree of jitter (Variability in sampling time) is unacceptable.

Eg-3:Data transfer for Image & Video:-

- The transfer of images & video require higher bandwidth.
- The bandwidth is the total data transfer capacity of a communication channel.
- To maintain high picture quality, The image should be represented by about 160kb, & it is transmitted 30 times per second for a total bandwidth if 44MB/s.

Plug & Play:-

- The main objective of USB is that it provides a plug & play capability.
- The plug & play feature enhances the connection of new device at any time, while the system is operation.
- The system should,
    - Detect the existence of the new device automatically.
    - Identify the appropriate device driver s/w.
    - Establish the appropriate addresses.
    - Establish the logical connection for communication.

USB Architecture:-

- USB has a serial bus format which satisfies the low-cost & flexibility requirements.
- Clock & data information are encoded together & transmitted as a single signal.
- There are no limitations on clock frequency or distance arising form data skew, & hence it is possible to provide a high data transfer bandwidth by using a high clock frequency.
- To accommodate a large no/. of devices that can be added / removed at any time, the USB has the tree structure.

Fig:USB Tree Structure

- Each node of the tree has a device called '**hub**', which acts as an intermediate control point b/w host & I/O devices.
- At the root of the tree, the 'root hub' connects the entire tree to the host computer.
- The leaves of the tree are the I/O devices being served.

UNIT V

PART A

1) What is I/O bus structure?
2) What is shared data and control lines memory mapped I/O?
3) What is asynchrome data transfer?
4) What is meant by strobe?
5) Write about hand shaking?
6) What is timing diagram?
7) Define Handshaking.
8) What are the functions of I/O module?
9) What are the I/O techniques?
10) What is programmed I/O?
11) What is interrupt driven I/O?
12) What is DMA?
13) What is meant by DMA break points and interrupt break points?
14) Define Vector Interrupt.
15) Based on what factors,the vector execution time will be calculated?
16) What is Interrupt Latency?

PART – B

1) Write the short notes on programmed I/O and memory mapped I/O
2) What is DMA? Describe how DMA is used to transfer data?
3) Explain briefly about PCI and USB Bus.
4) Discuss about the structure of a distributed shared memory system with various interconnection schemes and message passing mechanism.
5) Explain the bus interconnection structure and characteristics of SCSI bus Standard.

D 276

B.E./B.Tech. DEGREE EXAMINATION, APRIL/MAY 2003.

Fourth Semester

Computer Science and Engineering

CS 238 — COMPUTER ARCHITECTURE — I

Time : Three hours Maximum : 100 marks

Answer ALL questions.

PART A — (10 × 2 = 20 marks)

1. Define Absolute Addressing.

2. What are the different addressing modes?

3. Define underflow.

4. What is meant by bit sliced processor?

5. Define microinstruction.

6. What is Pipelining?

7. How Cache memory is used in reducing the execution time?

8. Define memory interleaving.

9. What is DMA?

10. Define dumb terminal.

PART B — (5 × 16 = 80 marks)

11. Describe Von Neuman Architecture in detail. (16)

12. (a) Explain the bit slice processor and its internal structure of the ALU.

Or

(b) How Floating point addition is implemented? Explain briefly with neat diagram.

13. (a) Describe the Microprogrammed control unit based on Wilke's original design.

Or

(b) Explain the different Hard wired controllers.

14. (a) How Cache blocks are mapped to main memory module using direct mapping method and Associate Mapping Method.

Or

(b) What is Virtual Memory? Explain the Virtual–memory address translation.

15. (a) Write short notes on programmed I/O and memory mapped I/O.

Or

(b) What is DMA? Describe how DMA is used to transfer Data from peripherals.

———————