15CS32T- COMPUTER ORGANISATION

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

UNIT- 1: Introduction to Data Communication

1.1 Functional Units

The basic functional units of a computer is as shown in the following Figure 1.1.

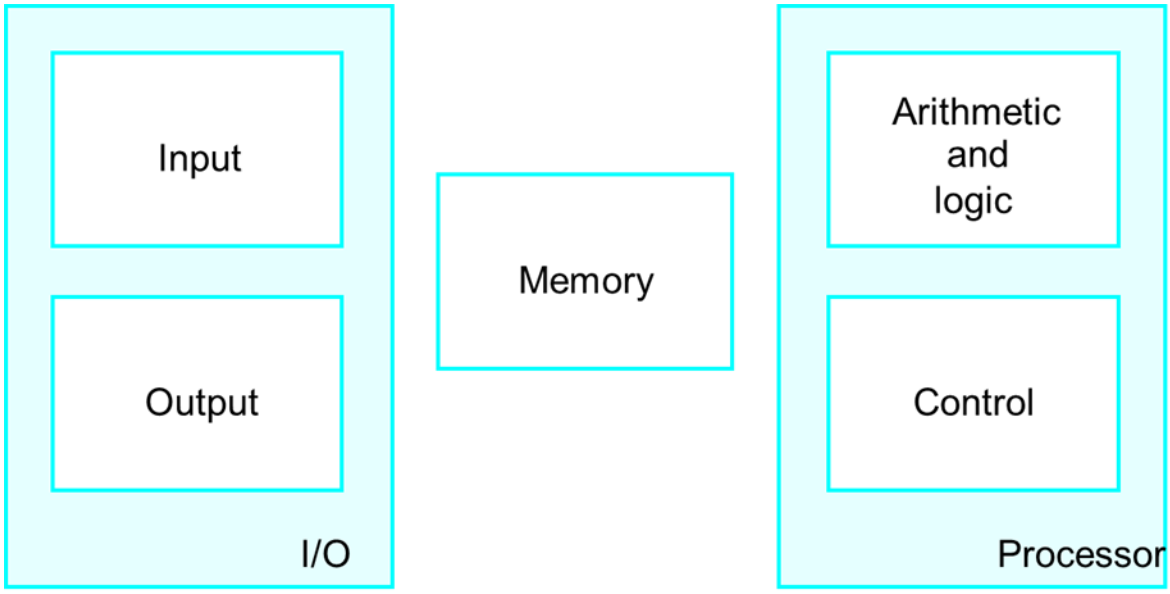


Figure 1.1. Basic functional units of a computer.

A computer system consists of five functionally independent main part:-

1. Input unit
2. Output unit
3. Memory unit
4. Arithmetic and Logic unit (ALU).
5. Control unit.

There are 2 types of information handled by a computer

1. Instructions/machine instructions:-

- These monitor the transfer of information within a computer and also between the computer and its I/O devices
- They also specify the arithmetic and logic operations to be performed

- A set of instructions that performs a particular task is called a Program. A computer is controlled by a stored program except for the external interruption by an operator or peripheral devices connected to the system.

2. Data:
    - These are the numbers and encoded characters that can be used as operands by the instructions.
    - An entire program is treated as a data if it is completely used by another program, like compiler (Compiler takes a high level language program as an input and produces a machine language program called Object program).

All the information handled by a computer must be in a suitable encoded format.

Numbers are represented in positional binary notation like BCD (Binary Coded Decimal)

Alphanumeric characters are represented using several coding schemes like ASCII(American Standard Code for Information Interchange) which uses seven bit code and EBCDIC (Extended Binary-Coded Decimal Interchange Code) which uses 8 bit code to denote a character.

1.1.1 Input Unit:-

Computer accepts encoded information through input unit. The standard input device is a keyboard. Whenever a key is pressed, the corresponding letter or digit is converted into its corresponding binary code and transmitted to either memory/processor.

Examples include Mouse, Joystick, Tracker ball, Light pen, Digitizer, Scanner etc.

1.1.2 Memory Unit:-

Memory unit stores the program instructions (Code), data and results of computations etc. Memory unit is classified as:

• Primary /Main Memory

• Secondary /Auxiliary Memory

Primary memory is a semiconductor memory that provides access at high speed. Programs must be stored in the primary memory while they are executed.

The memory contains a number of semiconductor storage cells. Each cell is capable of storing only one bit of information. These are rarely accessed as individual cells, but they are processed in groups of fixed size called **Words**. The number of bits in each word is termed as **word length** of a computer, which can vary from 16 to 64 bits. The time taken to access one word in a memory is called **memory access time**. The memory is organized in such a way that the contents of one word can be read or written in one basic operation. Every word in the memory is associated with a distinct address which will be used along with a control command to perform read and write operations.

Main memory is classified again as ROM and RAM. ROM (Read Only Memory) holds system programs and firmware routines such as BIOS, POST, I/O Drivers that are essential to manage the hardware of a computer. RAM (Random Access Memory) is termed as Read/Write memory or user memory that holds run time program instruction and data. The smallest and fast RAM units are called **cache.** Any location in RAM can be reached in short and fixed amount of time. While primary storage is essential, it is volatile in nature and expensive. Additional requirement of memory could be supplied as auxiliary memory at cheaper cost. **Secondary memories** are non volatile in nature.

1.1.3 Arithmetic and Logic Unit:-

ALU consist of necessary logic circuits like adder, comparator etc., to perform operations of addition, multiplication, comparison of two numbers etc. Any operation is initiated by bringing the needed operands into the processor and stored in high speed storage elements called registers, and then the operation is performed by ALU.

1.1.4 Output Unit:

Computer after computation returns the computed results, error messages, etc. via output unit. The standard output device is a video monitor, LCD/TFT monitor. Other output devices are printers, plotters etc.

1.1.5 Control Unit:

Control unit co-ordinates activities of all units by issuing timing signals. These signals govern the data transfers and then appropriate operations take place. Control unit interprets or decides the operation/action to be performed. A number of control lines (wires) carry the signals for timing and synchronization of the events in all the units.

The operations of a computer are as follows:

1. A set of instructions called a program reside in the main memory of computer.
2. The CPU fetches those instructions sequentially one-by-one from the main memory, decodes them and performs the specified operation on associated data operands in ALU.
3. Processed data and results will be displayed on an output unit.
4. All activities pertaining to processing and data movement inside the computer machine are governed by control unit.

1.2 Basic Operational Concepts:

An Instruction consists of two parts, an Operation code and operand/s as shown below:

OPCODE	OPERAND/s

Let us see a typical instruction: ADD LOCA, R0

This instruction is an addition operation. The following are the steps to execute the instruction:
Step 1: Fetch the instruction from main memory into the processor
Step 2: Fetch the operand at location LOCA from main memory into the processor
Step 3: Add the memory operand (i.e. fetched contents of LOCA) to the contents of register R0
Step 4: Store the result (sum) in R0.

The same instruction can be realized using two instructions as

Load LOCA, R1
Add R1, R0

The steps to execute the above instructions can be enumerated as below:

Step 1: Fetch the instruction from main memory into the processor
Step 2: Fetch the operand at location LOCA from main memory into the processor Register R1
Step 3: Add the content of Register R1 and the contents of register R0
Step 4: Store the result (sum) in R0.

Figure 1.2 below shows how the memory and the processor are connected. As shown in the diagram, in addition to the ALU and the control circuitry, the processor contains a number of registers used for several different purposes. The **instruction register** (IR) holds the instruction that is currently being executed. The **program counter** (PC) keeps track of the execution of the program. It contains the memory address of the next instruction to be fetched and executed. There are n general purpose registers R0 to Rn-1 which can be used by the programmers during writing programs.

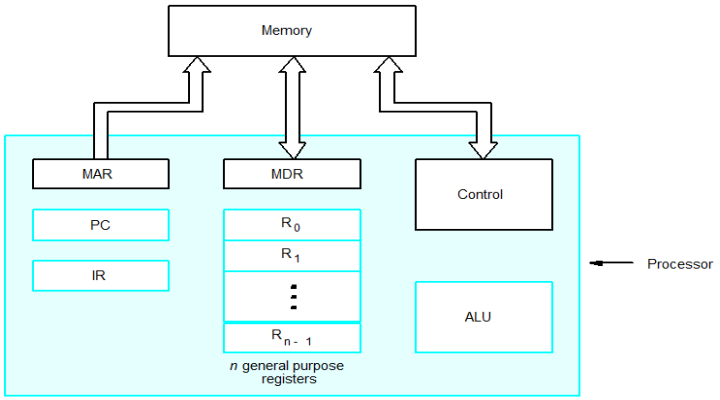


Figure 1.2. Connections between the processor and the memory

Two registers MAR (Memory Address Register) and MDR (Memory Data Register) facilitate communication with the memory. The address of the location to be accessed is stored in MAR. The data to be written into or read from the location is stored in MDR.

The execution of the program is started when the address of the first instruction of the program is stored in PC. The contents of PC is transferred to MAR and a read control signal is issued to the memory. After the memory access time is elapsed, the contents of the memory location is loaded into MDR. Next, the contents of MDR is transferred into IR. Then the instruction is decoded and executed.

The required operands for the instruction is fetched from the memory by sending its address to MAR and initiating a Read cycle. When the operand is brought into MDR, it is transferred to the ALU. Once all the required operands are fetched, the ALU performs the desired operation. If the result is to be stored into the memory, it is transferred to MDR and the corresponding address of the memory location is transferred to MAR and then a Write cycle is initiated. During the execution of the current instruction, in between, the PC is incremented to point to the next instruction to be executed. Hence, the execution of the next instruction will be started as soon as the current instruction is executed.

Some devices may require urgent services from the processor which will affect the normal execution of the programs. In order to handle such situations, the execution of the current program must be interrupted. This happens when the device raises an interrupt signal. An interrupt is nothing but a request for services from an I/O device to the processor. The processor executes an **interrupt service routine** to provide the required service to the I/O device.

1.3 Bus Structures:

There are many ways to connect different parts inside a computer together.

Figure 1.3. A single-bus structure.

Individual parts must communicate over a communication line or path for exchanging data, address and control information as shown in Figure 1.3. A group of lines that serves as a connecting path for several devices is called a *bus*. The simplest way to interconnect the functional units is through a single bus structure. All the units are connected to this bus. Only two units can actively communicate at any given time. Bus control lines are used to handle multiple requests for the use of the bus.

Advantages of single bus structure:

1. Low cost.
2. Flexible to attach peripheral devices.

Systems containing multiple buses can achieve more concurrency in operations by allowing two or more communications to be carried out

at the same time. This leads to better performance. The cost of multiple bus is high when compared to single bus.

Electro-mechanical devices such as keyboards, printers are relatively slow when compared to memory and processors which operate at electronic speeds. All these devices must communicate with each other over a bus. In order to smooth out the differences in timing among these devices, buffer registers are used along with the devices to hold the information during the transfers.

For example- Consider the transfer of information from a processor to a printer. The processor sends the character over the bus to the printer buffer. Once the buffer is loaded, the printer starts printing without any intervention by the processor. The bus and the processor are released for other activity until the printer prints the content in the buffer. The printer is also not available for further transfers until it finishes printing. Hence, the timing differences between the processor and the printer is handled using the buffer registers. Buffer registers also contributes in preventing a high speed processor from being locked by a slow device. This allows the processor to switch from one device to another during processing.

UNIT- 2: Machine Instructions & Programmes

2.1 MEMORY LOCATIONS AND ADDRESSES

Computer can handle nonnumeric text information consisting of characters. Characters can be alphabets, decimal digits, punctuation marks etc . The most widely used code for representing nonnumeric text is the *American Standards Committee on Information Interchange (ASCII) code.*

Numbers, character operands and instructions are stored in the memory of a Computer.
The memory consists of millions of storage *cells*, each of which can store a bit of information either the value 0 or 1.In memory a group of *n* bits is referred to a memory word and *n* is called the *word length*. The memory of a computer can be represented as shown in the fig 2.1

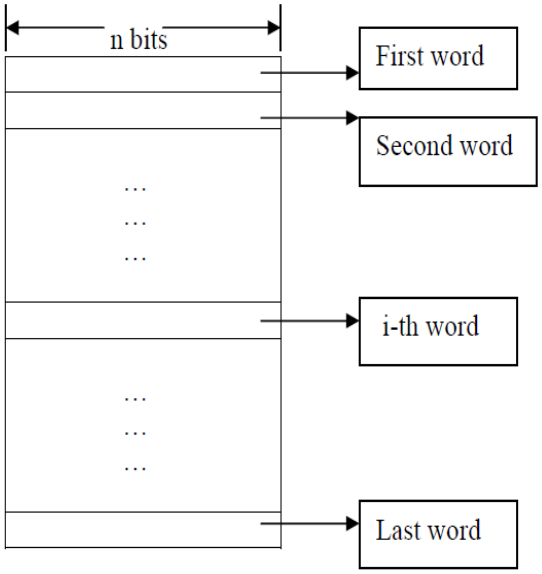


Figure 2.1 Memory words

Today's Modern computers have word lengths that typically range from 16 to 64 bits. A unit of 8 bits is called a *byte*. A group of 4 bytes(for 32 bit computers) or group of 8 bytes(for 64 bit computer) is referred as a single word. A memory word of 32 bit computer will hold 4 ASCII characters is as shown in Figure 2.2

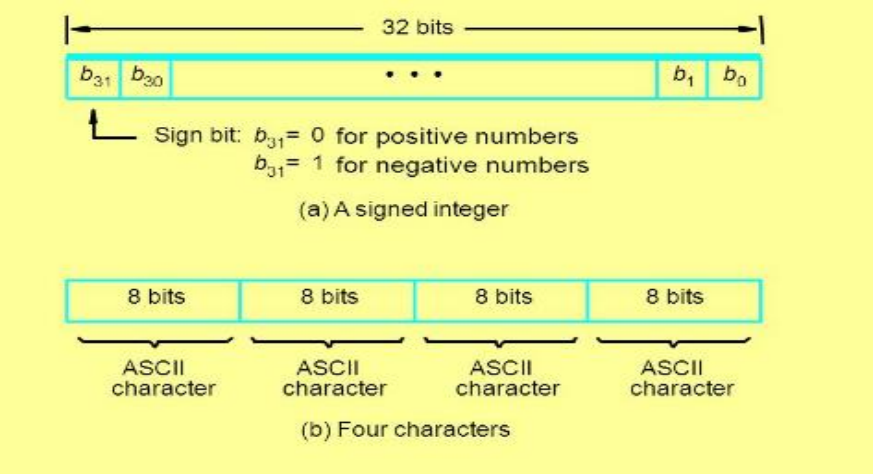
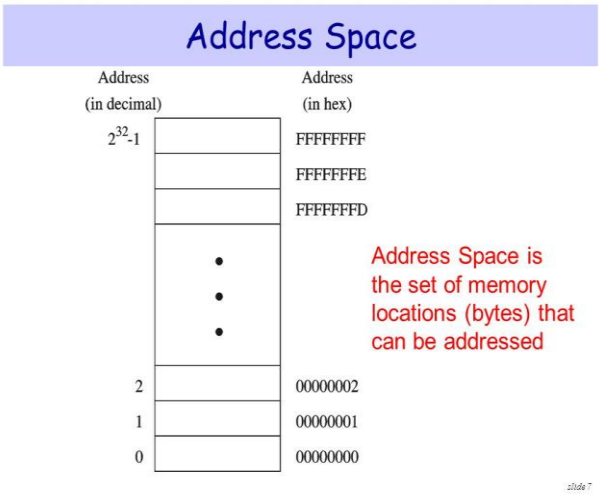


Figure 2.2 An example of encode information of 32 bit word



To retrieve/store information from/to memory, either for one word or one byte (8-bit), addresses for each location are needed. A *k*-bit address memory has 2^k memory locations, namely 0 to 2^k-1, called memory space.

Eg. 32-bit memory: 2^{32} = 0 to 2^{32}-1 Memory locations = 4G

64-bit memory: 2^{64} = 0 to 2^{64}-1 Memory locations

2.1.1 **Byte addressability**

A byte is always 8 bits, but the word length typically ranges from 16 to 64 bits. The best method to assign addresses to individual locations of memory is to have successive addresses for successive byte locations in the memory. The term *byte-addressable memory* is used for this assignment in modern computers.. Byte locations have addresses 0, 1, 2, ... Thus, if the word length of the machine is 32 bits, successive words are located at addresses 0, 4, 8, . . . , with each word of 4 bytes.

2.1.2 **BIG-ENDIAN AND LITTLE-ENDIAN ASSIGNMENTS**

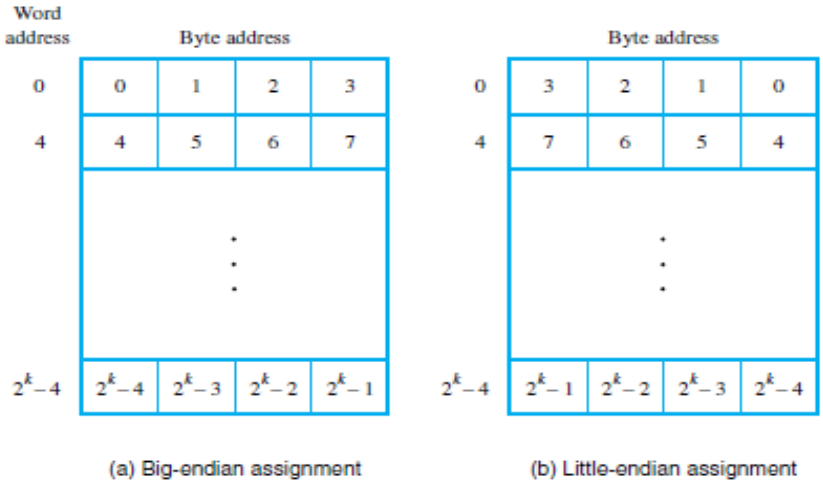


Figure : 2.3 Byte and word addressing

There are two ways that byte addresses can be assigned .They are Big-endian and Little-endian. In *big-endian* lower byte addresses are used for the most significant bytes (the leftmost bytes) of the word. In *little-endian* the lower byte addresses are used for the least significant bytes (the rightmost bytes) of the word.

In fig 2.3 (a) address 0 and 4 are assigned by taking most significant byte addresses 0 and 4. In fig 2.3 (b) address 0 and 4 are assigned by taking least significant byte addresses 0 and 4.

2.1.3 WORD ALIGNMENT

Words are said to be aligned in memory if they begin at a byte address, that is a multiple of the number of bytes in a word.

16-bit word (2 bytes) : word addresses: 0, 2, 4,….
32-bit word (4 bytes) : word addresses: 0, 4, 8,….
64-bit word (8 bytes) : word addresses: 0, 8,16,….

2.1.4 ACCESSING NUMBERS, CHARACTERS AND CHARACTER STRINGS

A number occupies one word and can be accessed in the memory by its word address. Similarly, individual characters can be accessed by their byte addresses.

In many applications, The beginning of the string is indicated by the address of the byte containing its first character. The next Successive byte locations contain successive characters of the string. The length of the string is indicated by 2 ways.

a) A special control character " end of string" is used at the last character of the string.
b) A memory word location or processor register can contain a number showing the length of the string in bytes.

2.2 MEMORY OPERATIONS

The data operands and the program instructions are stored in the memory. The execution of any instruction results in movement of operands and results between the memory and the processor.

The two basic memory operations are *Load* (or *Read* or *Fetch*) and *Store* (or *Write*).

1. **Load (or Read or Fetch) :** The Load operation sends a copy of the contents of a memory location to processor.

Step 1: The processor sends the address of the desired location to the memory.
Step 2:The memory reads the data stored at that address and sends them to the processor.
Here the contents of location are not changed.

2. **Store (or Write)** : The store operation sends an item from processor to a location.
Step 1: the processor sends both address of the location and the data to be written to that location.
Here the previous contents of location are destroyed.

2.3 INSTRUCTIONS AND INSTRUCTION SEQUENCING

The computer program processing, consists of a sequence of small steps, such as adding two numbers, testing for a particular condition, reading a character from the keyboard, or sending a character to be displayed on a display screen. A computer is provided with instructions that perform the following four types of operations:

- Data transfers between the memory and the processor registers
- Arithmetic and logic operations on data
- Program sequencing and control
- I/O transfers

2.3.1. REGISTER TRANSFER NOTATION

The possible locations in which transfer of information occurs are Memory Location, processor registers, or Registers in I/O sub-system. These locations are identified by symbolic names. These symbolic names are hardware binary addresses.

Ex: Symbolic names for memory locations – LOC, PLACE, VAR1,VAR2 etc.
Symbolic names for Processor registers – R0, R1.... etc.
Symbolic names for I/O registers -- DATAIN, DATAOUT etc.

The contents of a location are denoted by placing square brackets around the name of the location.

Ex: [LOC] , [R1],[DATAIN] etc

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

Note: Memory locations are identified by names A to Z and Processor registers are identified by R1 to R_N

2.3.2 ASSEMBLY LANGUAGE NOTATION

Another type of notation to represent machine instructions and programs is an *assembly language* format.

Ex1: An instruction to move from memory location LOC to processor register R1, is specified by the statement **Move LOC,R1**

The contents of LOC are transferred to R1. Here R1 contents are overwritten where as contents of LOC are unchanged.

Ex2: An instruction to add two numbers contained in processor registers R1 and R2 and placing their sum in R3 can be specified by the assembly language statement **Add R1,R2,R3**

2.3.3 BASIC INSTRUCTION TYPES

An instruction consists of two parts, an action (opcode) to perform and the operands upon which action is performed.

Ex : **Add A,B,C**

Where, Add is *opcode* and A,B,C are *operands*.

There are 4 types of basic instructions.

1. Three-address instructions.
2. Two-address instructions
3. One-address instructions
4. Zero-address instructions

Three-address instructions

An instruction which has 3 address fields (operands) is called three-address instructions.

General format : Operation source1,source2,destination

Ex: Add A,B,C

Here A and B are called source operands and C is the destination operand.

In the above example, the contents of memory locations A and B are added and stored in the memory location C.

Two-address instructions

An instruction which has 2 address fields (operands) is called two-address instructions.

General format : Operation source,destination

Ex: Add A,B

Here the contents of A and B are added and stored in B.

One-address instructions

An instruction which has 1 address fields (operands) is called two-address instructions.

Ex1: Add A

It means add the content of memory location A to the content of the accumulator register and stored the result in accumulator. Here accumulator is implicit in the instruction.
(Accumulator is a register used to perform arithmetic and logical operations)
Ex2: Load A
It copies the contents of memory location A into accumulator.
Ex3: Store A
It copies the contents of accumulator into memory location A.

Zero-address instructions
An instruction without address fields is called zero-address instructions.
Ex1. Push
Ex2. Pop
The above operations are used to push or pop the contents into or from the stack.

2.3.4 INSTRUCTION EXECUTION AND STRAIGHT-LINE SEQUENCING

Consider the example C ← [A] + [B] to add the contents of memory locations A and B and store the back to memory location C, is shown in the figure 2.4.

The above example can be written in assembly language as below

```
Move A , R0    ; Move the content from A to register R0
Add B, R0      ; Add the content of B directly with the content of the
                 register R0
Move R0, C     ; Move the result in R0 back to location C
```

The above three instructions of the program are in successive word locations, starting at location *i*. Since each instruction is 4 bytes long, the second and third instructions will start at addresses *i* + 4 and *i* + 8.

The above program is executed as below:
The processor contains a register called the *program counter* (PC), which contains the address of the next instruction to be executed . To begin executing any program, the address of its first instruction (*i* in our example) must be placed into the PC. Then, the processor will fetch and execute instructions one at a time, in the order of increasing addresses, as provided by the PC. This is called *straight-line sequencing*.
During the execution of each instruction, the PC is automatically incremented to point to the next instruction. In the above example PC is automatically incremented by 4 since it is 32 bit word.
The instructions are executed in two-phase. The first phase is called ***instruction fetch phase.*** In this phase, the instruction is fetched from the memory location specified by PC. Fetched instruction is placed in *instruction register* (IR) in the processor. The second phase is called ***instruction execution phase***. In this phase the instruction in the IR is decoded to perform the actual operation.

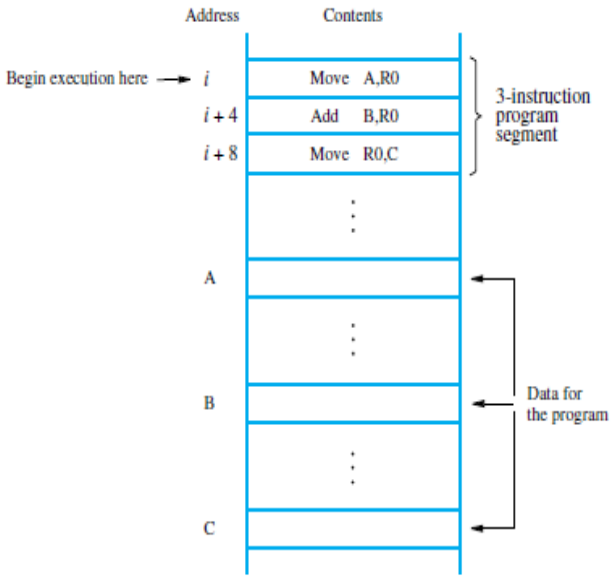


Figure 2.4 A program for C ← [A] + [B]

2.3.5 BRANCHING

Consider an example, shown in the figure 2.5 (a) to add N numbers and store the sum in one of the storage location. To add such N numbers, all the N numbers are placed in memory locations num1,num2,num3,num4.....then an Add instruction is used to add first two numbers. Then its sum is stored in one of the register .Then a second Add instruction is used to add third number with the previous sum. Thus it is continued until all n numbers are added .This method has to use many such add instructions and the program becomes lengthy.

This can be avoided by using a Looping statement (Loop is a straight line sequence of instructions executed many times as needed) and a single Add instruction can be used, as shown in figure 2.5(b)

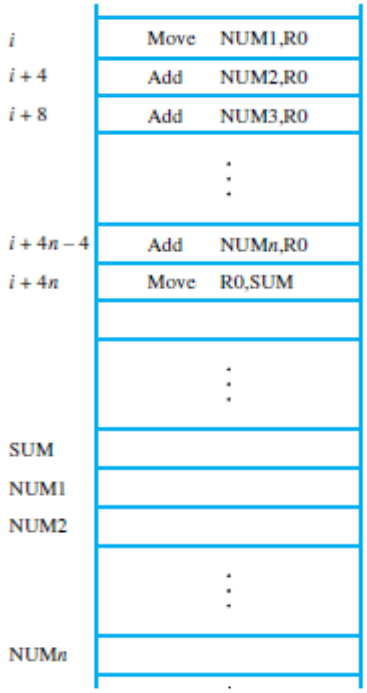


Figure 2.5(a) : Straight line program to add n numbers

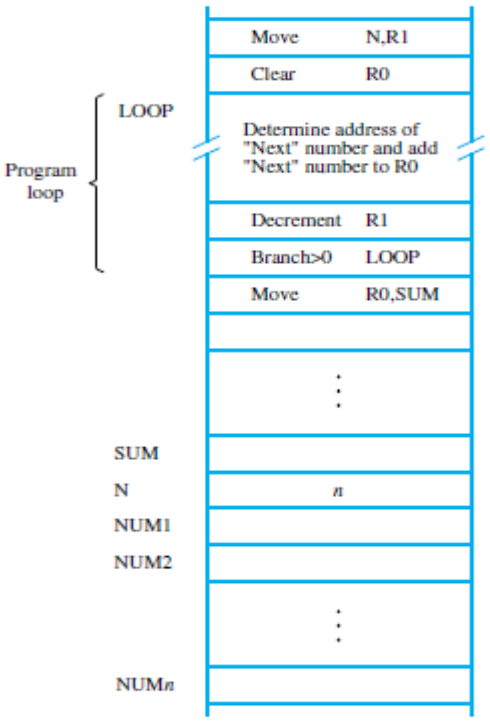


Figure 2.5(b) : Using loop to add N numbers

Let the number of entries in the list be N. which is stored in memory location N, as shown in the above figure.. Register R1 is used as a counter to determine the number of times the loop is repeated. The contents of location N are loaded into register R1 at the beginning of the program. Then, within the body of the loop, the instruction

Decrement R1

Decrements the contents of R1 by 1 each time the loop is executed.

Execution of the loop is repeated as long as the result of the decrement operation is greater than zero, as in the instruction **Branch > 0 LOOP.** This is a conditional branch statement which will be executed until the specified condition is met. (Unconditional assembly language branch statement includes **GOTO LABEL**)

When the branching condition becomes false, the loop is terminated. The result will be available in R0. The instruction **Move R0,SUM** moves the sum from R0 to memory location SUM.

2.3.6 CONDITION CODES

The processor keeps track of information about the results of various operations, Which is required by the subsequent conditional branch instructions. This is accomplished by recording the required information in individual bits, often called ***condition code flags***. These flags

are grouped together in a special processor register called ***condition code register*** **or** ***status register*****.** Individual condition code flags are set to 1 or reset to 0, depending on the result of the operation performed.

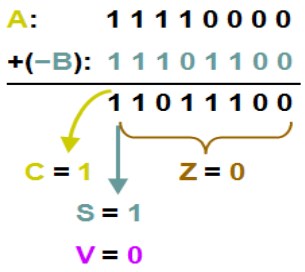
Four commonly used flags are

N (negative) Set to 1 if the result is negative; 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 a carry-out results from the operation; otherwise, cleared to 0

The N and Z flags indicate whether the result of an arithmetic or logic operation is Negative or zero. The N and Z flags are affected by instructions that transfer data, such as Move, Load, or Store.
The V flag indicates whether overflow has taken place. Overflow occurs when the result of an arithmetic operation exceeds the range of values. (That can be represented by the number of bits available for the operands)
The C flag is set to 1 if a carry occurs from the most significant bit position during an arithmetic operation.
How the flags are set or reset is shown in the below example.

- Example:
  - A: 1 1 1 1 0 0 0 0
  - B: 0 0 0 1 0 1 0 0



2.4 ADDRESSING MODES

The methods by which the location of an operand is specified in an instruction are referred to as *addressing modes.* (Or) It is a method of finding the EA(Effective address) of the operands in the instructions. The list of addressing modes is shown in the table 2.1

Table 2.1 Generic addressing modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand = Value
Register	R*i*	EA = R*i*
Absolute (Direct)	LOC	EA = LOC
Indirect	(R*i*) (LOC)	EA = [R*i*] EA = [LOC]
Index	X(R*i*)	EA = [R*i*] + X
Base with index	(R*i*,R*j*)	EA = [R*i*] + [R*j*]
Base with index and offset	X(R*i*,R*j*)	EA = [R*i*] + [R*j*] + X
Relative	X(PC)	EA = [PC] + X
Autoincrement	(R*i*)+	EA = [R*i*]; Increment R*i*
Autodecrement	−(R*i*)	Decrement R*i*; EA = [R*i*]

EA = effective address
Value = a signed number

The different addressing modes are explained in the following sections.

2.4.1 IMPLEMENTATION OF VARIABLES AND CONSTANTS

Variables and constants are the simplest data types that are found in computer programs. In assembly language, a variable is denoted by allocating a register or a memory location to hold its value. The two addressing modes to ***access variables*** are:
Register addressing mode, Absolute addressing mode.

Register addressing mode: *In this mode the operand will be a processor register, that is the* name of the register specified in the operand field of the instruction.

Move R1,R2

Absolute addressing mode(Direct addressing mode): In this mode the operand will be a memory location that is the address of the memory location is given explicitly in the instruction.

Move LOC,B

Address and data constants can be represented in assembly language using the Immediate mode.
Immediate addressing mode — The operand is specified explicitly in the instruction, i.e., it is used to specify the value of a source operand. Common convention used is # sign prefix to the operand.

Move 200*immediate*, R0
OR
Move #200,R0

2.4.2 INDIRECTION AND POINTERS

In this addressing mode, the instruction does not give the operand or its address explicitly, instead, it gives the information from which the memory address of the operand can be determined. The determined address is called the ***effective address*** **(EA)** of the operand.

Indirect addressing mode — The effective address of the operand is the contents of a register
or memory location whose address appears in the instruction.
The indirection is denoted by placing the name of the register or the memory address in parentheses as shown in the below example.

Add (R1),R0

Here R1 will hold address of memory location , thus (R1) it indirectly refers to content of memory location.

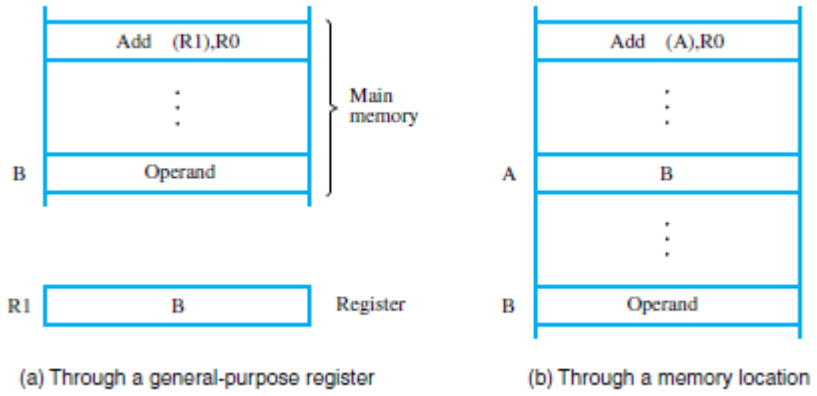


Figure 2.6 Indirect addressing

Indirect mode through general purpose register : **Add (R1),R0** is executed as shown in the above figure 2.6(a). The processor uses the value of B present in register R1. The contents of location B are read. Now this content is added to content of register R0.

Indirect mode through a memory location : **Add (A),R0** is executed as shown in figure 2.6 (b). The processor reads the contents of memory location A. Second read operation is requested using the value B as an address to read the operand.

2.4.3 INDEXING AND ARRAYS

The indexed addressing mode is useful in dealing with lists and arrays.
Indexed addressing mode - The effective address of the operand is generated by adding a constant value to the contents of a register.
A special register or any general purpose register can be used as a register in index register.
Symbolically index mode is represented as

X(Ri)

Where X indicates the constant value and Ri is the name of the register. EA is calculated as

Here the contents of index register are not changed
Ex: Add 20(R1),R2
In figure 2.7(a) The index register R1 contains the address of the memory location and X contains the offset , which is the displacement from this address to the location of the operand.
In figure 2.7(b) The constant X denotes the memory address and the contents of index register as offset. In both the cases, the Effective address is the sum of contents of index register and the memory address.

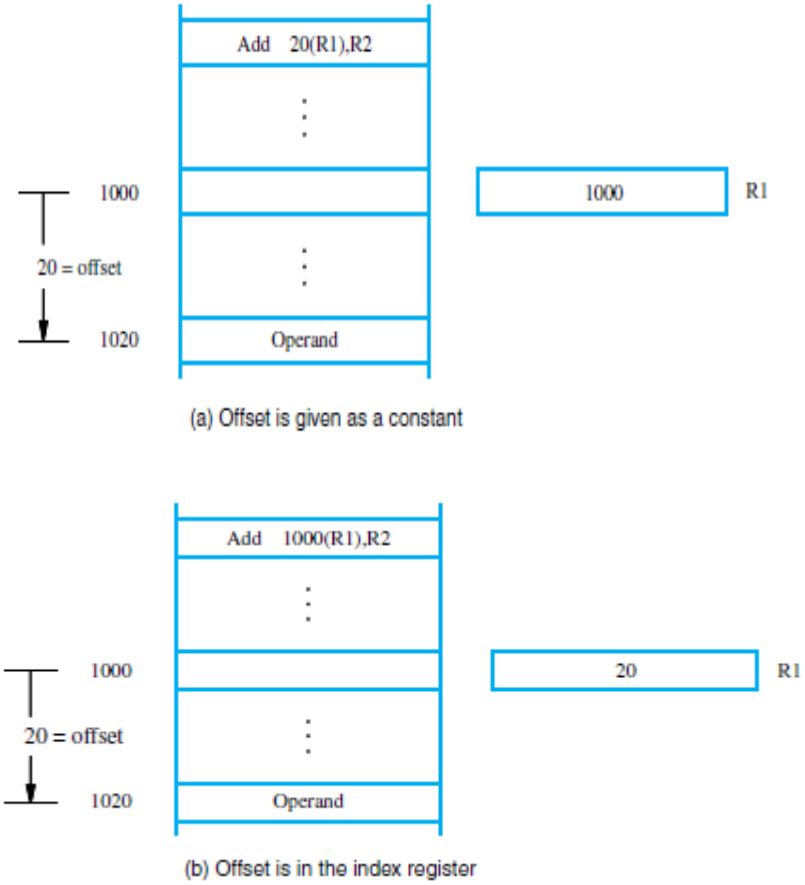


Figure 2.7 Indexed Addressing

The figure 2.8 below represents a two-dimensional array having *n* rows and four columns. Each row contains the entries for one student, and the columns give the IDs and test scores.

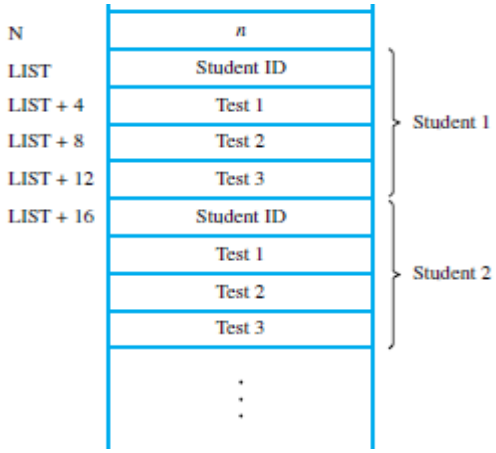


Figure 2.8 A list of student's marks

Index Mode	Assembler Syntax	Addressing Function	Comments
Index	X(Ri)	EA=[Ri]+X	Ri contents are added with constant value X
Base with Index	(Ri,Rj)	EA=[Ri]+[Rj]	Second register acts as base register and it will hold offset.
Base with Index and offset	X(Ri,Rj)	EA=[Ri]+[Rj] +X	Second register acts as base register and it will hold offset. It is added to constant value X

2.4.4 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 program counter in place of the general-purpose register R*i*.

Symbolically it is represented as **X(Ri)**

Effective address is calculated as EA= X + (PC)

2.4.5 ADDITIONAL MODES

There are two additional modes. They are
Auto-increment mode and 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.

It is symbolically represented as (Ri)+

The contents of Ri register is incremented

Ex: Add (R2)+,R0

Here, the content of R2 is added with R0 and then R2 is incremented to point to next item.

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.
It is symbolically represented as -(Ri)
The contents of Ri register is decremented
Ex: Add -(R2),R0
Here, the Contents of R2 is decremented to point to next item in the list. This is added with R0

2.5 ASSEMBLY LANGUAGE

The instructions such as move, for adding, to increment etc.. can be specified by using "mnemonics" such as MOV , INC , ADD , DECR, LOAD,STORE,MOVE etc.

LOAD - To load operand from memory
STORE - To store operand to memory
MOVE - To transfer data from one location to another location/Register

A complete set of such mnemonics and set of rules for their usage constitute an assembly language. This is written in alphanumeric text format called as **source program**. Program written in an assembly language can be automatically translated into machine instructions (**Object program**)by a program called Assembler.

The assembly language instruction can be written as An **opcode** followed by at least one blank space and then preceded by **operands**

Ex : ADD #5,R3
Ex : MOVE A,B

2.5.1 ASSEMBLER DIRECTIVES

Assembler directives are the commands written in assembly language program, which instructs the assembler to perform the task/operation. These are not translated into object program.
Some of the assembler directives are :

S EQU 150

EQU directs the assembler that the symbolic name S must be replaced with memory location Address 150,

ORIGIN 201

Instruct assembler to initiate data block at main memory locations starting from 201

N DATAWORD 40

Inform the assembler that value of N i.e. data value 40 is to be placed in the memory location 201.

ORIGIN 100

States that assembler directive must load machine instructions of the object program in the main
Memory starting from location 100.

END START

End of the program and the label of where program starts

N1 RESERVE 400

Reserve memory block of 400 bytes

2.5.2 ASSEMBLY AND EXECUTION OF PROGRAMS

As the assembler scans through a source program, it tabulates all the symbolic names and the corresponding numerical values in a ***symbol table***. When a name appears for the second time, it is replaced with its value from the table. A problem arises when a name appears as an operand before its value is defined. For example , if a forward branch is required the assembler will not be able to determine the branch target address, because the name referred to has not yet been recorded in the symbol table.

A simple solution to this problem is to have the assembler scan the source program twice. During the first pass, it creates a complete symbol table. At the end of this pass, all names will have been assigned numerical values. The assembler then goes through the source program a second time and substitutes values for all names from the symbol table. Such an assembler is called a ***two-pass assembler.***

The assembler stores the object program on a magnetic disk. Before a program is executed, the object program must be loaded into the memory of the computer. An utility program called a *loader* is needed to load into memory.

During execution of the object program, to detect and report syntax errors, a system software , Debugger is used .

2.6 Basic Input- Output Operations.

I/O operation means by which data are transferred between the processor and the I/O devices (outside world). To transfer data between I/O and processor, the user uses a method called program-controlled I/O. According to this method, rate of data transfer from keyboard to the computer is limited by the speed of the user. However the rate of transfer between processor and display is determined by processor's speed. Thus the speed of transfer between the processor and the I/O devices needs a mechanism to synchronize the transfer of data between them.

The mechanism of data transfer between processor, keyboard and display can be explained as below (figure 2.9)

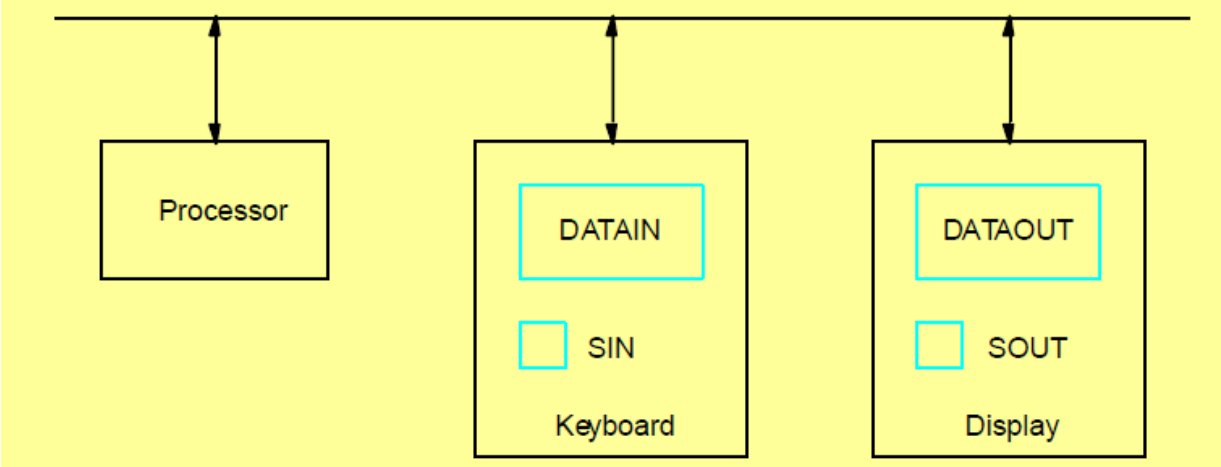


Figure 2.9 : Bus Connection for processor, keyboard and display

Keyboard/display Example:

Whenever the data is typed through the keyboard, it is sent to an 8-bit buffer register called DATAIN, available in keyboard. This availability of data in DATAIN is indicated to processor through a flag bit set in SIN register (SIN is a 1 bit flag). When data is stored in DATAIN, automatically SIN resets from 0 to 1. When SIN is set to 1, the processor reads the data of DATAIN. Once the character is transferred to processor, SIN is cleared to 0 automatically.

Similarly a buffer register DATAOUT, and a status control flag, SOUT are used in display unit. When SOUT is equal to 1, display is ready to receive a character, and thus it displays the character. After transferring character to display, SOUT is cleared to 0. Again SOUT is set to 1 , when display device is ready to receive a second character.

To perform I/O transfers, we need machine instructions that checks the status flags and transfer data between the processor and the I/O devices. To transfer a character from DATAIN to processor register R1 the following sequence of machine instructions are used.

READWAIT Branch to **READWAIT** if **SIN = 0**
Input from **DATAIN** to **R1**

To transfer a character from processor register R1 to display unit register DATAOUT, the following sequence of machine instructions are used.

WRITEWAIT Branch to **WRITEWAIT** if **SOUT = 0**
Output from **R1** to **DATAOUT**

Memory-mapped I/O: In this method, the memory address space will also refer to peripheral device buffer registers such as DATAIN and DATAOUT. (The I/O addresses are available in memory address space).

I/O mapped I/O : In this method, separate address space is available for I/O addresses and memory addresses.

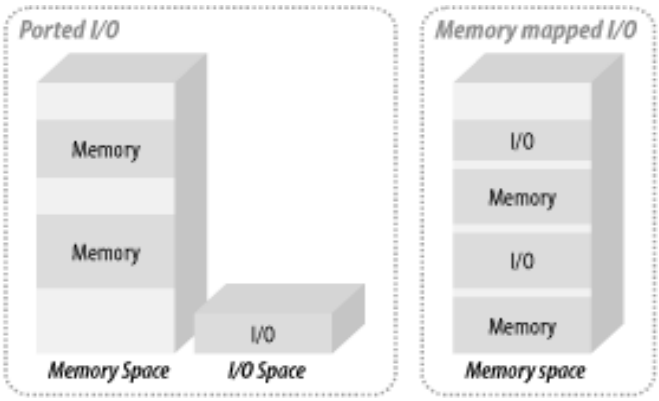


Figure 2.10 I/o mapped I/O and Memory mapped I/O.

Accessing I/O Devices, Interrupts, Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Controlling Device requests, Exceptions, Direct Memory Access, Bus arbitration, Buses, Synchronous bus, Asynchronous bus, Interface Circuits, Parallel port and Serial port (Basic concept only), Standard I/O Interfaces (Basic concepts only), Peripheral Component Interconnect (PCI) Bus , SCSI Bus(Basic concepts only), Universal Serial Bus (USB) (Basic concepts only)

UNIT- 3: Basic Processing Unit

3.1 Some Fundamental Concepts

To execute a program the processor fetches one instruction at a time. Processor uses PC to keep track of the address of the memory location containing the next instruction to be fetched. IR is used to store the instruction to be executed.

Following are the steps used to execute an instruction:

1. The contents of the PC is used to fetch the instruction to be executed and loaded into IR.

   $$IR \leftarrow [[PC]]$$

2. Assuming that the memory is byte addressable and each instruction occupies 4 bytes, increment the contents of the PC by 4 (Both steps 1 and 2 are collectively referred as Fetch phase).

   $$PC \leftarrow [PC] + 4$$

3. Carry out the actions specified by the instruction in the IR (execution phase).

   The building blocks of a processor can be organized and interconnected in a number of ways. The simplest way of organising is using a single-bus as shown in Figure 3.1

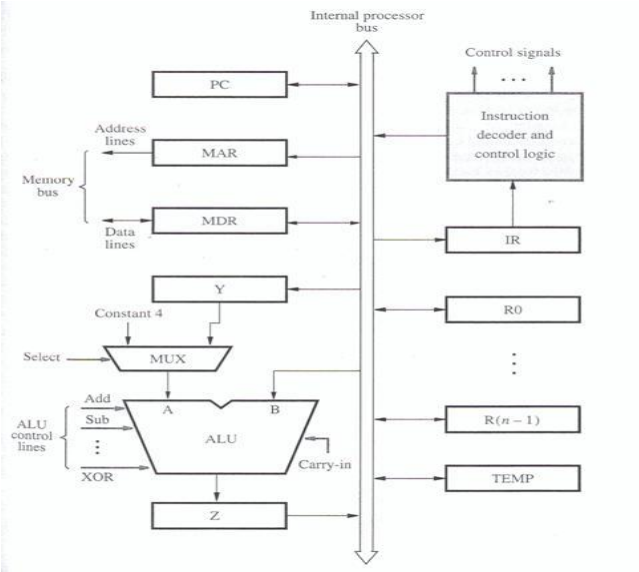


Figure 3.1 Single-bus organization of the datapath inside a processor

The ALU and all the registers are interconnected via a single common bus. This bus is the internal bus to the processor. An external bus is used to connect the processor to the memory and I/O devices.

The data and address lines of the external memory bus is connected to the internal processor bus through MDR and MAR respectively. Register MDR has two inputs and two outputs. Data may be loaded into MDR either from the memory bus or from the internal processor bus. The data stored in MDR may be placed on either bus.

The input of MAR is connected to the internal bus, and its output is connected to the external bus. The control lines of the memory bus are connected to the instruction decoder and control logic. Decoder and control logic unit is responsible for issuing the signals that control the operation of all the units inside the processor and for interacting with the memory bus. The registers, the ALU and the interconnecting bus are collectively called as **datapath**.

The number of registers used varies from one processor to another. Some of the registers are provided to the users for general purpose. Some others may be dedicated as special purpose registers like index registers or stack pointers.

The three registers Y, Z and TEMP shown in the figure are transparent to the programmer and are not explicitly referenced by any instruction. They are used only by the processor for temporary storage during the execution of some instructions. The MUX selects either the output of register Y (input from the bus) or a constant value 4 (assuming the each instruction occupies 4 bytes) to be provided as input A to the ALU based on the control input Select. The constant 4 is used to increment the contents of the program counter.

Following are the sequence of operations (with few exceptions) required for executing an instruction –

1. Transfer a word of data from one of the processor register to ALU or another processor register.
2. Perform an arithmetic or logic operation and store the result in a processor register.
3. Fetch the contents of a required memory location and load them into a processor register.
4. Store a word of data from a processor register into a required memory location.

3.1.1 Register transfers

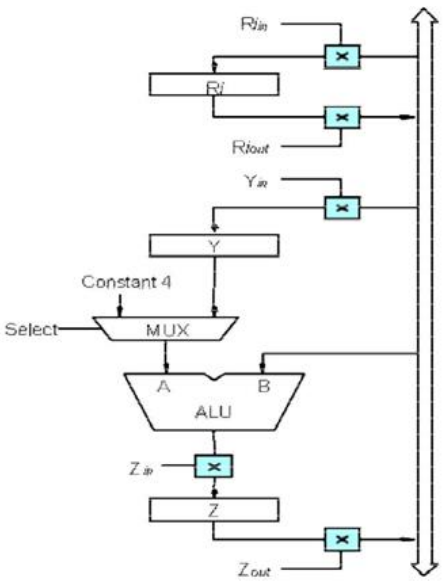


Figure 3.2 Input and output gating for the registers.

- Instruction execution involves a sequence of steps in which data are transferred from one register to another.

- For each register two control signals are used to place the contents of that register on the bus or to load the data on the bus into register.(as shown in Figure 3.2)

- The input and output of register Ri are connected to the bus via switches controlled by Ri_{in} and Ri_{out} respectively.
- When Ri_{in} is set to 1, the data on the bus are loaded into register Ri.

- When Ri_{out} is set to 1, the contents of register Ri are placed on the bus.
- While Ri_{out} is equal to 0, the bus can be used for transferring data from other registers.

Example

- Suppose we wish to transfer the contents of register R1 to register R4. This can be accomplished as follows.

- Enable the output of registers R1 by setting $R1_{out}$ to 1. This places the contents of R1 on the processor bus.

- Enable the input of register R4 by setting $R4_{in}$ to 1. This loads data from the processor bus into register R4.

All operations and data transfers within the processor take place within time periods defined by the processor clock. The control signals that govern a particular transfer are asserted at the start of the clock cycle. The registers consist of edge-triggered flip-flops. Hence, at the next active edge of the clock, the flip-flops of register R4 will load the data present at their inputs. At the same time, the control signals $R1_{out}$ and $R4_{in}$ will return to zero.

There are other schemes that are possible. For example, data transfers may use both the rising and falling edge of the clock. If edge-triggered flip-flops are not used, two or more clock signals may be needed to transfer the data. This is known as **multiphase clocking**.

The Figure 3.3 shows the implementation of one bit of register Ri. A two input multiplexer is used to select the data given to the input of an edge-triggered D flip-flop. When Ri_{in} is equal to 1, the multiplexer selects the data on the bus and loads into the flip-flop at the rising edge of the clock. When Ri_{in} is equal to zero, the multiplexer feeds back the value that is stored currently in the flip-flop. The output Q of the flip-flop is connected to the bus through a tri-state gate. When Ri_{out} is equal to zero, the gate's output is in the high impedance (electrically disconnected) state which corresponds to an open circuit state of a switch. When Ri_{out} is equal to 1, the gate drives the bus to 1 or zero, based on the value of Q.

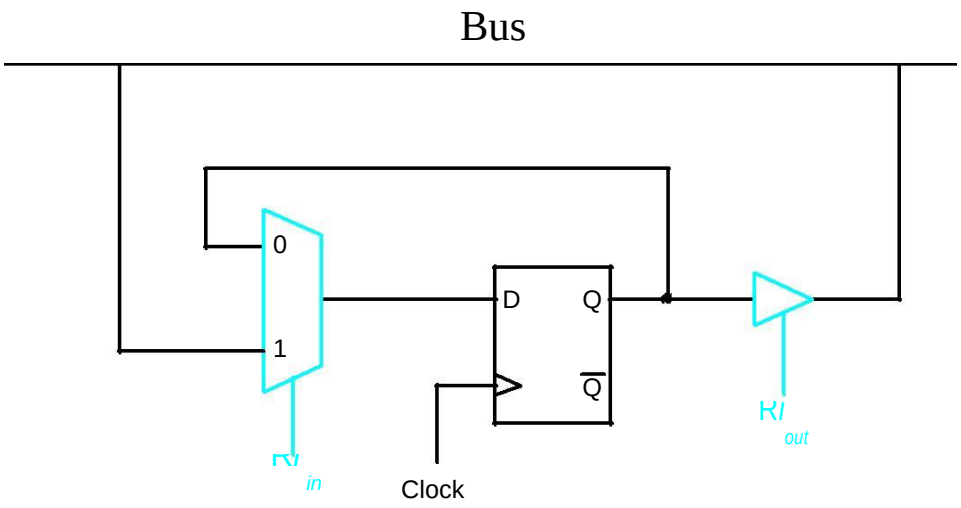


Figure 3.3. Input and output gating for one register bit.

3.1.2 Performing an Arithmetic or Logic operation

The ALU is a combinational circuit that has no internal storage. ALU gets the two operands from MUX and bus. The result is temporarily stored in register Z. The sequence of operations to add the contents of register R1 to that of R2 and store the result in R3 is as given below-

1. $R1_{out}$, Y_{in}
2. $R2_{out,}$ SelectY, Add, Z_{in}
3. Z_{out}, $R3_{in}$

In step 1, the output of register R1 and the input of register Y are enabled, causing the contents of R1 to be transferred over the bus to

Y.

In step 2, the multiplexer's select signal is set to Select Y, causing the multiplexer to gate the contents of register Y to input A of the ALU. At the same time, the contents of register R2 are gated onto the bus and, hence, to input B. The function performed by the ALU depends on the signals applied to its control lines. In this case, the ADD line is set to 1, causing the output of the ALU to be the sum of the two numbers at inputs A and B. This sum is loaded into register Z because its input control signal is activated.

In step 3, the contents of register Z are transferred to the destination register R3.

This last transfer cannot be carried out during step 2, because only one register output can be connected to the bus during any clock cycle.

3.1.3 Fetching a word from memory

To fetch a word from the memory, the processor has to specify the address of the memory location and request a Read operation. This applies for both instruction in a program or an operand specified by an instruction. The processor transfers the required address to the MAR, whose output is connected to the address lines of the memory bus. At the same time , the processor uses the control lines of the memory bus to indicate that a Read operation is needed.

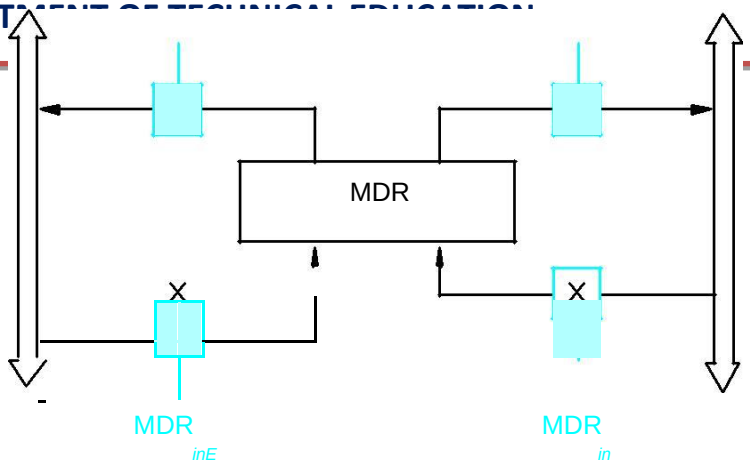


Figure 3.4. Connection and control signals for register MDR.

The connections for register MDR is as shown in Figure 3.4. The control signals MDR_{in} and MDR_{out} control the connection to the internal bus. The control signals MDR_{inE} and MDR_{outE} control the connection to the external bus.

During memory Read and Write operations, the timing of internal processor operations must be co-ordinated with the response of the device on the memory bus. When the requested data are received from the memory they are stored in register MDR, from where they can be transferred to other registers in the processor. To accommodate the difference in response times by different devices, the processor waits until it receives an indication that the requested read operation has been completed. MFC (Memory-Function-Completed) control signal is used for this purpose. The addressed device sets MFC to 1 to indicate that the contents of the specified locations are available on the data lines of the memory bus.

Example: The actions needed to execute the instruction **Move (R1), R2** are

1. MAR ← [R1]
2. Start a Read operation on the memory bus

3. Wait for the MFC response from the memory
4. Load MDR from the memory bus
5. R2 ← [MDR]

The above operations may be carried as separate steps or some can be combined into a single step. In the above example, action 3 requires one or more clock cycles, whereas the rest of the actions need only one clock cycle each.

For simplicity, let us assume that the output of MAR is enabled all the time. Thus the contents of MAR are always available on the address lines of the memory bus.

When a new address is loaded into MAR, it will appear on the memory bus at the beginning of the next clock cycle.(in Figure 3.5) A read control signal is activated at the same time MAR is loaded. This means memory read operations requires three steps, which can be described by the signals being activated as follows

1. $R1_{out}$, MAR_{in}, Read
2. MDR_{inE}, WMFC
3. MDR_{out}, $R2_{in}$

The processor's control circuitry waits for the arrival of WMFC control signal.

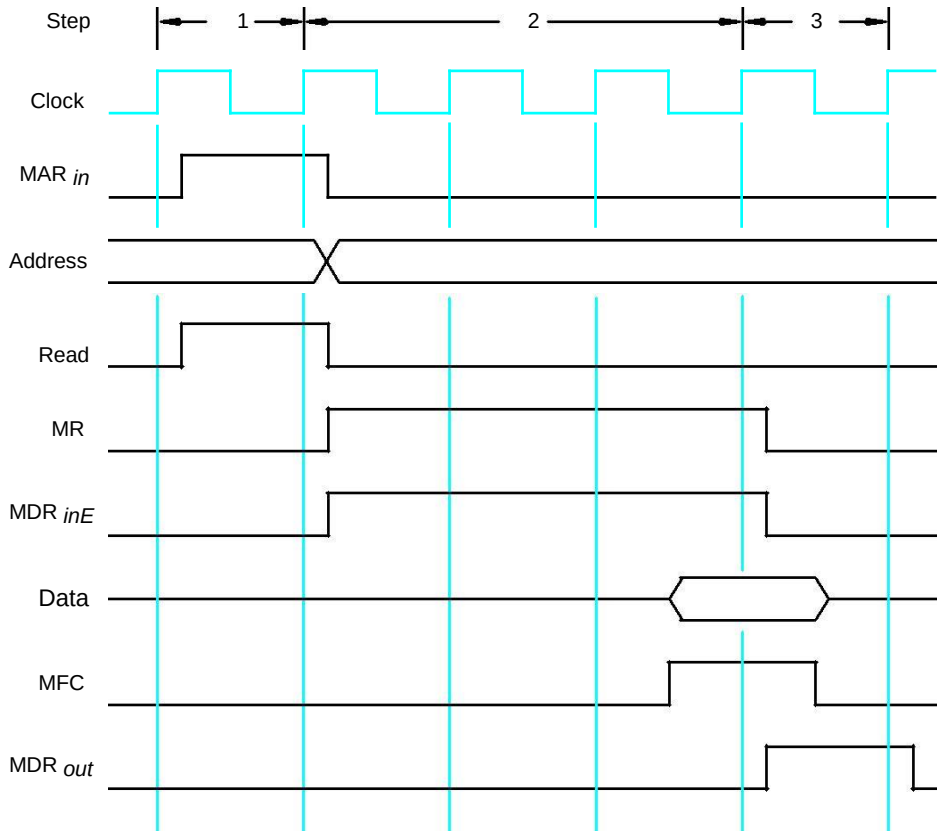


Figure 3.5. Timing of a memory Read operation.

3.1.4 Storing a word in memory

Writing a word into a memory location is same as fetching a word from

the memory. The desired address is loaded into MAR and the data to be written are loaded into MDR, and a write command is issued.

Example

Executing the instruction Move R2,(R1) requires the following steps-

1. $R1_{out}$,MAR_{in}
2. $R2_{out}$,MDR_{in},Write
3. MDR_{outE},WMFC

3.2 Execution of a Complete Instruction

Execution of the instruction Add (R3), R1 requires the following actions-

1) Fetch the instruction
2) Fetch the first operand (the contents of the memory location pointed to by R3)
3) Perform the addition
4) Load the result into R1

The sequence of control steps required to execute the same on a single bus organization are:

Step	Action
1	PC_{out} , MAR_{in} , Read,Select4,Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMFC
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMFC
6	MDR_{out} , SelectY,Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Figure 3.6. Control sequence for execution of the instruction Add (R3),R1.

Step 1 initiates instruction fetch operation by loading the require address from PC to MAR and sending a Read request to the memory. The Select signal is set to Select4 in order to select the constant 4 by the multiplexer to increment the PC, which is performed in step 2. In step 3, the word fetched from the memory is loaded into IR. Steps 1 through 3 is instruction fetch phase, which is same for all instructions. Instruction decoding takes place at the beginning of step 4. This enables the control circuitry to activate the control signals for the remaining steps which constitutes the execution phase. The contents of register R3 are transferred to MAR in step 4 and a memory Read is initiated. The contents of R1 are then transferred to register Y in step 5, to prepare for the addition operation. Once the Read operation is completed, the operand is available in MDR and the addition operation is performed in step 6. The contents of MDR is available to ALU's B input, as it is gated to the bus. The register Y is selected as second input and the operation is performed. The sum is stored in register Z and is transferred to register R1 in step 7.

3.2.1 Execution of a branch Instructions

A branch instruction replaces the contents of PC with the branch target address, which is usually obtained by adding an offset X given in the branch instruction. The offset X is usually the difference between the branch target address and the address immediately following the branch instruction. For example, if the branch instruction is at location 3000 and if the target address is 3050, the value of X must be 46 (Target address - PC value).

Figure 3.7 gives the control sequence to implement an unconditional branch instruction. Fetch phase ends when the instruction is loaded into IR. The offset value is extracted from the IR by the instruction decoding circuit. The value of the updated PC is already available in register Y, The offset Y is now gated to the bus in step 4 and an

addition operation is performed. The result is loaded into the PC in step 5.

Conditional branch:

For executing the conditional branch instruction, we need to check the status of condition codes before loading a new value into the PC

For example, to execute a Branch-on-negative (Branch < 0) instruction, step 4 in Figure 3.7 is replaced with

Offset-Field-Of-IR$_{out}$, Add, Z$_{in}$, If N=0 then End.

The processor returns to step 1 immediately after step 4 if N=0. If N=1, step 5 is performed to load the new value into the PC, thus performing the branch operation.

Step	Action
1	PC_{out} , MAR_{in} , Read, Select4,Add, Z_{in}
2	Z_{out}, PC_{in} , Y_{in}, WMFC
3	MDR_{out} , IR_{in}
4	Offset-field-of-IR_{out}, Add, Z_{in}
5	Z_{out}, PC_{in} , End

Figure 3.7. Control sequence for an unconditional branch instruction.

3.3 Multiple Bus Organization

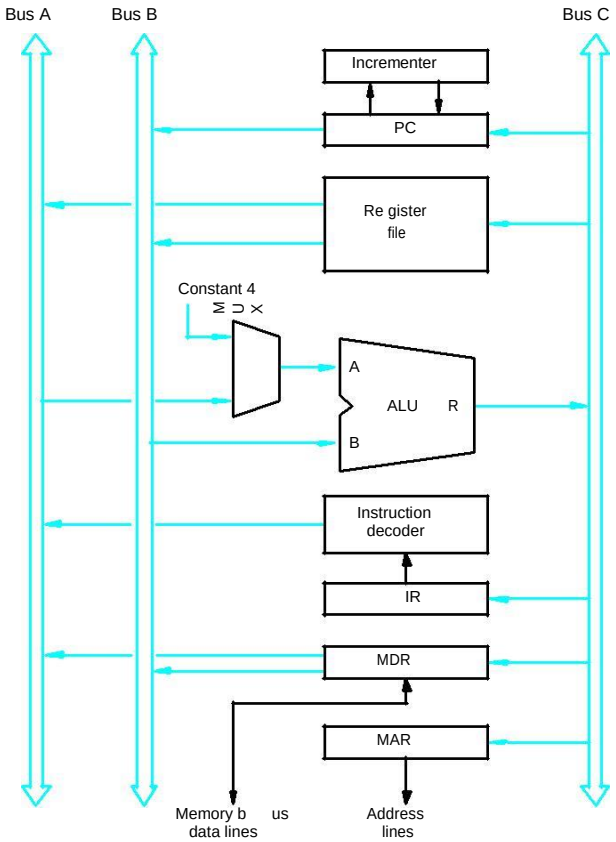


Figure 3.8 Three-bus organization of the datapath.

To reduce the number of steps needed for executing an instruction, most commercial processors provide multiple internal paths that enable transfers to take place in parallel. Figure 3.8 shows a three-bus structure to connect the registers and ALU of a processor.

All the general purpose registers are combined into a single block called **register file.** The register file in Figure 3.8 has 3 ports. There are two output ports which enables to access two different registers simultaneously and have contents placed on buses A and B. The 3rd port allows data on bus C to be loaded into third register in the same clock cycle.

The ALU may simply pass one of its two operands without modifying to bus C. R=A or R=B is the control signal for such an operation. Using the incrementer to PC eliminates the need to add 4 to the PC using the main ALU. The source for constant 4 at the multiplexer can be used to increment other addresses in case of LoadMultiple and StoreMultiple instructions.

The control sequence for executing the instruction **Add R4, R5, R6** is shown in Figure 3.9. In step 1, the contents of PC are loaded in to MAR using R=B control signal. At the same time, PC is incremented by 4 and the incremented value is loaded into the PC at the end of the clock cycle. In step 2, processor waits for MFC and loads the data received into MDR. In step 3, the contents of MDR are transferred to IR.

Finally the execution is performed using step 4. Thus, by providing more paths for data transfers, the number of clock cycles needed to execute an instruction is significantly reduced.

Step	Action
1	PC_{out}, R=B, MAR_{in}, Read, IncPC
2	WMFC
3	MDR_{outB}, R=B, IR_{in}
4	$R4_{outA}$, $R5_{outB}$, SelectA, Add, $R6_{in}$, End

Figure 3.9 Control sequence for the instruction Add R4,R5,R6 for the three-bus organization in Figure 3.8

3.4 Hardwired Control (basic block diagram only)

Processor needs to generate control signals in a proper sequence to execute instructions. There are 2 techniques, using which a processor can generate the control signals-

1. Hardwired control
2. Micro programmed control

Consider the sequence of control signals as shown in Figure 3.10. Each step here requires one clock period to complete. A counter, as shown in the Figure 3.11, keeps

track of the control steps. Each count of this counter corresponds to one control step.

Step	Action
1	PC_{out} , MAR_{in} , Read,Select4,Add, Z_{in}
2	Z_{out} , PC_{in} , Y_{in} , WMFC
3	MDR_{out} , IR_{in}
4	$R3_{out}$, MAR_{in} , Read
5	$R1_{out}$, Y_{in} , WMFC
6	MDR_{out} , SelectY,Add, Z_{in}
7	Z_{out} , $R1_{in}$, End

Figure 3.10. Control sequence for execution of the instruction Add (R3),R1.

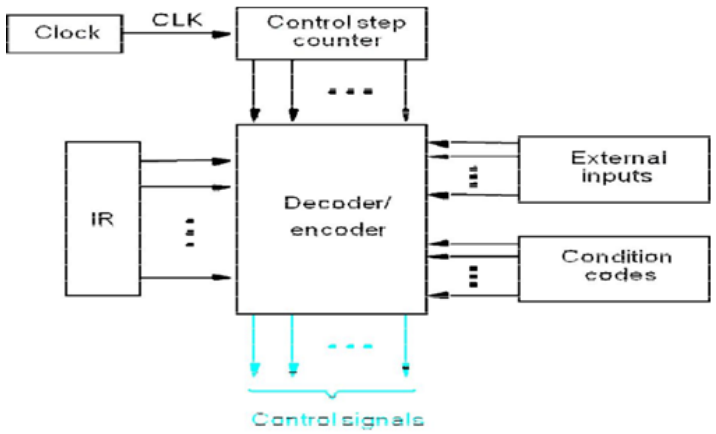
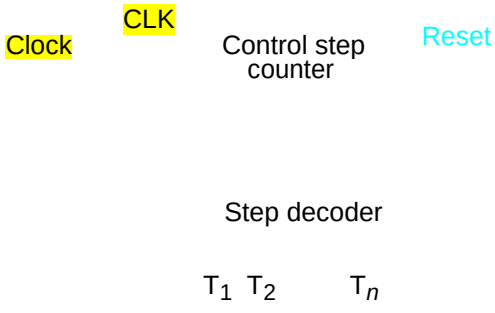


Figure 3.11. Control unit organization.

The necessary control signals can be determined by the information shown below-

1. Contents of the control step counter.
2. Contents of the Instruction register (IR)
3. Contents of the condition code flags
4. External input signals, such as MFC and interrupt requests.

The decoder/encoder is a combinational circuit that generates the necessary control outputs, depending on the state of all its inputs. The Figure 3.12 shows detailed block diagram after separating decoding and encoding functions.



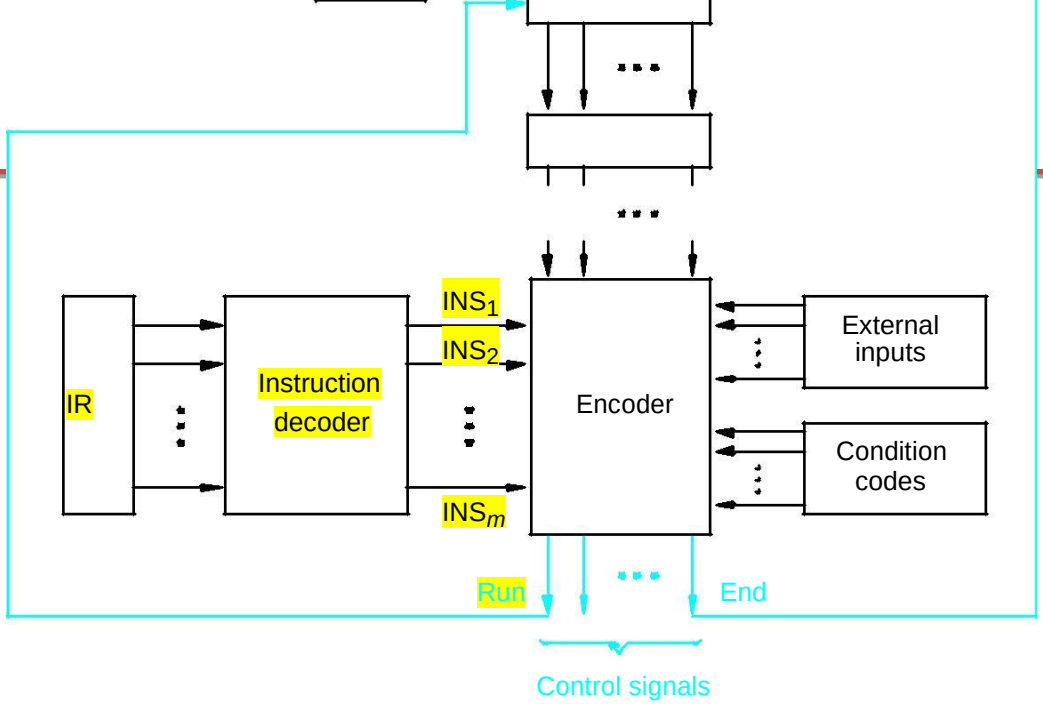


Figure 3.12. Separation of the decoding and encoding functions.

The step decoder provides a separate signal line for each step. In the same way, the output of the instruction decoder consists of separate line for each machine instruction. For any instruction loaded in IR, one of the output lines INS_1 through INS_m is set to 1, and rest of the lines are set to zero. The encoder block combines all inputs to generate individual control signals such as Y_{in}, PC_{out}, Add, End and so on.

As an example, the control signal Z_{in}, generated by the encoder for a single bus organization of a processor is as shown in Figure 3.13.

Generating Z_{in}

$$Z_{in} = T_1 + T_6 \cdot ADD + T_4 \cdot BR + \ldots$$

Branch Add

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

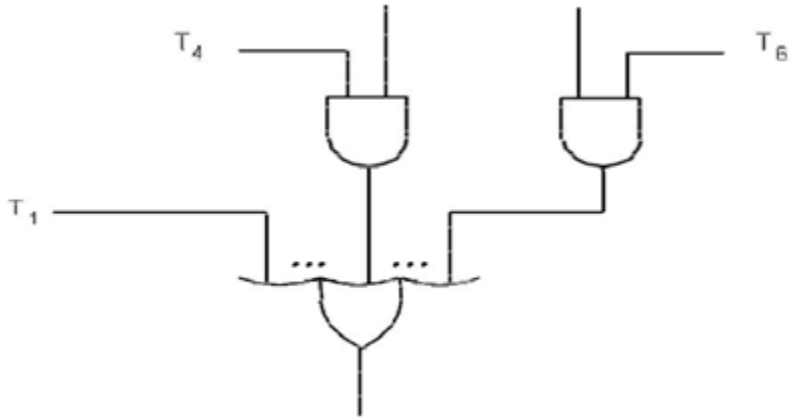


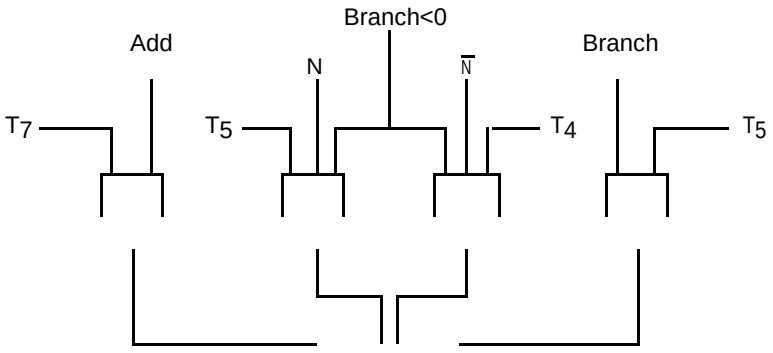
Figure 3.13 Generation of the Z_{in} control signal for the single-bus organization of the processor.

Z_{in} is set to 1, during the time slot T1, for all instructions, during T6 for an Add instruction, during T4 for an unconditional branch instruction and so on.

As an other example, the control signal End, generated by the encoder for a single bus organization of a processor is as shown in Figure 3.14.

Generating End

$$End = T_7 \cdot ADD + T_5 \cdot BR + (T_5 \cdot N + T_4 \cdot \overline{N}) \cdot BRN + \ldots$$



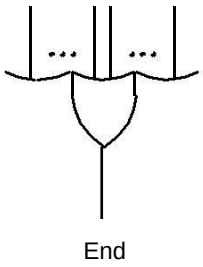


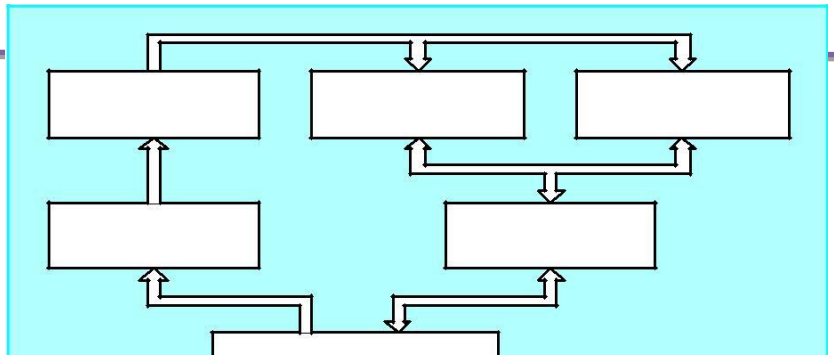
Figure 3.14. Generation of the End control signal.

The End signal starts the fetch cycle of a new instruction by reseting the control step counter to its starting value.

Figure 3.12 has a control signal RUN. When RUN is set to 1, it causes the counter to be incremented by 1 at the end of every clock cycle. When RUN is zero, the counter stops counting. This is required whenever WMFC signal is issued, to make the processor to wait for the reply from the memory.

The controlled hardware shown in the Figures 3.11 and 3.12, can be viewed as a state machine that changes its state from one to another in every clock cycle, based on the contents of the IR, the condition codes and the external inputs. The outputs of this state machine are the control signals. The wiring of the logic elements determine the sequence of operations carried out by this machine. Hence it is called **Hardwired control**. Hardwired system can operate at high speed; but with little flexibility. The complexity of the instruction set, it can implement is limited.

3.4.1 A complete processor



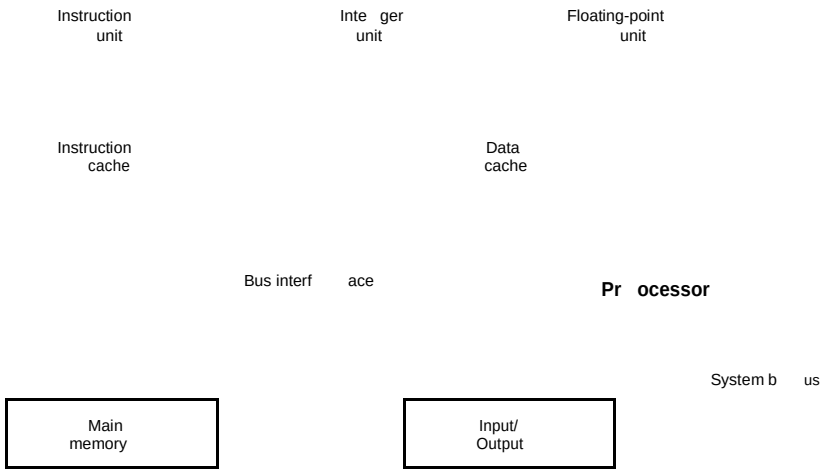


Figure 3.15. Block diagram of a complete processor .

The 3.15 shows how a complete processor can be designed. This structure has an instruction unit, which fetches instruction from an instruction cache or from the main memory when the required instruction is not available in the cache. It uses separate processing units for handling integer data and floating point data. A data cache is inserted between the memory and these units. There can be processors which use a single cache for instructions and data. The processor is connected to the system bus and a bus interface is used to connect to the rest of the computer.

3.5 Basic organization of Micro programmed Control Unit.

Control signals are generated by a program similar to machine language programs. Some of the common terminologies used are-

Control Word (CW) is a word whose individual bits represent the various control signals.

A sequence of control words corresponding to the control sequence of a machine instruction constitutes the **microroutine** for that instruction.

The individual control words in a microroutine are referred to **microinstructions**.

The microroutines for all instructions in the instruction set of a computer are stored in a special memory called the **control store.** The control unit generates the control signals for any instruction by sequentially reading the CW's of the corresponding microroutine from the control store. Thus, the organization of a control unit is as shown in Figure 3.17.

Micro - instruction		PC *in*	PC *out*	R *in*	Read	R *out*	IR *in*	Y *in*	Select	Add	Z *in*	Z *out*	R1 *out*	R1 *in*	R3 *out*	WMFC	End	
1		0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	
2		1	0	0	0	0	0	1	0	0	0	1	0	0	0	1	0	
3		0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	
4		0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	0	
5		0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	
6		0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	
7		0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	

Figure 3.16 An example of microinstructions for Add(R3),R1.

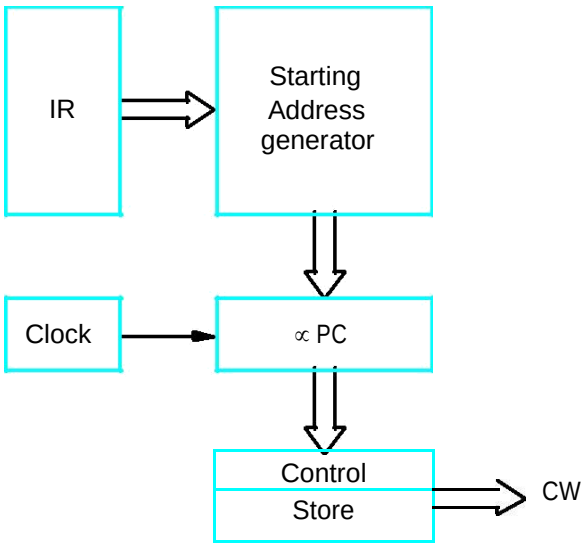


Figure 3.17.Basic organization of a microprogrammed control unit.

Figure 3.16 gives the control words corresponding to the seven steps for the execution of Add (R3),R1 (shown in Figure 3.6). Each of the control steps in the control sequence of an instruction defines a unique combination of 1s and 0s in the control word .

Address	Microinstruction
0	PC_{out} , MAR_{in} , Read, Select4, Add, Z_{in}
1	Z_{out} , PC_{in} , Y_{in} , WMFC
2	MDR_{out} , IR_{in}
3	Branch to starting address of appropriate microroutine
. . .	. . .
25	If N=0, then branch to microinstruction 0
26	Offset-field-of-IR_{out} , SelectY, Add, Z_{in}
27	Z_{out} , PC_{in} , End

Figure 3.18. Microroutine for the instruction Branch < 0.

In Figure 3.17, a microprogram counter (µPC) is used to read the control words sequentially from the control store. Every time, when an instruction is loaded into IR, the output of "starting address generator" is loaded into the µPC. To deliver the correct sequence of control signals to various parts of the processor, the µPC is automatically incremented by the clock so that, the successive microinstructions are read from the control store. The organization shown in Figure 3.17, cannot handle the situation when the control unit is required to check the status of the condition codes or external inputs to choose between alternative courses of action. In microprgrammed control, an alterantive approach is to use conditional branch microinstruction, that specifies which of the external inputs, condition codes, or bits of IR, should be checked as condition for branching to take place.

The microroutine for the Branch < 0 is as shown in Figure 3.18. After loading the Branch < 0 instruction into IR, the Branch microinstruction transfers the control to the corresponding microroutine. The address of the microroutine is the output of the starting address generator block in Figure 3.17. The microinstruction at that location tests the N bit of the condition codes. If N=0, a branch takes place to fetch a new machine instruction, otherwise, the microinstruction at the subsequent location is executed.

Figure 3.19 shows the organization of the control unit to allow the conditional branching in the microprogram. The starting address generator block of Figure 3.17 becomes the starting and the branch address generator. This block loads the new address into µPC when a microinstruction instructs it to do so. The conditional branching is implemented using external inputs, condition codes as well as the contents of the IR. The µPC is incremented every time a new instruction is fetched from the microprogram memory except in the following situation-

1. When a new instruction is loaded into the IR, the starting address of the microroutine of that instruction is loaded into

µPC.

2. When a branch microinstruction is encountered, the µPC is loaded with branch address if the branch condition is satisfied.
3. When an End microinstruction is encountered, the address of the first control word in the microroutine for the instruction fetch cycle is loaded into the µPC.

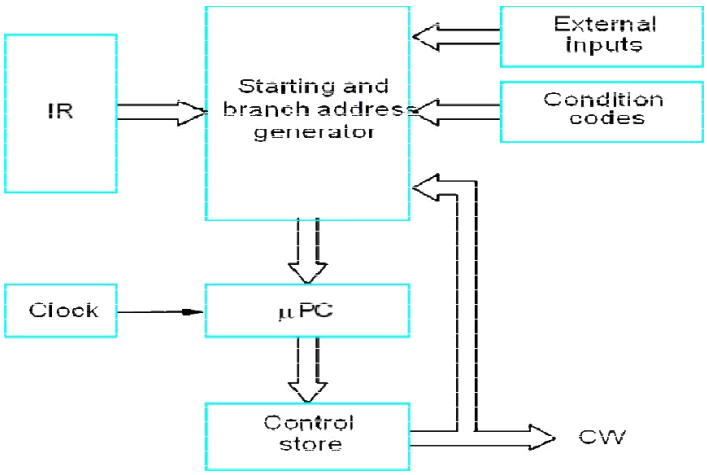


Figure 3.19. Organization of the control unit to allow the conditional branching in the microprogram.

UNIT- 4: Input output organization

4.1 ACCESSING I/O DEVICES

A simple arrangement to connect I/O devices to a computer can be done using a single bus arrangement as in figure 4.1. The bus enables all the devices connected to it to exchange information. A bus consists of three sets of lines, which are used to carry address, data, and control signals. All the I/O devices are assigned a unique set of addresses. First processor places a particular address on the address line. The respective device recognizes this address and responds to the commands issued on the control lines. The processor requests either a read/write operation, and the requested data are transferred over the data lines. The arrangement where I/O devices and the memory share the same address space, is called memory-mapped I/O.

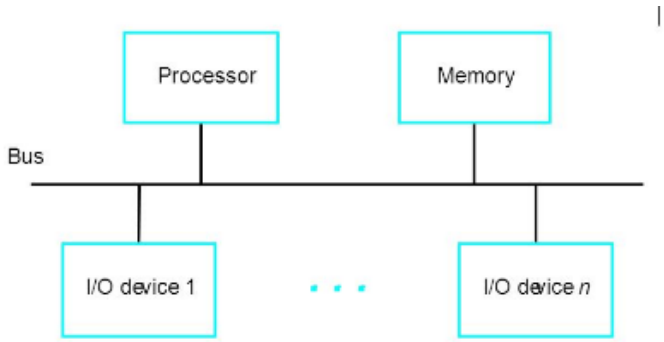


Figure 4.1. A single-bus structure.

The hardware required to connect an I/O device to the bus (I/O Interface) is shown in the figure 4.2

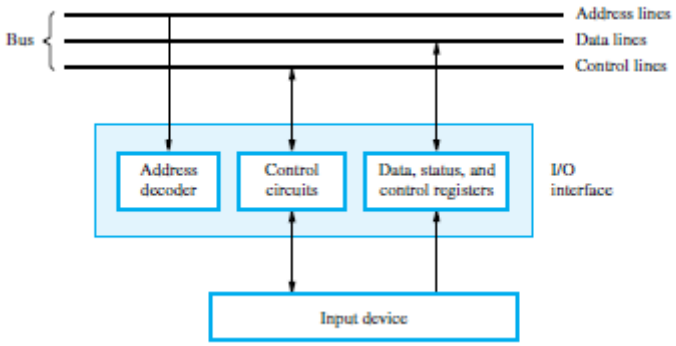


Figure 4.2 I/O interface for an output device

I/O interface is the hardware present between the I/O device and the bus. The interface consists of 3 parts – address decoder, control circuits and data and status registers. The address

decoder decodes the address available on the address lines. The data registers holds information relevant to operation of the I/O device. The control circuits coordinate the I/O transfers. This interface thus compensates the rate of transfer between the I/O device and the Processor.

Program controlled I/O : To transfer data between I/O and processor, the user uses a method called program-controlled I/O. According to this method, rate of data transfer from keyboard to the computer is limited by the speed of the user. In this method, the processor repeatedly checks the status flags to have synchronization between processor and the I/O device.

Accordingly the processor communicates with the I/O devices. i.e. processor polls the device. There are 2 mechanisms to implement I/O operations.

1. Interrupts
2. Direct memory access.

4.2 INTERRUPTS

During computation when a program needs I/O devices help, it constantly checks for I/O status. Meanwhile the processor spends time waiting for longer duration. To avoid processor waiting, a facility is provided for an I/O device to inform the processor only when the request is needed. It can do so by sending a hardware signal called an Interrupt to the processor. Thus the processor attends the interrupts and resumes back to its processing.

Interrupt service Routine : The routine executed in response to an interrupt request is called the interrupt-service routine, which is the PRINT routine in our example. Interrupts bear considerable resemblance to subroutine calls.

Consider an example of continuous computations to be performed and the results to be printed on a printer.

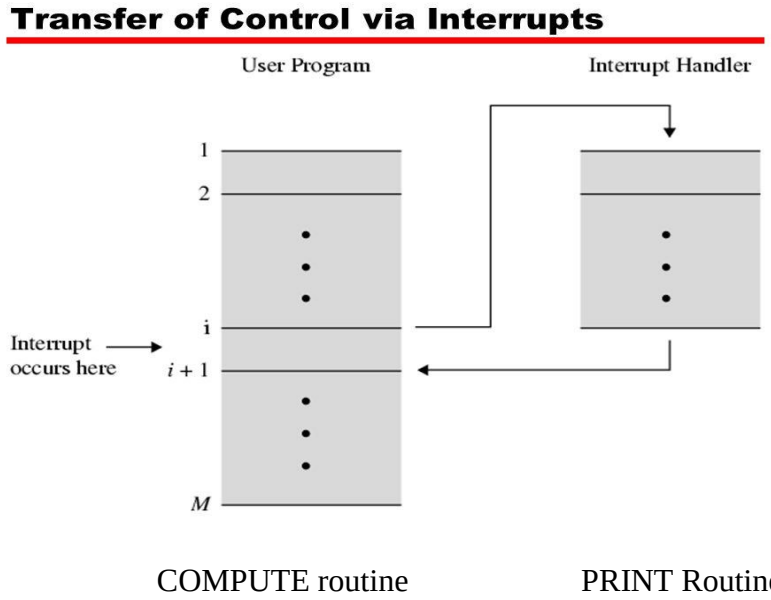


Figure 4.3: Transfer of control through the use of Interrupts.

Consider an example program having two routines **COMPUTE** and **PRINT**. This COMPUTE routine produces a set of lines of output, to be printed by PRINT routine. These are done by repeatedly executing COMPUTE routine and then PRINT routine. The printer accepts request for printing only one line at a time. Hence printer should send one line of text, wait it to be printed, then send the second line and so on, until all the lines printed. In this case processor spends certain amount of time to attend printer. It could be overcome by running both routines at the same time in parallel. First COMPUTE routine is executed to produce first n lines of output, then PRINT routine is executed to send first n lines to the printer. Here instead of waiting for the lines to be printed, PRINT routine may be temporarily suspended and execution of COMPUTE is continued. Thus whenever the printer becomes ready, it alerts the processor by sending an interrupt request signal. In response, the processor interrupts execution of COMPUTE routine and transfers the control to PRINT routine.

4.2.1 INTERRUPT HARDWARE:

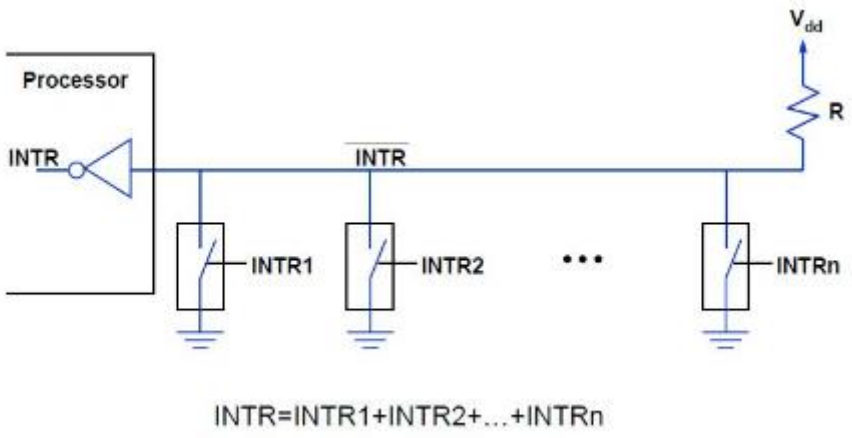


Figure 4.4 : an equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

An I/O device requests can interrupt by activating a bus line called interrupt-request. A single interrupt-request line can be used to serve n devices as in the above figure. All devices are connected to the line through switches to ground. When an interrupt is requested, a device closes its associated switch. If all the switches are open, means all interrupt-request signals INTR1 to INTRn are inactive. Thus the voltage on the interrupt-request line will be equal to V_{dd}, which is the inactive state of the line. Whenever a device requests can interrupt, by closing the switch, the voltage on line will drop to 0. This will cause the interrupt-request signal INTR, which is received by the processor, to go to 1. The value of INTR is the logical OR of the individual device requests, since the one or more switches are closed and line voltage is dropped to 0. In the above figure, special gates called, Open-collector (for bipolar circuits) or open-drain(for MOS circuits) are used to drive the INTR line.

4.2.2 ENABLING AND DISABLING INTERRUPTS

The possibility is to have the processor hardware to ignore the interrupt-request line until the first instruction of the interrupt-service routine is executed and completed**.** Now the interrupt-disable instruction can be used as the first instruction in interrupt-service routine. By this, the programmer can ensure that until the interrupt-enable instruction is executed, no further interruptions will occur. Typically the last instruction in the interrupt-service routine will be the interrupt-enable instruction, before the Return-from-interrupt instruction.

The second possibility, with only one interrupt-request line is explained as below. The processor automatically disables interrupts before the execution of interrupt-service routine. Next it saves the contents of PC and the program status register (PS) on the stack. One bit in

the Ps register, called interrupt-enable indicates whether interrupts are enabled or not. An interrupt request is received, will be executed if interrupt-enable bit is set to 1. The processor resets the interrupt-enable bit in PS register, thus disabling the further interruptions. When a return-from-interrupt instruction is executed, interrupt-enable bit is set back to 1, and hence, interrupts are again enabled.

4.2.3 HANDLING MULTIPLE DEVICES

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

1. How the processor can recognize the device requesting an interrupt?
2. Different devices will require different interrupt-service routines. How the processor can obtain the starting address of the appropriate routine in each case?
3. Should a device be allowed to interrupt the processor while another is being served?
4. How to handle 2 or more interrupt requests simultaneously?

These problems are resolved in many ways which vary from one computer to another. The modern computers are capable of performing above mentioned tasks. Such capabilities are implemented by using vectored interrupts and interrupt nesting.

The interrupt-service routine poll all the devices connected to the bus. The first device with its IRQ (Interrupt request) bit set is recognized as the device to be serviced. The appropriate subroutine is called to provide the requested service.

Vectored Interrupt:

To reduce the time taken in the polling process to identify the device, a device requesting an interrupt may identify itself directly to the processor. So the processor immediately starts executing the corresponding interrupt-service routine. This approach is called Vectored interrupts.

- 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 (length typically in the range of 4 to 8 bits) supplied by the processor indicates the starting address of the ISR for the device.
- 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. The I/O device responds by sending its interrupt-vector code and then turn off the INTR signal.

Interrupt Nesting:

For some devices, long delay in responding to an interrupt request may lead to errors in the operation of computer. Such interrupts are acknowledged and serviced even though processor is executing an ISR for another device. This mechanism is called interrupt nesting.

a) Multiple-level priority:

- We can 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 of 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.

b)Privileged Instruction:

The processor status word is used, in which the processor's priority is encoded in a few bits. This can be changed by program instructions that write into the PS. These instructions called privileged instructions can be executed only while processor is running in supervisor mode.

- The processor will be in supervisor mode only, when executing OS routines.
- Before beginning to execute the application programs, it has to switch to user mode.
- While in the user mode, if an attempt is made to execute a privileged instruction, it leads to a special type of interrupt called a privileged exception.

A multiple-priority scheme can easily be implemented using separate interrupt-request and interrupt-acknowledge lines for each device as shown in the figure 4.5

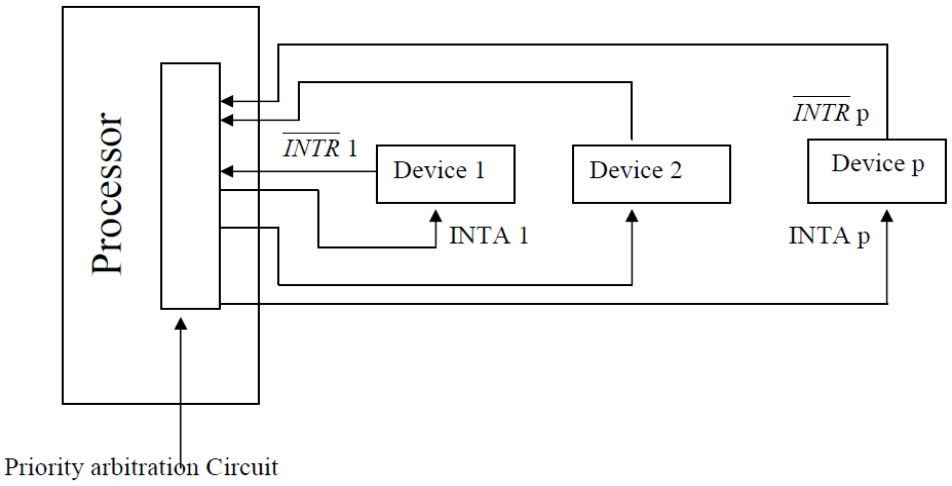


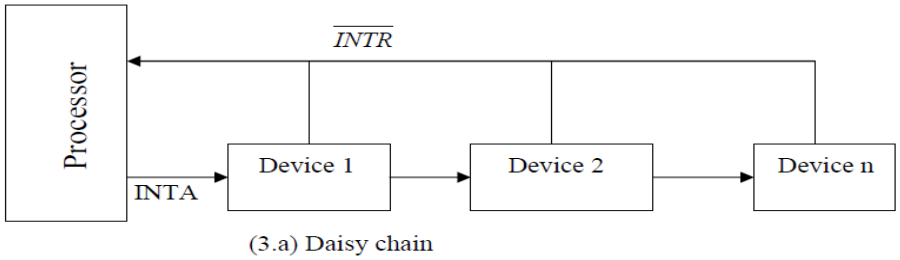
Figure 4.5 Implementation of Interrupt priority using individual interrupt-request and acknowledge lines

Simultaneous requests :

When interrupt requests arrive from two or more devices simultaneously, the processor has to decide which request should be serviced first and which one should be delayed.
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.

Daisy chain:

When vectored interrupts are used, we must ensure that only one device is selected to send its interrupt vector code. A widely used scheme for this is forming a daisy chain, as shown in the figure 4.6 below.



(3.a) Daisy chain

Figure 4.6 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.

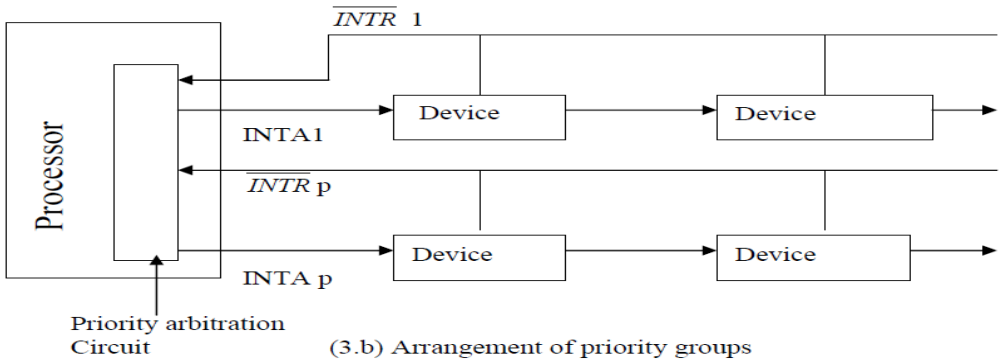
If devices1 has a pending request for interrupt blocks that INTA signal & proceeds to put its identification code on the data lines.

Therefore, the device that is electrically closest to the processor has the highest priority.

Arrangement of Priority Groups:

Daisy chain requires fewer wires than the individual connections.

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. This mechanism is used in many computers.



(3.b) Arrangement of priority groups

Figure 4.7 Arrangement of priority groups.

4.2.4 CONTROLLING DEVICE REQUESTS

Usually an I/O device interface generates an interrupt request whenever it is ready for an I/O transfer, whenever the SIN flag is set to 1. Interrupt requests are generated only by those I/O devices, which are being used by a given program. The devices which will be idle, are not allowed to generate interrupt requests, though they may be ready to participate in I/O transfer Operations. Hence, we need some mechanism in the interface circuits of individual devices to Control whether a device is allowed to generate an interrupt request or not.

The control which is needed is usually provided in the form of an **interrupt-enable** bit in the device's interface circuit. The keyboard **interrupt-enable →KEN**, and **display interrupt-enable →DEN**, flags in register **→CONTROL** perform this function. If either of these flags is set, the interface circuit generates an interrupt request whenever the corresponding status flag in register STATUS is set. At the same time, the interface circuit sets bit **KIRQ /DIRQ** to indicate that the keyboard / display unit is requesting an interrupt, respectively. If an interrupt-enable bit is reset to 0, the interface circuit will not generate an interrupt request, regardless of the state of the status flag.

4.2.5 EXCEPTIONS

An interrupt is an event which causes the execution of one program to be suspended and execution of another program to start.

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:

Computer has error-checking code in Main Memory , which allows detection of errors in the stored data. Whenever an error occurs, the control hardware detects the error, and informs the processor by raising an interrupt.

Sometimes the processor also interrupts the program. If processor detects an error or an unusual condition while executing, it suspends the program being executed and starts an exception- service routine. This routine takes an 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:

Whenever processor is in trace mode , after execution of every instruction an exception occurs 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 instruction in the program being debugged is executed. During the execution of the debugging program, the trace exception is disabled.

Break point:

Here the program being debugged is interrupted only at specific points selected by the user.
An instruction called the Trap (or) software- interrupt is used for this purpose.

While debugging the user may interrupt the program execution after instruction i.

When the program is executed and reaches that point, it is interrupted and debugging routine is activated.

Privilege Exception:

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

4.3 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 continuous intervention (interruption) 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 that requested the transfer in between, cannot be done 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.

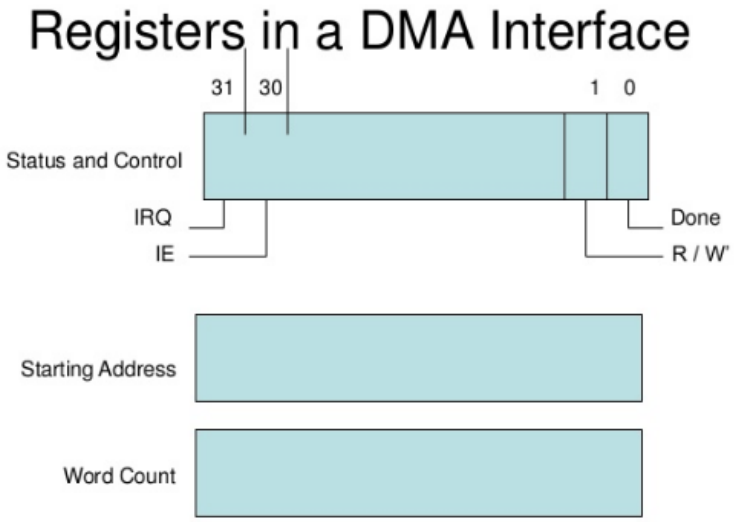


Figure 4.8 Registers in a DMA interface.

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

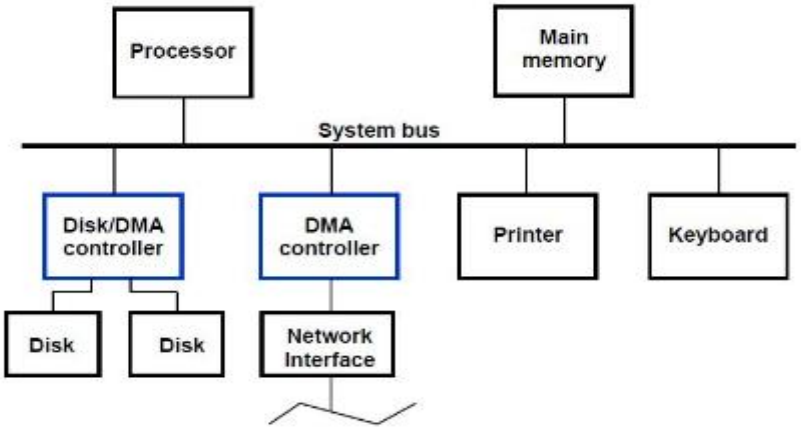


Figure 4.9 DMA controllers in a computer system

An example of a computer sytem showing how DMA controllers can be used is shown in figure 4.9. A DMA controller connects a high speed network to the computer bus. A disk controller, that controls two disks also has DMA capability and provides two DMA channels. It can perform two independent DMA operations at the same time.

To initiate a DMA transfer of block of data from main memory to one of the disks, a program loads the address and word count information into the registers of the corresponding channel of the disk controller.

After the DMA transfer is completed, this status is recorded in the status and control register of the DMA channel, when Done-bit is set. At the same time, if the IE bit is set, the DMA controller raises an interrupt request to the processor and sets IRQ bit.

Requests by DMA devices for using the bus are always given higher priority than processor requests. Among different DMA devices, highest priority is provided for high speed peripherals such as disks, high speed network interfaces or graphics display device. Among the most memory access cycles, provided by processor, the DMA controller always tries to access the system bus without giving chances to other devices. This interweaving technique of DMA is called " ***Cycle stealing".*** This kind of transfer of block of data without interruption to main memory is known as ***block or burst mode*** of DMA.

4.3.1. **BUS ARBITRATION**

During blocks of data transfer between I/O devices and processor, thus many devices compete to get access with the system bus. Hence there are chances for a conflict raised by all devices to get access with the system bus at the same time. The device that is allowed to access the system bus for the first time is called the BUS MASTER. BUS arbitration is the process by which the next device is allowed to become the bus master.

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:

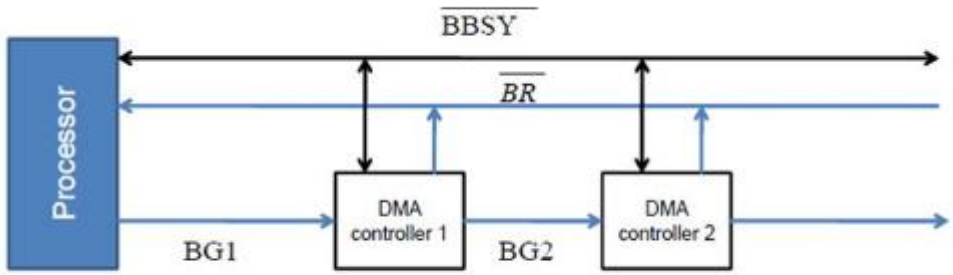


Figure 4.10 A simple arrangement for bus arbitration using daisy chain.

Here the processor is the bus master and it may grants bus master ship 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).

Distributed Arbitration:

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

4.4 BUSES

A bus protocol is the set of rules that govern the behaviour 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 information. (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.

4.4.1 SYNCHRONOUS BUS:-

In a synchronous bus, bus operations are synchronized with reference to a clock signal. The bus clock is generally derived from the computer system clock. A bus transaction often takes several clock cycles.

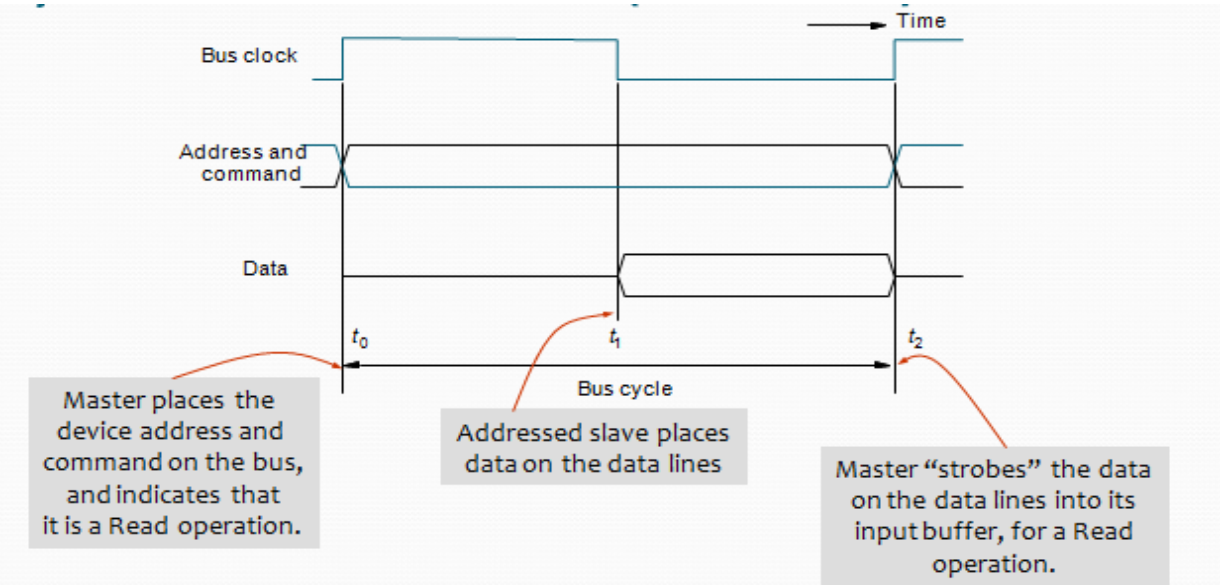


Figure 4.11 Timing of an synchronous bus

Note : Strobe means Capturing the values of data at a given instant and store them into a buffer.

- Once the master places the device address and command on the bus, it takes time for this information to propagate to the devices:
- Also, all the devices have to be given enough time to decode the address and control signals, so that the addressed slave can place data on the bus.
- At the end of the clock cycle, at time t_2, the master strobes the data on the data lines into its input buffer if it's a Read operation.
- When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.

In case of a Write operation, the master places the data on the bus along with the address and commands at time t_0.

- The slave strobes the data into its input buffer at time t_2.
-

Note: Synchronous buses are used in memory-processor buses

4.4.2 ASYNCHRONUS BUS

- Data transfers on the bus is controlled by a handshake between the master and the slave.
- Common clock in the synchronous bus case is replaced by two timing control lines:
  - Master-ready
  - Slave-ready.
- Master-ready signal is asserted by the master to indicate to the slave that it is ready to participate in a data transfer.
- Slave-ready signal is asserted by the slave in response to the master-ready from the master, and it indicates to the master that the slave is ready to participate in a data transfer.

Data transfer using the handshake protocol:

- Master places the address and command information on the bus.
- Asserts the Master-ready signal to indicate to the slaves that the address and command information has been placed on the bus.
- All devices on the bus decode the address.
- Address slave performs the required operation, and informs the processor it has done so by asserting the Slave-ready signal.
- Master removes all the signals from the bus, once Slave-ready is asserted.
- If the operation is a Read operation, Master also strobes the data into its input buffer.

Note : Asynchronous buses are used in I/O buses.

4.5 INTERFACE CIRCUITS

I/O interface consists of the circuits required to connect an I/O device to a computer bus. On one Side of the interface which connects to the computer have bus signals for: Address, **data and control.** On the other side of the interface data path is present, which controls data transfer between the interface and the I/O device. This side of the interface is called as "Port".

Ports can be classified into two:

Parallel port and Serial port.

4.5.1 PARALLEL PORT

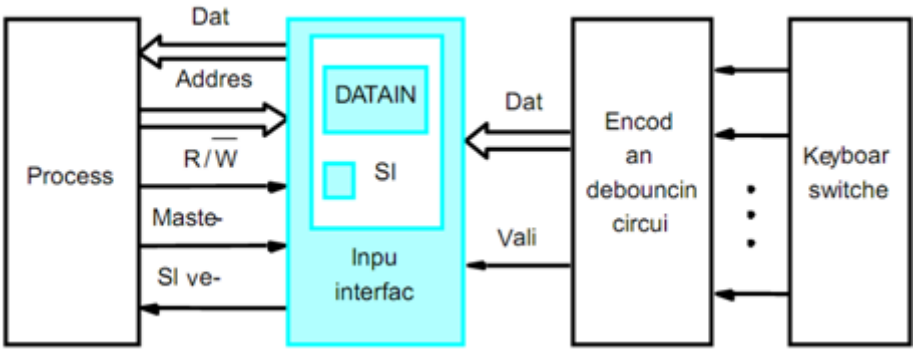


Figure 4.12 Keyboard to processor connection

Example 1: The above figure 4.12 shows the hardware components needed for connecting a keyboard to a processor. Keyboard consists of mechanical switches that are normally open. When a key is pressed, the switch is closed and establishes a path for the electrical signal. This signal is detected by an encoder circuit and it generates the ASCII code for the respective character. The debouncing circuit is used to detect the pressing of the key as single press, repeated pressing and some bouncing effect.

The mechanism of data transfer between the keyboard and processor is explained as below.

- The output of the encoder consists of the bits that represent the encoded character and one signal called **valid**, which 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.

Example 2: consider another example for parallel interface circuit, i.e., a printer connected to a processor.

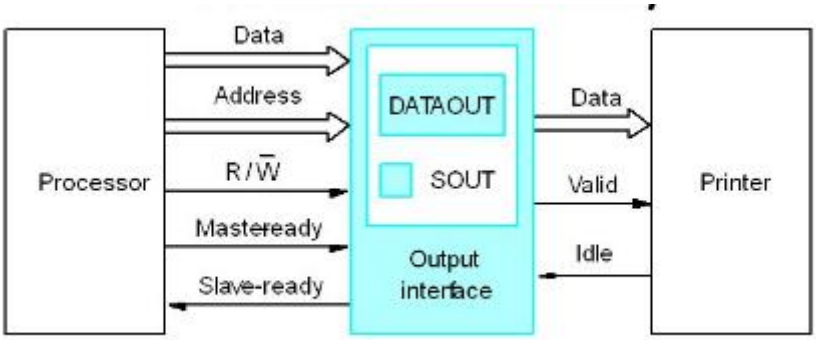


Figure 4.13 Printer to processor connection.

This printer functions under the control of handshake signals, Valid and idle. When the printer is ready to accept a character, the printer asserts its idle signal. The interface circuit then places a new character on the data lines and activates the valid signal. Thus the printer starts printing the new character and resets the idle signal and which in turn deactivate the valid signal.

The interface contain a data register DATAOUT and a status flag SOUT. SOUT is set to 1 when printer is ready to accept another character and is cleared to 0, when a new character is loaded into DATAOUT by the processor.

4.5.2 SERIAL PORT

Serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time. It consists of DATAIN and DATAOUT registers. The input shift register accepts bit –serial input from I/O device. When all 8 bits of data is received the contents of this shift register are loaded in parallel into the DATAIN register. Similarly Output data in DATAOUT register are loaded into output shift register in parallel. From the output shift register the date is sent in serial to the I/O device.

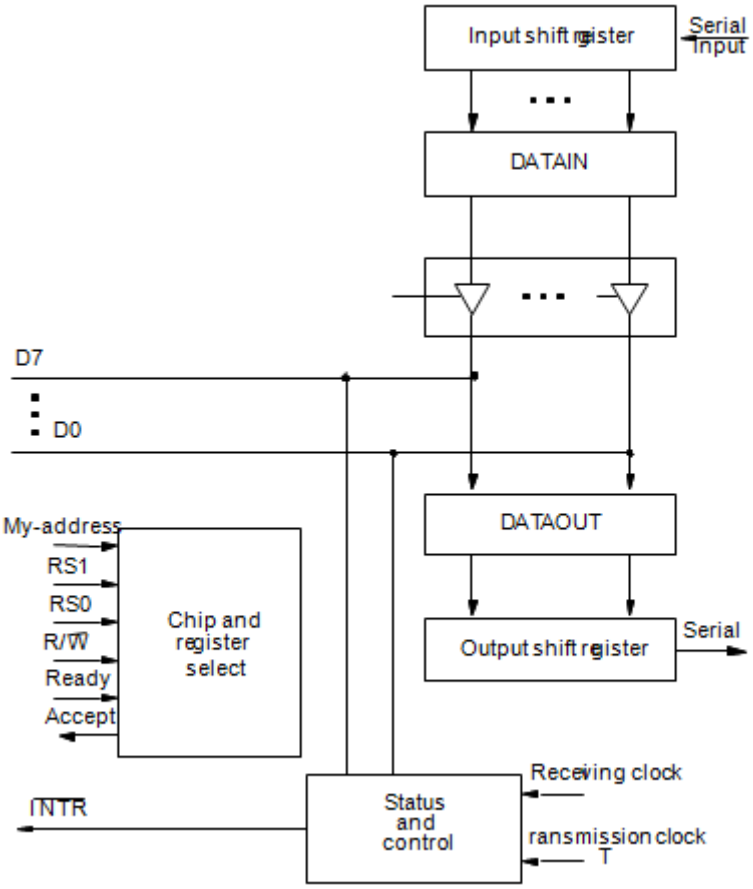


Figure 4.14 A serial interface.

Several standard serial interfaces have been developed:

- Universal Asynchronous Receiver Transmitter (UART) for low-speed serial devices.
- RS-232-C for connection to communication links.

4.6 STANDARD I/O INTERFACE

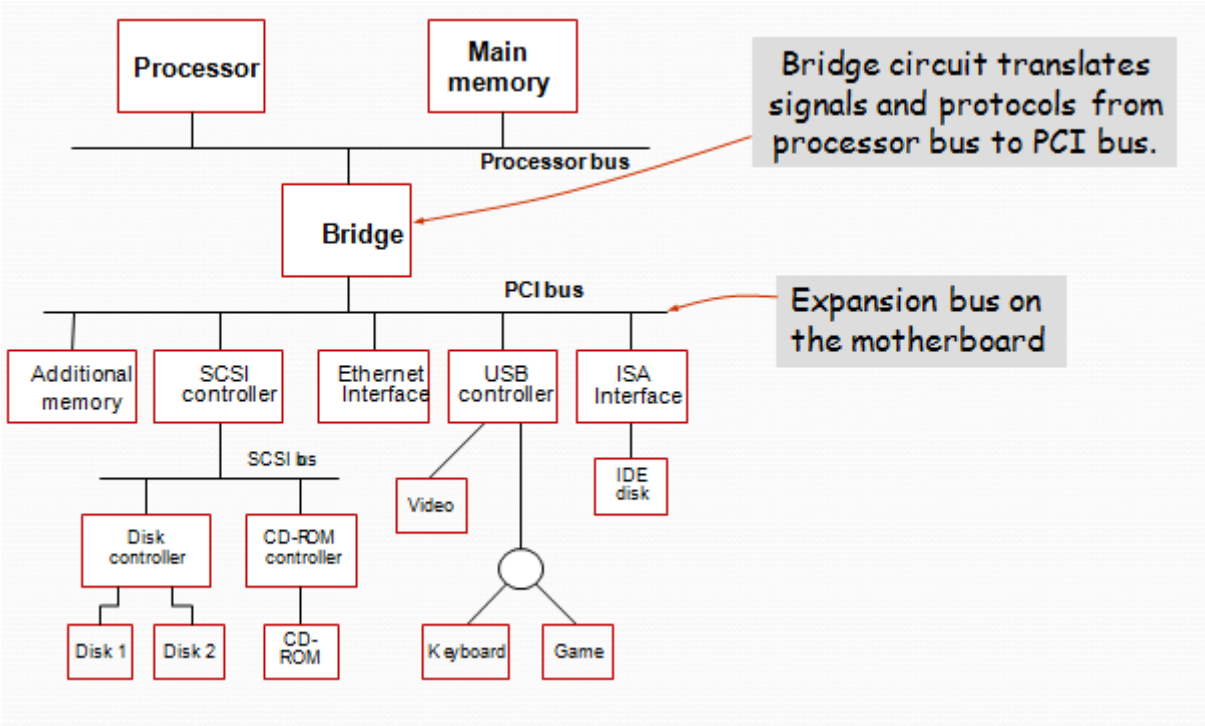


Figure 4.15 An example of a computer system using different interface standards.

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.

We have 3 Bus standards. These standards have been developed and approved by an organization such as IEEE (Institute of electronics and Electrical engineers) , ANSI (American National standards Institute) and ISO(International standards Organization). They are,

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

The way in which these standards are used in a computer system is illustrated in figure 4.15.

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.

4.6.1 PERIPHERAL COMPONENT INTER CONNECT (PCI) BUS

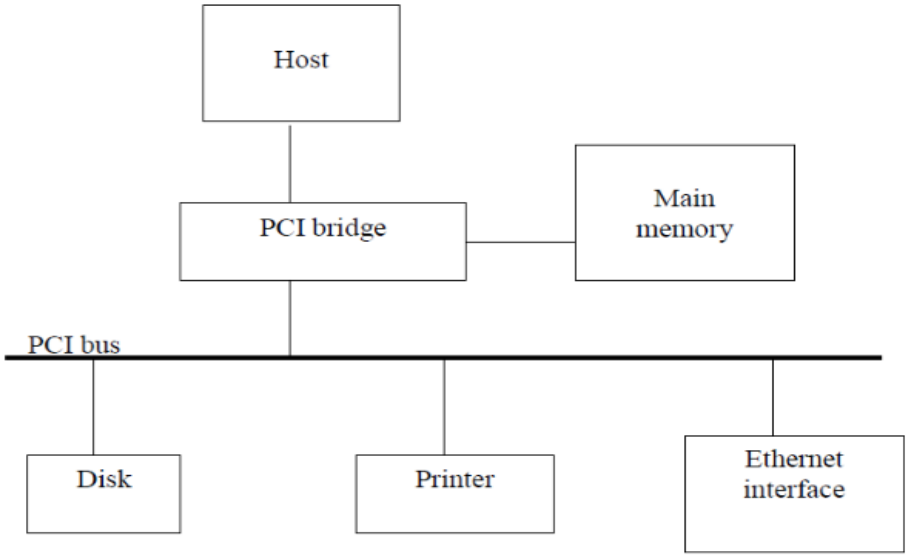


Figure 4.16 Use of PCI bus in a computer system

- PCI is a synchronous Bus. The devices connected to PCI bus, appear to the processor, as if they are connected directly to the processor bus.
- PCI is now installed on most new desktop computers
- PCI is developed as a low cost bus that is truly processor independent.
- PCI supports the feature of 'Burst Data Transfer'. With Burst Data Transfer, the data can be transferred at very high speeds.
- 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.
- PCI bus has much larger bandwidth than its previous buses (ISA, EISA and MCA). It can handle both 32 bit as well as 64 bit data.
- PCI transmits 32 bits at a time in a 124-pin connection (the extra pins are for power supply and grounding) and 64 bits in a 188-pin connection in an expanded implementation. The newer PCI Buses now supports maximum clock frequency of 66 MHz.

4.6.2 SCSI Bus:- (Small Computer System Interface)

SCSI refers to the standard bus which is defined by ANSI (American National Standard Institute). SCSI bus has 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 rate, the bus length should be 1.6m for SE signalling and 12m for LVD signalling.

The SCSI bus is connected to the processor bus through the SCSI controller.

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.

4.6.3 UNIVERSAL SERIAL BUS (USB)

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 system 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 limitations:

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.

Thus, 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 computer cover a wide range of functionality.

The speed, volume & timing constrains associated with data transfer to & from devices varies significantly.

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.

UNIT- 5: The Memory System

5.1 Some Basic Concepts

The Maximum size of the memory in any computer is determined by the number **address lines**, provided by processor used in the computer. For ex: if processor has 20 address lines, it is capable of addressing 2^{20} = 1M (mega) memory locations. Similarly, if processor has 32 address lines, it is capable of addressing 23^{2} = 4G (giga) memory locations.

Most modern computers are byte addressable. The two possible address assignments for a byte addressable 32-bit computer are Big-endian assignment and Little-endian assignment. 68000 processor uses Big-endian arrangement. Intel processors use Little-endian assignment. The ARM architecture can be configured to use any of these arrangements.

The memory is designed to store and retrieve data in word length quantities. For example, in a byte addressable computer, if an instruction generates 32-bit addresses, and when this is sent from processor to the memory unit, the high order 30 bits identify the word to be accessed and the low order 2 bits specify which byte location is involved. The Read operation other bytes may be fetched from the memory, but they are ignored by the processor. In a Write operation, the control circuitry must ensure that the other bytes of the same word are not changed.

The traditional architecture showing the connection of memory to the processor is as shown in the Figure 5.1. Data transfer between the memory and the processor takes place through the two processor registers MAR(memory address register) and MDR(memory data register). If MAR is k-bits long and MDR is n-bits long then the memory unit may contain up to 2^{k} addressable locations. The bus also includes control lines Read/ *Write (R/ W)* and MFC(memory function completed) for coordinating data transfers.

The processor reads the data from memory by loading the address of the required memory location into the MAR register and setting 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 the MFC signal. Upon receipt of the MFC, the processor loads the data on the data lines into the MDR register.

The processor writes the data into a memory location by loading the address of this location into MAR and loading the data into MDR. It indicates that a write operation is involved by setting the R/*W*line to 0.

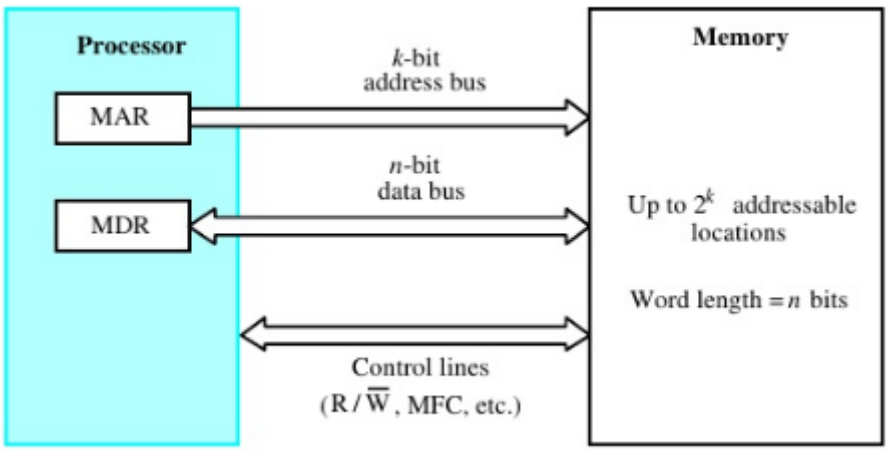


Figure 5.1. Connection of memory to the processor.

If read or write operations involve consecutive address locations in the main memory, then a "block transfer" operation can be performed by the starting address of the memory block and the number of locations to be accessed. A clock can be used for synchronizing the memory accesses or they may be controlled using special signals.

The speed of the memory units is measured using **memory access time**, which is the time elapsed between the initiation of an operation and its completion. i.e, the time between the Read and the MFC

signals. The minimum time delay between the initiation of two successive memory operations is called **memory cycle time**. For example, it is the time between the two successive Read operations.

The processor of a computer can usually process instructions and data faster than they can be fetched from the memory unit. Hence, the memory cycle time is the bottleneck in the system. One way to reduce the memory access time is to use a cache memory. Cache memory is a small, fast memory, that is placed in between the larger, slower main memory and the faster processor. Cache holds currently active segments of a program and their data.

In a situation when the Memory (RAM) which is required by the user is high from the available memory, we use the concept of virtual memory. Virtual memory is a method of using the computer hard drive to provide extra memory for the computer. In the virtual memory the physical memory (Hard Disk) will be treated as the logical memory. It means that, with the help of virtual memory, we can also increase the size of logical memory. With the help of virtual memory, all the space of hard disk can be used as the logical memory so that a user can execute any number of programs. In such a case, an address generated by the processor is referred to as logical address or virtual address. The virtual address is mapped on to the physical memory, where data are actually stored. The mapping function is implemented as the memory management unit. The mapping function can be changed during the program execution according to the requirements.

5.2 Semiconductor RAM Memories

Semiconductor memories are available in a wide of speeds. The cycle times of semiconductor memories range from 100ns to less than 10ns. When the semiconductor memories were introduced, they were expensive than magnetic-core memories. Because of rapid advances in VLSI technology, the cost of semiconductor memories dropped.

Hence, semiconductor memories are now used almost exclusively in implementing memories.

5.2.1 Internal Organization of memory chips

Random access memory (RAM) is the best known form of computer memory. RAM is considered "random access" because you can access any memory cell directly if you know the row and column that intersect at that cell. RAM data, on the other hand, can be accessed in any order.

RAM memory consists of memory cells. Each memory cell represents a single bit of data (logic 1 or logic 0). Memory cells are etched onto a silicon wafer in an array of columns (bit lines) and rows (word lines). The intersection of a bit line and word line constitutes the address of the memory cell. The cells in each column are connected to a Sense/Write circuit by two bit lines. The Sense/Write circuits are connected to the data input/output lines of the chip.

During a Read operation, these circuits Sense/Read the information stored in the cells selected by a word line and transmit this information to the output data lines. During Write operation, the Sense/Write circuits receive input information and store it in the cells of the selected word.

Figure 5.2 is an example of a very small memory chip with 16 words of 8 bits each. It is called as 16 X 8 organization. The data input and output of each Sense/Write circuit are connected to single bi-directional data line, that can be connected to the data bus of a computer. Two control lines R/ $\overline{W}$ and CS, are provided in addition to address and data lines. The R/ $\overline{W}$ input specifies a required operation, and the CS input is used to select the given chip in a multi-chip memory system.

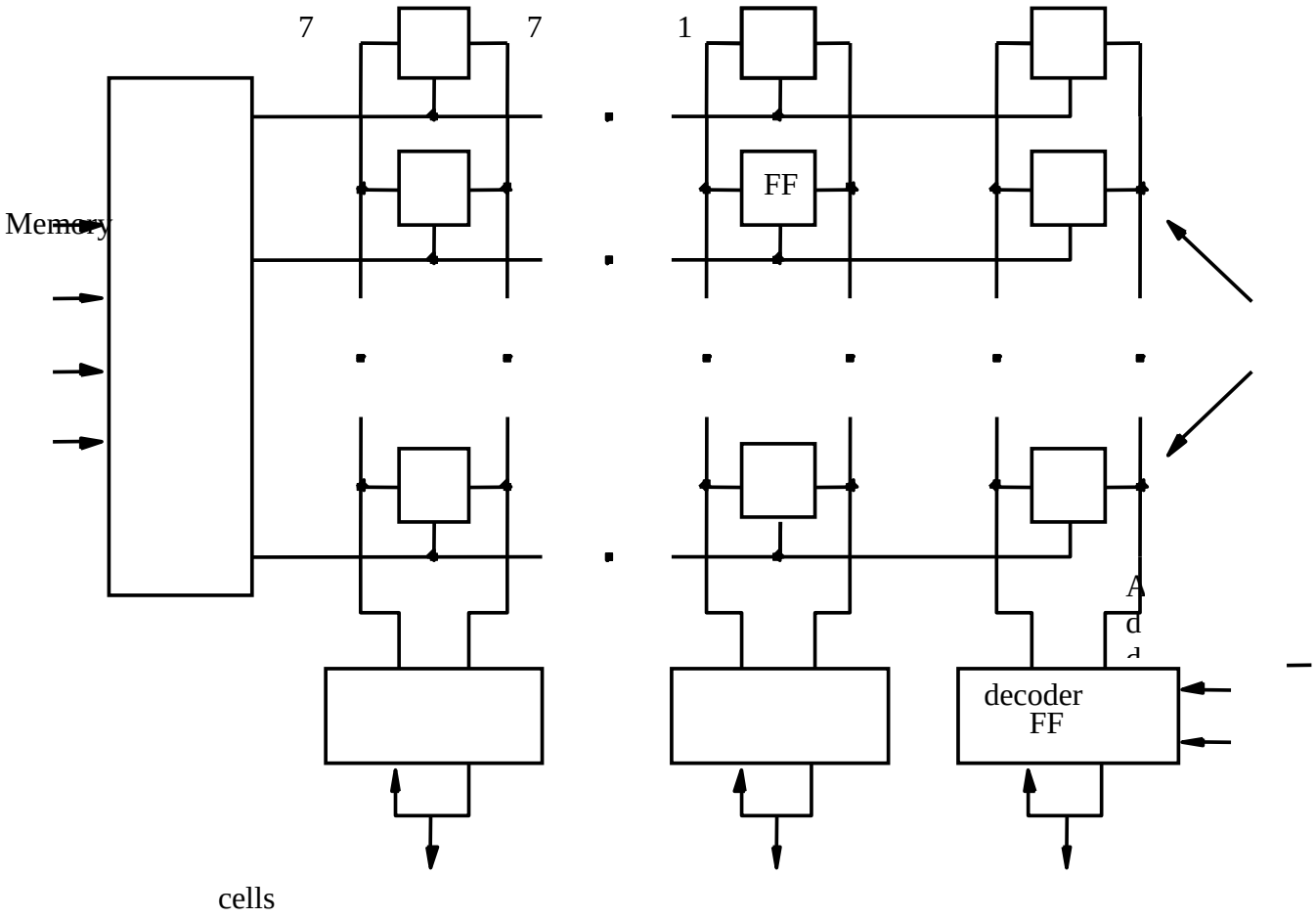


Figure 5.2 Organization of bit cells in a memory chip

The circuit in the Figure 5.2 stores 128 bits and needs 16 external connections for address (4 connections), data (8 connections), control lines (2 connections) and 2 lines for power supply and ground connections are also required.

If the memory circuit contains 1K (1024) memory cells, then the circuit can be organized as 128 X 8 memory, requiring 19 external connections (7 lines for address, 8 lines for data, 2 lines for control signals, and 2 lines for power and ground connections) in total. The same number of cells can be organized into 1K X 1 format. In this case, a 10-bit address is needed. As a result, it requires only 15 external connections (10 lines for address, 1 line for data, 2 lines for control signals and 2 lines for power and ground connections). The Figure 5.3 shows such an organization.

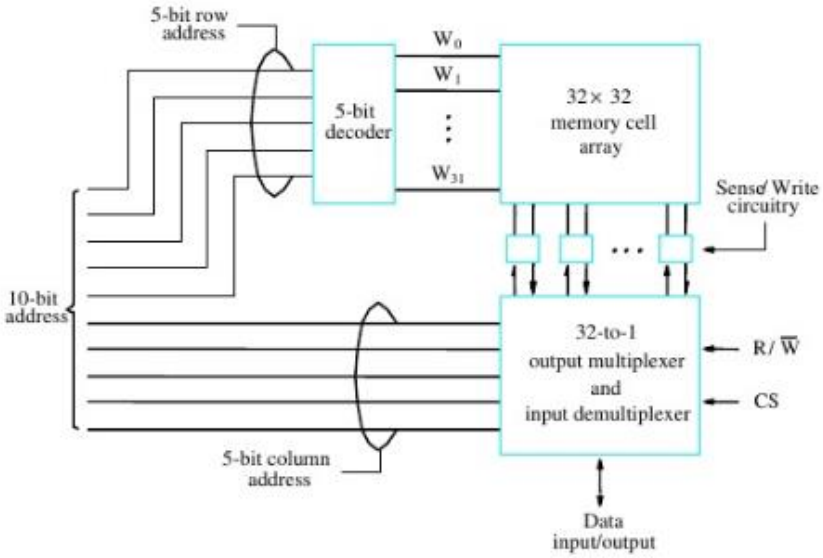


Figure 5.3 Organization of 1K X 1 memory chip.

The required 10-bit address is divided into 2 groups of 5 bits each to form the row and column addresses for the cell array. A row address selects a row of 32 cells, which can be accessed parallely. But in case of column address, only one of these cells is connected to the external data by output multiplexer and input demultiplexer. Commercially available memory chips contain much larger number of memory cells than shown in Figures 5.2 and 5.3. For example, a 4M bit chip may have 512 X 8 organization. This organization comprises of 19 address and 8 data input/output pins.

5.2.2 Static Memories

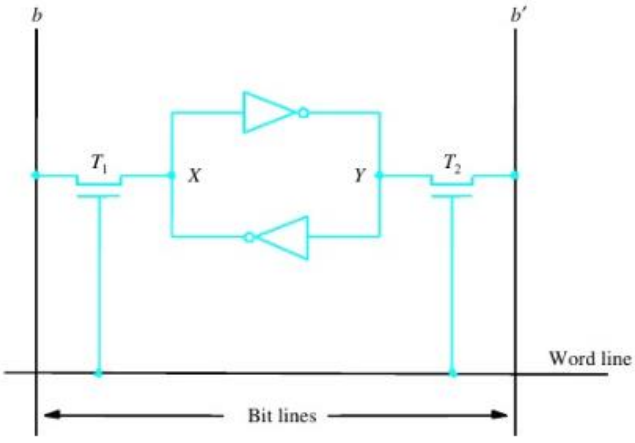


Figure 5.4 A static RAM cell

Static memories are the memories, that consists of circuits, which can retain their state as long as power is applied. Figure 5.4 shows how a static RAM (SRAM) cell may be implemented.

Two transistor inverters are cross connected to implement a basic flip-flop. The cell is connected to one word line and two bits lines by transistors T1 and T2. When word line is at ground level, the transistors are turned off and the latch retains its state.

Read operation: In order to read state of SRAM cell, the word line is activated to close switches T1 and T2. Sense/Write circuits at the bottom monitor the state of b and b' and set the output accordingly.

Write operation: The data to be written into the cell is placed on bit line b and its complement on b', and the word line is activated. The required signals on the bit line are generated by Sense/Write circuits.

CMOS cell: A CMOS implementation of a SRAM is as shown in Figure 5.5.

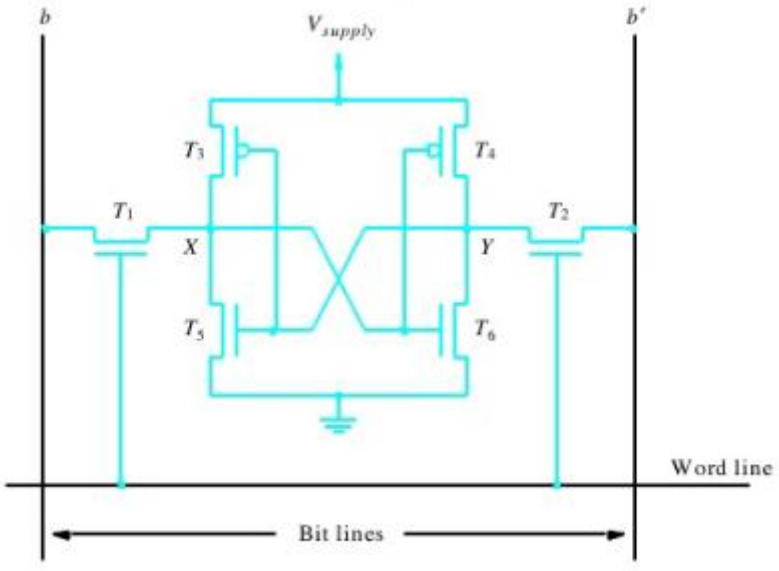


Figure 5.5 An example of a CMOS memory cell.

Transistor pairs (T3, T5) and (T4, T6) form the inverters in the latch. The read and write operations are performed as explained above. The

power supply voltage V_{supply} is 5volts in older CMOS SRAMs or 3.3volts in new low voltage versions.
Continuous power is needed for the cell to retain its state. If power is interrupted, the contents of the cell is lost. When the power is restored, the latch will settle down to a stable state which need not be same as before. Hence, SRAMs are said to be volatile memories, as they lose the contents when the power is interrupted.

The major advantages of CMOS SRAMs are –

1. Very low power consumption.
2. Can be accessed very quickly in few nanoseconds.
3. Useful in applications where speed is of critical concern.

5.2.3 Asynchronous DRAMs

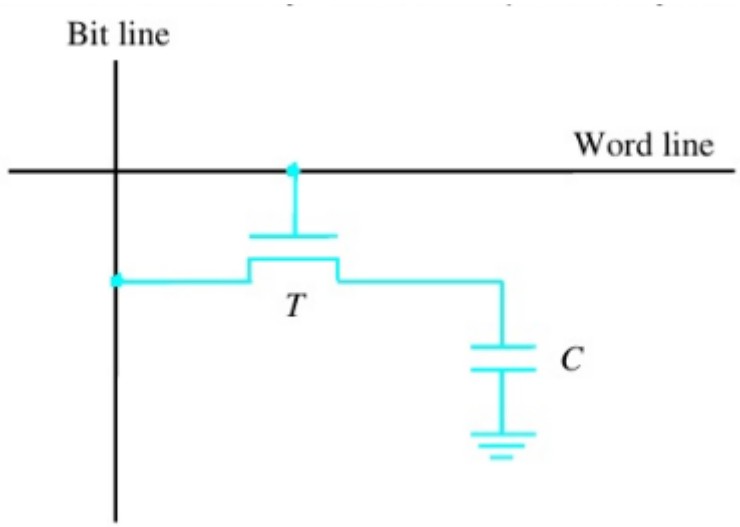


Figure 5.6 A single-transistor dynamic memory cell.

Static RAMs are fast, but costly. Hence, less expensive RAMs can be implemented if simpler cells are used. But, they do not retain their state indefinitely. Hence, they are called **dynamic RAMs (DRAMs)**.

Information is stored in DRAMs in the form of a charge on a capacitor and this charge can be maintained only for 10s of milliseconds. To store the information for a much longer time, the

contents must be periodically refreshed. The contents may be refreshed while accessing them for reading.

Figure 5.6 is an example of a DRAM cell, consisting of a capacitor C, and a transistor T. To store the information in this cell, the transistor is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor. After the transistor is turned off, the capacitor begins to discharge. Hence, the information stored in the cell can be read correctly only before the charge on the capacitor drops below the threshold value.

During the read operation, the transistor in a selected cell is turned on. The sense amplifier connected to the bit line recharges the capacitor to the full charge to represent the logic 1, if the charge stored on the capacitor is above the threshold value. If the sense amplifier detects that the charge on the capacitor is below the threshold value, it pulls the bit line to ground level to represent a logic zero. Thus, the contents may be refreshed while accessing the cells for reading.

A 16-megabit DRAM chip configured as 2M X 8 is as shown in Figure 5.7

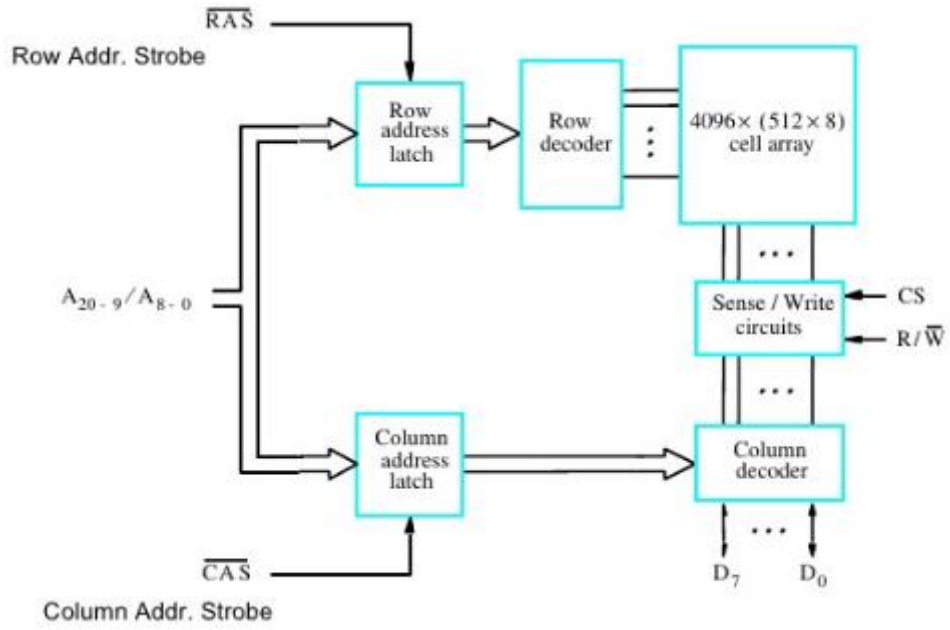


Figure 5.7 Internal organization of a 2M X 8 dynamic memory chip

The 4096 cells in each row are divided into 512 groups of 8 bits. Hence, a row can store 512 bytes. Therefore, 12 address bits are used to select a row, and 9 address bits are used to select a group in a row. As a result, a total of 21 address bits are needed to access a byte. The higher order 12 bits form the row address and the lower order 9 bits form the column address. To reduce the number of pins needed for external connections, the row and column addresses are multiplexed on 12 pins.

During read and write operation, first apply the row address, the RAS signal latches the row address into row address latch. Then apply the column address, CAS signal latches the column address into column address latch. Then, if a read operation is initiated, the information in the column address latch is decoded and the appropriate group of 8 Sense/Write circuits are selected. The output values of the selected circuits are transferred to the data lines, D_{7-0}. For a write operation, the information on D_{7-0} lines are transferred to the selected circuits, which is used to overwrite the contents of the selected cells in the corresponding 8 columns.

To maintain the contents of DRAM, each row of cells must be accessed periodically. Many dynamic memory chips incorporate a **refresh circuit** within the chip themselves.

In the DRAM described above, the timing of the memory unit is controlled by a specialized unit which generates RAS and CAS. Hence, this is called **asynchronous DRAM.**

DRAMs are low cost, and high density memories. Hence, they are widely used in computers. They range in size from 1M to 256 M bits and even larger chips are being developed.

Fast page mode: Suppose if we want to access the consecutive bytes in the selected row, it can be done without having to reselect the row. Add a latch at the output of the sense circuits in each row. All the

latches are loaded when the row is selected. Different column addresses can be applied to select and place different bytes on the data lines.

Consecutive sequence of column addresses can be applied under the control signal CAS, without reselecting the row. This allows a block of data to be transferred at a much faster rate than random accesses. A small collection/group of bytes is usually referred to as a block. This transfer capability is referred to as the fast page mode feature.

5.2.4 Synchronous DRAMs

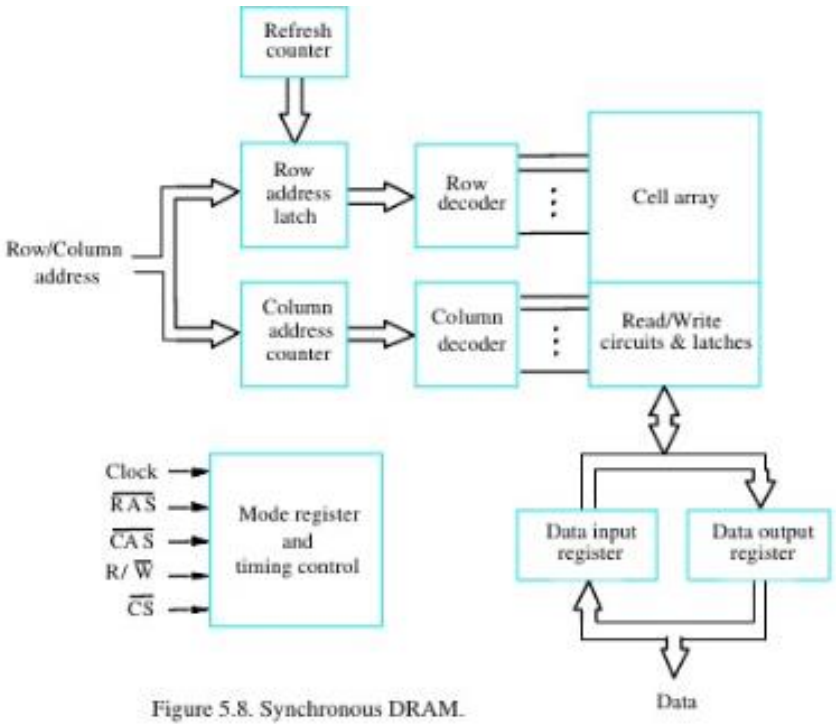


Figure 5.8. Synchronous DRAM.

DRAMs whose operation is directly synchronized with processor clock signal are called as synchronous DRAMs. Figure 5.8 shows the structure of a SDRAM. The address and data connections are buffered using registers. The output of each sense amplifier is connected to a latch.

During a Read operation, the contents of the cells in a row are loaded onto the latches. During a refresh operation, the contents of the cells are refreshed without changing the contents of the latches. Data held

in the latches correspond to the selected columns, are transferred to the output. For a burst mode of operation, successive columns are selected using column address counter and clock. CAS signal need not be generated externally. A new data is placed on the data lines in each clock cycle. All actions are triggered by the raising edge of the clock. Figure 5.9 shows the timing diagram for a typical burst read of length 4.

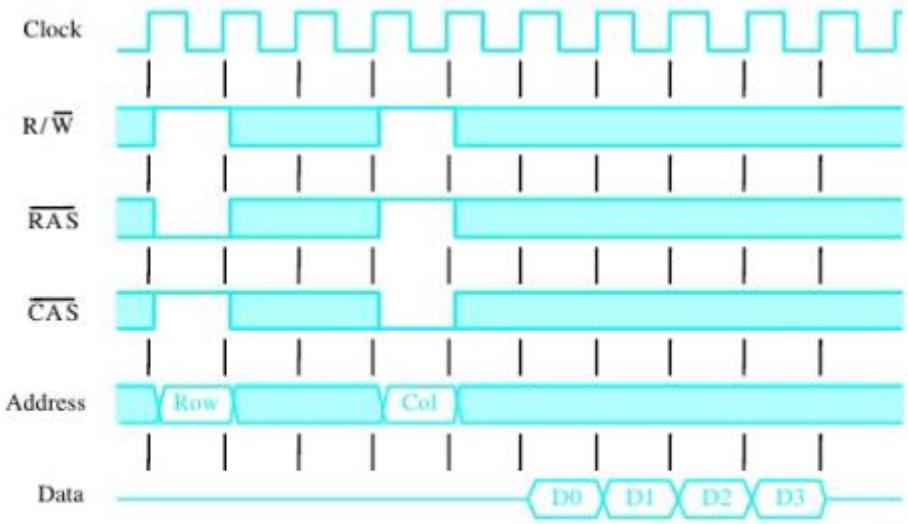


Figure 5.9. Burst read of length 4 in an SDRAM.

SDRAMs have built-in refresh circuitry. In a typical SDRAM, each row must be refreshed at least every 64 milliseconds. Commercial SDRAMs can be used with clock speeds above 100 MHz. For example, Intel's PC100 AND PC133 bus specification has a system bus controlled by 100 or 133MHz clocks respectively. Hence, manufacturers of memory chips produce 100 to 133MHz SDRAM chips.

Latency and bandwidth :

Memory latency is the time it takes to transfer a word of data to or from memory. Memory bandwidth is the number of bits or bytes that can be transferred in one second. The bandwidth of a memory unit depends on the speed of access to the stored data and on the number of bits that can be accessed in parallel. Bandwidth also depends on the transfer capability of the links that connect the memory and the processor. Thus, the bandwidth is the product of the rate at which the

data is transferred and the width of the data bus. Memory chips are designed to meet the speed requirements of popular buses.

Double-Data-Rate SDRAM:
The SDRAM does all the actions on the rising edge of the clock signal. There is a similar device, that accesses the cell array in the same way, but the data transfers take place on both edges of the clock. Since the data is transferred on both the edges of the clock, bandwidth seems to be doubled for long burst transfers. Such devices are called **double-data-rate SDRAMs (DDR SDRAMs).**

For faster access of the data, the cell array is organized into two banks, each of which can be accessed separately. Consecutive words of a block are stored on different banks, so that the two words, which are transferred on successive edges of the clock, can be simultaneously accessed.

5.2.5 Structure of larger memories

Memory chips may be connected in order to form a larger memory. Figure 5.10 illustrates one such connection to develop a memory of size 2M words of 32 bits each.

Static memory systems:
Figure 5.10 uses 512 x 8 static memory chips to form a memory consisting of 2M words of 32 bits each. Each column consists of 4 chips. Each chip implements one byte position.

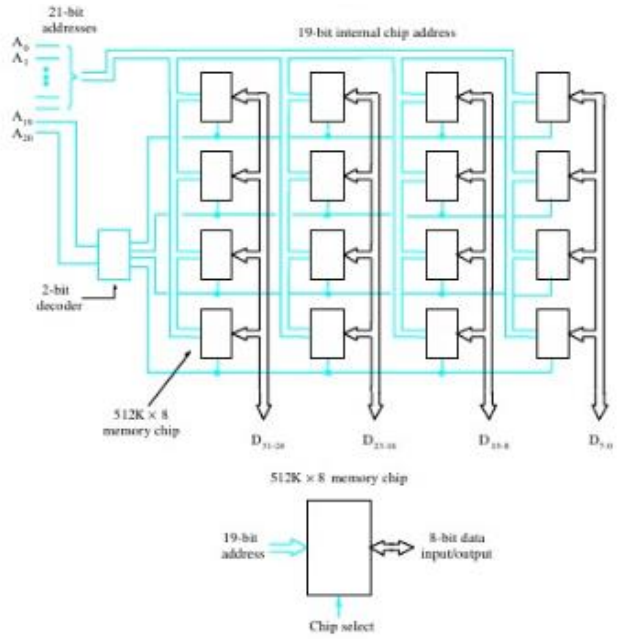


Figure 5.10 Organization of a 2M X 32 memory module using 512K X 8 static memory chips.

A chip is selected by setting its chip select control line to 1. Selected chip places its data on the data output line, outputs of other chips are in high impedance state. 21 bit address is needed to access a 32-bit word. High order 2 bits are needed to select the row, by activating the four Chip Select signals. 19 bits are used to access specific byte locations inside the selected chip.The R/ $\overline{W}$ inputs of all chips are tied together to provide a common control.

Dynamic memory systems:

Large dynamic memory systems can be implemented using DRAM chips in a similar way to static memory systems. Placing large memory systems directly on the motherboard will occupy a large amount of space. Also, this arrangement is inflexible since the memory system cannot be expanded easily.

Packaging considerations have led to the development of larger memory units known as SIMMs (Single In-line Memory Modules) and DIMMs (Dual In-line Memory Modules). Memory modules are an assembly of memory chips on a small board that plugs vertically

onto a single socket on the motherboard. SIMMs and DIMMs of different sizes are designed to use the same size socket. For example, 4M X 32, 32M X 32 bit DIMMs, both use the 100 pin socket. Similarly, 8M X 64, 16M X 64, 32M X 64 and 64M X 72 DIMMs use 168 pin socket. These modules occupy less space on the motherboard and allow for easy expansion by replacement.

5.2.6 Memory system consideration

For a given application, choosing a RAM chip depends on the factors such as cost, speed, power dissipation, and size of the chip. When speed becomes the primary requirement, static RAMs are used. The cost of static RAMs depends on the complexity of the circuit that realizes the basic cell. They are used mostly in cache memories. Dynamic RAMs are used for implementing computer main memories. The high densities achievable in dynamic RAMs make large memories economically feasible.

Memory controller:

In order to reduce the number of pins in a dynamic memory chip, multiplexed addresses are used. Address is divided into two parts- high-order address bits, which select a row in the array, and low-order address bits select a column in the row.

High-order bits are provided first, and latched using RAS signal. Low-order bits are provided later, and latched using CAS signal. However, a processor issues all address bits at the same time. In order to achieve the multiplexing, memory controller circuit is inserted between the processor and memory. Figure 5.11 shows the use of memory controller.

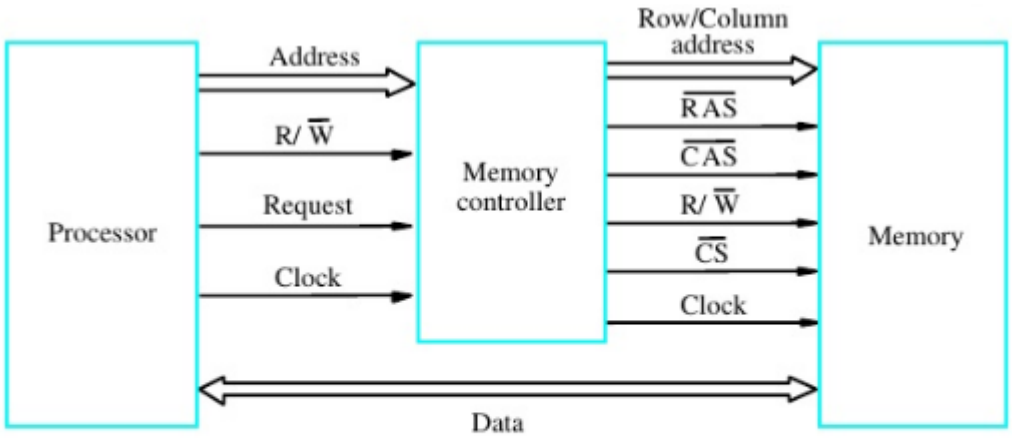


Figure 5.11. Use of a memory controller.

The memory controller accepts a complete address, request signal, and R/ $\overline{W}$ signal from the processor and forwards the row and column portions of the address to the memory and generates $\overline{RAS}$ and CAS signals. The controller provides RAS-CAS timing, R/ $\overline{W}$, and CS signal to the memory. CS, RAS and CAS are all active low signals. Data lines are connected directly between the processor and the memory. Clock signal is needed in SDRAM chips.
When used with DRAM chips without self refreshing capabilities, the memory controller has to provide the information (refresh counter that provides successive row addresses) needed to control the refreshing process. This causes the refreshing of all rows to be done within the period specified for a particular device.

Refresh overhead:

All dynamic memories need refreshing. In older DRAMs, 16ms was needed to refresh all rows. In typical SDRAMs, a typical period is 64ms.

As an example, let us consider an SDRAM whose cells are arranged in 8K (8192) rows. If it takes 4 clock cycles to access each row, then it takes 8192 X 4 = 32,768 cycles to refresh all rows. With a clock rate of 133MHz, the time needed for refreshing all rows is 32,768/

$(133 \text{ X } 10^6) = 246 \text{ X } 10^{-6}$ s. i.e, the refreshing process occupies 0.246ms in each 64ms time interval. Hence, the refresh overhead is 0.246/ 64 = 0.0038 , which is less than 0.4% of the total time for accessing the memory.

5.2.7 Rambus memory

A very wide bus is expensive and requires a lot of space on the mother board. A narrow and much faster bus can be implemented as an alternative approach. Rambus Inc. developed one such proprietary design known as **Rambus.** Fast signalling method used to transfer information between chips is the key feature of rambus technology. Instead of using voltage levels like 0 or V_{supply} to represent logic values, the signals with much smaller voltage swings around a refrence voltage, V_{ref} (2 volts) is used. The logic values represented by 0.3V swings above and below V_{ref}. This type of signalling is known as differential signalling. Small voltage swings needs short transition time, which contributes to the high speed of the transmission.

Differential signalling and high transmission rates requires special circuit interfaces. Rambus provides a complete specification for the design of such communication links, called the **rambus channel.** Current rambus designs allow for clock frequency of 400MHz. Data are transmitted on both edges of clock and hence, the data transfer rate is 800MHz. Rambus memory chips use cell arrays based on the standard DRAM technology. Multiple banks of cell arrays are used to access more than one word at a time. Circuitry needed to interface to the rambus is included in the chip. Such chips are called **Rambus DRAMs (RDRAMs).**

Rambus provides 8 data lines and a 9^{th} data line is used for parity checking. A two channel rambus known as Direct RDRAM, has 18 data lines to transfer 2 bytes of data at a time. There are no separate address lines.

Communication between processor or some other device (master) and RDRAM modules (slaves) is carried out by means of packets transmitted on data lines. The 3 types of packets are request, acknowledge and data. A request packet issued by a master indicates the operation to be performed. A request packet contains the address of desired memory location and the number of bytes involved in the transfer. The operations may be memory read/write and reading and writing of various control registers in the RDRAM chip. When the master issues a request packet, the slave responds with a positive acknowledgement, if it can satisfy the request immediately. Otherwise, the slave responds with the busy acknowledgement by returning a negative acknowledgement packet in which case the master will try again. The number of bits in the request packet is more than the number of data lines, which needs several clock cycles to transmit the entire packet. A very high transmission rate compensates for the narrow communication link.

RDRAM chips can be assembled into larger modules similar to SIMMs and DIMMs. RIMM is an example for one such module which can hold upto 16 RDRAMs.

5.3 Read-Only Memories

SRAM and SDRAM chips are volatile, as they lose the contents when the power is turned off. Many applications need memory devices to retain contents after the power is turned off. For example, When computer is turned on, the operating system must be loaded from the disk into the memory. It is required to store instructions which would load the OS from the disk, in a memory, that will not lose its contents after the power is turned off. Hence, we need to store the instructions into a non-volatile memory. Non-volatile memory is read in the same manner as volatile memory. Separate writing process is needed to place information in this memory. Normal operation involves only reading of data, this type of memory is called Read-Only memory (ROM).

5.3.1 ROM

Figure 5.12 shows a possible configuration for a ROM cell. When the transistor is connected to ground at point P, a logic value 0 is stored in the cell, else logic 1 is stored. The bit line is connected to the power supply through a register. To read the contents of the cell, the word line is activated. A sense circuit at the end of the bit line is used to generate the proper output value. Data are written into a ROM when it is manufactured.

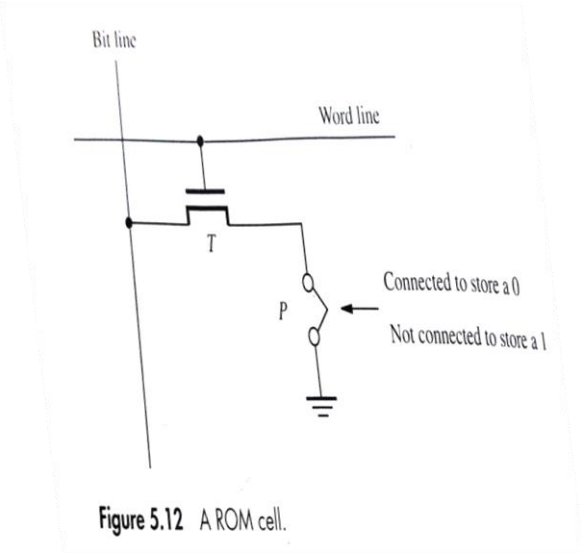


Figure 5.12 A ROM cell.

5.3.2 PROM

Some ROMs allow the data to be loaded by a user. Such ROMs are called Programmable ROMs (PROMs). PROM is made programmable by inserting a fuse at point P in Figure 5.12. PROM contains all 0s before it is programmed. The user can insert 1 at the required location by burning the fuses at that locations using high current pulses. The process of inserting the data is irreversible. Storing information specific to a user in a ROM is expensive. PROMs provide a faster and considerably less expensive approach as they can be programmed directly by the user.

5.3.3 EPROM

Another type of ROM chip allows the stored data to be erased and new data to be loaded, which are called as Erasable Programmable ROM (EPROM). It provides considerable flexibility during development phase of digital systems. EPROM cell has a structure similar to ROM cell as shown in Figure 5.12. However, the connection to ground is made at P, using a special transistor which can be turned off. This transistor can be programmed to behave like a permanently open switch, by injecting charge into it, that becomes trapped inside. Advantages of EPROM chips is that their contents can be erased by dissipating the charges trapped in the transistor and reprogrammed. For this reason, EPROM chips are mounted in packages that have transparent windows through which the chips can be exposed to UV light for erasing the data.

5.3.4 EEPROM

To erase the contents of EPROMs, they have to be exposed to ultraviolet light by physically removing from the circuit which erases the entire contents. In Electrically erasable PROMs (EEPROMs), the contents can be stored and erased electrically. They do not have to be removed for erasure. It is possible to erase cell contents selectively. But, different voltages are needed for reading, writing and erasing purpose.

5.3.5 Flash Memory

This approach is similar to EEPROM. A flash cell is based on a single transistor controlled by trapped charge. In EEPROM, the contents of a single cell were possibly read and written, whereas in a flash device, the contents of a single cell are possibly read, but can only write the contents of an entire block of cells. Flash devices have greater density, leading to higher capacity and low cost per bit. Power consumption of flash memory is very low, making it attractive for use in equipment that is battery-driven. For example, hand held computers, cell phones, digital cameras and MP3 music players. In

hand held computers, the software that operates the equipment is stored in the flash memory. Flash memory holds the picture data in case of digital cameras. In MP3 players, the sound data occupies the flash memory.

Single flash chips are not sufficiently large, so flash cards and flash drives are used to implement larger memory modules.

Flash cards

A larger module can be constructed by mounting flash chips on a small card. Such flash cards possess standard interface which allows it to be used in a variety of products. A card is plugged into a easily accessible slot. The typical sizes of flash cards are 8, 32, and 64 Mbytes.

Flash Drives

Flash drives are designed to fully emulate the hard disks. The flash drives have lower storage capacity. Right now, the flash drives have the capacity less than one gigabyte.

Advantages:

1. Flash drives are solid state electronic devices, which do not have movable parts. This leads to faster response.
2. Flash drives consume less power, making it attractive for use in the applications that are battery-driven.
3. Flash drives are insensitive to vibration

Disadvantages

1. Flash drives have smaller capacity and higher cost per bit.
2. The flash memory deteriorate after it has been written a number of times, say one million times.

5.4 Speed, Size and Cost

A big challenge in the design of a computer system is to provide a sufficiently large memory, with a reasonable speed at an affordable cost.

Static RAM:

SRAM chips can be used to implement very fast memory. But, SRAM chips are expensive, because a basic SRAM cells have6 transistors, making it impossible to pack a large number of cells onto a single chip. Hence, it is practically impossible to construct larger memories using SRAMs.

Dynamic RAM:

DRAMs have simpler basic cell circuit, hence are much less expensive, but significantly slower than SRAMs. Dynamic memory units in the range of hundreds of megabytes can be implemented at a reasonable cost, but the size is still small compared to the demand. Hence, secondary storage such as magnetic disks can be used to implement large memory spaces. Very large magnetic disks are available at reasonable cost. However, they are much slower than the semiconductor memory units. Hence, large amount of cost-effective storage can be achieved through magnetic disks and a large main memory can be constructed with DRAMs. SRAMs can thus be used in smaller units such as cache memories, where speed is of prime importance.

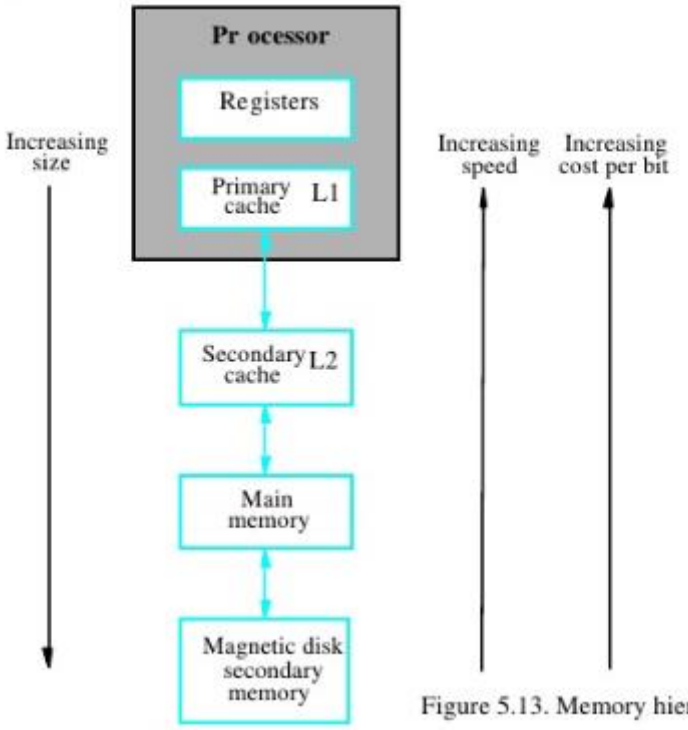


Figure 5.13. Memory hierarchy.

Figure 5.13 shows the hierarchy of the entire computer memory. The data stored in processor registers are fast accessible. Hence, processor registers are at the top of the hierarchy. Processor cache (primary cache) occupies the next level, as it holds the copies of instructions and data stored in the larger memory, that is provided externally. This cache is also called as level 1 (L1) cache. It is small as it competes for the space on the processor chip. A larger, secondary cache is placed in between the primary cache and the rest of the memory. It is referred as level 2 (L2) cache and are implemented using SRAM chips. It is possible not to have a cache on the processor chip. It is also possible to have both caches within the processor chip.

The next level in the hierarchy is the main memory, which are implemented using DRAMs, in the form of SIMMs, DIMMs, or RIMMs. The main memory is much larger, but slower than the cache memory. The access times of the main memory is 10 times more than the L1 cache. Since magnetic disks provide inexpensive storage, they can be used as the secondary storage as shown in the Figure 5.13.

5.5 Cache Memories

Processor is much faster than the main memory. As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory. This is a major obstacle towards achieving good performance.
Speed of the main memory cannot be increased beyond a certain point, cache memory can be used, which is an architectural arrangement which makes the main memory appear faster to the processor than it really is.

Cache memory mechanism is based on the property of computer programs known as "locality of reference". Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently. These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly. This is called **"locality of reference".**

Locality of reference can be described as temporal or spatial locality of reference. Temporal locality of reference means that the recently executed instruction is likely to be executed again very soon. Spatial locality of reference means that the instructions with addresses close to a recently executed instruction are more likely to be executed soon.

Placing the active segments of a program in the fast cache memory will significantly reduce the total execution time of that program. Consider the arrangement in the Figure 5.14. When a processor issues a Read request, a block of memory words containing the location specified are transferred from the main memory to the cache, one word at a time. Subsequent references to this block of words fetch the desired contents directly from the cache.

At any given point of time, only some blocks in the main memory can be held in the cache. The correspondence between the blocks in the main memory and those in the cache is determined by a "**mapping function".**

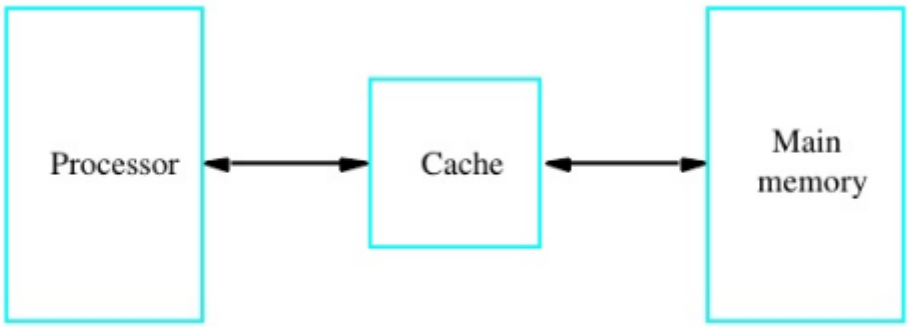


Figure 5.14. Use of a cache memory.

When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. The cache control hardware decides which block is to be removed to make space for the new block. The set of rules for making such decision form the **"replacement algorithm".**

The processor need not to know the existence of the cache. It simply issues Read or Write requests. The cache control hardware finds out whether the requested word is currently in the cache or not. If the word exists in the cache, the Read or Write operation is performed. This is called read or write hit is said to have occurred.

In a Read operation, main memory is not involved. In case of Write operation, two techniques are used. In the first technique, **write-through protocol** is used, where in the cache location and the main memory location are updated simultaneously. In second technique, only the cache location is updated and it is marked as updated with a flag bit called as **dirty or modified bit.** The main memory location is updated later, when this marked block is to be removed from the cache to make space for new block. This is called as **write-back, or copy-back protocol.**

If the data is not present in the cache, then a Read miss or Write miss occurs. When a read miss occurs, the block of words containing this requested word is transferred from the memory. After the block is loaded, the desired word is forwarded to the processor. Alternatively,

the desired word may also be forwarded to the processor as soon as it is read from the main memory, thereby reducing the waiting time of the processor. This is called **load-through or early-restart.**

Write-miss occurs when the addressed word is not present in the cache. If the write-through protocol is used, then the contents of the main memory are updated directly. If write-back protocol is used, then the block containing the addressed word is first brought into the cache and then the desired word is overwritten with new information.

UNIT- 6: Processors And Pipelining

INTRODUCTION TO PROCESSORS

6.1 PROCESSORS

A processor is the logic circuitry that responds to and processes the basic instructions that drive a computer. The four primary functions of a processor are fetch, decode, execute and write back.

Most processors today are multi-core, which means that the IC(Integrated Circuit) contains two or more processors for enhanced performance, reduced power consumption and more efficient simultaneous processing of multiple tasks (*see:* parallel processing). Multi-core set-ups are similar to having multiple, separate processors installed in the same computer, but because the processors are actually plugged into the same socket, the connection between them is faster.

6.2 ADVANCED PROCESSOR TECHNOLOGY

Advanced processor technology includes RISC, CISC, superscalar, VLIW, Super-pipelined, Vector and symbolic processors. Scalar and vector processors are used for numerical computations. Symbolic processors are developed for AI(Artificial Intelligence) applications.

Various processor families can be categorised according to clock rate versus cycles per instruction (CPI) as shown in the figure.

- RISC and CISC processors are designed for multi-core chips, embedded applications , low cost or low power consumption and tend to have lower clock space. However high performance processors must necessarily be designed to operate at high clock speeds.
- Superscalar processor is a subclass of RISC processors. It allows multiple instructions to be issued simultaneously during one clock cycle.
- The VLIW (very long instruction word) processors use more functional units than superscalar processor.
- Processors in vector supercomputers use multiple functional units for concurrent scalar and vector operations.

6.3 INSTRUCTION SET ARCHITECTURE

A complete instruction set is often referred to as the *Instruction Set Architecture* (ISA). It is the part of the processor that is visible to the programmer or compiler writer. The ISA serves

as the boundary between software and hardware. ISA specifies the addressing modes used for accessing data operands and the processor registers available for use by the instructions.

The first two philosophies to instruction sets were: reduced (RISC) and complex (CISC).

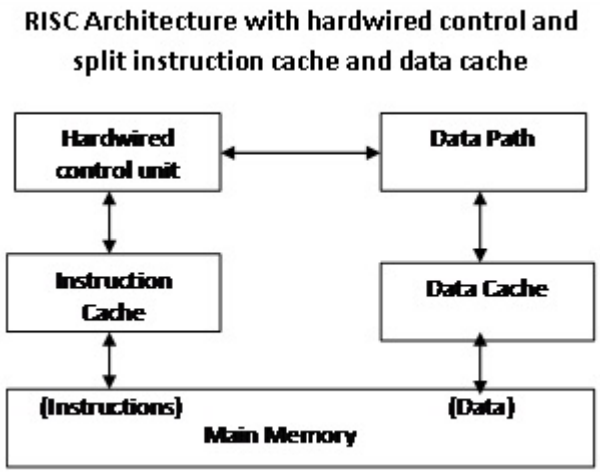


Figure 6.2

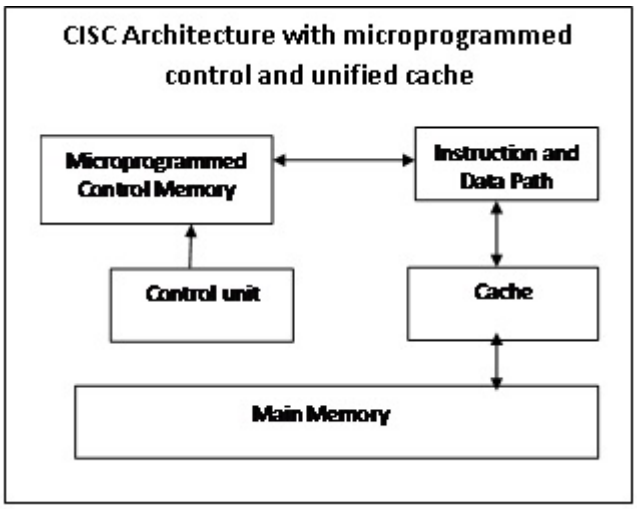


Figure 6.1

6.3.2 RISC ARCHITECTURE or *Reduced Instruction Set Computer*:

It is a type of microprocessor architecture that utilizes a small, highly-optimized set of instructions, rather than a more specialized set of instructions often found in other types of architectures.

RISC provides higher performance by executing instructions using fewer microprocessor cycles per instruction.

RISC processors have a CPI (clock per instruction) of one cycle. This is due to the optimization of each instruction on the CPU and a technique called pipelining.

Features of RISC architecture:

- In RISC processors there is one instruction per machine cycle.
- RISC instructions are simpler and can execute as fast as microinstruction on CISC machines.
- RISC provide multiple set of registers.
- It uses simple addressing modes.
- It uses simple instruction formats with fixed instruction length.
- It simplifies the design of control unit.
- To speed-up instruction execution RISC uses instruction pipelining.

RISC families include DEC Alpha, AMD Am29000, ARC, ARM, Atmel AVR, Blackfin, Intel i860 and i960, MIPS, Motorola88000, PA-RISC, Power (including PowerPC), RISC-V, SuperH, and SPARC.

6.3.2 CISC ARCHITECTURE

- CISC is an acronym for Complex Instruction Set Computers.
- CISC processors use micro programmed control.
- It is based on concept of using very large instruction set (VLSI) having simple as well as complex instructions and keeps the program length as small as possible.
- In CISC complex instructions may take multiple cycles for execution.
- CISC instructions vary in size, specify a sequence of operations, and require serial (slow) decoding algorithms.
- They tend to have few registers, and the registers may be special purpose.
- To add the flexibility in the instruction set they support more and complex addressing modes.
- Examples of CISC are IntelX86, Motorola 68000 series, DEC VAX etc.

6.3.3 COMPARISON BETWEEN RISC AND CISC

RISC Architecture uses separate instructions and data caches. Hence their access paths are different. Where as in CISC processor, there is a unified cache for holding both instructions and data. Therefore they have to share the same path for data and instructions.

The hardwired control is found in most RISC processors while the traditional CISC processors use microprogrammed control. Thus the control memory is needed in CISC processors. The modern CISC processors may also use hardwired control.

Sl no.	Characteristics	RISC	CISC
1	Instruction size	Fixed	Varies
2	Instruction length	4 bytes	1,2,3 or 4 bytes
3	Number of Instruction	Less	More
4	Instruction decodig	Easy to decode	Serial (Slow) to decode
5	Instruction semantics	Almost always one simple operation	Has simple to complex instructions, many operations per instruction
6	Addressing modes	Complex addressing mode are synthesized(reduced)	Supports complex addressing modes
7	Instruction execution speed	Medium	Slow
8	Instruction execution	Takes one cycle to execute one instruction.	Complex instruction taking multiple cycles
9	Registers	Many general purpose	Few maybe special purpose
10	Hardware	Simple	Complicated
11	Memory access	Rarely	Frequently
12	Instruction format	Regular, consistent	Field placement varies.

		placement of fields	
13	Pipelined	Highly pipelined.	Not pipelined or less pipelined.
14	compiler	complicated	Simple
15	examples	Intel X86,Motorola 68000 series	ARM,8051,ATMEL etc.

Table 6.1 Comparision Between CISC and RISC.

6.4 SUPER SCALAR PROCESSOR

Modern processors have the performance of executing one instruction per cycle by fetching, decoding and executing concurrently. This mode of operation is called superscalar.

- Superscalar processor executes multiple independent instructions in parallel.
- Common instructions (arithmetic, load/store etc) can be initiated simultaneously and executed independently
- Most operations are on scalar quantities
- Superscalar was designed to improve the performance of these operations by executing them concurrently in multiple pipelines
- In superscalar multiple independent instruction pipelines are used. Each pipeline consists of multiple stages, so that each pipeline can handle multiple instructions at a time and need less than half a clock cycle.
- Super-pipelining is the breaking of stages of a given pipeline into smaller stages (thus making the pipeline deeper) in an attempt to shorten the clock period
- Simple pipelined system performs only one pipeline stage per clock cycle.
- Super pipeline is capable of performing two pipeline stages per clock cycle.
- Superscalar performs only one pipeline stage per clock cycle in each parallel pipeline.

Advantages:

1. Hardware detects parallelism between instructions.
2. Hardware tries to issue as many as instructions as possible in parallel.

Disadvantage:

1. Very complex, much hardware is needed for run time detection.
2. Power consumption can be very large

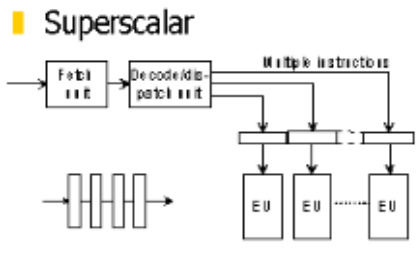
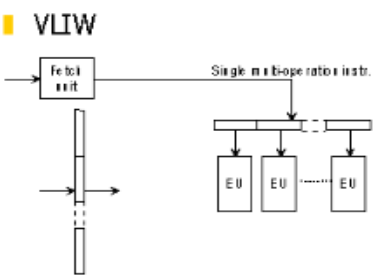
6.5 VLIW ARCHITECTURE

- Very Long Instruction Word architecture are suitable for exploiting instruction-level parallelism in programs. That is for executing more than one basic instructions at a time.
- These processors contain multiple functional units, fetch a very long instruction word having several primitive instructions from the instruction cache and dispatch the entire for parallel execution.
- These capabilities are exploited by compilers which generate code for basic instructions executable in parallel.
- Have relatively simple control logic.
- They have been described as successor to RISC, because it moves complexity from the hardware to the compiler, allowing simpler, faster processors.

Limitations of VLIW:

- Need for powerful compiler, increased code size, larger memory bandwidth.
- Binary compatibility across implementations with varying number of functional units.
- Advantages of VLIW:
- Simple hardware.
- Power consumption can be reduced.
- Compilers can detect parallelism based on global analysis of the whole program.
- Simpler instruction issue logic.
- Instructions being implemented with shorter clock cycles, than superscalar processor.

6.6 COMPARISON BETWEEN SUPER SCALAR AND VLIW.

Sl no	Super scalar architecture	VLIW architecture
1		
2	Complex hardware	Simpler hardware
3	Power consumption can be very large.	Power consumption can be reduced
4	Take longer clock cycles to execute instructions.	Take shorter clock cycles to execute instructions
5	Superscalar processors allow to fit less execution units onto a given amount of chip space than VLIW processors.	VLIW processors allow to fit more execution units onto a given amount of chip space than Superscalar processors.

Note: This is only Basic Information for students. Please refer "Reference Books" prescribed as per syllabus

6	Dynamic scheduling	Static scheduling

6.7 Multi core architecture

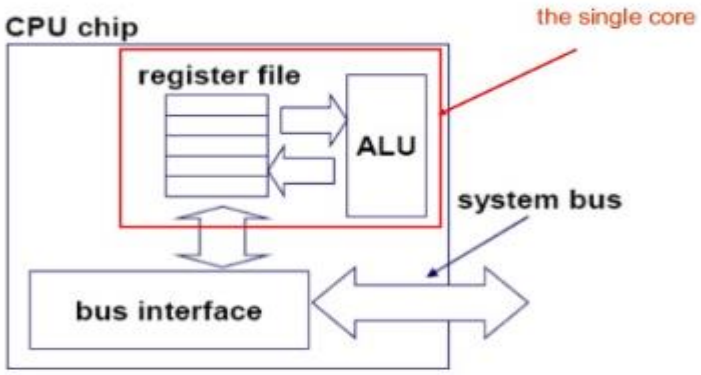


Figure 6.5(a) single core CPU chip

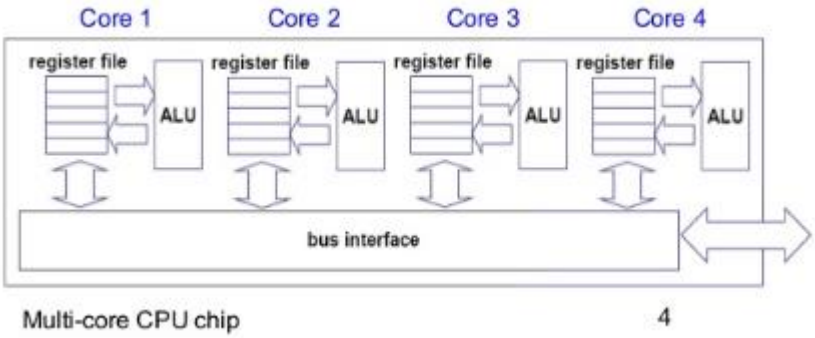


Figure 6.5(b) Multicore CPU chip

When a processor has more than one core to execute all necessary functions of a computer, then that processor is known as Multi core architecture. In other words, a chip with more than one CPU.

The multi core CPU design require much less printed circuit board space than multi chip designs.

A dual core processor uses slightly less power than two-coupled single core processors. Thus they have high performance scientific applications. They are used in mobile handsets.

6.8 Pipelining Introduction

Pipeline is an implementation technique where the phases of a computer instruction cycle overlap in execution. Pipelining is a technique of decomposing a sequential process into sub-operations, with each sub-process being executed in a special dedicated segment that operates concurrently with all other segments.

Pipelining exploits parallelism among instructions by overlapping the instructions, and it is called Instruction Level Parallelism (ILP). The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments. The name "pipeline" implies the flow of information. It is the characteristic of pipelines that several computations can be in progress in distinct segments at the same time.

6.9 Pipeline principles

A pipeline processor works based on the following two principles-

1. Linear pipeline processor
2. Non linear pipeline processor

6.9.1 Linear pipeline processor

In linear pipelining, specialized hardware stages are linearly connected to perform a fixed function. The stream of data flows from one end to the other end as shown in Figure

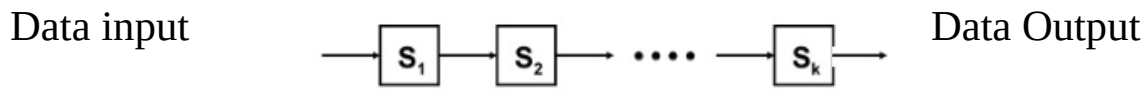


Figure 6. Basic structure of pipeline processors.

Pipelining can be applied for instruction execution, arithmetic computation, and memory accessing operations. The processor supporting such a hardware architecture is called pipeline processor.

A pipeline processor is constructed with k processing stages. Data inputs are given into the pipeline at the first stage S_1. The processed results are passed from S_1 to S_2, S_2 to S_3, and so on to the last stage S_k. A specific control mechanism is used to control the data flow along the pipeline stages. Based on the control mechanism used, we can classify pipelines into

1. Asynchronous pipeline
2. Synchronous pipeline.

6.9.1.1 Asynchronous model

In asynchronous pipeline model, the data is controlled by handshaking protocol as shown in Figure 6. When stage S_i is ready to transmit its results, it sends a ready signal to S_{i+1}. The stage S_{i+1} accepts output of S_i as input data and returns an acknowledgement signal to S_i.

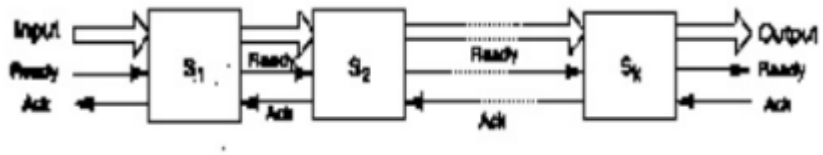


Figure 6. Asynchronous pipeline model

6.9.1.2 Synchronous model

Clocked high speed latches are used as interface between pipeline stages in synchronous pipeline model. All latches transfer data to the next stages simultaneously at the falling edge of the clock pulse. Figure 6. illustrates synchronous pipeline model.

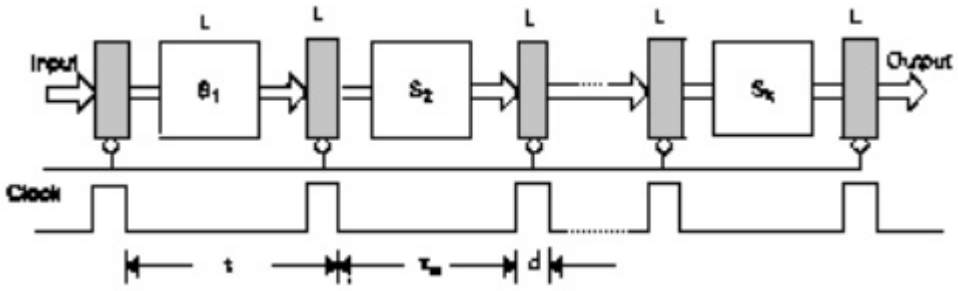


Figure 6. Synchronous pipeline model

6.9.2 Non linear pipeline processor

Linear pipeline processors do not have feedback connections and their inputs and outputs are totally independent. Some processors allow feedback and feed-forward connections in addition to streamline connections. Such processors are called non-linear pipeline or general pipeline or dynamic pipeline processors. Figure 6. shows a three stage non-linear pipeline. Along with the streamline connections from S_1 to S_2 and S_2 to S_3, there are two feedback connections from S_3 to S_2 and S_3 to S_1 and a feed-forward connection from S_1 to S_3.

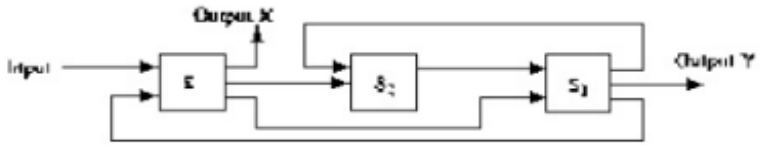


Figure 6. Three stage non-linear pipeline

In case of non-linear pipeline, the output is not necessarily from the last stage. It can also have more than one outputs, based on the data flow patterns to evaluate different functions.

6.10 Classification of pipeline processor

According to the levels of processing, Handler in 1977, has proposed the following classification scheme for pipeline processors:

Arithmetic Pipeline :
The arithmetic logic units of a computer can be segmentized for pipeline operations in various data formats. Processors can have multiple arithmetic logic units. Well-known arithmetic pipeline examples are the four-stage pipes used in Star-100, the eight-stage pipes used in the TI-ASC, upto 14 pipeline stages used in the Cray-1, and up to 26 stages per pipe in the Cyber-205.

Instruction Pipelining :
The execution of a stream of instruction can be pipelined by overlapping the execution of the current instruction with the fetch, decode, and operand fetch of subsequent instruction. This technique is also known as instruction lookahead. Almost all high-performance computers are now equipped with instruction-execution pipelines.

Processor Pipelining :
This refers to the pipeline processing of the same data stream by a cascade of processors, each of which processes a specific task. The data stream passes the

first processor with results stored in a memory block which is also accessible by the second processor. The second processor then passes the refined results to the third, and so on.

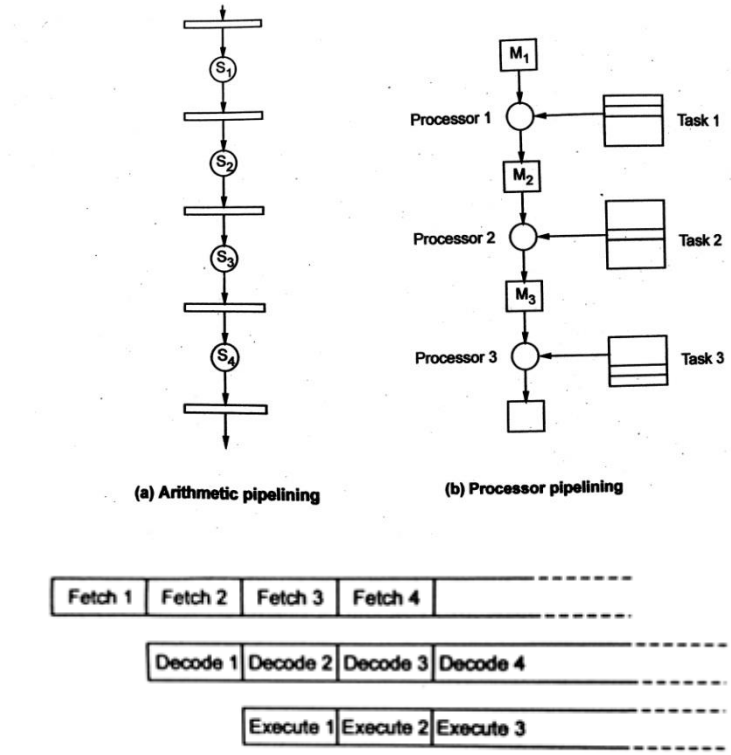


Figure 6. Handler classification of pipelined processor.

According to pipeline configurations and control strategies, Ramamooorthy and Li in 1977 have proposed the following three pipeline classification schemes:

Unifunction Vs. Multifunction Pipelines :

A pipeline unit with a fixed and dedicated function, such as the floating-point adder is called unifunctional. The Cray-1 has 12 unifunctional pipeline units for various scalar, vector, fixed-point, and floating-point operations. A multifunction pipe may perform different subsets of stages in the pipeline. The TI-ASC has four multifunction pipeline processors, each of which is reconfigurable for a variety of arithmetic logic operations at different times.

Static Vs. Dynamic Pipelines:

A static pipeline may assume only one functional configuration at a time. Static pipelines can be either unifunctional or multi-functional. Pipelining is made possible in static pipes only if instructions of the same type are to be executed continuously. The function performed by a static pipeline should not change frequently. Otherwise, its performance may be very low. A dynamic pipeline processor permits several functional configurations to exist simultaneously. In this sense, a dynamic pipeline must be multifunctional. On the other hand, a unifunctional pipe must be static. The dynamic configuration needs much more elaborate control and sequencing mechanisms than those for static pipelines. Most existing computers are equipped with static pipes, either unifunctional or multifunctional.

Scalar Vs. Vector Pipelines :

Depending on the instruction or data types, pipeline processors can be also classified as scalar pipelines and vector pipelines. A scalar pipeline processes a sequence of scalar operands under the control of DO loop. Instructions in a small DO loop are often prefetched into the instruction buffer. The required scalar operands for repeated scalar instructions are moved into a data cache in order to continuously supply the pipeline with operands. The IBM System/360 Model 91 is typical example of a machine equipped with scalar pipelines. However, the Model 91 does not have a cache. Vector pipelines are specially designed to handle vector instructions over vector operands. Computers having vector instructions are often called vector processors. The design of a vector pipeline is expanded from that of a scalar pipeline. The handling of vector operands in vector pipelines is under firmware and hardware controls (rather than under software controls as in scalar pipelines).