P. R. ENGINEERING COLLEGE

Vallam, Thanjavur - 613 403.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CS2253-COMPUTER ORGANIZATION AND ARCHITECTURE

Prepared By

S.JANCY SICKORY DAISY

Assistant
Professor/CSE

(Common to CSE & IT)

CS 2253 COMPUTER ORGANIZATION AND ARCHITECTURE

L T P C

3 0 0 3

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 Input/Output -Interrupts – Direct Memory Access – Buses – Interface circuits – Standard I/O Interfaces (PCI, SCSI, USB), I/O devices and processors.

TOTAL: 45 PERIODS

TEXT BOOK:

1. Carl Hamacher, Zvonko Vranesic and Safwat Zaky, “Computer Organization”, Fifth Edition, Tata McGraw Hill, 2002.

UNIT I - 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.

Digital computer systems consist of three distinct units. These units are as follows: Input unit Central Processing unit Output unit these units are interconnected by electrical cables to permit communication between them. This allows the computer to function as a system.

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.

Computer hardware:

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

1. **Functional Units**
   Digital computer systems consist of three distinct units. These units are as follows: Input unit Central Processing unit Output unit these units are interconnected by electrical cables to permit communication between them. This allows the computer to function as a system. Input Unit A computer must receive both data and program statements to function properly and be able to solve problems. The method of feeding data and programs to a computer is accomplished by an input device. The brain of a computer system is the central processing unit (CPU). The CPU processes data transferred to it from one of the various input devices. It then transfers either an intermediate or final result of the CPU to one or more output devices.

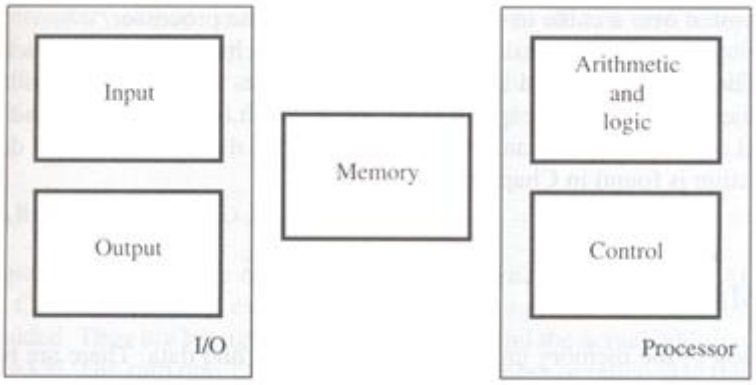
   A computer consists of 5 main parts.

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

   - 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.

Fig: Basic Functional units of a Computer



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.
-

2. 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
```

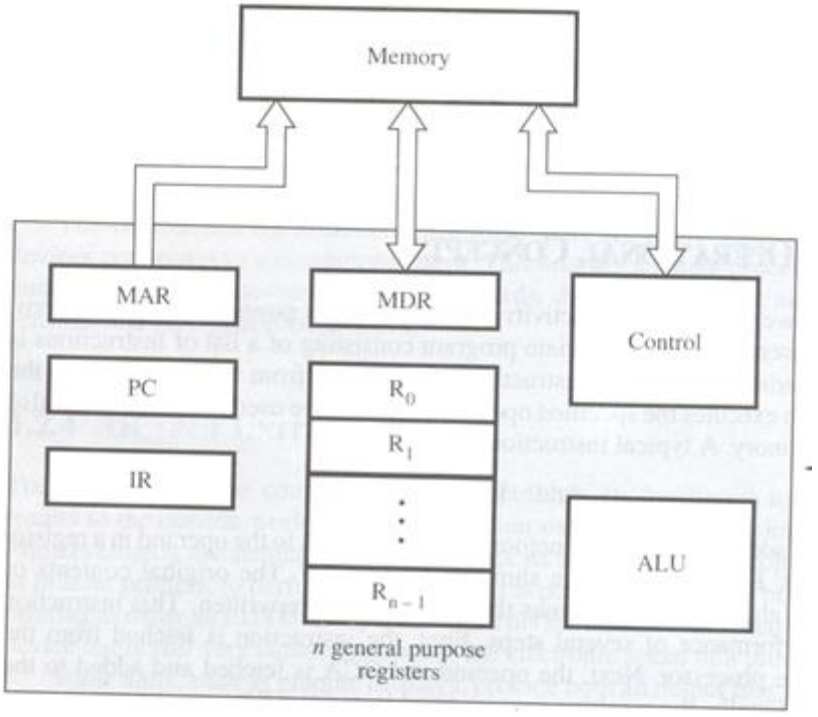
Transfer the contents of memory location A to the register R1.

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



- Instruction Register(IR)
- Program Counter(PC)
- 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.

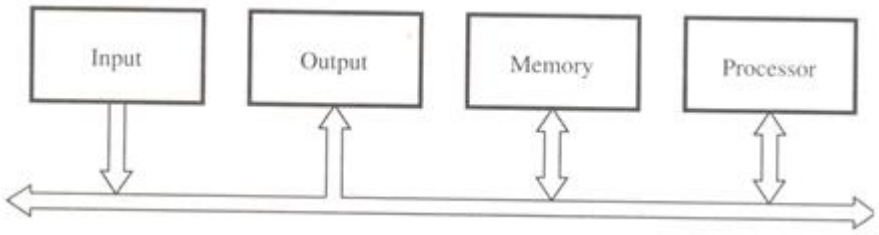
Interrupt:

- 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.

2. BUS STRUCTURES:

Bus structure and multiple bus structures are types of bus or computing. A bus is basically a subsystem which transfers data between the components of a Computer components either within a computer or between two computers. It connects peripheral devices at the same time. A Bus may be lines or wires or one bit per line.The lines carry data or address or control signal.

Fig: Single Bus Structure



There are 2 types of Bus structures. They are

Single Bus Structure

Multiple Bus Structure

3.1.Single Bus Structure:

- A Single bus structure is very simple and consists of a single server. A bus cannot span multiple cells. And each cell can have more than one buses.
- Published messages are printed on it. There is no messaging engine on 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:

- A multiple Bus Structure has multiple inter connected service integration buses and for each bus the other buses are its foreign buses.
- It allows two or more transfer at a time.
- It costs high.
- It provides concurrency in operation.
- Its Performance is high.

4. SOFTWARE:

Computer software, or just **software** is a general term used to describe the role that computer programs, procedures and documentation play in a computer system.

Software Characteristics

- Software is developed and engineered.
- Software doesn't "wear-out".
- Most software continues to be custom built

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:

Application software allows end users to accomplish one or more specific (not directly computer development related) tasks. 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.

Typical applications include:

- industrial automation
- business software
- computer games
- quantum chemistry and solid state physics software
- telecommunications (i.e., the internet and everything that flows on it)
- databases
- educational software
- medical software

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)

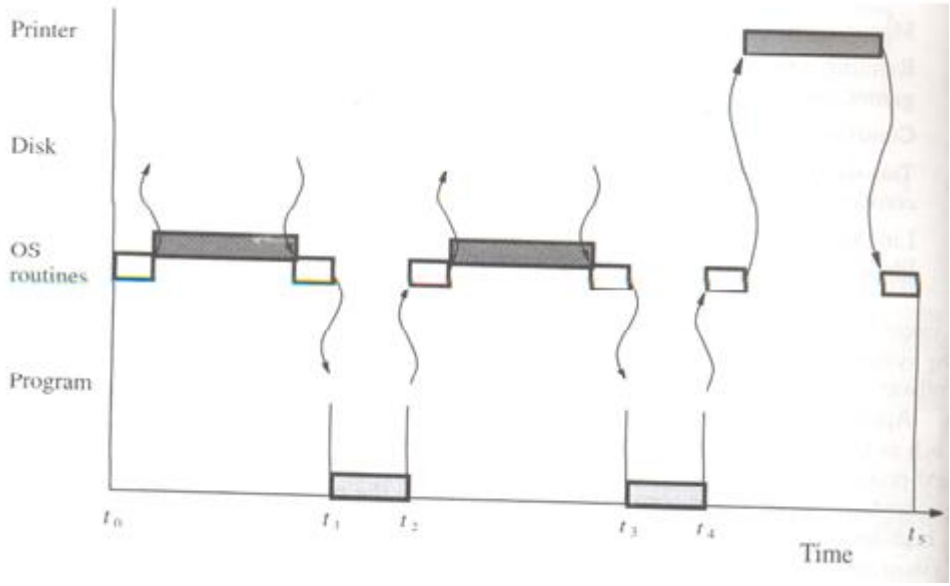
Operating System :

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.

4.1. PERFORMANCE:

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

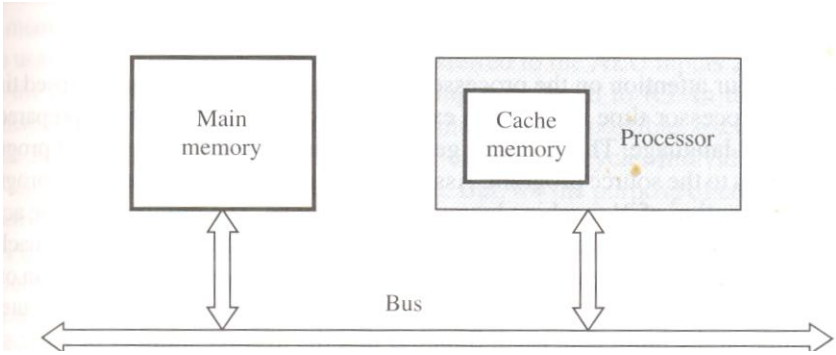
Elapsed Time→the total time required to execute the program is called the elapsed time.

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.

Clock Rate,R =1/P

Where, P→Length of one clock cycle.

Basic Performance Equation:

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 every 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:

RISC stands for Reduced Instruction Set Computer. The ISA is composed of instructions that all have exactly the same size, usually 32 bits. Thus they can be pre-fetched and pipelined successfully. All ALU instructions have 3 operands which are only registers. The only memory access is through explicit LOAD/STORE instructions.
Thus A = B + C will be assembled as:

```
LOAD  R1,A
LOAD  R2,B

ADD   R3,R1,R2
STORE C,R3
```

Although it takes 4 instructions we can reuse the values in the registers.

To make all instructions the same length the number of bits that are used for the opcode is reduced. Thus less instruction are provided.

The older architecture is called CISC (Complete Instruction Set Computer).

The primary goal of CISC architecture is to complete a task in as few lines of assembly as possible. This is achieved by building processor hardware that is capable of understanding and executing a series of operations. For this particular task, a CISC processor would come prepared with a specific instruction (we'll call it "MULT"). When executed, this instruction loads the two values into separate registers, multiplies the operands in the execution unit, and then stores the product in the appropriate register. Thus, the entire task of multiplying two numbers can be completed with one instruction:

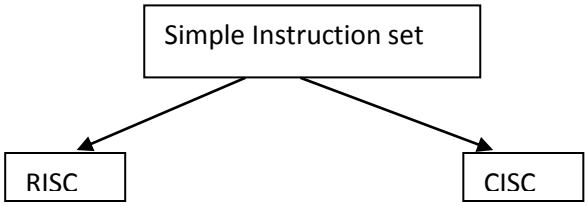
```
MULT 2:3, 5:2
```

MULT is what is known as a "complex instruction." It operates directly on the computer's memory banks and does not require the programmer to explicitly call any loading or storing functions. It closely resembles a command in a higher level

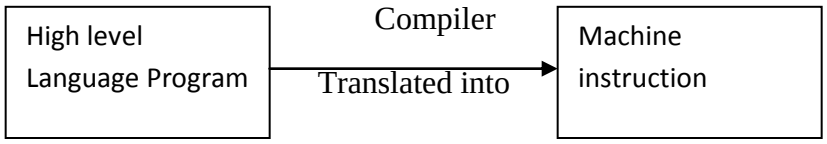
language. For instance, if we let "a" represent the value of 2:3 and "b" represent the value of 5:2, then this command is identical to the C statement "a = a *

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.



Compiler:



Functions of Compiler:

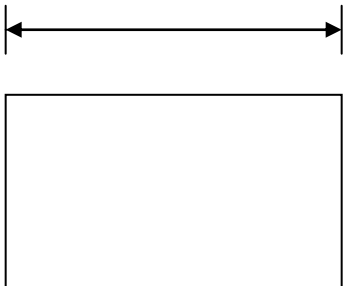
- The compiler re-arranges the program instruction to achieve better performance.
- The high quality compiler must be closely linked to the processor architecture to reduce the total number of clock cycles.

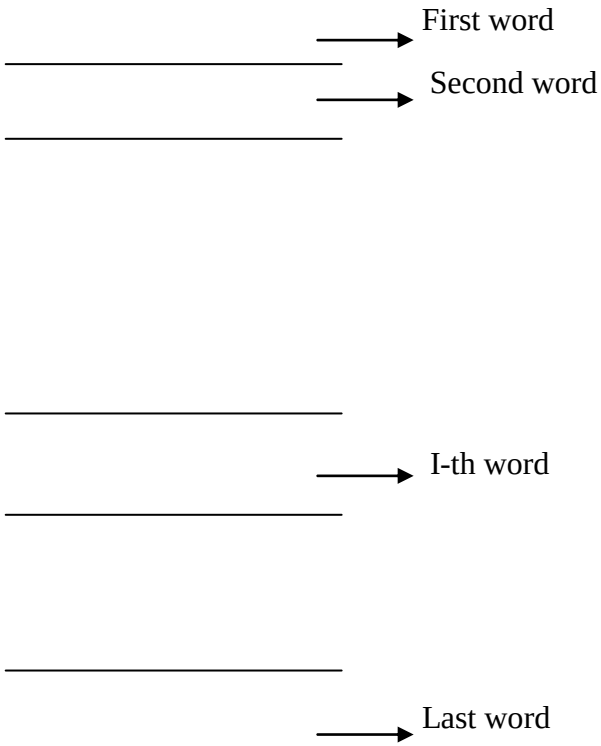
5. MEMORY LOCATION AND ADDRESSES:

- The memory consists of many millions of storage cells, each of which can store a bi of information having value 0 or 1.
- The group of 'n' bits is referred as 'word of information' and 'n' is called the word length.
- The word length ranges from 16 to 64 bits.(ie) if the word length of a computer is 32 bits , a single word can store a 32 bit 2's complement number or four ASCII characters.
- A unit of 8 bits is called a byte.
- Accessing the memory to store or retrieve the items of information requires the distinct address or names.
- It ranges from 0 to 2^k -1.

Where, 2^k → It indicates the address space of the computer.

5.1. Memory words n bits





5.2. Byte Addressability:

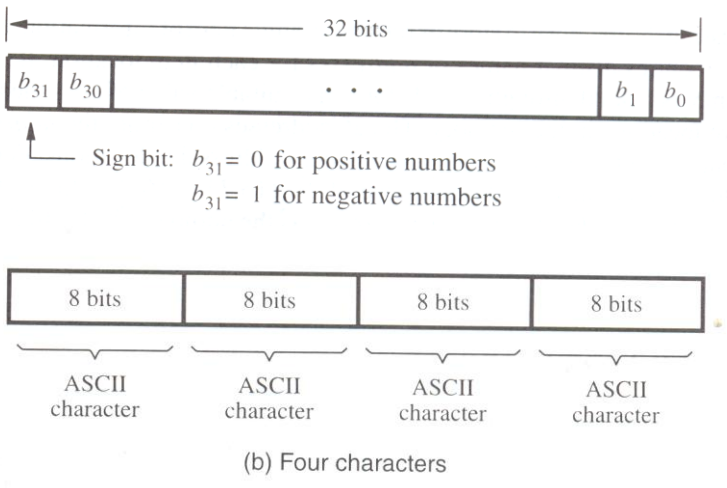
Bit →0 or 1

Byte→8 bits

Word→4 bytes

It is impractical to assign distinct address to individual bit locations in memory. The most practical assignment is to have successive address refer to successive byte locations in the memory.

Fig: A Signed Integer



5.3.Big Endian and Little Endian Assignment:

There are 2 ways in which the byte address can be assigned across words.

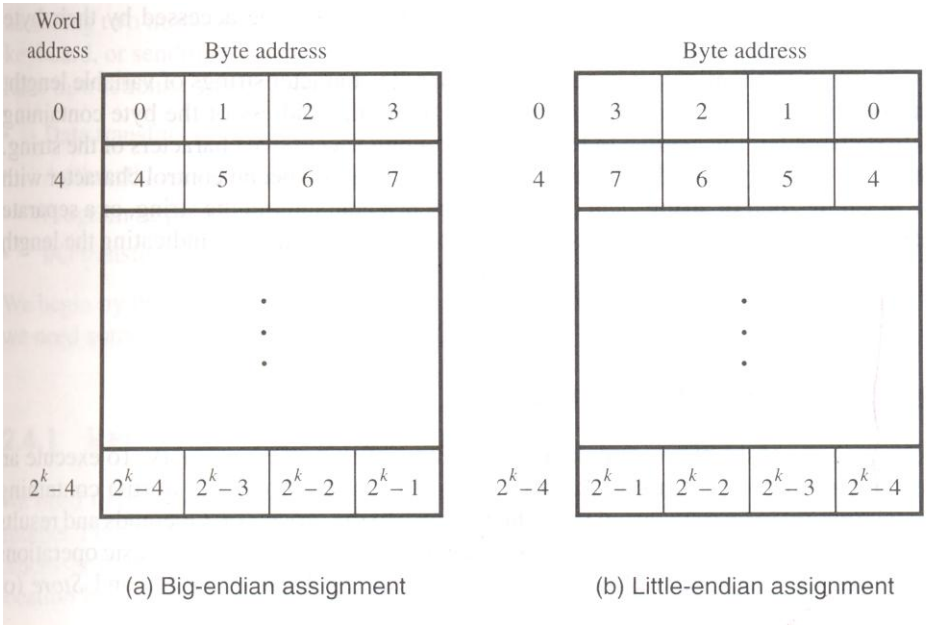
Big-Endian Format:

It is used when lower byte address are used for the most significant bit (leftmost bytes).

Little-Endian Format:

It is used when lower byte address are used for the less significant bit (rightmost bytes).

.



In both cases , byte addresses 0,4,8,…… are taken as the addresses of successive words in the memory and are the addresses used when specifying memory read and write operations for words.

5.4.Word Alignment:

Word locations have aligned addresses. Words are said to be aligned in memory, if they begin at a byte address ie. Multiple of the number of bytes in a word.

For practical reasons associated with manipulating binary coded addresses, the number of words in a word is a power of 2.

Word length	Addresses
16	0,2,4….
32	0,4,8…..
64	0,8,16….

Words that can't begin at arbitrary byte locations are said to be unaligned addresses.

Accessing Numbers, Characters and Character Strings:

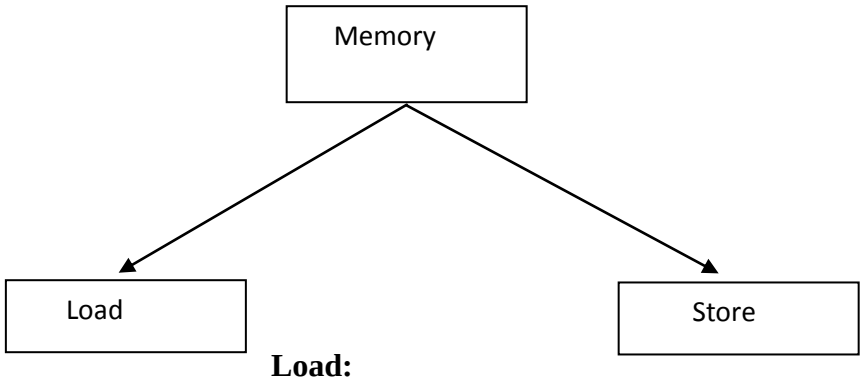
- Numbers are accessed by their word address.
- Character is accessed by their byte locations.
- For character string, the beginning of the String is indicated by giving the address of the byte containing first character.
- Successive byte locations contain successive characters of the string.

There are 2 ways to indicate the length of the string. They are,

- ❖ Special Control Character with the meaning of 'End Of String'. Eg.Hai\0
- ❖ Separate memory word locations or processor register can contain a number of indications of the length of the String in bytes.

6. MEMORY OPERATIONS:

Both the program instructions and data are stored in the memory.



Load:

- The Load operation transfers a copy of a specific memory location to the processor.
- The memory contents remain unchanged.
- To start load operation, the processor send the address of the desired location to the memory and request its contents be read.

- The Memory sends the data stored at that address and sends them to the processor.

Store:

- The Store operation transfers the item of information from the processor to the specific memory location and destroying the former contents of that location.
- The processor sends the address of the desired location to the memory; together with data to be written into that location.

7. INSTRUCTION AND INSTRUCTION SEQUENCING:

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.

7.1. 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

7.2. Assembly Language Notation:

AssemblyLanguage 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.

7.3. 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.

7.4. 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.

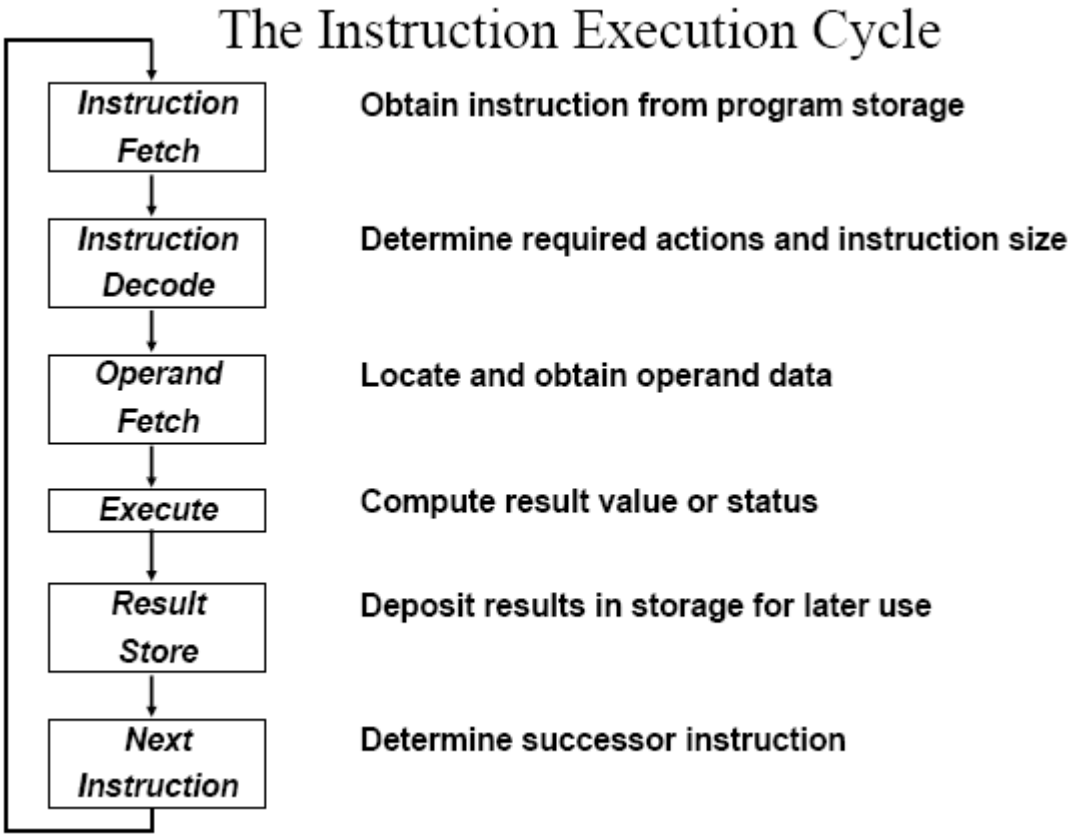
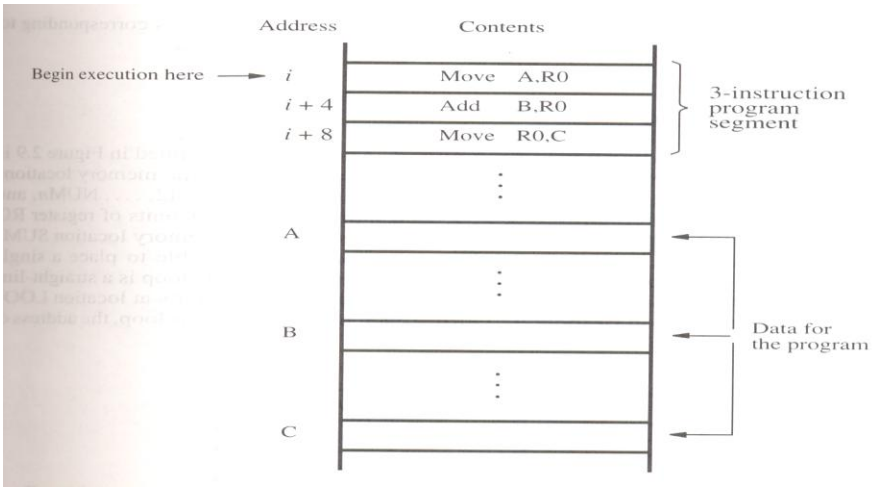


Fig: Program Execution

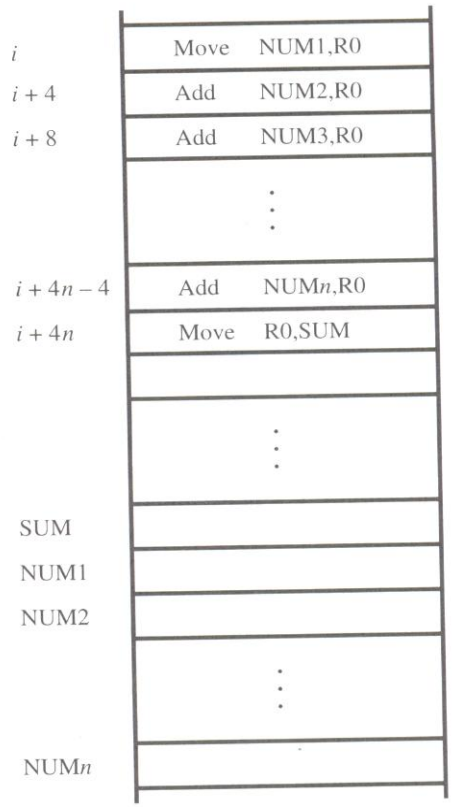


7.5.Branching:

- 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.

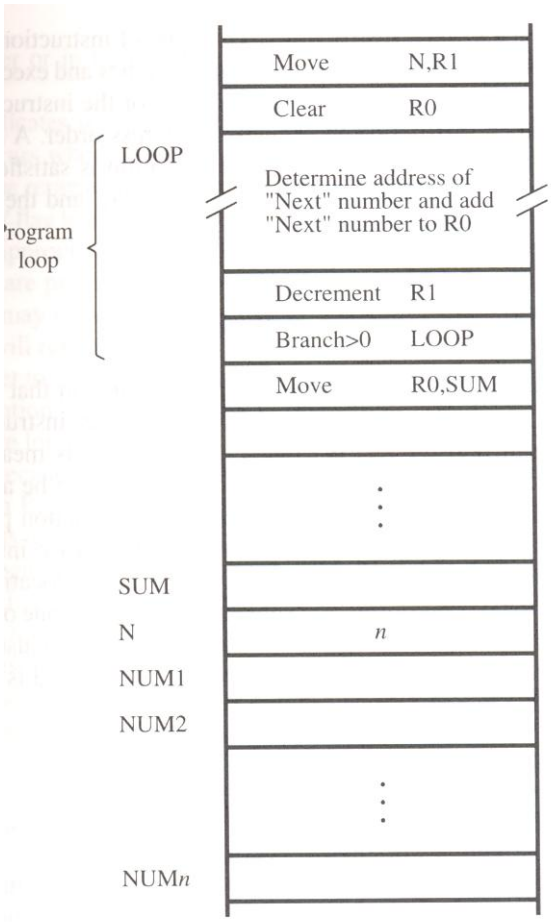
Decrement R1

- 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:



7.6. 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.

8. 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

8.1. 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.

Immediate Mode.

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

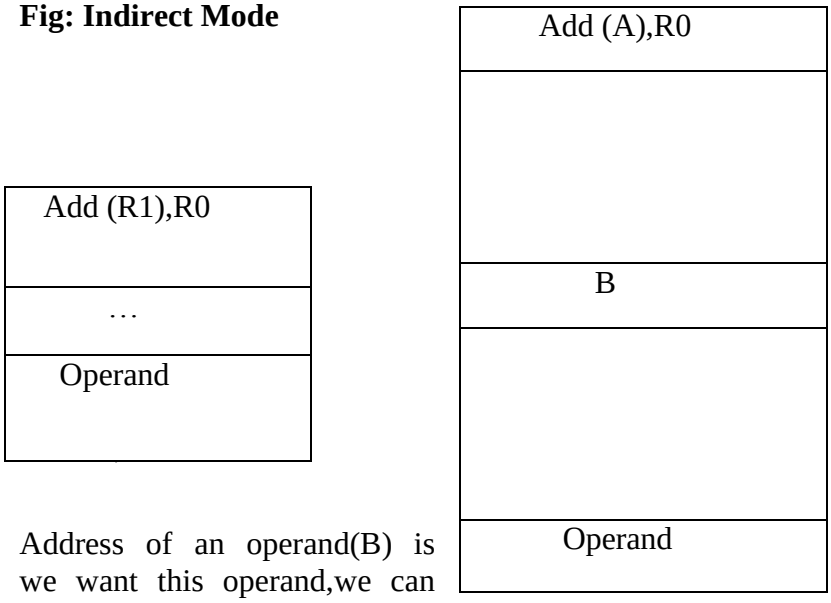
9.2. Indirection and Pointers:

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]

9.3. Indexing and Arrays:

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,

$$X(R_i)$$

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,

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

9.4. 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

9.5. 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

10. ASSEMBLY LANGUAGE:

When writing programs for a specific computer,such words are normally replaced by acronym called mnemonics such as MOV ,ADD , INC and BR.

A complete set of such symbolic names and rules constitute a programming language is referred to as an assembly language.

The set of rules for using the mnemonics in the specification of complete instruction is called the syntax of the language.

The programs written in assembly language can be automatically translated into sequence of machine instruction by a program called an assembler.

The user program is called the source program. The assembled machine language program is called the object program.

Assembler Directives:

The statements or commands are called the assembler directives, which are used by the assembler to translate the source program into the object program.

EQU → It assigns numerical values to any name used in the program.

Eg. SUM EQU 200

If the assembler is to produce an object program according to this arrangement, it has to know,

- How to interpret the names
- Where to place the instruction in the memory
- Where to place the data operands in the memory

Types of Directive:

ORIGIN Directive→It tells the assembler program where the memory has to place the data block.

DATAWORD Directive→It is used to inform the assembler of requirement.(how many data need in one program?).

RESERVE Directive→Allocates the total memory needed by the program.

RETURN Directive→Identifies the point at which the execution of the program to be terminated.

END Directive→It indicates the end of the source program.

It includes the label START which is the address of the location at which the execution of the program is to begin.

Assembly Language program:

It is a collection of instructions. These instructions depend on 4 fields.They are,

Label→ optional eg.Loop/ Label

Operation→ Which type of operation eg.SUM .MOVE

Operand(s)→ What are the operands to be used eg.A,B

Comment → It is not used in instruction.It is used when we start the preparation of the document.

Assembly and Execution of Program:

- A Source program witten in assembly language is stored in secondary storage devices(magnetic disk).
- The object program must be loaded into the memory of the computer before it is executed.

Loader:

A Loader is system software that is used to fetch the input operations needed to transfer the machine language program from the disk into a specified place in memory. Then the information is stored into the main memory using assembler and then it is translated into object form(byte code or opcode).

Types of Assembler:

There are 2 types of assembler. They are,

One pass assembler

Two pass assembler

One pass assembler:

- It is used to create the symbol table, which contains only variables used in the program.
- It is used to scan the program only one time.

Two pass assembler:

- Here the symbol table contains the variables and the original values for those variables.
- It is used to scan the program for two times.
- Two pass assembler can overcome the demerits of One pass assembler and it is most commonly used.
- The assembler can detect and report syntax errors. Other programming errors are find out by the system software debugger program and the user must able to find the error.

Number Notation:

Eg: To add 93 value to the register R

Decimal: ADD #93,R1 → #(immediate mode)

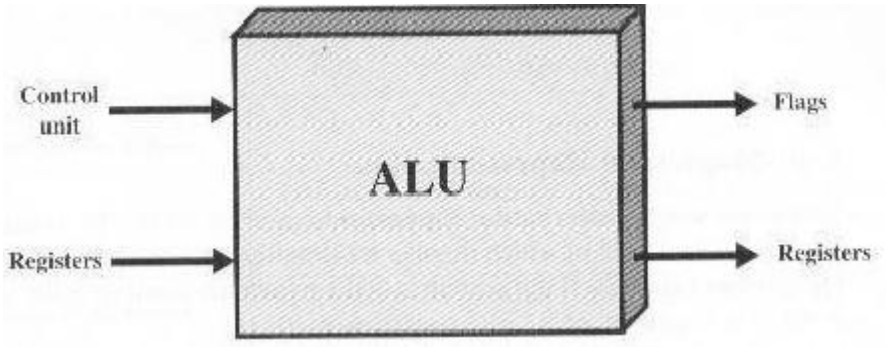
Hexa-Decimal: ADD #$5D,R1 →$(used to represent/assign hexadecimal value to R1)

Binary : ADD # % 01011101,R1 →% used to represent/assign hexadecimal value to R1

THE ARITHMETIC AND LOGIC UNIT

The ALU is the part of the computer that actually performs arithmetic and logical operations on data.

Fig: All inputs and outputs



Multiplication

More complicated than addition

• Accomplished via shifting and addition

More time and more area

Multiplication example: 0010 x 0110

Iteration	Multiplicand	Original Algorithm	
		Step	Product
0	0010	Initial values	0000 0110
1	0010	1: **0** -> No operation	0000 0110
		2: Shift right	0000 0011
2	0010	1a: **1** -> Product = Product + Multiplicand	**0010** 0011
		2: Shift right	0001 0001
3	0010	1a: 1 -> Product = Product + Multiplicand	**0011** 0001
		2: Shift right	0001 1000
4	0010	1: **0** -> No operation	0001 1000
		2: Shift right	0000 1100

Signed Multiplication

The easiest way to deal with signed numbers is to first convert the multiplier and multiplicand to positive numbers and then remember the original sign. It turns out that the last algorithm will work with signed numbers provided that when we do the shifting steps we extend the sign of the product.

Speeding up multiplication (Booth's Algorithm)

The way we have done multiplication so far consisted of repeatedly scanning the multiplier, adding the multiplicand (or zeros) and shifting the result accumulated.

Observation:

if we could reduce the number of times we have to add the multiplicand that would make the all process faster

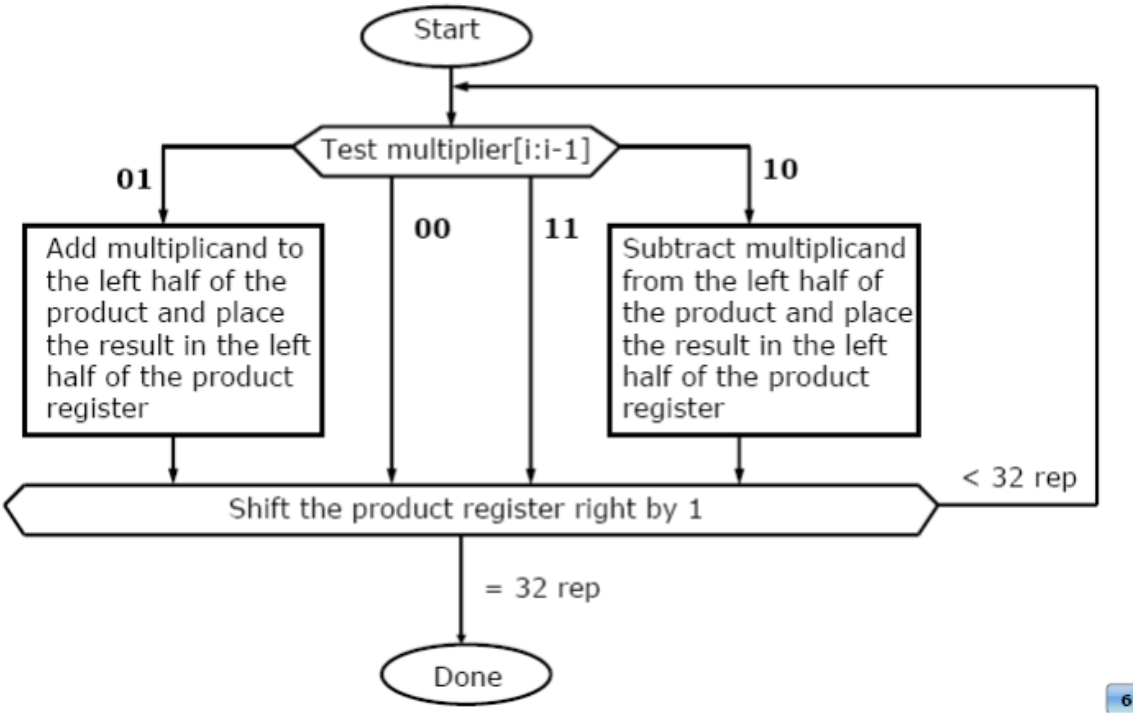
Booth's Algorithm

Observation: If besides addition we also use subtraction, we can reduce the number of consecutives additions and therefore we can make the multiplication faster.

This requires to "recode" the multiplier in such a way that the number of consecutive 1s in the multiplier (indeed the number of consecutive additions we should have done) are reduced.

The key to Booth's algorithm is to scan the multiplier and classify group of bits into the beginning, the middle and the end of a run of 1s

Booth's algorithm



6

Booth's algorithm example

Iteration	Multiplicand	Booth's Algorithm	
		Step	Product
0	0010	Initial values	0000 1101 **0**
1	0010	**10** -> Product = Product - Multiplicand	1110 1101 0
		Shift right	1111 011**0 1**
2	0010	**01** -> Product = Product + Multiplicand	**0001** 0110 1
		Shift right	0000 101**1 0**
3	0010	**10** -> Product = Product - Multiplicand	**1110** 1011 0
		Shift right	1111 010**1 1**
4	0010	**11** -> No operation	1111 0101 1
		Shift right	**1111 1010** 1

Division

Even more complicated can be accomplished via shifting and addition/subtraction More time and more area we will look at 3 versions based on grade school algorithm

0011 | 0010 0010 (Dividend)

(Divisor) 0011 | 0010 0010 (Dividend)

Negative numbers: Even more difficult There are better techniques, we won't look at them

Division

```
                    1001  (Quotient)
Divisor 1000   ┌  1001010  (Dividend)
               -1000
                    10
                    101
                    1010
                   -1000
                      10  (Remainder)
```

Dividend = Quotient x Divider + Remainder

Restoring division example

Iteration	Divisor	Divide Algorithm	
		Step	Product
0	0010	Initial values	0000 0111
		Shift reminder left by 1	0000 1110
1	0010	2. Reminder = Reminder - Divisor	**1110** 1110
		3b. (Reminder < 0); +Div; Shift left, R0 = 0	0001 110**0**
2	0010	2. Reminder = Reminder - Divisor	**1111** 1100
		3b. (Reminder < 0); +Div; Shift left, R0 = 0	0011 100**0**
3	0010	2. Reminder = Reminder - Divisor	**0001** 1000
		3a. (Reminder > 0); Shift left, R0 = 1	0011 000**1**
4	0010	2. Reminder = Reminder - Divisor	**0001** 0001
		3a. (Reminder > 0); Shift left, R0 = 1	0010 001**1**
Done	0010	Shift left half of reminder right by 1	**0001** 0011

Non-restoring division

How can we avoid adding the divisor back to the reminder?

- Note that this addition is performed whenever the reminder is negative!

So, what exactly are we doing when the reminder is negative?

- We have a certain reminder: R (R < 0)
- We add the divisor back to it: R + D
- We shift the result left by 1: 2*(R + D) = 2*R + 2*D
- We subtract the divisor again in the next step: 2*R + 2*D - D = 2*R + D

- Equivalent of left shifting the reminder R by 1 bit
- Add the divisor in the next step, instead of subtracting

Non-restoring division example

Iteration	Divisor	Divide Algorithm	
		Step	Product
0	0010	Initial values	0000 0111
		Shift reminder left by 1	0000 1110
1	0010	Reminder = Reminder - Divisor	**1110** 1110
		(Reminder < 0); Shift left, R0 = 0	1101 110**0**
2	0010	**Reminder = Reminder + Divisor**	**1111** 1100
		(Reminder < 0); Shift left, R0 = 0	1111 100**0**
3	0010	**Reminder = Reminder + Divisor**	**0001** 1000
		(Reminder > 0); Shift left, R0 = 1	0011 000**1**
4	0010	Reminder = Reminder - Divisor	**0001** 0001
		(Reminder > 0); Shift left, R0 = 1	0010 001**1**
Done	0010	Shift left half of reminder right by 1	**0001** 0011

Floating point numbers (a brief look)

We need a way to represent

• Numbers with fractions, e.g., 3.1416
• Very small numbers, e.g., 0.000000001
• Very large numbers, e.g., 3.15576 x 109

Representation

• Sign, exponent, significand: (–1)sign x significand x 2exponent
• More bits for significand gives more accuracy
• More bits for exponent increases range

IEEE 754 floating point standard

• Single precision: 8 bit exponent, 23 bit significand
• Double precision: 11 bit exponent, 52 bit significand

Floating point add/subtract

To add/sub two numbers
• We first compare the two exponents
• Select the higher of the two as the exponent of result
• Select the significand part of lower exponent number and shift it right by the amount

equal to the difference of two exponent
• Remember to keep two shifted out bit and a guard bit
• Add/sub the signifand as required according to operation and signs of operands
• Normalize significand of result adjusting exponent
• Round the result (add one to the least significant bit to be retained if the first bit being thrown
away is a 1
• Re-normalize the result

Floating point multiply

To multiply two numbers

• Add the two exponent (remember access 127 notation)
• Produce the result sign as exor of two signs
• Multiply significand portions
• Results will be 1x.xxxxx… or 01.xxxx….
• In the first case shift result right and adjust exponent
• Round off the result
• This may require another normalization step

Floating point division

To divide two numbers

• Subtract divisor's exponent from the dividend's exponent (remember access 127 notation)
• Produce the result sign as exor of two signs
• Divide dividend's significand by divisor's significand portions
• Results will be 1.xxxxx… or 0.1xxxx….
• In the second case shift result left and adjust exponent
• Round off the result
• This may require another normalization step

UNIT II - BASIC PROCESSING UNIT

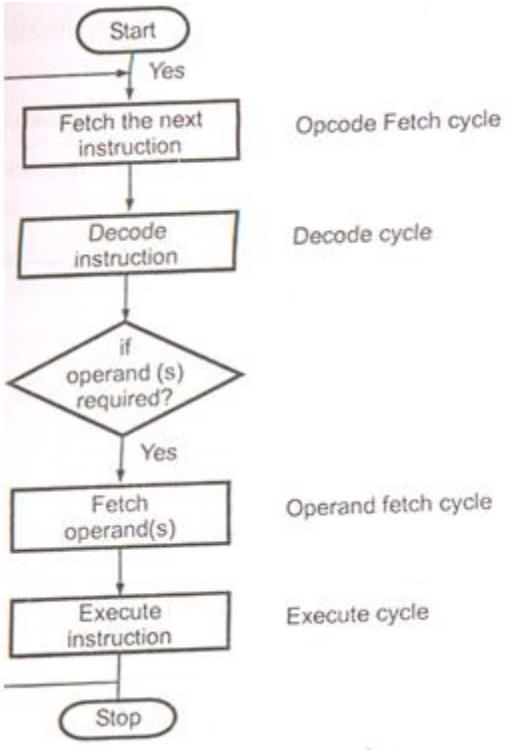
Basic fundamental concepts

Execution of one instruction requires the following three steps to be performed by the CPU:

1. Fetch the contents of the memory location pointed at by the PC. The contents of this location are intepreted as an instruction to be executed. Hence, they are stored in the instruction register (IR). Simbolically, this can be written as:
   IR ←[[PC]]
2. Assuming that the memory is byte addressable, increment the contents of the PC by 4, that is
   PC ← [PC] + 4
3. Carry out the actions specified by the instruction stored in the IR

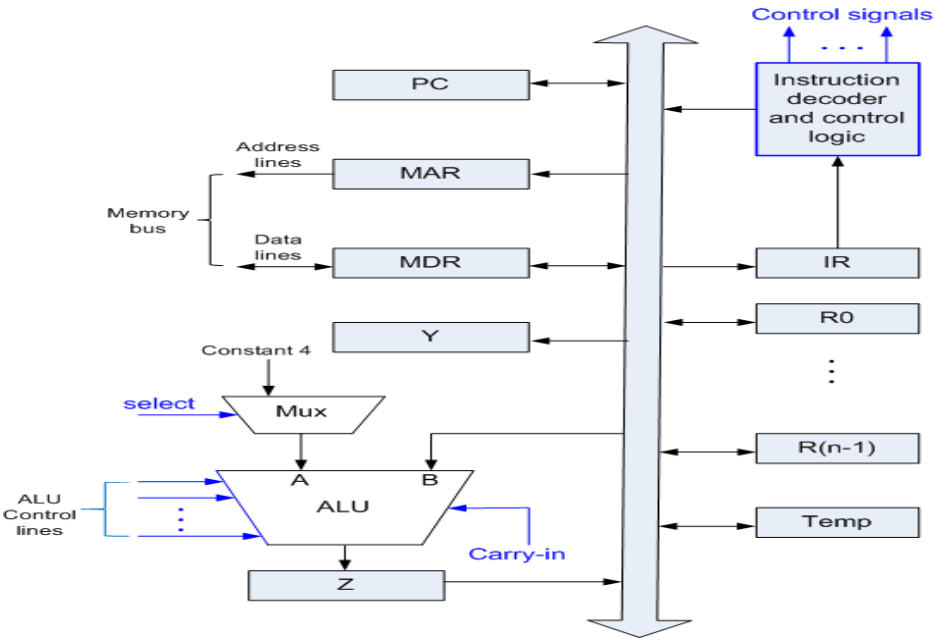
- ✓ But, in cases where an instruction occupies more than one word, steps 1 and 2 must be repeated as many times as necessary to fetch the complete instruction.
- ✓ Two first steps are ussually referred to as the fetch phase.
- ✓ Step 3 constitutes the execution phase

Fig: 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



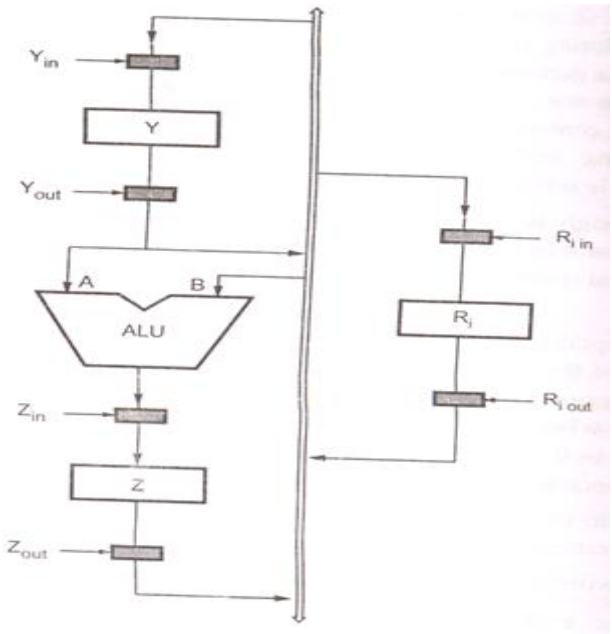
- The registers Y,Z and Temp are used only by the processor unit for temporary storage during the execution of some instructions.
- These registers are never used for storing data generated by one instruction for later use by another instruction.
- The programmer cannot access these registers.
- The 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.

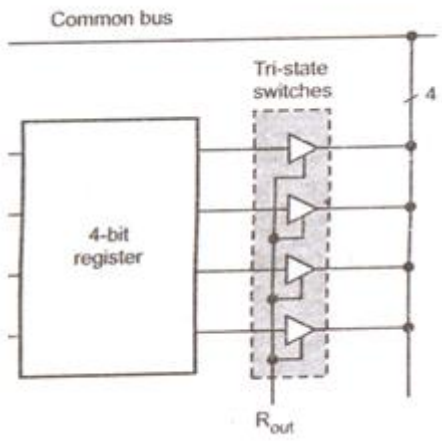
Fig: Input and Output Gating for the Registers



The input and output gates are nothing but the electronic swithches 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

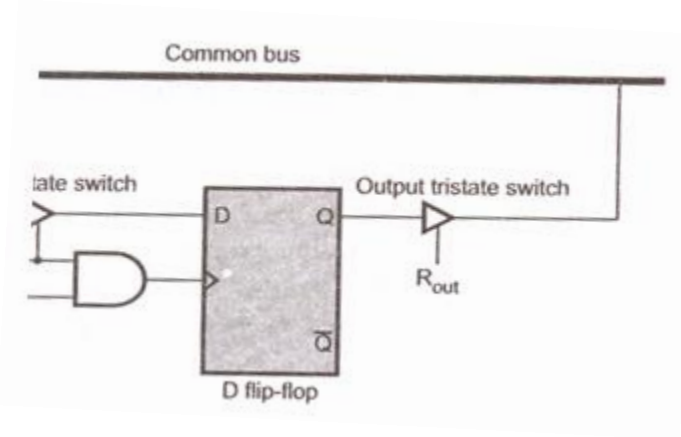


Consider that we have transfer data from register R_1 to R_2

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



- 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}

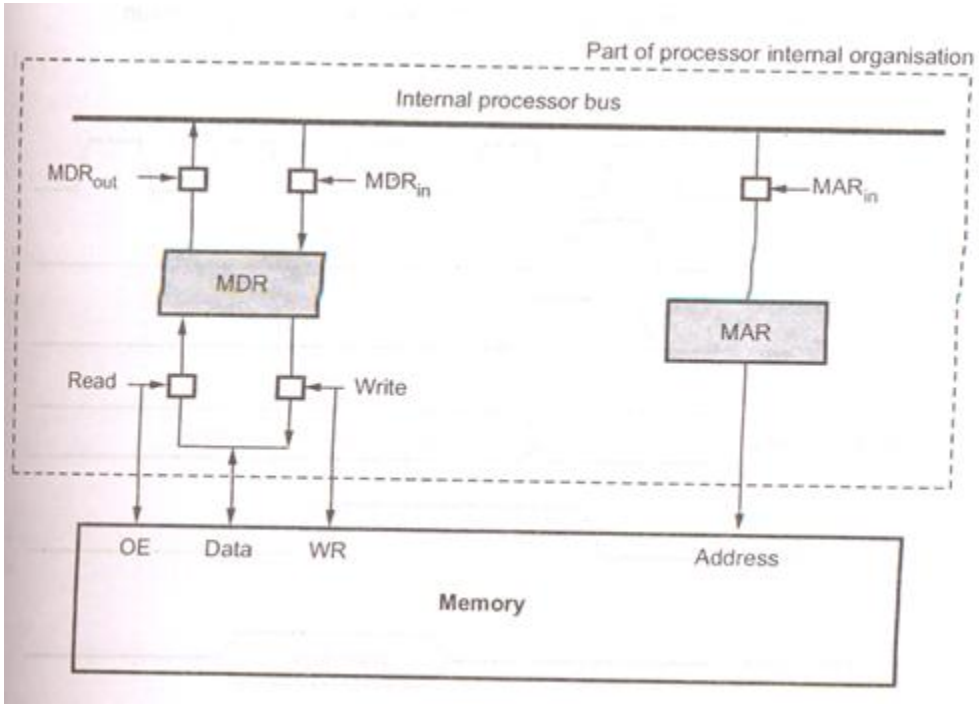
Step 1: contents from register R_1 are loaded into register Y.

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.



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.

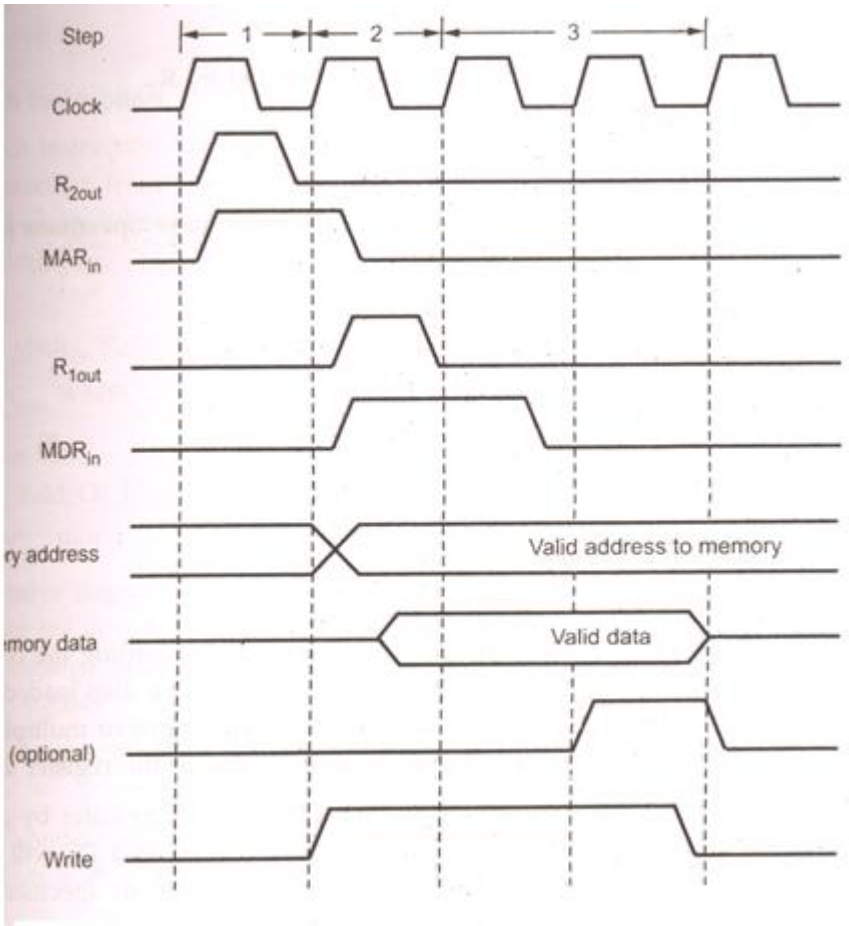
The actions needed to execute this instruction are as follows**:**

a) MAR⟵ [R_2]
b) MDR⟵ [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

Fig: 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).

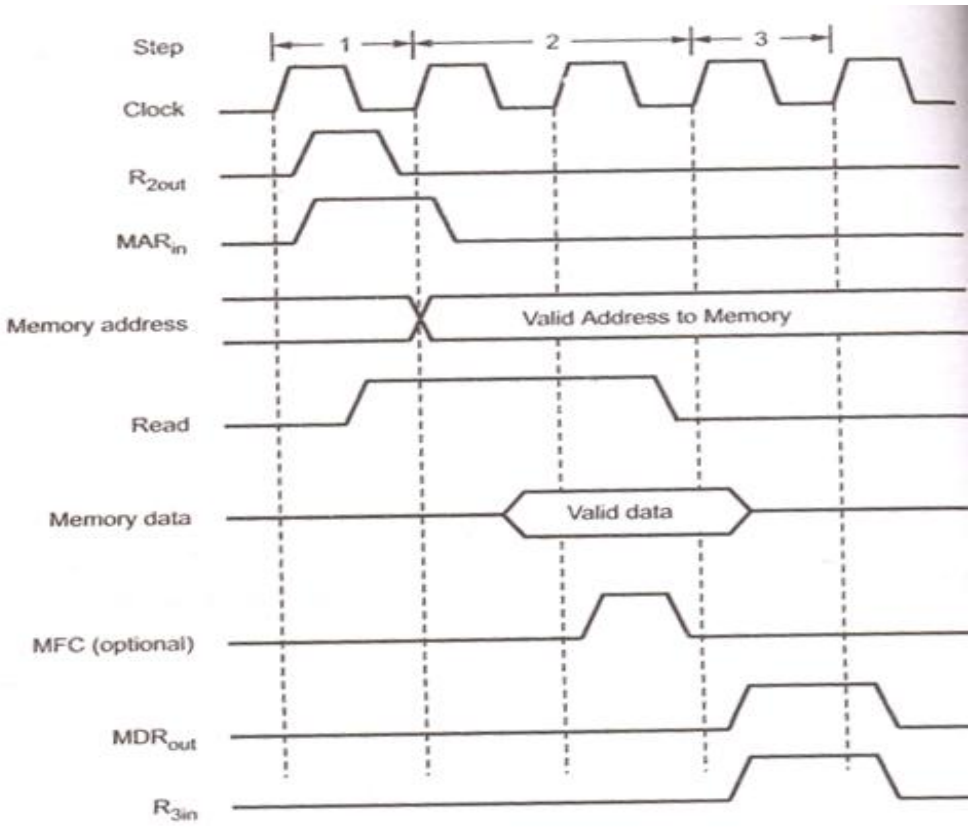
The actions needed to execute this instruction are as follows:

a) MAR ← [R_2]
b) Activate the control signal to perform the Read operation
c) Load MDR from the memory bus
d) R_3 ← [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 : MOVE R_3,(R_2)



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:

- 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.

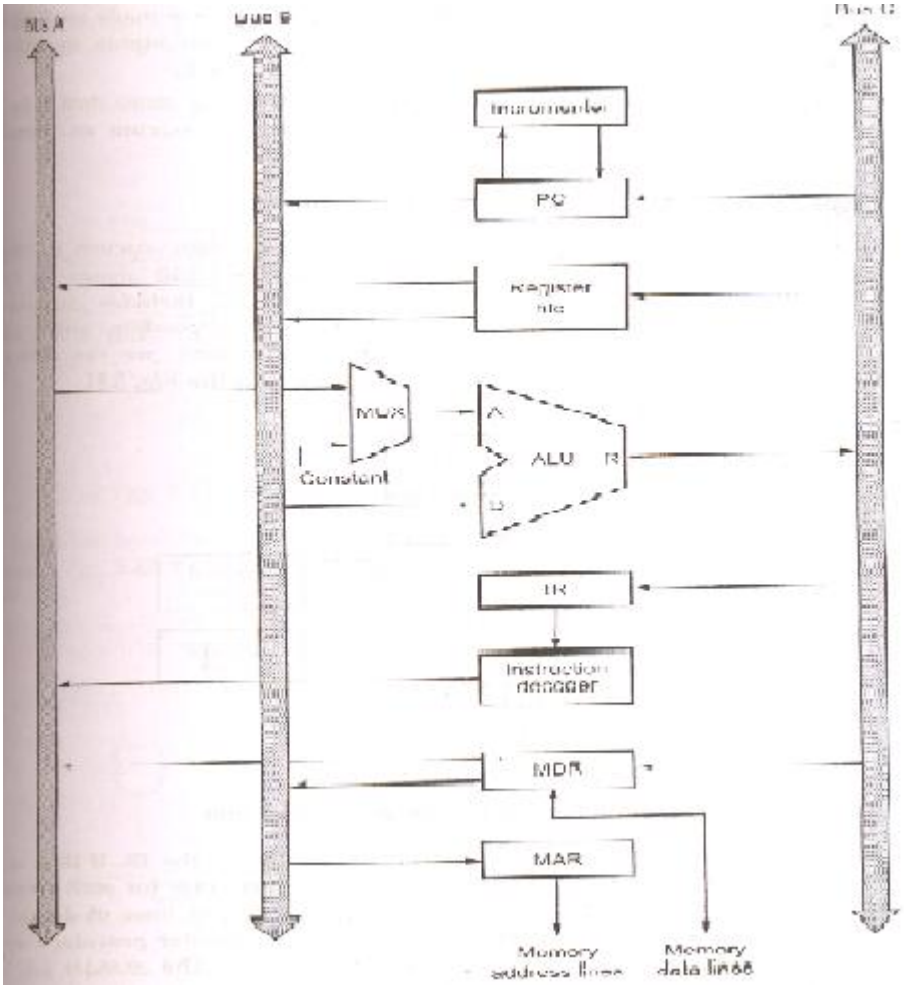
  Step 3: The instruction code is transferred from MDR to IR register. At the beginning of step 4, the instruction decoder interprets the contents of the IR.

  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.

Fig:Multiple Bus Organisation



Hardwired 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:Typical hardwired control

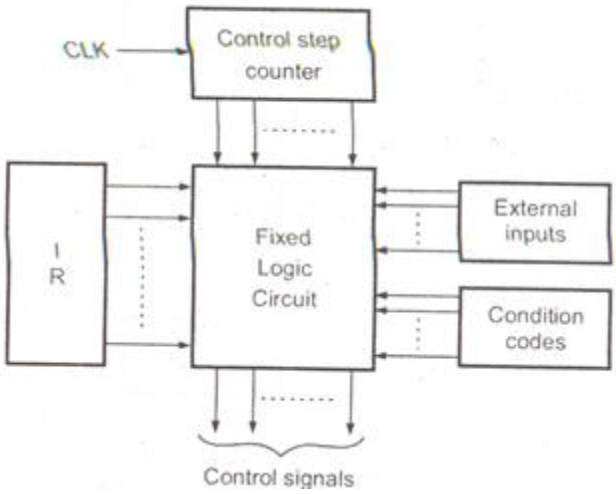
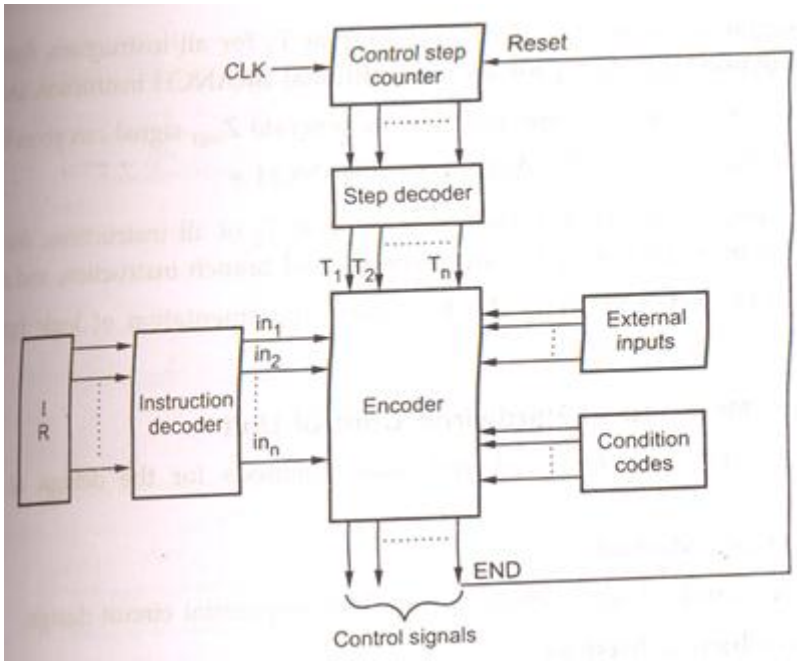


Fig:Detailed block diagram for hardwired control



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 which 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}

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

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

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

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.

Interger unit – To process integer data

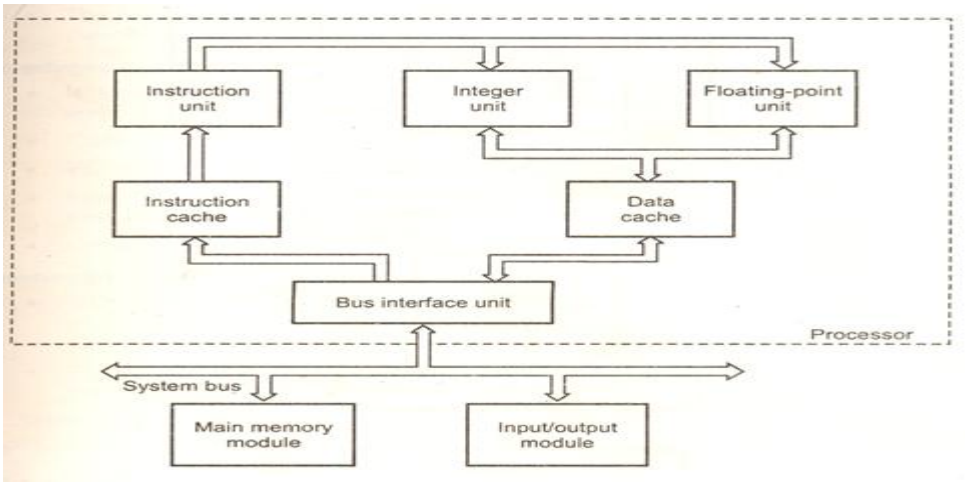
Floating unit – To process floating –point data

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: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 microoperations.

Each microoperation is associated with a specific set of control lines which, when activated, causes that microoperation 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.

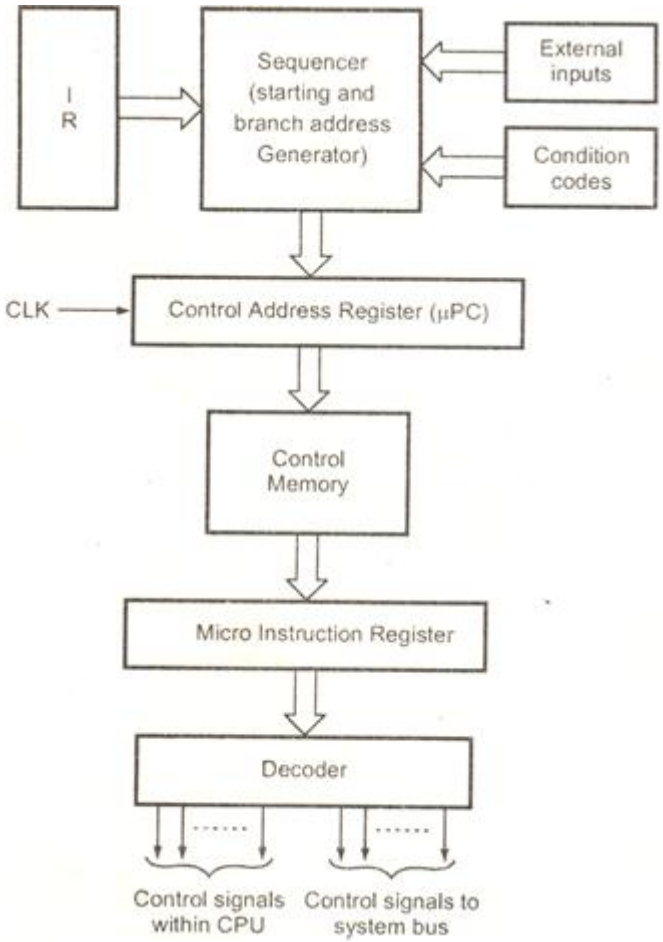
Microprogramming is a method of control unit design in which the control signal memory CM.

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 microoperations designed to control specific operation, such as addition , multiplication is called a microprogram.

The microprograms for all instructions are stored in the control memory.

Fig: 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.

Micro instruction

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.

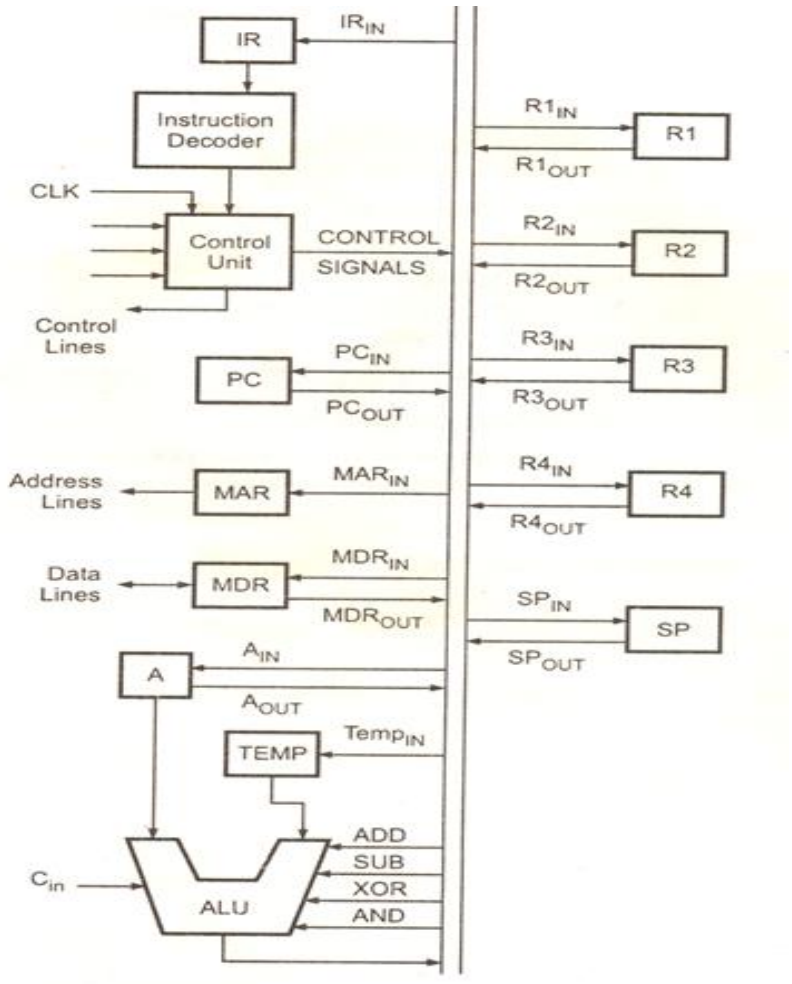
It is not at all necessary to use all 46 bits for every microinstruction because by using grouping of control signals we minimize number of bits for microinstruction.

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: Single Bus CPU structure with control signals



- 46 control signals can be grouped in 7 different groups.
- 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 organisation

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

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.

ii. 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 microinstructions.

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

Nanoprogramming

– Use a 2-level control storage organization
– Top level is a vertical format memory
» Output of the top level memory drives the address register of the bottom (nano-level) memory

Nanomemory uses the horizontal format

» Produces the actual control signal outputs

– The advantage to this approach is significant saving in control memory *size* (bits)

– Disadvantage is more complexity and slower operation (doing 2 memory accesses fro each microinstruction)

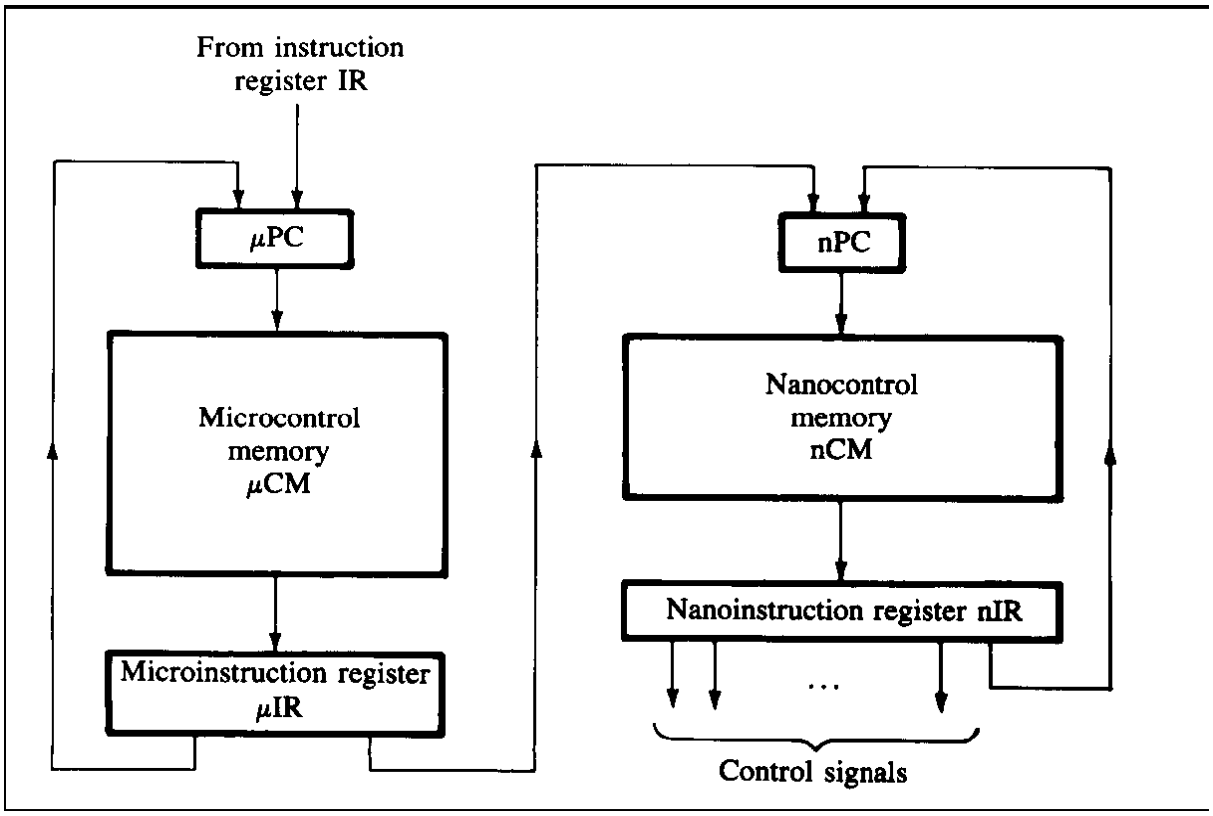
NANOSTORAGE AND NANOINSTRUCTION

The decoder circuits in a vertical microprogram storage organization can be replaced by a ROM
=> Two levels of control storage
First level - *Control Storage*
Second level - *Nano Storage*

Two-level microprogram

First level
-*Vertical* format Microprogram
Second level
-*Horizontal* format Nanoprogram
- Interprets the microinstruction fields, thus converts a vertical microinstruction format into a horizontal nanoinstruction format.

Usually, the microprogram consists of a large number of short microinstructions, while the nanoprogram contains fewer words with longer nanoinstructions.



Example: Supppose that a system is being designed with 200 control points and 2048 microinstructions

Assume that only 256 different combinations of control points are ever used

A single-level control memory would require 2048x200=409,600 storage bits

A nanoprogrammed system would use

» Microstore of size 2048x8=16k

» Nanostore of size 256x200=51200

» Total size = 67,584 storage bits

Nanoprogramming has been used in many CISC microprocessors

UNIT III PIPELINING

The Pipeline Defined

John Hayes provides a definition of a pipeline as it applies to a computer processor.

"A pipeline processor consists of a sequence of processing circuits, called segments or stages, through which a stream of operands can be passed.
"Partial processing of the operands takes place in each segment.
"... a fully processed result is obtained only after an operand set has passed through the entire pipeline."

In everyday life, people do many tasks in stages. For instance, when we do the laundry, we place a load in the washing machine. When it is done, it is transferred to the dryer and another load is placed in the washing machine. When the first load is dry, we pull it out for folding or ironing, moving the second load to the dryer and start a third load in the washing machine. We proceed with folding or ironing of the first load while the second and third loads are being dried and washed, respectively. We may have never thought of it this way but we do laundry by **pipeline processing**.

A Pipeline

is a series of stages, where some work is done at each stage. The work is not finished until it has passed through all stages.

Let us review Hayes' definition as it pertains to our laundry example. The washing machine is one "sequence of processing circuits" or a **stage**. The second is the dryer. The third is the folding or ironing stage.
Partial processing takes place in each stage. We certainly aren't done when the clothes leave the washer. Nor when they leave the dryer, although we're getting close. We must take the third step and fold (if we're lucky) or iron the cloths. The "fully processed result" is obtained only after the operand (the load of clothes) has passed through the entire pipeline.

We are often taught to take a large task and to divide it into smaller pieces. This may make a unmanageable complex task into a series of more tractable smaller steps. In the case of manageable tasks such as the laundry example, it allows us to speed up the task by doing it in overlapping steps.

ROLE OF CACHE MEMORY

Each stage in a pipeline is expected to complete its operation in one clock cycle. Hence, the clock period should be sufficiently long to complete the task being performed in any stage. If different units require different amounts of time, the clock period must allow the longest task to be completed. A unit that completes its task early is idle for the remainder of

the clock period. Hence, pipelining is most effective in improving performance if the tasks being performed in different stages require about the same amount of time.

This consideration is particularly important for the instruction fetch step, which is assigned one clock period. The clock cycle has to be equal to or greater than the time needed to complete a fetch operation. However, the access time of the main memory may be as much as ten times greater than the time needed to perform basic pipeline stage operations inside the processor, such as adding two numbers. Thus, if each instruction fetches required access to the main memory, pipelining would be of little value.

The use of cache memories solves the memory access problem. In particular, when a cache is included on the same chip as the processor, access time to the cache is usually the same as the time needed to perform other basic operations inside the processor. This makes it possible to divide instruction fetching and processing into steps that are more or less equal in duration. Each of these steps is performed by a different pipeline stage, and the clock period is chosen to correspond to the longest one.

PIPELINE PERFORMANCE

The pipelined processor completes the processing of one instruction in each clock cycle, which means that the rate of instruction processing is four times that of sequential operation. The potential increase in performance resulting from pipelining is proportional to the number of pipeline stages. However, this increase would be achieved only if pipelined operation could be sustained without interruption throughout program execution. Unfortunately, this is not the case.

For a 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 example, 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. Figure 3.3 shows an example in which the operation specified in instruction I2 requires three 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, stages 1 are blocked from accepting new instructions because the information in B1 cannot be overwritten. Thus, steps D4 and F5 must be postponed as shown.

Pipelined operation is said to have been stalled for two clock cycles. Normal pipelined operation resumes in cycle 7. Any condition that causes the pipeline to stall is called a hazard. We have just seen an example of a data hazard. A data hazard is any condition 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.

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. The effect of a cache miss on pipelined operation is illustrated in Figure 3.4. Instruction I1 is fetched from the cache in cycle 1, and its execution proceeds normally. However, the fetch operation for instruction I2, which is started in cycle 2, results in a cache miss. The instruction fetch unit must now suspend any further fetch requests and wait for I2 to arrive. We assume that instruction I2 is received and loaded into buffer B1 at the end of cycle 5. The pipeline resumes its normal operation at that point.

It gives the function performed by each pipeline stage in each clock cycle. Note that the Decode unit is idle in cycles 3 through 5, the Execute unit is idle in cycles 4 through 6, and the Write unit is idle in cycles 5 through 7. Such idle periods are called stalls. They are also often referred to as bubbles in the pipeline. Once created as a result of a delay in one of the pipeline stages, a bubble moves downstream until it reaches the last unit.

A third type of hazard that may be encountered in pipelined operation is known as a structural hazard. This is the situation when two instructions require the use of a given hardware resource at the same time. The most common case in which this hazard may arise is in access to memory. One instruction may need to access memory as part of the Execute or Write stage while another instruction is being fetched. If instructions and data reside in the same cache unit, only one instruction can proceed and the other instruction is delayed. Many processors use separate instruction and data caches to avoid this delay.

An example of a structural hazard is shown in Figure 3.5. This figure shows how the load instruction

Load X(R1),R2

can be accommodated in our example 4-stage pipeline. The memory address, X+[R1], is computed in stepE2 in cycle 4, then memory access takes place in cycle 5. The operand read from memory is written into register R2 in cycle 6. This means that the execution step of this instruction takes two clock cycles (cycles 4 and 5). It causes the pipeline to stall for one cycle, because both instructions I2 and I3 require access to the register file in cycle 6. Even though the instructions and their data are all available, the pipeline isStalled because one hardware resource, the register file, cannot handle two operations at once. If the register file had two input ports, that is, if it allowed two simultaneous write operations, the pipeline would not be stalled. In general, structural hazards are avoided by providing sufficient hardware resources on the processor chip.

It is important to understand that pipelining does not result in individual instructions being executed faster; rather, it is the throughput that increases, where throughput is measured by the rate at which instruction execution is completed. Any time one of the stages in the pipeline cannot complete its operation in one clock cycle, the pipeline stalls, and some degradation in performance occurs. Thus, the performance level of one instruction completion in each clock cycle is actually the upper limit for the throughput achievable in a pipelined processor organized .

An important goal in designing processors is to identify all hazards that may cause the pipeline to stall and to find ways to minimize their impact. In the following sections we discuss various hazards, starting with data hazards, followed by control hazards. In each case we present some of the techniques used to mitigate their negative effect on performance. We return to the issue of performance assessment in Section 3.8.

DATA HAZARDS

A data hazard is a situation in which the pipeline is stalled because the data to be operated on are delayed for some reason. We will now examine the issue of availability of data in some detail.

Consider a program that contains two instructions, I1 followed by I2. When this program is executed in a pipeline, the execution of I2 can begin before the execution of I1 is completed. This means that the results generated by I1 may not be available for use by I2. We must ensure that the results obtained when instructions are executed in a pipelined processor are identical to those obtained when the same instructions are executed sequentially. The potential for obtaining incorrect results when operations are performed concurrently can be demonstrated by a simple example. Assume that A=5, and consider the following two operations:

$$A \leftarrow 3 + A$$

$$B \leftarrow 4 \times A$$

When these operations are performed in the order given, the result is B = 32. But if they are performed concurrently, the value of A used in computing B would be the original value, 5, leading to an incorrect result. If these two operations are performed by instructions in a program, then the instructions must be executed one after the other, because the data used in the second instruction depend on the result of the first instruction. On the other hand, the two operations

$$A \leftarrow 5 \times C$$

$$B \leftarrow 20 + C$$

can be performed concurrently, because these operations are independent.

This example illustrates a basic constraint that must be enforced to guarantee correct results. When two operations depend on each other, they must be performed sequentially in the correct order. This rather obvious condition has far-reaching consequences. Understanding its implications is the key to understanding the variety of design alternatives and trade-offs encountered in pipelined computers.

The data dependency just described arises when the destination of one instruction is used as a source in the next instruction. For example, the two instructions

```
Mul R2,R3,R4

Add R5,R4,R6
```

give rise to a data dependency. The result of the multiply instruction is placed into register R4, which in turn is one of the two source operands of the Add instruction. Assuming that the multiply operation takes one clock cycle to complete; execution would proceed as shown in Figure 3.6. As the Decode unit decodes the Add instruction in cycle 3, it realizes that R4 is used as a source operand. Hence, the D step of that instruction cannot be completed until the W step of the multiply instruction has been completed. Completion of step D2 must be delayed to clock cycle 5, and is shown as step D2A in the figure. Instruction I3 is fetched in cycle 3, but its decoding must be delayed because step D3 cannot precede D2. Hence, pipelined execution is stalled for two cycles.

OPERAND FORWARDING

The data hazard just described arises because one instruction, instruction I2, is waiting for data to be written in the register file. However, these data are available at the output of the ALU once the Execute stage completes step E1. Hence, the delay can be reduced, or possibly eliminated, if we arrange for the result of instruction I1 to be forwarded directly for use in step E2.It shows a part of the processor datapath involving the ALU and the register file. This arrangement is similar to the three-bus structure ,except that registers SRC1, SRC2, and RSLT have been added. These registers constitute the interstage buffers needed for pipelined operation. Registers SRC1 and SRC2 are part of buffer B2 and RSLT is part of B3. The data forwarding mechanism is provided by the blue connection lines. The two multiplexers connected at the inputs to the ALU allow the data on the destination bus to be selected instead of the contents of either the SRC1 or SRC2 register.

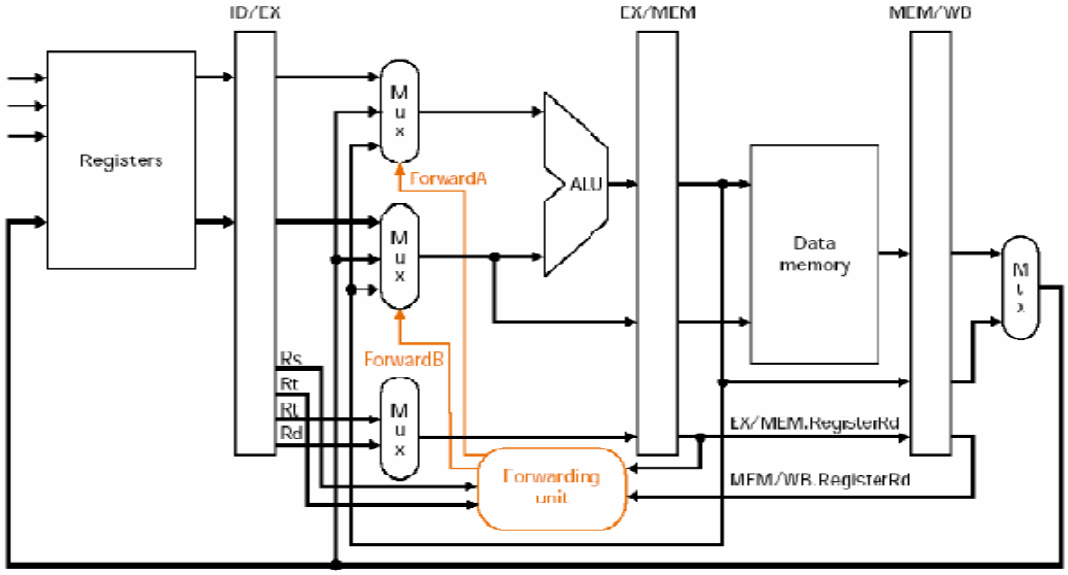
When the instructions in Figure 3.6 are executed in the datapath of Figure 3.7, the operations performed in each clock cycle are as follows. After decoding instruction I2 and detecting the data dependency, a decision is made to use data forwarding. The operand not involved in the dependency, register R2, is read and loaded in register SRC1 in clock cycle 3. In the next clock cycle, the product produced by instruction I1 is available in register RSLT, and because of the forwarding connection, it can be used in step E2. Hence, execution of I2 proceeds without interruption.

HANDLING DATA HAZARDS IN SOFTWARE

The data dependency is discovered by the hardware while the instruction is being decoded. The control hardware delays reading register R4 until cycle 5, thus introducing a 2-cycle stall unless operand forwarding is used. An alternative approach is to leave the task of detecting data dependencies and dealing with them to the software. In this case, the compiler can introduce the two-cycle delay needed between instructions I1 and I2 by inserting NOP (No-operation) instructions, as follows:

```
I1: Mul R2,R3,R4
    NOP
    NOP
I2: Add R5,R4,R6
```

Fig: Operand forwarding



If the responsibility for detecting such dependencies is left entirely to the software, the compiler must insert the NOP instructions to obtain a correct result. This possibility illustrates the close link between the compiler and the hardware. A particular feature can be either implemented in hardware or left to the compiler. Leaving tasks such as inserting NOP instructions to the compiler leads to simpler hardware. Being aware of the need for a delay, the compiler can attempt to reorder instructions to perform useful tasks in the NOP slots, and thus achieve better performance. On the other hand, the insertion of NOP instructions leads to larger code size. Also, it is often the case that a given processor architecture has several hardware implementations, offering different features. NOP instructions inserted to satisfy the requirements of one implementation may not be needed and, hence, would lead to reduced performance on a different implementation.

SIDE EFFECTS

The data dependencies encountered in the preceding examples are explicit and easily detected because the register involved is named as the destination in instruction I1 and as a source in I2. Sometimes an instruction changes the contents of a register other than the one named as the destination. An instruction that uses an autoincrement or autodecrement addressing mode is an example. In addition to storing new data in its destination location, the instruction changes the contents of a source register used to access one of its operands. All the precautions needed to handle data dependencies involving the destination location must also be applied to the registers affected by an autoincrement or autodecrement operation. 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. For example, stack instructions, such as push and pop, produce similar side effects because they implicitly use the autoincrement and autodecrement addressing modes.

Another possible side effect involves the condition code flags, which are used by instructions such as conditional branches and add-with-carry. Suppose that registers R1 and R2 hold a double-precision integer number that we wish to add to another double precision number in registers R3 and R4. This may be accomplished as follows:

Add R1,R3

Add With Carry R2,R4

An implicit dependency exists between these two instructions through the carry flag. This flag is set by the first instruction and used in the second instruction, which performs the operation

$$R4 \leftarrow [R2] + [R4] + carry$$

Instructions that have side effects give rise to multiple data dependencies, which lead to a substantial increase in the complexity of the hardware or software needed to resolve them. For this reason, instructions designed for execution on pipelined hardware should have few side effects. Ideally, only the contents of the destination location, either a register or a memory location, should be affected by any given instruction. Side effects, such as setting the condition code flags or updating the contents of an address pointer, should be kept to a minimum. However, Chapter 2 showed that the autoincrement and autodecrement addressing modes are potentially useful. Condition code flags are also needed for recording such information as the generation of a carry or the ccurrence of overflow in an arithmetic operation. we show how such functions can be provided by other means that are consistent with a pipelined organization and with the requirements of optimizing compilers.

INSTRUCTION HAZARDS

The purpose of the instruction fetch unit is to supply the execution units with a steady stream of instructions. Whenever this stream is interrupted, the pipeline stalls, as Figure 3.4 illustrates for the case of a cache miss. A branch instruction may also cause the pipeline to stall. We will now examine the effect of branch instructions and the techniques that can be used for mitigating their impact. We start with unconditional branches.

UNCONDITIONAL BRANCHES

A sequence of instructions being executed in a two-stage pipeline. Instructions I1 to I3 are stored at successive memory addresses, and I2 is a branch instruction. Let the branch target be instruction Ik . In clock cycle 3, the fetch operation for instruction I3 is in progress at the same time that the branch instruction is being decoded and the target address computed. In clock cycle 4, the processor must discard I3, which has been incorrectly fetched, and fetch instruction Ik . In the meantime, the hardware unit responsible for the Execute (E) step must be told to do nothing during that clock period. Thus, the pipeline is stalled for one clock cycle.

The time lost as a result of a branch instruction is often referred to as the branch penalty. The branch penalty is one clock cycle. For a longer pipeline, the branch penalty may

be higher. The effect of a branch instruction on a four-stage pipeline. We have assumed that the branch address is computed in step E2. Instructions I3 and I4 must be discarded, and the target instruction, Ik , is fetched in clock cycle 5. Thus, the branch penalty is two clock cycles.

Reducing the branch penalty requires the branch address to be computed earlier in the pipeline. Typically, the instruction fetch unit has dedicated hardware to identify a branch instruction and compute the branch target address as quickly as possible after an instruction is fetched. With this additional hardware, both of these tasks can be performed in step D2, leading to the sequence of events shown in Figure 3.9b. In this case, the branch penalty is only one clock cycle.

Instruction Queue and Prefetching

Either a cache miss or a branch instruction stalls the pipeline for one or more clock cycles. To reduce the effect of these interruptions, many processors employ sophisticated fetch units that can fetch instructions before they are needed and put them in a queue. Typically, the instruction queue can store several instructions. A separate unit, which we call the dispatch unit, takes instructions from the front of the queue and sends them to the execution unit. This leads to the organization shown in Figure 3.10. The dispatch unit also performs the decoding function.

To be effective, the fetch unit must have sufficient decoding and processing capability to recognize and execute branch instructions. It attempts to keep the instruction queue filled at all times to reduce the impact of occasional delays when fetching instructions. When the pipeline stalls because of a data hazard, for example, the dispatch unit is not able to issue instructions from the instruction queue. However, the fetch unit continues to fetch instructions and add them to the queue. Conversely, if there is a delay in fetching instructions because of a branch or a cache miss, the dispatch unit continues to issue instructions from the instruction queue.

We have assumed that initially the queue contains one instruction. Every fetch operation adds one instruction to the queue and every dispatch operation reduces the queue length by one. Hence, the queue length remains the same for the first four clock cycles. (There is both an F and a D step in each of these cycles.) Suppose that instruction I1 introduces a 2-cycle stall. Since space is available in the queue, the fetch unit continues to fetch instructions and the queue length rises to 3 in clock cycle 6.

Instruction I5 is a branch instruction. Its target instruction, Ik , is fetched in cycle 7, and instruction I6 is discarded. The branch instruction would normally cause a stall in cycle 7 as a result of discarding instruction I6. Instead, instruction I4 is dispatched from the queue to the decoding stage. After discarding I6, the queue length drops to 1 in cycle 8. The queue length will be at this value until another stall is encountered.

Instructions I1, I2, I3, I4, and Ik complete execution in successive clock cycles. Hence, the branch instruction does not increase the overall execution time. This is because the instruction fetch unit has executed the branch instruction (by computing the branch

address) concurrently with the execution of other instructions. This technique is referred to as branch folding.

Note that branch folding occurs only if at the time a branch instruction is encountered, at least one instruction is available in the queue other than the branch instruction. If only the branch instruction is in the queue, execution would proceed. Therefore, it is desirable to arrange for the queue to be full most of the time, to ensure an adequate supply of instructions for processing. This can be achieved by increasing the rate at which the fetch unit reads instructions from the cache. In many processors, the width of the connection between the fetch unit and the instruction cache allows reading more than one instruction in each clock cycle. If the fetch unit replenishes the instruction queue quickly after a branch has occurred, the probability that branch folding will occur increases.

Having an instruction queue is also beneficial in dealing with cache misses. When a cache miss occurs, the dispatch unit continues to send instructions for execution as long as the instruction queue is not empty. Meanwhile, the desired cache block is read from the main memory or from a secondary cache. When fetch operations are resumed, the instruction queue is refilled. If the queue does not become empty, a cache miss will have no effect on the rate of instruction execution.

In summary, the instruction queue mitigates the impact of branch instructions on performance through the process of branch folding. It has a similar effect on stalls caused by cache misses. The effectiveness of this technique is enhanced when the instruction fetch unit is able to read more than one instruction at a time from the instruction cache.

CONDITIONAL BRANCHES AND BRANCH PREDICTION

A conditional branch instruction introduces the added hazard caused by the dependency of the branch condition on the result of a preceding instruction. The decision to branch cannot be made until the execution of that instruction has been completed.

Branch instructions occur frequently. In fact, they represent about 20 percent of the dynamic instruction count of most programs. (The dynamic count is the number of instruction executions, taking into account the fact that some program instructions are executed many times because of loops.) Because of the branch penalty, this large percentage would reduce the gain in performance expected from pipelining. Fortunately, branch instructions can be handled in several ways to reduce their negative impact on the rate of execution of instructions.

Delayed Branch

The processor fetches instruction I3 before it determines whether the current instruction, I2, is a branch instruction. When execution of I2 is completed and a branch is to be made, the processor must discard I3 and fetch the instruction at the branch target. The location following a branch instruction is called a branch delay slot. There may be more than one branch delay slot, depending on the time it takes to execute a branch instruction. The

instructions in the delay slots are always fetched and at least partially executed before the branch decision is made and the branch target address is computed.

A technique called delayed branching can minimize the penalty incurred as a result of conditional branch instructions. The idea is simple. The instructions in the delay slots are always fetched. Therefore, we would like to arrange for them to be fully executed whether or not the branch is taken. The objective is to be able to place useful instructions in these slots. If no useful instructions can be placed in the delay slots, these slots must be filled with NOP instructions. This situation is exactly the same as in the case of data dependency.

Register R2 is used as a counter to determine the number of times the contents of register R1 are shifted left. For a processor with one delay slot, the instructions can be reordered. The shift instruction is fetched while the branch instruction is being executed. After evaluating the branch condition, the processor fetches the instruction at LOOP or at NEXT, depending on whether the branch condition is true or false, respectively. In either case, it completes execution of the shift instruction. The sequence of events during the last two passes in the loop. Pipelined operation is not interrupted at any time, and there are no idle cycles. Logically, the program is executed as if the branch instruction were placed after the shift instruction. That is, branching takes place one instruction later than where the branch instruction appears in the instruction sequence in the memory, hence the name "delayed branch."

The effectiveness of the delayed branch approach depends on how often it is possible to reorder instructions. Experimental data collected from many programs indicate that sophisticated compilation techniques can use one branch delay slot in as many as 85 percent of the cases. For a processor with two branch delay slots, the compiler attempts to find two instructions preceding the branch instruction that it can move into the delay slots without introducing a logical error. The chances of finding two such instructions are considerably less than the chances of finding one. Thus, if increasing the number of pipeline stages involves an increase in the number of branch delay slots, the potential gain in performance may not be fully realized.

Branch Prediction

Another technique for reducing the branch penalty associated with conditional branches is to attempt to predict whether or not a particular branch will be taken. The simplest form of branch prediction is to assume that the branch will not take place and to continue to fetch instructions in sequential address order. Until the branch condition is evaluated, instruction execution along the predicted path must be done on a speculative basis. Speculative execution means that instructions are executed before the processor is certain that they are in the correct execution sequence. Hence, care must be taken that no processor registers or memory locations are updated until it is confirmed that these instructions should indeed be executed. If the branch decision indicates otherwise, the

Instructions and all their associated data in the execution units must be purged, and the correct instructions fetched and executed.

The Compare instruction is followed by a Branch>0 instruction. Branch prediction takes place in cycle 3, while instruction I3 is being fetched. The fetch unit predicts that the branch will not be taken, and it continues to fetch instruction I4 as I3enters the Decode stage. The results of the compare operation are available at the end of cycle 3. Assuming that they are forwarded immediately to the instruction fetch unit, the branch condition is evaluated in cycle 4. At this point, the instruction fetch unit realizes that the prediction was incorrect, and the two instructions in the execution pipe are purged. A new instruction, Ik , is fetched from the branch target address in clock cycle 5.

If branch outcomes were random, then half the branches would be taken. Then the simple approach of assuming that branches will not be taken would save the time lost to conditional branches 50 percent of the time. However, better performance can be achieved if we arrange for some branch instructions to be predicted as taken and others as not taken, depending on the expected program behavior. For example, a branch instruction at the end of a loop causes a branch to the start of the loop for every pass through the loop except the last one. Hence, it is advantageous to assume that this branch will be taken and to have the instruction fetch unit start to fetch instructions at the branch target address. On the other hand, for a branch instruction at the beginning of a program loop, it is advantageous to assume that the branch will not be taken.

A decision on which way to predict the result of the branch may be made in hardware by observing whether the target address of the branch is lower than or higher than the address of the branch instruction. A more flexible approach is to have the compiler decide whether a given branch instruction should be predicted taken or not taken. The branch instructions of some processors, such as SPARC, include a branch prediction bit, which is set to 0 or 1 by the compiler to indicate the desired behavior. The instruction fetch unit checks this bit to predict whether the branch will be taken or not taken.

With either of these schemes, the branch prediction decision is always the same every time a given instruction is executed. Any approach that has this characteristic is called static branch prediction. Another approach in which the prediction decision may change depending on execution history is called dynamic branch prediction.

Dynamic Branch Prediction

The objective of branch prediction algorithms is to reduce the probability of making a wrong decision, to avoid fetching instructions that eventually have to be discarded. In dynamic branch prediction schemes, the processor hardware assesses the likelihood of a given branch being taken by keeping track of branch decisions every time that instruction is executed.

In its simplest form, the execution history used in predicting the outcome of a given branch instruction is the result of the most recent execution of that instruction. The processor assumes that the next time the instruction is executed, the result is likely to be the same. Hence, the algorithm may be described by the two-state machine. The two states are:

LT: Branch is likely to be taken
LNT: Branch is likely not to be taken

Suppose that the algorithm is started in state LNT. When the branch instruction is executed and if the branch is taken, the machine moves to state LT. Otherwise, it remains in state LNT. The next time the same instruction is encountered, the branch is predicted as taken if the corresponding state machine is in state LT. Otherwise it is predicted as not taken.

This simple scheme, which requires one bit of history information for each branch instruction, works well inside program loops. Once a loop is entered, the branch instruction that controls looping will always yield the same result until the last pass through the loop is reached. In the last pass, the branch prediction will turn out to be incorrect, and the branch history state machine will be changed to the opposite state. Unfortunately, this means that the next time this same loop is entered, and assuming that there will be more than one pass through the loop, the machine will lead to the wrong prediction.

Better performance can be achieved by keeping more information about execution history. An algorithm that uses 4 states, thus requiring two bits of history information for each branch instruction. The four states are:

ST: Strongly likely to be taken

LT: Likely to be taken

LNT: Likely not to be taken

SNT: Strongly likely not to be taken

Again assume that the state of the algorithm is initially set to LNT. After the branch instruction has been executed, and if the branch is actually taken, the state is changed to ST; otherwise, it is changed to SNT. As program execution progresses and the same instruction is encountered again, the state of the branch prediction algorithm continues to change as shown. When a branch instruction is encountered, the instruction fetch unit predicts that the branch will be taken if the state is either LT or ST, and it begins to fetch instructions at the branch target address. Otherwise, it continues to fetch instructions in sequential address order.

It is instructive to examine the behavior of the branch prediction algorithm in some detail. When in state SNT, the instruction fetch unit predicts that the branch will not be taken. If the branch is actually taken, that is if the prediction is incorrect, the state changes to LNT. This means that the next time the same branch instruction is encountered, the instruction fetch unit will still predict that the branch will not be taken. Only if the prediction is incorrect twice in a row will the state change to ST.

After that, the branch will be predicted as taken.

Let us reconsider what happens when executing a program loop. Assume that the branch instruction is at the end of the loop and that the processor sets the initial state of the algorithm to LNT. During the first pass, the prediction will be wrong (not taken), and hence the state will be changed to ST. In all subsequent passes the prediction will be correct, except

for the last pass. At that time, the state will change to LT. When the loop is entered a second time, the prediction will be correct (branch taken).

We now add one final modification to correct the mispredicted branch at the time the loop is first entered. The cause of the misprediction in this case is the initial state of the branch prediction algorithm. In the absence of additional information about the nature of the branch instruction, we assumed that the processor sets the initial state to LNT. The information needed to set the initial state correctly can be provided by any of the static prediction schemes discussed earlier. Either by comparing addresses or by checking a prediction bit in the instruction, the processor sets the initial state of the algorithm to LNT or LT. In the case of a branch at the end of a loop, the compiler would indicate that the branch should be predicted as taken, causing the initial state to be set to LT. With this modification, branch prediction will be correct all the time, except for the final pass through the loop. Misprediction in this latter case is unavoidable.

The state information used in dynamic branch prediction algorithms may be kept by the processor in a variety of ways. It may be recorded in a look-up table, which is accessed using the low-order part of the branch instruction address. In this case, it is possible for two branch instructions to share the same table entry. This may lead to a branch being mispredicted, but it does not cause an error in execution. Misprediction only introduces a small delay in execution time. An alternative approach is to store the history bits as a tag associated with branch instructions in the instruction cache. We will see in Section 3.7 how this information is handled in the SPARC processor.

INFLUENCE ON INSTRUCTION SETS

We have seen that some instructions are much better suited to pipeline execution than others. For example, instruction side effects can lead to undesirable data dependencies. In this section, we examine the relationship between pipelined execution and machine instruction features. We discuss two key aspects of machine instructions—addressing modes and condition code flags.

ADDRESSING MODES

Addressing modes should provide the means for accessing a variety of data structures simply and efficiently. Useful addressing modes include index, indirect, autoincrement, and autodecrement. Many processors provide various combinations of these modes to increase the flexibility of their instruction sets. Complex addressing modes, such as those involving double indexing, are often encountered.

In choosing the addressing modes to be implemented in a pipelined processor, we must consider the effect of each addressing mode on instruction flow in the pipeline. Two important considerations in this regard are the side effects of modes such as auto increment and autodecrement and the extent to which complex addressing modes cause the pipeline to stall. Another important factor is whether a given mode is likely to be used by compilers.

To compare various approaches, we assume a simple model for accessing operands in the memory. The load instruction Load X(R1),R2 takes five cycles to complete execution, as indicated in Figure 3.5. However, the instruction

Load (R1),R2

can be organized to fit a four-stage pipeline because no address computation is required. Access to memory can take place in stage E. A more complex addressing mode may require several accesses to the memory to reach the named operand. For example, the instruction

Load (X(R1)),R2

may be executed , assuming that the index offset, X, is given in the instruction word. After computing the address in cycle 3, the processor needs to access memory twice — first to read location X+[R1] in clock cycle 4 and then to read location [X+[R1]] in cycle 5. If R2 is a source operand in the next instruction, that instruction would be stalled for three cycles, which can be reduced to two cycles with operand forwarding.

To implement the same Load operation using only simple addressing modes requires several instructions. For example, on a computer that allows three operand addresses, we can use

Add #X,R1,R2

Load (R2),R2

Load (R2),R2

The Add instruction performs the operation R2←X+[R1]. 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 original, single Load instruction.

This example indicates that, in a pipelined processor, complex addressing modes that involve several accesses to the memory do not necessarily lead to faster execution. The main advantage of such modes is that they reduce the number of instructions needed to perform a given task and thereby reduce the program space needed in the main memory. Their main disadvantage is that their long execution times cause the pipeline to stall, thus reducing its effectiveness. They require more complex hardware to decode and execute them. Also, they are not convenient for compilers to work with.

The instruction sets of modern processors are designed to take maximum advantage of pipelined hardware. Because complex addressing modes are not suitable for pipelined execution, they should be avoided. The addressing modes used in modern processors often have the following features:

- Access to an operand does not require more than one access to the memory.
- Only load and store instructions access memory operands.
- The addressing modes used do not have side effects.

Three basic addressing modes that have these features are register, register indirect, and index. The first two require no address computation. In the index mode, the address can be computed in one cycle, whether the index value is given in the instruction or in a register. Memory is accessed in the following cycle. None of these modes has any side effects, with one possible exception. Some architectures, such as ARM, allow the address computed in the index mode to be written back into the index register. This is a side effect that would not be allowed under the guidelines above. Note also that relative addressing can be used; this is a special case of indexed addressing in which the program counter is used as the index register.

The three features just listed were first emphasized as part of the concept of RISC processors.

CONDITION CODES

In many processors, the condition code flags are stored in the processor status register. They are either set or cleared by many instructions, so that they can be tested by subsequent conditional branch instructions to change the flow of program execution. An optimizing compiler for a pipelined processor attempts to reorder instructions to avoid stalling the pipeline when branches or data dependencies between successive instructions occur. In doing so, the compiler must ensure that reordering does not cause a change in the outcome of a computation. The dependency introduced by the condition-code flags reduces the flexibility available for the compiler to reorder instructions.

Consider the sequence of instructions and assume that the execution of the Compare and Branch=0 instructions proceeds . The branch decision takes place in step E2 rather than D2 because it must await the result of the Compare instruction. The execution time of the Branch instruction can be reduced by interchanging the Add and Compare instructions. This will delay the branch instruction by one cycle relative to the Compare instruction. As a result, at the time the Branch instruction is being decoded the result of the Compare instruction will be available and a correct branch decision will be made. There would be no need for branch prediction. However, interchanging the Add and Compare instructions can be done only if the Add instruction does not affect the condition codes.

These observations lead to two important conclusions about the way condition codes should be handled. First, to provide flexibility in reordering instructions, the condition-code flags should be affected by as few instructions as possible. Second, the compiler should be able to specify in which instructions of a program the condition codes are affected and in which they are not. An instruction set designed with pipelining in mind usually provides the desired flexibility. Figure 3.17b shows the instructions reordered assuming that the condition code flags are affected only when this is explicitly stated as part of the instruction OP code. The SPARC and ARM architectures provide this flexibility.

DATAPATH AND CONTROL CONSIDERATIONS

Consider the three-bus structure. To make it suitable for pipelined execution, it can be modified to support a 4-stage pipeline. The resources involved in stages F and E and those used in stages D and W in black. Operations in the data cache may happen during stage E or at a later stage, depending on the addressing mode and the implementation details.

1. There are separate instruction and data caches that use separate address and data connections to the processor. This requires two versions of the MAR register, IMAR for accessing the instruction cache and DMAR for accessing the data cache.

2. The PC is connected directly to the IMAR, so that the contents of the PC can be transferred to IMAR at the same time that an independent ALU operation is taking place.

3. The data address in DMAR can be obtained directly from the register file or from the ALU to support the register indirect and indexed addressing modes.

4. Separate MDR registers are provided for read and write operations. Data can be transferred directly between these registers and the register file during load and store operations without the need to pass through the ALU.

5. Buffer registers have been introduced at the inputs and output of the ALU. These are registers SRC1, SRC2, and RSLT in Figure 3.7. Forwarding connections are not included in Figure 3.18. They may be added if desired.

6. The instruction register has been replaced with an instruction queue, which is loaded from the instruction cache.

7. The output of the instruction decoder is connected to the control signal pipeline. The need for buffering control signals and passing them from one stage to the next along with the instruction is discussed in Section 3.1. This pipeline holds the control signals in buffers B2 and B3.

The following operations can be performed independently in the processor:

- Reading an instruction from the instruction cache
- Incrementing the PC
- Decoding an instruction
- Reading from or writing into the data cache
- Reading the contents of up to two registers from the register file
- Writing into one register in the register file
- Performing an ALU operation

Because these operations do not use any shared resources, they can be performed simultaneously in any combination. The structure provides the flexibility required to implement the four-stage pipeline. For example, let I1, I2, I3, and I4 be a sequence of four instructions. The following actions all happen during clock cycle 4:

- Write the result of instruction I1 into the register file
- Read the operands of instruction I2 from the register file
- Decode instruction I3
- Fetch instruction I4 and increment the PC.

PERFORMANCE CONSIDERATIONS

We pointed that the execution time, T , of a program that has a dynamic instruction count N is given by

$$T = \frac{N^* S}{R}$$

where S is the average number of clock cycles it takes to fetch and execute one instruction,and R is the clock rate. This simple model assumes that instructions are executed one after the other, with no overlap. A useful performance indicator is the instruction throughput, which is the number of instructions executed per second. For sequential execution, the throughput, Ps is given by

$$Ps = R/S$$

we examine the extent to which pipelining increases instruction throughput. The only real measure of performance is the total execution time of a program. Higher instruction throughput will not necessarily lead to higher performance if a larger number of instructions is needed to implement the desired task. For this reason, the SPEC ratings provide a much better indicator when comparing two processors. A four-stage pipeline may increase instruction throughput by a factor of four. In general, an n-stage pipeline has the potential to increase throughput n times. Thus, it would appear that the higher the value of n, the larger the performance gain. This leads to two questions:

- How much of this potential increase in instruction throughput can be realized in practice?
- What is a good value for n?

Any time a pipeline is stalled, the instruction throughput is reduced. Hence, the performance of a pipeline is highly influenced by factors such as branch and cache miss penalties. First, we discuss the effect of these factors on performance, and then we return to the question of how many pipeline stages should be used.

EFFECT OF INSTRUCTION HAZARDS

The effects of various hazards have been examined qualitatively in the previous sections. We now assess the impact of cache misses and branch penalties in quantitative terms. Consider a processor that uses the four-stage pipeline of Figure 3.2. The clock rate, hence the time allocated to each step in the pipeline, is determined by the longest step. Let the delay through the ALU be the critical parameter. This is the time needed to add two integers. Thus, if the ALU delay is 2 ns, a clock of 500 MHz can be used. The on-chip instruction and data caches for this processor should also be designed to have an access time of 2 ns. Under ideal conditions, this pipelined processor will have an instruction throughput, Pp, given by

$$Pp = R = 500 \text{ MIPS (million instructions per second)}$$

NUMBER OF PIPELINE STAGES

The fact that an n-stage pipeline may increase instruction throughput by a factor of n suggests that we should use a large number of stages. However, as the number of pipeline stages increases, so does the probability of the pipeline being stalled, because more instructions are being executed concurrently. Thus, dependencies between instructions that are far apart may still cause the pipeline to stall. Also, branch penalties may become more significant. For these reasons, the gain from increasing the value of n begins to diminish, and the associated cost is not justified.

Another important factor is the inherent delay in the basic operations performed by the processor. The most important among these is the ALU delay. In many processors, the cycle time of the processor clock is chosen such that one ALU operation can be completed in one cycle. Other operations are divided into steps that take about the same time as an add operation. It is also possible to use a pipelined ALU. For example, the ALU of the Compaq Alpha 21064 processor consists of a two-stage pipeline, in which each stage completes its operation in 5 ns.

Many pipelined processors use four to six stages. Others divide instruction execution into smaller steps and use more pipeline stages and a faster clock. For example, the Ultra SPARC II uses a 9-stage pipeline and Intel’s Pentium Pro uses a 12-stage pipeline. The latest Intel processor, Pentium 4, has a 20-stage pipeline and uses a clock speed in the range 1.3 to 1.5 GHz. For fast operations, there are two pipeline stages in one clock cycle.

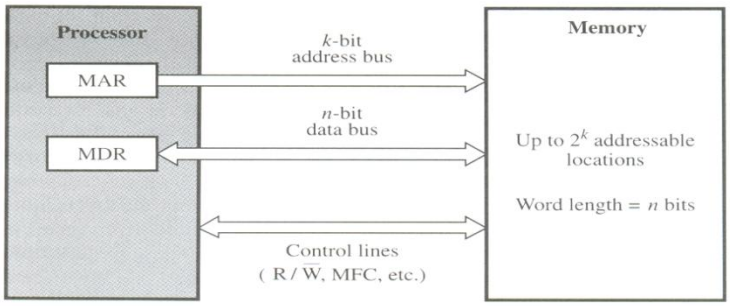
UNIT IV MEMORY SYSTEM

1. 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 initiation 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 program 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.

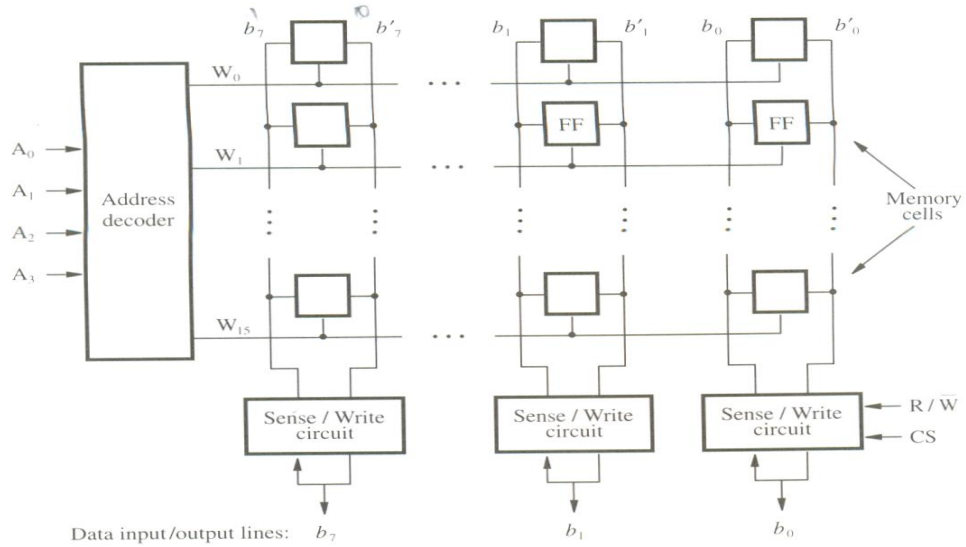
2. SEMI CONDUCTOR RAM MEMORIES:

- 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 information.
- 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 circuit are connected to a single bidirectional data line that can be connected to a data bus of the cpu.

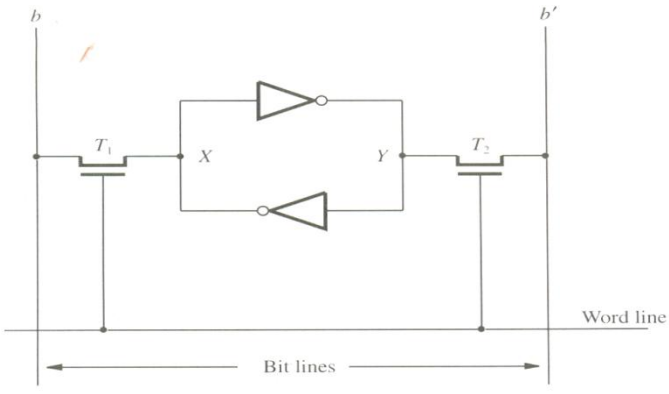
R / W →Specifies the required operation.
CS →**Chip Select** input selects a given chip in the multi-chip memory system

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.

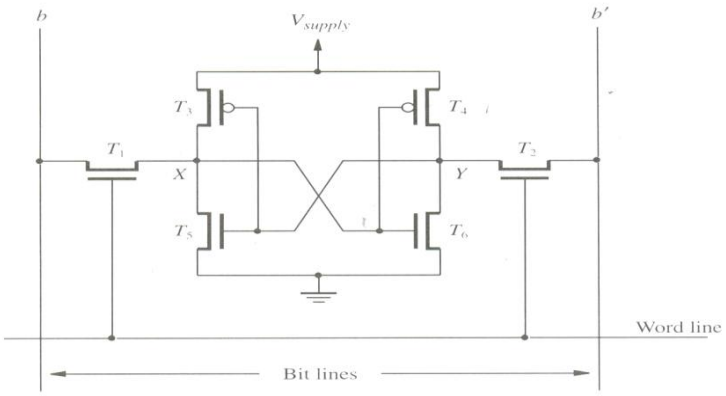
Read Operation:

- 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.

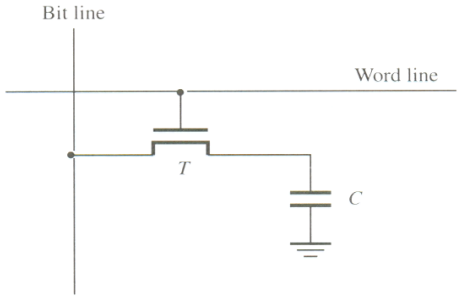
Demerit:

- 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.

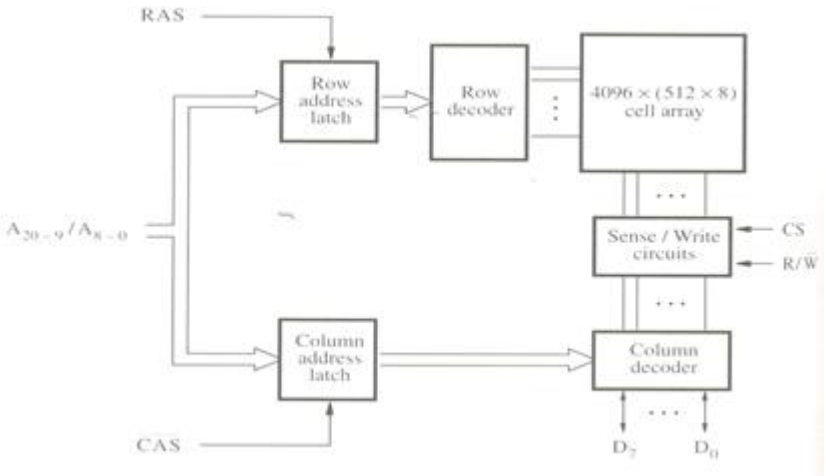
Fig:A single transistor dynamic Memory cell



- 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.
- 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.
- 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.

Fig:Synchronous DRAM

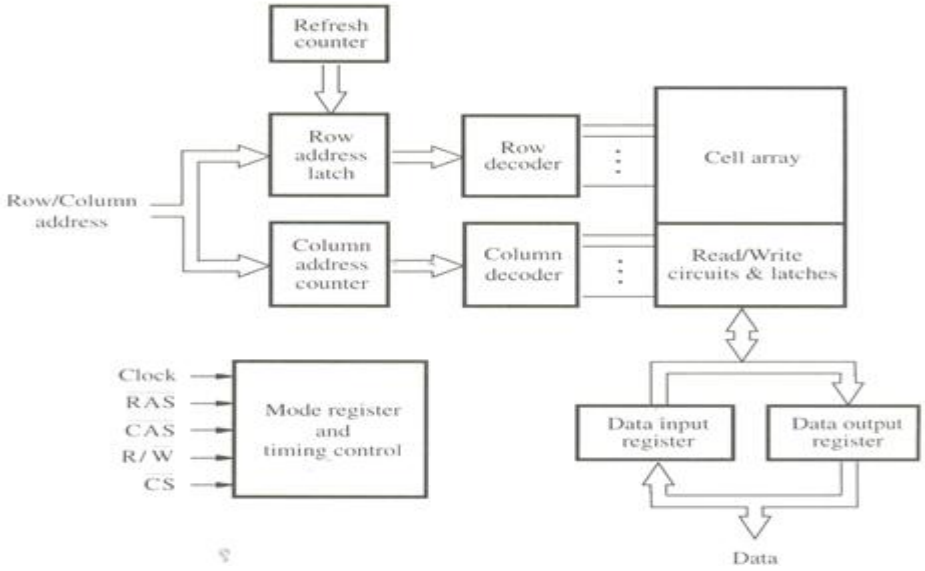
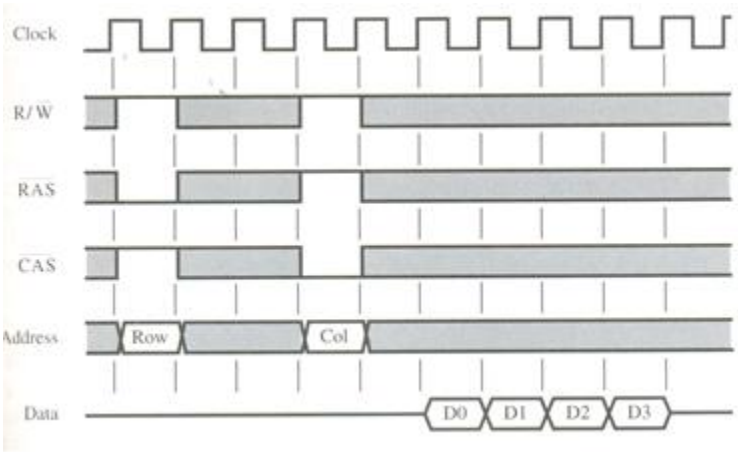


Fig:Timing Diagram →Burst Read of Length 4 in an SDRAM



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

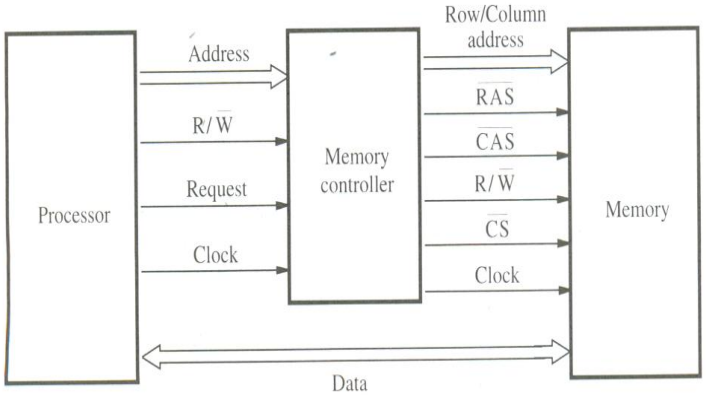
- SIMM & DIMM consists of several memory chips on a separate small board that plugs vertically into single socket on the motherboard.

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

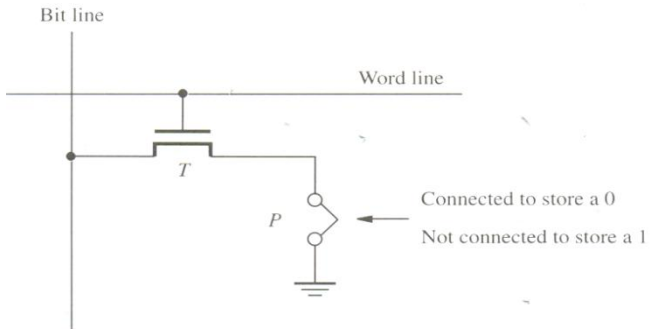
Rambus Memory:

- 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,9^{th} 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

3. READ ONLY MEMORY:

- 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.)

4. 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

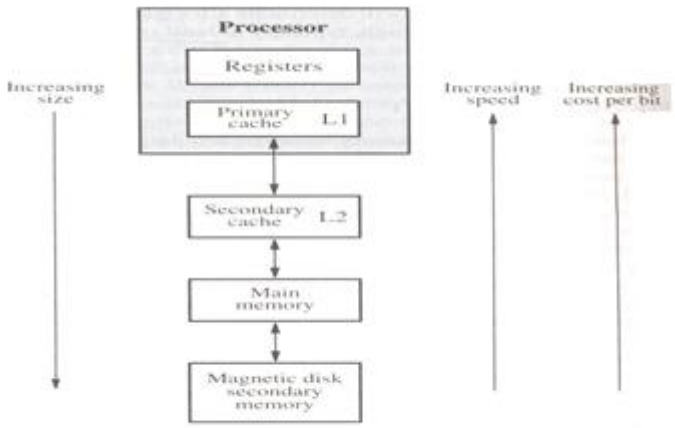
Magnetic Disk:

- 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.

5.CACHE MEMORIES:

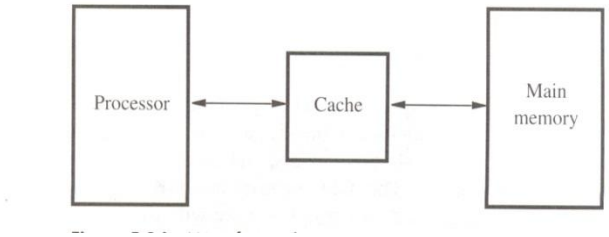
- The effectiveness of cache mechanism is based on the property of '**Locality of reference'.**

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:

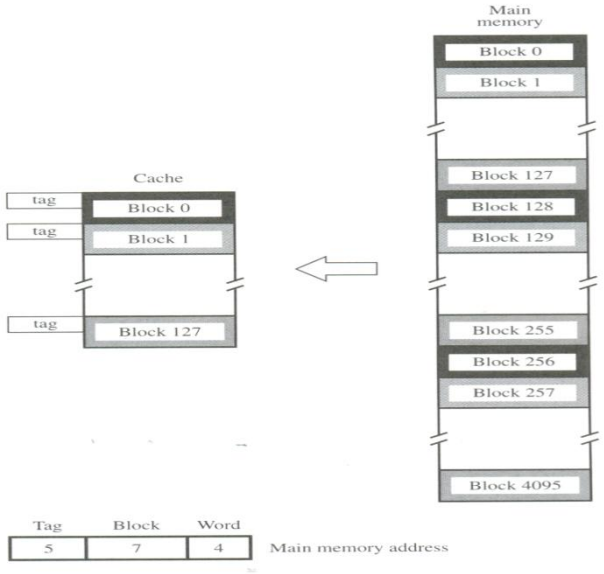
- After the entire block is loaded into cache, the particular word requested is forwarded to the processor.
- If the requested word not exists in the cache during write operation, then **Write Miss** will occur.
- If Write through protocol is used, the information is written directly into main memory.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

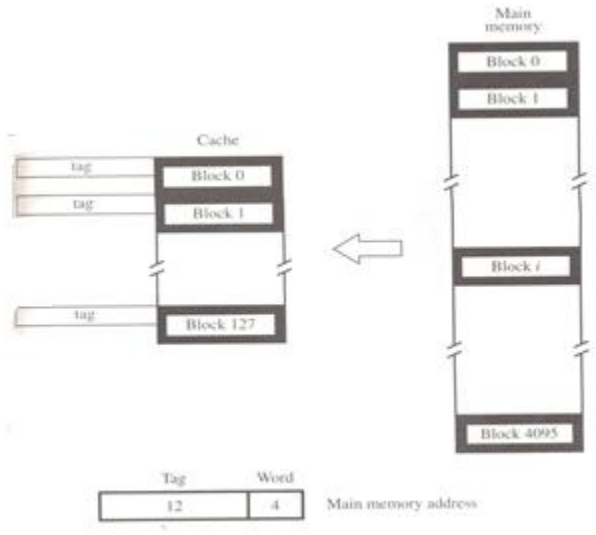
Demerit:

- 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:

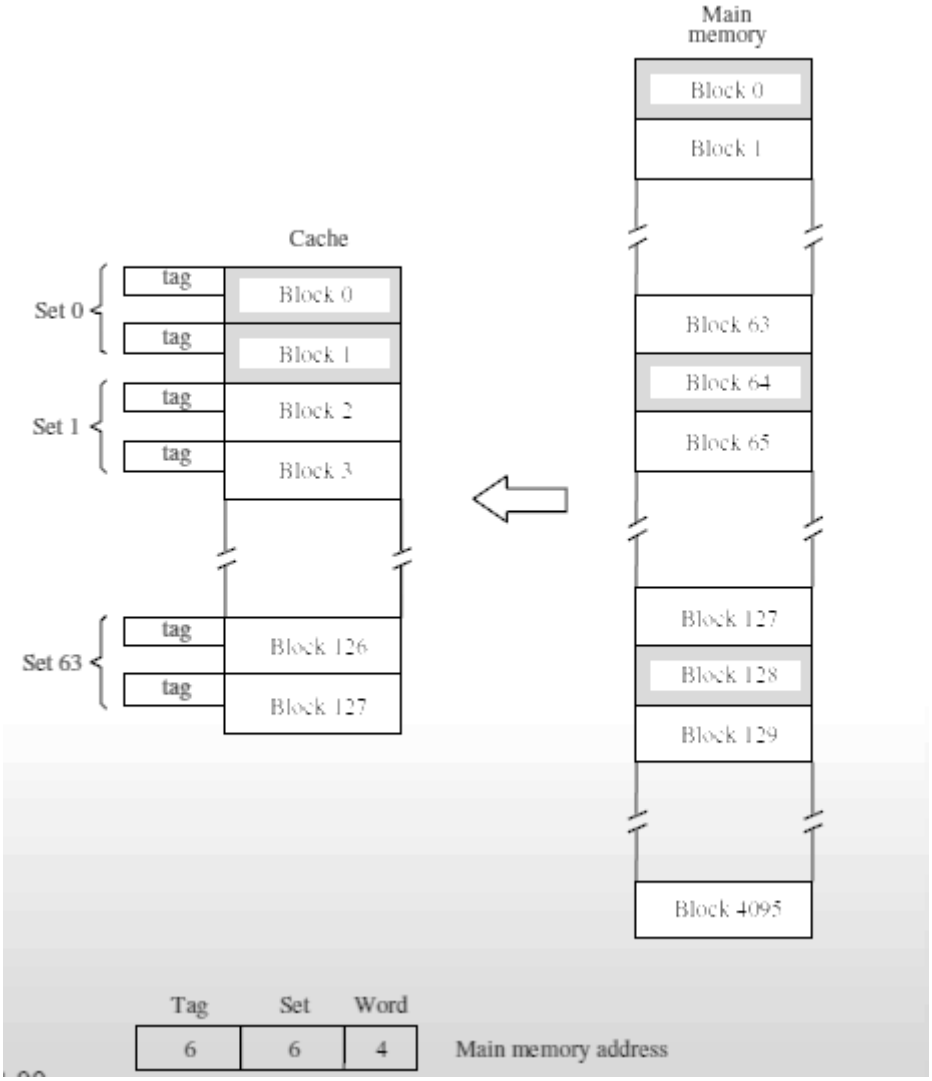
- It is the combination of direct and associative mapping.
- The blocks of the cache are grouped into sets and the mapping allows a block of the main memory to reside in any block of the specified set.
- In this case,the cache has two blocks per set,so the memory blocks 0,64,128……..4032 maps into cache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

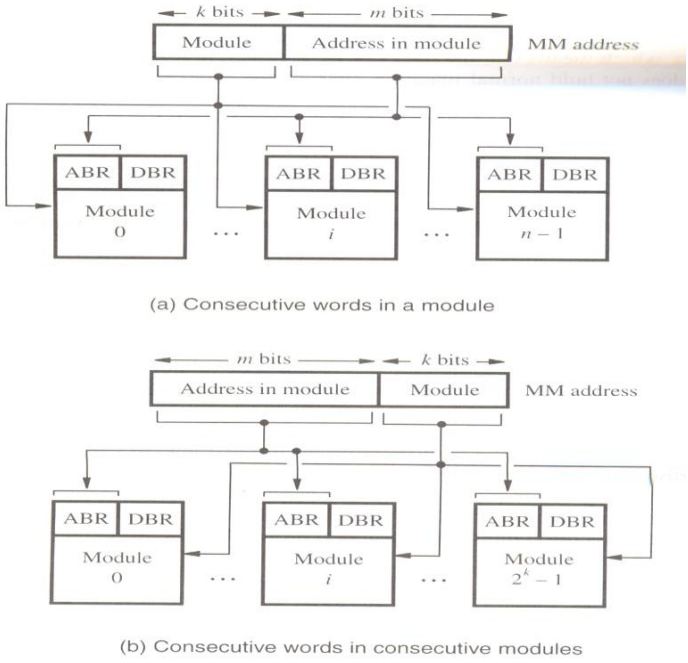
- The performance of LRU algorithm is improved by randomness in deciding which block is to be over-written.

6.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



7.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.

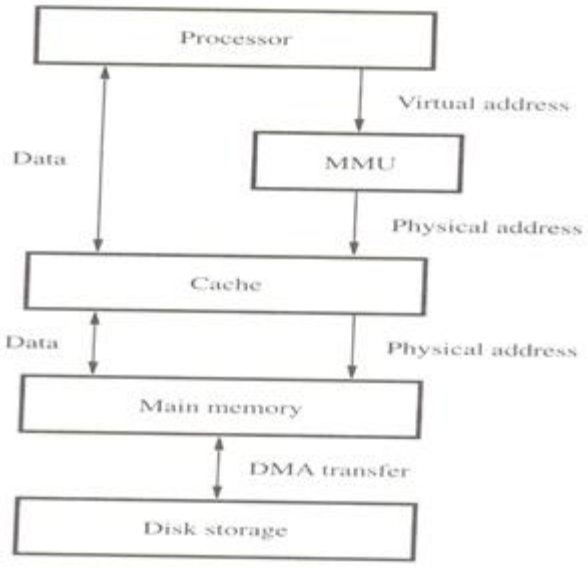
Address Translation:

- 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.

Fig:Virtual Memory Organisation



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.

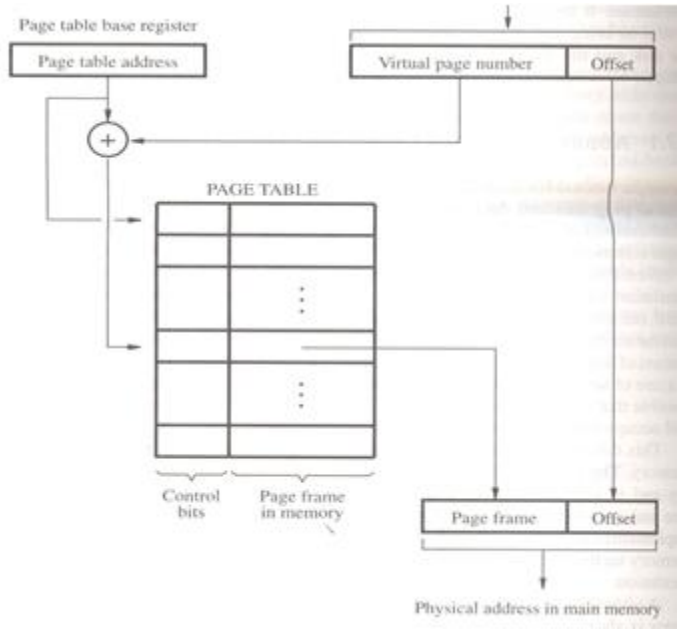
Control Bits in Page Table:

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

Function:

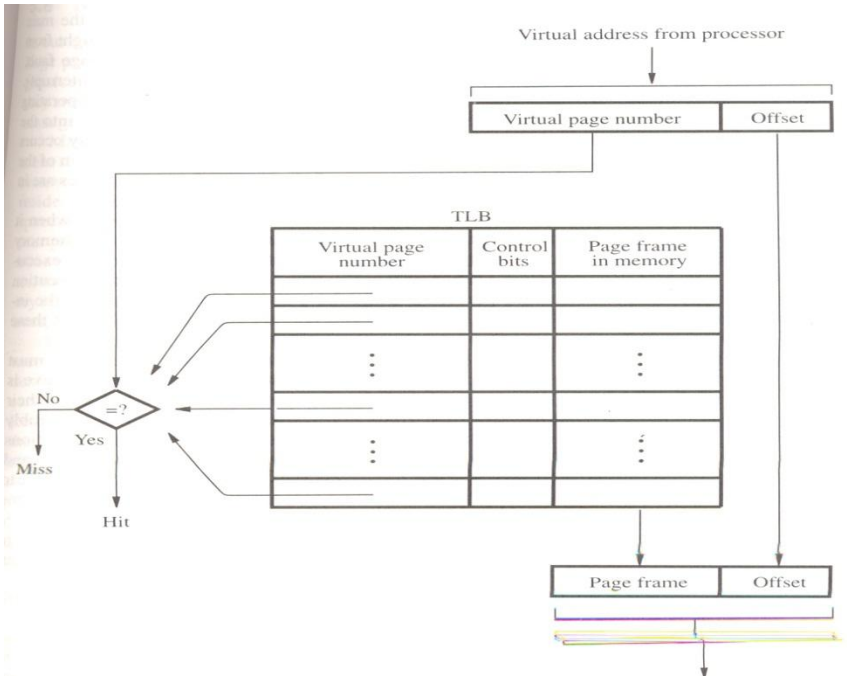
- The control bit indicates the validity of the page ie)it checks whether the page is actually loaded in the main memory.
- It also indicates that whether the page has been modified during its residency in the memory;this information is needed to determine whether the page should be written back to the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

Fig:Use of Associative Mapped TLB



- 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.

8.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.

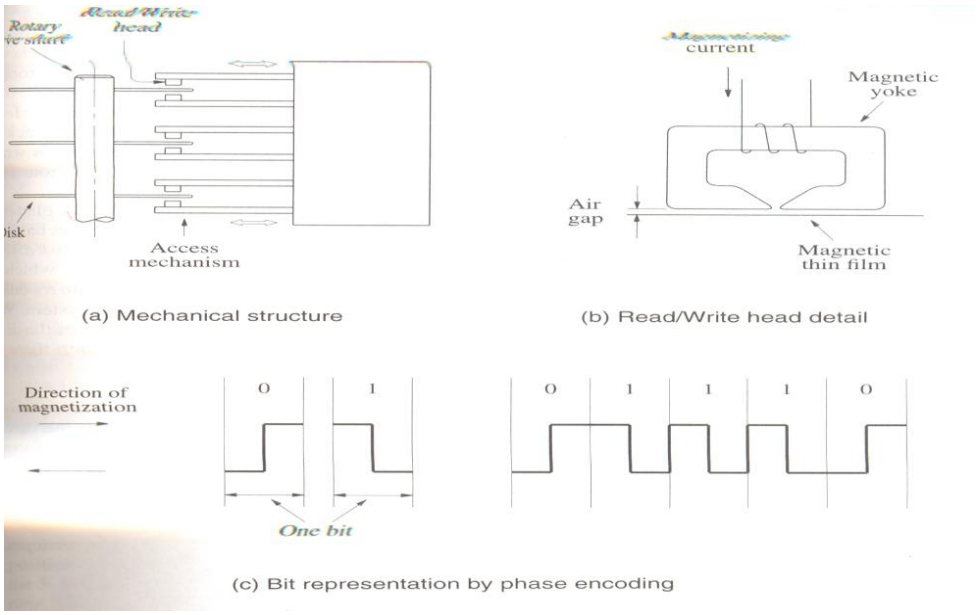
9.SECONDARY STORAGE:

- 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.

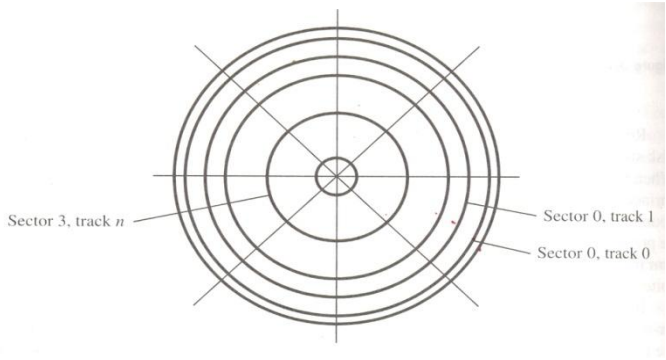
Wanchester Technology:

- 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

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).

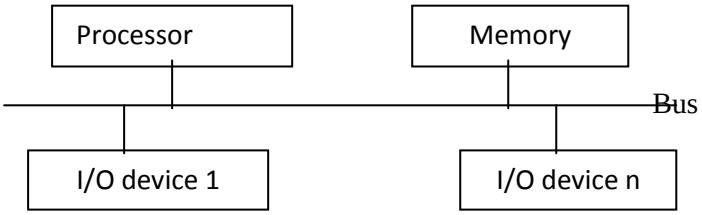
If the data is available in the cache, it can be accessed and placed on SCSI bus . If it is not available then the data will be retrieved from the disk.

UNIT V - I/O ORGANIZATION

1.1 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.**

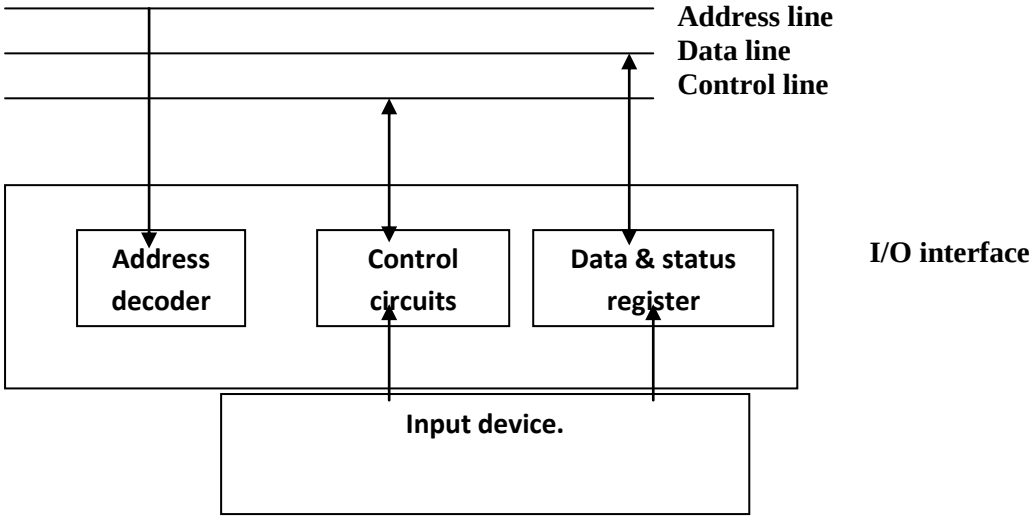
Single Bus Structure



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



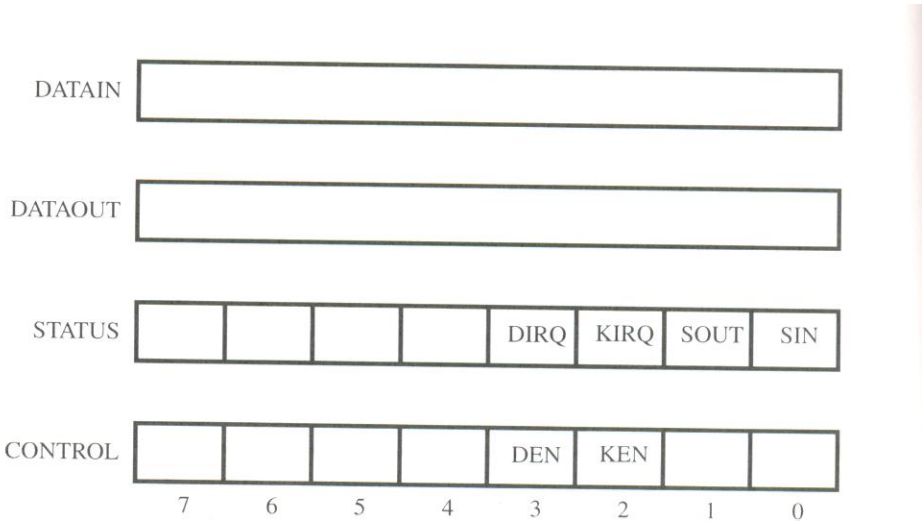
1.2.Address Decoder:

- 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.

1.3.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
```

1.4.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.

1.5.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)

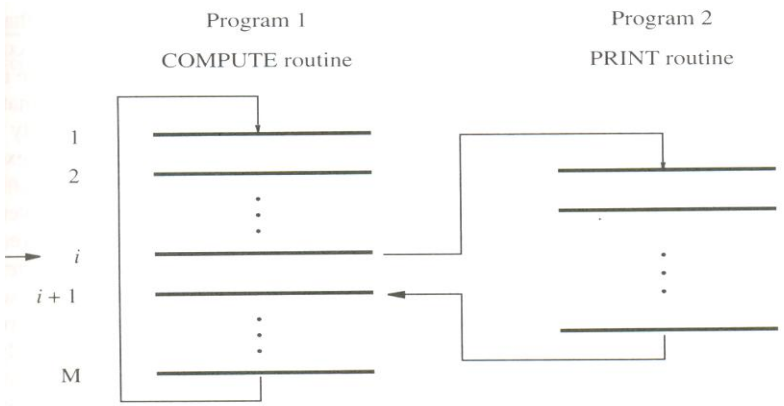
1.6.DMA:

- 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.

2.1.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.

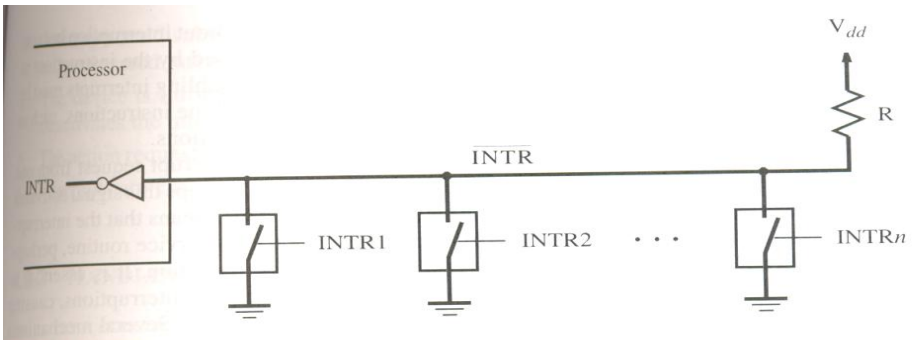
Fig:Transfer of control through the use of interrupts



- 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 $\overline{\text{drive}}$ 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.

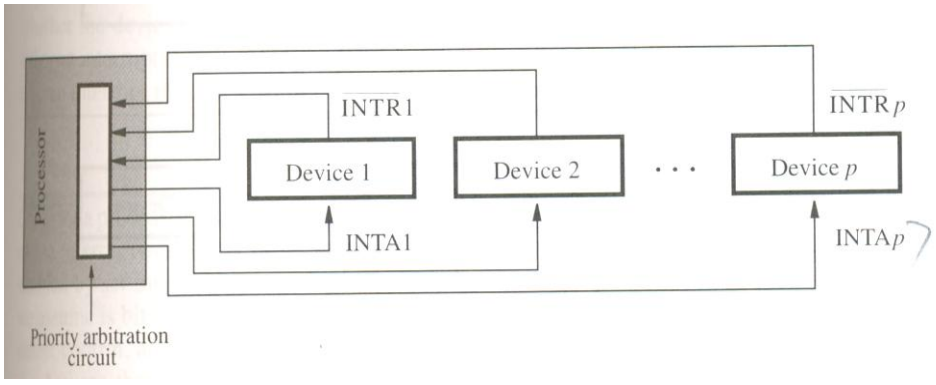
Privileged Instruction:

- 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:

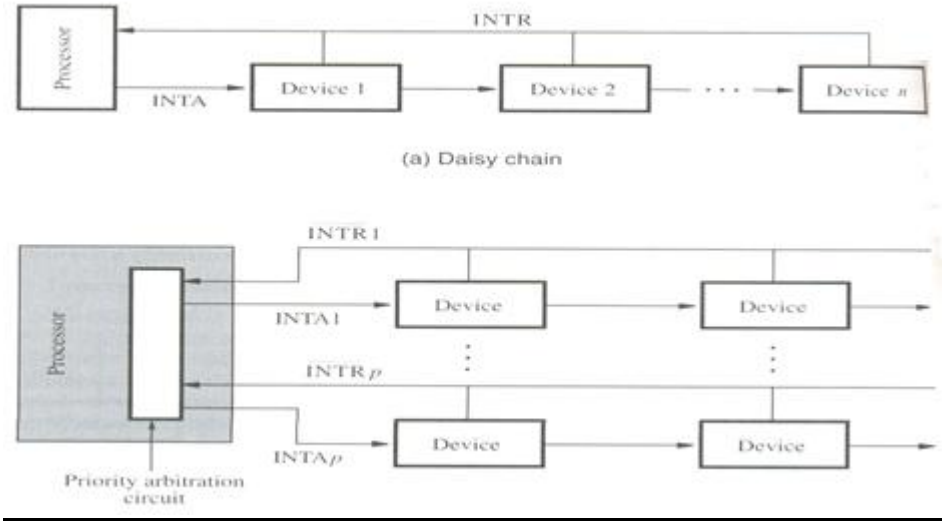
- User program cannot accidently or intentionally change the priority of the processor & disrupts the system operation.
- An attempt to execute a privileged instruction while in user mode, leads to a special type of interrupt called the privileged exception.

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



- 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:



- 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:

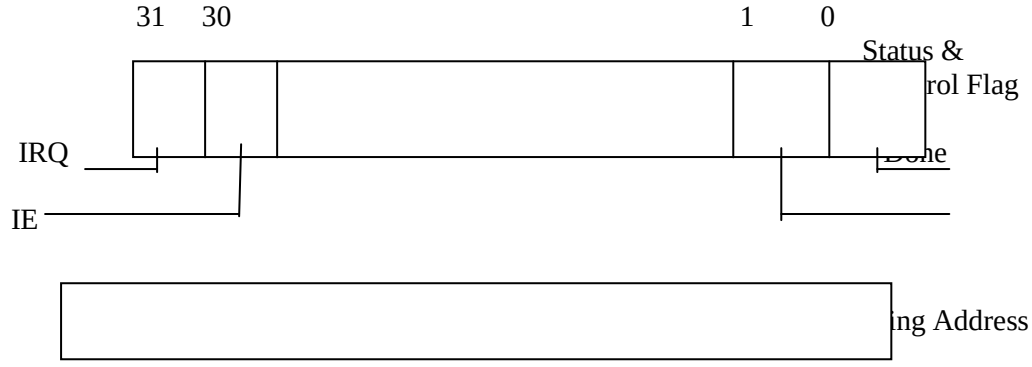
- To protect the OS of a computer from being corrupted by user program certain instance can be executed only when the processor is in supervisor mode. These are called privileged exceptions.
- When the processor is in user mode, it will not execute instance (ie) when the processor is in supervisor mode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

3.1.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

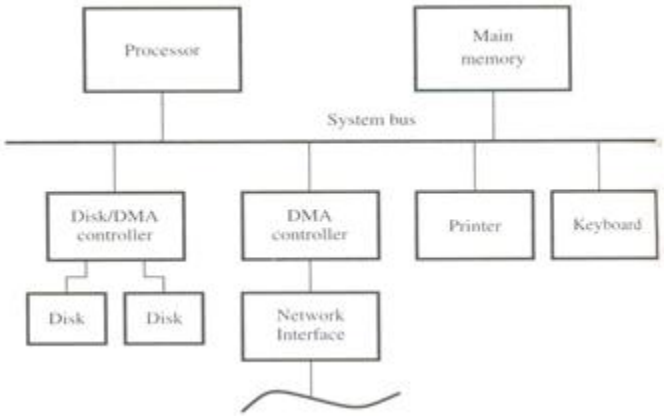


	Word Count

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

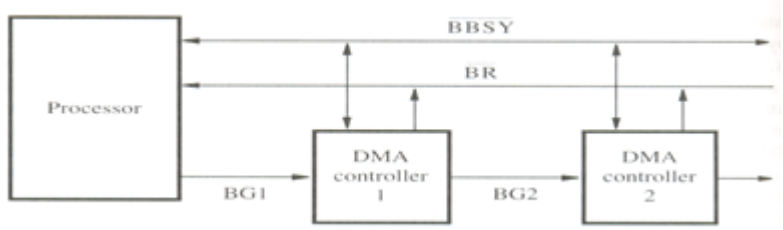
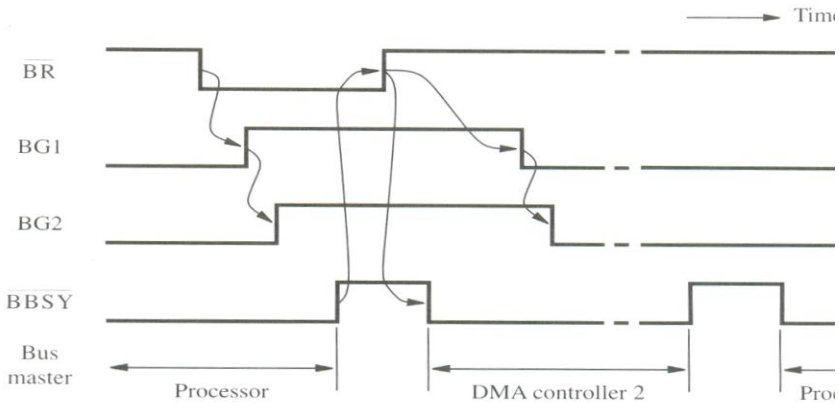


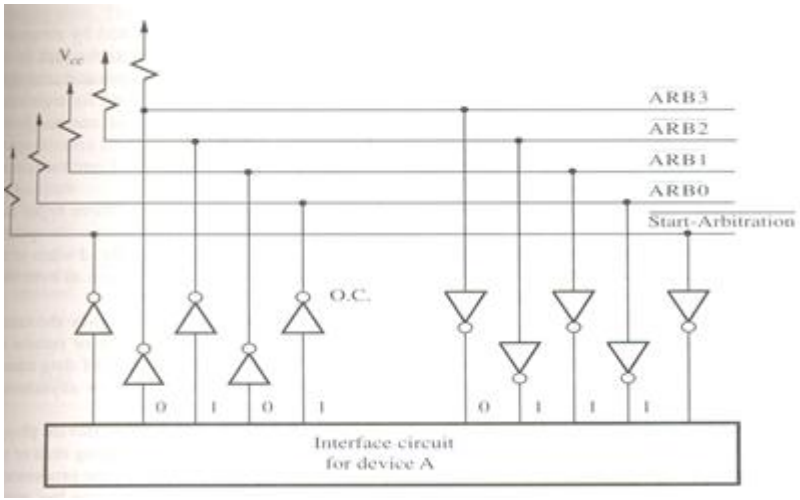
Fig: Sequence of signals during transfer of bus mastership for the devices



- 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.

4.1.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:

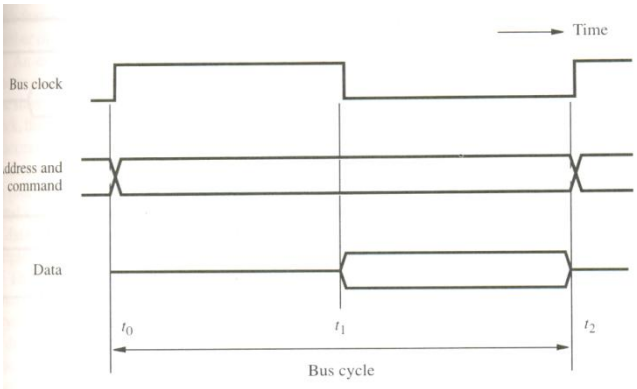
There are 2 types of buses. They are,

- Synchronous Bus
- Asynchronous Bus.

Synchronous Bus:-

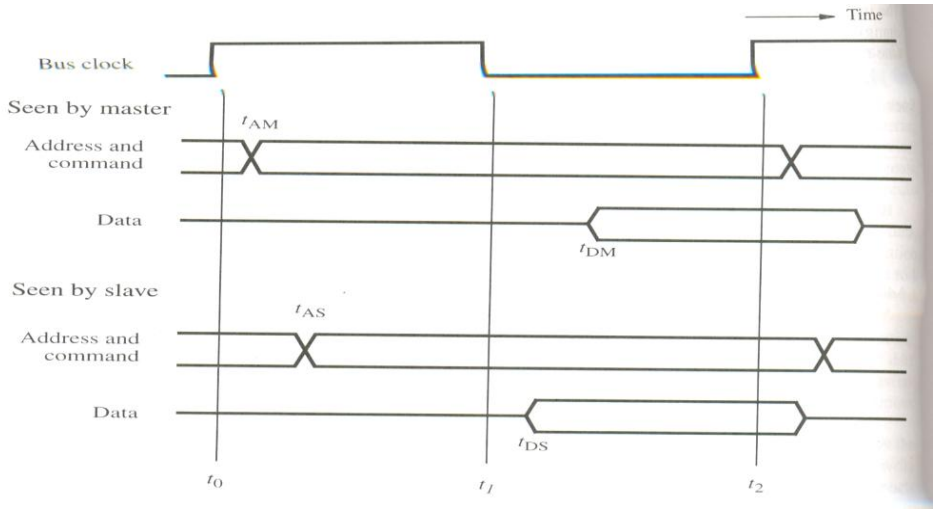
- In synchronous bus, all devices derive timing information from a common clock line.
- Equally spaced pulses on this line define equal time.
- During a '**bus cycle**', one data transfer on take place.
- The '**crossing points**' indicate the tone at which the patterns change.
- A '**signal line'** 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 to, 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.

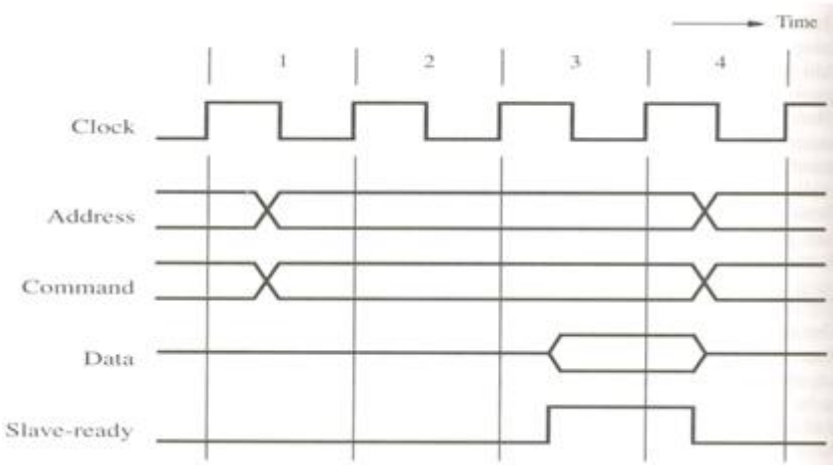
Demerits:

- 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.

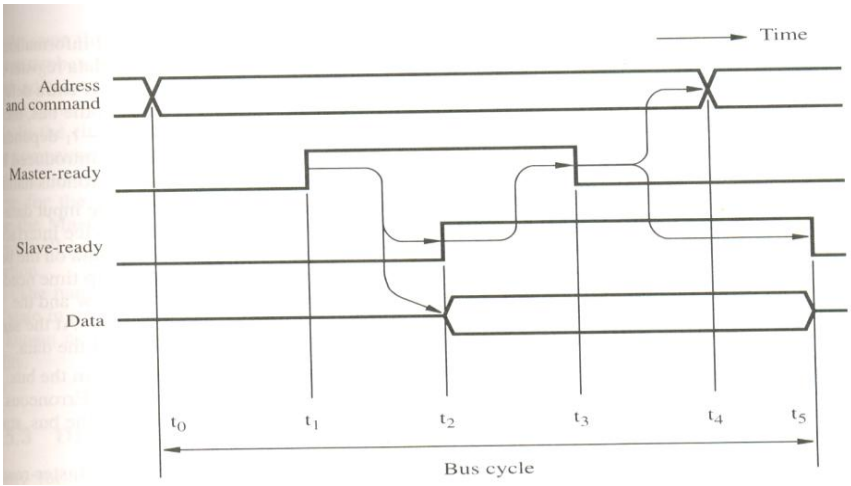
Fig:An input transfer using multiple clock cycles



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.

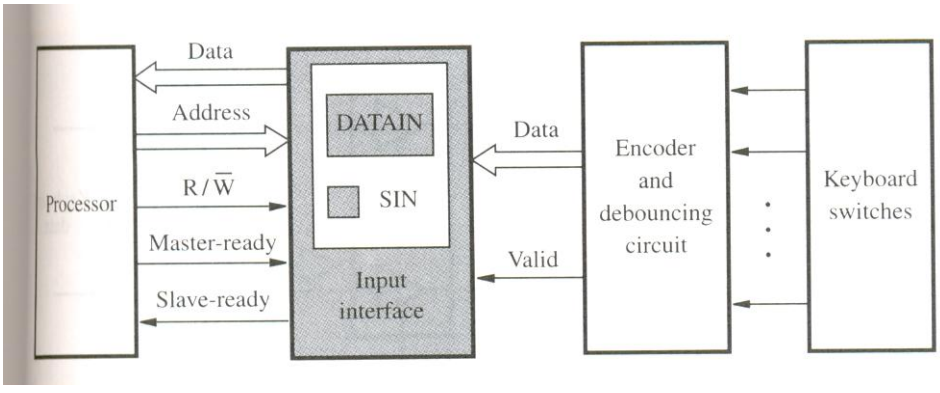
- In this diagram, the master place the output data on the data lines and at the same time it transmits the address and command information.
- The selected slave strobes the data to its o/p buffer when it receives the Master-ready signal and it indicates this by setting the slave – ready signal to 1.
- 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

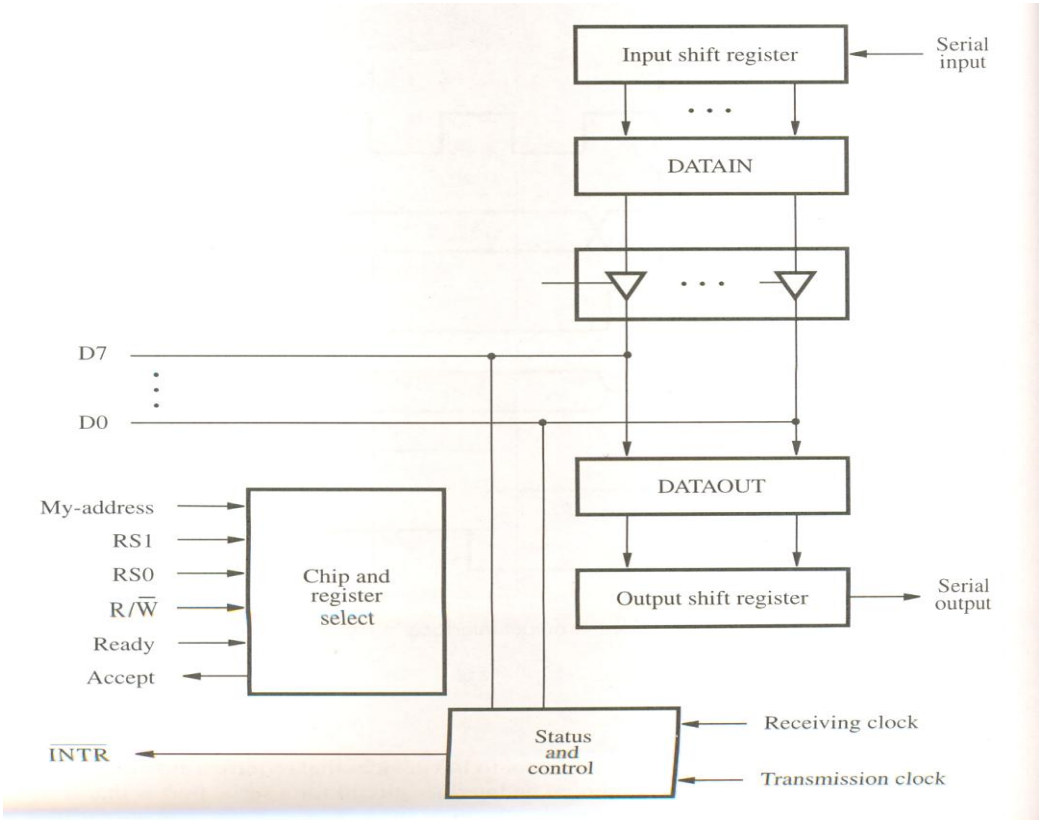
- Parallel Port
- Serial Port

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:

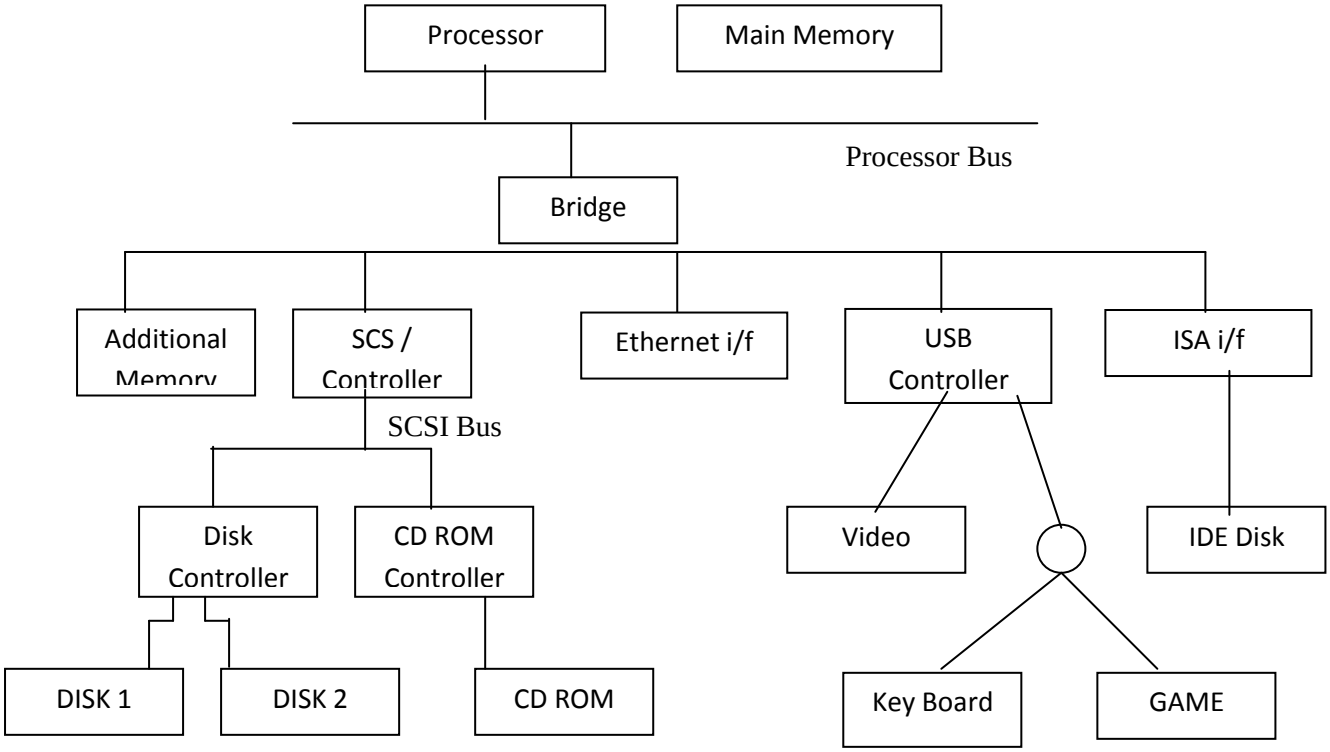


A serial port used to connect the processor to I/O device that requires transmission one bit at a time.
It is capable of communicating in a bit serial fashion on the device side and in a bit parallel fashion on the bus side.

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



There are 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:

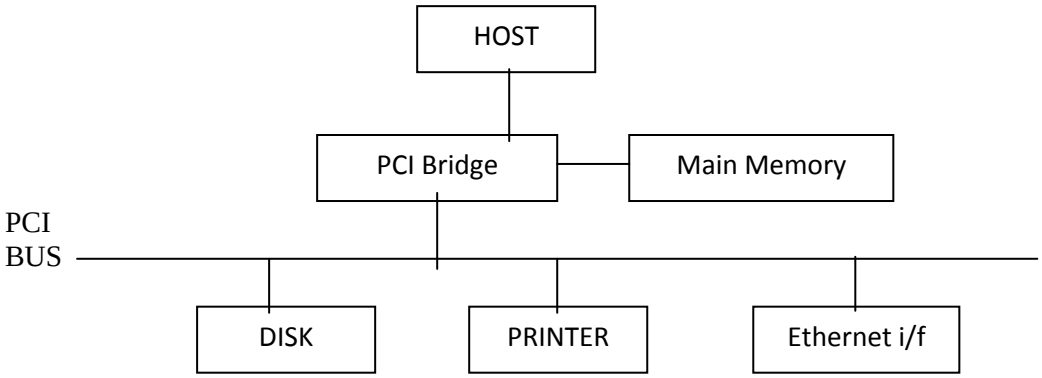
- The data are transferred between cache and main memory is the bursts of several words and they are stored in successive memory locations.

- 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

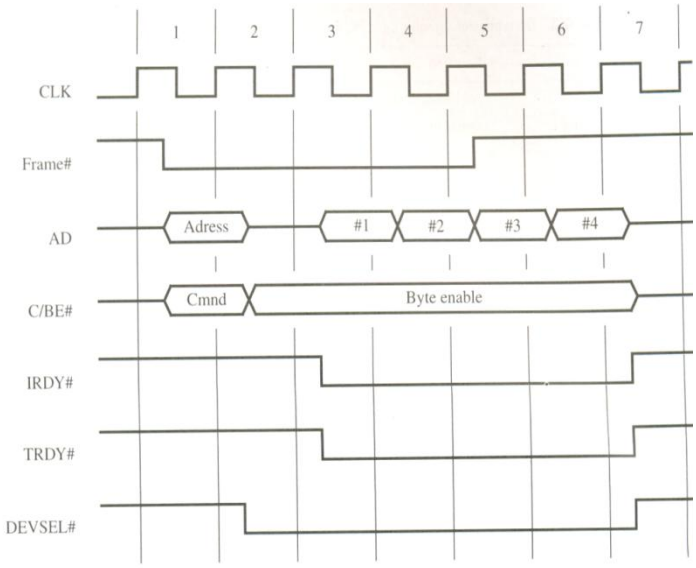


Data Transfer Signals on PCI Bus:

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

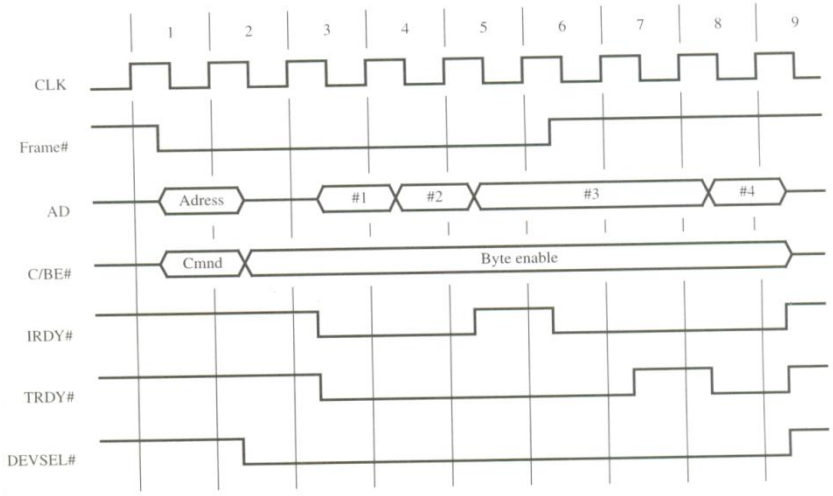
Individual word transfers are called '**phases'**.

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 4^{th} 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.

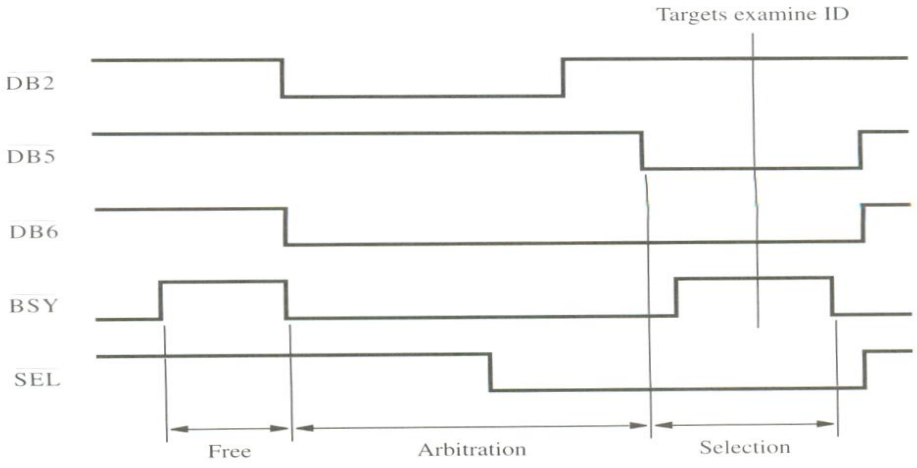
Phases in SCSI Bus:-

- 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.

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.

QUESTION BANK

SUBJECT CODE: CS2253 YEAR : II SEM : IV

SUBJECT NAME: COMPUTER ORGANIZATION AND ARCHITECTURE

UNIT I BASIC STRUCTURE OF COMPUTERS

PART-A (2 MARKS)

1. Define the term Computer Architecture.
2. Define Multiprocessing.
3. What is meant by instruction?
4. What is Bus? Draw the single bus structure.
5. Define Pipeline processing.
6. Draw the basic functional units of a computer.
7. Briefly explain Primary storage and secondary storage.
8. What is register?
9. Define RAM.
10. Give short notes on system software.
11. Write down the operation of control unit?
12. Define Memory address register.
13. Define Addressing modes.
14. Write the basic performance equation?
15. Define clock rate.
16. List out the various addressing techniques.
17. Draw the flow of Instruction cycle.
18. Suggest about Program counter.
19. List out the types in displacement addressing.
20. What is meant by stack addressing?
21. Define carry propagation delay.
22. Draw a diagram to implement manual multiplication algorithm.
23. Perform the 2's complement subtraction of smaller number(101011) from larger number(111001).

PART-B (16 Marks)

1. Write briefly about computer fundamental system?
2. Explain memory unit functions.
3. Explain memory locations and addresses.
4. Explain Software interface.
5. Explain instruction set Architecture? Give examples.
6. What is bus explain it in detail?
7. 9. Discuss about different types of addressing modes.
8. 10. Explain in detail about different instruction types and instruction sequencing.
9. 11. Explain Fixed point representation.
10. 12. How floating point addition is implemented. Explain briefly with a neat diagram.
11. 13. Give the difference between RISC and CISC.
12. 14. Write an algorithm for the division of floating point number and illustrate with an example.

UNIT II BASIC PROCESSING UNIT

PART-A (2 MARKS)

1. What are the basic operations performed by the processor?
2. Define Data path.
3. Define Processor clock.
4. Define Latency and throughput.
5. Discuss the principle operation of micro programmed control unit.
6. What are the differences between hardwired and micro programmed control units?
7. Define nanoprogramming.
8. What is control store?
9. What are the advantages of multiple bus organization over a single bus organization?
10. Write control sequencing for the executing the instruction. Add R4,R5,R6.
11. What is nano control memory?
12. What is the capacity of nano control memory?
13. Define micro routine.
14. What is meant by hardwired control?
15. What are the types of micro instruction?
16. Name the methods for generating the control signals.

PART-B(16 MARKS)

1.With a neat sketch, explain how execution of complete instruction is carried out.
1. Draw and explain typical hardware control unit.
2. Draw and explain about micro program control unit.
3. Write short notes on
(i)Micro instruction format (ii) Symbolic micro instruction.
4. Explain multiple bus organization in detail.

UNIT III PIPELINING

PART-A (2 MARKS)

1. What is Pipelining?
2. What are the major characteristics of a Pipeline?
3. What are the various stages in a Pipeline execution?
4. What are the types of pipeline hazards?
5. Define structural, data, and control hazard.
6. List two conditions when processor can stall.
7. List the types of data hazards.
8. List the techniques used for overcoming hazard.
9. What is instruction level parallelism?
10. What are the types of dependencies?
11. What is delayed branching?
12. Define deadlock.
13. Draw the hardware organization of two stage pipeline.
14. What is branch prediction?
15. Give two examples for instruction hazard.
16. List the various pipelined processors.
17. Why we need an instruction buffer in a pipelined CPU?
18. What are the problems faced in instruction pipeline?

19. Write down the expression for speedup factor in a pipelined architecture.

PART-B (16 MARKS)

1. Explain different types of hazards that occur in a pipeline.
2. Explain various approaches used to deal with conditional branching.
3. Explain the basic concepts of pipelining and compare it with sequence processing with a neat diagram.
4. Explain instruction pipelining.
5. What is branch hazard? Describe the method for dealing with the branch hazard?
6. What is data hazard? Explain the methods for dealing with data hazard?
7. Explain the function of six segment pipeline and draw a space diagram for six segment pipeline solving the time it takes to process eight tables.
8. Explain the influence of instruction sets.
9. Draw and explain data path modified for pipelined execution.
10. Explain about various exceptions.

UNIT IV MEMORY SYSTEM

PART- A (2 MARKS)

1. What is Memory system?
2. Give classification of memory.
3. Define cache.
4. What is Read Access Time?
5. Define Random Access Memory.
6. What are PROMS?
7. Define Memory refreshing.
8. What is SRAM and DRAM?
9. What is volatile memory?
10. Define data transfer or band width.
11. What is flash memory?
12. What is multi level memories?
13. What is address translation page fault routine, page fault and demand paging?
14. What is associate memory?
15. Define Seek time and latency time.
16. What is TLB?
17. Define Magneto Optical disk.
18. Define Virtual memory.
19. What are the enhancements used in the memory management?
20. Define the term LRU and LFU.
21. Define memory cycle time.
22. What is static memories?
23. What is locality of reference?
24. Define set associative cache.
25. What is meant by block replacement?
26. List the advantages of write through cache.
27. Give formula to calculate average memory access time.
28. Define conflict.
29. What is memory interleaving?
30. What is DVD?
31. Give the features of ROM cell.
32. List the difference between static RAM and dynamic RAM.

33. What is disk controller?
34. How a data is organized in the disk?

PART-B (16 MARKS)

1. Illustrate the characteristics of some common memory technologies.
2. Describe in detail about associative memory.
3. Discuss the concept of Memory interleaving and give its advantages.
4. Discuss the different mapping techniques used in cache memories and their relative merits and demerits.
5. Comparing paging and segmentation mechanisms for implementing the virtual memory.
6. What do you mean by virtual memory? Discuss how paging helps in implementing virtual memory.
7. Discuss any six ways of improving the cache performance.
8. Explain the virtual memory translation and TLB with necessary diagram.
9. Explain the organization of magnetic disk and magnetic tape in detail.

UNIT V I/O ORGANIZATION

PART-A (2 MARKS)

1. Define intra segment and inter segment communication.
2. Mention the group of lines in the system bus.
3. What is bus master and slave master?
4. Differentiate synchronous and asynchronous bus.
5. What is strobe signal?
6. What is bus arbitration?
7. Mention types of bus arbitration.
8. What is I/O control method?
9. What is DMA?
10. Why does the DMA priority over CPU when both request memory transfer?
11. List out the types of interrupts.
12. What is dumb terminal?
13. What is the need for DMA transfer?
14. List down the functions performed by an Input/Output unit.

PART-B(16 MARKS)

1. Explain with the block diagram the DMA transfer in a computer system.
2. Describe in detail about IOP Organization.
3. Describe the data transfer method using DMA.
4. Write short notes on the following

(a) Magnetic disk drive

(b) Optical drives.

5. Discuss the design of a typical input or output interface.
6. What are interrupts? How are they handled?
7. Give comparison between memory mapped I/O and I/O mapped I/O.
8. Explain the action carried out by the processor after occurrence of an interrupt.
9. What is DMA? Describe how DMA is used to transfer data from peripherals.
10. Explain various data transfer modes used in DMA.
11. Explain SCSI bus standards.
12. Describe the working principle of USB.