

DATA PATH DESIGN

Computer system consists of three main parts: CPU (Processor), Memory and I/O.

The main function of the CPU is to execute instructions.

Instructions are stored in memory by the programmer. A **set of instructions** doing a specific task is called a program.

Fetch Cycle:

Since instructions are stored in the memory, first each instruction needs to be fetched. The **process of fetching an instruction from the memory** is called as Fetch Cycle. Once fetched, the instruction is decoded, to find out the required task to be performed.

Execution Cycle:

It is the **process of executing an instruction.**

Instruction Cycle:

It is the **total time** taken to fetch, decode and execute an instruction.

MEMORY:

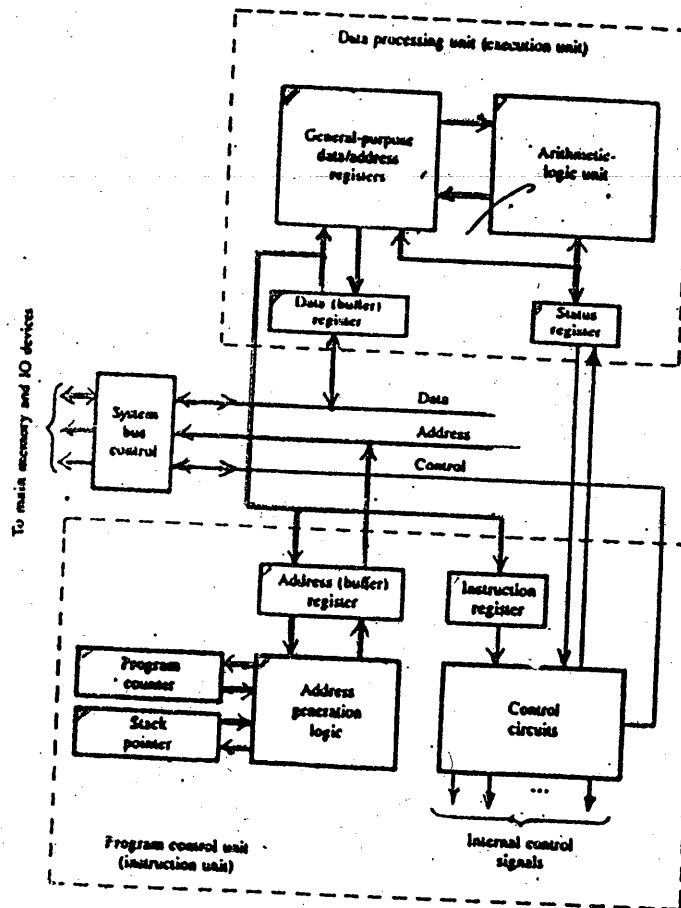
All **information stored** inside the computer system is in its memory. This information could be **programs or data**. **Permanent** information is stored in **ROM** and **runtime** information is stored in **RAM**.

INPUT/OUTPUT:

This section is responsible for **transferring data in and out** of the computer system. Keyboard, monitor, mouse etc are examples of I/O devices.

ORGANIZATION OF A TYPICAL CPU {VON NEUMANN MODEL}

Stack \Rightarrow only 1
add is req to stack
Queue \Rightarrow 2 add



The above diagram is the basis for all Processor designs. It consists of the minimum set of registers and the necessary circuit to execute instructions. It has the following parts:

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

1) Data Processing Unit

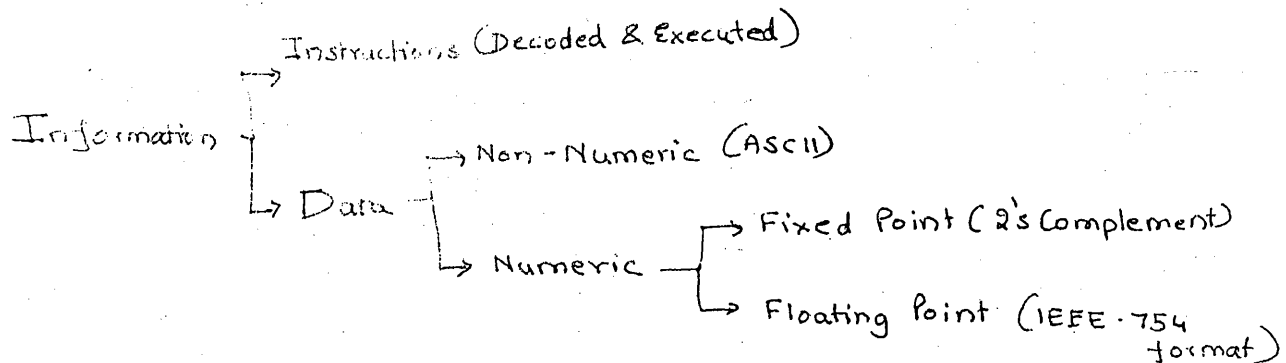
- In this unit all the **data operations** are performed as per the instruction.
- An **ALU** is present to perform **arithmetic/logic operations**.
- The **data** is **stored** in General Purpose Registers (**GPRs**).
- These registers **provide operands** to the ALU and also **store the result** of operations.
- The most important register is the **Accumulator** as it **acts as the default operand** in many operations.
- Another important register is the **Status Register** (Flag Register). It **holds the "Status" of the current result**. It has several **conditional bits**, which are changed by the ALU after every operation. They indicate conditions such as **sign, carry, zero, parity etc**. This register is checked by the Control Unit to perform a conditional branch operation (Eg: Jump on Carry --- JC)
- The programmer can read this register to find out the status of the previous operation
- Additionally there is a bi-directional data buffer between the main memory and the CPU.

2) Program Control Unit

- This unit is used to **fetch instructions** and also **transfer data** in and out of the processor.
- Instructions are fetched from the memory.
- **Program Counter (PC)** is a register that **stores address of the next instruction** to be fetched from the memory. It is **incremented after every instruction** and hence the program proceeds **sequentially**.
- The fetched instruction is stored in the **instruction register**.
A CPU may have buffers to store several instructions.
- **During a branch** the contents of **PC** are **changed** to point to the branch location.
- **Stack Pointer** is a register that contains the **address of the top-most element of the stack**. A Stack is a set of memory locations organized in a **LIFO** manner. When we **PUSH** an element into the stack, **SP** is decremented. It is incremented when we **POP** an element from the stack.
- When a program invokes (calls) a sub-routine (function), the **return address is stored in the stack**. This address is used to come back from the sub-routine to the main program.
- The **Address Generation logic** generates the memory address using **PC** or **SP**.
- The **Control Circuit** decodes the instruction and produces the appropriate control signals for its execution.

INFORMATION REPRESENTATION

Any form of information is stored in computers by means of binary sequences organized into words. A word is a unit of fixed length of bits. The no. of bits is called as the word size. The basic information types are as follows:



FIXED POINT BINARY NUMBERS

An n -bit binary number ($b_{n-1} \dots b_0$) can be represented in three forms:

- Signed Magnitude
- 1's Complement
- 2's Complement

In all the three methods, -ve numbers are represented with a "1" at the MSB position. Let's examine the three methods for a 4-bit number:

Number	Sign Magnitude	1's Complement	2's Complement
+7	0 1 1 1	0 1 1 1	0 1 1 1
+6	0 1 1 0	0 1 1 0	0 1 1 0
+5	0 1 0 1	0 1 0 1	0 1 0 1
+4	0 1 0 0	0 1 0 0	0 1 0 0
+3	0 0 1 1	0 0 1 1	0 0 1 1
+2	0 0 1 0	0 0 1 0	0 0 1 0
+1	0 0 0 1	0 0 0 1	0 0 0 1
+0	0 0 0 0	0 0 0 0	0 0 0 0
-0	1 0 0 0	1 1 1 1	0 0 0 0
-1	1 0 0 1	1 1 1 0	1 1 1 1
-2	1 0 1 0	1 1 0 1	1 1 1 0
-3	1 0 1 1	1 1 0 0	1 1 0 1
-4	1 1 0 0	1 0 1 1	1 1 0 0
-5	1 1 0 1	1 0 1 0	1 0 1 1
-6	1 1 1 0	1 0 0 1	1 0 1 0
-7	1 1 1 1	1 0 0 0	1 0 0 1
-8	Extra Combination →		1 0 0 0

Advantage of 2's Complement method is that the same bit pattern specifies +ve 0 as well as -ve 0, which is correct.

Moreover, since 0 (+ or -) occupies the same bit pattern, we are left with 1 spare bit pattern (1000), which can be used to represent -8. This would increase the range by 1 number.

Thus Range for a 4-bit 2's complement numbers is: -8 ... 0 ... +7; whereas that for both sign Magnitude Form and 1's complement numbers is: -7 ... 0 ... +7 for a 4-bit number.

Hence 2's complement method is preferred. (Viva)

Similarly, for an 8-bit number the range will be from -128 ... 0 ... +127.

FLOATING POINT NUMBERS

Floating point numbers are those where the position of the point is not fixed (i.e. the point floats in the number).

Eg: 10101.0110001

Such numbers are first Normalized.

Normalization: Shifting the point left or right so that there is only one non-zero digit to the left of the point is called Normalization.

A normalized number is of the form:

$$(-1)^S \times 2^E \times 1.M$$

S = Sign of the number: (1 \rightarrow -ve; 0 \rightarrow +ve)

E = Exponent.

M = Mantissa.

Eg: Consider the floating point number 10101.0110001

Shifting the number right by 4 places we get: (PS: right-shifting the number means left-shifting the point!)
 1.01010110001×2^4

Since it is a +ve number, we can write it as:

$$(-1)^0 \times 2^4 \times 1.01010110001$$

This is the normalized form of the given number.

IEEE 754 32-bit Format {Single Precision / Short Real}

S	E	M
Sign (1)	Exponent (8)	Mantissa (23)

- 1) The IEEE 754 format stores a Normalized number as shown above.
Here: **S**: 1-bit used to indicate the sign. (1: -ve; 0: +ve)
E: 8-bit Biased exponent. (Biased Exponent = True exponent + 127)
M: 23 bit Mantissa.
- 2) Instead of storing the true exponent, a **bias value of 127** is added to the exponent. This is done to **allow -ve exponents** to be stored **without using another sign bit**. Since the exponent field is 8-bit, the max number here can be 255. (1111 1111₂). Therefore range of the biased exponent that can be stored is: $0 \leq E \leq 255$. Note that both 0 as well as 255 are considered as **error conditions**, hence the valid range of the biased exponent is: $1 \leq E \leq 254$. #Contact Bharat Sir for any doubts on 98204 08217.
VIVA: The bias value is taken as 127 because it lies exactly in the middle of 0 ... 255. Thus it supports an equal number of +ve and -ve exponents.
- 3) In the Mantissa field, the "1" of 1.M need not be stored. This is because all Normalized numbers would have a 1 before the point. Not storing the "1" **saves 1-bit space** in every number and hence **increases the precision**.

Ex: Convert $(9.0)_{10}$ into IEEE 754 32-bit format.

Step 1> Convert the number into binary.
 $(9.0)_{10} = (1001.0)_2$

Step 2> Normalize the number.
 $1001.0 = (-1)^0 \times 2^3 \times 1.0010$
Hence $S = 0$; $M = 0010$; True Exp = 3.

Step 3> Calculate the Biased Exponent.
 $BE = TE + Bias$
 $Bias = 127$
 $BE = 3 + 127 = 130$.

Step 4> Convert the Biased Exponent into binary.
 $BE = (130)_{10} = (1000\ 0010)_2$
#Refer lecture notes for short-cuts.

Step 5> Represent the Number in the required format:

0	1000 0100	00100000....
Sign (1)	Exponent (8)	Mantissa (23)

#An additional step to convert this binary series into hex may be performed though not compulsory.

More sums:

2) Convert 2A3BH into Single Precision format

Converting the number into binary we get:
0010 1010 0011 1011

Normalizing the number we get:

$$(-1)^0 \times 1.0101000111011 \times 2^{13}$$

Here $S = 0$; $M = 0101000111011$; True Exponent = 13.

Bias value for Single Precision format is 127:

$$\begin{aligned}\text{Biased Exponent (BE)} &= \text{True Exponent} + \text{Bias} \\ &= 13 + 127 \\ &= 140.\end{aligned}$$

Converting the Biased exponent into binary we get:
Biased Exponent (BE) = (1000 1100)

Representing in the required format we get:

0	10001100	010100011101100...
S (1)	Biased Exp (8)	Mantissa (23)

Converting the number into hexadecimal form we get:
4628EC00H ... 32 bits.

3) Convert $(12.125)_d$ into Single Precision format

Converting the number into binary we get:
 1100.001 © For doubts contact Bharat Sir on 98204 08217

Normalizing the number we get:

$$(-1)^0 \times 1.100001 \times 2^3$$

Here $S = 0$; $M = 100001$; True Exponent = 3.

Bias value for Single Precision format is 1023:

$$\begin{aligned}\text{Biased Exponent (BE)} &= \text{True Exponent} + \text{Bias} \\ &= 3 + 1023 \\ &= 1026.\end{aligned}$$

Converting the Biased exponent into binary we get:

Biased Exponent (BE) = (100 0000 0010)

Representing in the required format we get:

0	10000000010	10000100000...
---	-------------	----------------

S	Biased Exponent	Mantissa
(1)	(11)	(52)

Converting the number into hexadecimal form we get:
4028 4000 0000 0000 H (64 bits).

Exceptions/Errors in Floating Point representation:

1) Overflow

- When the **exponent is too large to be stored**, the condition is called as overflow. It occurs when the magnitude of the number becomes so large that it goes out of range.
- This error is represented by **Exponent = 255** and is called as **NaN (Not a Number)**.
- Eg:** True exp = 128.
Therefore $BE = TE + \text{bias}$.
 $BE = 128 + 127$.
BE = 255. This is NaN.
- An extreme case of NaN is when the magnitude goes on increasing till it is no longer finite. i.e. the number has reached Infinity. **Infinity is denoted by E = 255 and M = 0.**

2) Underflow

- When the **exponent is too small to be stored**, the condition is called as underflow. It occurs when the number becomes so small that it cannot be normalized.
- This error is represented by **Exponent = 0** and is called as **Denormal Number**.
- Eg:** True exp = -128.
Therefore $BE = TE + \text{bias}$.
 $BE = -128 + 127$.
BE = -1. This cannot be stored as it is still -ve even after biasing.
This is a Denormalized Number
- An extreme case of Denormal Number is when the magnitude goes on decreasing till it reaches 0. **Zero is denoted by E = 0 and M = 0.**

Summary:

TYPE	EXPONENT	MANTISSA	CONDITION
Normal (Valid Number)	$0 < E < 255$	$M = X$	Normal
NaN	$E = 255$	$M = X$	Overflow
Infinity	$E = 255$	$M = 0$	
Denormal Number	$E = 0$	$M = X$	Underflow
Zero	$E = 0$	$M = 0$	

IEEE 754 (64-bit) Format {Double Precision / Long Real}

S	E	M
Sign (1)	Exponent (11)	Mantissa (52)

- This format is very similar to the 32-bit format except that the sizes of Exponent and Mantissa fields are increased. This **increases the range** of numbers that can be stored.
- Since the Exponent is 11-bits the **Bias Value is = 1023**. $\{(2^{11-1}) - 1 = 1023 \text{ --- VIVA}\}$

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

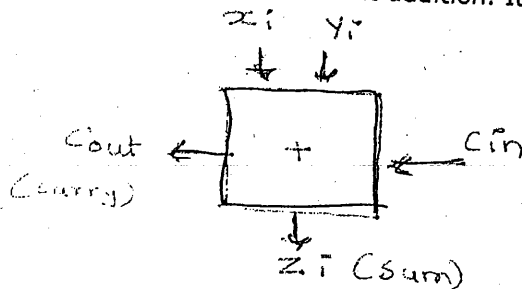
ADDER CIRCUITS

Half Adder:

It is a circuit, which simply adds 2 bits without taking into account the previous carry. It produces the sum and the carry due to the current addition.

Full Adder:

This circuit not only adds the current two bits but also considers the previous carry. It produces the sum and the carry due to the current addition. It is represented as:



X_i	Y_i	C_{in}	Z_i	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

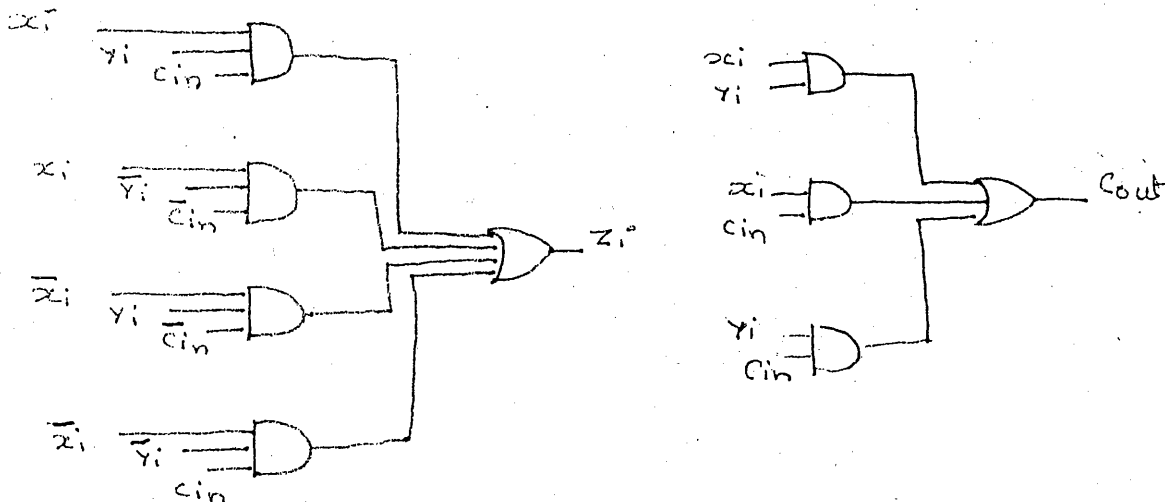
Input: X_i, Y_i : bits to be added; C_{in} : Previous Carry input;
Output: Z_i : Sum; C_i : Carry

Calculation: $Z_i = X_i \oplus Y_i \oplus C_{in}$.

$C_i = X_i Y_i + (X_i + Y_i) C_{in}$

i.e: $Z_i = X_i Y_i C_{in} + X_i \bar{Y}_i \bar{C}_{in} + \bar{X}_i Y_i \bar{C}_{in} + \bar{X}_i \bar{Y}_i C_{in}$.
i.e: $C_i = X_i Y_i + X_i C_{in} + Y_i C_{in}$.

Circuit:



drawbacks:- serial adder, \therefore takes long time.

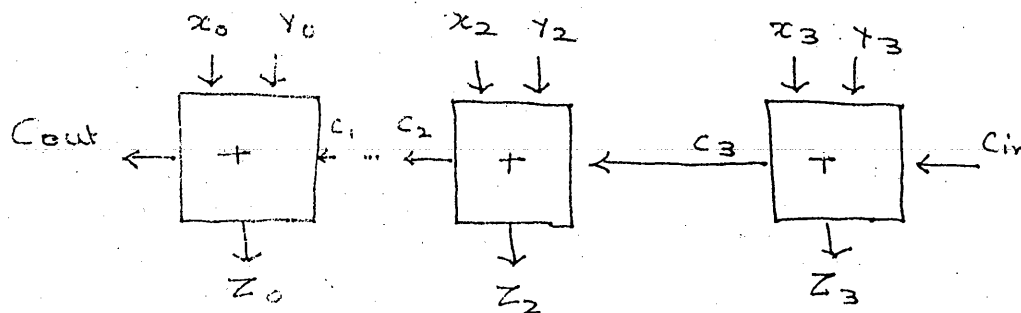
Addition of n-bits {Eq: 4-bit}

Addition of multiple bits can be carried out in 2 ways:

1) Ripple Carry Adder: {Serial Adder}

- A Ripple Carry adder to add n-bits is formed by simply connecting n full-adders serially such that the carry from the LSB propagates serially up to the MSB, as shown.

Inputs: $X = X_0 X_1 X_2 X_3$
 $Y = Y_0 Y_1 Y_2 Y_3$: Bits to be added;
 C_{i+1} : Previous Carry input;
Outputs: Z_i : Sum; C_i : Carry



- Though **simple**, the disadvantage here is that, addition of the various bits takes place one **after the other** and not **concurrently**. Hence this **method is slow**. This delay can make the ALU very slow.

~~Parally~~
Parally

C_{i+1}	X_i	Y_i	C_{i+1}	Z_i	C_i
00	0	0	0	0	0
01	0	0	0	1	0
11	0	1	0	1	0
10	1	0	0	1	0
	1	1	1	1	1
	1	0	1	0	1
	1	1	1	0	1
	1	1	1	1	1

$$X_i Y_i + Y_i C_{i+1} + X_i C_{i+1}$$

$$= X_i Y_i + [X_i + Y_i] C_{i+1}$$

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

★ 2) Carry Look Ahead Adder: {Parallel Adder.} (Imp) ★

- The Carry-Look Ahead Adder has the advantage that all the bits of the 2 numbers can be **added simultaneously**.
- This makes the operation much faster as compared to serial adder.
- But here, the **carry** required at each bit position needs to be **predicted** (calculated) first. This is done as follows:

Inputs: $X = X_0 X_1 X_2 X_3$
 $Y = Y_0 Y_1 Y_2 Y_3$: bits to be added;
 C_{in} : Previous Carry input;
Outputs: $Z = Z_0 Z_1 Z_2 Z_3$: Sum;
 $C_{out} = C_0$: Carry

We know that, at any stage the current carry produced C_i is:

$$C_i = X_i Y_i + X_i C_{i+1} + Y_i C_{i+1}$$

$$= \underbrace{X_i Y_i}_{g_i} + \underbrace{(X_i + Y_i)}_{p_i} C_{i+1}$$

Meaning:

$$g_i = X_i \cdot Y_i$$

It is the carry to be **generated** by the current addition.

g_i will be 1 only when both X_i and Y_i are 1.

$$p_i = X_i + Y_i$$

The carry to be **propagated** to the next stage.

- p_i will be 1 when **atleast** X_i or Y_i is 1, AND there is a previous carry (C_{i+1}).

Also notice p_i will be 0 if both X_i and Y_i are 0 irrespective of (C_{i+1}).

Please refer Bharat Sir's Lecture Notes for examples on this

$$\therefore C_i = g_i + p_i C_{i+1}$$

Now $C_3 = g_3 + p_3 C_{in}$

Also $C_2 = g_2 + p_2 C_3$

$$\therefore C_2 = g_2 + p_2 (g_3 + p_3 C_{in})$$

$$\therefore C_2 = g_2 + p_2 g_3 + p_2 p_3 C_{in}$$

Also $C_1 = g_1 + p_1 C_2$

$$\therefore C_1 = g_1 + p_1 (g_2 + p_2 g_3 + p_2 p_3 C_{in})$$

$$\therefore C_1 = g_1 + p_1 g_2 + p_1 p_2 g_3 + p_1 p_2 p_3 C_{in}$$

Finally $C_0 = g_0 + p_0 C_1$

$$\therefore C_0 = g_0 + p_0 (g_1 + p_1 g_2 + p_1 p_2 g_3 + p_1 p_2 p_3 C_{in})$$

$$\therefore C_0 = g_0 + p_0 g_1 + p_0 p_1 g_2 + p_0 p_1 p_2 g_3 + p_0 p_1 p_2 p_3 C_{in}$$

As we notice, all the four Carries required to perform the operation can be calculated beforehand, and thus **all** the four **bits** can then be **added together**. Though a little more complicated than serial adder, this method is much **faster**.

Due to the complex calculation, Look Ahead Adders are useful for adding **upto 8 bits**. For more bits, a series of such adders can be used.

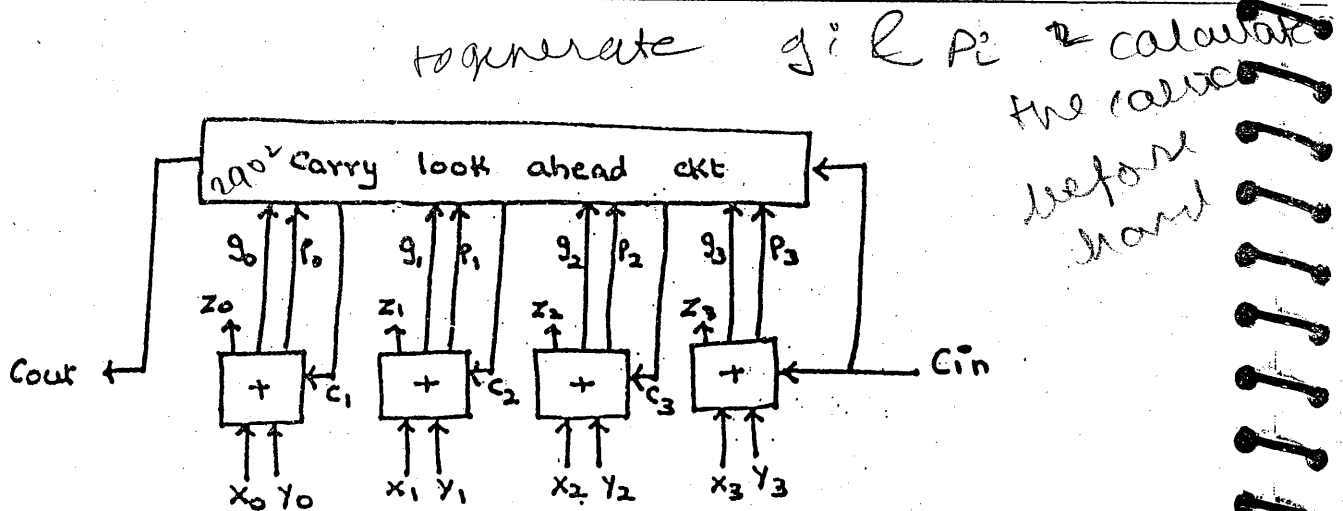
CHOPRA ACADEMY

Computer Organization
and Architecture

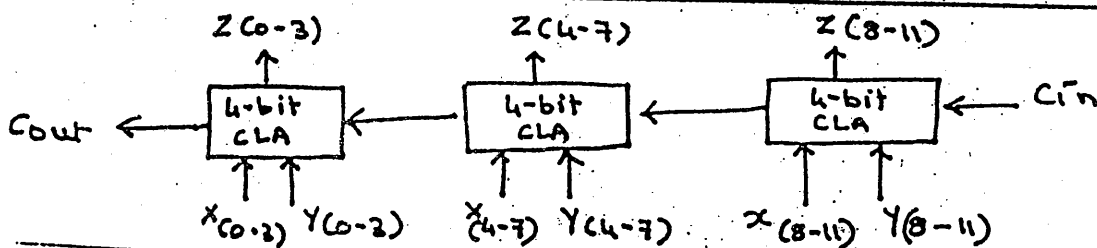
Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

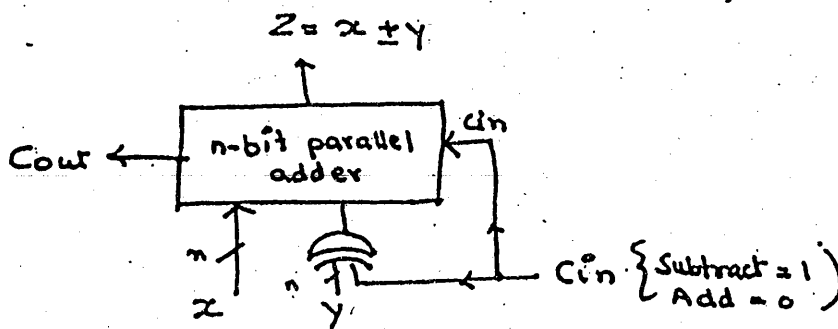
Circuit:



12 Bit adder using 3 4-bit LAC's in series:-



General Representation of a Parallel Adder: (to add n-bits)



Status: $S = 0$ --- Addition;

$S = 1$ --- Subtraction;

S	X	Y	Ex-OR output	Explanation
0	0	0	0	Add X and Y
0	1	1	1	" "
1	0	1	1	Subtract using 2's complement of y
1	1	0	0	" "

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra : 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

MULTIPLICATION

1) Conventional Method {Pencil-Paper method}

Eg:

	1010 ... (10) Multiplicand	
x	11110 ... (30) Multiplier	
	0000	} Partial products
	1010x	
	1010x	
	1010x	
	+ 1010x	
	<u>100101100</u> ... (300) Result	

- As seen above, multiplication is carried out in stages.
- At each stage the Multiplicand (here 10), is multiplied with a corresponding bit of the multiplier (here 30), to produce a partial product.
- Each partial product is **shifted** by one position and **then added** to the previous one.
- This goes on till all the bits of the multiplier are used.
- The final product is thus produced.
- Notice that in binary no. system, the **partial product** at any step is **either the Multiplicand itself** (when multiplier bit = 1), or **0** (when multiplier bit = 0).

Algorithm:

- The Multiplier is scanned right to left. For each bit of the multiplier, the partial product is formed as follows:
- If (Bit = 1) then **Multiplicand (M) is added** to the partial product, and all the bits are **shifted to the right**.
If (Bit = 0) then addition is not performed.
All bits are **simply shifted** to the right.
- Step (ii) is repeated for all bits of the multiplier.

Advantage: This is the simplest method of multiplication for +ve numbers, and the circuit design is easy.

Disadvantage: This method fails for -ve operands.

Eg:

	1010 ... (+10) Multiplicand	
x	100010 ... (-30) Multiplier	
	0000	} Partial products
	1010	
	0000	
	0000	
	0000	
	+1010	
	<u>-101010100</u>	

→ 2's complement → 010101100 = 172 ... WRONG!

Hence the method fails for -ve operands.

Implementation:

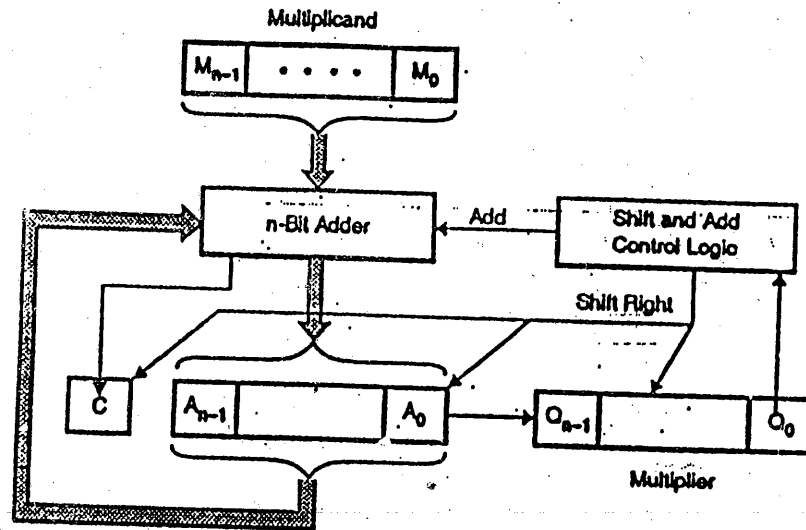


Fig. 2.11 : Unsigned Multiplication : Block Diagram

Here: B = Multiplicand Q = Multiplier
A, Q = Final Result C = Intermediate carry

Flowchart:

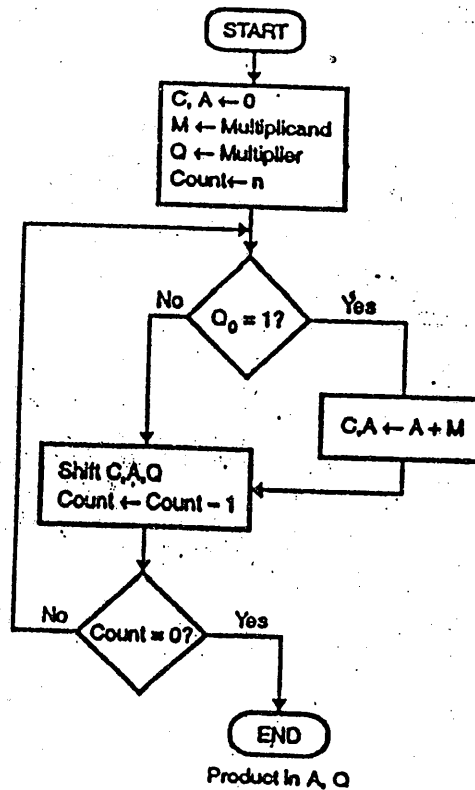


Fig. 2.12 : Flowchart for Unsigned Binary Multiplication

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

2) 2's Complement Method {Robertson's method}

This method works for both +ve and -ve operands. It modifies the conventional method to handle -ve operands. It has the following 4 cases for the example $5 \times 4 = 20$:

5 \rightarrow 0101 \rightarrow 2's complement \rightarrow 1011
4 \rightarrow 0100 \rightarrow 2's complement \rightarrow 1100
20 \rightarrow 010100 \rightarrow 2's complement \rightarrow 101100

Case 1> Both Multiplicand and Multiplier are +ve $\{(+5) \times (+4) = (+20)\}$.
Here the method works just like the conventional method.

Eg: 0101 ... (+5) Multiplicand
x 0100 ... (+4) Multiplier
 0000
 0000
 0101
+ 0000

 0010100 ... (+20) Result

Partial products

Handwritten calculation for Case 1:
0101
x 0100

10101
0000

1010100

Case 2> Multiplicand is -ve $\{(-5) \times (+4) = (-20)\}$.

Here at each step, when a non-zero partial product is produced, it is shifted with "leading 1's" as it is a -ve product. This is because the multiplicand was -ve.

Eg: 1011 ... (-5) Multiplicand
x 0100 ... (+4) Multiplier
 0000 ... zero hence nothing special
 0000 ... zero hence nothing special
 ①1011 ... non-zero hence padded with leading 1.
+ 0000 ... zero hence nothing special

 1101100 ... (-20) Result

Handwritten calculation for Case 2:
1011
x 0100

1011001
0000

101100100

Case 3> Multiplier is -ve $\{(+5) \times (-4) = (-20)\}$.

Here at the last step, the partial product that is produced due to the MSB of the multiplier is subtracted instead of being added. It is treated as a -ve partial product as it is produced by multiplying the Multiplicand with the sign bit of the Multiplier.

Eg: 0101 ... (+5) Multiplicand
x 1100 ... (-4) Multiplier
 0000 ...
 0000 ...
+ 0101 ... add the partial products so far.
 0010100 ...
- 0101000 ... now subtract the final partial product.

 1101100 ... (-20) Result

Case 4> Both Multiplier and Multiplicand are -ve $\{(-5) \times (-4) = (+20)\}$.

Here rules of case 2 as well as case 3 apply.

Eg: 1011 ... (-5) Multiplicand
x 1100 ... (-4) Multiplier
 0000 ...
 0000 ...
+ ①1011 ... non-zero hence padded with leading 1.
 1101100 ... add the partial products produced so far
- 0101000 ... now subtract the final partial product.

 0010100 ... (+20) Result

3) Booth's Algorithm {V.Imp}

This is a faster method of multiplication which works equally well on both +ve and -ve nos.

It uses a simple concept as shown:

$$10 \times 30 = 10 \times (32 - 2)$$

$$\begin{array}{r} 1010 \dots (10) \quad (-) \\ x 100000 \dots (32) \\ \hline 1 \text{ add} \\ \text{and 5 shift} \\ \text{operations} \end{array}$$

$$\begin{array}{r} 1010 \dots (10) \\ x 10 \dots (2) \\ \hline 1 \text{ add} \\ \text{and 1 shift} \\ \text{operations} \end{array}$$

100
1000

As we notice, here only 2 Add operations are required, while in the conventional method, 4 Add operations were required.

Addition operations take much more time as compared to shift operations.

As the no. of addition steps is reduced, this algorithm is faster than the earlier method.

$$\text{Now: } 10 \times 30 = 10 \times (32 - 2)$$

$$= 10 \times (100000 - 0010)$$

$$= 10 \times (1000 - 10) \rightarrow \text{This is the Recoded Booth's Multiplier Format.}$$

The Booths Algorithm works as follows:

Examine 2 adjacent bits of the Multiplier from right to left (Assuming a 0 after the LSB), and generate the Recoded Booths Multiplier as follows:

If moving from **0 to 1**: **Subtract multiplicand** from partial product and shift. **(-1)**

1 to 0: **Add multiplicand** to partial product and shift. **(+1)**

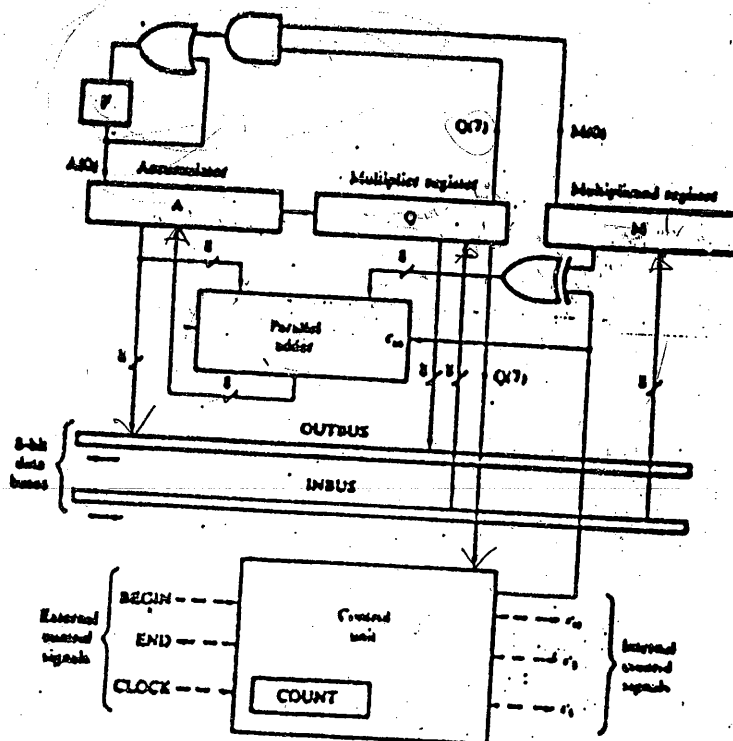
0 to 0: No addition or subtraction required, just shift. **(0)**

1 to 1: No addition or subtraction required, just shift. **(0)**

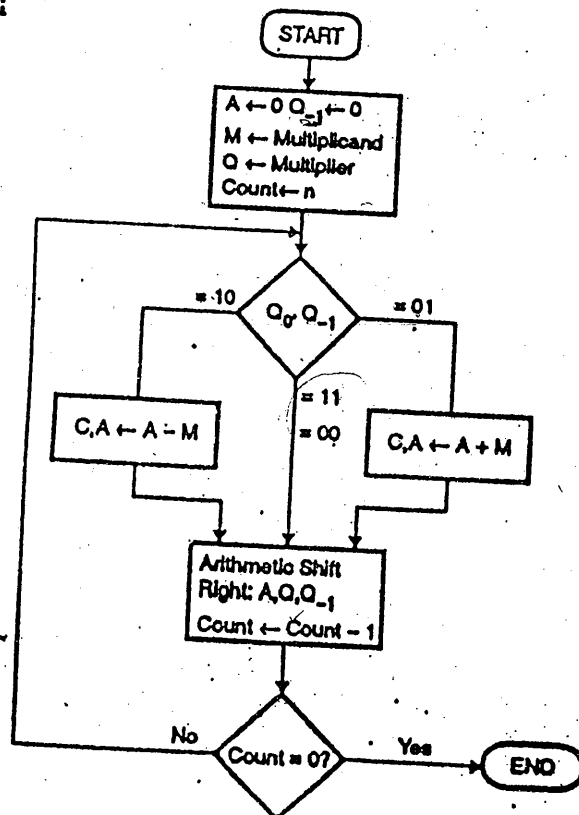
{Refer lecture notes for more on this}

Thus by performing the above process, the multiplication is performed.

Hardware Implementation:



Flowchart:



CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Examples:

1) (+10) x (+30)

Multiplicand = +10 (1010)

Multiplier = +30 (11110)

The Recoded Booth's Multiplier is generated as follows:

0 1 1 1 1 0 0
N N N N N
+1 0 0 0 -1 0

The Multiplication is thus performed as follows:

	1010	...	(10) Multiplicand
x	+1000-10	...	Booth's Recoded Multiplier
	0000000000		
	1111101100		(2's complement of 10, preceded by leading 1's as it is -ve)
	0000000000		
	0000000000		
	0000000000		
+	0101000000		
=	<u>0100101100</u>	→	This is the result (+300).

Since the MSB (Sign bit) is 0 the Result is +ve and hence no more steps are required.

2) (-10) x (+30)

Multiplicand = -10 (Since the Multiplicand is -ve, use its 2's Comp. = 10110)

Multiplier = +30 (11110)

The Recoded Booth's Multiplier is generated as follows:

0 1 1 1 1 0 0
N N N N N
+1 0 0 0 -1 0

The Multiplication is thus performed as follows:

	10110	...	(-10) Multiplicand
x	+1000-10	...	Booth's Recoded Multiplier
	0000000000		
	0000010100		(2's complement of -10 = -1 x -10 = 10)
	0000000000		
	0000000000		
	0000000000		
+	1011000000		
=	<u>1011010100</u>		

Since the MSB (Sign bit) is 1 the Result is -ve and hence take its 2's Complement:

1011010100 → 1's Complement → 0100101011

+ 1

0100101100 → This is the answer (-300)

#In case of doubts please contact Bharat Sir: 98204 08217.

3) (+10) x (-30)

Multiplicand = +10

Multiplier = -30 (Since it is -ve, use its 2's Comp = 011110 → 100001 + 1 → 100010)

The Recoded Booth's Multiplier is generated as follows:

1 0 0 0 1 0 0
~~~~~  
-1 0 0 +1 -1 0

The Multiplication is thus performed as follows:

|              |                                                              |
|--------------|--------------------------------------------------------------|
| 1010         | ... (10) Multiplicand                                        |
| x -100+1-10  | ... Booth's Recoded Multiplier                               |
| 0000000000   |                                                              |
| 1111101100   |                                                              |
| 0000101000   | (2's complement of 10, preceded by leading 1's as it is -ve) |
| 0000000000   |                                                              |
| 0000000000   |                                                              |
| + 1011000000 |                                                              |
| = 1011010100 |                                                              |

Since the MSB (Sign bit) is 1 the Result is -ve and hence take its 2's Complement:  
1011010100 → 1's Complement → 0100101011

+ 1  
0100101100 → This is the answer (-300)

### 4) (-10) x (-30)

Multiplicand = -10 (Since the Multiplicand is -ve, use its 2's Comp. = 10110)

Multiplier = -30 (Since it is -ve, use its 2's Comp = 011110 → 100001 + 1 → 100010)

The Recoded Booth's Multiplier is generated as follows:

1 0 0 0 1 0 0  
~~~~~  
-1 0 0 +1 -1 0

The Multiplication is thus performed as follows:

10110	... (-10) Multiplicand
x -100+1-10	... Booth's Recoded Multiplier
0000000000	
0000010100	
1111011000	(2's complement of -10 = -1 x -10 = 10)
0000000000	(-10 = 10110 preceded by 1's as it is a -ve no)
0000000000	
+ 0101000000	
= 0100101100	→ This is the result (+300).

Since the MSB (Sign bit) is 0 the Result is +ve and hence no more steps are required.
#In case of any doubts please contact Bharat Sir on 98204 08217.

Disadvantage: Booth's Algo is most effective when there is a continuous series of 1s or 0s in a number (Eg: 11110). But if the number has alternate 1s and 0s then Booth's algo take more time than even conventional multiplication. Eg: 101010 ... here Booth's algo will need 3 additions and 3 subtractions, while conventional method will need 3 additions and 3 shift operations.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

BOOTH'S ALGORITHM IMPLEMENTATION IN TABULAR FORM (Ref. Book: William Stallings)

Some college profs like you to solve Questions on Booths Algorithm in a tabular form as shown:

Q: Solve all cases for 7 x 6 using Booth's Algorithm:

Case1: (+7) x (+6)

M: Multiplicand = 7 \rightarrow 0111 2's Comp \rightarrow 1001 (for A-M)
Q: Multiplier = 6 \rightarrow 0110

0 0 1 0
0 0 0 1 0
0 1 0 1
1 1 0 1

Step	Operation	(A) Accumulator	(Q) (6) Multiplier	Q ₋₁	(M) (7) Multiplicand
	Initialize	0 0 0 0	0 1 1 0	0	0 1 1 1
1	Right-Shift	0 0 0 0	0 0 1 1	0	0 1 1 1
2	A \leftarrow A - M	1 0 0 1	0 0 1 1	0	0 1 1 1
	Right-Shift	1 1 0 0	1 0 0 1	1	0 1 1 1
3	Right-Shift	1 1 1 0	0 1 0 0	1	0 1 1 1
4	A \leftarrow A + M	0 1 0 1	0 1 0 0	1	0 1 1 1
	Right-Shift	0 0 1 0	1 0 1 0	0	0 1 1 1

\leftarrow Answer = 42 \rightarrow

Count
4
3
2
1
0

Case2: (-7) x (+6)

M: Multiplicand = -7 \rightarrow 1001 2's Comp \rightarrow 0111 (for A-M)
Q: Multiplier = 6 \rightarrow 0110

-7 \Rightarrow 0111
(1) 1001

Step	Operation	(A) Accumulator	(Q) (6) Multiplier	Q ₋₁	(M) (-7) Multiplicand
	Initialize	0 0 0 0	0 1 1 0	0	1 0 0 1
1	Right-Shift	0 0 0 0	0 0 1 1	0	1 0 0 1
2	A \leftarrow A - M	0 1 1 1	0 0 1 1	0	1 0 0 1
	Right-Shift	0 0 1 1	1 0 0 1	1	1 0 0 1
3	Right-Shift	0 0 0 1	1 1 0 0	1	1 0 0 1
4	A \leftarrow A + M	1 0 1 0	1 1 0 0	1	1 0 0 1
	Right-Shift	1 1 0 1	0 1 1 0	0	1 0 0 1

\leftarrow Answer = -42 \rightarrow

Count

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Case3: (+7) x (-6)

M: Multiplicand = +7 → 0111; 2's Comp → 1001 (For A-M)
Q: Multiplier = -6 → 1010 (2's complement)

Step	Operation	(A) Accumulator	(Q) Multiplier	Q _n	(M) Multiplicand	Count
	Initialize	0 0 0 0	1 0 1 0	0	0 1 1 1	
1	Right-Shift	0 0 0 0	0 1 0 1	0	0 1 1 1	
2	A ← A - M	1 0 0 1	0 1 0 1	0	0 1 1 1	
	Right-Shift	1 1 0 0	1 0 1 0	1	0 1 1 1	
3	A ← A + M	0 0 1 1	1 0 1 0	1	0 1 1 1	
	Right-Shift	0 0 0 1	1 1 0 1	0	0 1 1 1	
4	A ← A - M	1 0 1 0	1 1 0 1	0	0 1 1 1	
	Right-Shift	1 1 0 1	0 1 1 0	1	0 1 1 1	

← Answer = -42 →

Case4: (-7) x (-6)

M: Multiplicand = -7 → 1001; 2's Comp → 0111 (For A-M)
Q: Multiplier = -6 → 1010

Step	Operation	(A) Accumulator	(Q) Multiplier	Q _n	(M) Multiplicand	Count
	Initialize	0 0 0 0	1 0 1 0	0	1 0 0 1	
1	Right-Shift	0 0 0 0	0 1 0 1	0	1 0 0 1	
2	A ← A - M	0 1 1 1	0 1 0 1	0	1 0 0 1	
	Right-Shift	0 0 1 1	1 0 1 0	1	1 0 0 1	
3	A ← A + M	1 1 0 0	1 0 1 0	1	1 0 0 1	
	Right-Shift	1 1 1 0	0 1 0 1	0	1 0 0 1	
4	A ← A - M	0 1 0 1	0 1 0 1	0	1 0 0 1	
	Right-Shift	0 0 1 0	1 0 1 0	1	1 0 0 1	

← Answer = 42 →

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

May 2004 (Comp/IT): Multiply (-7) x (14) using Booth's Algorithm

M: Multiplicand = -7 → 1001; 2's Comp → 0111
Q: Multiplier = 14 → 01110

Step	Operation	(A) Accumulator	(Q) Multiplier	Q ₋₁	(M) Multiplicand
	Initialize	0 0 0 0	0 1 1 1 0	0	1 0 0 1
1	Right-Shift	0 0 0 0	0 0 1 1 1	0	1 0 0 1
2	A ← A - M	0 1 1 1	0 0 1 1 1	0	1 0 0 1
	Right-Shift	0 0 1 1	1 0 0 1 1	1	1 0 0 1
3	Right-Shift	0 0 0 1	1 1 0 0 1	1	1 0 0 1
4	Right-Shift	0 0 0 0	1 1 1 0 0	1	1 0 0 1
5	A ← A + M	1 0 0 1	1 1 1 0 0	1	1 0 0 1
	Right-Shift	1 1 0 0	1 1 1 1 0	0	1 0 0 1

← Answer = -98 →

#For more, easy-to-understand examples, refer Bharat Sir's lecture notes.

11001110
001100001
71

1100010
1000001

69

25

98

DIVISION:

Division is performed by continuously subtracting the Divisor from the Dividend till the Dividend becomes lesser than the Divisor or becomes 0. Whatever remains of the Dividend is called as the Remainder and the number of times the subtraction takes place is the Quotient.

Circuit:

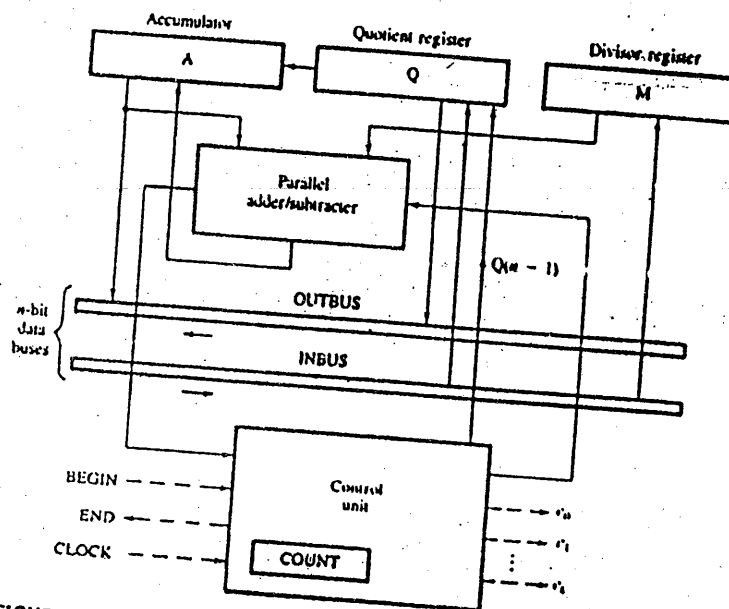


FIGURE 3.53
Block diagram of a sequential n -bit binary divider.

Steps:

- 1) The circuit positions the Divisor approximately w.r.t. the Dividend. Subtraction is performed (Dividend - Divisor).
- 2) If the remainder is 0 or +ve:
 - The Quotient bit is "1".
 - Remainder is extended by another bit of the dividend.
 - Divisor is repositioned for another subtraction.
 - This is called as a successful subtraction and restoring is not required.
- 3) Else (remainder is -ve):
 - The Quotient bit is "0".
 - Since the subtraction was unsuccessful, restoring will be required.
 - The dividend is **restored by adding back** the divisor.
 - Divisor is repositioned for another subtraction.
- 4) Steps 2 and 3 are performed till all bits of the dividend are used.

Given : $B \leftarrow \text{Divisor}; \quad Q \leftarrow \text{Dividend}$
Result : $A \leftarrow \text{Remainder}; \quad Q \leftarrow \text{Quotient}$

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

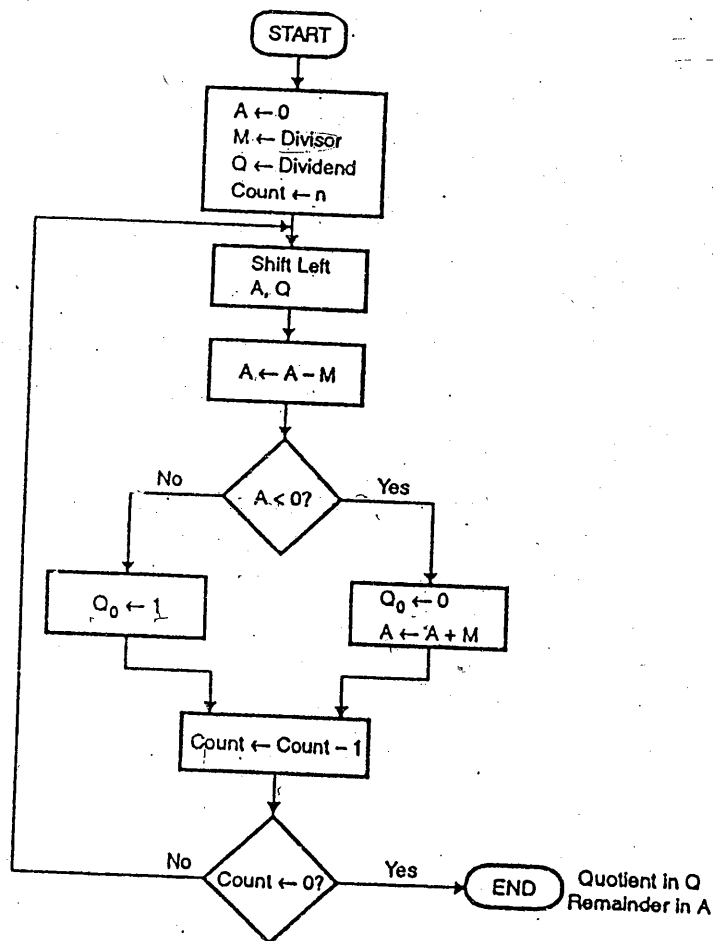
There are 2 ways of performing Division:

1) Restoring Method

Here, the dividend is restored after each unsuccessful subtraction operation.

- Shift A and Q by left by 1 position.
- Perform $A \leftarrow A - B$
- If sign bit of A = 1: (unsuccessful)
 - i. **Restore A as:** $A \leftarrow A + B$.
 - ii. $Q_0 = 0$
- If sign bit of A = 0: (successful)
 - i. $Q_0 = 1$
- Repeat above steps till all bits of the dividend are used.

Flowchart:



CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Example: $10/4 = 2(\text{Quotient}), 2(\text{Remainder})$

A = 0;

Q = Dividend = 10; (1010)

B = Divisor = 4; (0100) ... 2's Complement = 11100.

	<u>A Register</u>	<u>Q Register</u>	Dividend
Initially	0 0 0 0 0	1 0 1 0	
Shift	0 0 0 0 1	0 1 0	First Cycle
Subtract B	+ 1 1 1 0 0		
(-ve hence $Q_0=0$)	① 1 1 0 1		
Restore (A + B)	+ 0 0 1 0 0		
	0 0 0 0 1	0 1 0 0	
Shift	0 0 0 1 0	1 0 0	Second
Subtract B	+ 1 1 1 0 0		
(-ve hence $Q_0=0$)	① 1 1 1 0		
Restore (A + B)	+ 0 0 1 0 0		
	0 0 0 1 0	1 0 0 0	
Shift	0 0 1 0 1	0 0 0	Third
Subtract B	+ 1 1 1 0 0		
(+ve hence $Q_0=1$)	① 0 0 0 1		
No Need to restore	0 0 0 0 1	0 0 0 1	
Shift	0 0 0 1 0	0 0 1	Fourth
Subtract B	+ 1 1 1 0 0		
(-ve hence $Q_0=0$)	① 1 1 1 0		
Restore (A + B)	+ 0 0 1 0 0		
	0 0 0 1 0	0 0 1 0	
	<u>Remainder</u>	<u>Quotient</u>	

$448 \div 7$

$448 \rightarrow \text{Dividend}$

1110000000

2/14

10
4

0000

1110000000

10001

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

2) Non-Restoring Method

Here, the dividend is NOT restored after each unsuccessful subtraction operation. Instead, the following logic is followed:

If the current remainder is +ve then...

$Q_0 = 1;$

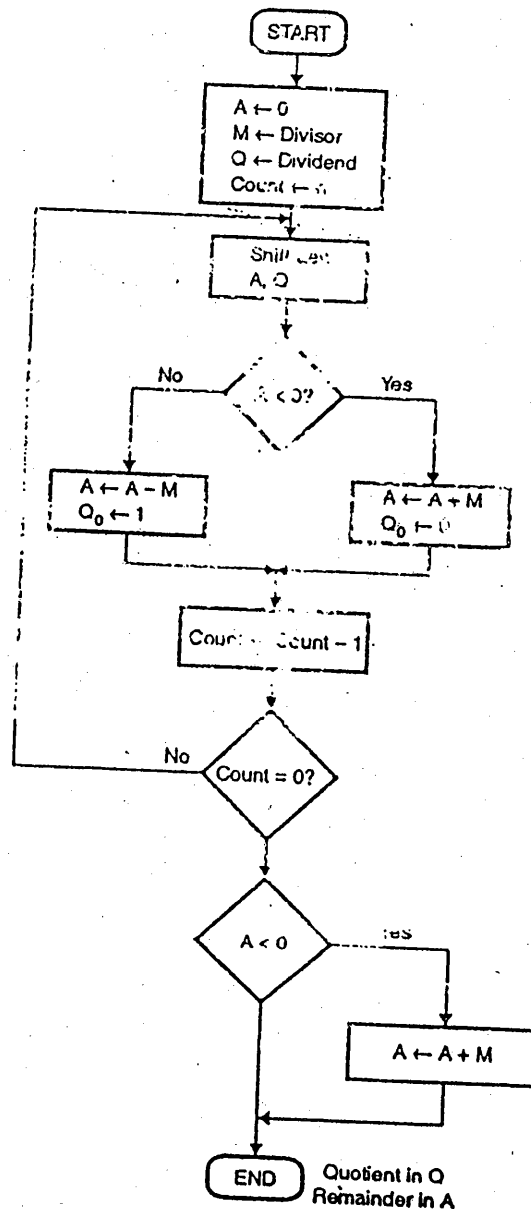
Next operation will be shift and **subtract**;

Else (remainder -ve), then...

$Q_0 = 0;$

Next operation will be shift and **add**.

Flowchart:



CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Example: $10/2 = 2(\text{Quotient}), 2(\text{Remainder})$.

A = 0;

Q = Dividend = 10; (1010)

B = Divisor = 4; (0100) ... 2's Complement = 11100.

Initially	A Register 0 0 0 0 0	Q Register 1 0 1 0	Dividend
Shift	0 0 0 0 1	0 1 0 <input type="checkbox"/>	First Cycle
Subtract B	+ 1 1 1 0 0	0 1 0 <input type="checkbox"/>	
-ve: $Q_0=0$; next ADD	① 1 1 0 1	0 1 0 0	
Shift	1 1 0 1 0	1 0 <input type="checkbox"/> <input type="checkbox"/>	Second
Add B	+ 0 0 1 0 0	1 0 <input type="checkbox"/> <input type="checkbox"/>	
-ve: $Q_0=0$; next ADD	① 1 1 1 0	1 0 0 0	
Shift	1 1 1 0 1	0 <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	Third
Add B	+ 0 0 1 0 0	0 <input type="checkbox"/> <input type="checkbox"/> <input type="checkbox"/>	
+ve: $Q_0=1$; next SUB	① 0 0 0 1	0 0 0 1	
Shift	0 0 0 1 0	0 <input type="checkbox"/> <input type="checkbox"/> 1 <input type="checkbox"/>	Fourth
Subtract B	+ 1 1 1 0 0	0 <input type="checkbox"/> <input type="checkbox"/> 1 <input type="checkbox"/>	
(-ve hence $Q_0=0$)	① 1 1 1 0	0 0 1 0	
Restore the final remainder (A + B)	+ 0 0 1 0 0		
	0 0 0 1 0		
	<u>Remainder</u>	<u>Quotient</u>	

Thus in Non-Restoring division, the partial remainder need not be restored after each unsuccessful subtraction. This reduces the number of steps and hence makes the algorithm faster.

ALU DESIGN

ALU - ARITHMETIC LOGIC UNIT:

The various circuits required to perform arithmetic and logic operations are merged into a single unit called the ALU.
It can perform operations such as add, sub, mul, div, and or etc.

Basic ALU Organization:

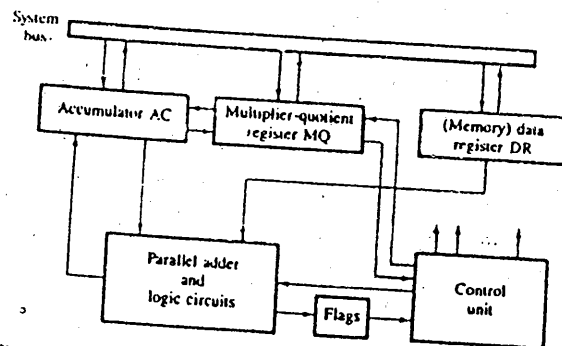


FIGURE 3.59
Structure of a basic fixed-point ALU.

AC, MQ and DR are all registers used to store data.

AC – Accumulator

MQ – Multiplier Quotient

DR – Data Register

AC and MQ can be used as a single register AC.MQ capable of left-right shifting.

These registers are typically used as shown:

ADD: $AC \leftarrow AC + DR$

SUB: $AC \leftarrow AC - DR$

MUL: $AC.MQ \leftarrow MQ \times DR$

DIV: $AC.MQ \leftarrow MQ \div DR$

AND: $AC \leftarrow AC \wedge DR$

OR: $AC \leftarrow AC \vee DR$

XOR: $AC \leftarrow AC \oplus DR$

BIT-SLICED ALU:

- During the development stages, it was realized that instead of making a single ALU to handle large numbers, it would be better to use a set of smaller ALUs. This would not only make the system very **flexible**, but also make it **easy to maintain and upgrade**.
 - Hence if we design a small **ALU** to process **m-bits**, and then **combine k such ALUs**, we could directly **process (k x m) bits**.
 - Such a combinational circuit is called as a "**Bit-Sliced ALU**".
 - Here each individual chip processes an **m-bit "slice"**, of the total (K x m) bits.
 - The immediate advantage of Bit-Sliced ALU is that **any desired word size** can be processed using a no of such slices.
- Eg: A **16-bit ALU** can be constructed using **4 slices of 4-bit ALUs**.
- The **data** to be operated is **split up** and given to each slice, while a **common control signal** is given to all the slices to perform the operation on the data.
 - The **result** of all the slices is **combined** to produce the final result.
 - A block diagram of such an ALU is as shown:

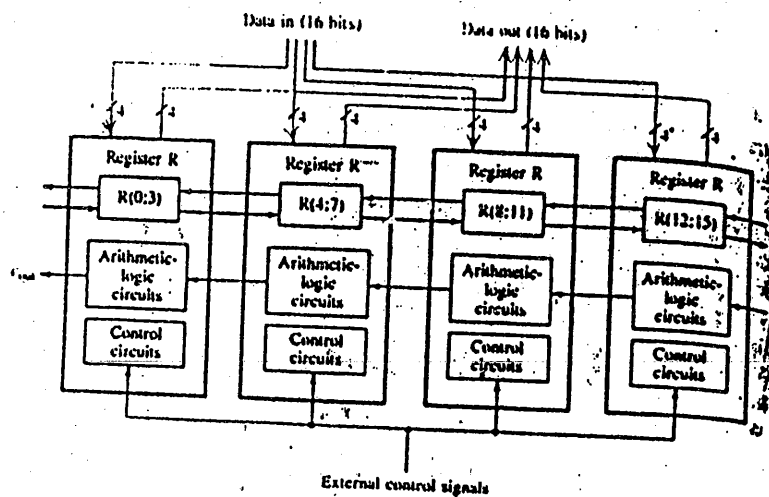
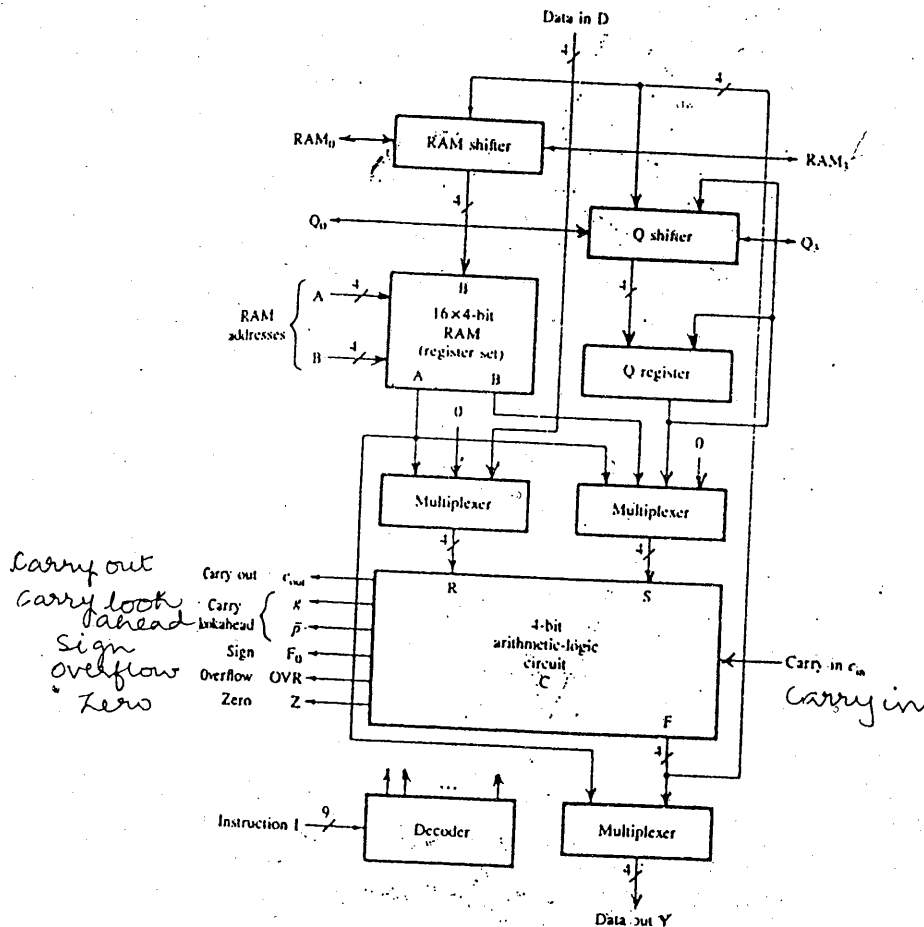


FIGURE 3.60
A 16-bit bit-sliced ALU composed of four 4-bit slices.

AMD 2901 Bit-Sliced ALU

AMD 2901 is a powerful and widely used 4-bit slice ALU. Its organization is as follows:

Structure:



Description:

- 1) AMD 2901 can perform **4-bit operations**.
- 2) The operation to be performed is indicated by a **9-bit instruction**.
- 3) Operands are provided by **16 RAM registers**, each 4-bit. Additionally a **Q register** is also provided to act as a Multiplier/Quotient register.
- 4) A pair of **shift registers** (RAM shifter and Q shifter) is also used for shifting during **multiply and divide** operations.
- 5) A combinational circuit **C**, performs the following 4-bit arithmetic and logic operations.
Arithmetic: ADD, SUB and 2's Complement.
Logic: AND, OR, XOR, XNOR and Shift.
- 6) The circuit C accepts **inputs** from the **RAM**, **Q**, **external data bus** or an all **zero** input.
- 7) The RAM register to be used (RAM 0 ... RAM 15) is decided by the input A and B.
- 8) The **result** generated by C can be stored internally in the **RAM** or in **Q** register; or can be sent out using the **external data bus Y**.
- 9) The **9-bit instruction I** has **3 fields** of 3 bits each: **I_s** (Source), **I_F** (Function) and **I_D** (Dest).

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Instruction $I = I_s I_F I_D$.

Source: $I_s = I_0 I_1 I_2$: This decides the 2 inputs to the ALU.

$I_2 I_1 I_0$	Input R	Input S
0 0 0	RAM (A)	Q
0 0 1	RAM (A)	RAM (B)
0 1 0	0	Q
0 1 1	0	RAM (B)
1 0 0	0	RAM (A)
1 0 1	D	RAM (A)
1 1 0	D	Q
1 1 1	D	0

Function: $I_F = I_3 I_4 I_5$: This decides the operation to be performed by the ALU.

$I_3 I_4 I_5$	ALU Function
0 0 0	$R + S + C_{in}$
0 0 1	$S - R - C_{in}$
0 1 0	$R - S - C_{in}$
0 1 1	R OR S
1 0 0	R AND S
1 0 1	R AND S
1 1 0	$R \oplus S$
1 1 1	$R \oplus S$

Destination: $I_D = I_6 I_7 I_8$: This decides the where the result has to be stored.

$I_6 I_7 I_8$	Y	RAM (B)	Q
0 0 0	F	-	F
0 0 1	F	-	-
0 1 0	RAM (A)	F	-
0 1 1	F	F	-
1 0 0	F	F/2	Q/2
1 0 1	F	F/2	-
1 1 0	F	2xF	2xQ
1 1 1	F	2xF	-

10) For Carry Look Ahead Adders, 2901 also produces g, (generate), and p (propagate).

11) For Ripple Carry Adders, 2901 uses C_{in} and C_{out} .

12) Additionally 2901 also provides the various Flag signals:

OVR - Overflow: This bit is set (1), when the current operation causes an overflow.

F₀ - Sign: This bit is set (1), to indicate that the result is -ve (MSB of result = 1).

Z - Zero: This bit is set (1), when all bits of the result are zero.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

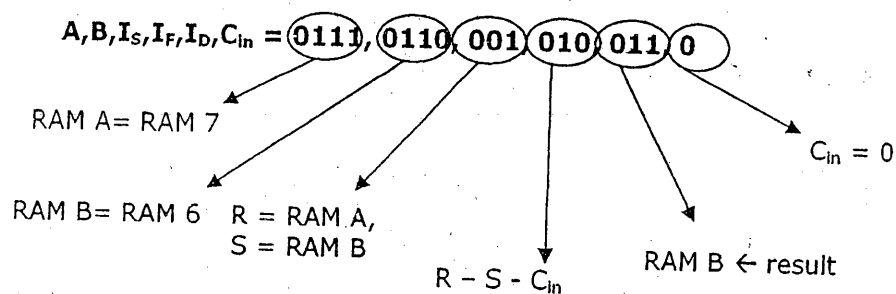
Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

The complete instruction format is:

$A, B, I_S, I_F, I_D, C_{in}$

- A (4-bit): Specifies the RAM register to be used (0 ... 15) as the first operand
B (4-bit): Specifies the RAM register to be used (0 ... 15) as the second operand
 C_{in} (1-bit): Previous carry input.

Eg: The operation: $RAM(6) \leftarrow RAM(7) - RAM(6)$
is specified by the following micro-instruction:



#Refer more examples from Bharat Sir's lecture notes.

16-bit ALU using 4 - 2901 Slice ALUs:

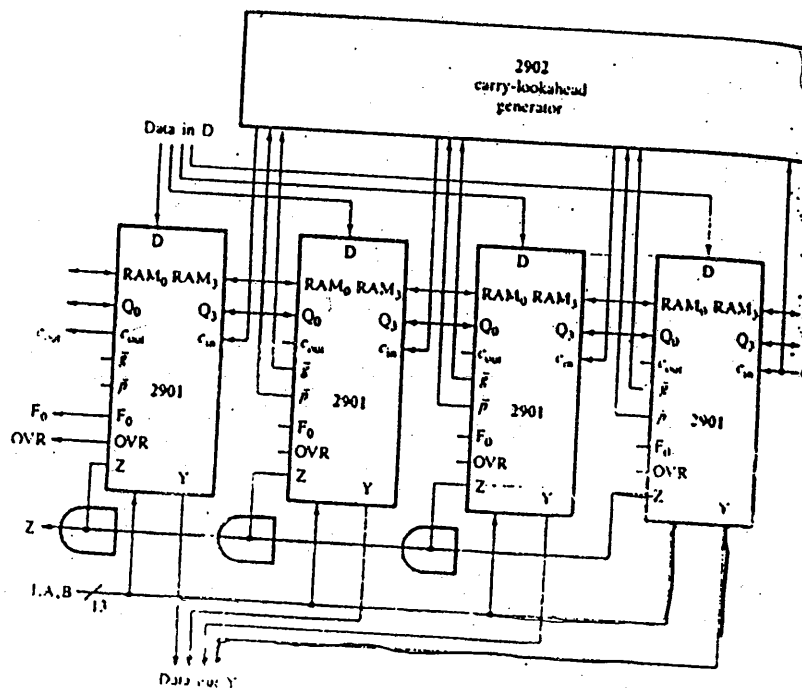


FIGURE 3.62
A 16-bit 4-slice array of 2901s employing carry lookahead.

- The above circuit forms a **16-bit ALU using 4 2901's**.
- For fast calculations AMD has designed the **IC 2902**, which is a **Carry Look Ahead** generator. It accepts the g and p from each slice and produces the C_{in} for the next slice.
- The **Z** o/p of all the slices is passed are **ANDed** to produce the **final Zero Flag**. This is because an AND gate will produce a 1 only when all i/ps are 1, i.e. the answer will be zero **only when all slices have a zero result**.
- The **RAM** and **Q** shifters are used to shift data among the various 2901's (for Mul and Div).
- A **common Instruction "I"** is provided to all 2901s so that all do the same task.
- **OVR** is considered only from the **final 2901**, as the result will overflow only when the MSB 2901 has overflowed.
- Also, the **F0 (Sign) bit** is considered only from the **final 2901**, as it contains the **MSB** of the final 16-bit result.

FLOATING POINT ARITHMETIC

While performing arithmetic operations, a Floating point number is treated as two fixed point numbers: Exponent and Mantissa.

Consider X and Y as two Floating point numbers.

$$X = X_m \cdot 2^{X_e}$$

$$Y = Y_m \cdot 2^{Y_e}$$

Then the arithmetic operations on these two numbers will be performed as:

OPERATION	MANTISSA	EXPONENT
Addition	Add	Equalize
Subtraction	Sub	Equalize
Multiplication	Mul	Add
Division	Div	Sub

From the above table it is evident that Mantissa and Exponent are dealt with, in different ways. Hence a Floating Point ALU has two units internally: Exponent Unit and Mantissa Unit.

Diagram:

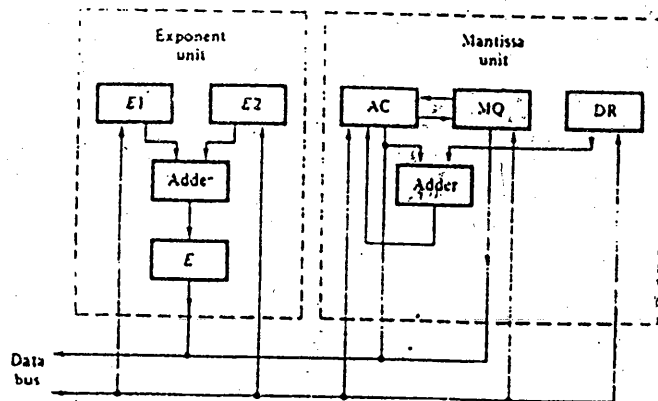


FIGURE 3.67

Data processing part of a simple floating-point arithmetic unit.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra : 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Multiplication and Division are fairly simple operations for Floating Point numbers.
In Multiplication, the Mantissas are multiplied and the Exponents are added.
In Division, the Mantissas are divided and the Exponents are subtracted.

For Addition the **Exponents** need to be **equalized** only then the **Mantissas** can be added.
The same applied for subtraction.

General Steps for Floating Point Addition/Subtraction:

Step 1> **Equalize the Exponent**

Select the smaller of the two exponents.

Right-Shift its Mantissa (thereby increase its exponent), till both exponents are same.

Step 2> **Now Add the Mantissas.**

The larger of the two given exponents will be the exponent of the result.

Step 3> **Normalize the result.**

#Refer easy-to-understand example from Bharat Sir's lecture notes.

Exp Unit :-

Its E1 & E2 registers are initialised with exponents x_e & y_e resp. On these exponents only operations expected are addition & subtraction (No complex mul & div). The result of comparison including sign are then hold by Ereg. So that it can be available on sys bus for mantissa unit. The mantissa unit does the result to understand which exponent is smaller & accordingly mantissa is initialised to AE register. for exp equalisation

Mantissa Unit :- It is a generic sequential arithmetic unit which performs basic arithmetic operations on mantissa of float. It has its AE & MReg are responsible for performing operations like exp equalisation, result normalisation. Final normalised result is available on system bus from AE & MReg so that it can

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Floating Point Addition Algorithm: {Imp. Write complete algorithm to get full marks}

LOAD: $E_1 \leftarrow X_E, E_2 \leftarrow Y_E; \dots$ {Exponents}
 $AC \leftarrow X_M, DR \leftarrow Y_M; \dots$ {Mantissas}
 $Error \leftarrow 0, AC_Overflow \leftarrow 0; \dots$ {Error Variables}

{Compare and Equalize}

COMPARE: $E \leftarrow E_1 - E_2;$

EQUALIZE: If $(E < 0)$ then
 $AC \leftarrow \text{right-shift}(AC);$
 $E \leftarrow E + 1;$
go to Equalize;

Else

If $(E > 0)$ then
 $DR \leftarrow \text{right-shift}(DR);$
 $E \leftarrow E - 1;$
go to Equalize;

{Add the mantissas}

ADD: $AC \leftarrow AC + DR;$
 $E \leftarrow \text{Max}(E_1, E_2);$

{Adjust for overflow}

OVERFLOW: If $(AC_Overflow = 1)$ then
If $(E = E_{\max})$ then go to ERROR;
 $AC \leftarrow \text{right-shift}(AC);$
 $E \leftarrow E + 1;$
go to END;

{Adjust for Zero result}

ZERO: If $(AC = 0)$ then
 $E \leftarrow 0;$

{Normalize the result}

NORMALIZE: If AC is normalized then
go to END;

UNDERFLOW: If $E > E_{\min}$ then
 $AC \leftarrow \text{left-shift}(AC);$
 $E \leftarrow E - 1;$
go to Normalize;

{Set error flag}

ERROR: Error $\leftarrow 1;$

{End the program}

END: End of process.

INSTRUCTIONS & CONTROL UNIT DESIGN

An instruction defines a task to be performed by the processor.
Instructions are stored in the memory.

Instruction Cycle: The sequence of events required to fetch and execute an instruction is called an Instruction Cycle. It is composed of Fetch Cycle and Execution Cycle.

Fetch Cycle: It is the process of obtaining the instruction from the memory.

Execution Cycle: It is the process of decoding and executing the fetched instruction.
During execution Operands may be transferred to or from the memory.

Inside the processor (CPU), there are various registers which are used for fetching and executing instructions. They are of two types: Visible and Invisible to the programmer.

Accessible (Visible) Registers:

I. General Purpose Registers (GPRs):

As the name suggests, these registers are used to store general data during the program. They can be changed by arithmetic operations using the ALU.

II. Address Registers:

These registers are used to hold **memory addresses**.

For higher processors, where segmentation is used, there are separate registers to hold **segment address** and **offset address**. Additionally, there is an **SP (Stack Pointer)** register. It holds the **address of top of stack**. A stack is a set of memory locations working in the LIFO manner.

III. Status Register (Flag Register):

This register contains various flag bits, which indicate status of the result after the current ALU operation. These bits are changed by the ALU. Different processors support different set of flags, but the most commonly used ones are Sign flag, Zero flag, Carry flag etc.

8086

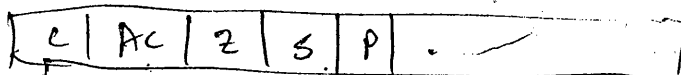
AX AX
BX BX
CX CL
DX DL

SI
DI
SP
IP

8085

A, B, C, D

Status Register



User Inaccessible (Invisible) Registers:

I. PC (Program Counter):

It contains the address of the next instruction to be fetched.
After every instruction is fetched, value of PC is incremented by the processor.
The programmer cannot change PC directly.
PC gets a new value when a branch instruction is encountered.

II. Instruction Register:

It is used to temporarily hold the newly fetched instruction for decoding.

III. MAR (Memory Address Register):

It contains the address of the current memory transfer.

IV. MBR (Memory Buffer Register):

It contains the data that is currently transferred.

Also called as Memory Data R

INSTRUCTION SET:

Characteristics of an Instruction Set:

Any instruction set should have the following features/characteristics:

- i. **Complete**
It should be able to perform all the required operations, only then it is called complete.
- ii. **Efficient**
All the operation should be executed within the smallest amount of time.
Classic examples of this are the "String Instructions" of 8086.
Here instead of making the programmer write a big program for transferring a block of data, a single string instruction does the same job. Thus the operation becomes more efficient.
- iii. **Regular**
It should contain all the general instructions, which are expected in any processor.
Eg: Add, Sub, Mov etc.
- iv. **Compatible**
The instruction set should be designed in such a way that it is up-ward compatible, for further development. Such compatibility is very useful when the system is upgraded to a higher processor.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

• Types of Instructions:

i. Data Transfer Instructions

These instructions transfer data from one location/register to another.

The most important point here is that, there is no arithmetic or logical operation performed on the data. Hence the Flags are not affected as the ALU is not used.

Eg: MOV A, B → 8085 ... register to register
LDA 2000 → 8085 ... register to memory

ii. Data Processing Instructions

These perform Arithmetic or Logic operations on the data.
Hence the Flags are affected.

Eg: ADD AL, BL → 8086 ... register + register
ADD AL, 25 → 8086 ... register + number

iii. Control Instructions

These control the flow of the program, and are hence also called branch control instructions.
Their main job is to change the value of PC, thereby causing a branch.

Eg; JMP 2000 → 8085 ... Go to location 2000 for the next instruction (PC ← 2000).

• Instruction Format

Every instruction has an Opcode. Additionally, it may have 1 or more operands.

Opcode indicates the operation to be performed by the instruction. Every instruction has a unique Opcode.

Opcode	Operand 1	Operand 2	...	Operand n
--------	-----------	-----------	-----	-----------

i. Instructions having three operands

These instructions **specify both** the source **operands** as well as the **result** (dest.).

They are very **flexible**, but are very long and hence **occupy more memory**.

They are available in processors like **CDC 6600**.

Format: ADD Z, X, Y → Z gets the result after adding X and Y.

ii. Instructions having two operands

These instructions **specify both** the source **operands**.

The **result** is implied to be one of the two operands, generally the first one.

They are less **flexible**, but **occupy lesser memory**.

They are available in processors like **Intel 8086**.

Format: ADD AL, BL → AL gets the result after adding AL and BL.

iii. Instructions having one operand

These instructions **specify only one operand**.

A default register acts as second operand, and also stores the result. (Accumulator)

They are less **flexible**, but **occupy lesser memory**.

They are available in processors like **Intel 8085**.

Format: ADD B → A gets the result after adding A and B.

Hence, as the number of operands increases:

- Flexibility for the programmer increases
- Processor design becomes more complex
- Size of instruction increases

Sp. Case: Zero Operand Instructions

These are used in machines implementing **stacks**.

A Stack is an organized form of memory that works in **last in first out** manner i.e. Reverse Polish Form.

Here once we Push (insert) the operands on the top of stack; we can directly use the stack for providing operands. There is no need to provide the address as the stack can **only be used from the top**. This **eliminates the need of any operand** in the instruction. Such instructions are available in **Intel 8087 Co-Processor**.

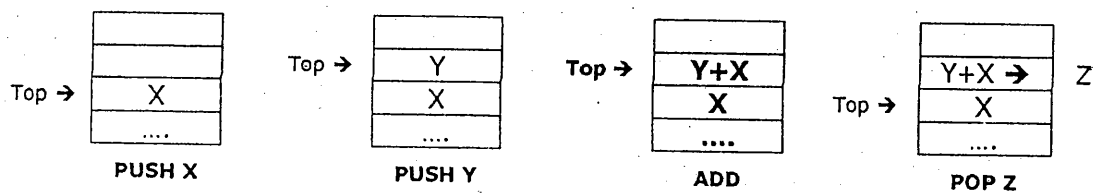
Eg: PUSH X

PUSH Y

ADD

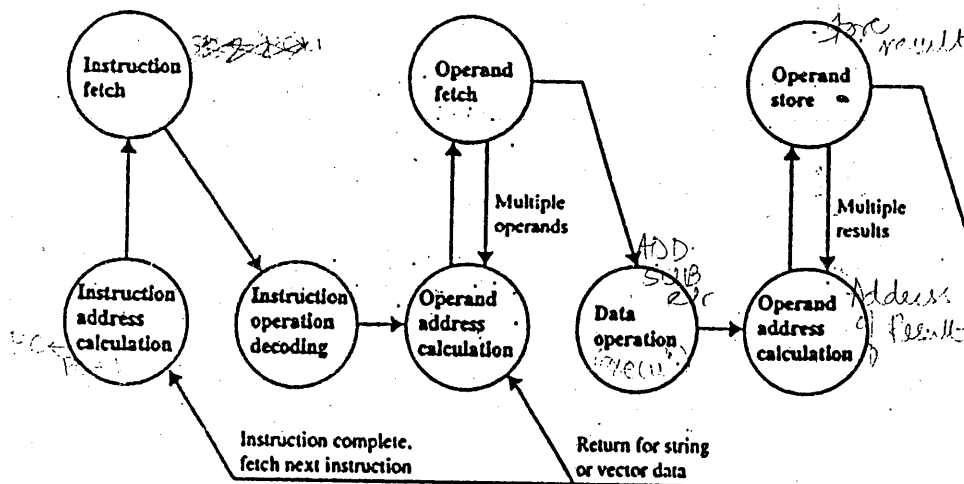
POPZ

; there are no operands but this will add the top two elements of the stack



{Notice that automatically the top two elements of the stack are added}

Instruction cycle state transition diagram



Chain must be followed & should be
but not executed randomly.

ADDRESSING MODES:

Definition: The manner in which operands are fetched for an instruction.
Different processors implement different types of addressing modes based on their level of complexity. Basically there are 5 addressing modes.

1) Immediate Addressing Mode

In this mode, the **Operand** is specified in the **Instruction** itself. *Data in Inst.*
Thus when the instruction is fetched, the operand comes along with it, so the processor can act on it immediately.

Eg: **MVI A, 35H** ; Move immediately the value 35H into the Accumulator.
; i.e. $A \leftarrow 35H$
ADI 25H ; Add immediately the value 25H to the register A.
; i.e. $A \leftarrow A + 25H$

Advantage: Programmer can easily **identify** the **operands**.

Disadvantage: Since operands are readily available, execution is fast.
Always more than one byte hence requires **more space**.
Since instructions are long, fetching them takes more time.

2) Register Addressing Mode

In this mode, the **Operand** is specified in **Registers**. *Data in Reg.*
These instructions use the registers present inside the processor to supply the data.

Eg: **MOV B, C** ; Move the Contents of C-Register into B-Register.
; i.e. $B \leftarrow C$
ADD B ; Adds the value of B register to the value of A register.
; i.e. $A \leftarrow A + B$

Advantage: Since these instructions are small (often 1 byte), **fetching is fast**.

Disadvantage: Also, as the data is from an internal register, **execution is also fast**.
Operands **cannot** be easily **identified**.

3) Direct Addressing Mode

Address in instruction
In this mode, the **instruction** only **contains** the **address of the operand**.
The processor will then use this address to fetch the operand from the memory.

Eg: **LDA 2000H** ; Loads the Accumulator with the Contents of Location 2000.
; i.e. $A \leftarrow [2000]$
STA 2000H ; Stores the Contents of the Accumulator at the Location 2000.
; i.e. $[2000] \leftarrow A$

Advantage: The programmer **can identify** the address of the operand.
It is used to operate on data that is already stored in the memory.

Disadvantage: As instructions contain address, they are long and **fetching is slow**.
After fetching the instruction, the processor has to fetch the operand from the address given in the instruction. This double work makes **execution slow**.

4) Indirect Addressing Mode

*Address
Data in Registers*

This is the most interesting addressing mode.

Here, the **address** of the operand is given indirectly using **register**.

As the value of the register can be changed, the same instruction can be used to operate on different memory locations in a loop.

Eg: **STAX B** ; Stores the contents of the Accumulator at the location whose address is given by the contents of BC pair.
; i.e. $[[BC]] \leftarrow A$.
; So if contents of BC pair = 4000 i.e. $[BC] = 4000$ then
; $[4000] \leftarrow A$. #Please refer Bharat Sir's Lecture Notes for this ...
Now the same instruction can be used in a loop and by incrementing the value of BC pair inside the loop, we can access a set of memory locations

Advantage: Can be used in a loop to **access a block of memory**.

Disadvantage: Size of the instruction is **smaller** compared to direct addressing.
Requires initialization of the register pair in a prior instruction.

Indirect addressing mode is further sub-divided into various modes in advanced processors.
They are as follows:

- **Base Relative:** Address = Base Register + a Displacement value
- **Base Indexed:** Address = Base Register + an Index Register
- **Base Relative plus Indexed:** Address = Base + Displacement + Index Reg

5) Implied Addressing Mode

In this mode, the **Operand** is **implied by the instruction**.

Here we don't need to mention any operand, as the instruction is designed to work with a particular operand only.

Eg: **STC** ; Sets the Carry Flag in the Flag register.
; $Cy \leftarrow 1$.
CMC ; Complements the Carry Flag in the Flag register.

Advantage: Instructions are generally **only one byte**.

Disadvantage: Programmer **cannot** easily **identify** the value of the operand.

Note: These are the fundamental Addressing Modes.

Additionally there are derived Addressing Modes available in processors such as 8086:

- **Register Relative** (Base relative):
Here address is given as a sum of a Base Register and a Displacement
Eg: $MOV\ CL, [BX + 5]$; Here CL gets the data from the address formed by adding BX register + 5.
- **Base Indexed:**
Here address is given as a sum of a Base Register and an Index Register
Eg: $MOV\ CL, [BX + SI]$; Here CL gets the data from the address formed by adding BX and SI registers.
- **Base relative Plus Indexed:**
Here address is given as a sum of a Base Register and an Index Register and a Displacement
Eg: $MOV\ CL, [BX + SI + 5]$; Here CL gets the data from the address formed by adding BX and SI registers and the value 5.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98261 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

INSTRUCTION PIPELINING

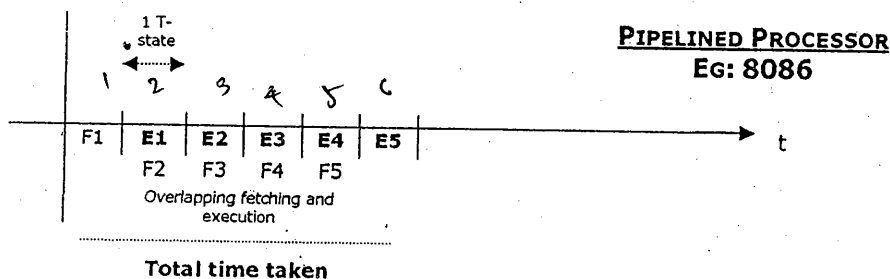
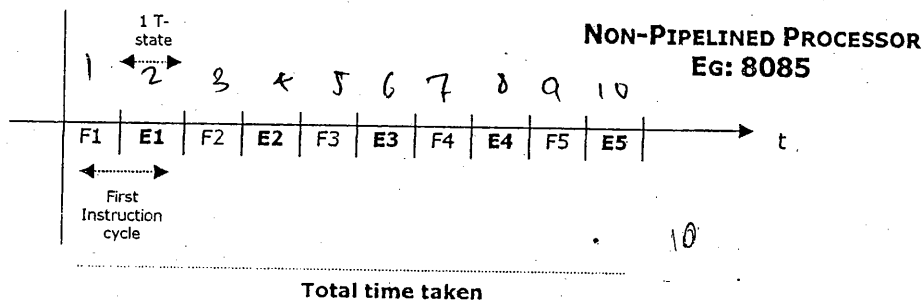


We know that the processor executes a given program instruction by instruction. Each instruction is **first fetched, then decoded and finally executed**. Then the process repeats for further instructions.

Pipeline processors perform the above task in an efficient manner.

Every activity inside a processor takes place on one pulse (tick) of the system clock. **This time duration is called a T-state.**

Let us assume that an instruction requires one T-state for fetching and one T-state for execution. Let five such instructions be in a program. We will now compare how a conventional (non-pipelined) and a pipelined processor execute this program.



Definition: Overlapping the various stages of an instruction cycle is called **instruction pipelining**.

Intel 8086 performs 2-stage pipelining wherein, the two stages: fetching and execution are overlapped. Here, **fetching of the next instruction takes place while executing the current instruction**. This **saves** a lot of time of the processor as shown above.

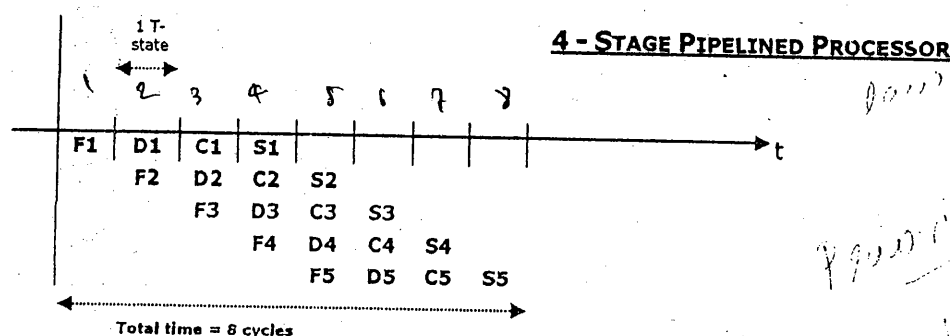
To make the process even more efficient, an instruction cycle can be divided into various stages and all these stages can then be overlapped. This would give a greater degree of pipelining, and thus save more time. This is called **Multi-stage pipelining** and is implemented in higher processors like **Intel 80386 and Pentium**.

Advantage:

- 1) **Saves a lot of time** if used effectively.
If N = no of instructions and K = no of stages that are pipelined, then:
Non-Pipelined processor takes $(K \times N)$ cycles, but a
Pipelined processor only needs $(K + (N-1))$ cycles.
Eg: $N = 5$ instructions,
 $K = 4$ stages of pipelining {Fetch, Decode, Calculate Result, Store Result}
F D C S

Non-Pipelined processor $\rightarrow 20$ cycles.

Pipelined processor $\rightarrow 8$ cycles.



Limitations:

- 1) The biggest problem of pipelining is a **Branch Instruction**.
During execution of an instruction the sequentially **next instruction is pre-fetched**. So if the **2nd instruction** of a program is a **branch to the 9th instruction**, then while executing the 2nd instruction the processor would have **pre-fetched** the 3rd instruction. Now the 3rd instruction should be discarded (**flushed**), as it is not the next instruction to be executed.
- 2) In multistage pipelining, there should not be any **data dependency**.
This means, **the result of the current instruction should not be the operand of the next instruction**, as in such a case, the next instruction cannot be processed until the result of the current instruction is stored. This is also called forward-reference problem.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

RISC v/s CISC DESIGNS

Instructions can be of fixed or variable length.

The more restrictions we put on the size and format of the instruction, lesser is the flexibility to the programmer but the processor design becomes simpler. This gives rise to two types of processor designs CISC, and RISC.

CISC {Complex Instruction Set Computers}:

Here the level of **flexibility** is very **high**. Instructions are of **variable length**. Advanced addressing modes are available to operate on memory operands. **Fewer registers** are required onchip. The processor design is **micro program controlled**. Though the instruction set becomes very powerful, the processor design becomes more **complicated**.

RISC {Reduced Instruction Set Computers}:

RISC on the other hand **reduces the flexibility** but **increases the simplicity** of the processor. Here instructions are of **fixed length**. **Basic Addressing modes** are used, hence instructions are **easily decoded**. More operations are performed using the internal registers, hence **more registers** are required. Design is **hard-wired** and hence executes **faster**.

No.	RISC	CISC
1	Size of the instruction is fixed	Instructions are of variable sizes.
2	Format of instructions is fixed	Format of instructions is variable .
3	Limited number of instructions	Large number of instructions.
4	Instructions are more register oriented	More memory based instructions.
5	Simpler addressing modes	Very advanced addressing modes.
6	More user registers required	Less numbers of registers required.
7	Uses Hard-Wired Control Unit	Uses Microprogrammed Control Unit.
8	One Machine Cycle per instruction	Complex instructions require more cycles .
9	More degree of pipelining	Lesser degree of pipelining
10	Complexity in Compiler design	Complexity in Microprogram design.
Eg:	Intel i860	Intel 80x86 family

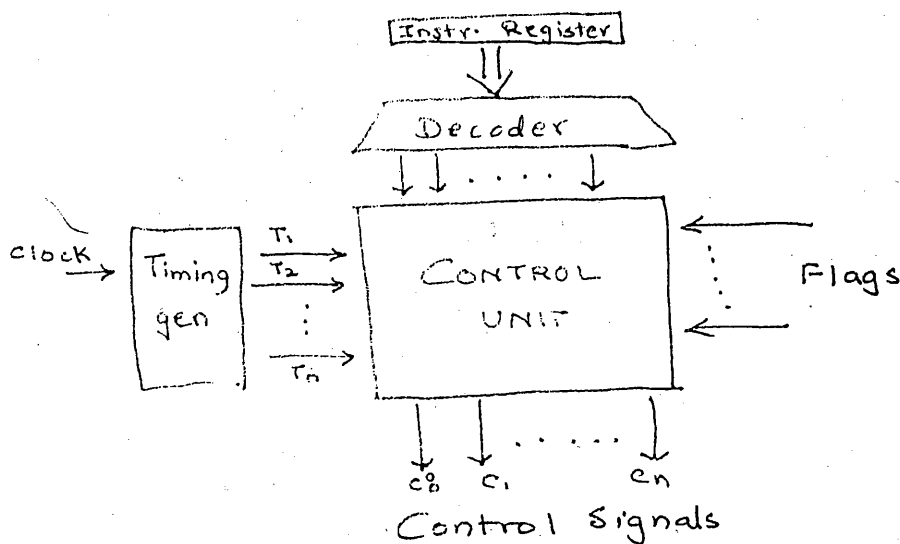
Note: From the above table it is clear the both RISC and CISC have their merits and demerits. Thus in modern day processors, **both the above technologies are combined** for optimum performance. **Eg: Intel Pentium Processors**

ISHAN
SHAW

CONTROL UNIT DESIGN

MICRO-OPERATIONS

Every instruction is further divided into a set of very basic Micro-operations.
A Micro-operation takes one clock pulse i.e. 1 T-state duration.



Micro-operations for a simple Fetch Cycle:

- | | | |
|-----|-------------------------|--|
| T1: | MAR \leftarrow [PC] | ; load the address of the instruction to be fetched |
| T2: | MBR \leftarrow Memory | ; get the instruction from the memory |
| T3: | IR \leftarrow [MBR] | ; load the instruction into the instruction register |
| | PC \leftarrow PC + I | ; increment PC to point to the next instruction |
| | | Here I = Length of the current instruction |

Notice that the last two operations took place in the same cycle. This is possible because **when two or more Micro-operations are independent of each other, they can be performed simultaneously in one clock cycle.**

Another way of performing Instruction Fetch operation:

- | | | |
|-----|-------------------------|---|
| T1: | MAR \leftarrow [PC] | : |
| T2: | MBR \leftarrow Memory | : |
| | PC \leftarrow PC + I | : |
| T3: | IR \leftarrow [MBR] | : |

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Now we will see Micro-operations for entire instructions:

- 1) Give the Micro-instructions for: **ADD A, B** {Dec 99}

Soln:

T1:	MAR	←	[PC]	}	Fetching
T2:	MBR	←	Memory		
T3:	IR	←	[MBR]		
	PC	←	PC + I	}	Execution
T4:	A	←	A + B		

Note that the two operands to be added were **registers already present in the processor**. Hence **no additional steps** were required to fetch them. If an operand is from the memory, then additional steps will be required to fetch it.

Such instructions are used in **Intel 8085**.

- 2) Give the Micro-instructions for: **ADD A, X** {X is a memory address}

Soln:

T1:	MAR	←	[PC]	;load the address	}	Fetching
T2:	MBR	←	Memory	;fetch instruction		
T3:	IR	←	[MBR]	;store instr in IR		
	PC	←	PC + I	;point to next instruction	}	Execution
T4:	MAR	←	IR (address)	;load addr of operand		
T5:	MBR	←	Memory	;fetch opr from mem		
T6:	A	←	[A] + [MBR]	;perform the addition		

Here the instruction contained the address of a memory operand. This address was fetched into the IR during the fetching phase.

During execution first this address from IR is used to fetch the memory operand. This operand is then added to A as required.

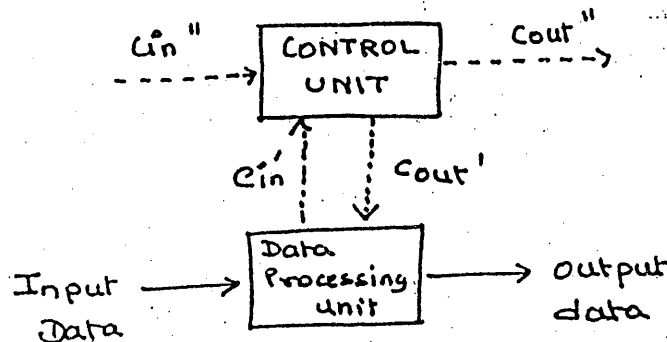
Such instructions are used in **Intel 8086**.

Note1: For SUB A, X → T6: A ← [A] - [MBR]

Note2: If the memory operand is from the stack the Stack Pointer register will give the address. i.e.: T4: MAR ← [SP]

CONTROL UNIT

- All the **Micro-operations** are performed by the **Control Unit**.
- The **main task** of Control Unit is to **generate internal control signals** to all internal registers and also to enable activities on the internal bus of the processor.
- If these control signals are generated using a **combinational logic circuit** then it is called a **Hardwired Control Unit**.
- If these control signals are generated by **software** then it is called a **Microprogrammed Control Unit**.



Signal description:

- Cout'** → This signal **directly controls all data activities** of the processor. The main task of the Control Unit is to generate this signal.
- Cin'** → This provides the **status information** to Control Unit.
Eg: **Flag** information.
- Cin''** → These can be used for **synchronization** like "start" or "stop". They can also be used to accept **Interrupt or DMA requests**.
- Cout''** → These can be used for **synchronization** like "busy" or "free". They can also be used to send **Intr or DMA acknowledgement**.

HARDWIRED CONTROL UNIT

There are three types of Hardwired Control designs:

1) State Table Method

- The **behavior** of a Control Unit is represented in the form of a **state table** as shown.

STATES	Input I_1	I_2	...	I_m
S_1	$S_{1,j}, Z_{1,j}$	$S_{1,2}, Z_{1,2}$...	$S_{1,m}, Z_{1,m}$
S_2	$S_{2,j}, Z_{2,j}$	$S_{2,2}, Z_{2,2}$...	$S_{2,m}, Z_{2,m}$
...
S_n	$S_{n,j}, Z_{n,j}$	$S_{n,2}, Z_{n,2}$...	$S_{n,m}, Z_{n,m}$

Next State

Output Control Signal to be generated $\{C_{out}\}$

- The table shows what corresponding output should be generated when different inputs are applied at various states.
- Let C_{in} and C_{out} denote the input and output variables of the Control Unit.
- Each row in the state table corresponds to an **internal state** $\{S_i\}$ of the Control Unit. A state is determined by the **information stored** in the processor at that unit of time.
- Each column denotes the different set of external **input signals** applied to the control unit $\{C_{in}\}$.
- The **intersection** of the row S_i and column I_j means the following:
 - When Input I_j is applied to state S_i , we get $S_{i,j}, Z_{i,j}$.
 - $Z_{i,j}$ is the set of **output** signals to be activated.
 - $S_{i,j}$ should become the **next state** of the control unit
- A circuit is then constructed based on this table.

Advantage:

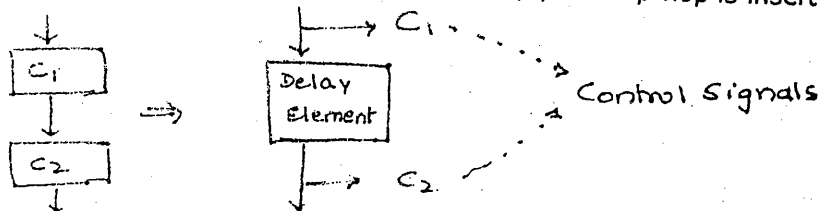
- This is the **simplest method** to implement a Hardwired Control Unit.

Disadvantage:

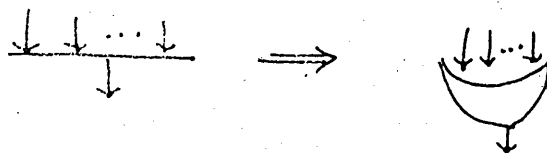
- If the processor has a **large number** of states and input combinations, then this method cannot be used as the **size of the table** will become **too large** and the circuit will become **very difficult** to implement.
- A state table tends to **hide useful information** like the existence of loops and repeated patterns.
- Even if two **different states** have the **same behavior**, we will require **separate hardware** for both as this **information is hidden**.
- Circuits designed using this method tends to have a **random structure** and are thus **difficult to debug and maintain**.

2) Delay Element Method

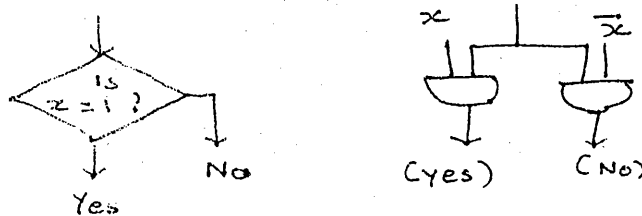
- Here the **behavior** of a Control Unit is represented in the form of a **flowchart**.
- Every step** of the flowchart at time t_i will activate $\{C_{i,j}\}$, where C_i is the **control signal** at **time t_i** , for the execution of the instruction j .
i.e: at t_1 : Activate $\{C_{1,j}\}$
at t_2 : Activate $\{C_{2,j}\}$
at t_n : Activate $\{C_{n,j}\}$
- Once the flowchart is complete, then individual circuits for each $\{C_{i,j}\}$ are formed.
- It is obvious that the instruction j will be executed when all the steps of $\{C_{i,j}\}$ are performed from $C_{1,j}, C_{2,j}, C_{3,j}, \dots, C_{n,j}$.
- But, all these steps should not be performed together; instead there should be a **finite time gap (delay)**, between every two steps.
- The delay in the circuit is introduced by a "D" flip-flop.
Delay time = $t_2 - t_1 = t_3 - t_2 = \text{one clock pulse}$.
- Thus, one after the other, all steps are performed, in the order of the flowchart.
- The different parts of a flowchart are handled as follows:
 - Between Two successive steps simply a D flip-flop is inserted.



- Multiple inputs are combined by an OR gate.
This is because, if we arrive at the gate from any of the inputs, we should proceed ahead. An OR gate will produce a 1 when any of the inputs is 1.



- A Conditional branch (decision box) of the flowchart is implemented by a pair of AND gates as shown.



If $X = 1$: the gate on the left will be active, but
If $X = 0$: the gate on the right will be active.

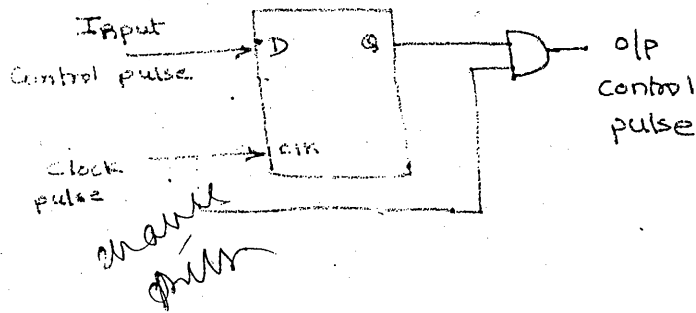
CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

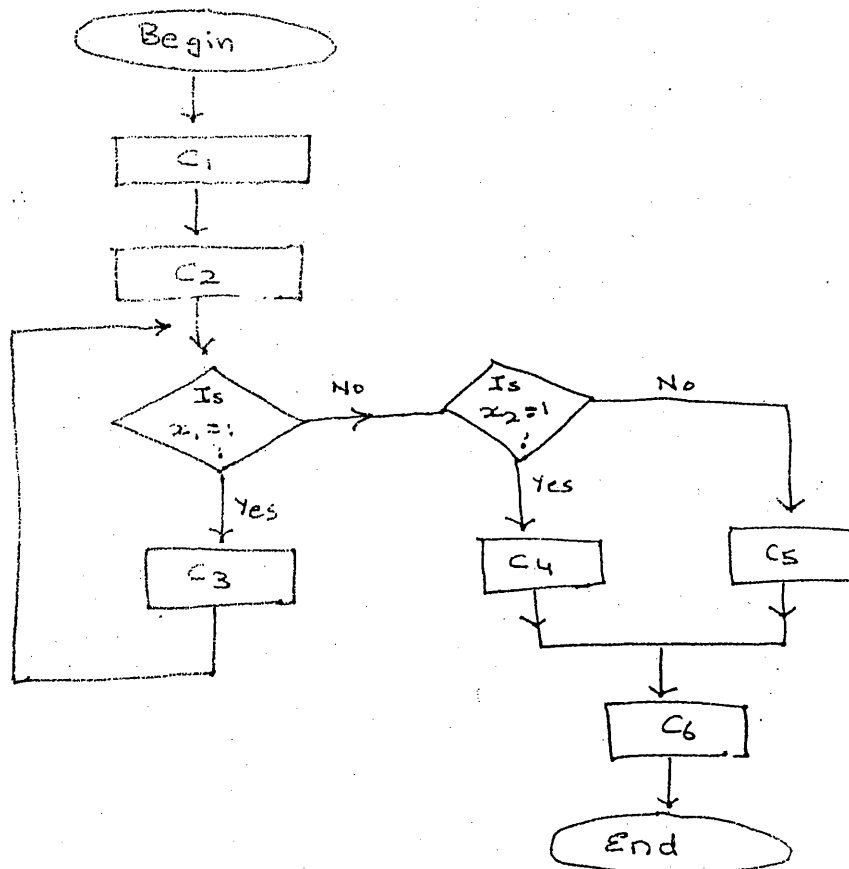
Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Delay element:

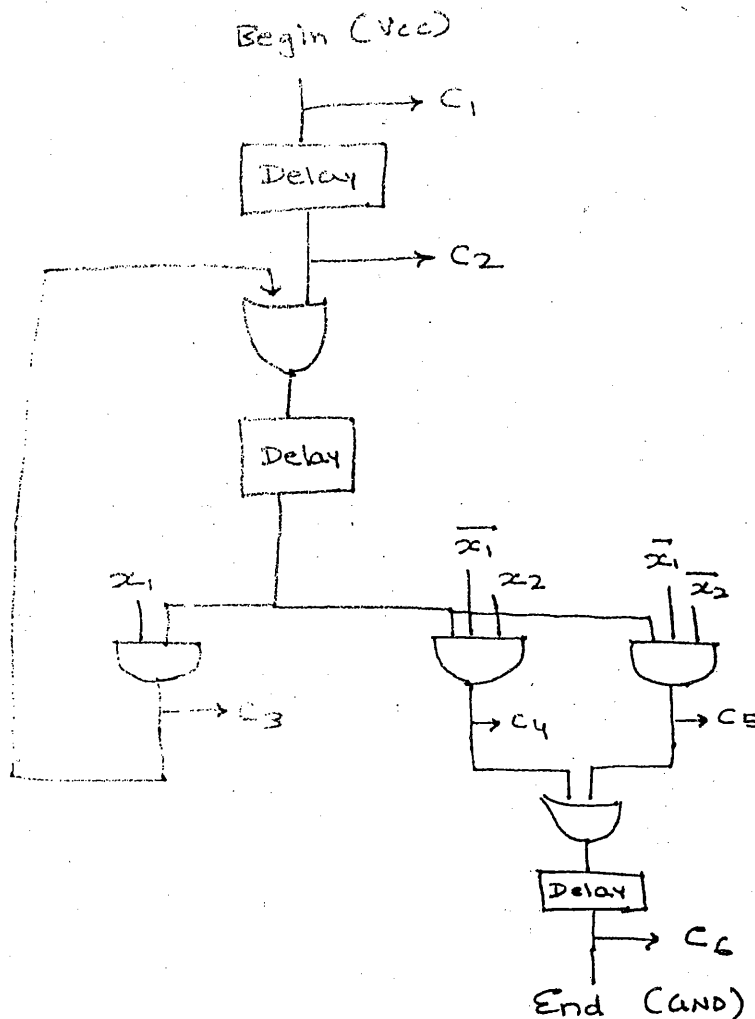


Example:

Consider the following flowchart...



The circuit based on the above flowchart using the Delay element method is as shown:



Advantage:

- In case of **looping programs**, the **same hardware can be used in a loop** as shown in the above example. This is a clear **advantage over state table method** where looping required duplicate hardware. Thus **hardware is reduced**.

Disadvantage:

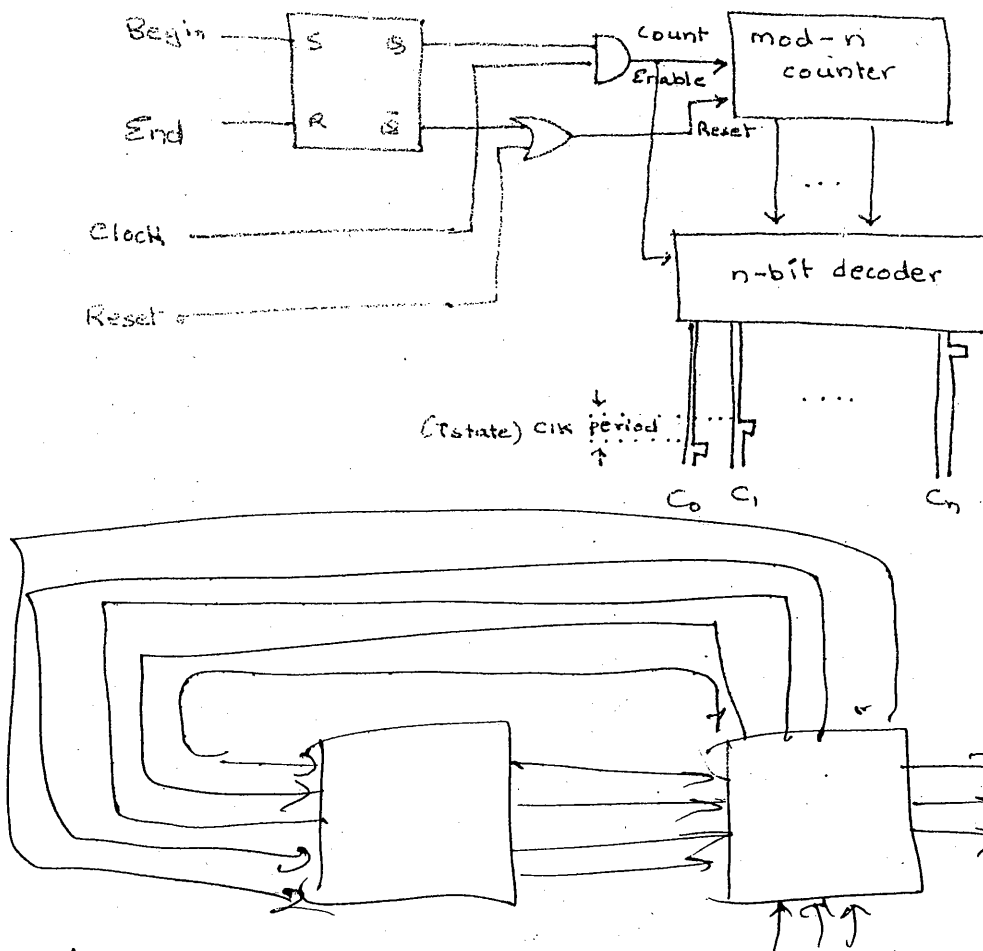
- The **number of delay elements** is approximately **equal to the number of states**. Thus if there are **lot of states**, the **circuit can become too large**.
- Moreover, if there are a **lot of D flip-flops** then the **synchronization of delay** can become **difficult**.

Note: Since out of all the D flip-flops, **only one is activated ("hot")** at a time, this method is also called **"One Hot Method"**.

3) Sequence Counter Method

- This is the **best method** for Hardwired design.
- It is an improved version of the delay element method, where "k" delay elements are replaced by a single **modulo-K counter**.
- The **concept** is very simple:
 - Our objective is that the "k" control signals should be produced one after the other, with a **fixed time interval of one clock cycle** between each of them.
 - For this we had used "k" D flip-flops, each giving one clock pulse delay.
 - Now we use a single **modulo-k counter** and provide the same clock input to it.
 - The **output** of this counter is connected to a **1:k decoder**.
 - Thus we will get the "k" different **outputs** from the decoder, each after one clock pulse, as required.

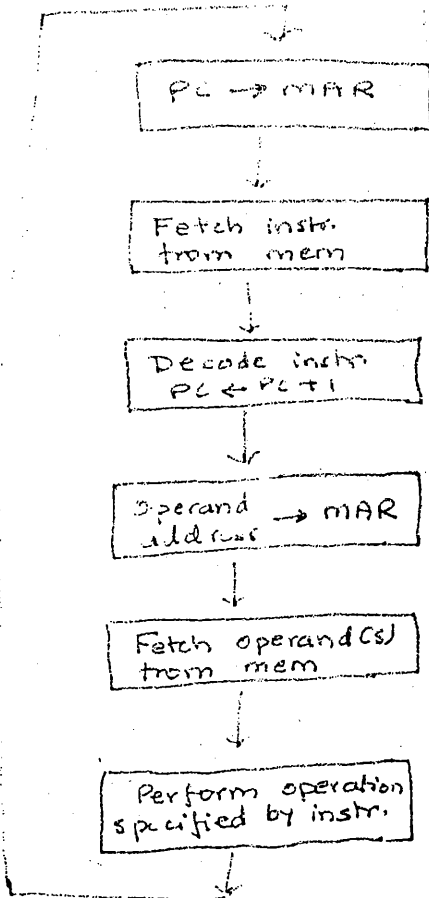
Diagram:



- As seen above, the time gap between each of the "k" control signals produced is of one clock pulse.
- Additionally the begin, End and reset signals are also provided, to control the sequence of counting.

Example:

Consider the closed loop behavior of a typical CPU



- These six steps are performed repeatedly in a loop.
- This can be implemented using a modulo-6 counter.
- To enable repetition, the Q_6 output should be connected to the begin input in the circuit. This will cause the counting to restart each time, after reaching the last state.

Advantage:

- The use of hardware is minimal, and hence it is the best method for large, complex Control Units.

Disadvantage:

- For simple Control Units this method can be expensive due to the use of a sequence counter and a decoder.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

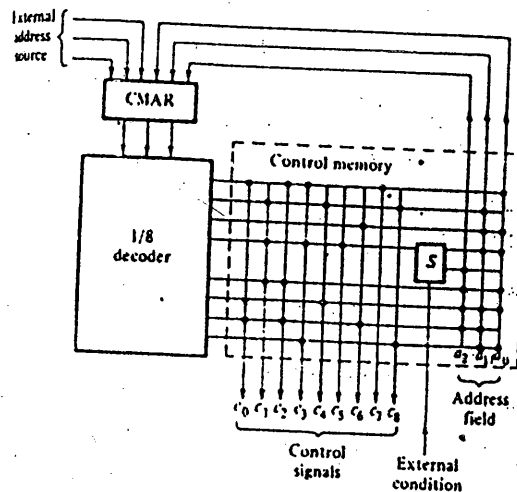
Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

MICROPROGRAMMED CONTROL UNIT



- With large number of instructions, the Hardwired Control Unit becomes very **complicated**.
- Moreover, by nature, the hardwired design is **not flexible**.
Each time a **change** has to be made, the entire Control Unit has to be **re-wired**.
- This gives rise to a new approach for control unit design which was **more software based**. It is called **Microprogrammed Control Unit**.
- The task of a Control Unit is to decide which **control signals** are to be **activated** for a particular micro-operation.
- In Microprogrammed Control Unit, **for every instruction**, there are **Micro-instructions**. These **Micro-instructions indicate the control signals to be activated**.
- Micro-instructions are **stored in RAM** or RAM, called as **Control Memory**.
- A set of related **Micro-instructions** used for a **single machine instruction** is called a **Microprogram**.
- As Micro-instructions lie in the Control Memory, they have to be **fetched** (just like a program). The **address** for the next Micro-instruction is given by the **μPC – Micro Program Counter**.
- Each Micro-instruction specifies the address of the next Micro-instruction either explicitly or implicitly. This is called of **Micro-instruction sequencing**.
- The set of **Microprograms** for all the instructions of a **language** is called as its '**emulator**'.
- The **main advantage** of Microprogrammed Control Unit is that the Micro-instructions can be **changed quite easily**, and hence it is **more flexible**.
- Moreover, one processor can be made to **execute programs of another processor** by simply **loading its emulator** in the Control Memory.
This is done in case of co-processors (like Intel 8087).
- The **disadvantage** is that **time is wasted in fetching** Micro-instructions from the Control Memory, hence this is **slower** than Hardwired Control Unit.

WILKEY'S DESIGN {imp}



- 1) Here the CPU has an on-chip Control Memory.
- 2) The Control memory contains various Micro-instructions, which will specify the control signal to be activated.
- 3) Each Micro-instruction has two parts:
 - **Control Field:** It specifies the control signal to be activated.
 - **Address Field:** It specifies the address of the next Micro-instruction in the Control Mem.
- 4) Each bit of the control field directly corresponds to a control signal. Hence for any Micro-instruction, the bits which are high will be the signals to be generated.
- 5) This allows us to change the control unit very easily (by just changing the bits).
- 6) The starting address of the Microprogram is provided by an external source (as shown), into the CMAR (control memory address register).
- 7) Based on this address, the Micro-instruction is selected.
- 8) Using the Micro-instruction, the corresponding control signals are generated, and also the address of the next Micro-instruction is placed into the CMAR.
- 9) The switch "S", is very useful for conditional instructions.
- 10) For such an instruction, we may have two methods of proceeding ahead depending on whether the condition is true or false. (Eg: JC - Jump if Carry Flag is High).
- 11) Based on the value of the Switch "S", two possible addresses can be loaded into CMAR and the program will proceed according to the current condition.
- 12) In the original design, the Control Memory was implemented using ROM. But this makes the Control Memory rigid. A Control Memory in RAM is more flexible but increases the cost of the system. It is called WCM (Writable Control Memory).

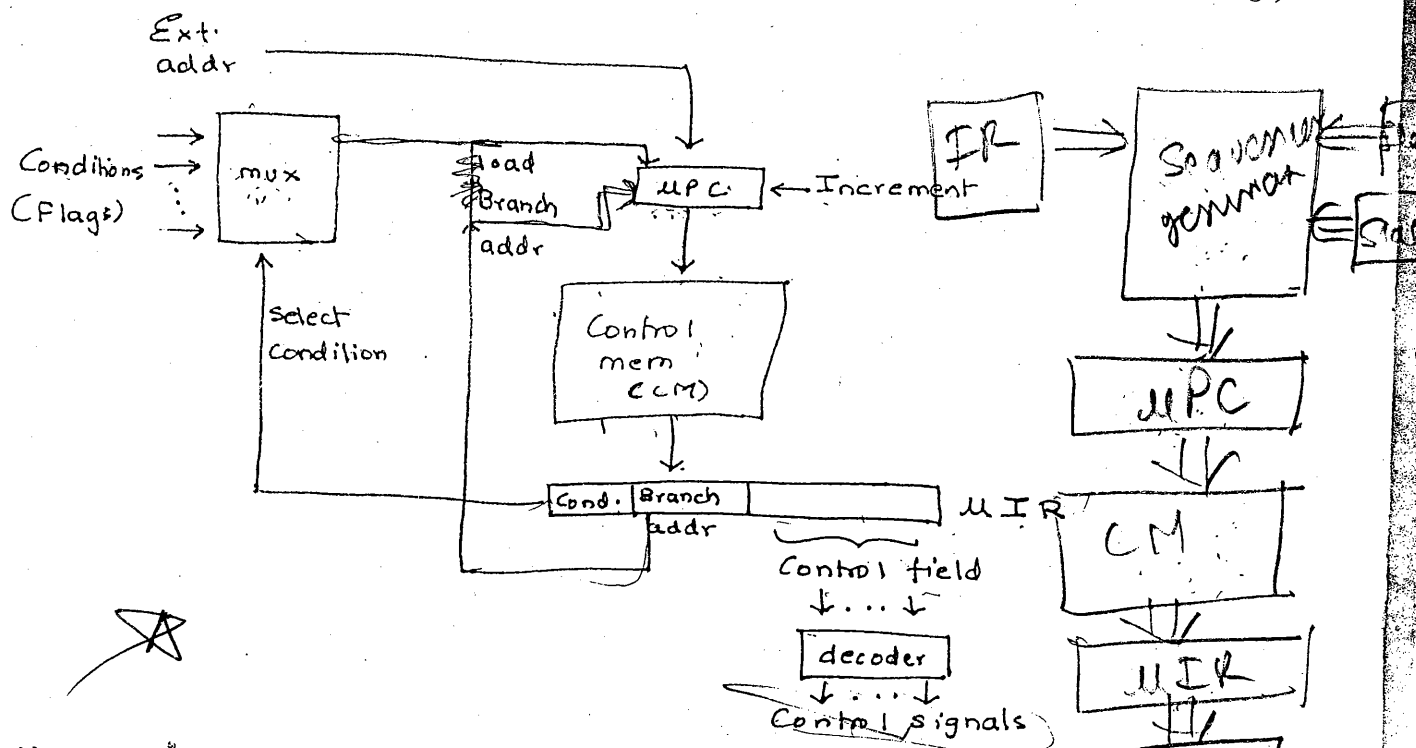
CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

TYPICAL MICROPROGRAM CONTROL UNIT {Note: This is different from Wilkes' Design}



- 1) This structure is more organized and more detailed than the Wilkes' Design.
- 2) Even here, the Control Memory contains the various μ -Programs to be executed by the CPU.
- 3) A μ PC (Micro Program Counter), is used to give the address of the next μ -Instruction in the Control Memory.
- 4) The μ PC is initially loaded from an external source.
- 5) Thereafter, μ PC is constantly incremented after every instruction as long as the program flow is sequential. This avoids the need to carry the next address in every instruction, unless it is a branch instruction.
- 6) Once we take a branch, the new branch address is loaded into the μ PC.
- 7) Using the address given by μ PC, the μ -Instruction is fetched from Control Memory into μ IR (Micro Instruction Register).
- 8) The μ -Instruction has three fields:
 - The Control field which indicated the control signals to be produced.
 - The branch address in case of a branch instruction.
 - The condition for the branch.
- 9) If the above μ -Instruction is sequential, then the control field is decoded and control signals are produced. μ PC is simply incremented.
- 10) If the μ -Instruction is a branch instruction, then first, the condition is checked. Since there could be various conditions (Eg: JCarry, JZero, JSign etc), a mux is used to select the current condition.
- 11) Only if the condition is satisfied, the new branch address from the μ -Instruction is Loaded into the μ PC.

AMD-2909 --- MICRO-PROGRAM SEQUENCER

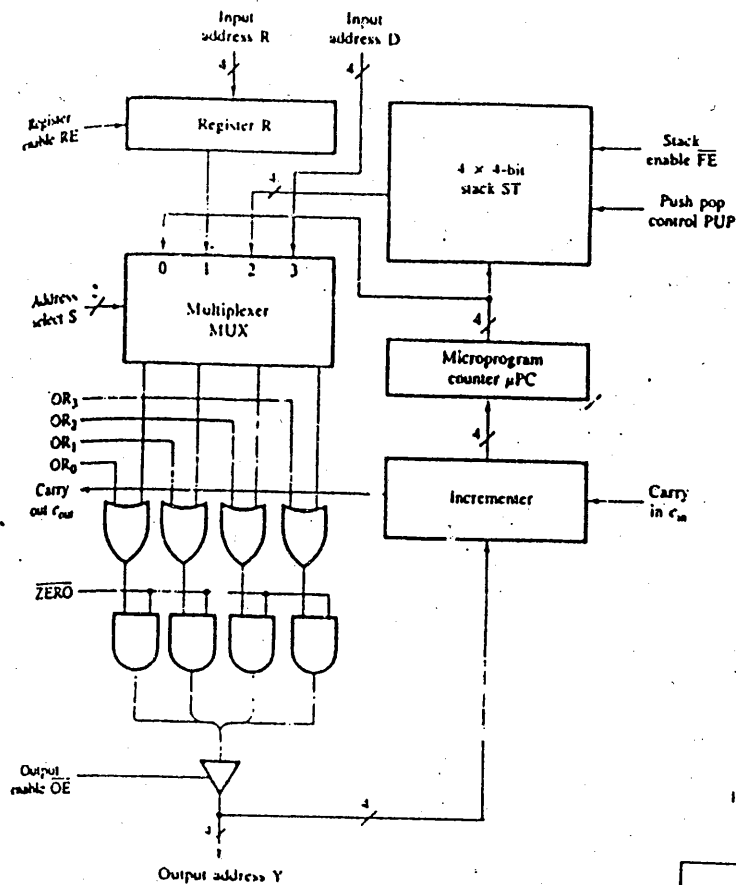


FIGURE 4.40
Structure of the 2909 microprogram sequencer slice.

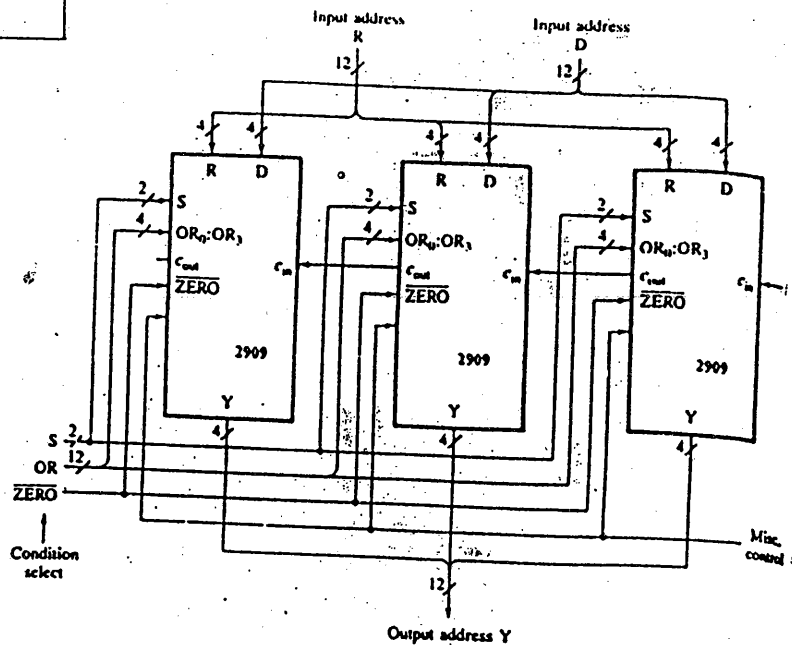


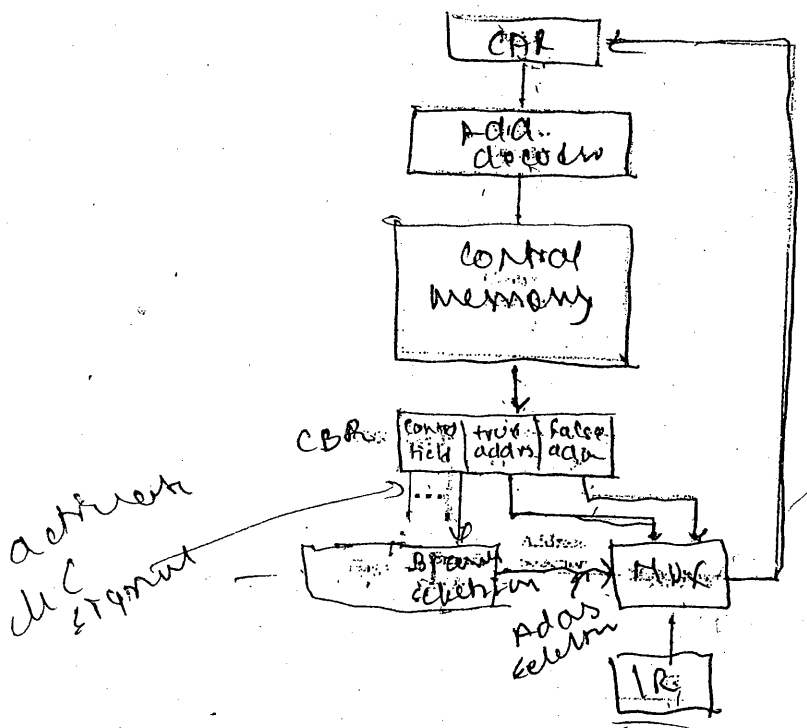
FIGURE 4.41
2909-based bit-sliced microprogram sequencer for 12-bit addresses.

Micro-program Sequencing

It is the technique of determining the address of the next Micro-instruction and hence deciding the flow of the micro-program.

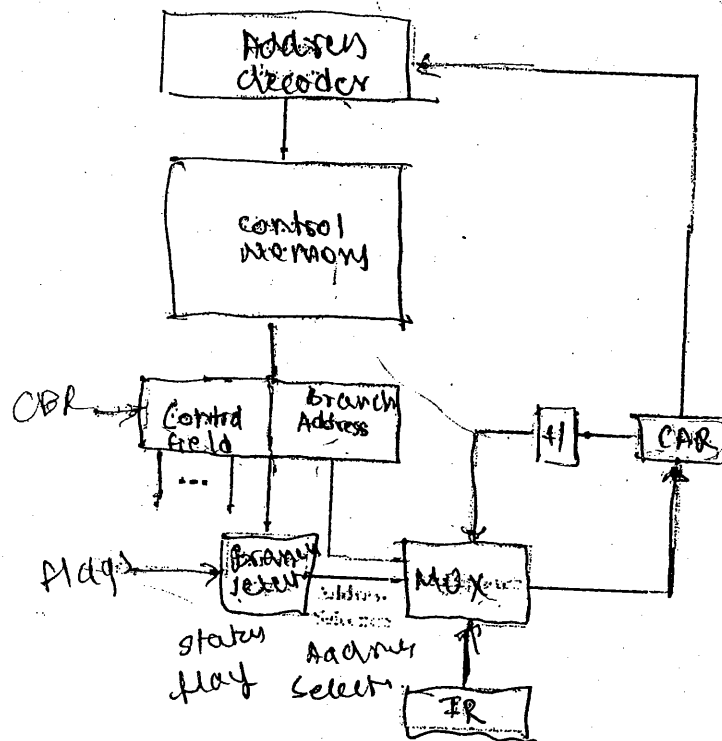
A Microprogram that always executes sequentially is called a Straight Line Microprogram (SLM). But not all Microprograms are SLMs. Hence a sequencing technique is required. Microprogram sequencing can be done in two ways: Dual Field Micro-instruction or Single Field Micro-instruction.

Dual Field Micro-Instruction



- 1) The address of the first Micro-instruction is loaded into Car from the main IR.
- 2) Here-after every Micro-instruction carries address of the next Micro-instruction.
- 3) In case there is a **conditional branch** Micro-instruction, then it will carry two addresses: For true condition and for false condition. Hence it is called dual field Micro-instruction.
- 4) The branch logic will check the status flags to determine if the condition is true or false.
- 5) Accordingly the correct address will be loaded using the multiplexer into the CAR.

Single Field Micro-Instruction



- 1) The address of the first Micro-instruction is loaded into Car from the main IR.
- 2) Here-after every Micro-instruction **will NOT** carry address of the next Micro-instruction.
- 3) Instead, the address from **CAR** will **simply be incremented**.
- 4) This will go on as long as the execution is **sequential** (Straight Line Microprogram - SLM)
- 5) In case there is a **branch** Micro-instruction, only then it will **carry the branch address**. Hence it is called Single Address Field Micro-instruction.
- 6) If it is an **unconditional branch**, then the address will be **directly loaded into CAR**.
- 7) But if it is a **conditional branch**, the **Flags** will be **checked** and if the condition is **satisfied**, the **branch address** will be loaded into CAR.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

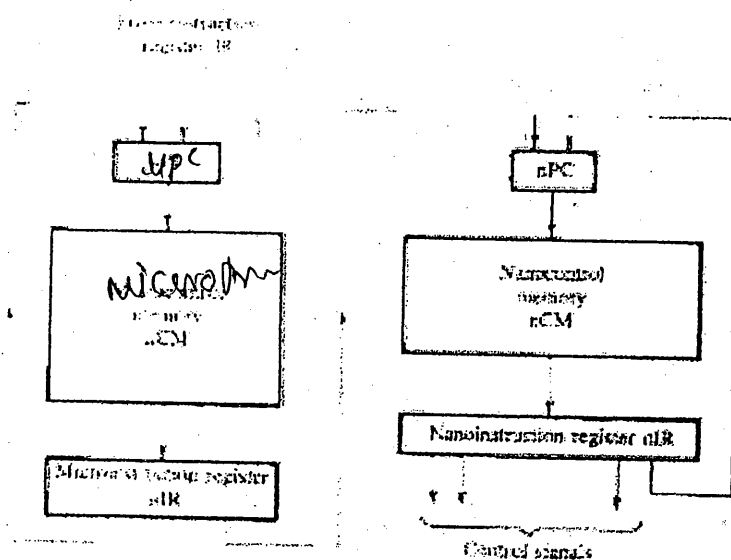
Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

μ -Instruction Design:

The main task of the μ -Instruction is to generate the control signals.
The Control field within the μ -Instruction indicates the control signals.
There are two ways to design the μ -Instructions, each having its own pros and cons:

	Horizontal μ -Instructions	Vertical μ -Instructions
1	<p>Here each bit of the μ-Instruction represents a control signal to be produced. Hence whichever bits are "1", those signals are produced.</p> <p>Eg: 8-bit μ-instruction</p> <pre> c₇ c₆ c₅ c₄ c₃ c₂ c₁ c₀ 0 1 0 0 1 1 0 1 v v v v v v v v c₆ c₃ c₂ c₀ ----- Control signals </pre>	<p>Here the bits of the μ-Instruction are decoded. The decoder produces the various control signals.</p> <p>Eg: 3-bit μ-instruction</p> <pre> b₂ b₁ b₀ v v v [3:8 decoder] c₇ ... c₁ c₀ ----- Control signals. </pre>
2	"n" bits will produce only "n" control signals. Hence if we want more control signals the instructions will have to contain more bits. Hence they grow "horizontally".	"n" bits will produce "2n" control signals through the decoder. Hence less number of bits are required in the instruction.
3	Multiple control signals can be produced using the same instruction by simply making more than one bits "1".	One instruction, upon decoding will produce only one control signal. Hence to produce more control signals, multiple instructions should be written. Hence the μ -program grows "vertically".
4	No external decoder required hence the circuit is simple.	Since a decoder is required, cost and complexity increases.
5	Multiple signals can be produced simultaneously. {Parallelism}	Only one signal can be produced at a time.

Nano-Programming:



- 1) For large Control units requiring many control signals, neither vertical nor horizontal designs are optimal.
- 2) **Horizontal** design makes **operations fast** but will tremendously **increase the size** of the Micro-instruction.
- 3) **Vertical** design will keep the **size** of the Micro-instruction **small** but can produce **only one control signal** at a time resulting in slow operations.
- 4) Hence a **combination** of both designs is used, giving rise to a new technique called nano-programming.
- 5) Here **two-level control memory storage** is used.
- 6) The higher level **Microcontrol Memory (μCM)** uses **vertical format** Micro-instructions.
- 7) The lower level **Nanocontrol memory (nCM)** uses **horizontal format** Micro-instructions.
- 8) The **Micro-instruction** from the μCM is **decoded to select the nano-instruction**.
- 9) The **nano-instruction** from the nCM actually **produces the required control signals**.
- 10) Hence various control signals can be produced simultaneously without increasing the size of the Micro-instruction.
- 11) However, operations become **slower** as two memories have to be accessed.

CHOPRA ACADEMY

Computer Organization
and Architecture

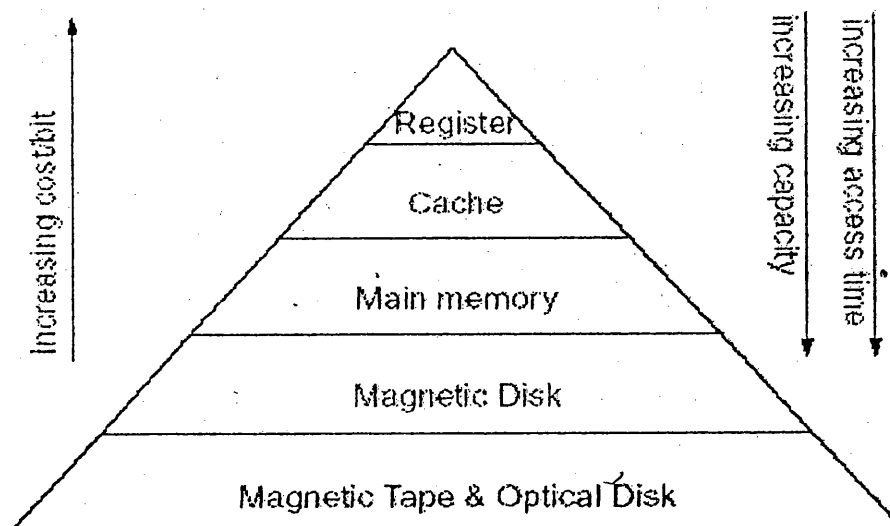
Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

✓	Hardwired Control Unit	Microprogrammed Control Unit
1	Control signals are generated using hardware	Control signals are generated using a Microprogram
2	Since hardware is used, the circuit design is rigid	Since μ -Program is used, design is flexible and can be changed by simply changing a few μ -Instructions
3	Debugging such a circuit is difficult	Debugging a Microprogram is easier
4	Since there is no software, no time is wasted in fetching and decoding the μ -Instructions. Hence Faster	Since software is used, time is wasted in fetching and decoding the μ -Instructions. Hence Slower .
5	Cheaper as Control Memory is not required.	Cost of control memory increases the system cost .
6	Due to their rigid-ness, they are more suited to RISC computers.	Due to their flexibility, they are more suited to CISC computers.
7	Difficult to handle complex instructions	Easier to handle complex instructions
8	Emulation is NOT possible	Emulation is possible by simply copying the control memory of one CPU into another.

MEMORY

MEMORY HIERARCHY:



see location
cost/bit

Main Memory:

- This is the actual memory accessed by the CPU.
- The size of this memory depends on the size of the address bus of the CPU.
- It is implemented using semiconductor chips.
- It comprises mainly of RAM and a small amount of ROM.
RAM is used to store the current programs and data.
ROM is used for storing permanent programs like the BIOS.
- These memories are located on the motherboard.

Secondary Memory:

- This memory was invented mainly to increase the storage space.
- It is independent of the size of the address bus of CPU.
- It is implemented in the form of magnetic storage devices which have a lower cost per bit than semiconductor chips. Eg: Hard Disk.
- Only its connector is present on the motherboard.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Cache Memory:

- This memory was invented mainly to **increase the speed of operations**.
- It is **implemented using SRAM** chips.
- **SRAM** is much **faster than DRAM** but is **more expensive, larger in size** and consumes **more power**. Hence only a **limited amount** of cache can be successfully implemented in a system.
- It is located very close to the CPU on the **motherboard**.

Offline Memory:

- It is **implemented using magnetic tapes**.
- Its main purpose is to **increase the storage capacity**.
- It is **very slow** as compared to the internal memories of the computer.
- It is **easily portable** as it can be **physically detached** and carried elsewhere.

Conclusion:

As we move away from the CPU in the memory hierarchy, the following behavior is shown:

- **Speed decreases**
- **Storage space increases**
- **Cost per bit decreases.**

MEMORY CHARACTERISTICS:



1) Location

- Onchip: Present **inside the CPU**. Eg: L1 Cache, **Internal registers** of the CPU.
- Internal: Present **on the Motherboard**. Eg: **Cache and Main Memory**.
- External: Only **connectors** present on the motherboard. Eg: **Hard disk, CD ROM, e**

2) Storage Capacity

This indicates the amount of data stored in the memory.
It should be as **large** as possible.

It is expressed in terms of: $m \times n$

m = No of memory locations (*no of words*);

n = no of bits stored per location (*word size*).

3) Transfer Modes

- Word Transfer: Here only the amount of information **required** is transferred.
Eg: **Main Memory to CPU**
- Block Transfer: Here **more** information than what is required is transferred.
Eg: **Virtual Memory to Main Memory**.

4) Access Modes

- Serial Access: Locations are accessed **one after the other** in a sequential manner.
Time of access depends on the location where the data is stored.
Eg: **Magnetic tapes**.
- Random Access: Locations can be **accessed in any order**.
Since the required location can be accessed directly, **time of access is the same for all locations**.
Eg: **RAM** (Random Access memory).

5) Physical Properties

- Volatile: **Contents are retained after the power is switched off**. Eg: **ROM**.
 - Non-Volatile: Here **contents are lost after the power is switched off**. Eg: **RAM**.
 - Erasable: Here the contents **can be changed**. Eg: **RAM**.
 - Non-Erasable: Here contents **cannot be changed**. Eg: **ROM**.
- Desirable: Non-Volatile & erasable** Eg: **Hard-Disk**.

6) Access time (Performance --- t_A)

This is the **time** between a **request** is made and the information is **available**.
It should be as **low** as possible. #Please refer Bharat Sir's Lecture Notes for this ...

7) Reliability

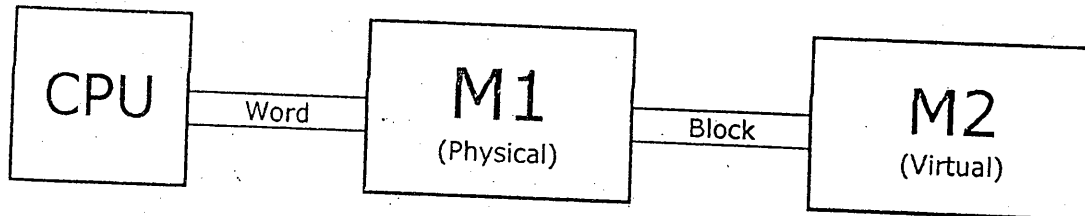
It is measured as the **Mean Time To Failure (MTTF)**.
It should be as **high** as possible.

8) Cost

This indicates the cost of the memory.
It should be as **small** as possible.
It is expressed in terms of: **Cost/bit**.

It is clear that no single type of memory can satisfy all the above characteristics; hence a **memory hierarchy is required**. With this we can get a large storage space at low cost and get a high performance. #Please see numericals solved in the class...

PERFORMANCE CHARACTERISTICS OF A TWO LEVEL MEMORY: {V Imp}



$C_1 \rightarrow$ Cost/bit
 $S_1 \rightarrow$ Capacity
 $tA_1 \rightarrow$ Access Time

$C_2 \rightarrow$ Cost/bit
 $S_2 \rightarrow$ Capacity
 $tA_2 \rightarrow$ Access Time

- M_1 is present on the motherboard and is physically connected to the system bus of the CPU.
- M_2 is NOT present on the motherboard, only its connector is present.

- 1) Whenever CPU needs to access data, it first checks in M_1 .
If data is found in M_1 , it is called a **HIT**. Operation is performed at M_1 itself, as it is faster.
This access time is called tA_1 . Hence:

$$tA_1 = \text{time } (M_1 \rightarrow CPU)$$

- 2) If data is NOT found in M_1 , it is called as **MISS**. Now the block containing the data is first transferred to M_1 , and then the data is operated from M_1 as it is faster.
This access time is called tA_2 . Hence:

$$tA_2 = \text{time } (M_2 \rightarrow M_1) + \text{time } (M_1 \rightarrow CPU)$$

Notice that the entire Block is transferred, as the remaining data from the block may be required in the future and such accesses will be a HIT.

- 3) **Hit Ratio (H)** is defined as the No. of hits / Total No. of attempts
If N_1 = no. of Hits and N_2 = no. of Misses then:

$$H = N_1 / (N_1 + N_2) \quad \text{\#Please refer Bharat Sir's Lecture Notes for this ...}$$

H gives the probability that the required data can be accessed from the fast M_1 .
Hence Max value of $H = 1$. This will happen when all attempts will be Hits

- 4) **Average Access Time (tA)** is defined as:

$$tA = H.tA_1 + (1-H).tA_2$$

- 5) **Average Cost (C)** is defined as the Total Cost/ Total Storage.

$$C = \{ (C_1 S_1) + (C_2 S_2) \} / (S_1 + S_2)$$

- 6) **Access Efficiency (E)** is the ratio by which the Average Access Time differs from its min value.

$$E = tA_1 / tA$$

It should be as high as possible. Max Value of $E = 1$.

- 7) **Speed ratio (R)** denotes how slow M_2 , is compared to M_1 .

$$R = tA_2 / tA_1$$

It should be as low as possible. Min Value of $R = 1$.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

- Q) For the above Memory hierarchy, solve the following: *{repeated exam question, for 10-16m}*
- When will the Average cost approach C_2 ?
 - What is the Average access time for this Hierarchy?
 - Express E in terms of R and H
 - Plot E against H for $R = 100$ and $R = 520$. What are your comments on the performance?

Soln:

a) $C = \{(C_1 S_1) + (C_2 S_2)\} / (S_1 + S_2)$

Here, if $S_2 \gg S_1$, then S_1 can be ignored

Hence $C \approx C_2 S_2 / S_2$

Hence $C \approx C_2$

Thus the Avg Cost will approach C_2 if the capacity of M2 is very large as compared to M1.

b) Average access time $t_A = H \cdot t_{A1} + (1-H) \cdot t_{A2}$ Notice that $t_{A1} < t_A < t_{A2}$.

c) Average Efficiency $E = t_{A1} / t_A$ & Speed Ratio $R = t_{A2} / t_{A1}$

Now

Divide by t_{A1}

Hence

$$t_A = H \cdot t_{A1} + (1-H) \cdot t_{A2}$$

$$t_A / t_{A1} = H + (1-H) \cdot t_{A2} / t_{A1}$$

$$1/E = H + (1-H) \cdot R$$

Hence

$$E = 1 / \{H + (1-H)R\}$$

Or

$$E = 1 / \{R + (1-R)H\}$$

- d) The various values of E w.r.t H and R are as follows:

	H = 0	H = 0.2	H = 1 (Max)
R = 520 →	E = 0.0019	E = 0.0024		E = 1
R = 100 →	E = 0.01	E = 0.0124		E = 1
R = 1 (min) →	E = 1	E = 1		E = 1

Graph: (refer solved examples from lecture notes)

Comments:

- As seen above, for all general cases, E depends upon both, H(Hit Ratio) and R(Speed Ratio).
- If $H = 1$ (all access made from fast M1), then E will always be its max value (1), irrespective of R.
- If $R = 1$ (M1 and M2 are of same speed), then E will always be 1, irrespective of H.
- If $H = 0$ (No hits at all), then performance (E) simply depends upon R. Smaller the value of R, better is the Efficiency. #Please refer Bharat Sir's Lecture Notes for this ...

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Q) For the given Memory hierarchy, solve the following: {Exam Question}

$$t_{A1} = 10^{-7} \text{ sec } \{\text{Fast M1}\}$$

$$t_{A2} = 10^{-2} \text{ sec } \{\text{Slow M2}\}$$

$$E = 0.9$$

$$H = ?$$

Soln:

We Know

$$R = t_{A2} / t_{A1}$$

Given

$$R = 10^{-2} / 10^{-7}$$

Hence

$$R = 10^5 \text{ sec}$$

Now

$$t_A = H \cdot t_{A1} + (1-H) \cdot t_{A2}$$

Divide by t_{A1}

$$t_A / t_{A1} = H + (1-H) \cdot t_{A2} / t_{A1}$$

Hence

$$1/E = H + (1-H) \cdot R$$

$$\text{Hence Efficiency } E = 1 / \{R + (1-R)H\}$$

Given

$$E = 0.9$$

Substituting E \rightarrow

$$0.9 = 1 / \{10000 + (1 - 10000)H\}$$

$$\text{Hence Hit Ratio } H = 0.999 \text{ (Note: Don't approximate } H \rightarrow 1)$$

Q) For the given Memory hierarchy, solve the following: {Exam Question}

Fast M1:

$$S_1 = 2^{10} \text{ (1KB)}$$

$$t_{A1} = 10^{-8} \text{ sec}$$

$$C_1 = 0.1$$

$$H = 0.9$$

Slow M2:

$$S_2 = 2^{16} \text{ (64KB)}$$

$$t_{A2} = 10^{-6} \text{ sec}$$

$$C_2 = 0.01$$

Calculate Avg Cost, and Avg t_A ?

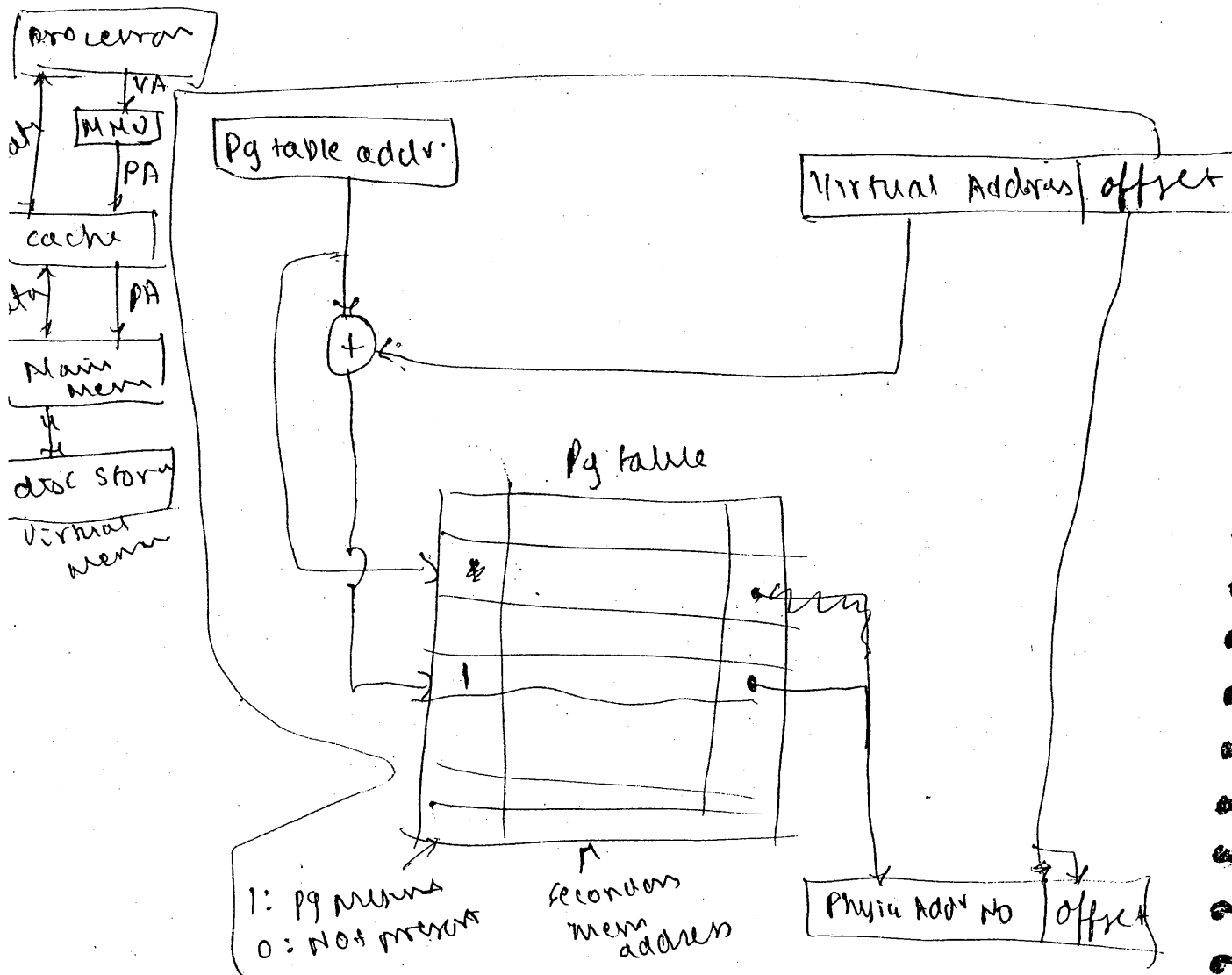
Soln:

$$\begin{aligned} 1) \text{ Avg Cost per bit (C)} &= \{(C_1 S_1) + (C_2 S_2)\} / (S_1 + S_2) \\ &= \{(0.1 \times 2^{10}) + (0.01 \times 2^{16})\} / (2^{10} + 2^{16}) \\ \text{Hence (C)} &= 0.011 \text{ (Since } S_2 \gg S_1, C \approx C_2) \end{aligned}$$

$$\begin{aligned} 2) \text{ Avg access time (tA)} &= H \cdot t_{A1} + (1-H) \cdot t_{A2} \\ &= 0.9 \times 10^{-8} + (1-0.9) \times 10^{-6} \\ \text{Hence (tA)} &= 10.9 \times 10^{-7} \text{ (0.109 } \mu\text{Sec)} \end{aligned}$$

VIRTUAL MEMORY {IMP FOR VI SEM --- ELECTRONICS BRANCH}

- The need for introducing Virtual Memory in the system was to **increase the memory size** locations).
- **Main Memory** size is **restricted** by the size of the **address bus**. (N-bit address bus: 2^n locations).
Eg: With an 80386 CPU the Address Bus is 32 bits.
Hence it can have 4 Giga Bytes (2^{32}) of Main Memory.
But it can also access 64 Terra Bytes (2^{46}) of Virtual Memory.
- **Virtual Memory** is implemented by using **cheaper storage technology** i.e. Magnetic (HDD) and Optical Storage (CD-ROM). Hence we can get a **large amount of Memory** at a **low cost**.
- Whenever the CPU wants data from the Virtual Memory, the **data is first copied from Virtual Memory to Main Memory** and then accessed by the CPU from the Main Memory.
- Instead of copying only the required data, the **entire block** containing the data is copied so that further access will be from the Main Memory directly.
- To implement this, two methods can be used: Paging and Segmentation.



PAGING

- 1) Here the **Virtual Memory** space is **divided into** equal size **Pages** (generally 4KB).
- 2) The **Main Memory** space is **divided into** equal size **page frames**.
Each frame of Main Memory can hold any page from Virtual Memory.
- 3) When the CPU wants to access a page, it first looks at the Main Memory.
Hit: If the required page is found in the Main Memory it is called a HIT. Now transfer is between the CPU and Main Memory.
- 4) **Page Fault:**
If the CPU needs a page that is not present in the Main Memory then it is called a Page Fault. Now the page has to be loaded from Virtual Memory to Main Memory.
- 5) **Page Replacement:**
After a page fault, if there are **no empty frames** in the Main Memory, then an **old page** from Main Memory is **removed** and the **new page is inserted**. This is called Page Replacement. There are various replacement algorithms, such as FIFO, LRU, LFU, Random etc.
- 6) **FIFO: First In First Out.**
Here, the page that entered the Main Memory first will be the one that will leave first.
i.e. The oldest page of Main Memory should be replaced.
- 7) **LRU: Least Recently Used.**
Here, the page that has not been used for the longest time is replaced.
- 8) **LFU: Least Frequently Used.** # Practice numericals on replacement algorithms from Bharat Sir's lecture notes.
Here, the page that has been used the least number of times is removed.
- 9) **Dirty Pages:**
During replacement, if the old page has been **modified in the Main Memory**, then it needs to be **first saved** into the Virtual Memory and then replaced. The CPU keeps track of all such "updated" pages by maintaining a "**Dirty**" bit for each page.
When the page is updated in the Main Memory, its Dirty bit is **set (1)**, and the page is said to be Dirty. Dirty pages are first copied into the Virtual Memory, and then replaced.
- 10) **Thrashing:**
When a program causes a **lot of page faults**, it is said to be thrashing. It should be **avoided**.
- 11) **Demand Paging:** © In case of doubts, contact Bharat Sir: - 98204 08217.
If pages are loaded into the Main Memory only when required by the CPU, then it is called Demand Paging. Thus pages are **loaded only after page faults**.
- 12) **Working Set Model:**
This method is the **opposite of Demand Paging**. Here the **pages required by application** are grouped together as its **working set**. Whenever an application loads, its **entire working set is loaded** into the Main Memory. Hence the application will not cause any page-faults while running, and therefore **run faster**. But it may tend to **monopolize the Main Memory** space as all its pages are loaded, and leave **no space for other programs**. Further, the application will require a **lot of time to start-up** as compared to Demand Paging as all its required pages have to be loaded on start-up. #Please refer Bharat Sir's Lecture Notes for this ...
- 13) **Translation Look-Aside Buffer. (TLB)**
This is an on-chip buffer within the CPU, used to speed up the Paging process.
Since a page from Virtual Memory can get stored into any frame of Main Memory, the OS (operating system), maintains a **Page Table** which indicates **which page of Virtual Memory** is stored in each page frame of the Main Memory.
Hence for accessing the page the **CPU has to do 2 memory operations:**
First access the page table to get information about where the page is stored in Main memory, **then access the Main Memory for the page**. This double process is obviously time-consuming and brings down the performance.
To **solve** this problem, the **CPU copies the Page table information** of the most recently used pages in the on-chip **TLB**. Thereafter any **subsequent access** to the page will be **faster** as the **information will be provided by the TBL** and the CPU need not access the page table.

SEGMENTATION:

- 1) Here the memory space is divided into logically different areas called segments. This makes the memory very organized and hence easy to use for the programmer.
Eg: In 8086 the 4 segments are Code Segment, Stack, Data And Extra Segment.
- 2) Programmer is aware of segmentation and can relocate the segments during the program.
- 3) Segments don't have a fixed size and hence unnecessary wastage of space called fragmentation of the Main Memory is prevented. #Please refer Bharat Sir's Lecture Notes for this ...
- 4) Here the Best fit algorithm is widely used due to the flexible size of segments.
- 5) Address is given as a combination of segment address and offset address.
Segment address indicates the starting address of the segment.
Offset address indicates the offset of the current location within the segment.
The Physical Address is calculated before performing the memory operation.
- 6) Since within a segment, only the offset address changes, the max size of the segment depends on the size of the offset address.

	PAGING	SEGMENTATION
1	Pages are of fixed size	Segments are of variable size.
2	Causes Main Memory fragmentation.	Does not cause Main Memory fragmentation.
3	Programmer is unaware of paging.	Programmer is aware of segmentation.
4	Uses algorithms like LRU, LFU etc	Uses algorithms like First Fit, Best Fit etc
5	Has physical boundaries.	Has logical Boundaries
6	Difficult to implement	Easy to implement
7	Cannot differentiate between code and data	Differentiates between the logical elements

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Numericals on Page Replacement Policies: {Imp type: 8m}

Q: Main Memory has 3 pages and the processor requires pages form Virtual Memory in the following order: 2 3 2 1 5 2 4 5 3 2 5 2. Show the Implementation of FIFO, LRU and LFU.

Soln:

FIFO	2	3	2	1	5	2	4	5	3	2	5	2	HIT Ratio =
Frame 1	2*	2*	2*	2*	5	5	5*	5*	3	3	3	3*	0.25
Frame 2		3	3	3	3*	2	2	2	2*	2*	5	5	
Frame 3				1	1	1*	4	4	4	4	4*	2	
			HIT					HIT		HIT			

LRU	2	3	2	1	5	2	4	5	3	2	5	2	HIT Ratio =
Frame 1	2	2	2	2	2	2	2	2	3	3	3	3	0.42
Frame 2		3	3	3	5	5	5	5	5	5	5	5	
Frame 3				1	1	1	4	4	4	2	2	2	
			HIT			HIT		HIT			HIT	HIT	

LFU	2	3	2	1	5	2	4	5	3	2	5	2	HIT Ratio =
Frame 1	2 (1)	2 (1)	2 (2)	2 (2)	2 (2)	2 (3)	2 (3)	2 (3)	2 (3)	2 (4)	2 (4)	2 (5)	0.50
Frame 2		3 (1)	3 (1)	3 (1)	5 (1)	5 (1)	5 (1)	5 (2)	5 (2)	5 (2)	5 (3)	5 (3)	
Frame 3				1 (1)	1 (1)	1 (1)	4 (1)	4 (1)	3 (1)	3 (1)	3 (1)	3 (1)	
			HIT			HIT		HIT		HIT	HIT	HIT	

Q: M1 = 8K, M2 = 16K, Page Size = 2K. Show FIFO, LRU and LFU for the following order 7 5 3 2 1 0 4 1 6 7 4 2

Soln: M1 has 4 pages and M2 has 8 pages.

FIFO	7	5	3	2	1	0	4	1	6	7	4	2	HIT Ratio =
Frame 1	7	7	7	7	1	1	1	1	1	7	7	7	0.16
Frame 2		5	5	5	3	3	3	3	3	5	5	5	
Frame 3			3	3	2	2	2	2	2	3	3	3	
Frame 4				2	2	2	2	2	2	2	2	2	

LRU	7	5	3	2	1	0	4	1	6	7	4	2	HIT Ratio =
Frame 1	7	7	7	7	1	1	1	1	1	7	7	7	0.16
Frame 2		5	5	5	3	3	3	3	3	5	5	5	
Frame 3			3	3	2	2	2	2	2	3	3	3	
Frame 4				2	2	2	2	2	2	2	2	2	

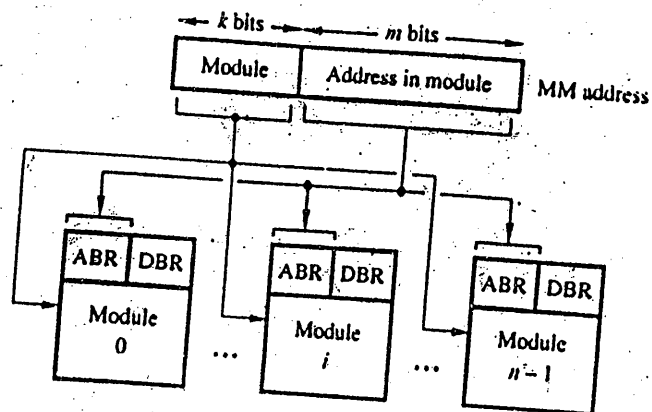
LFU	7	5	3	2	1	0	4	1	6	7	4	2	HIT Ratio =
Frame 1	7	7	7	7	1	1	1	1	1	7	7	7	0.16
Frame 2		5	5	5	3	3	3	3	3	5	5	5	
Frame 3			3	3	2	2	2	2	2	3	3	3	
Frame 4				2	2	2	2	2	2	2	2	2	

INTERLEAVED MEMORIES {VIMP}

Interleaving is a mechanism to increase the total memory accessed combining a set of memory modules (chips). There are two main types of interleaving. Consider a total of 4 KB memory. Here we have 4096 locations having addresses 0 to 4095. Instead of having the above memory in a single module, it can be comprised of 2 modules in the following 2 methods:

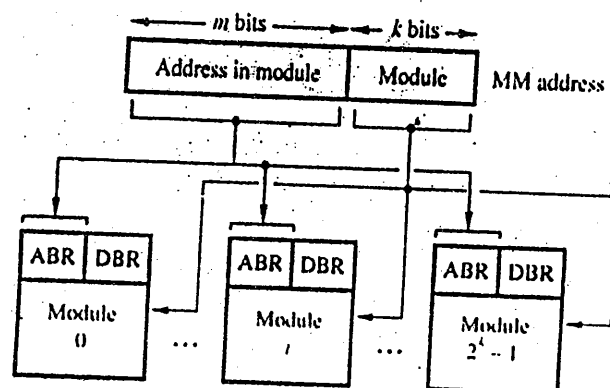
	HIGHER ORDER INTERLEAVING				LOWER ORDER INTERLEAVING			
1	<div>Address Data</div> <div>2048 <input type="text"/></div> <div>2049 <input type="text"/></div> <div>2050 <input type="text"/></div> <div>2051 <input type="text"/></div> <div> . <input type="text"/></div> <div> . <input type="text"/></div> <div>4095 <input type="text"/></div> <div><u>Bank 1</u></div>		<div>Address Data</div> <div>0000 <input type="text"/></div> <div>0001 <input type="text"/></div> <div>0002 <input type="text"/></div> <div>0003 <input type="text"/></div> <div> . <input type="text"/></div> <div> . <input type="text"/></div> <div>2047 <input type="text"/></div> <div><u>Bank 0</u></div>		<div>Address Data</div> <div>0001 <input type="text"/></div> <div>0003 <input type="text"/></div> <div>0005 <input type="text"/></div> <div>0007 <input type="text"/></div> <div> . <input type="text"/></div> <div> . <input type="text"/></div> <div>4095 <input type="text"/></div> <div><u>Bank 1 (Odd)</u></div>		<div>Address Data</div> <div>0000 <input type="text"/></div> <div>0002 <input type="text"/></div> <div>0004 <input type="text"/></div> <div>0006 <input type="text"/></div> <div> . <input type="text"/></div> <div> . <input type="text"/></div> <div>4094 <input type="text"/></div> <div><u>Bank 0 (Even)</u></div>	
2	Here, the higher address bits are decoded to select the memory module; hence it is called Higher Order Interleaving.				Here, the lower address bits are decoded to select the memory module; hence it is called Lower Order Interleaving.			
3	Consecutive locations lie on the same chip. <i>#Please refer Bharat Sir's Lecture Notes for this ...</i>				Consecutive locations lie on separate chips.			
4	The main goal here is to increase the number of memory locations .				The main goal here is to increase the size of data to be accessed simultaneously.			
5	Does not increase the speed of memory access.				Increases memory access speed as more data is accessed simultaneously . Eg: 8086 accesses 16-bits simultaneously by having 2 banks, each providing 8-bits. Pentium accesses 64 bits from 8-banks.			
6	Adding more modules is easy .				Adding more modules is difficult .			
7	Any number of modules is possible .				Number of banks is always 2ⁿ .			
8	Reliability is good . Even if one module fails, others can still be used.				Reliability is poor . If one bank fails, others cannot be used.			
9	Can be very effective in multiprocessor environment , as different processors can use data from different modules as data of different modules is not related.				No such advantage as parts of the same data is spread across different banks. <i>#Please refer Bharat Sir's Lecture Notes for this ...</i>			
10	It is optional and is used by most processors to increase the number of memory locations .				It is compulsory for processors like 8086 (16-bit), 80386(32 bit), Pentium(64 bit) etc, which require large size of data simultaneously .			

HIGHER ORDER INTERLEAVING



(a) Consecutive words in a module

LOWER ORDER INTERLEAVING



(b) Consecutive words in consecutive modules

FIGURE 5.24
Addressing multiple-module memory systems.

RAID --- Redundant Array of Inexpensive (Independent) Disks

- RAID is a method of combining multiple hard disks into a single logical unit for more data availability, and high level of performance.
- This offers better reliability and higher performance than a single large disk drive.
- With multiple disks, data can be divided into non-related parts (data mirroring) and spread across various disks. These disks can be simultaneously accessed by different processors, and thus we can get a high degree of parallelism. Further, even if one of these disks fails, we can still use the remaining data from the other disks and thus the system is more reliable.
- On the other hand, data can be "striped" across various disks and accessed by the same processor from all disks simultaneously, again providing high speed data access.
- In some cases, data is duplicated in more than one disk. This results in "Redundant" disks, but increases tolerance, as failure of one disk doesn't affect the system at all.
- For data recovery, parity information of all disks can be stored in the redundant disk.
- RAID is mostly used in Main Frame computers and data servers, where fast and reliable data access is of high importance. #Please refer Bharat Sir's Lecture Notes for this ...
- There are six levels of RAID, depending upon the type of usage required and the cost.

Advantages of RAID:

- 1) Allows parallel data access, in multiprocessor systems using "data mirroring".
- 2) Allows high speed data access even in uni-processor systems using "data-striping".
- 3) Makes system reliable due to the "Redundant" disk.
- 4) Allows data-recovery using parity information in the redundant disk.

Disadvantages of RAID:

- 1) Cost of system is high.
- 2) Writes are slower than reads due to the bottleneck at the redundant disk.
- 3) System is complex to implement. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Levels of RAID:

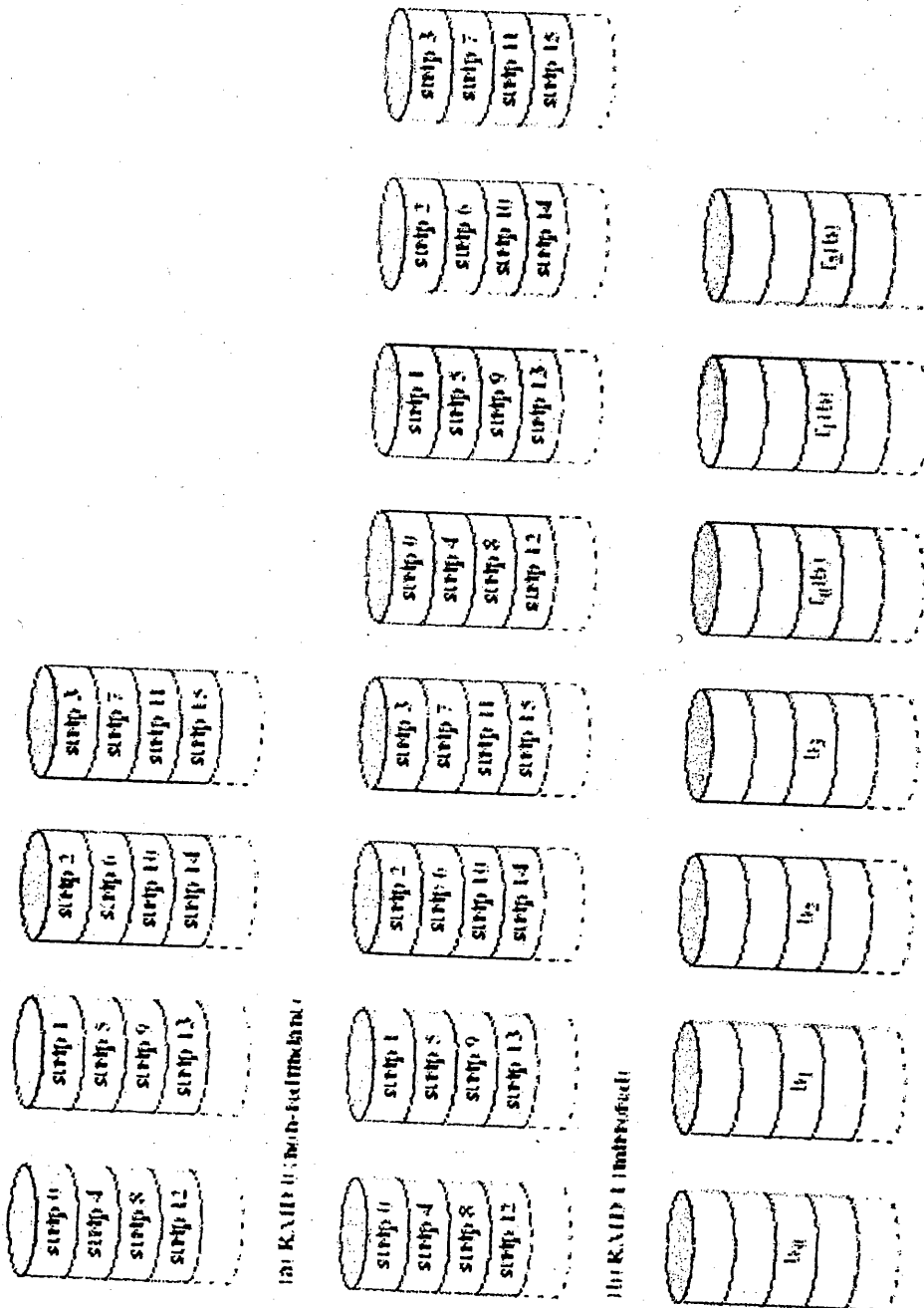
CATEGORY	LEVEL	DESCRIPTION	TRANSFER RATE (RD/WR)	ADVANTAGE	DISADVANTAGE	APPLICATION
Striping	0	Non-Redundant Data Striping: Data is striped and written across various disks	Fast/Fast	High Transfer Rate. Easy to implement.	No Fault tolerance as non-redundant. Hence not truly RAID.	Applications requiring high performance for non critical data.
Mirroring	1	Data Mirroring: Data is duplicated across various disks	Fast/Slow	High tolerance as complete copy of data is available even if one disk fails.	Expensive as cost is twice. Reads are fast but writes are slow as data is written twice.	System drives; Critical Files
Parallel Access	2	Redundant via Hamming Code	Fast/Slow	None	Multiple redundant disks hence expensive.	Obsolete
	3	Bit-Interleaved Parity: Simple parity bit for each set of corresponding bits	Fast/Slow	Fault tolerant. Only one disk used for storing parity.	Poor performance on writes. Bottleneck created at the parity disk during write operations as all processors need to update the parity.	Good for large multimedia files such as CAD etc.
Independent Access	4	Block-Interleaved Parity: Bit by bit parity calculated across stripes on each disk	Fast/Slow	Fault tolerant. Only one disk used for storing parity.	Poor performance on writes. Bottleneck created at the parity disk during write.	Good for applications with high read request rates.
	5	Block-Interleaved Distributed Parity: Data Striping and Parity Striping across all disks.	Fast/Moderate	Fault tolerant. Good for transaction processing as less bottlenecks during writes.	Still cannot match RAID 0 performance.	Network Servers, Data lookup operations.
	6	Block-Interleaved Distributed Parity: Two parity calculations stored on different disks	Fast/Moderate	Highly Fault tolerant. Three disks need to fail for data loss	Significant write penalty Expensive	Applications requiring extremely high data availability.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217



RAID 2 (Parity) through 4 disks

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

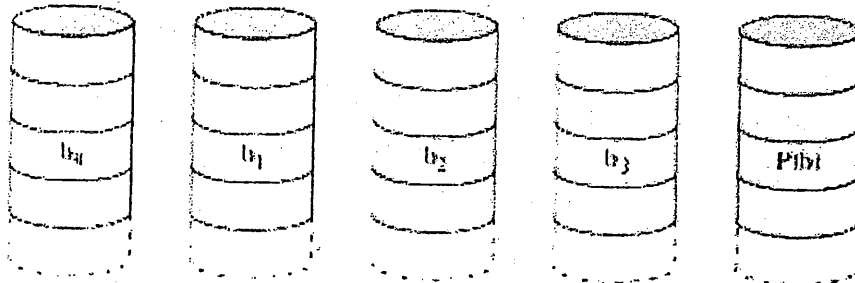


Fig. RAID 3 (bit-interleaved parity)

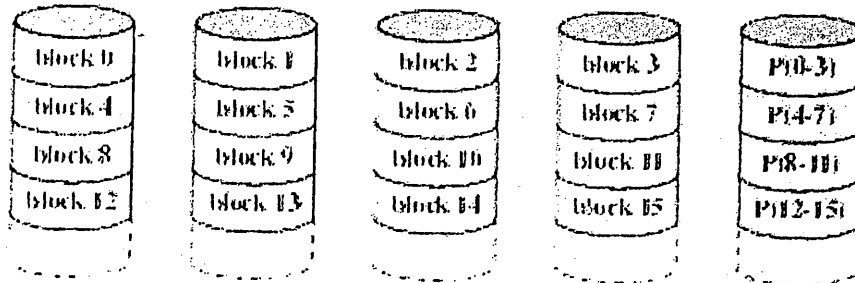


Fig. RAID 4 (block-level parity)

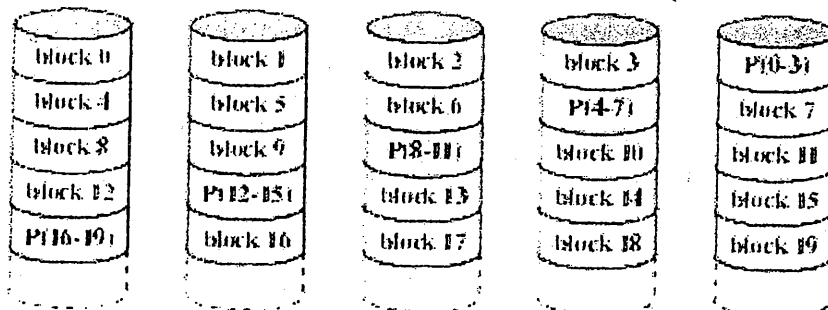


Fig. RAID 5 (block-level distributed parity)

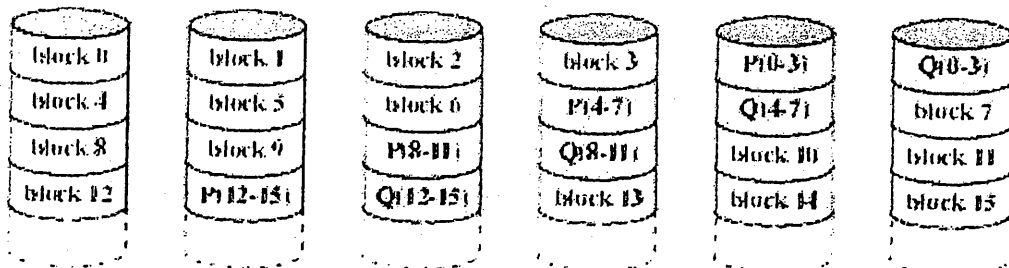
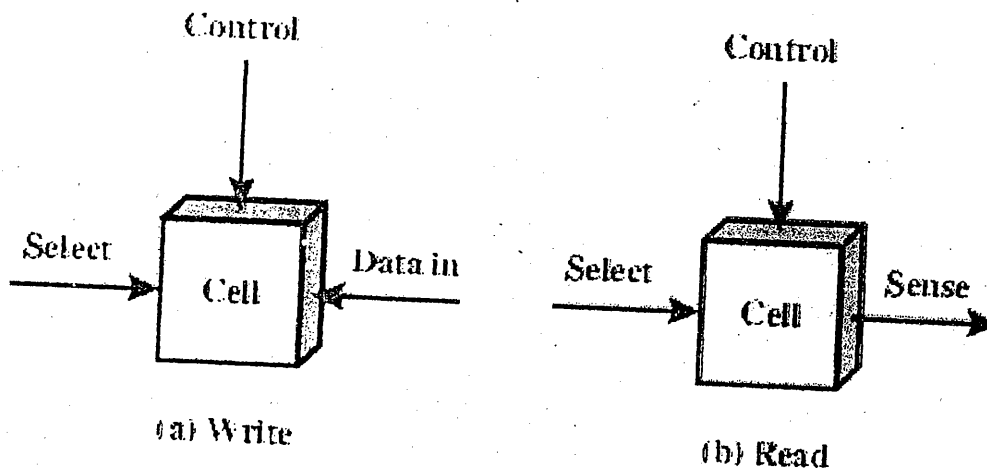


Fig. RAID 6 (dual redundancy)

Semiconductor Memories (Main Mem)

Memory Type	Category	Erase	Write Mechanism	Volatility
Random-access memory (RAM)	Read-write memory	Electrically, byte-level	Electrically	Volatile
Read-only memory (ROM)	Read-only memory	Not possible	Masks	
Programmable ROM (PROM)				
Erased PROM (EPROM)		UV light, chip-level		Nonvolatile
Electrically Erasable PROM (EEPROM)	Read-mostly memory	Electrically, byte-level	Electrically	
Flash memory		Electrically, block-level		

Memory Cell Operation:



Non-Volatile Memory (ROM)

1) ROM (Read Only Memory)

- ROM is Read Only Memory i.e. data once written in the ROM **cannot be changed**.
- It is **non-volatile** hence data is retained even without power supply.
- During the manufacturing process, **data is permanently wired** into the chip, hence it is never lost.
- Some important applications of ROM are: **Microprogram storage**, Storing permanent system programs like BIOS, storing data look-up tables etc.

2) PROM (Programmable Read Only Memory)

- Like ROM even PROM is **non-volatile and permanent**.
- Its only advantage is that it can be **easily written (programmed)** as compared to ROM.
- Hence if fewer chips are to be manufactured, PROM chips seem to be a **cheaper** option than ROM.

3) EPROM (Erasable Programmable Read Only Memory)

- A better version of PROM is the EPROM, which can be **ERASED** and then re-programmed.
- This gives far more **flexibility** to ROM.
- The basic characteristic of being **non-volatile** is still preserved.
- **To erase** the data, the EPROM chip must be **exposed to UV light**.
- Then by applying a **high voltage**, data can be **re-programmed** into the EPROM.

4) EEPROM (Electrically Erasable Programmable Read Only Memory)

- *EEPROM provides a great advantage over EPROM, that it **NEED NOT BE entirely ERASED** before it can be reprogrammed.
- This allows the user to **write only at specified locations** without losing all other data on the chip. *#Please refer Bharat Sir's Lecture Notes for this ...*
- Reading is fast, but **writing is considerably slower** ($>100 \mu\text{sec}/\text{byte}$).
- But EEPROM is **more expensive** than EPROM and also is less dense, providing **lesser storage capacity**.

5) Flash ROM

- It is like EEPROM, but it can be **erased very quickly (in a flash!)**. Hence the name Flash ROM.
- Like EEPROM, **only selected data can be erased** (though not at byte level).
- Data is stored as 1 bit per transistor hence **storage density is high**. This **increases the storage capacity**.
- **Cost wise**, Flash is **between EPROM and EEPROM**.
- It is most popular in **microcontroller based embedded systems** like mobile phones etc.

Volatile Memory (RAM)

1) SRAM v/s DRAM

	STATIC RAM	DYNAMIC RAM
1	Circuit:	Circuit:
2	Data is stored in Flip/Flops	Data is stored in charged capacitors
3	Faster	Slower
4	Occupies more space, hence provides low data storage.	Occupies less space, hence provides low data storage. <i>more</i>
5	Consumes more power supply	Consumes less power supply
6	More expensive	Less Expensive
7	Does not require a refreshing circuit	Capacitors need to be refreshed.
8	Used in small quantities in the form of Cache memory .	Used in large quantities in the form of Main Memory .

2) Advanced DRAM

- Since the **DRAM** constitutes most part of the **Main Memory**, maximum time of the system is spent in interacting with the DRAM.
- It is very important to make the DRAM very efficient, as a slow DRAM will cause a major **system bottleneck** and hence the performance of the entire system will drop.
- The **CPU** makes a request for data transfer with the DRAM. Now the **DRAM** gets "ready" for the transfer.
- If the DRAM is **slow**, this will **take time**. During this time the **CPU** is forced to "**WAIT**". This is what degrades the system performance.

- If for every memory operation, the CPU will have to perform "**wait-states**", then the system on the whole **will be slow**.
- To overcome this drawback, some **advanced techniques** have been incorporated in the new DRAMS. The two most popular ones are **SDRAM** and **RDRAM**.

a) Synchronous DRAM (SDRAM) ★

- This is one of the **most popular** forms of DRAM.
- Here the SDRAM **exchanges data with the CPU based on a clock**, which is running at the max speed of the CPU/ Bus clock. Due to this synchronization, transfers can be performed **without the need of "wait-states"**.
- In **typical DRAM** (not synchronized), once the **CPU makes a request** for data transfer, it has to "**WAIT**" for the DRAM to become ready. This waiting period is the access time of the DRAM, required for it to start the transfer. *#Please refer Bharat Sir's Lecture Notes for this ...*
- Here the **SDRAM moves data** in and out **under control of the system clock**.
- Just like the previous case, the **processor makes a request** for the data transfer.
- The **DRAM will respond after a set number of clock cycles**.
- Since the CPU and DRAM work on the same clock, the **CPU can perform SOME OTHER ACTIVITY** during this time **instead of entering WAIT STATES**.
- This is how SDRAM are **superior** over earlier asynchronous DRAMS.
- To improve the performance further, it uses the **Burst mode** of transfer. Here the **initialization time is required only for the first bit**, and the subsequent bits can be transferred **without any delay**. This is very useful for **large sequential data access** like graphic files.
- In addition SDRAM also has an **internal multi-bank architecture** to improve **on-chip parallelism**.
- Recently an enhanced version of SDRAM called **DDR (Double Data Rate) SDRAM** has been introduced. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- The DDR-SDRAM can **transfer data twice per single clock cycle** (once on rising edge, once on falling edge). Hence it **doubles the transfer rate**.

b) Rambus DRAM (RDRAM) ★

- In this method of implementing the DRAM, a special RAM "**BUS**" is used, hence the name Rambus DRAM.
- The special bus delivers address and control signals using an **asynchronous block-oriented protocol**.
- After an initial access time of 480 ns, this bus produces very high rate of data transfer upto 1.6 GBps.
- The high transfer rates are possible because the RDRAM gets control signals for memory requests over this high speed bus itself.
- The configuration consists of a DRAM controller connected to various DRAM modules over the common bus.
- The bus has 16 data lines and 2 parity bits.
- The data cycles at twice the clock speed, because 2 bits are sent per clock (one bit at leading edge, and one at falling edge).
- A separate set of 8 lines are used for address and control signals.

CHOPRA ACADEMY

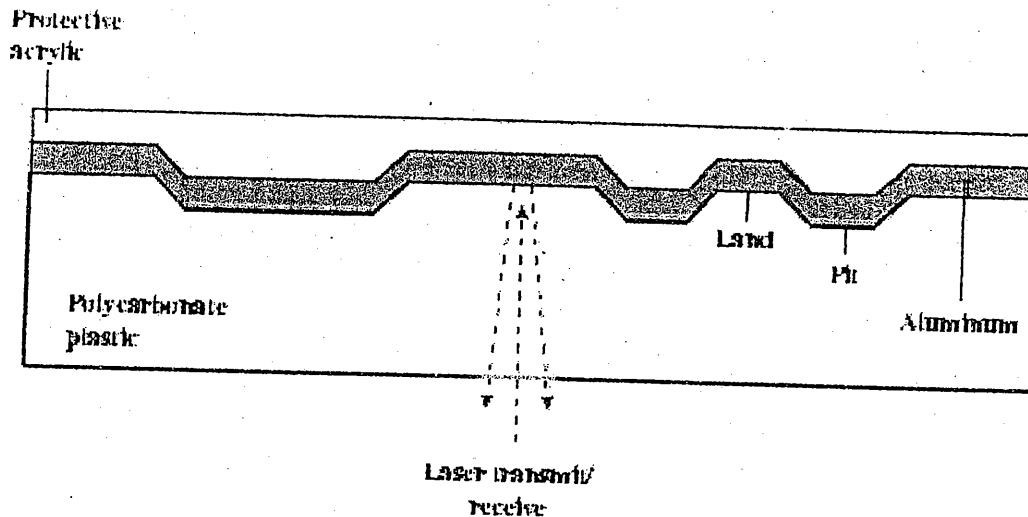
Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Memory Type	Technology	Size	Access Time
Cache	Semiconductor RAM	128-512 KB	10 ns
Main Memory	Semiconductor RAM	4-128 MB	50 ns
Magnetic Disk	Hard Disk	Gigabyte	10 ms, 10 MB/sec
Optical Disk	CD-ROM	Gigabyte	300 ms, 600 KB/sec
Magnetic Tape	Tape	100s MB	Sec-min, 10MB/min

Optical Memory ---CD (Compact Disk)



Introduction

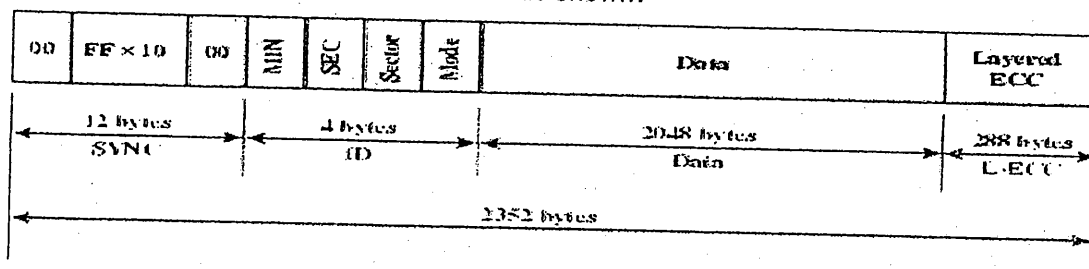
- CDs were introduced as **non-erasable disks** which could store upto **60 mins of music**.
- After their huge success, more development resulted into various types of optical storage disks.

Manufacturing/Writing

- The disk is formed from a **resin** such as **polycarbonate**.
- Digitally recorded information (Music or any general data) is **imprinted as a series of microscopic pits** on the surface of the polycarbonate.
- This is done with a finely focused, high-intensity **laser** to create a master disk.
- This master disk then creates a die to stamp out the polycarbonate, for copies of the same disk.
- The pitted surface is then coated with a **highly reflective surface** like **Aluminum** or **Gold**.
- The shiny surface is then protected from dust and scratches by a top coat of **clear acrylic**.

Reading

- Reading is done using a **low power laser** present inside the CD player/ CD ROM Drive.
- A motor spins the disk past a laser.
- A **photo-sensor** detects the **intensity of the reflected light** and decides whether it encountered a **pit or a land**. #Please refer Bharat Sir's Lecture Notes for this ...
- A **beginning/end of a pit** represents a **"1"**; **no change** in elevation is recorded as a **"0"**.
- Instead of storing data in concentric tracks, it is stored on **one large spiral** track starting from the center and spiraling out. This **increases the storage capacity**.
- Information is packed at **equal density** near the center and at the end of the disk.
- The disk is rotated at **slower speeds** as we move **more and more on the outer sides**.
- The format of data stored in a CD is as shown:



CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

CD - R (Recordable)

- Here the CD need not be recorded by the manufacturer, but instead, it can be recorded later by an end user.
- To make this possible, instead of creating physical changes (pits) on the resin surface, a different method is used. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- Here the **resin surface includes a dye** which can provide variable reflectivity.
- A laser "**programs**" the **dye to give different reflections** for 1s and 0s.
- Thereafter, the dye can be read as usual by a CD player.

CD - RW (Rewritable)

- This CD can be **re-written several times**.
- It used the concept of **Phase Change**.
- The **resin material** is such that it provides **two different reflections** in two different phases.
- A beam of **laser** can **change** the material from **one phase to another**.
- Hence the CD can be **re-written** several times.

DVD (Digital Versatile Disk)

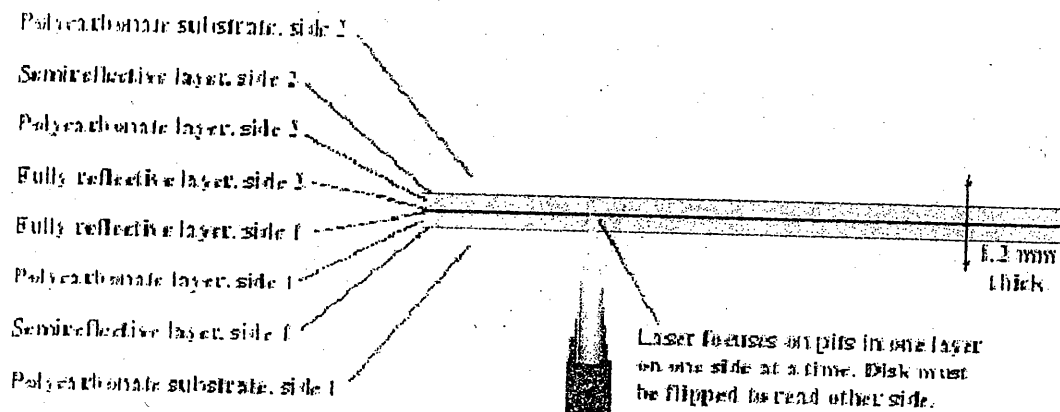


Fig. DVD-RW, double-sided, dual-layer - Capacity 17 GB

- In principal, a DVD is similar to CD.
- **Its advantage is that it can store much more data (max upto 17 GB).**
- This is possible due to the following three main reasons:

1) Bits are placed more closely on a DVD.

The distance between the spiral loop and between pits and lands is reduced. A **higher resolution laser** is used to read such compact data. This itself increases the storage capacity **seven times** more than a CD.

2) DVD employs two layers of pits.

The outer layer is semi transparent.

By **adjusting the focus** of the laser beam, the data can be read from the two layers independently. This further **doubles the storage capacity**.

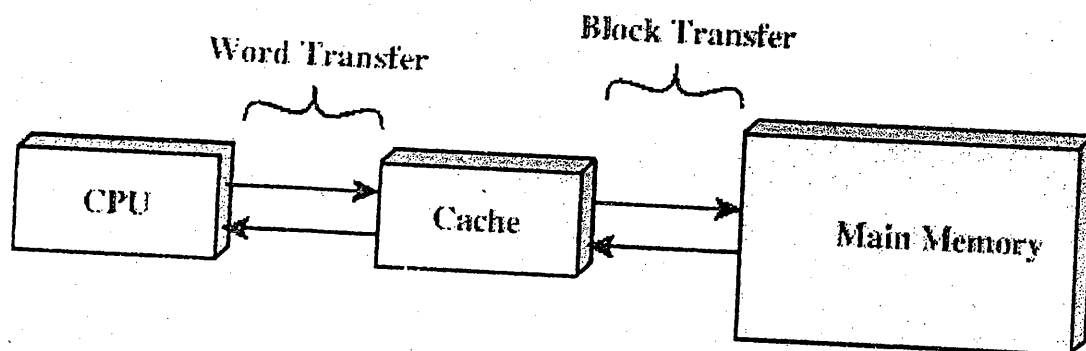
3) DVD ROM can be two sided.

Data can be stored on both sides of a DVD ROM, as opposed to only one side of a CD. This further **doubles the storage capacity** to a **max level of 17GB**.

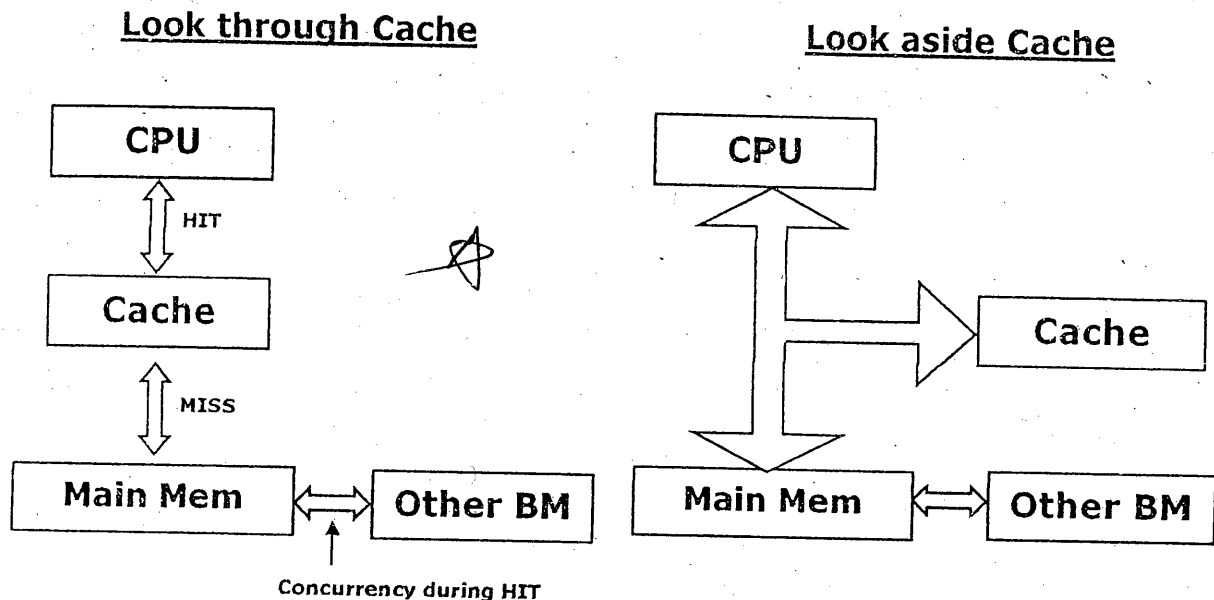
Similarly there are DVD -R (recordable) and DVD -RW (rewritable) like CD. But here only one-sided DVD can be used. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

CACHE MEMORY

- 1) Cache is a **high speed memory** introduced in the system to improve its performance.
- 2) As processors became faster (80386 CPU → 35 MHz), they required faster memory chips, otherwise memory operations would require the CPU to wait (Wait States).
- 3) Cache Memories were then introduced to **avoid these Wait States**.
- 4) **Cache** is implemented using **SRAM** (Static RAM) chips, while **main memory** uses **DRAM** (Dynamic RAM).
- 5) **SRAM is much faster** than DRAM but take a lot **more space**, is **more expensive** and is more **power consuming**. Hence only a **little amount of Cache** should be implemented in a system.
- 6) When the CPU wants to perform a memory operation, it **first checks** if the data is present in the **cache**. If found, it is called a **Cache Hit**. Now the operation is performed on the Cache itself.
- 7) If not found, it is called a **MISS**. Now the **Block** containing the data is **copied from the Main Memory to the Cache Memory**.
- 8) **Hit Rate = No of Hits / Total No of attempts**.
- 9) During a miss if there are no empty blocks in the Cache, then the same replacement policies such as FIFO, LRU, LFU etc are used. *#refer numericals solved in the class..*



CACHE ARCHITECTURES/ LAYOUT:



Look through Cache:

- 1) Here the CPU will **FIRST SEARCH** the **CACHE**, to find the required data.
- 2) **If found (HIT)**, the **Transfer** is performed **between the CPU and the Cache** Memory itself.
- 3) But if there is a **MISS**, then the CPU will **REPEAT** the **SEARCH**, now into the **Main Memory**.

Advantage:

During a **HIT**, as the transfer is between CPU and Cache Memory, the Main Memory is free to be used by other bus masters. This is called **Concurrency**. It improves the system performance.

Disadvantage:

During a **MISS**, time is wasted in searching the Cache Memory first and then searching Main Memory. This is called **Look-up Penalty**. It lowers system performance.

Usage: #Please refer Bharat Sir's Lecture Notes for this ...

It is preferred in **Multiprocessor systems**, due to the advantage of **Concurrency**.

Look aside Cache:

- 1) Here the CPU will **SIMULTANEOUSLY SEARCH** the **CACHE** and **Main Memory**, for the data.
- 2) Wherever found, the transfer will be performed from that memory.

Advantage:

During a **MISS**, there is **no Look-up Penalty** as the search was performed simultaneously.

Disadvantage:

As the CPU is always searching the Main Memory, the other bus masters cannot access the Main Memory at the same time. Hence **concurrency is not possible**.

Usage:

It is preferred in **Uniprocessor systems**, as it is **simple** and there is **no Look-up Penalty**.

CACHE CONSISTENCY

- When the **data** of a location in the **Cache** is **different** from the **data** of the same location in the **Main Memory** then the Cache is said to be **inconsistent**. Inconsistency should be avoided as it may result in **severe data bugs** in the system.
- Inconsistency may arise when the **CPU writes** into a location in the **Cache Memory**. Now if any **other Bus Master accesses** the **same data** from the **Main Memory**, it will get **old/stale** data.
- To avoid inconsistency, the following Write Policies may be used:
 - 1) Write through policy
 - 2) Write Back / Buffered Write / Delayed Write
 - 3) Snoopy Writes

1) Write through policy

- Here **whenever** the **CPU writes** into the **Cache Memory**, it will **also perform the write operation on the Main Memory**. Hence the Main Memory will always contain updated data.
- As an advantage there is a **high level of consistency**.
- Further, it is very **simple** to implement.
- But here, as a **disadvantage**, the **write operations will be slow** as the CPU writes into both the Cache Memory and the Main Memory.

2) Write Back / Buffered Write / Delayed Write

- Here **CPU only writes** into the **Cache Memory**.
- Additionally, there is a **Cache Controller** provided into the system.
- The Cache Controller **keeps track** of all locations that are **updated** in Cache Memory. This is done by maintaining a "changed bit" for each Cache block (very much like the dirty bit paging).
- Whenever** it finds that the **CPU is free**, or that the block will be removed from the cache, it **copies the updated contents from the Cache Memory to Main Memory**.
- The **advantage** here is that, since the CPU does only one write operation (CM), **Write operations are faster**. #Please refer Bharat Sir's Lecture Notes for this ...
- This has a **disadvantage** that the Main Memory does contain stale data, till it has been updated, which may happen after a long time. Hence there is a **low level of consistency**.

3) Snoopy Writes

- This is a **very efficient method** of Cache updating.
- Here the **Cache Controller snoops** (monitors) the **activities of other Bus Masters on the System Bus**. *If it finds that a Bus Master is trying to read a location from the Main Memory that has been modified in the Cache Memory, it will first copy the updated data from Cache Memory to Main Memory, and only then allow the Main Memory operation to continue.*
- The performance of such a system is very high as **no unnecessary updates** are performed. #Please refer Bharat Sir's Lecture Notes for this ...
- Moreover, the other **Bus Masters will always get updated data**.
- The only disadvantage is that, a **very advanced and expensive Cache Controller** would be required to perform this, which may increase the **cost** and complexity of the system. **Eg: With 80386 CPU, 80385 Cache Controller is required.**

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

CACHE ORGANIZATION / MAPPING TECHNIQUES {V IMP, 10-12 M}

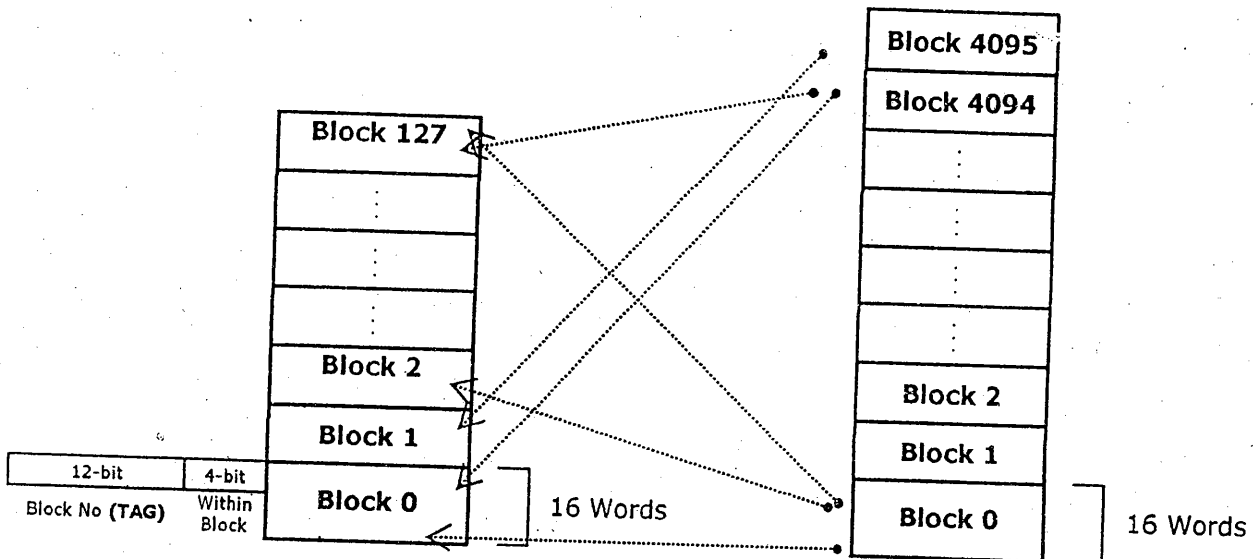
There are various ways in which, data from Main Memory can be "mapped" (brought) into Cache Memory. These are called Cache Mapping techniques.

Consider Main Memory = 64K words, Cache Memory = 2K words, Block Size = 16 Words.
{Note: 64K words means 64K locations}

I. ~~Full~~ Associative

Cache Memory (2K Words)

Main Memory (64K Words)



Organization: Here the 64K - Main Memory is divided into 4096 blocks each of 16 Words. The 2K - Cache Memory is divided into 128 blocks each of 16 Words.

Rule: In ~~Full~~ Associative Mapping, a Block of Main Memory can be mapped **ANYWHERE** in the Cache Memory. Hence the full Cache Memory is available for mapping. Eg: "Block 0" of Main Memory can come at any block of Cache Memory as shown above.

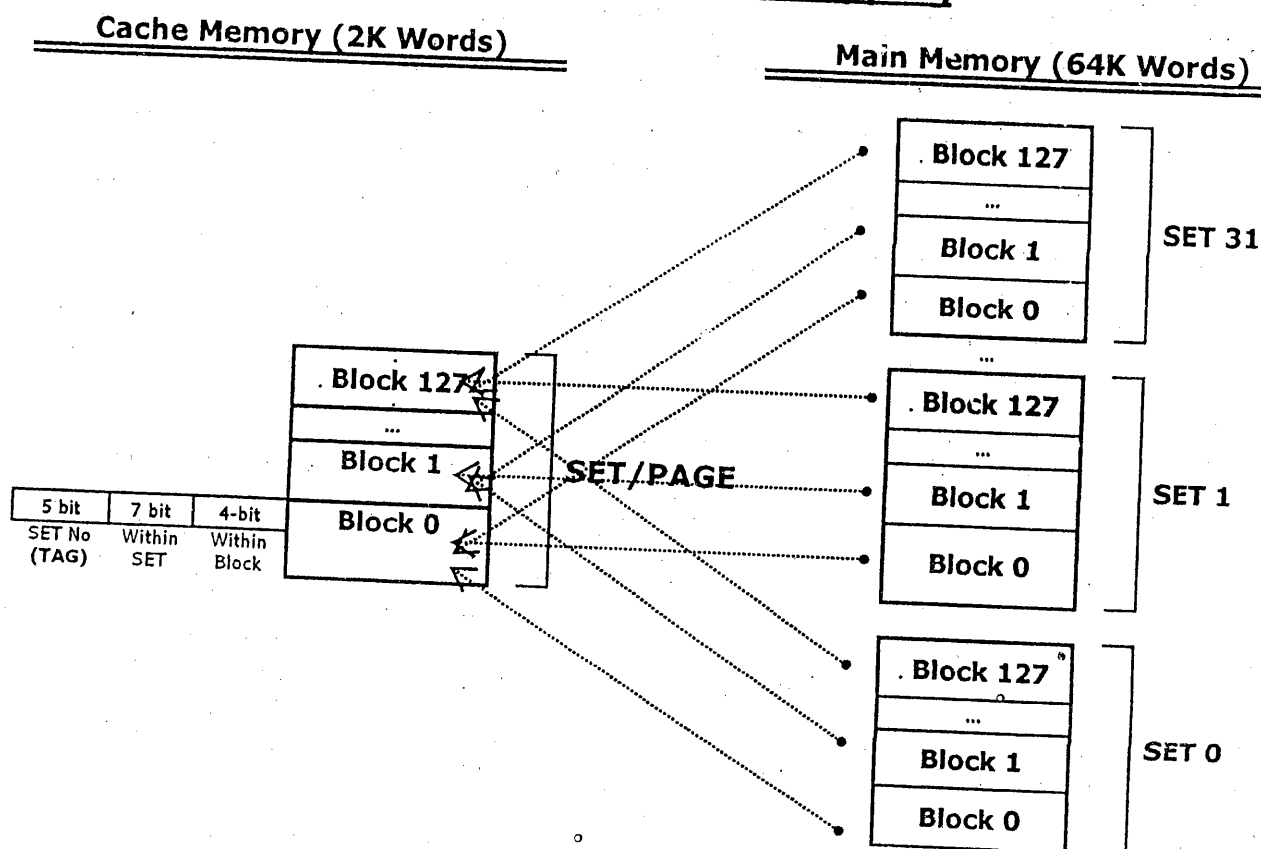
TAG: Every block in Cache Memory has a tag which identifies the block of Main Memory that is currently stored in that respective block of Cache Memory. Since a block in Cache Memory can contain **ANY** block of Main Memory, the tag size = 12-bit as $2^{12} = 4096$.

Search: Whenever the CPU needs to access a location, it searches in the Cache Memory. As the block could be located anywhere, it would have to search **ALL 128 blocks** of Cache.

Advantage: Generally gives a **high HIT RATE**, as the block could be mapped at any free location in Cache Memory. #Please refer Bharat Sir's Lecture Notes for this ...

Disadvantage: Tag size is large, and Search time is high as all blocks of Cache Memory have to be searched. Hence performance is poor.

II. One Way Set Associative / Direct Mapping



Organization: Here, the Cache Memory is considered a **SINGLE SET**, having 128 blocks. The Main Memory is divided into 32 such **SETS**, each SET having 128 Blocks. In both cases, a Block contains 16 words.

Rule: In Direct Mapping, Block "n" of any Set of Main Memory, can only be mapped at one place i.e. Block "n" of Cache Memory. Hence it is also called ONE-WAY set associative method. Eg: "Block 0" of any set of Main Memory, can only come at "Block 0" of Cache as shown above.

TAG: Since Block "n" of Cache Memory can only contain Block "n" of any set of Main Memory, we just need to identify which SET the block belongs to. #Please refer Bharat Sir's Lecture Notes for this ... Hence the TAG size is 5-bit as there are $2^5 = 32$ SETS in Main Memory.

Search: To search Block "n" of Main Memory, the CPU needs to search **ONLY** block "n" of Cache.

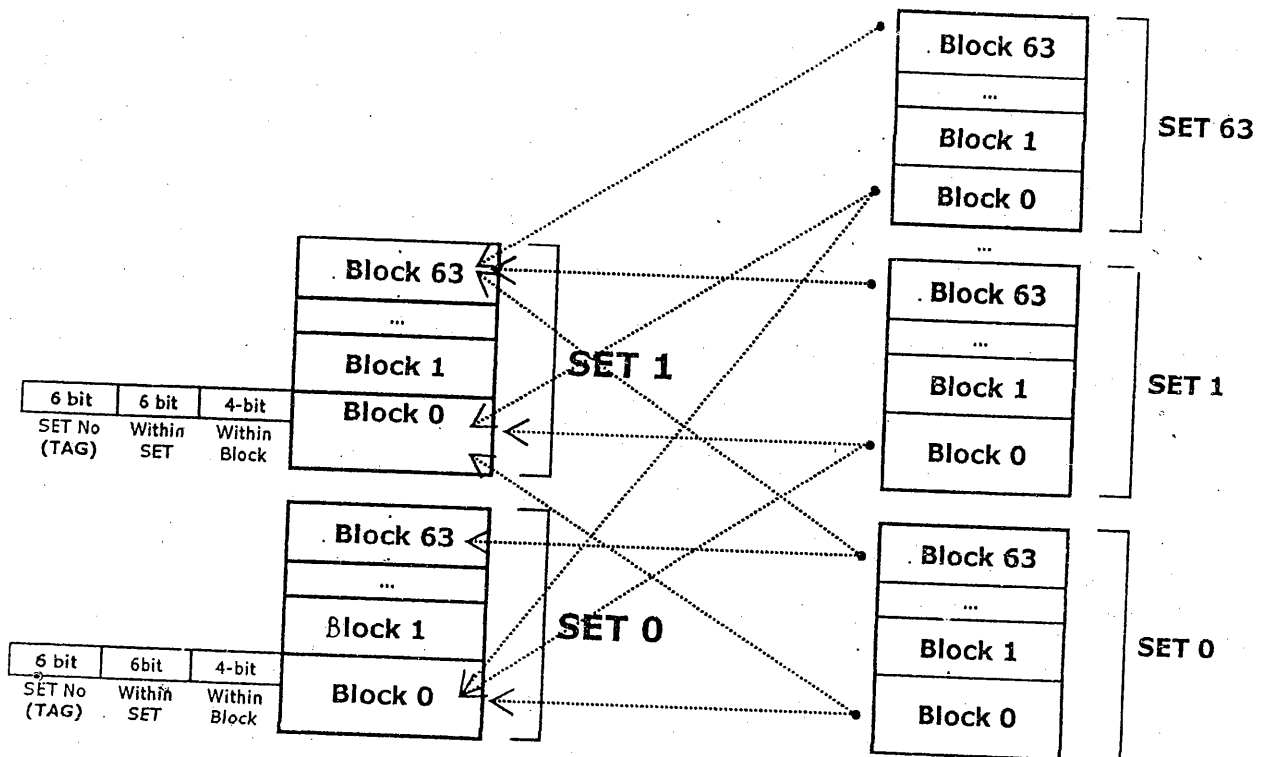
Advantage: Search is faster (Only one location). Tag Size is smaller (Only 5 bit SET Number).

Disadvantage: May give poor HIT Rate at times when Block "n" of several sets, are needed to be accessed together. Eg: If a program requires Block "0" of SET 0 and SET 1 repeatedly, it will have a poor HIT Rate, as Both Block 0's cannot be present in the Cache simultaneously, even if there are empty blocks in Cache Memory.

III. Two-Way Set-Associative

Cache Memory (2K Words)

Main Memory (64K Words)



Organization: Here, the **Cache Memory** is divided into **2 SETS**, each SET having **64 blocks**. The **Main Memory** is divided into **64 such SETS**, each SET having **64 Blocks**. In both cases, a **Block** contains **16 words**.

Rule: In **TWO-WAY SET** Associative Mapping, Block "n" of any Set of Main Memory, can only be mapped at **2 places** i.e. Block "n" of any of the **2 SETS** in **Cache Memory**. Hence it is also called **TWO-WAY** set associative method. Eg: "Block 0" of any set of Main Memory, can only come at "Block 0" of "SET 0" or "Block 0" of "SET 1" of Cache as shown above.

TAG: Since Block "n" of Cache Memory can only contain Block "n" of any set of Main Memory, we just need to **identify which SET** the block belongs to. ☺ In case of doubts, contact Bharat Sir: - 98204 08217. Hence the **TAG size is 6-bit** as there are $2^6 = 64$ SETS in Main Memory.

Search: To search Block "n" of Main Memory, the CPU needs to search at **two places**, i.e. Block "n" of SET 0 and Block "n" of SET 1.

Advantage: Search is fast (Only 2 locations). Tag Size is small (Only 6 bit SET Number).

Disadvantage: HIT Rate better than One-Way Set Associative, but still than Fully Associative. #Please refer Bharat Sir's Lecture Notes for this ...

IV. Four Way Set-Associative (Diagram not required)

Organization: Here, the **Cache Memory** is divided into **4 SETS**, each SET having **32 blocks**. The **Main Memory** is divided into **128 such SETS**, each SET having **32 Blocks**. In both cases, a **Block** contains **16 words**.

Rule: In **FOUR-WAY SET Associative Mapping**, Block "n" of any Set of Main Memory, can only be mapped at **4 places** i.e. Block "n" of any of the **4 SETS** in **Cache Memory**. Hence it is also called **FOUR-WAY set associative method**. Eg: "Block 0" of any set of Main Memory, can only come at "Block 0" of "SET 0" or "SET 1" or "SET 2" or "SET 3" of Cache.

TAG: Since Block "n" of Cache Memory can only contain Block "n" of any set of Main Memory, we just need to **identify which SET** the block belongs to. Hence the **TAG size is 6-bit** as there are $2^7 = 128$ **SETS** in Main Memory.

Search: To search Block "n" of Main Memory, the CPU needs to **search at 4 places**, i.e. Block "n" of **SET 0** or **SET 1** or **SET 2** or **SET 3**.

Advantage: Search is fast (Only 4 locations). **Tag Size is small** (Only 7 bit SET Number).

Disadvantage: **HIT Rate** better than **Two-Way Set Associative**, but still lower than **Fully Associative**.

CONCLUSION: Similarly, if we go on **increasing the number of ways**, we will get: **Better HIT Rate, more search time and bigger tag size...** till we **finally reach Fully Associative Technique**. This is shown below: #Please refer Bharat Sir's Lecture Notes for this ...

8 Way--- 8 Sets in CM having 16 Blocks --- 256 Sets in MM --- **Tag:8-bits** --- Search: **8 locⁿ**.
16 Way--- 16 Sets in CM having 8 Blocks --- 512 Sets in MM --- **Tag:9-bits** --- Search: **9 locⁿ**.
32 Way--- 32 Sets in CM having 4 Blocks --- 1024 Sets in MM --- **Tag:10-bits** --- Search: **10 locⁿ**.
64 Way--- 64 Sets in CM having 2 Blocks --- 2048 Sets in MM --- **Tag:11-bits** --- Search: **11 locⁿ**.
128 Way- 128 Sets in CM (**Fully Associative**)- 4096 Sets in MM - **Tag:12-bits** - Search: **12 locⁿ**.

INPUT/OUTPUT UNIT

Data transfers that take place using the I/O devices are generally **very large in nature**. To accommodate such large scale transfers many methods are used. Either we use a **dedicated I/O Processor** to take care of the I/O Transfer, or may have **multiple buses** in the system so that I/O transfers can take place parallel with other system activities. The communication between the CPU and the I/O device can be done in two ways:

	Serial Communication	Parallel Communication
1)	Data is transmitted bit-by-bit (one bit at a time)	More than one bits are transmitted together.
2)	Less costly	More costly.
3)	Less hardware is required	More hardware is required
4)	Transmission is slow	Transmission is fast
5)	Most suitable for long distance communication	Best used for local communication
6)	Eg: RS 232 Bus Standard	Eg: IEEE 488 bus standard

Further, Serial Communication can be performed in two ways:

	Synchronous Transfer	Asynchronous Transfer
1)	Here the sender and the receiver are synchronized using a clock signal transferred by the sender to the receiver.	Sender and receiver are not synchronized as no clock is transferred. Instead, the data is packed with start and stop bits .
2)	Consumes an extra line for the clock signal.	Clock is not used hence an extra line is not required .
3)	Provides higher data transfer rates (Baud Rate) as time is not wasted in start and stop bits.	Lower data transfer rate due to start and stop bits. <i>#Please refer Bharat Sir's Lecture Notes for this ...</i>
4)	The arrival of new data is indicated by pulses on the clock line.	The arrival of new data is indicated by start bit. The start bit is "0" and stop bit is "1". Thus after the previous data is transferred, the line would be high due to the stop bit. When the new data comes, first the line is pulled low by the start bit. This is how a new data is identified.
5)	Not suitable for very long distance communications as the clock may start going out of phase after a while.	Best suited for very long distance communication.
6)	More suited for block data transfers	More suited for character-wise data transfers

Q: Calculate transfer rate in char/sec for serial comm. Having 7 data bits, 1 parity bit and 2 stop bits at bit rate of 600 bits/sec, for the following 2 formats:

1>

7 data bits	1 parity bit	2 stop bits
----------------	-----------------	----------------

2>

2 synch bits	Char 1 7 bit	Char 2 7 bit	Char 3 7 bit	...	Char 100 7 bit
-----------------	-----------------	-----------------	-----------------	-----	-------------------

..... May 2004 (10 m)

Soln: ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

First format:

Bit Rate = 600 bits/sec
1 Char = 10 bits
Transfer Rate = (600 bits/sec) / (10 bits/char)
= 60 Char/sec

Second format:

Bit Rate = 600 bits/sec
Total bits required to transfer 100 char = 700 + 2 = 702 bits.
Total time required to Transfer 100 char = 702/600 = 1.17 sec.
Total time required to Transfer 1 char = 1.17/100 = 0.0117 sec.
Hence in 1 second we can transfer = 1/0.0117 char.

Hence transfer rate = 85.47 char/sec.

CHOPRA ACADEMY

Computer Organization
and Architecture

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

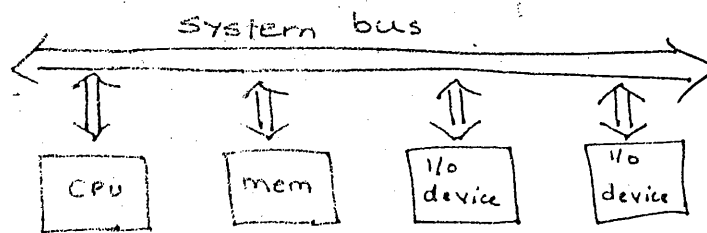
Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

BUS INTER-CONNECTION SCHEMES

Different I/O systems use different kind of Bus Inter-Connection Schemes as shown below:

1) Single Bus Scheme

- Here, both the CPU and the I/O module (I/O processor) share the same bus.
- Such a multiprocessor system is also called "Closely Coupled" system.



Advantages

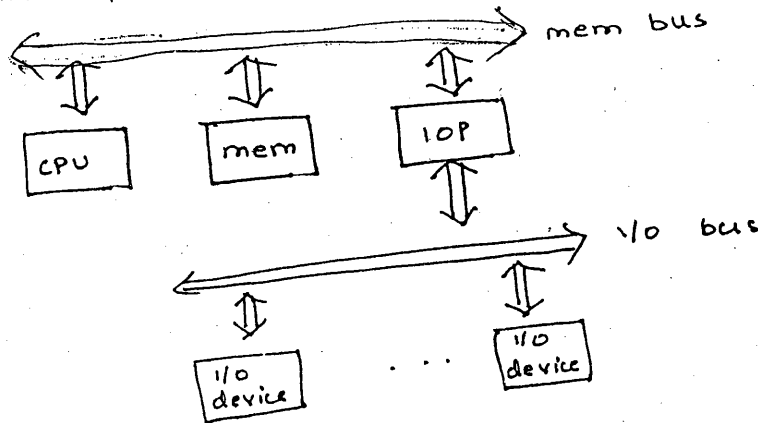
- Such systems are **cheap** and **easy** to implement.
- They are best suited for a **uni-processor** environment.

Disadvantages

- In multiprocessor systems, having only one bus would lead to **poor performance** as only one processor can be the bus master at a time. So when **one processor** is using the bus, the **other processor** will have to **wait**. #Please refer Bharat Sir's Lecture Notes for this ...
- **Reliability** of the system is **very poor**. If the bus fails then the entire system will stop working.

2) Dual Bus Scheme

- Here, both the CPU and the I/O module (I/O processor) **have individual buses**.
- The **CPU** carries out its **memory operations** on the **system bus**, and the **I/O Processor** has a **dedicated I/O bus** for its **I/O operations**.
- Such a multiprocessor system is also called "**Loosely Coupled**" system.



Advantages

- They are best suited for a **multi-processor** environment.
- **Performance** of the system is **better** as both processors can use their buses **simultaneously**.
- **Reliability** of the system is **better**.
Even if the I/O Bus fails, the CPU can carry out its operations on the system bus.

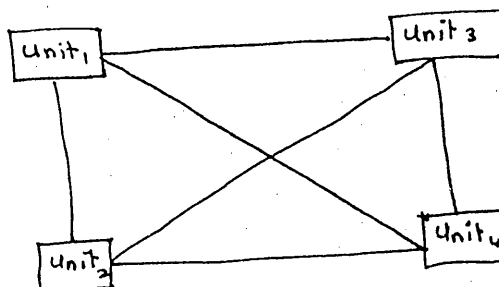
Disadvantages

- **Cost** of the system is **high**.
- System is **complex** to implement.

3) Multiple Bus Scheme

- This is used in systems which have a **lot of processors**.
- **Each processor** is provided with a **dedicated bus** and all processors are interlined on a network.
- Such systems are extremely **powerful** and **fault resistant**, but are **costly** and **complex**.
- The various ways of connecting them are: **Dedicated links**, **Cross Bar**, **Hypercube** etc.

Dedicated Links:



CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

Computer Organization
and Architecture

Notes by: BHARAT SIR
BharatSir@hotmail.com
Cell: 98204 08217

Hypercube:

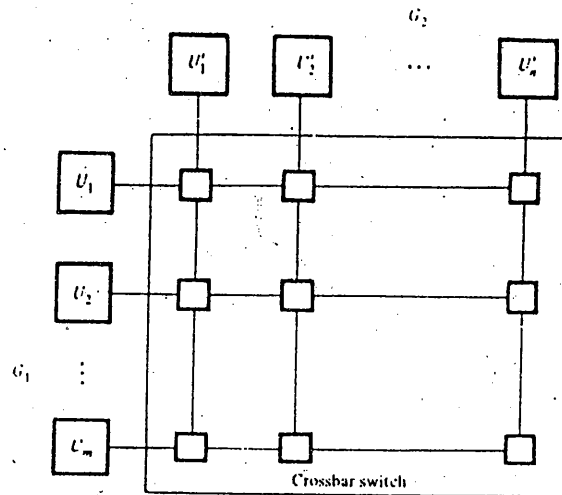
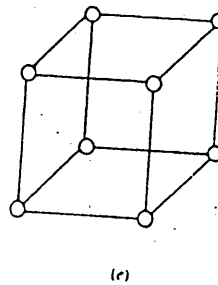


FIGURE 6.7
Crossbar connection of two groups of units.



Ring-Structure:

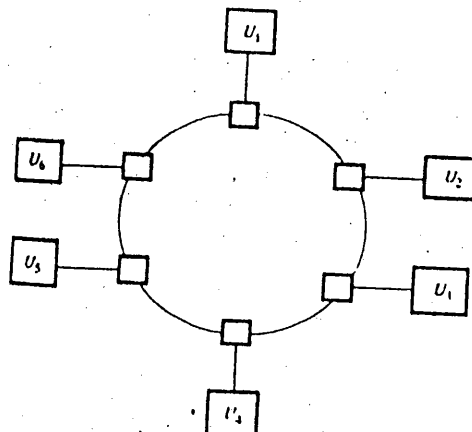


FIGURE 6.8
A ring-structured communication network.

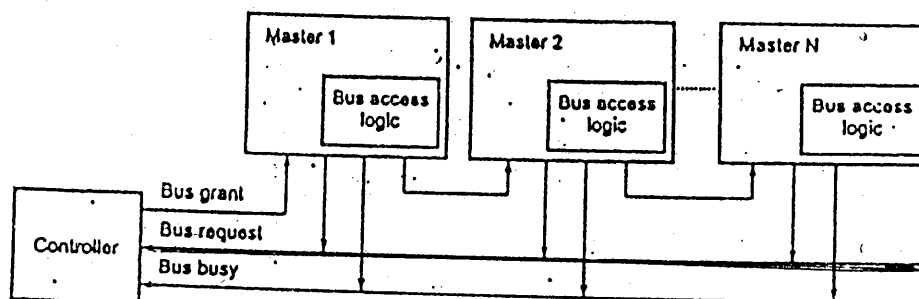
COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

BUS CONTENTION / ARBITRATION / PRIORITY RESOLVING {V Imp}

- In multiprocessor system (like the LAN network in Your College), there are more than one processors, that require control of the system bus at a time.
- Hence an appropriate priority resolving mechanism is required, to decide which processor should get control the bus (become bus master).
- This is called **bus contention**.
- The following **three methods** are used to resolve bus contention:

A) Daisy Chain Method



- All bus masters use the same line for Bus Request.
- If the Bus Busy line is inactive, the Bus Controller gives the Bus Grant signal.
- Bus Grant signal is propagated serially through all masters starting from nearest one.
- The bus master, which requires the system bus, stops this signal, activates the Bus Busy line and takes control of the system bus. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.

Advantage:

- i. Design is simple.
- ii. The number of control lines is less. Also adding new bus masters is easy.

Disadvantage:

- i. Priority of bus masters is rigid and depends on the physical proximity of the bus masters with the bus arbiter i.e. The one nearest to the Bus Arbiter gets highest priority.
- ii. Bus is granted serially and hence a propagation delay is induced in the circuit.
- iii. Failure of one of the devices may fail the entire system.

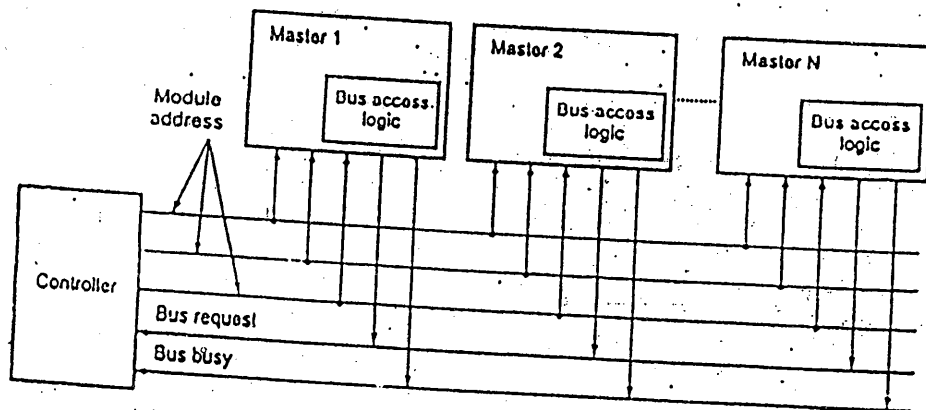
CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

B) Polling Method



- Here also all bus masters use the same line for Bus Request.
- Here the controller generates binary address for the master.
Eg: To connect 8 bus masters we need 3 address lines ($2^3 = 8$).
- In response to a Bus Request, the controller "polls" the bus masters by sending a sequence of bus master addresses on the address lines. Eg: 000, 010, 100, 011 etc
- When a requesting master recognizes its address, it activates the Bus Busy line and takes control of the bus. #Please refer Bharat Sir's Lecture Notes for this ...

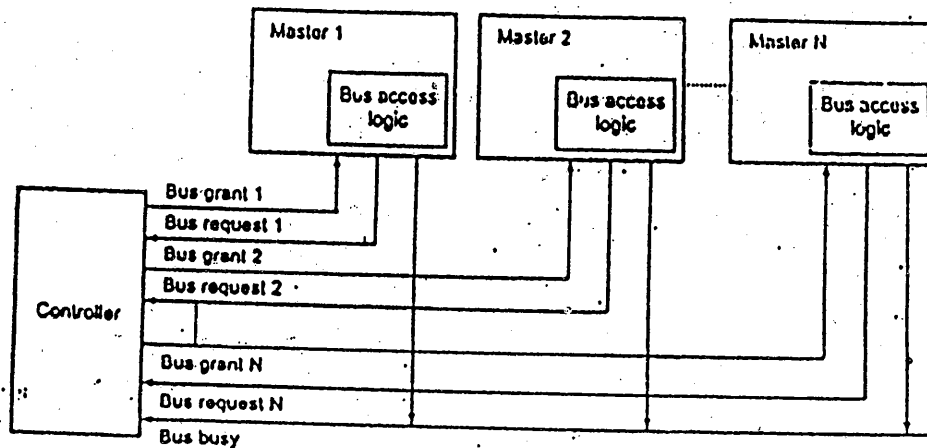
Advantage:

- i. The Priority is flexible and can easily be changed by altering the polling sequence.
- ii. If one module fails, the entire system does not fail.

Disadvantage:

- i. Adding more bus masters is difficult as increases the number of address lines of the circuit. Eg: In the above circuit to add the 9th Bus Master we need 4 address lines.

C) Independent Request Method



- Here, all bus masters have their individual Bus Request and Bus Grant lines.
- The controller thus knows which master has requested, so bus is granted to that master.
- Priorities of the masters are predefined so on simultaneous Bus Requests, the bus is granted based on the priority, provided the Bus Busy line is not active.
- The Controller consists of encoder and decoder logic for the priorities.

Advantage:

- i. The Bus Arbitration is fast.
- ii. The speed of Bus Arbitration is independent of the number of devices connected.

Disadvantage:

- i. The number of control lines required is more ($2n$ line required for n devices).
Hence connecting a large number of bus masters is difficult.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

I/O DATA TRANSFER TECHNIQUES:

There are various ways in which data can be transferred to and from an I/O device. These mainly differ in the level of involvement of the CPU.

I. PROGRAM CONTROLLED DATA TRANSFER

- This kind of transfer is **completely controlled by the CPU** through a user written program.
- The I/O transfer is performed simply by a **set of instructions**, written by the user.
- The key element here is that the user should have the exact knowledge of **when the transfer should be performed** otherwise time could be wasted in unnecessary attempts if the device is not ready for the transfer. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- Eg: The user gives a command to **send data to an I/O device**.
The CPU will inform the I/O device and **check if the I/O device is ready**.
If the device is **NOT READY**, the CPU will have to **WAIT** for the device to get ready.
Once the device is ready, the transfer is completed.

Here the I/O devices could be mapped in the system in two ways:

Memory Mapped I/O:

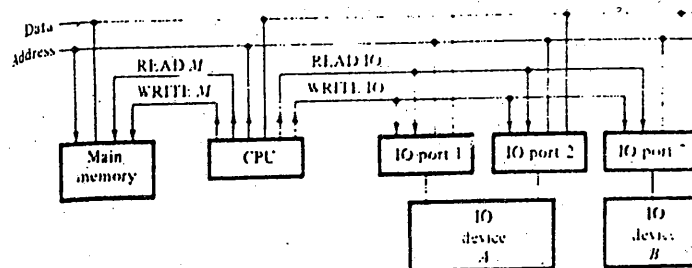


FIGURE 6.34
Programmed IO with separate memory and IO address space (IO-mapped IO).

I/O Mapped I/O:

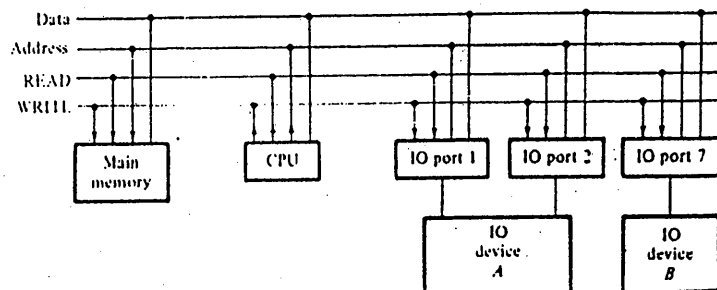


FIGURE 6.33
Programmed IO with shared memory and IO address space (memory-mapped IO).

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

	Memory Mapped I/O	I/O Mapped I/O
1)	The I/O devices are mapped into the memory address space, i.e. I/O devices are assigned memory addresses.	The I/O devices are mapped into the I/O address space, i.e. I/O devices are assigned I/O addresses.
2)	The size of the I/O address is the same as the size of the memory address.	The size of the I/O address is independent of the memory address.
3)	Increases the number of I/O ports interfaced. Eg: 65536 i.e. 64 K in 8085	Restricts the number of I/O Ports interfaced. Eg: 256 in 8085.
4)	Only MEMR, MEMW signals are required as CPU treats even I/O devices as memory devices	IOR, IOW, MEMR, MEMW signals are required as CPU treats I/O and Memory devices separately.
5)	Data transfer can take place between any register and an I/O device	Data transfer can take place only between the Accumulator and an I/O device.
6)	Most of the instructions can be used to access the I/O devices	Only IN and OUT instructions can be used to access the I/O devices.
7)	Execution is slower as access time is more	Execution is faster as access time is less. #Please refer Bharat Sir's Lecture Notes for this ...
8)	Decoding 16-bit address requires more hardware	Decoding 8-bit address requires less hardware

I. INTERRUPT DRIVEN DATA TRANSFER

- An interrupt is a condition that makes the CPU suspend the current program and execute an ISR.
- An ISR (Interrupt Service Routine) is a specially written program to service the condition the caused the Interrupt.
After an interrupt is serviced, the CPU returns the main program, exactly at the next instruction from where it had left.
- In interrupt driven data transfer, whenever the I/O device is ready for the data transfer, it will interrupt the CPU. #Please refer Bharat Sir's Lecture Notes for this ...
- In the ISR, the CPU will perform the data transfer.
- This method is better than the previous method, as here the CPU does not have to waste time in checking the status of the I/O device.
- Eg: Instead of the CPU checking when data is available on the keyboard, the keyboard should interrupt the CPU when a key is pressed. Thus time will not be wasted in repeatedly checking the keyboard when the user is not typing at all.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105

Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

Interrupts have the following characteristics:

1) Software Interrupts:

- These interrupts are **invoked** by the programmer using instructions.
- They **cannot be disabled**.
- They are **always vectored** i.e. they have a fixed address of their respective ISR.
- Eg: for 8086 CPU "**INT N**" is a software interrupt instruction.

2) Hardware Interrupts:

- These interrupts **occur through pins** which are connected from external devices to the CPU.
- Whenever a **device needs service** it **sends an Interrupt** to the CPU.
- The CPU will now **execute the ISR** to service the interrupt.
- They **can be disabled**.
- They may be **vectored or non-vectored** i.e. their ISR address may not be fixed.
- Eg: for 8086 CPU "**NMI**" is a vectored hardware interrupt and "**INTR**" is non-vectored.

3) Vectored Interrupts:

- These interrupts have a **fixed address** of ISR.
- Since the ISR address is fixed, they are **rigid** and hence **cannot be used to accept multiple interrupts**.
- Since the ISR address is fixed, they are **executed faster** as **no time is wasted** in obtaining the ISR address from the external device.
- They **do not require an acknowledge** signal. ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- Eg: **NMI** interrupt of 8086.

4) Non- Vectored Interrupts:

- These interrupts do not have a **fixed address** of ISR.
- When a device interrupts on such a line, the CPU sends an acknowledgement.
- Now the device which sent the interrupt will provide the ISR address to the CPU.
- Since the ISR address is not fixed, they are **flexible** and hence **can be used to accept multiple interrupts**. Thus the **interrupt structure** can be **expanded** on non-vectored interrupts.
- Since the ISR address is not fixed, they are **executed faster** as **no time is wasted** in obtaining the ISR address from the external device. #Please refer Bharat Sir's Lecture Notes for this ...
- They **require an acknowledge** signal to obtain the ISR address.
- Eg: **INTR** interrupt of 8086 is non vectored. It has an acknowledgement signal called **INTA**.

5) Maskable Interrupts:

- These interrupts **can be disabled** by the programmer.
- They have **lower priority** than the non-maskable interrupts.

6) Non-Maskable Interrupts:

- These interrupts **cannot be disabled** by the programmer.
- They have **higher priority** than the maskable interrupts.

7) Multi-Line Interrupts:

- Here **different devices** interrupt on **different lines**.
- **Identifying** which device has interrupted is **easy**.

8) Single-Line Interrupts:

- Here **different devices** interrupt on a **single line**. #Please refer Bharat Sir's Lecture Notes for this ...
- **Identifying** which device has interrupted is **NOT easy**. This is done by either **checking each device** (polling), or by using an **Interrupt Controller Chip** like **8259 PIC**.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

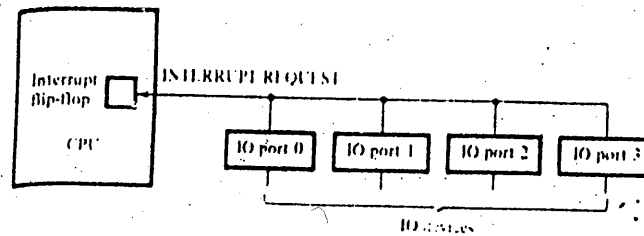


FIGURE 6.43
Single-line interrupt system.

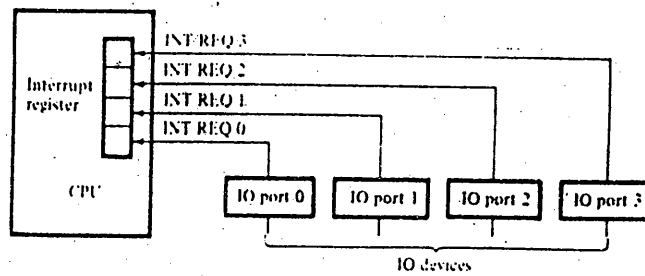
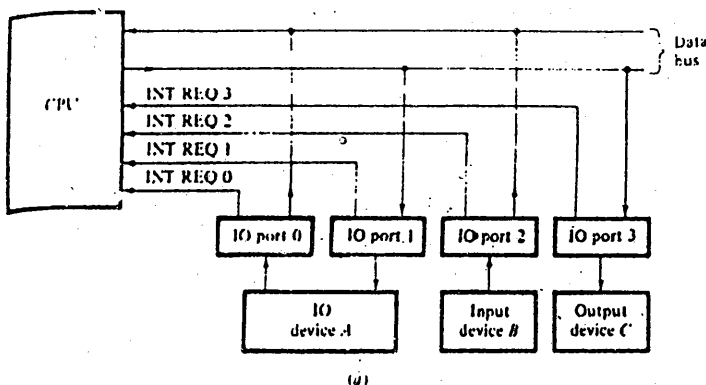
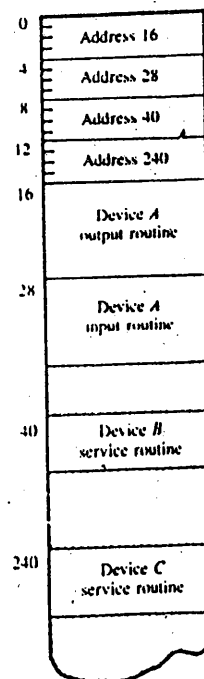


FIGURE 6.44
Multiple-line interrupt system.



(a)



(b)

FIGURE 6.46
(a) A system with vectored IO interrupts. (b) Location of the interrupt servicing programs in main memory.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION.

Notes by: Bharat Sir
Cell: 98204 08217

III. DMA CONTROLLED DATA TRANSFER

CONCEPT OF DMA

- DMA Transfer is a **hardware controlled I/O Transfer technique**.
- A device known as the DMA Controller (Eg: 8237 DMAC) is responsible for the DMA transfer.
- It is an efficient method for **high-speed data transfer between I/O and Memory**.
- In Program Controlled I/O or Interrupt driven I/O the speed of transfer is **slow** mainly because **Instructions** need to be decoded and then executed for the transfer.
- In DMA transfers, the **DMA Controller, becomes the bus master**, and performs the entire transfer **without using any instructions/program**.
- Since DMA transfer is **software independent** it is much faster.
- To perform DMA transfers, the **DMAC has the following**:
 - a) **DREQ** (Data Request)
 - b) **HRQ** (Hold Request)
 - c) **HLDA** (Hold Acknowledgement)
 - d) **DACK** (Data Acknowledgement)
 - e) **Address Register**
 - f) **Count Register**

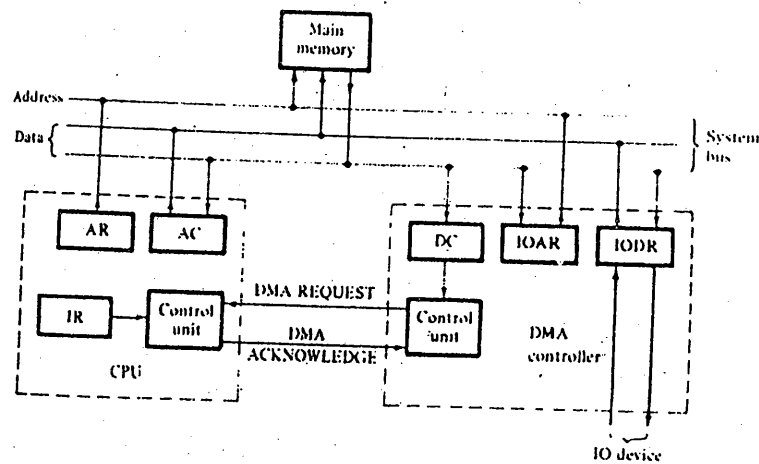


FIGURE 6.42
Circuitry required for direct memory access (DMA).

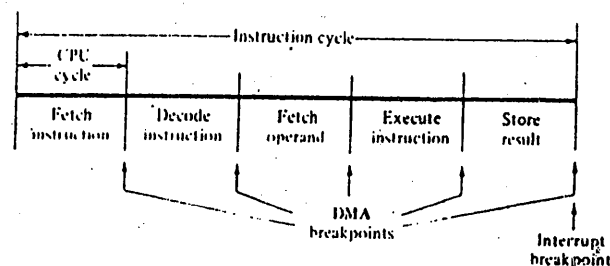


FIGURE 6.41
DMA and interrupt breakpoints during an instruction cycle.

CHOPRA ACADEMY

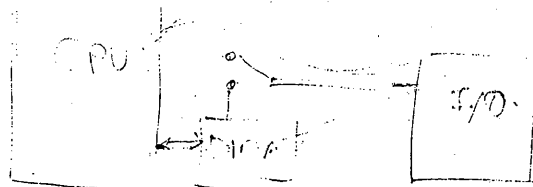
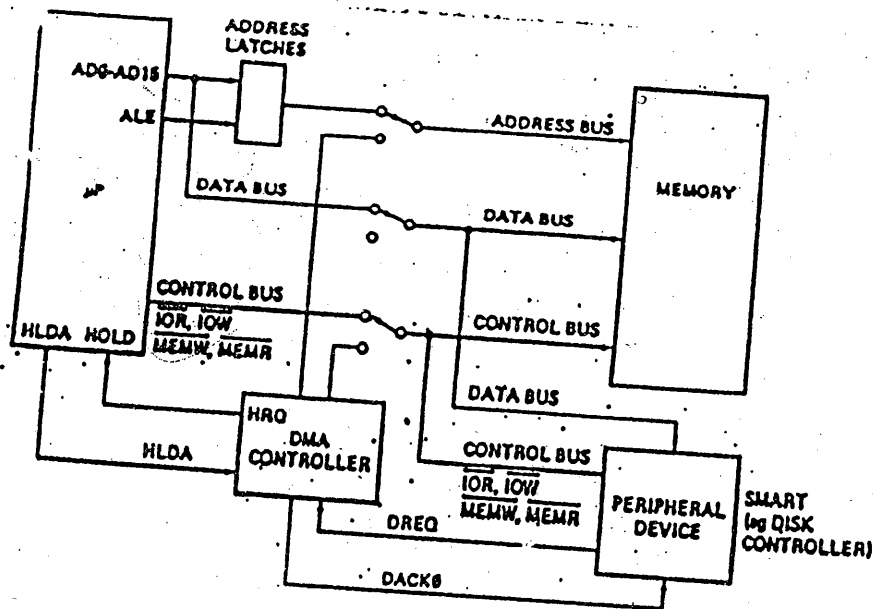
Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

Steps for DMA transfer:

- 1) The CPU sends the starting address of the transfer and the number of bytes to be transfers in the address and count registers of the DMA controller. Now the DMAC waits for the I/O device to get ready.
- 2) I/O device sends the **DREQ** signal to the **DMAC**, to indicate it is ready for the transfer.
- 3) The **DMAC** in turn sends a request signal to the **μP**, through the **HRQ (HOLD) line**.
- 4) The CPU finishes the current machine cycle and releases the system bus (gets disconnected from it). ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
It also acknowledges receiving the HOLD signal through the **HLDA line**.
- 5) The **DMAC** acquires control of the system bus.
The **DMAC** sends the **DACK** signal to the I/O peripheral and the **DMA transfer begins**.
- 6) After every byte is transferred the address register is automatically incremented and the count register is decremented.
- 7) Transfer is stopped when count becomes zero (TC - Terminal Count)
- 8) Now, the system bus is released by the **DMAC**.
The CPU takes control of the system bus and continues its operation.



CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105

Bandra: 2655 9366 / 2642 4106.

Ban
N

COMPUTER ORGANIZATION

Notes by: Bharat Sir

Cell: 98204 08217

Types of DMA Transfers:

1) Block Transfer (Burst Transfer)

- In this mode, once the **DMAC** gets the system bus, it transfers all the required bytes and only then returns the bus back to the CPU.
- This is the **fastest method** of DMA Transfer.
- However, the **CPU** remains idle for a very long time.

2) Cycle Stealing (Single Byte Transfer)

- In this mode, once the **DMAC** gets the system bus, it transfers only one byte and returns the system bus to the μP .
- Thereafter for each byte request is made for the system bus and thus the operation continues.
- Thus, if **10 bytes** are to be transferred, the **DMAC** requests and subsequently gets the system bus **10 times**. Each time it gets the system bus it transfers only one byte.
- It is a **slower method** of DMA Transfer. *#Please refer Bharat Sir's Lecture Notes for this ...*
- However, the **CPU** remains idle for a small amount of time (time required for transferring only one byte).

3) Demand Transfer

- It is very similar to Block Transfer, except that the **DREQ** must active throughout the DMA operation.
- If during the operation **DREQ** goes low, the DMA operation is stopped and the busses are returned to the μP . *#Please refer Bharat Sir's Lecture Notes for this ...*
- In the meantime, the **CPU** can continue with its own operations.
- Once **DREQ** goes high again, the DMA operation continues from where it had stopped.

4) Transparent (Hidden Mode)

- There are clock states between machine cycles when the CPU does not use the address or data bus.
- During these states, these buses contain unspecified data (eg: T_4 , T_5 , T_6 of opcode fetch).
- During these states, the **DMAC** uses the system bus to transfer one or more bytes.
- The **CPU** is completely unaware of the transfer and hence the name.
- **DMAC** can also transfer data during the **IDLE** machine cycles of the μP .
- This is a **very slow** method of DMA transfer.
- However, as the CPU is unaware of the DMA transfer the CPU does not remain idle at all.

COMPUTER ORGANIZATION.

Notes by: Bharat Sir

Cell: 98204 08217

IV. DATA TRANSFER USING I/O PROCESSOR (I/O CHANNEL)

- This is the **most powerful method** of data transfer.
- Here, a **dedicated I/O processor** is used to carry out I/O Transfers.
- Thus the **CPU is free** to perform any other type of operation.
- This makes the system very **efficient**.
- For best results, such a system should have a **separate I/O bus** for the I/O processor.
- Eg: With 8086 CPU, 8089 I/O Processor is used.

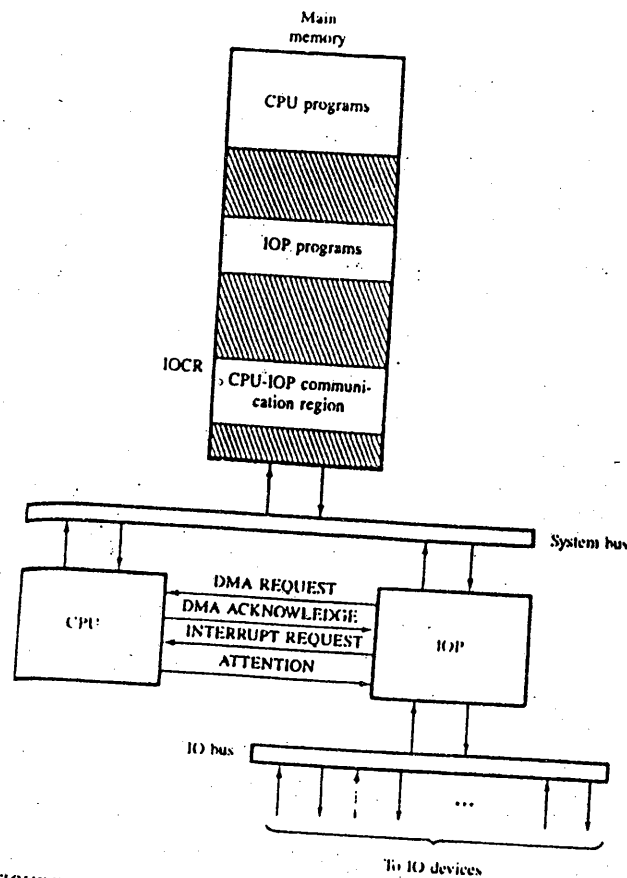


FIGURE 6.52
Representative system containing an IOP.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

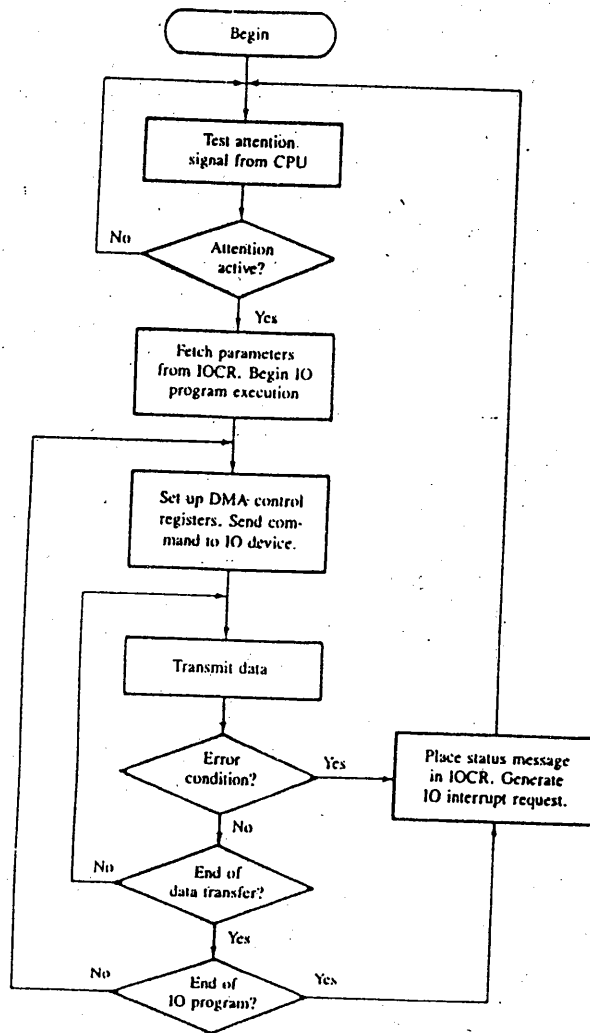


FIGURE 6.53
Behavior of a typical IOP.

Steps for I/O Processor data transfer:

- 1) All that the IOP needs is a message from the CPU containing transfer parameters like (address and count etc). ☺ In case of doubts, contact Bharat Sir: - 98204 08217.
- 2) The CPU stores the message at a predefined memory location in the CPU-IOP communication region as shown above. #Please refer Bharat Sir's Lecture Notes for this ...
- 3) Now the CPU will activate the CA (Channel Attention) line to trigger the IOP.
- 4) Now the IOP will take control of the bus to perform the I/O transfer.
- 5) Firstly, the IOP will read the message to get the transfer parameters.
- 6) Then it will perform the I/O transfer as per the message.
- 7) Once the transfer is completed, the IOP will store an appropriate message in the CPU IOP Communication region and will indicate it to the CPU by sending an interrupt.
- 8) If an error was encountered, the IOP will store a corresponding message for it in the memory.
- 9) This is how I/O transfers are performed using an IOP.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

- Q: Assume CPU operating at 25 MHz.
Read and write operations take 2 cycles each.
DMA Processing time is 12 cycles.
Find max data transfer rate for Programmed I/O and DMA Controlled data transfer.
- (May 2000 - 8m)

Soln:

Programmed I/O:

$$\begin{aligned} 1 \text{ Transfer} &= 1 \text{ Read} + 1 \text{ Write} \\ &= 2 + 2 \\ &= 4 \text{ cycles.} \end{aligned}$$

$$1 \text{ Transfer} = 4 \times \left(\frac{1}{25 \text{ MHz}} \right) = 4 \times \frac{1}{25} \times 10^{-6} = 0.16 \times 10^{-6}$$
$$= 160 \text{ nsec.}$$

Hence in 1 Sec

$$\begin{aligned} (\text{transfer rate}) &= 1/160 \text{ nsec} \\ &= \underline{\underline{6.25 \text{ M Bytes/sec.}}} \end{aligned}$$

$$= 160 \times 10^{-9} = \underline{\underline{160 \text{ nsec}}}$$

$$\begin{aligned} \therefore \text{in 1 sec} &= 6250000 \\ &= 6.25 \times 10^6 \\ &= \underline{\underline{6.25 \text{ MBytes/sec}}} \end{aligned}$$

DMA Controlled data transfer:

Initialization for DMAC takes 12 cycles.

Assuming that the count register is 16 bits, the max block size = 2^{16} Bytes = 64 KBytes.

Time required to

$$\begin{aligned} \text{Transfer 64 K Bytes} &= 12 + 64 \text{K Cycles} \\ &= (12 + 64 \text{K})/25 \text{MHz} \end{aligned}$$

Time required to

$$\text{Transfer 1 Byte} = \{(12 + 64 \text{K})/25 \text{MHz}\} / (64 \text{K}) \text{ sec}$$

Hence in 1 Sec

$$\begin{aligned} (\text{transfer rate}) &= (64 \text{K}) / \{(12 + 64 \text{K})/25 \text{MHz}\} \\ &= \underline{\underline{24.99 \text{ Mbytes/sec.}}} \end{aligned}$$

✓ 9.2 Flynn's Classification of Parallel Processing Systems

A typical central processing unit (CPU) operates by fetching instructions and operands from main memory, executing the instructions, and placing the results in the main memory. The steps associated with the processing of an instruction form an instruction cycle. The instruction can be viewed as forming an instruction stream

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

FLYNN'S CLASSIFICATION OF MULTIPROCESSOR SYSTEMS {IMP}

flowing from main memory to the processor, while the operands form another stream, the data stream, flowing to and from the processor, as shown in Fig. 9.1.

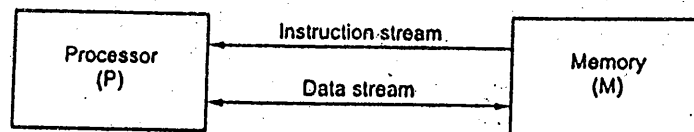


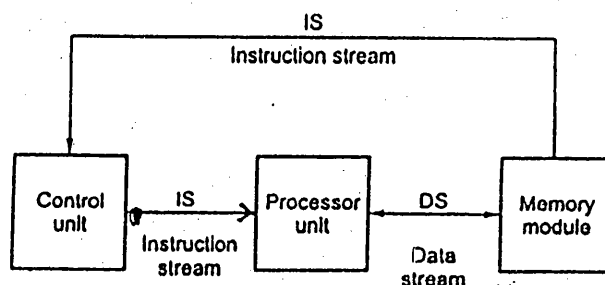
Fig. 9.1 Instruction and data streams in a simple computer

In 1966, Michael J. Flynn has made an informal and widely used classification of processor parallelism based on the number of simultaneous instruction and data streams seen by the processor during program execution. Suppose that a processor P is operating at its maximum capacity, so that its full degree of parallelism is being exhibited. Let m_i and m_d denote the minimum number of instruction and data streams, respectively, that are being actively processed. m_i and m_d are termed the instruction and data-stream multiplicities of P , and measure its degree of parallelism. Note that m_i and m_d are defined by the minimum, instead of the maximum number of streams flowing at any point, since the most limiting components of the system (its bottlenecks) determine the overall parallel processing capabilities.

The classification made by Michael J. Flynn divides computers into four major groups based on the values of m_i and m_d associated with their CPUs.

- Single instruction stream-single data stream (SISD).
- Single instruction stream-multiple data streams (SIMD).
- Multiple instruction streams-single data stream (MISD).
- Multiple instruction streams-multiple data streams (MIMD).

1. Single instruction stream single data stream (SISD) : $m_i = m_d = 1$. Most conventional machines with one CPU containing a single arithmetic-logic unit capable only of scalar arithmetic fall into this category. SISD computers and sequential



COMPUTER ORGANIZATION.

Notes by: Bharat Sir
Cell: 98204 08217

computers are thus synonymous. In SISD computers instructions are executed sequentially but may overlap in their execution stages (Pipelining). They may have more than one functional unit, but all functional units are control by a single control unit.

2. Single instruction stream multiple data stream (SIMD) : $m_i = 1$, $m_D > 1$. This category corresponds to array processors. They have multiple processing/execution units and one control unit. Therefore, all processing/execution units are supervised by the single control unit. Here, all processing elements received same instruction from control unit but operate on different data sets from distinct data streams.

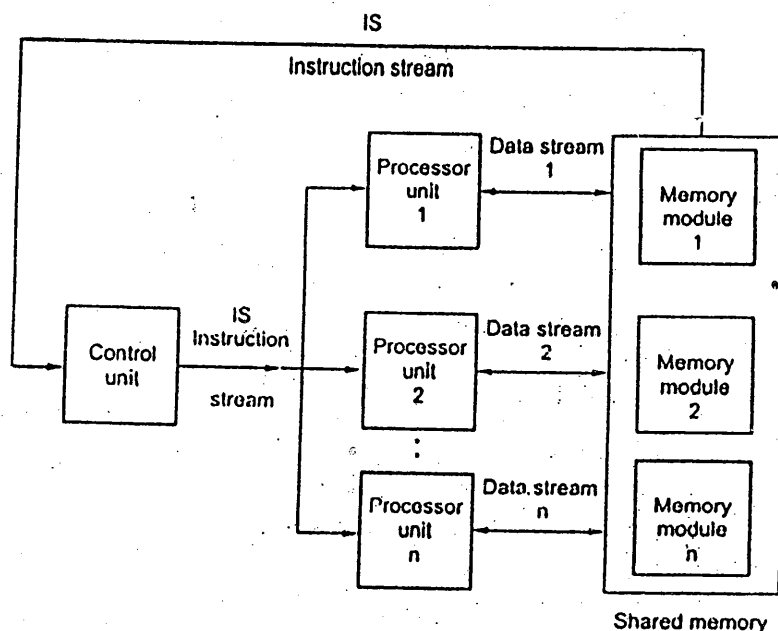


Fig. 9.3 SIMD computer

3. Multiple instruction stream single data stream (MISD) : $m_i > 1$, $m_D = 1$. Not many parallel processors fit well into this category. In MISD, there are n processor units. Each receiving distinct instructions operating over the same data stream and its derivatives. The results of one processor become the input of the next processor in the micropipe. The fault-tolerant computers where several processing units process the same data using different programs belongs to the MISD class. The results of such apparently redundant computations can be compared and used to detect and eliminate faulty results.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105

Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir

Cell: 98204 08217

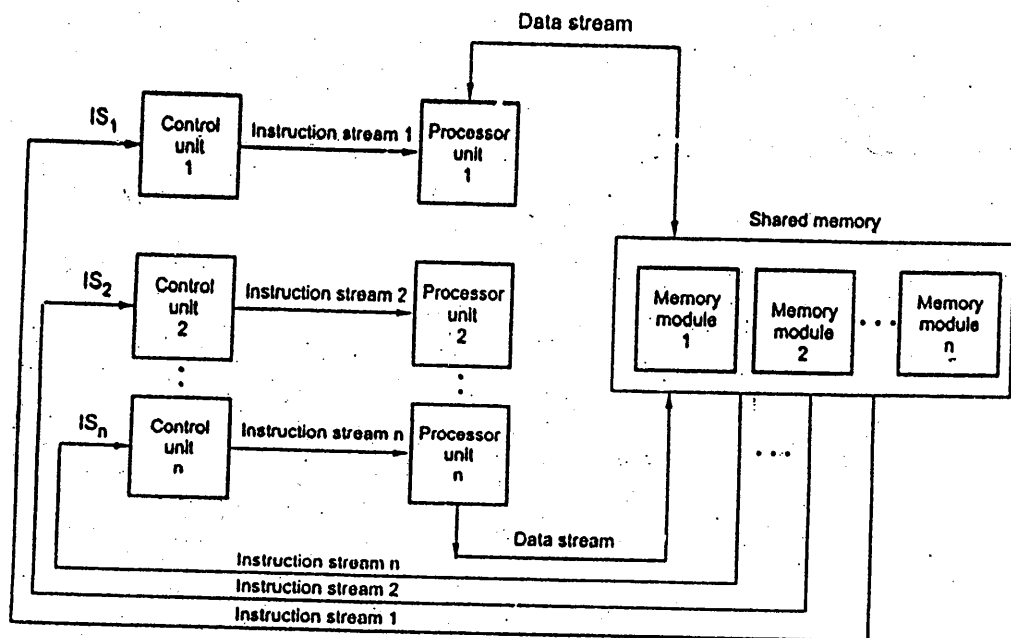
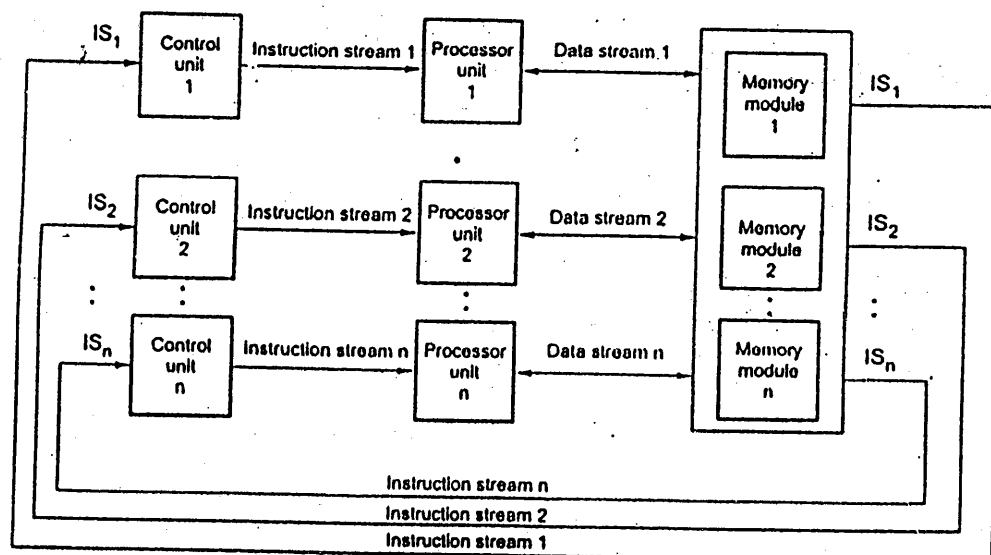


Fig. 9.4 MISD computer

4. Multiple instruction stream multiple data stream (MIMD) : $m_i > 1$, $m_n > 1$. Most multiprocessors system and multiple computers system can be classified in this category. In MIMD, there are more than one processor unit having the ability to



execute several programs simultaneously.

CHOPRA ACADEMY

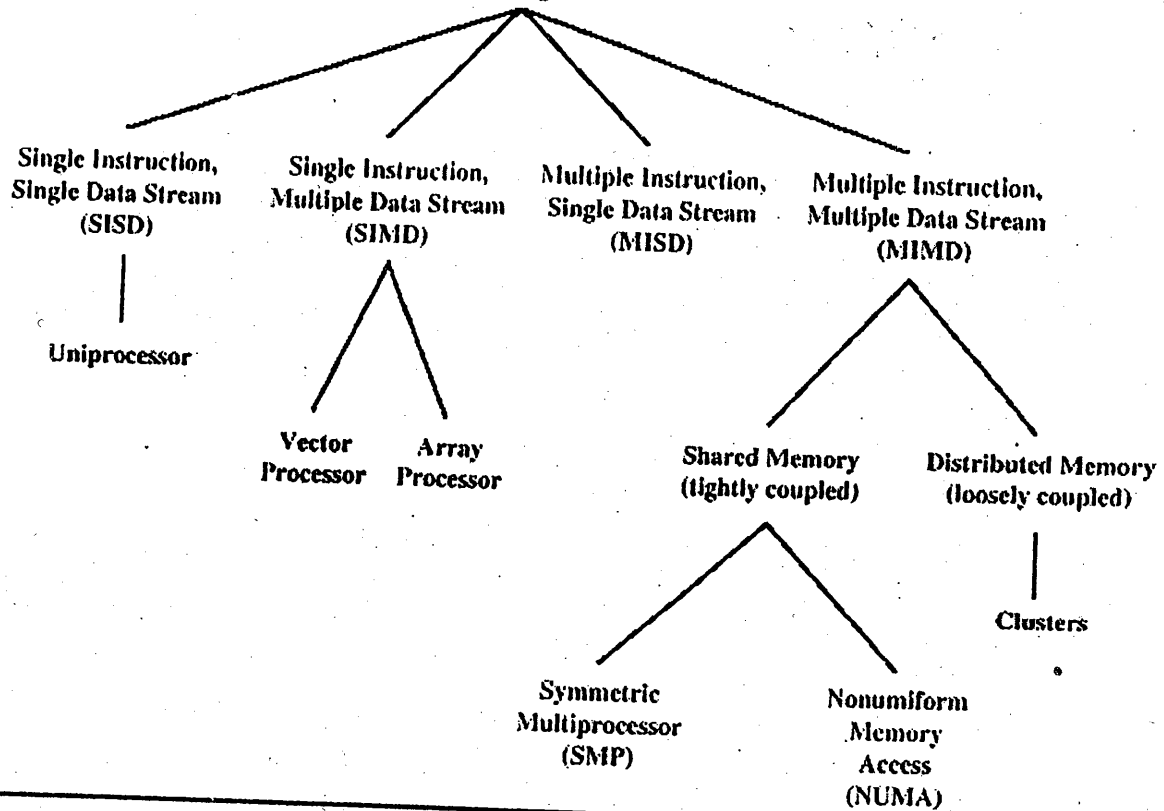
Mulund : 98201 20744 Vashi : 5591 1105

Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

Processor Organizations



CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

5

COMPUTER ORGANIZATIOC

Notes by: Bharat Sir
Cell: 98204 08217

PCI BUS STRUCTURE {IMP}

8.3 PCI Bus Structure

In early 1992, Intel formed another industrial group with the same goals as the VESA group in relation to the PC bus. The main intension behind the formation of group was to overcome the weaknesses in the ISA and EISA buses. They designed PCI (Peripheral Component Interface) specifications in June 1992, and updated in April 1993.

8.3.1 Features

1. It is designed to economically meet the I/O requirements of modern systems. It requires very few chips to implement and support other buses attached to the PCI bus.
2. It bypasses the standard I/O bus, uses the system bus to increase the bus clock speed and take full advantage of the CPU's data path.
3. It has an ability to function with a 64-bit data bus.
4. It has high bandwidth. The information is transferred across the PCI bus at 33 MHz, at the full data width of the CPU. When the bus is used in conjunction with a 32-bit CPU, the bandwidth is 132 Mbytes/sec. It is calculated as follows :
$$33 \text{ MHz} \times 32\text{-bit} = 1,056 \text{ M bits/sec}$$
$$1,056 \text{ M bits/sec} \div 8 = 132 \text{ M bytes/sec}$$
5. PCI bus is designed to support a variety of microprocessor based configurations including both single and multiprocessor systems.
6. The PCI bus can operate concurrently with the processor bus. The CPU can be processing data in a external cache while the PCI bus is busy transferring information between other parts of the system.
7. The PCI bus is processor - independent bus that can function as a mezzanine , or peripheral bus.
8. It makes use of synchronous timings and centralized arbitration scheme.
9. It delivers better system performance for high - speed I/O subsystems (e.g. graphic display adapters, network interface controllers, disk controllers and so on).
10. The PCI interface contains a 256 bytes configuration memory which allows the computer to interrogate the PCI interface. This feature allows the system to automatically configure itself for the PCI plug-board and hence it is referred to as plug-and-play.

8.3.2 PCI Configurations

As mentioned earlier, PCI bus is designed to support single processor as well as multiprocessor systems. Fig. 8.49 (a) shows a typical use of PCI in a single processor system.

Notice that the processor bus is separate and independent of the PCI bus. The processor connects to the PCI bus through an integrated circuit called a PCI bridge. The memory controller and PCI bridge provides tight coupling with the processor and delivers data at high speeds.

CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105
Bandra: 2655 9366 / 2642 4106.

COMPUTER ORGANIZATION

Notes by: Bharat Sir
Cell: 98204 08217

(Traditional Bus architecture)

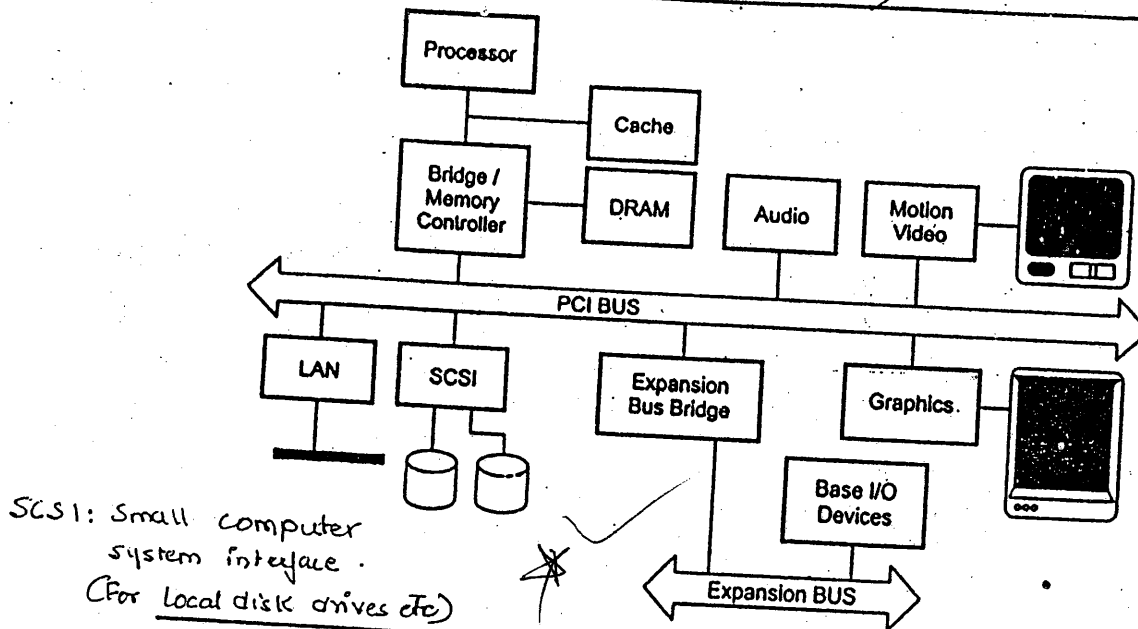
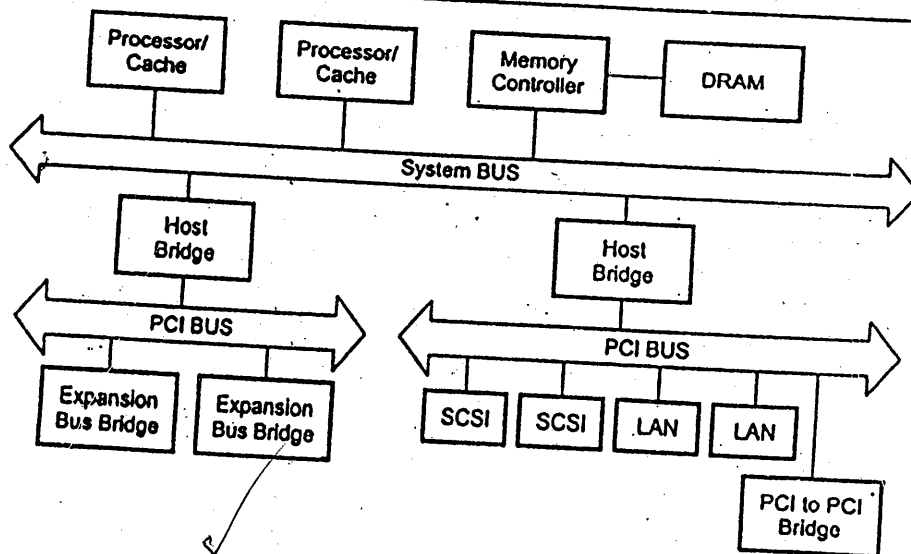


Fig. 8.49 (a) Conceptual diagram of PCI bus for single processor system

Fig. 8.49 (b) shows a typical use of PCI in a multiprocessor system. As shown in the Fig. 8.49 in multiprocessor systems one or more PCI configurations may be connected by bridges to the processor's system bus. Again, the use of bridges keeps the PCI independent of the processor speed yet provides the ability to receive and deliver data rapidly.



CHOPRA ACADEMY

Mulund : 98201 20744 Vashi : 5591 1105

Bandra: 2655 9366 / 2642 4106.

PCI Bus structure

Mandatory pins

- (i) System pins : V_{CC} , GND , CLK , $reset$.
- (ii) Add. data pins : carrying address & data in a time multiplexed form. & signals to demux them
- (iii) Interface pins control : generate control signals.
- (iv) Arbitration pins :- for arbitration amongst PCI devices.
- (v) Error reporting pins :- Report parity & errors.

Optional Pins

- (i) Int. pins :- To receive interrupt from ^{PCI} ~~other~~ devices.
- (ii) Cache Support pins :- To support onboard cache mem. Allows Snoopy cache protocol.
- (iii) Boundary scan pins :- Support testing procedures defined in IEEE 1149.1 std.
- (iv) 64bit Expansion pins :- Additional A/D lines for expansion from 32bit to 64bit.

