

Atmiya Infotech

*Computer
Organization
&
Architecture*

ATMIYA

Ketan Solanki

● Syllabus	5
● Bit 4	5
● BCA	6
● MCA	7
● Digital Logic Circuits	9
● Digital Computers	9
● Computer Organization, Computer Design, Computer Architecture	10
● Logic Gates	10
● Boolean Algebra.....	12
● Basic Identities of Boolean Algebra	12
● De-Morgan's Theorem	13
● Complement of a function.....	14
● Map Simplification	14
● Variable Maps	15
● Sum-of-Products Simplification.....	15
● Product-of-Sums Simplification.....	17
● Don't Care Conditions	19
● Combinational Circuits	19
● Half-Adder.....	20
● Full-Adder	20
● T Flip-Flop.....	23
● Sequential Circuits	25
● Design Procedure	25
● Integrated Circuits.....	27
● Integrated Circuits	27
● Decoders	28
● Encoders.....	31
● Registers	33
● Shift Registers.....	33
● Memory Unit	34
● Random-Access Memory (RAM).....	35
● Read-Only Memory (ROM).....	35
● Central Processing Unit	37
● The CPU	37
● General Register Organization.....	37
● Memory Stack.....	41
● Three-Address Instructions	43
● Two-Address Instructions	44
● One-Address Instructions	44
● Zero-Address Instructions	44
● Data Transfer and Manipulation ♣.....	46
● Data Transfer Instructions	46
● Data Manipulation Instructions	47
● Logical and Bit Manipulation Instructions	48

● Shift Instruction	49
● Program Control ♣	50
● Status Bit Conditions	51
● Conditional Branch Instructions	51
● Types of Interrupt	52
● RISC(Reduced Instruction Set Computer) ♣	52
● Input-Output Organization	54
● Peripheral Devices.....	54
● ASCII Alphanumeric Characters.....	55
● Input-OutPut interface.....	57
● I/O Bus and Interface Modules	57
● I/O versus Memory Bus.....	59
● Isolated versus Memory Mapped I/O	59
● Example of I/O Interface	60
● Asynchronous Data Transfer.....	61
● Strobe control.....	62
● Handshaking	63
● Asynchronous Serial Transfer	66
● Asynchronous Communication Interface	68
● First In First Out Buffer.....	70
● Modes of transfer.....	72
● Example of Programmed I/O.....	73
● Interrupt-initiated I/O	74
● Priority Interrupt.....	74
● Daisy Chaining Priority	75
● Parallel Priority Interrupt	77
● Priority Encoder	79
● Interrupt Cycle	79
● Software Routines	80
● Initial And Final Operations.....	81
● Direct Memory Access.....	82
● DMA Controller	83
● DMA Transfer.....	84
● Input-Output Processor.....	86
● CPU-IOP Communication	87
● Serial Communication♣.....	89
● Character-Oriented Protocol.....	91
● Transmission Example	92
● Memory Organisation	94
● Main Memory.....	94
● Memory Address Map.....	97
● Memory Connection to CPU.....	98
● Associative Memory.....	99
● Hardware Organization.....	100
● Cache Memory.....	101

● Associative Mapping	101
● Direct Mapping.....	102
● Virtual Memory.....	103
● Address Space and Memory Space.....	104
● Associative Memory Page Table	105
● Page Replacement.....	105
● Bibliography	107
● BCA-Question Papers	108
● Bca-Apr/May2001	108
● BCA-AprMay2000.....	110
● BIT Question Papers	112
● Bit-MarApr2002	112
● BIT-MarApr2001	113
● MCA-Question Paper	115
● M.C.A. November – 1999	115



ATMIYA

Syllabus**Bit 4**

CS-20- Computer System Organization and Architecture		
Digital Logic Circuits	Logic gates Boolean Algebra Map Simplification Combinational Circuits Universal Gates PLA Comparator Flip-Flops Sequential Circuits	20
Digital Components	Decoders Encoders Multiplexers Demultiplexers Registers Parity generator and checker Shift Registers Binary Counters	20
Memory Organization	Main Memory Associative Memory Cache Memory Virtual Memory	20
Input - Output Organization	Input-Output Interface Asynchronous Data Transfer Modes of Transfer Priority Interrupt DMA IOP Serial Communication	20
Central Processing Unit	General Register Organization Stack Organization Instruction Formats Addressing Modes Data Transfer and Manipulation Program Control RISC	20



ATMIYA

BCA

CS-8- Introduction to Computer Organization and Architecture		
Digital Logic Circuits	- Logic gates - Boolean Algebra - Map Simplification - Combinational Circuits - Universal Gates - Flip-Flops - Sequential Circuits	25
Digital Components	- Decoders - Encoders - Multiplexers - Demultiplexers - Registers	20
Memory Organization	- Main Memory - Associative Memory - Cache Memory	15
Input - Output Organization	- Input-Output Interface - Asynchronous Data Transfer: - Modes of Transfer - DMA - IOP	20
Central Processing Unit	- General Register Organization - Stack Organization - Instruction Formats - Addressing Modes	20

MCA

Paper no.: 104 Computer organization and assembly language programming

Basic computer architecture and peripherals

Components of personal computer - ALU, registers, control unit, memory, internal & external bus, I/O controllers & peripheral devices.

Digital circuits

Basic digital circuits - arithmetic circuits, encoder, decoder, Multiplexers, Demultiplexers, comparator, Half adder, full adder, binary adder subtractor, parity generator & checker programmable logic array (PLA), integrated circuits, sequential circuits concepts, flip-flops, registers, shift registers & counters, 8 bit left-right shifter.

Memory organization

An introduction, CPU - memory interaction, storage technology, semiconductor memory cell, memory organization, 2D memory array.

Microprocessor chips and buses

General concepts of microprocessor chips, examples of microprocessor chips, like 80386, 80486 etc., computer buses, synchronous & asynchronous buses, bus arbitration, examples of buses, interfacing with I/O devices, programmed I/O, interrupt driven I/O & DMA controllers.

An assembly language programming

Segments and addressing, registers, assembling linking & executing a program, defining & moving data, program logic & control jump, conditional jump etc., I/O instruction for key-board & screen, arithmetic operations and screen operations.

Advanced computing

Introduction to multiprogramming, principle of pipelining parallel processing, multiprocessors, fault-tolerant computers, CISC & RISC architecture.

Dear Students,

As we know that computers have now become a part of our routine life. The smart work that it performs with a given instruction is worth exploring. It may be a question in your mind as you keep yourself thinking that what kind of process is going on in a computer when any instruction you give. What happens inside it? When I was a student I always thought that what makes the computer so smart that it gives us such outputs? Well that is the thing that we are going to explore with this subject. As per the name of our subject **Computer Organization and Architecture** let us first differentiate the term Organization and Architecture.

Computer Organization is concerned with the way the hardware components are connected together to form a computer system.

Computer Architecture is concerned with the structure and behavior of the various functional modules of the computer and how they interact to provide the processing needs of the user.

This notes provides you the basic knowledge necessary to understand the hardware operation of digital computers this notes are commonly prepared for the students of BCA, BIT and MCA-I. .

There are some instructions for you to better follow before you proceed. As the notes are common, the topics that doesn't fall in your syllabus are denoted by the a graphical symbol following by the your stream for eg. ♣ Stands for the BCA students, which means that the particular topic is not in BCA syllabus. Apart from this the question papers of previous exams, examples, the reference materials and name of the related sites are included at the end.

Best Of Luck.

ATMIYA

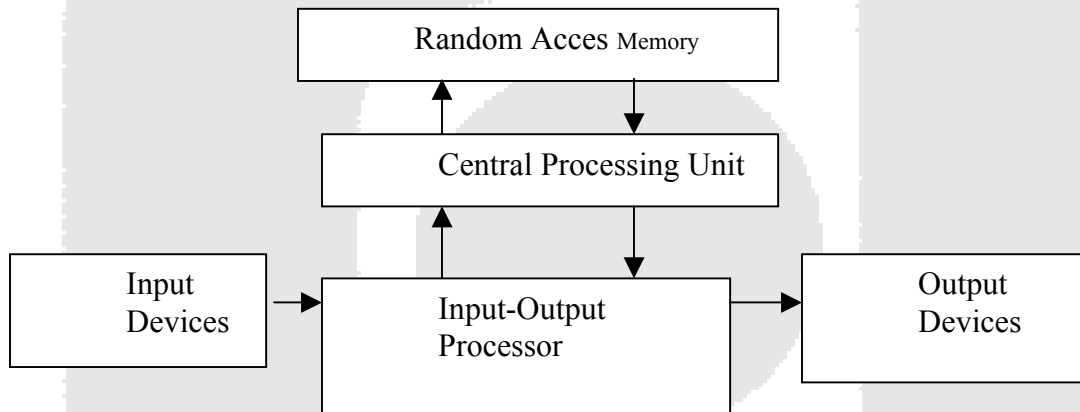
Digital Logic Circuits

This chapter introduces the fundamental knowledge needed for the design of digital systems constructed with the individual gates and flip – flops. It covers Boolean algebra, combinational circuits and sequential circuits. This provides the necessary background for understanding the digital circuits to be presented.

Digital Computers

- Digital computers use the binary number system, which has two digits, 0 and 1.
- A binary digit is called a *bit*.
- Bits are grouped together as *bytes* and *words* to form some type of representation within the computer.
- A sequence of instructions for the computer is known as *program*.

Block diagram of a digital computer (BIT Apr2001, Apr2002) (*Bit-Mar02*)



The hardware of the computer is usually divided into three major parts.

- The Central processing Unit (CPU) contains an arithmetic and logic unit for manipulating data, a number of registers for storing data, and control circuits for fetching and executing instructions.
- The memory of a computer contains storage for instructions and data, it is called a Random Access Memory (RAM) ,the CPU can access any location in memory at random and retrieve the binary information within a fixed interval of time.
- The input and output processor contains electronic circuit for communication and controlling the transfer of information between the computer and the outside world.
- The input and device connected to the computer include keyboards, printers, terminals, magnetic disk drives and other communication devices.

Computer Organization, Computer Design, Computer Architecture

- *Computer Organization* is concerned with the way the hardware computer operate and the way they are connected together to form the computer system. The various components are assumed to be in place and the task is to investigate the organizational structure to verify that the computer parts operation as intended.
- *Computer Design* is concerned with the hardware design of the computer. Once the computer specification is formulated, it is the task of the designer to develop hardware for the system. Computer Design is concerned with the determination of what hardware should be used and how the parts should be connected. This aspect of computer hardware is sometimes referred to as computer implementation.
- *Computer Architecture* is concerned with the structure and behavior of the computer as seen by the user. It includes the information formats, the instruction set and techniques for addressing memory.

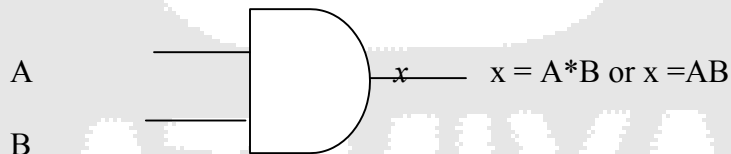
Logic Gates

Bca-Aug99, (Bit-may01) (Bit-Mar02)

- Binary information is represented in digital computers using electrical signals.
- These signals can be represented by voltage to specify one of two possible states. For example, if a wire contains a signal of 3 volts, it is considered to contain the digital value 1.
- Likewise, if the wire contains 1.5 volts, then it represents the digital value 0.
- The manipulation of binary information in a computer is done using logic circuits called *gates*.

The gates are:

AND

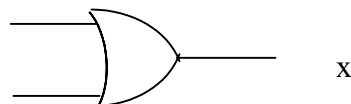


A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

OR

A

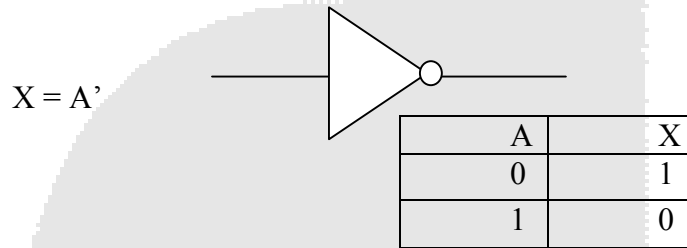
B



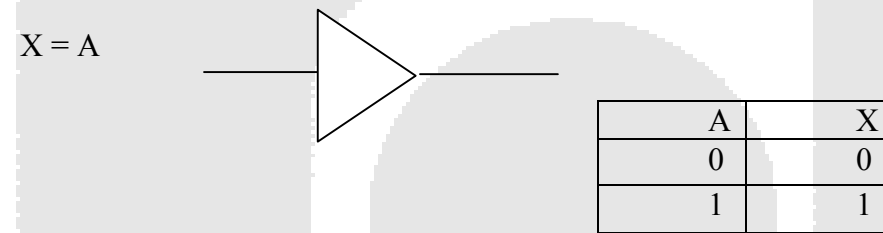
$X = A + B$

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

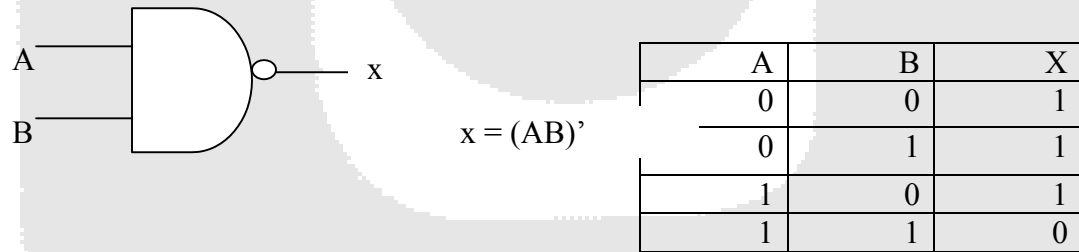
Inverter



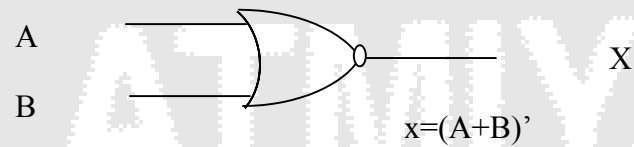
Buffer



NAND



NOR

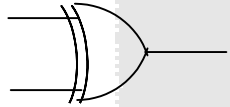


A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Exclusive-OR (XOR) Bca-Aug99
or $A'B + AB'$

A

B

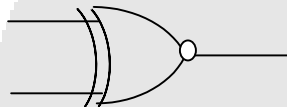


$x = A \oplus B$
or
 $x = A'B + AB'$

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR

$X = A \oplus B$ or $A'B + AB'$



A	B	X
0	0	1
0	1	0
1	0	0
1	1	1

Boolean Algebra

BcaAug99 (Bit-may01)(Bit-Mar02)

Either can represent a Boolean function:

- Truth tables
- Logic diagrams
- Algebraic expression

- Boolean algebra is an algebra that deals with binary variables and logic operations.
- Variables are designated by letters such as A , B , x , and y .
- A Boolean function can be expressed algebraically with binary variables, the logic operation symbols, parentheses, and equal sign.
- The result of a Boolean function is either 0 or 1.
- Consider the following Boolean function:

$$F = xy + z'$$

- The function F is equal to 1 if either both x and y are 1 or z' is 1; F is equal to 0 otherwise.
- Notice that saying $z' = 1$ is equivalent to saying $z = 0$ since z' is the **complement** of z .

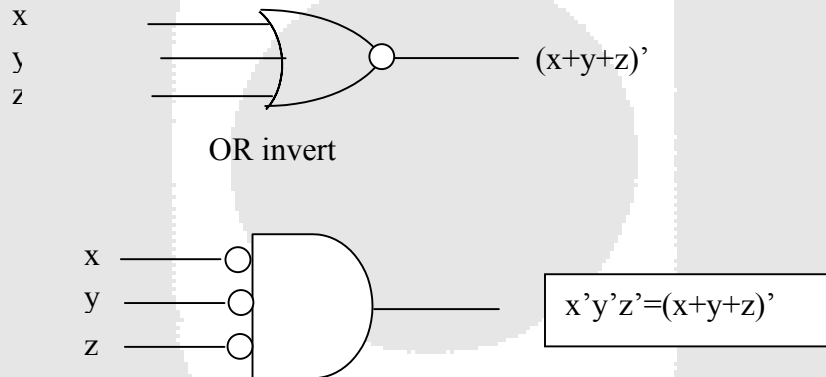
Basic Identities of Boolean algebra

- | | |
|-----------------|-----------------|
| (1) $x + 0 = x$ | (2) $x * 0 = 0$ |
| (3) $x + 1 = 1$ | (4) $x * 1 = x$ |

- | | |
|----------------------------------|--------------------------------|
| (5) $x + x = x$ | (6) $x * x = x$ |
| (7) $x + x' = 1$ | (8) $x * x' = 0$ |
| (9) $x + y = y + x$ | (10) $xy = yx$ |
| (11) $x + (y + z) = (x + y) + z$ | (12) $x(yz) = (xy)z$ |
| (13) $x(y + z) = xy + xz$ | (14) $x + yz = (x + y)(x + z)$ |
| (15) $(x + y)' = x'y'$ | (16) $(xy)' = x' + y'$ |
| (17) $(x')' = x$ | |

De-Morgan's Theorem

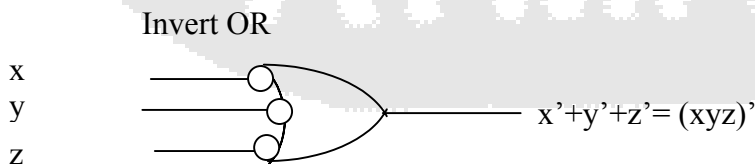
- This theorem is very important in dealing with NOR and NAND gates. It states that a NOR gate that performs the $(x+y)'$ function is equivalent to the function $x'y'$. Similarly a NAND function can be expressed by either $(xy)'$ or $(x'+y')$. For this reason the NOR and NAND gates have two distinct graphic symbols.



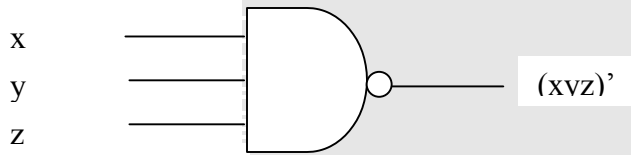
invert AND

- The invert AND symbol for the NOR gate follows from the De-Morgan's theorem and from the convention that small circles denote complementation.

Similarly the NAND gates have two distinct symbols as shown below.



AND-invert



Complement of a function

- The complement of a function F when expressed in a truth table is obtained by interchanging 1's and 0's in the values of F in the truth table.
- When the function is expressed in algebraic form the complement of the function can be derived by means of De-Morgan's Theorem.
- The general form of DeMorgan's theorem can be expressed as follows:
 $(x_1+x_2+x_3+\dots+x_n)' = x_1'x_2'x_3'\dots x_n'$
 $(x_1x_2x_3\dots x_n)' = x_1'+x_2'+x_3'+\dots+x_n'$
- By changing all OR operation to AND operation and all AND operations to OR operations and then complementing each individual letter variable we can derive a simple procedure for obtaining the complement of an algebraic expression.

Eg.

$$F = AB+C'D'+B'D$$

$$F'=(A'+B')(C+D)(B+D')$$

- The complement expression is obtained by interchanging AND and OR operations and complementing each individual.

Map Simplification

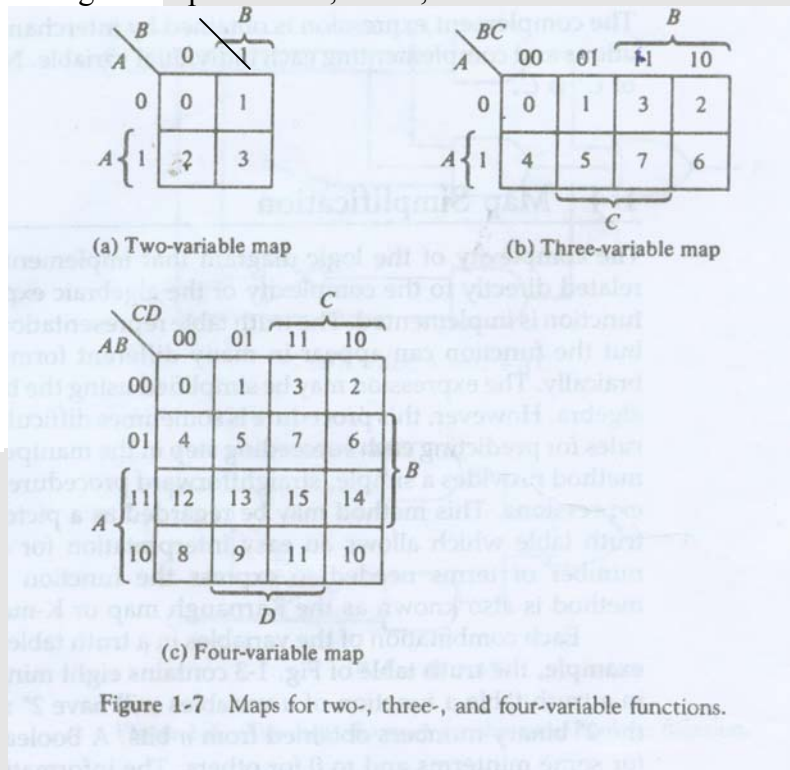
- In addition to using Boolean algebra to simplify a Boolean function, a technique called map simplification can also be utilized.
- The map method is known as the Karnaugh map or K-map.
- Each combination of the variables in a truth table is called a minterm.
- There are 2^n minterms for a function of n variables.
- A fourth representation of a Boolean function can be given as the sum of the functions minterms. (Sum-of-Products)

Examples

$$F(x,y,z)=\Sigma(1,4,5,6,7)$$

Variable Maps

The following are maps for two-, three-, and four-variable function:



- The variable names are listed across both the sides of the diagonal line into the corner of the map.
- The 0's and the 1's marked along each row and each column designate the value of the variables.
- Each variable under the brackets contain half of the squares in the map where that variable appears unprimed.
- The minterm represent by a square is determined from the binary assignment of the variable along the left top edges in the map.
- Here the min term 5 the three variable maps are 101 of the second column. This minterm represents a value for the binary variables A, B and C with A and C being unprimed and B being primed.

Sum-of-Products Simplification

- A Boolean function represented by a truth table is plotted into the map by inserting 1's into those squares where the function is 1.
- Boolean functions can then be simplified by identifying adjacent squares in the Karnaugh map that contain a 1.
- A square is considered adjacent to another square if it is next to, above, or below it. In addition, squares at the extreme ends of the same horizontal

row are also considered adjacent. The same applies to the top and bottom squares of a column.

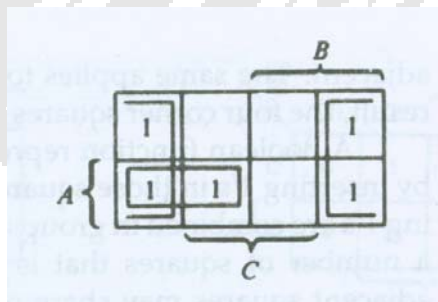
- The objective to identify adjacent squares containing 1's and group them together.
- Groups must contain a number of squares that is an integral power of 2.
- Groups of combined adjacent squares may share one or more squares with one or more groups.
- Each group of squares represents an algebraic term, and the **OR** of those terms gives the simplified algebraic expression for the function.
- To find the most simplified algebraic expression, the goal of map simplification is to identify the least number of groups with the largest number of members.
- We will simplify the Boolean function.
- $F(A,B,C) = \Sigma(3,4,6,7)$
- The three variable maps for this function is shown in the above figure.
- There are four squares marked with 1's, one for each minterm that produces 1 for the function. These squares belong to minterm 3,4,6,7 and are recognized from the figure b.
- Two adjacent squares are combined in the third column. This column belongs to both B and C produces the term BC.
- The remaining two squares with 1's in the two corner of the second row are adjacent and belong to row columns of C', so they produce the term AC'.

The simplified expression for the function is the or of the two terms:

- The second example simplifies the following Boolean function:
 $F(A,B,C) = \Sigma(0,2,4,5,6)$
 $F = BC + AC'$
 - The five minterms are marked with 1's in the corresponding squares of the three variable maps.
- The four squares in the first and the fourth columns are adjacent and represent the term C'.
- The remaining square marked with a 1 belongs to minterm 5 and can be combined with the square of minterm 4 to produce the term AB'. The simplified function is

Fig 1.8 Map for $F(A,B,C) = \Sigma(3,4,6,7)$

$$F = C' + AB'$$

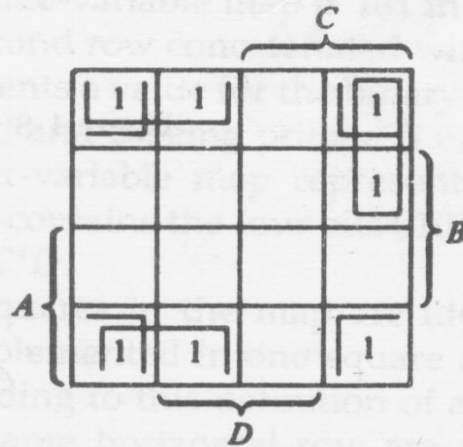


The third variable map.

example needs a four-

$$F(A,B,C,D)=\Sigma(0,1,2,6,8,9,10)$$

Figure 1-10 Map for $F(A, B, C, D) = \Sigma(0,1,2,6,8,9,10)$.



- The area in the map covered by this four variable consists of the squares marked with 1's in fig 1.10. The function contains 1's in the four corners that when taken as groups give the term $B'D'$. This is possible because these four squares are adjacent when the map is considered with the top and bottom or left and right edges touching.
- The two 1's on the bottom row are combined with the two 1's on the left of the bottom row to give the term $B'C'$.
- The remaining 1 in the square of minterm 6 is combined with the minterm 2 to give the term $A'CD'$. The simplified function is

Product-of-Sums Simplification

$$F = B'D' + B'C' + A'CD'$$

- Another method for simplifying Boolean expressions can be to represent the function as a product of sums.
- This approach is similar to the Sum-of-Products simplification, but identifying adjacent squares containing 0's instead of 1's forms the groups of adjacent squares.
- Then, instead of representing the function as a sum of products, the function is represented as a product of sums.

Examples

$$F(A,B,C,D) = \Sigma(0,1,2,5,8,9,10)$$

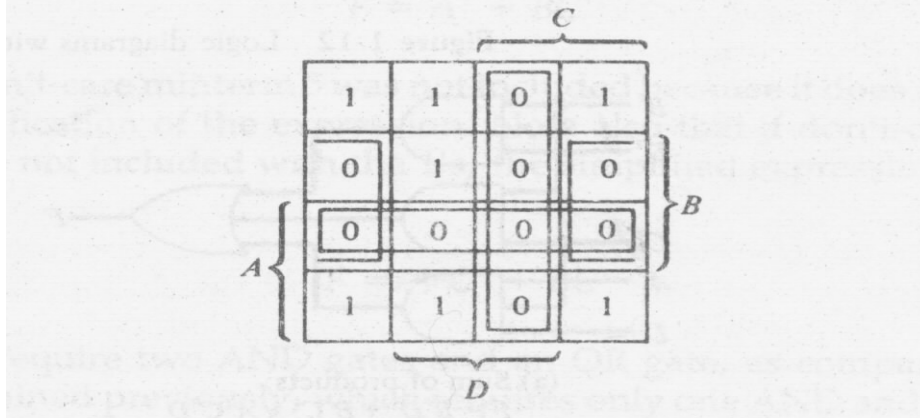
- The 1's marked in the map of fig 1-11 represents the minterms that produces a 1 for the function,
- The squares marked with 0's represent the minterm not included in F and therefore denote the complement of F.
- Combining the squares with 1's gives the simplified function in sum-of-products form:

$$F = B'D + B'C' + A'C'D$$

- If the squares marked with 0's are combined as shown in the diagram, we obtain the simplified complement function:

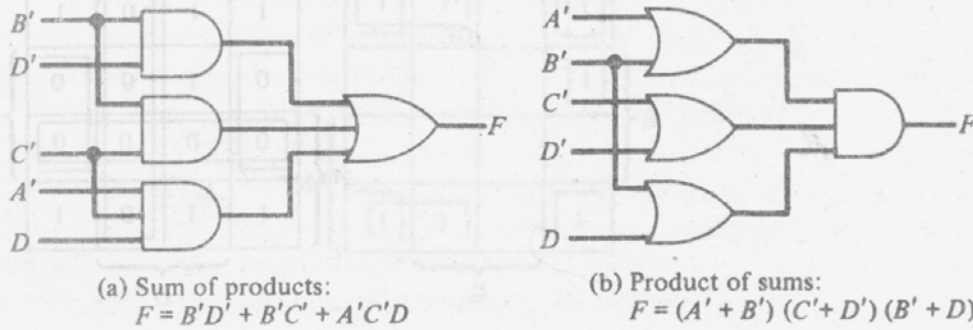
$$F' = (A' + B')(C'D')(B' + D)$$

Figure 1-11 Map for $F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$.



The logic diagram of the two simplified expression are shown in fig 1.12

Figure 1-12 Logic diagrams with AND and OR gates.



- The sum of product expression is implemented in fig 1.12(a) with a group

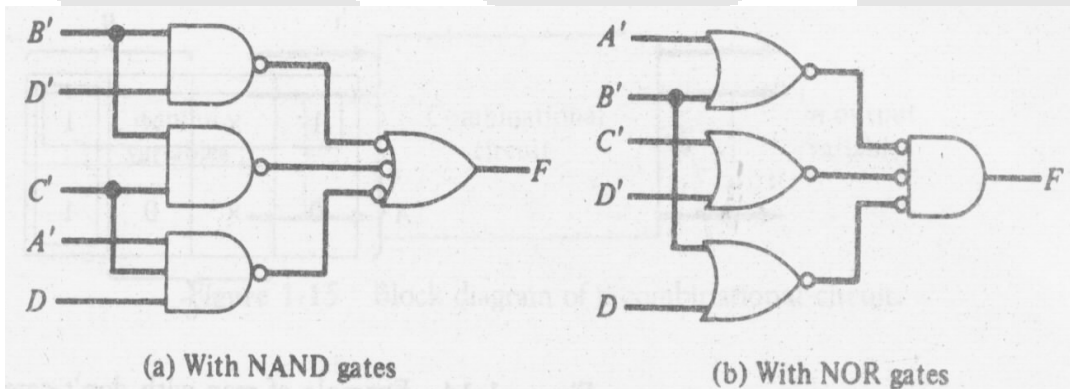


Figure 1-13 Logic diagrams with NAND or NOR gates.

- of AND gates, one for each AND term.
- The output of the AND gates are connected to the inputs of a single Or gate. The same function is implemented in fig1.12b in product of sums

forms with a group of OR gates, one for each OR term, the outputs of the OR gates are connected to the inputs of a single And gate.

- In each case it is assumed that the input variable are directly available in their complement, so inverter are not included.

Don't Care Conditions

- On occasion, it doesn't matter whether a function produces a 0 or 1 for a given minterm.
- When this condition occurs, an X is used in the map to represent the *don't care* condition.
- Then, when performing map simplification, a square containing an X can be used in both the Sum-of-Products approach and the Product-of-Sums approach.
- When choosing adjacent squares for the function in the map, the x 's may be assumed to be either 0 or 1, whichever gives the simplest expression/
- In addition an x need not to be used at all if it does not contribute to the simplification of the function.
- In each case the choice depends only on the simplification that can be achieved. As example consider the following Boolean function together with the don't care minterms:

$$F(A,B,C) = \Sigma(0,2,6)$$

$$d(A,B,C) = \Sigma(1,3,5)$$

- The minterm listed with F produce a 1 for the function. The don't care minterms listed with d may produce either a 0 or 1 for the function. The remaining minterms 4,7 produce a 0 for the function.
- The map is shown fig 1.14. The minterms of F are marked with 1's those of d are marked with x 's and the remaining squares are marked with 0's.
- The 1's and x 's are combined in any convenient manner so as to enclose the maximum number of adjacent squares.
- It is not necessary to include the don't care minterms 1 and 3 with the 1's in the first row we obtain the term, BC' . The simplified expression is

Combinational Circuits

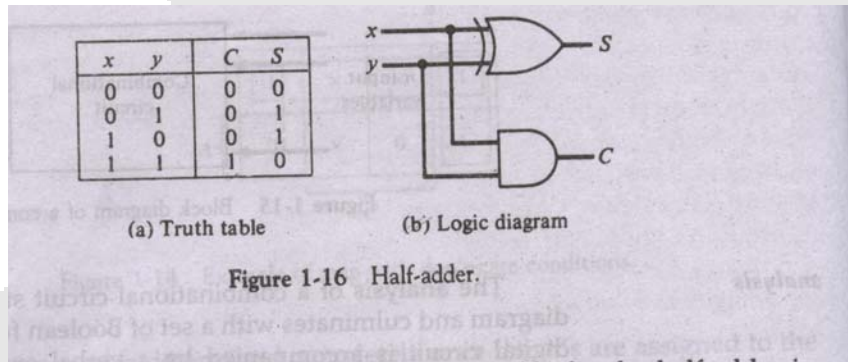
Bca-Aug99(Bit-may01)

$$F = A' + BC$$

- A combinational circuit is a connected arrangement of logic gates with a set of inputs and outputs.
- At any given time, the binary values of the outputs are a function of the binary values of the inputs.
- The design of a combinational circuit starts from a verbal outline of the problem and ends in a logic circuit diagram. The procedure involves the following steps:
- The problem is stated.
- The input and output variables are assigned letter symbols.
- The truth table that defines the relationship between inputs and outputs is derived.
- The simplified Boolean functions for each output are obtained.
- The logic diagram is drawn.

Half-Adder

- The most basic digital arithmetic circuit.
- Performs the addition of two binary digits.
- The input variables of a half-adder are called the *augends* and the *addend*.
- The output variables of a half-adder are called the *sum* and the *carry*.



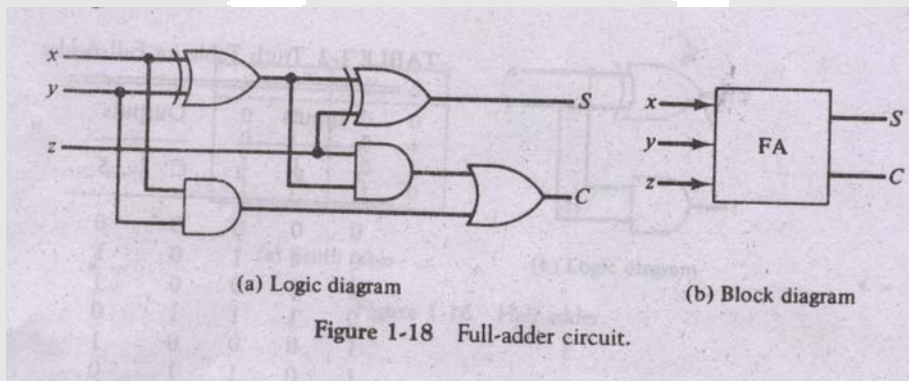
$$S = x'y + xy' = x \oplus y$$

$$C = xy$$

Full-Adder

(Bit-Mar02)

- A full-adder performs the addition of three binary digits.



Two half-adders can be combined to form a full-adder..

ATMIYA

TABLE 1-2 Truth Table for Full-Adder

Inputs			Outputs	
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

ation of the variables (see the discussion at the

- Although a full adder has three inputs, it still only has two outputs since the largest number is $1+1+1 = 3$, and 3 can be represented by two bits.
- The full adder circuit contains two half adders and an OR gate.

Flip Flops Bca-Aug99

- A Flip-flop is a binary cell capable of storing one bit of information.
- It has two outputs, one for the normal value and one for the complement value of the bit stored in it.
- Flip-flops are storage elements utilized in synchronous sequential circuits.
- Synchronous sequential circuits employ signals that effect storage elements only at discrete instances of time.
- A timing device called a clock pulse generator that produces a periodic train of clock pulses achieves synchronization.
- Values maintained in the storage elements can only change when the clock pulses.
- Hence, a flip-flop maintains a binary state until directed by a clock pulse to switch states.
- The difference in the types of flip flops is in the number of inputs and the manner in which the inputs affect the binary state.
- Flip-flops can be described by a *characteristic table* which permutates all possible inputs (just like a truth table).
- The characteristic table of a flip-flop describes all possible outputs (called the *next state*) at time $Q(t+1)$ over all possible inputs and the *present state* at time $Q(t)$.
- The most common types of flip flops are:
 - SR Flip-Flop
 - D Flip-Flop
 - JK Flip-Flop
 - T Flip-Flop

SR Flip-Flop Bca-Aug99

- Inputs:
- S (for set)
- R (for reset)
- C (for clock)

- Outputs:
- Q
- Q'

Characteristic Table			
S	R	Q(t+1)	Description
0	0	Q(t)	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	?	Indeterminate

- The operation of the SR flip-flop is as follow.
- If there is no signal at the clock input C, the output of the circuit cannot change irrespective of the values at inputs S and R.
- Only when the clock signals changes from 0 to 1 can the output be affected according to the values in inputs S and R
- If S =1 and R = 0 when C changes when C changes from 0 to 1 output Q is set to 1. If S = 0 and R =1 when C changes from 0 to 1.
- If both S and R are 0 during the clock transition, output does not change.
- When both S and R are equal to 1, the output is unpredictable and may go to either 0 or 1, depending on internal timing that occur within the circuit

D Flip-Flop

Inputs:

D (for data)

C (for clock)

Outputs:

Q

Q'

Characteristic Table		
D	Q(t+1)	Description
0	0	Clear to 0
1	1	Set to 1

JK Flip-Flop Bca-Aug99 (*Bit-may01*) (*Bit-Mar02*)

Inputs:

J

K

C (for clock)

Outputs: Q Q'

Characteristic Table			
J	K	Q(t+1)	Description
0	0	Q(t)	No change
0	1	0	Clear to 0
1	0	1	Set to 1
1	1	Q'(t)	Complement

T Flip-Flop

Inputs:

T (for toggle)

C (for clock)

Outputs:

Q

Q'

Characteristic Table		
T	Q(t+1)	Description
0	Q(t)	No change
1	Q'(t)	Complement

- Most flip-flops are edge-triggered flip-flops, which means that the transition occurs at a specific level of the clock pulse.
- A positive-edge transition occurs on the rising edge of the clock signal.
- A negative-edge transition occurs on the falling edge of the clock signal.
- Another type of flip-flop is called a master-slave flip-flop that is basically two flip-flops in series.

- Flip-flops can also include special input terminals for setting or clearing the flip-flop *asynchronously*. These inputs are usually called *preset* and *clear* and are useful for initialing the flip-flops before *clocked* operations are initiated.

Flip-Flop Excitation Tables

- During the design of sequential circuits, the required transition from present state to next state is known. Transition.
- What the designer needs to know is what input conditions must exist to implement the required.
- This requires the use of flip-flop excitation tables.

Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Q(t)	Q(t+1)	D
0	0	0
0	1	1
1	0	0
1	1	1

Q(t)	Q(t+1)	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

Q(t)	Q(t+1)	T
0	0	0
0	1	1
1	0	1
1	1	0

Sequential Circuits

Bca-Aug99

- When a circuit contains just gates, it is called a *combinational* circuit.
- However, if a circuit uses both gates and flip-flops, it is called a *sequential* circuit.
- Hence, a sequential circuit is an interconnection of flip-flops and gates.
- If we think of a sequential circuit as some *black box* that, when provided with some *external* input, produces some *external* output, a typical sequential circuit would function as follows:
- The *external* inputs constitute *some* of the inputs to the combinational circuit.
- The *internal* outputs of the combinational circuit are the *internal* inputs to the flip-flops.
- The *internal* outputs of the flip-flops constitute the remaining inputs to the combinational circuit.
- The *external* outputs are some combination of the outputs from the combinational circuit and flip-flops.
- The behavior of a sequential circuit is determined from the inputs, the outputs, and the state of the flip-flops.
- Both the outputs and the next state are determined by the inputs and the present state. A state table can picture this relationship.
- A state diagram can represent the information in a state table graphically, where states are represented by circles (vertices) and transitions on specific input is represented by the *labels* on the directed lines (edges) connecting the circles.

Design Procedure

1. Formulate behavior of circuit using a state diagram.
2. Determine # of flip-flops needed (equal to # bits in circles).
3. Determine # inputs (specified on edges of diagram).
4. Create state table, assigning letters to flip-flops, input, and output variables.*
5. For each row, list the next state as specified by the state diagram.
6. Select flip-flop type to be used in circuit.
7. Extend state table into an excitation table by including columns for each input of each flip-flop.
8. Using excitation table and present state-to-next state transitions, formulate input conditions for flip-flops.
9. Construct truth table for combinational circuit using present-state and input columns of excitation table (for inputs) and flip-flop inputs (for outputs).
10. Use map simplification of truth table to obtain flip-flop input equations.**
11. Determine *external* outputs of sequential circuit (flip-flop outputs and potentially combinational circuit outputs).
12. Draw logic diagram as follows:

13. Draw flip-flops and label all their inputs and outputs.
14. Draw combinational circuit from the Boolean expressions given by the flip-flop input equations.
15. Connect outputs of flip-flops to inputs in the combinational circuit.
16. Connect outputs of combinational circuit to flip-flop inputs.

* For m flip-flops and n inputs, the state table will consist of m columns for the present state, n columns for the inputs, and m columns for the next state. The number of rows in the table will be up to 2^{m+n} , one row for each binary combination of present state and inputs.

** Each flip-flop input equation specifies a logic diagram whose output must be connected to one of the flip-flop inputs.



ATMIYA

Integrated Circuits

This chapter explains in detail the logical operation of the most common standard digital components. It includes decoders, multiplexer's registers, counters and memories. These digital components are used as building blocks for the design of larger units in the chapter that follow.

Integrated Circuits

- Digital circuits are constructed with integrated circuits.
- An integrated circuit (IC) is a small silicon semiconductor crystal, called a *chip*, containing the electronic components for the digital gates.
- Gates are interconnected inside the chip to form the required circuit.
- The chip is mounted in a ceramic or plastic container, and thin gold wires to external pins to form the integrated circuit weld connections.
- The number of pins may range from 14 in a small IC package to thousands in a large IC package.
- As technology improved, the number of gates that could be put on a single chip increased considerably.
- Chips are classified in the following categories primary based on the number of gates on the chip:
 - Small-scale integration (SSI)
 - Number of gates usually less than 10
 - Inputs and outputs typically connected directly to pins.
 - Medium-scale integration (MSI)
 - Number of gates: between 10 and 200
 - Typically perform specific elementary digital functions.
 - Examples: decoders, adders, registers
 - Large-scale integration (LSI)
 - Number of gates: between 200 and a few thousand
 - More complex digital systems.
 - Examples: processors, memory chips, programmable modules
 - Very-large-scale integration (VLSI)
 - Number of gates: thousands
 - Even more complex digital systems.
 - Examples: large memory arrays, complex microcomputer chips
- Digital integrated circuit are classified not only by their logic operation but also by the specific circuit technology (*digital logic family*) to which they belong.
- The most popular digital logic families are:
 - TTL - Transistor-transistor logic
 - Widespread logic family

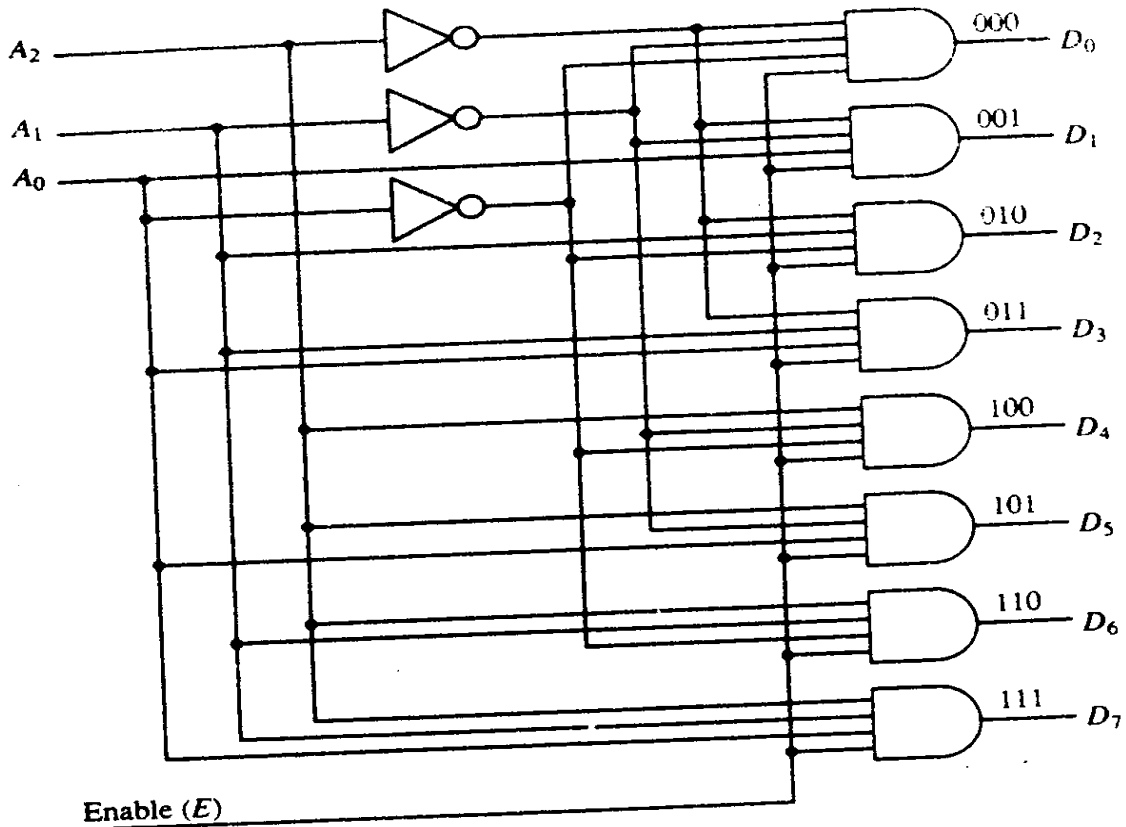
- Uses transistors to implement gates
- ECL - Emitter-coupled logic
 - Used in high-speed systems
 - Uses special nonsaturated transistors to achieve super fast speed
- MOS - Metal-oxide semiconductor
 - Used in circuits that have high component density
 - uses unipolar transistors
- CMOS - Complementary metal-oxide semiconductor
 - also used in circuits that have high component density
 - also uses unipolar transistors but connects them in a complementary fashion
 - more economical than MOS because of low power consumption

Decoders

Bca-Aug99(Bit-may01) (Bit-Mar02)

- A binary code of n bits is capable of representing up to 2^n distinct elements in coded information.
- A *decoder* is a combinational circuit that converts binary information from the n coded inputs to a maximum of 2^n unique outputs.

ATMIYA



3-8 line Decoder

- If the n -bit coded information has unused bit combinations, the decoder may have less than 2^n outputs.
- An n -to- m -line (or $n \times m$) decoder, where $m \leq 2^n$, has up to m output variables and for n input variables.
- Decoders typically include one or more enable inputs to control (turn on/off) the operation of a circuit.

Truth Table for 3-to-8-Line Decoder

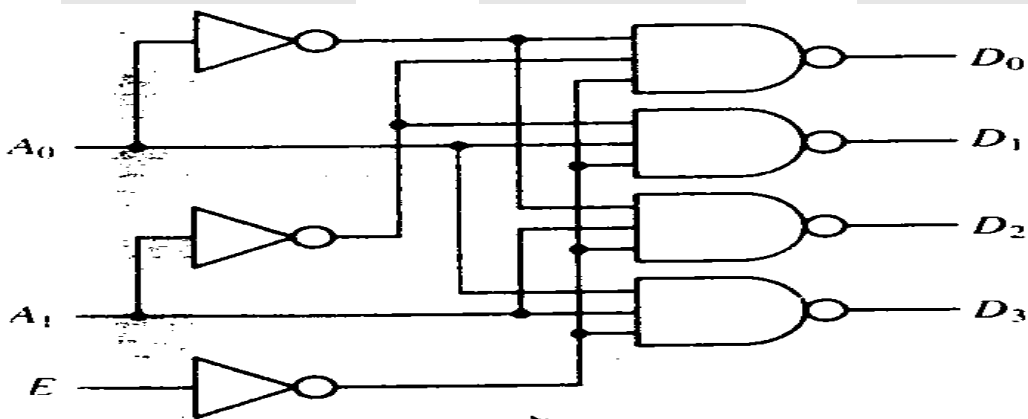
Enable	Inputs			Outputs							
	A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	1	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

Enable	Inputs			Outputs								
	E	A ₂	A ₁	A ₀	D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀
0	X	X	X	X	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0	0
1	1	0	1	1	0	0	1	0	0	0	0	0
1	1	1	0	0	0	1	0	0	0	0	0	0
1	1	1	1	1	1	0	0	0	0	0	0	0

The output of the 3-to-8-line decoder can be expressed by the following binary functions:

$$\begin{aligned}
 D_0 &= EA_2'A_1'A_0' \\
 D_1 &= EA_2'A_1'A_0 \\
 D_2 &= EA_2'A_1A_0' \\
 D_3 &= EA_2'A_1A_0 \\
 D_4 &= EA_2A_1'A_0' \\
 D_5 &= EA_2A_1'A_0 \\
 D_6 &= EA_2A_1A_0' \\
 D_7 &= EA_2A_1A_0
 \end{aligned}$$

- A decoder can also be constructed using either AND gates or NAND gates.
- Fig 2.2 2 to 4 line decoder with NAND gates

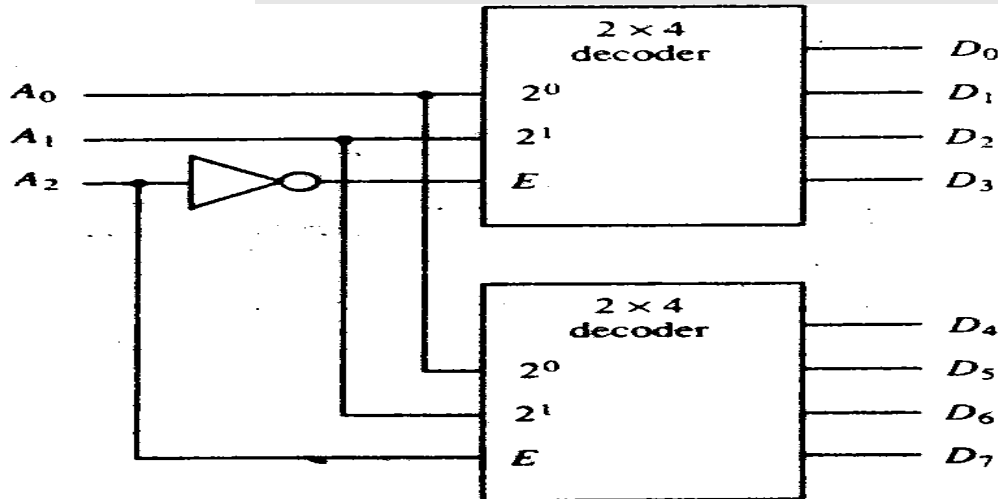


E	A ₁	A ₀	D ₀	D ₁	D ₂	D ₃
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0
1	X	X	1	1	1	1

b) Truth Table

- A technique called **decoder expansion** can be utilized to construct larger decoders out of smaller ones. For example, two 2-to-4-line decoders can be combined to construct a 3-to-8-line decoder.

Fig 2.3 3-8-line decoder constructed with two 2x4 decoders



- The above given Figure 2-3 shows how the decoders with enable inputs can be connected to form a larger decoder.
- As you can see that there are two 2-to-4-line decoders are combined to achieve a 3-to-8-line decoder.
- The two least significant bits of the input are connected to both decoders.
- The most significant bit is connected to the enable input of one decoder and through an inverter to the enable input of the other decoder.
- It is assumed that each decoder is enabled when its E input is equal to 1.
- When E is equal to 0, the decoder is disabled and all its outputs are in the 0 level. When $A_2 = 0$, the upper decoder is enabled and the lower is disabled.
- The lower decoder outputs become inactive with all outputs at 0. The outputs of the upper decoder generate outputs D_0 through D_3 , depending on the values of A_1 and A_0 (while $A_2 = 0$).
- When $A_2 = 1$, the lower decoder is enabled and the upper is disabled. The lower decoder output generates the binary equivalent D_4 through D_7 since these binary numbers have a 1 in the A_2 position.

Encoders

- A *encoder* is a combinational circuit that performs the inverse operation of a decoder.
- The output lines generate the binary code corresponding to the input values.

- An encoder has 2^n (or less) unique inputs and n outputs.

Truth Table for 8-to-3-Line Encoder

Inputs								Outputs		
D ₇	D ₆	D ₅	D ₄	D ₃	D ₂	D ₁	D ₀	A ₂	A ₁	A ₀
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	0
0	0	1	0	0	0	0	0	1	0	1
0	1	0	0	0	0	0	0	1	1	0
1	0	0	0	0	0	0	0	1	1	1

- The output of the 8-to-3-line encoder can be expressed by the following binary functions:

$$A_0 = D_1 + D_3 + D_5 + D_7$$

$$A_1 = D_2 + D_3 + D_6 + D_7$$

$$A_2 = D_4 + D_5 + D_6 + D_7$$

Multiplexers Bca-Aug99

- A *multiplexer* is a combinational circuit that receives input from one of 2^n input data lines and directs it to a **single** output line.
- The selection of a particular input data line for a particular output is determined by a set of selection inputs.
- A 2^n -to-1 multiplexer has 2^n input data lines and n input selection lines to determine which input line to select the data from to be placed on the single output line.
- A *function table* can be used to describe the functionality of a multiplexer.

Function Table for 4-to-1-Line Multiplexer

Select		Output
S ₁	S ₀	Y
0	0	I ₀
0	1	I ₁
1	0	I ₂
1	1	I ₃

- Like encoders, Multiplexers can have an enable input to control the operation of the unit.

- Additionally, Multiplexers can be combined together to perform more complex tasks.

Registers

- A *register* is a group of flip-flops capable of storing one bit of information.
- An n -bit register has a group of n flip-flops and is capable of storing any binary information of n bits.
- In addition to flip-flops, registers can have combinational gates that perform certain data-processing tasks. The gates control how and when new information is transferred into the registers.
- The transfer of new information into a register is referred to as a *register load*. If the loading occurs simultaneously at a common clock pulse transition, we say that the load is done in *parallel*.
- The *load input* in a register determines the action to be taken with each clock pulse.
- When the load input is 1, the data from the input lines is transferred into the register's flip-flops. When the load input is 0, the data inputs are *inhibited* and the flip-flop maintains its present state.

Shift Registers

- A *register* capable of shifting its binary information in one or both directions is called a *shift register*.
- Shift registers are constructed by connecting flip-flops in *cascade*, where the output of one flip-flop is connected to the input of the next flip-flop.
- All flip-flops receive common clock pulses that initiate the shift from one stage to the next.
- A *serial input* shift register has a single external input (called the serial input) entering an outermost flip-flop. Each remaining flip-flop uses the output of the previous flip-flop as its input, with the last flip-flop producing the external output (called the serial output).
- A register capable of shifting in one direction is called a *unidirectional* shift register.
- A register that can shift in both directions is called a *bi-directional* shift register.
- The most general shift register has the following capabilities:
 - An input for clock pulses to synchronize all operations.
 - A shift-right operation and a serial input line associated with the shift-right.
 - A shift-left operation and a serial input line associated with the shift-left.
 - A parallel load operation and n input lines associated with the parallel transfer.
 - N parallel output lines.
 - A control state that leaves the information in the register unchanged even though clock pulses are applied continuously.
 - A mode control to determine which type of register operation to perform.

Function Table for Shift Register Mode Control

Mode Control		Register operation
S ₁	S ₀	
0	0	No Change
0	1	Shift right (down)
1	0	Shift left (up)
1	1	Parallel load

Binary Counters (Bit-Mar02)

- A *register* that goes through a predetermined sequence of states upon the application of input pulses is called a *counter*.
- The input pulses may be clock pulses or may originate from an external source. They may occur at uniform times or at random.
- A counter that follows a binary sequence is called a *binary* counter. An *n*-bit binary counter is a register of *n* flip-flops along with a combinational circuit that continually produces a binary count of *n* bits having a value from 0 to $2^n - 1$.
- The most general binary counter register has the following capabilities:
- An input for clock pulses to synchronize all operations.
- An increment operation that signals the register to increment its value by 1.
- A parallel clear operation that sets all the flip-flop values to 0.
- A parallel load operation that sets all the flip-flop values according to the values of *n* input lines associated with the parallel load.
- *n* parallel output lines.
- A control state that leaves the information in the register unchanged even though clock pulses are applied continuously.

Function Table for Binary Counter Register

Clear	Load	Increment	Operation
0	0	0	No Change
0	0	1	Increment count by 1
0	1	X	Load inputs I ₀ -I _{n-1}
1	X	X	Clear outputs to 0

Memory Unit

- A *memory unit* is a collection of storage cells together with associated circuits needed to transfer information in and out of storage.
- Memory stores binary information in groups of bits called *words*.

- A memory word can be used to represent any type of binary-coded information.
- A group of eight bits is called a *byte*. Most systems use memory words that are a multiple of eight.
- Thus, a 16-bit word contains two bytes. A 32-bit word contains four bytes, and so on.
- The number of words it contains and the number of bits in each word define the internal structure of a memory unit.
- Special input lines called *address lines* select one particular word.
- Each word in memory is assigned a unique *address* ranging from 0 to 2^{k-1} , where k is the number of address lines.
- Applying the k -bit binary address to the address lines does the selection of a specific word in memory.
- A decoder inside the memory accepts the address and opens the paths needed to select the bits of the specified word.
- Computer memories range from 1024 words, requiring an address of 10 bits, to 2^{32} words, requiring 32 address bits.
- It is customary to refer to the number of words in a memory with one of the letters K (kilo), M (mega), G (giga), where K is equal to 2^{10} , M is equal to 2^{20} , G is equal to 2^{30} .
- Thus, $64K = 2^{16}$, $2M = 2^{21}$, and $4G = 2^{32}$.
- Two major types of memory are used in computer systems:

Random-Access Memory (RAM)

- The concept of *random-access* comes from the fact that the process of locating a word in memory is the same and requires an equal amount of time regardless of its physical location.
- Data transfers occur through the use of data input and output lines, address selection lines, and control lines.
- The two operations that a random-access memory can perform are the *read* and *write*.
- To perform the read operation, the address of the desired word in memory is placed on the address selection lines and the *read* control line is activated. The memory unit then places the desired data onto the data output lines.
- To perform the write operation, the address of the desired word to be placed into memory is placed on the address selection lines, data to be written is placed on the data input lines, and the *write* control line is activated.

Read-Only Memory (ROM)

Bca-Aug99

- The *read-only* memory can only perform the read operation.
 - Data placed in ROM must be done so during the hardware production of the unit.
 - The read operation on a ROM is identical to that of a RAM with the exception being that there is no need for a read control line.
- “Yogidham” Kalavad Road, Rajkot. Ph:- 572365 576681

Atmiya Infotech

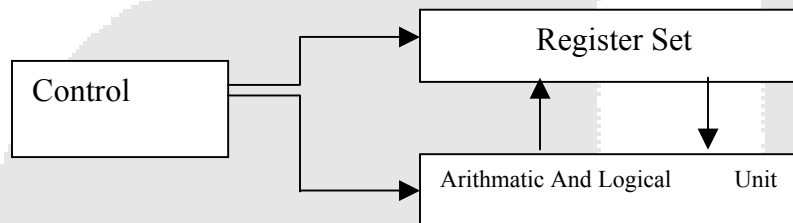
- One special type of ROM, called a *programmable read-only memory* (PROM) allows the purchaser of the memory to write to the ROM **one time** only.
- Another type of ROM, called an *erasable PROM* (EPROM) provides the purchaser of the system the capability to modify the data in a ROM. However, this must be done physically by placing the memory unit under ultraviolet light for a specified amount of time.



Central Processing Unit

The CPU

- The CPU is the focal point of the computer, guiding all actions that take place in the system
- Contains an Arithmetic and Logic Unit (or ALU) to perform all addition, subtraction, logic operations, etc.



- Also contains several *registers*, these function like very small very fast pieces of memory, used to store the data the ALU is using right now
 - Contains all the control logic to coordinate actions between all elements of the CPU as well as all the other computer hardware components
 - The CPU is generally contained entirely on one integrated circuit (IC) or *chip*
- ### General Register Organization
- The set of registers in a computer are connected to the ALU using busses and Multiplexers.
 - A 14-bit control word specifies two *source* registers (SELA & SELB), a *destination* register (SELD), and an operation (OPR).

ATMIYA

- The registers can be specified using three bits each as follows:

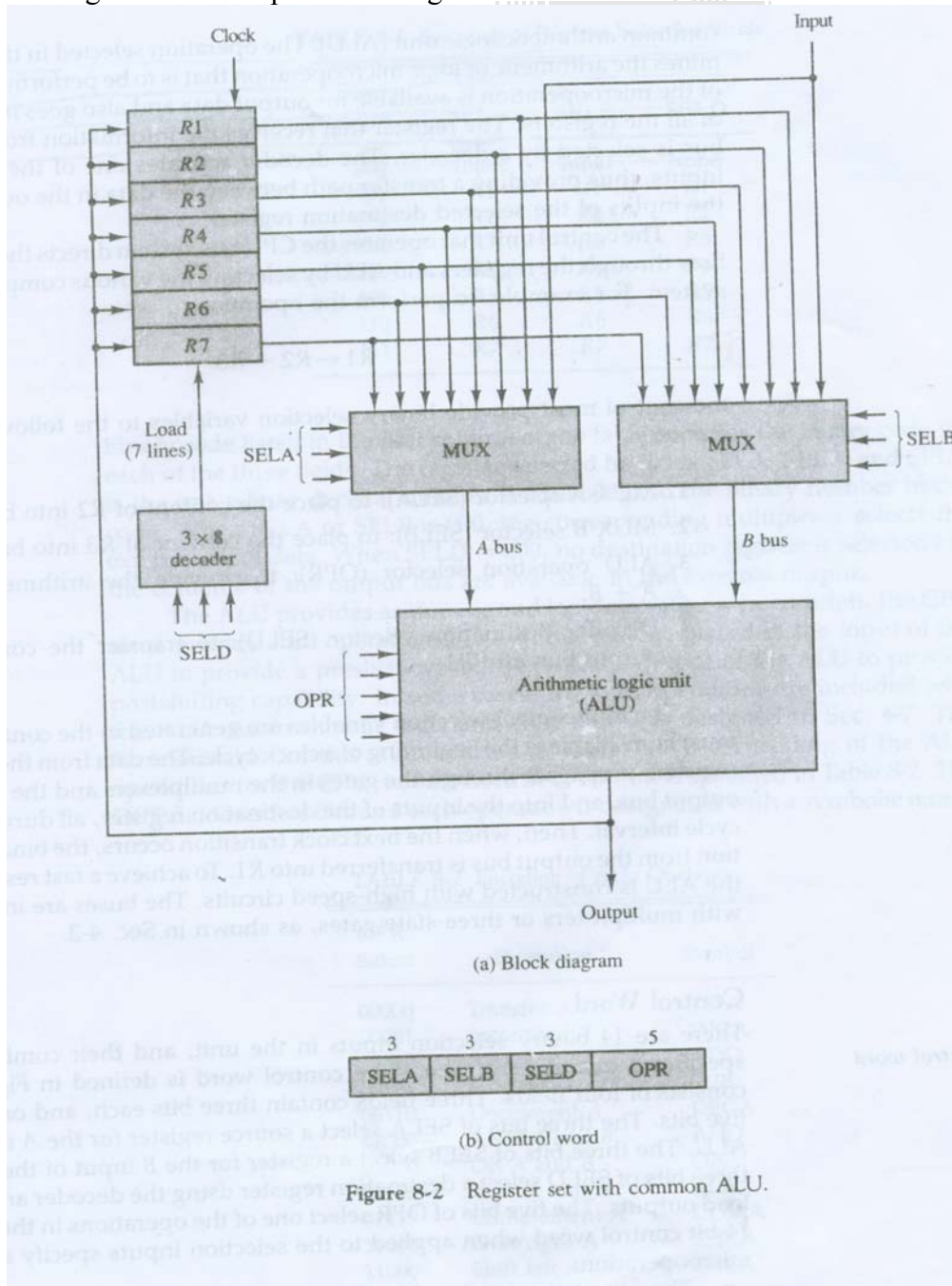


Figure 8-2 Register set with common ALU.

Binary Code	SELA	SELB	SELD
000	Input	Input	None
001	R1	R1	R1
010	R2	R2	R2
011	R3	R3	R3
100	R4	R4	R4
101	R5	R5	R5

110	R6	R6	R6
111	R7	R7	R7

The remaining four bits of the control word can be used to specify the following ALU operations:

OPR Select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	Add A + B	ADD
00101	Subtract A - B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	Shift right A	SHRA
11000	Shift left A	SHLA

Examples of Micro operations Bca-Aug99(Bit-may01)

- A control word of 14 bits is needed to specify a micro operation in the CPU. The control word for a given micro operation can be derived from the selection variables .for eg.the subtract micro operation given by the statement

$$R1 \leftarrow R2 - R3$$

specifies R2 for the *A* input of the ALU, R3 for the *B* input of the ALLJ, R1 / the destination register, and an ALU operation to subtract *A - B*. 3
- Thus t control word-is specified by-the four fields and the corresponding binary value for each field is obtained from the encoding listed in Tables 8-1 and 8-2.
- The binary control word for the subtract micro operation is 010 Oil 001 00101 ai is obtained as follow;

Field:	SELA	SELE	SELD	OPR
Symbol:	R2	R3	R1	SUB
Control word	010	011	001	0010
- The control word for this micro operation and a few others are listed Table 8-3.. The increment and transfer micro operations do not use the B input of *tt* ALU. For these cases, the B field is marked with a dash. We assign 000 to ar unused field when formulating the binary control word, although any other binary number may be used. To place the content of a register into the output terminals we place the content of the register into the *A*

input of the ALU, b none of the registers are selected to accept the data. The ALU operation TSF places the data from the register, through the ALU, into the output terminal I ~ J • t ~ I I The direct transfer from input to output is accomplished with a control word of all 0's (making the B field 000). A register can be cleared to 0 with an exclusive-OR operation. This is because $x @ x = 0$. It is apparent from these examples that many other

TABLE 8-3 Examples of Micro operations for the CPU

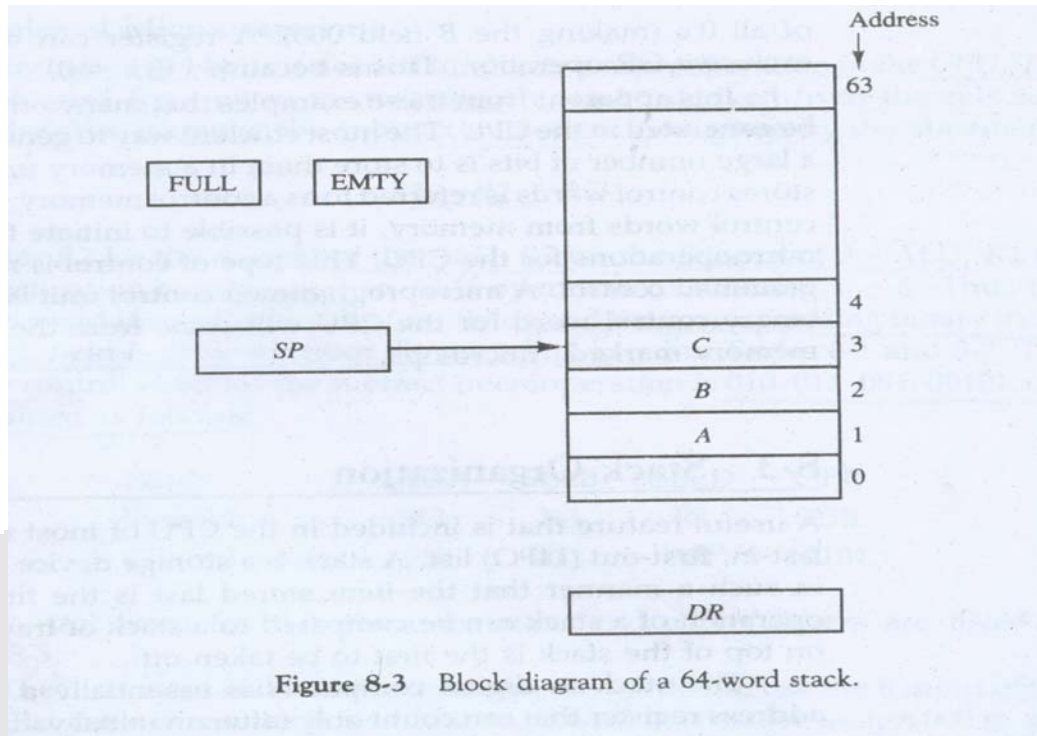
Micro operation	Symbolic Designation				Control Word
	SELA	SELB	SELD	OPR	
R1←R2-R3	R2	R3	R1	SUB	010 011 001 00101
R4←R4∨R5	R4	R5	R4	OR	100 101 100 01010
R6←R6+1	R6	-	R6	INCA	110 000 110 00001
R7←R1	R1	-	R7	TSFA	001 000 111 00000
Output ←R2	R2	-	None	TSFA	010 000 000 00000
Outputs←-Input	Input	-	None	TSFA	000 000 000 00000
R4←shl R4	R4	R5	R4	SHLA	100 000 100 11000
R5← 0	R5	R5	R5	XOR	101 101 101 01100

microoperations can be generated in the CPU. The most efficient way to generate control words with a large number of bits is to store them in a memory unit. A memory unit that stores control words is referred to as a control memory. By reading consecutive control words from memory, it is possible to initiate the desired sequence of micro operations for the CPU. This type of control is referred to as microprogrammed control. A microprogrammed control unit is shown in Fig. 7-8. The binary control word for the CPU will come from the outputs of the control memory marked "micro-ops."

Stack Organization Bca-Aug99 (Bit-may01) (Bit-Mar02)

- A stack is a storage device that stores information in a last-in, first-out (LIFO) fashion.
- A stack has two operations: *push*, which places data onto the stack, and *pop*, which removed data from the stack.
- A computer can have a separate memory reserved just for stack operations. However, most utilize main memory for representing stacks.
- Hence, all assembly programs should allocate memory for a stack.

ATMIYA



- The SP register is initially loaded with the address of the top of the stack.
- In memory, the stack is actually upside-down, so when something is *pushed* onto the stack, the stack pointer is decremented.

$$SP \leftarrow SP - 1$$

$$M[SP] \leftarrow DR$$
- And, when something is *popped* off the stack, the stack pointer is incremented.

$$DR \leftarrow M[SP]$$

$$SP \leftarrow SP + 1$$
- The push and pop instructions can be explicitly executed in a program. However, they are also implicitly executed for such things as procedure calls and interrupts as well.
- Care must be taken when performing stack operations to ensure that an *overflow* or *underflow* of the stack does not occur.

Memory Stack

Figure shows a portion of computer memory partitioned in to three segments.

Program

Data

Stack

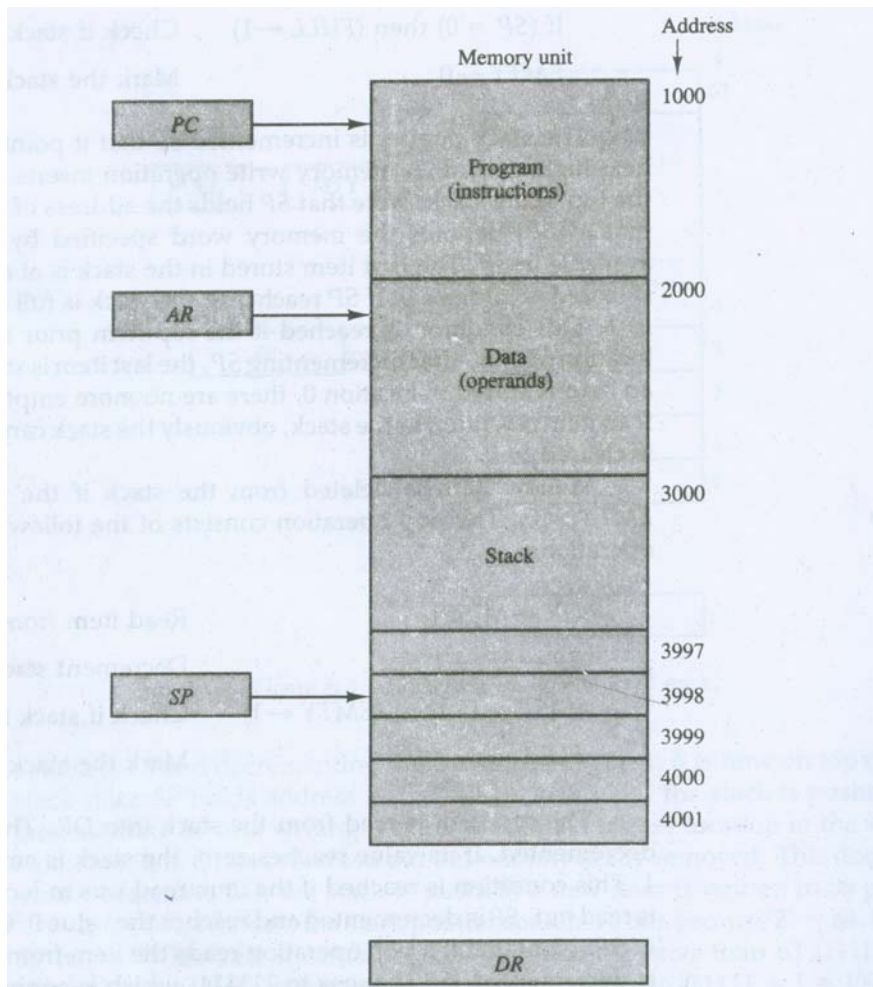


Figure 8-4 Computer memory with program, data, and stack segments.

- The program counter PC points at the address of the next instruction in the program.
- The address registers AR points at an array of data.
- The stack pointer SP points at the top of the stack.
- The three registers are connected to a common address bus, and either one can provide an address for memory.
- PC is used during the fetch phase to read an instruction.
- AR is used during the execute phase to read an operand.
- SP is used to push or pop items into or from the stack.
- The initial value of SP is 4001 and the stack grows with decreasing addresses. Thus the first item stored in the stack is at address 4000, the second item is stored at last address that can be used for the stack is 3000.
- We assume that the item in the stack communicate with data register DR. A new item is inserted with the push operation as follow:

$$SP \leftarrow SP - 1$$

$$M[SP] \leftarrow DR$$
- The stack pointer is decremented so that it points the address of the next word.

- A memory write operation inserts the word from DR into the top of the stack.
- A new item is deleted with a pop operation as follows:
 $DR \leftarrow M[SP]$
 $SP \leftarrow SP + 1$
- The top item is read from the stack into DR.
- The stack pointer is then incremented to point at the next in the stack.

Instruction Formats Bca-Aug99

- You have already been exposed to a couple different instruction formats. In fact, there are several different types of formats.
- However each type of format usually has the following in common:
 - An operation code field that specifies the operation to be performed.
 - An address field that designates a memory address or processor registers.
 - A mode field that specifies the way the operand or the effective address is determined.
- Instructions can also vary in length. The length is typically a factor of how many addresses are specified within the instruction.
- The number of addresses used by instructions depends upon the internal organization of the computer registers. Most computers fall into one of three types of CPU organizations:
 - General register organization
 - Single accumulator organization
 - Stack organization

Three-Address Instructions

Computers with three-address instruction formats fall under the general register organization category.

The three addresses can be used to specify processor registers or memory addresses.

Example

Add	R1,	A,	B	$R1 \leftarrow M[A] + M[B]$
Add	R2,	C,	D	$R2 \leftarrow M[C] + M[D]$
Mul	X,	R1,	R2	$M[X] \leftarrow R1 * R2$

It is assumed that the computer has two processor registers, R1 and R2.

The symbol $M[A]$ denotes the operand at memory address symbolized by A.

The advantage of the three-address format is that it results in short programs when evaluating arithmetic expression.

The disadvantage is that the binary coded instructions require too many bits to specify three addresses.

Two-Address Instructions

Bca-Aug99

- Computers with one-address instruction formats also fall under the single accumulator organization category.
- The one address always specifies a memory addresses since it is *implied* that the accumulator (AC) register will be used for all data manipulations.
- The program to evaluate $X=(A+B)*(C+D)$ is as follows:

Example

```
MOV R1,  A    R1<-M[A]
ADD R1,  B,   R1<-R1+M[B]
MOV R2,  C    R2<-M[C]
ADD R2,  D    R2<-R2+M[D]
MUL R1,  R2   R1<-R1*R2
MOV X,   R1   M[X]<-R1
```

One-Address Instructions

Bca-Aug99

- Computers with one-address instruction formats fall under the accumulator organization category.
- The once address must specify a memory addresses as it is *implied* that the accumulator (AC) register will be used for all data manipulations.
- The program to evaluate $X=(A+B)*(C+D)$

Example

```
LOAD A    AC<-M[A]
ADD B     AC<-a[C]+M[B]
STORE T   M[T]<-AC
LOAD C    AC<-M[C]
ADD D     AC<-AC+M[D]
MUL T     AC<-AC*M[T]
STORE X   M[X]<-AC
```

Zero-Address Instructions

Bca-Aug99

- Computers with zero-address instruction formats fall under the stack organization category.
- Although, called a zero address instruction, one memory address must be specified when pushing or popping the stack.
- However, instructions such as ADD or MUL require no addresses.
- The following program shows how $X=(A+B)*(C+D)$ will be written for a stack organized computer. TOS stands for top of stack.

Example

```

PUSH      A      TOS<-A
PUSH      B      TOS<-B
ADD       TOS<-(A+B)
PUSH      C      TOS<-C
PUSH      D      TOS<-D
ADD       TOS<-(C+D)
MUL      TOS<-(C+D)*(A+B)
POP       X      M[X]<-TOS
    
```

Addressing Modes Bca-Aug99(Bit-May01)

Implied Mode - Operands specified implicitly in the definition of the instruction (accumulator or zero-address instruction).

Immediate Mode - Operand specified in the instruction itself. Operand field contains the actual operand to be used in conjunction with the operation specified in the instruction.

Register Mode - Address field specifies processor register. Operands are registers.

Register Indirect Mode - Instruction specified a register whose contents give the address of the operand in memory (i.e. register contains address, **not** a value).

Autoincrement & Autodecrement Mode - Same as register indirect except address of register is incremented or decremented following operation.

Direct Address Mode - Effective address equal to address part of instruction. Operand is in memory and specified by address field.

Indirect Address Mode - Address field gives address where effective address is stored in memory. Control fetches instruction from memory and uses its address part again to read the effective address.

Relative Address Mode - Content of PC is added to address part of instruction. Address part can be + or - number. Result produces an effective address relative to current (or next) instruction (used for branching).

Indexed Addressing Mode - Content of an index register is added to the address part of the instruction to obtain the effective address. The address field specifies the beginning address of a data array in memory. Each operand in the array is stored in memory relative to the beginning address. The distance between the beginning address and the address of the operand is stored in the index register.

Base Register Addressing Mode - The content of a base register is added to the address part of the instruction to obtain the effective address. This is

similar to the indexed addressing mode except that the register is now called a base register instead of an index register.

Data Transfer and Manipulation *

- Computers provide an extensive set of instructions to give the user the flexibility to carry out various computational tasks.
- The instruction set of different computers differ from each other mostly in the way the operands are determined from the address and mode fields.
- The actual operations available in the instruction set are not very different from one computer to another.
- It so happens that the binary code assignments in the operation code field are different in different computers, even for the same operation.
- It may also happen that the symbolic name given to instructions in the assembly language notation is different in different computers, even for the same instruction.
- Most computer instructions can be classified into three categories
 - Data transfer instructions
 - Data manipulation instructions
 - Program control instructions
- Data transfer instructions cause transfer of data from one location to another without changing the binary information content.
- Data manipulation instructions are those that perform arithmetic, logic, and shift operations.
- Program control instructions provide decision-making capabilities and change the path taken by the program when executed in the computer.
- The instruction set of a particular computer determines the register transfer operations and control decisions that are available to the user.

Data Transfer Instructions

- Data transfer instructions move data from one place in the computer to another without changing the data content.
- The most common transfers are between memory and processor registers, between processor registers and input or output, and between the processor registers themselves.

Table 8-5 gives a list of eight data transfer instructions used in many computers.

Name	Mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH
Input	IN
Output	OUT
Push	PUSH
Pop	POP

- The *load* instruction has been used mostly to designate a transfer from memory to a processor register, usually an accumulator.
- The *store* instruction designates a transfer from a processor register into memory.

- The *move* instruction has been used in computers with multiple CPU registers to designate a transfer from one register to another. It has also been used for data transfers between CPU registers and memory or between two memory words.
- The *exchange* instruction swaps information between two registers or a register and a memory word.
- The *input* and *output* instructions transfer data among processor registers and input or output terminals.
- The *push* and *pop* instructions transfer data between processor registers and a memory stack.

Mode	Assembly Convention	Register Transfer
Direct address	LD ADR	$AC \leftarrow M[ADR]$
Indirect address	LD @ADR	$AC \leftarrow M[M[ADR]]$
Relative address	LD \$ADR	$AC \leftarrow M[PC + ADR]$
Immediate operand	LD #NBR	$AC \leftarrow NBR$
Index addressing	LD ADR(X)	$AC \leftarrow M[ADR + XR]$
Register	LD R1	$AC \leftarrow R1$
Register indirect	LD (R1)	$AC \leftarrow M[R1]$
Autoincrement	LD (R1)+	$AC \leftarrow M[R1], R1 \leftarrow R1 + 1$

Table 8-6 shows the recommended assembly language convention and the actual transfer accomplished in each case.

- *ADR* stands for an address; *NBR* is a number or operand.
- *X* is an index register, *R1* is a processor register, and *AC* is the accumulator register.
- The @ character symbolizes an indirect address.
- The \$ character before an address makes the address relative to the program counter PC.
- The # character precedes the operand in an immediate-mode instruction. A register that is placed in parentheses after the symbolic address recognizes an indexed mode instruction.
- The register mode is symbolized by giving the name of a processor register. In the register indirect mode, the name of the register that holds the memory address is enclosed in parentheses.
- The auto increment mode is distinguished from the register indirect mode by placing a plus after the parenthesized register.
- The auto decrement mode would use a minus instead. To be able to write assembly language programs for a computer, it is necessary to know the type of instructions available and also to be familiar with the addressing modes used in the particular computer.

Data Manipulation Instructions

- The data manipulation instruction in a typical computer is usually divided into three types.
- Arithmetic instructions

- The four basic arithmetic operations are addition, subtraction, multiplication, and division.
- A list of typical arithmetic instructions is given in Table 8-7.

TABLE 8-7 Typical Arithmetic Instructions

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Subtract with borrow	SUBB
Negate (2's complement)	NEG

- The increment instruction adds 1 to the value stored in a register or memory word. One common characteristic of the increment operations when executed in processor registers is that a binary number of all 1's when incremented produces a result of all 0's.
- The decrement instruction subtracts 1 from a value stored in a register or memory word.
- A number with all 0's, when decremented, produces a number with all 1's.
- The add, subtract, multiply, and divide instructions may be available for different types of data.
- The data type assumed to be in processor registers during the execution of these arithmetic operations is included in the definition of the operation code.
- An arithmetic instruction may specify fixed-point or floating-point data, binary or decimal data, single-precision or double-precision data.

Logical and Bit Manipulation Instructions

- Logical and Bit Manipulation Instructions Logical instructions perform binary operations on strings of bits stored "in registers.
- They are useful for manipulating individual bits or a group of bits that represent binary-coded information.
- The AND instruction is used to clear a bit or a selected group of bits of an operand. For any Boolean variable x , the relationships $x \text{ AND } 0 = 0$ and $x \text{ AND } 1 = x$ dictate that a binary variable ANDed with a 0 produces a 0; but the variable does not change in value when ANDed with a 1. Therefore, the AND instruction can be used to clear bits of an operand selectively by ANDing the operand with another operand that has 0's in the bit positions that must be cleared. The AND instruction is also

called a *mask* because it masks or inserts 0's in a selected portion of an operand.

- The OR instruction is used to set a bit or a selected group of bits of an operand. For any Boolean variable x , the relationships $x + 1 = 1$ and $x + 0 = x$ dictate that a binary variable OR ed with a 1 produces a 1; but the variable does not change when ORed with a 0.
- Therefore, the OR instruction can be used to selectively set bits of an operand by OR ing it with another operand with 1's in the bit positions that must be set to 1.
- Similarly, the XOR instruction is used to selectively complement bits of an operand.
- This is because of the Boolean relationships $x \oplus 1 = x'$ and $x \oplus 0 = x$. Thus a binary variable is complemented when XORED with a 1 but does not change in value when XORED with a 0.

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
Exclusive-OR	XOR
Clear carry	CLRC
Set carry	SETC
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

- Numerical examples showing the three logic operations are given in Sec. 4-5. A few other bit manipulation instructions are included in Table 8-8. Individual bits such as a carry can be cleared, set, or complemented with appropriate instructions.

Shift Instruction

- Shifts are operations in which the bits of a word are moved to the left or right. The bit shifted in at the end of the word determines the type of shift used. Shift instructions may specify either logical.

Table 8.9 Typical Shift operation

Name	Mnemonic
Logical Shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA

Arithmetic shift left	SHLA
Rotate Right	ROR
Rotate Left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

Program Control ♣

- Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed.
- Each time an instruction is fetched from memory, the program counter is incremented so that it contains the address of the next instruction in sequence.
- After the execution of a data transfer or data manipulation instruction, control returns to the fetch cycle with the program counter containing the address of the instruction next in sequence.
- On the other hand, a program control type of instruction, when executed, may change the address value in the program counter and cause the flow of control to be altered.
- In other words, program control instructions specify conditions for altering the content of the program counter, while data transfer and manipulation instructions specify conditions for data-processing operations.
- The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution. This is an important feature in digital computers, as it provides control over the flow of program execution and a capability for branching to different program segments.
- Some typical program control instructions are listed in Table 8-10.
- The branch and jump instructions are used interchangeably to mean the same thing, but sometimes they are used to denote* different addressing modes.
- The branch is usually a one-address instruction. It is written in assembly language as BR ADR, where ADR is a symbolic name for an address.
- When executed, the branch instruction causes a transfer of the value of ADR into the program counter. Since the program counter contains the address of the instruction to be executed, the next instruction will come from location ADR. Branch and jump instructions may be conditional or unconditional. An unconditional branch instruction causes a branch to the specified address without any conditions.

TABLE 8-10 Typical Program Control Instructions

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET

Compare (by subtraction) CMP
 Test (by ANDing) TST

Status Bit Conditions

- Status bits are also called *condition-code* bits or *flag* bits.

Figure 8-8 shows the block diagram of an 8-bit ALU with a 4-bit status register.

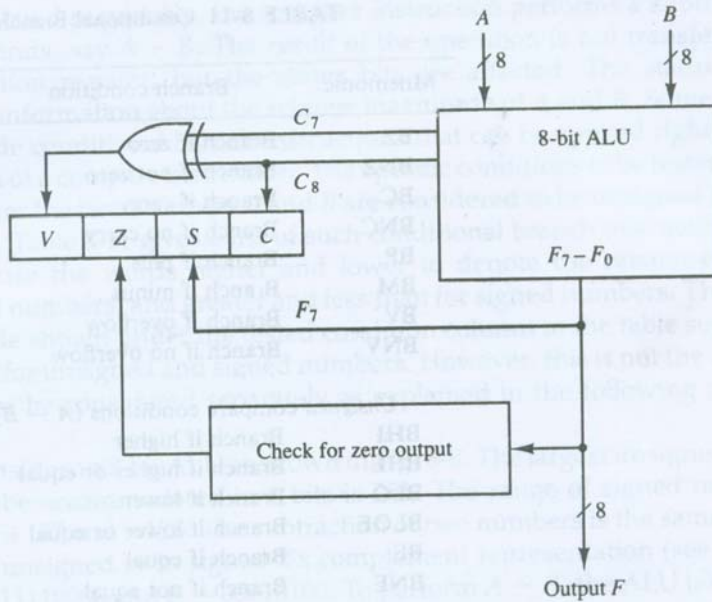


Figure 8-8 Status register bits.

- The four status bits are symbolized by C , S , Z , and V . The bits are set or cleared as a result of an operation performed in the ALU.
- Bit C (carry) is set to 1 if the end carry C_8 is 1. It is cleared to 0 if the carry is 0.
- Bit S (sign) is set to 1 if the highest-order bit F_7 is 1. It is set to 0 if the bit is 0.
- Bit Z (zero) is set to 1 if the output of the ALU contains all 0's. It is cleared to 0 otherwise. In other words, $Z = 1$ if the output is zero and $Z = 0$ if the output is not zero.
- Bit V (overflow) is set to 1 if the exclusive-OR of the last two carries is equal to 1, and cleared to 0 otherwise. This is the condition for an overflow when negative numbers are in 2's complement (see Sec. 3-3) For the 8-bit ALU, $V = 1$ if the output is greater than +127 or less than -128.

Conditional Branch Instructions

- Table 8-11 gives a list of the most common branch instructions. Each mnemonic is constructed with the letter B (for branch) and an abbreviation of the condition name.
- When the opposite condition state is used, the letter N (for no) is

TABLE 8-1 1 Conditional Branch Instruction

Mnemonic	Branch condition	Tested condition
BZ	Branch if zero	$Z = 1$
BNZ	Branch if not zero	$Z = 0$
BC	Branch if carry	$C = 1$
BNC	Branch if no carry	$C = 0$
BP	Branch if plus	$S = 0$
BM	Branch if minus	$S = 1$
BV	Branch if overflow	$V = 1$
BNV	Branch if no overflow	$V = 0$
<i>Unsigned compare conditions (A — B)</i>		
BHI	Branch if higher	$A > B$
BHE	Branch if higher or equal	$A \geq B$
BLO	Branch if lower	$A < B$
BLOE	Branch if lower or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$
<i>Signed compare conditions (A — B)</i>		
BGT	Branch if greater than	$A > B$
BGE	Branch if greater or equal	$A \geq B$
BLT	Branch if less than	$A < B$
BLE	Branch if less or equal	$A \leq B$
BE	Branch if equal	$A = B$
BNE	Branch if not equal	$A \neq B$

Types of Interrupt

External Interrupts

- Comes from input output (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source.
- Internal Interrupts
- Arise from illegal or erroneous use of an instruction or data.
- Are also known as traps.
- Eg. register overflow, attempt to divide by zero.

Software Interrupt

- Is a special call instruction that behaves like an interrupt procedure at any desired point in the program?

RISC(Reduced Instruction Set Computer) *
(Bit-Mar02)

- The concept of RISC architecture involves an attempt to reduce execution time by simplifying the instruction set of the computer. The major characteristics of a RISC processor are
 1. Relatively few instructions
 2. Relatively few addressing modes
 3. Memory access limited to load and store instructions.
 4. All operation done within the registers of the CPU
 5. Fixed length, easily decoded instruction format.
 6. Single cycle instruction execution
 7. Hardwired rather than micro programmed control.
- RISC processor architecture follows constraints which are shown below:
 1. A relatively large number of registers in the processor unit.
 2. Use of overlapped register windows to speed up procedure call and return.
 3. Efficient instruction pipeline
 4. Compiler support for efficient translation of high-level language programs into machine language programs.

The logo for Atmiya Infotech is a large, light gray circle with a white border. Inside the circle is a white square with rounded corners, containing a white circle. Below the square, the word "ATMIYA" is written in a bold, white, sans-serif font.

ATMIYA

Input-Output Organization

This chapter discusses the techniques that computers use to communicate with input and output devices. Interface units are presented to show the way that the processor interacts with the external peripherals. The procedure for asynchronous transfer is discussed; programmed I/O; interrupt initiated transfer, direct memory access and the use of input-output processors. Specific examples illustrate procedures for serial data transmission.

Peripheral Devices

- The input-output subsystem of a computer is referred to as I/O.
- This system provides an efficient mode of communication between the central system and the outside environment.
- Devices that are under the direct control of the computer are said to be *connected on-line*.
- These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be part of the total computer system.
- Input or output devices attached to the computer are also called *peripherals*

Monitor and Keyboard

- The most familiar means of entering information into a computer is through a typewriter-like keyboard that allows a person to enter alphanumeric information directly.
- Every time a key is depressed, the terminal sends a binary coded character to the computer.

CRT

- The CRT contains an electronic gun that sends an electronic beam to a phosphorescent screen in front of the tube.
- The beam can be deflected horizontally and vertically. To produce a pattern on the screen, a grid inside the CRT receives a variable voltage that causes the beam to hit the screen and make it glow at selected spots.
- Horizontal and vertical signals deflect the beam and make it sweep across the tube, causing the visual pattern to appear on the screen.
- A characteristic feature of display devices is a cursor that marks the position in the screen where the next character will be inserted

Printers

- Printers provide a permanent record on paper of computer output data or text.
- There are three basic types of character printers: daisywheel, dot matrix, and laser printers

Daisywheel

- The *daisywheel printer* contains a wheel with the characters placed along the circumference.
- . To print a character, the wheel rotates to the proper position and an energized magnet then presses the letter against the ribbon

Dot Matrix

- The dot matrix printer contains a set of dots along the printing mechanism.
- . For example, a 5x7 dot matrix printer that prints 80 characters per line has seven horizontal lines, each consisting of $5 \times 80 = 400$ dots.
- Each dot can be printed or not, depending on the specific characters that are printed on the line.

Laser Printer

- The laser printer uses a rotating photographic drum
- The laser printer uses a rotating photographic drum that is used to imprint the character images.
- The pattern is then transferred onto paper in the same manner as a copying machine

Magnetic tapes

- Magnetic tapes are used mostly for storing files of data: for example, a company's payroll record
- Access is sequential and consists of records that can be accessed one after another as the tape moves along a stationary read-write mechanism.
- It is one of the cheapest and slowest methods for storage and has the advantage that tapes can be removed when not in use.

Magnetic disks

- Magnetic disks have high-speed rotational surfaces coated with magnetic material.
- Access is achieved by moving a read-write mechanism to a track in the magnetized surface.
- Disks are used mostly for bulk storage of programs and data.

ASCII Alphanumeric Characters

- The standard binary code for the alphanumeric characters is ASCII (American Standard Code for Information Interchange).
 - It uses seven bits to code 128 characters.
 - The seven bits of the code are designated by *b1* through *b7r* with *b7* being the most significant bit.
 - The letter A, for example, is represented in ASCII as 1000001 (column 100, 'row 0001).
 - "The ASCII code contains 94 characters that can be printed and 34 nonprinting characters used for various control functions
 - The printing characters consist of the 26 uppercase letters A through Z, the 26 lowercase letters, the 10 numerals 0 through 9, and 32 special printable characters such as %, *, and \$.
-
- The 34 control characters are designated in the ASCII table with abbreviated names.

TABLE 11-1 American Standard Code for Information Interchange (ASCII)

$b_4b_3b_2b_1$	$b_7b_6b_5$							
	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	.	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	_	o	DEL

Control characters

NUL	Null	DLE	Data link escape
SOH	Start of heading	DC1	Device control 1
STX	Start of text	DC2	Device control 2
ETX	End of text	DC3	Device control 3
EOT	End of transmission	DC4	Device control 4
ENQ	Enquiry	NAK	Negative acknowledge
ACK	Acknowledge	SYN	Synchronous idle
BEL	Bell	ETB	End of transmission block
BS	Backspace	CAN	Cancel
HT	Horizontal tab	EM	End of medium
LF	Line feed	SUB	Substitute
VT	Vertical tab	ESC	Escape
FF	Form feed	FS	File separator
CR	Carriage return	GS	Group separator
SO	Shift out	RS	Record separator
SI	Shift in	US	Unit separator
SP	Space	DEL	Delete

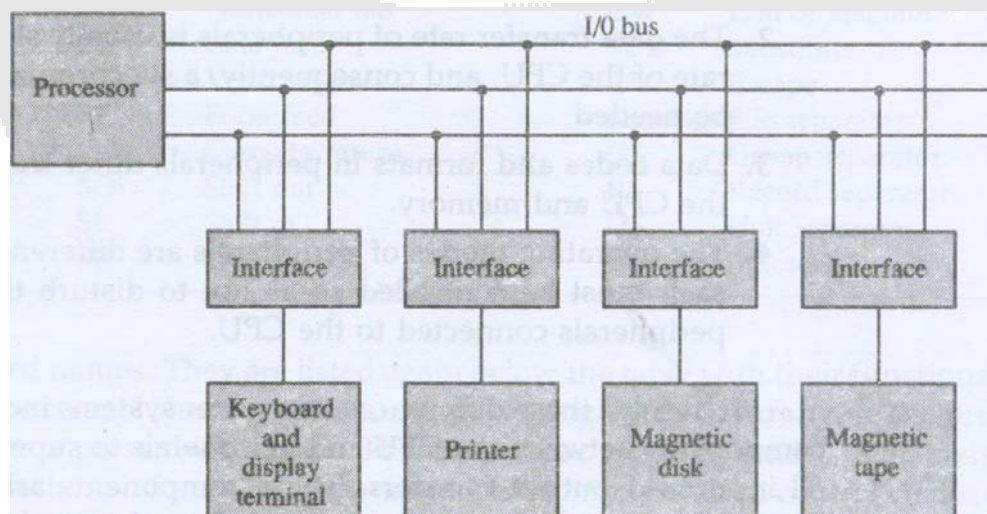
ATMIYA

Input-Output interface

- Input-output interface provides a method for transferring information between internal storage and external I/O devices.
- Peripherals, connected to computer need special communication links for interfacing with the central processing Unit.
- The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are: 'he major differences are:
- Peripherals are electromechanical and electromagnetic devices and their. Manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required. Values may be required.
- The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
- Data codes and formats in peripherals differ from the word format in the CPU and memory.
- The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

I/O Bus and Interface Modules

- The I/O bus consists of data lines, address lines, and control lines.
- The magnetic disk, printer, and terminal are employed in practically any general-purpose computer.
- The magnetic tape is used in some computers for backup storage.
- Each peripheral device has associated with it an interface unit.
- Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller.
- As you can see the I/O bus from the processor is attached to all peripheral interfaces.



- To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines.
- When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled by their interface.
- At the same time that the address is made available in the address lines, the processor provides a function code in the control lines.
- The interface selected responds to the function code and proceeds to execute it.
- The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached peripheral unit.
- There are four types of commands that an interface may receive.
- Control
 - A *control command* is issued to activate the peripheral and to inform it what to do.
 - ***For example***, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction.
 - The particular control command issued depends on the peripheral, and each peripheral receives its own distinguished sequence of control commands, depending on its mode of operation.
- Status
 - A *status command* is used to test various status conditions in the interface and the peripheral.
 - ***For example***, the computer may wish to check the status of the peripheral before a transfer is initiated.
 - During the transfer, one or more errors may occur which are detected by the interface.
 - Setting bits in a status register that the processor can read at certain intervals designates these errors.
- Data Output
 - A *data output command* causes the interface to respond by transferring data from the bus into one of its registers.
 - Consider an example with a tape unit. The computer starts the tape moving by issuing a control command.
 - The processor then monitors the status of the tape by means of a status command.
 - When the tape is in the correct position, the processor issues a data output command.
 - The interface responds to the address and command and transfers the information from the data lines in the bus to its buffer register.
 - The interface then communicates with the tape controller and sends the data to be stored on tape.

Data Input

- The *data input* command is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register.
- The processor checks if data are available by means of a status command and then issues a data input command.
- The interface places the data on the data lines, where the processor accepts them.

I/O versus Memory Bus

- The memory bus contains data, address, and read/write control lines.
- There are three ways that computer buses can be used to communicate with memory and I/O.
- Use two separate buses, one for memory and the other for I/O.
- Use one common bus for both memory and I/O but have separate control lines for each.
- Use one common bus for memory and I/O with common control lines.

Isolated versus Memory Mapped I/O

Isolated I/O

- In the isolated I/O configuration, the CPU has distinct input and output instructions, and each of these instructions is associated with the address of an interface register.
- When the CPU fetches and decodes the operation code of an input or output instruction, it places the address associated with the instruction into the common address lines.
- At the same time, it enables the I/O read (for input) or I/O write (for output) control line.
- This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word.
- On the other hand, when the CPU is fetching an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control line.
- This informs the external components that the address is for a memory word and not for an I/O interface.
- The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space. The other alternative is to use the same address space for both memory and I/O.

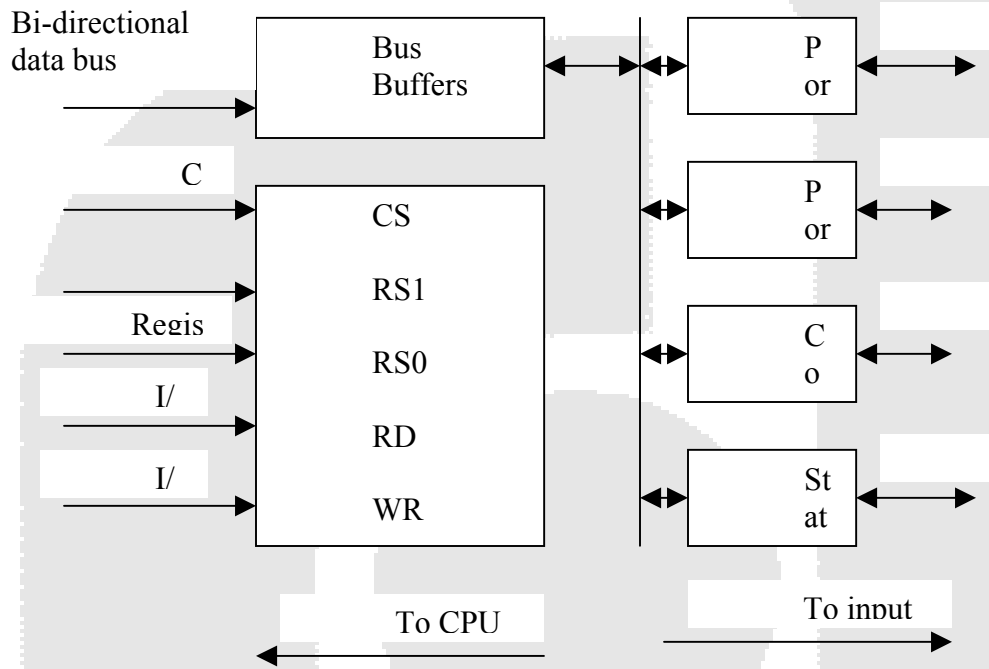
Memory Mapped I/O

- Computers with memory-mapped I/O can use memory-type instructions to access I/O data.
- It allows the computer to use the same instructions for either input-output transfers or for memory transfers.

- The advantage is that the load and store instructions used for reading and writing from memory can be used to input and output data from I/O registers.
- In a typical computer, there are more memory-reference instructions than I/O instructions.
- With memory- mapped I/O all instructions that refer to memory are also available for I/O.

Example of I/O Interface

An example of an I/O interface unit is shown in block diagram



CS	RS1	RS0	Register Selected
0	x	x	None data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

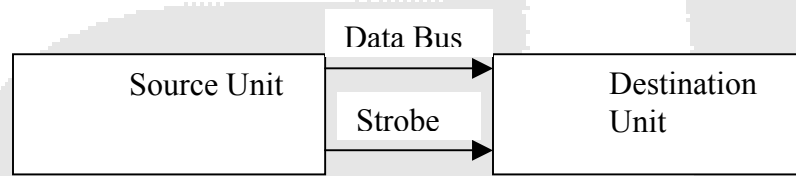
- It consists of two data registers called *ports*, a control register, a status register, bus buffers, and timing and control circuits.
- The interface communicates with the CPU through the data bus. The chip select and register select inputs determine the address assigned to the interface

- The I/O read and writes are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.
- The I/O data to and from the device can be transferred into either port *A* or port *B*.
- The interface may operate with an output device or with an input device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data.
- A magnetic disk unit transfers data in both directions but not at the same time, so the interface can use bi-directional lines.
- A command is passed to the I/O device by sending a word to the appropriate interface register. In a system like this, the function code in the I/O bus is not needed because control is sent to the control register, status information is received from the status register, and data are transferred to and from ports *A* and *B* registers. Thus the transfer of data, control, and status information is always via the common data bus.
- The distinction between data, control, or status information is determined from the particular interface register with which the CPU communicates.
- The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of operating modes.
- For example, port *A* may be defined as an input port and port *B* as an output port. A magnetic tape unit may be instructed to rewind the tape or to start the tape moving in the forward direction.
- The bits in the status register are used for status conditions and for recording errors that may occur during the data transfer. For example, a status bit may indicate that port *A* has received a new data item from the I/O device.
- Another bit in the status register may indicate that a parity error has occurred during the transfer.
- The interface registers communicate with the CPU through the bi-directional data bus. The address bus selects the interface unit through the chip select and the two register select inputs.
- A circuit must be provided externally (usually, a decoder) to detect the address assigned to the interface registers. This circuit enables the chip select (*CS*) input when the address bus selects the interface.
- The two register select inputs *RSI* and *RSO* are usually connected to the two least significant lines of the address bus.
- These two inputs select one of the four registers in the interface as specified in the table accompanying the diagram.
- The content of the selected register is transfer into the CPU via the data bus when the I/O read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.

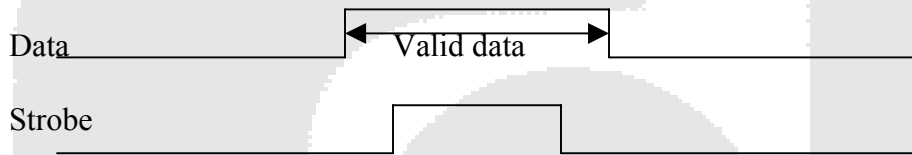
Asynchronous Data Transfer

Synchronous Transfer

- The registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous.
- Asynchronous Transfer
- The internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. In that case, the two units are said to be asynchronous to each other.
- One way of achieving this is by means of a **strobe** pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
- The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as **handshaking**.



b) Block Diagram



b) Timing diagram

Source initiated strobe for data transfer

Strobe control

- The strobe control method of asynchronous data transfer employs a single control line to time each transfer.
- Either the source or the destination unit may activate the strobe. The above Fig. shows a source-initiated transfer.
- The data bus carries the binary information from source unit to the destination unit.
- Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.
- As shown in the timing diagram of Fig. (b), the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse.
- The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- Often, the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers.

- The source removes the data from the bus a brief period after it disables its strobe pulse.
- The fact that the strobe signal is disabled indicates that the data bus does not contain valid data. New valid data will be available only after the strobe is enabled again.
- In this case the destination unit activates the strobe pulse, informing the source to provide the data.
- The source unit responds by placing the requested binary information on the data bus. The data must be valid and remain in the bus long enough for the destination unit to accept it.
- The falling edge of the strobe pulse can be used again to trigger a destination register. The destination unit then disables the strobe.
- The source removes the data from the bus after a predetermined time interval. In many computers the strobe pulse is actually controlled by the clock pulses in the CPU. The CPU is always in control of the buses and informs the external units how to transfer data.

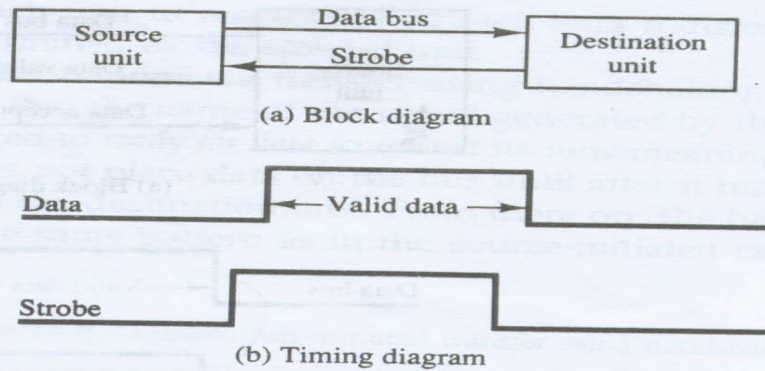


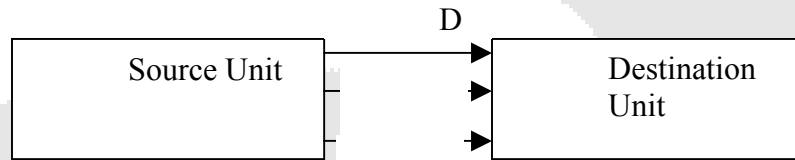
Figure 11-4 Destination-initiated strobe for data transfer.

- The disadvantage of the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus.
- Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus

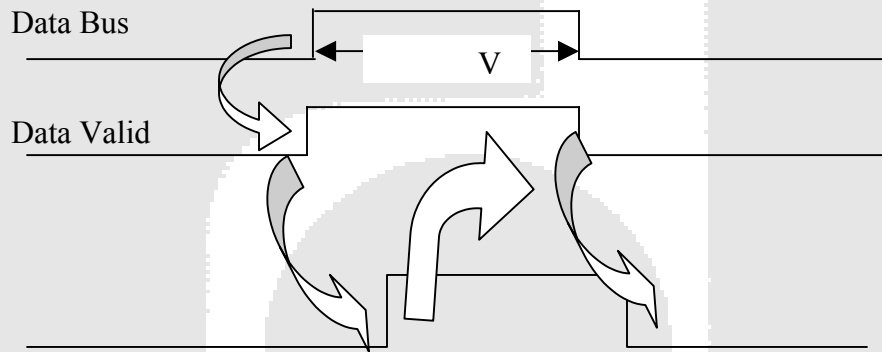
Handshaking

- The handshake method uses a second control signal that provides a reply to the unit that *two-wire control* initiates the transfer.
- The basic principle of the two-wire handshaking method of data transfer is as follows.
- One control line is in the same direction as the data flow in the bus from the source to the destination.
- The source unit to inform the destination unit whether there are valid data in the bus uses it. The other control line is in the other direction from the destination to the source.

- The destination unit to inform the source whether it can accept data uses it. The sequence of control during the transfer depends on the unit that initiates the transfer.
- The two-handshaking lines are *data valid*, which is generated by the source unit, and *data accepted*, generated by the destination unit.
- The timing diagram shows the exchange of signals between the two units. The sequence of events listed in part (c) shows the four possible states.



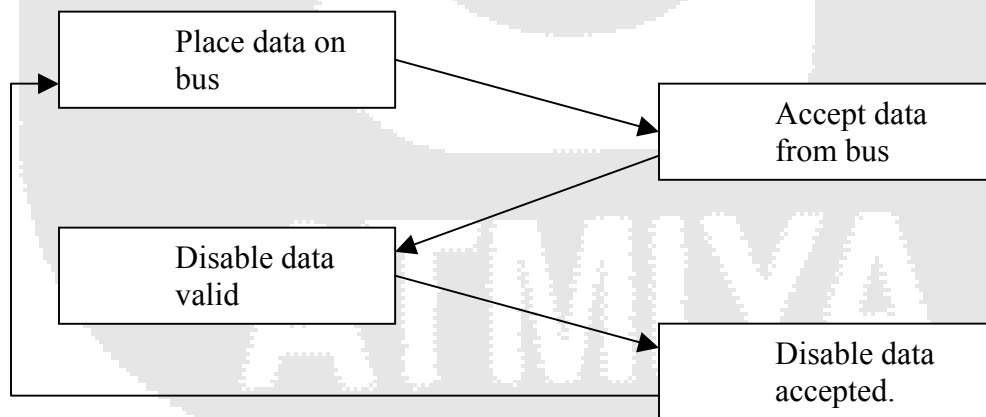
Block diagram



Timing Diagram

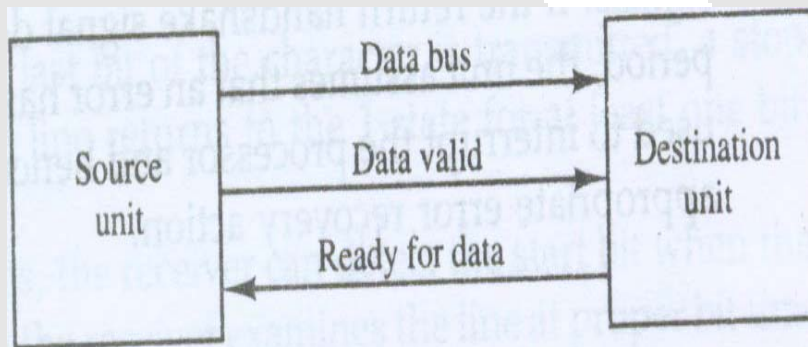
Source Unit

Destination Unit

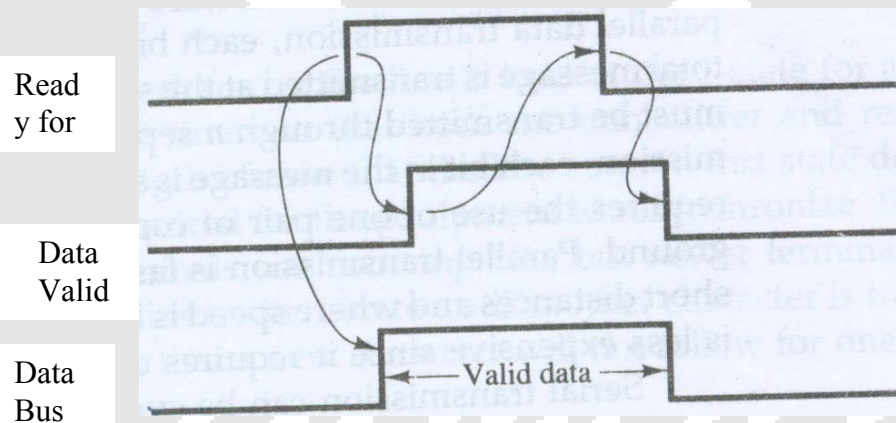


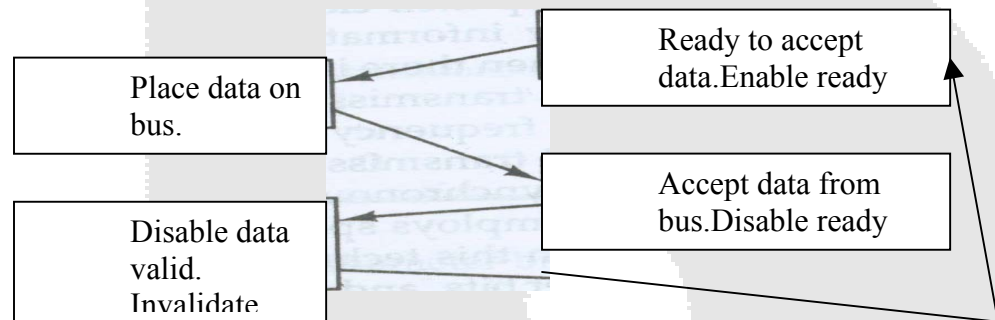
Sequence of events

- The source unit initiates the transfer by placing the 1 on the bus and enabling its *data valid* signal.
- The *data accepted* signal is active, by the destination unit after it accepts the data from the bus.
- The source then disables its *data valid* signal, which invalidates the data on the bus. Destination unit then disables its *data accepted* signal and the system goes its initial state. The source does not send the next data item until after destination unit shows its readiness to accept new data by disabling its *accepted* signal.
- The destination-initiated transfer using handshaking lines is shown in Fig.
- Note that the name of the signal generated by the destination has been changed to *ready for data* to reflect its new meaning. The source in this case does not place data on the bus until after it receives the *ready for data* signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source-initiated case.



Block diagram





Sequence of events

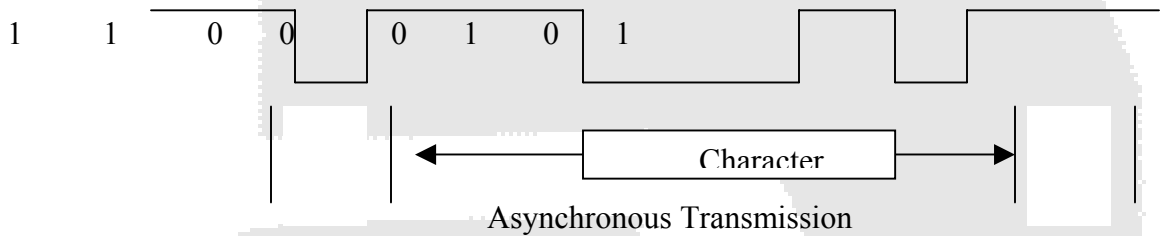
- In fact, the only difference between the source-initiated and the destination-initiated transfer is in their choice of initial state.
- The handshaking scheme provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units.
- If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a *timeout* mechanism, which produces an alarm if the data transfer is not completed within a predetermined time.
- The timeout is implemented by means of an internal clock that starts counting time when the unit enables one of its handshaking control signals.
- If the return handshake signal does not respond within a given time period, the unit assumes that an error has occurred.
- The timeout signal can be used to interrupt the processor and hence execute a service routine that takes used to interrupt the processor.

Asynchronous Serial Transfer

- The transfer of data between two units may be done in parallel or serial. Parallel data transmission, each bit of the message has its own path and the total message is transmitted at the same time.
- This means that an n -bit message must be transmitted through n separate conductor paths.
- In serial data transmission, each bit in the message is sent in sequence one at a time. This method requires the use of one pair of conductors or one conductor and a common ground. Parallel transmission is faster but requires many wires.
- It is used *in* short distances and where speed is important. Serial transmission is slower but is less expensive since it requires only one pair of conductors. Serial transmission can be synchronous or asynchronous.
- In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses.

- In long distant serial transmission, a separate clock of the frequency drives each unit.
- Synchronization signals are transmitted periodically between two units to keep their clocks in step with each other.
- In asynchronous transmission, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted.
- This is in contrast to synchronous transmission, where bits must be transmitted continuously keep the clock frequency in both units synchronized with each other.
- Synchronous serial transmission is discussed further in Sec. 11-8. A serial asynchronous data transmission technique used in many interactive terminals employs special bits that are inserted at both ends of the character code.
- With this technique, each character consists of three parts: a start bit, the character bits, and stop bits.
- The convention is that the transmitter receives at the 1-state when no characters are transmitted.
- The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character.
- The last bit called the stop bit is always a 1. An example of this format is shown in Fig.
- A transmitted character can be detected by the receiver from knowledge of the transmission rules:
- When a character is not being sent, the line is kept in the 1-state.
- The initiation of a character transmission is detected from the start bit, which is always 0.
- The character bits always follow the start bit.
- After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for at least one bit time.
- Using these rules, the receiver can detect the start bit when the line goes from 1 to 0.
- A clock in the receiver examines the line at proper bit times.
- The receiver knows the transfer rate of the bits and the number of character bits to accept. After the character bits are transmitted, one or two stop bits are sent.
- The stop bits are always in the 1-state and frame the end of the character to signify the idle or wait state. At the end of the character the line is held at the 1-state for a period of at least one or two bit times so that both the transmitter and receiver can resynchronize.
- The length of time that the line stays in this state depends on the amount of time required for the equipment to resynchronize.
- Some older electromechanical terminals use two stop bits, but newer terminals use one stop bit.
- The line remains in the 1-state until another character is transmitted.
- The stop time ensures that a new character will not follow for one or two bit times.

- As an illustration, consider the serial transmission of a terminal whose transfer rate is 10 characters per second. Each transmitted character consists of a start bit, eight information bits, and two stop bits, for a total of 11 bits.



- Ten characters per second with an 11-bit format has a transfer rate of 110 baud.
- The terminal has a keyboard and a printer. Every time a key is depressed, the terminal sends 11 bits serially along a wire.
- To print a character in the printer, an 11-bit message must be received along another wire.
- The terminal interface consists of a transmitter and a receiver. The transmitter accepts an 8-bit character from the computer and proceeds to send a serial 11-bit message into the printer line.
- The receiver accepts a serial 11-bit message from the keyboard line and forwards the 8-bit character code into the computer.
- Integrated circuits are available which are specifically designed to provide the interface between computer and similar interactive terminals. Such a circuit is called an *asynchronous communication interface* or a *universal asynchronous receiver-transmitter (UART)*.

Asynchronous Communication Interface

- The block diagram of an asynchronous communication interface is shown in Fig.
- It functions as both a transmitter and a receiver. The interface is initialized for a particular mode of transfer by means of a control byte that is loaded into its control register.
- The transmitter register accepts a data byte from the CPU through the data bus. This byte is transferred to a shift register for serial transmission. The receiver portion receives serial information into another shift register, and when a complete data byte is accumulated, it is transferred to the receiver register.

The CPU can select the receiver register to read the byte through the data bus. The

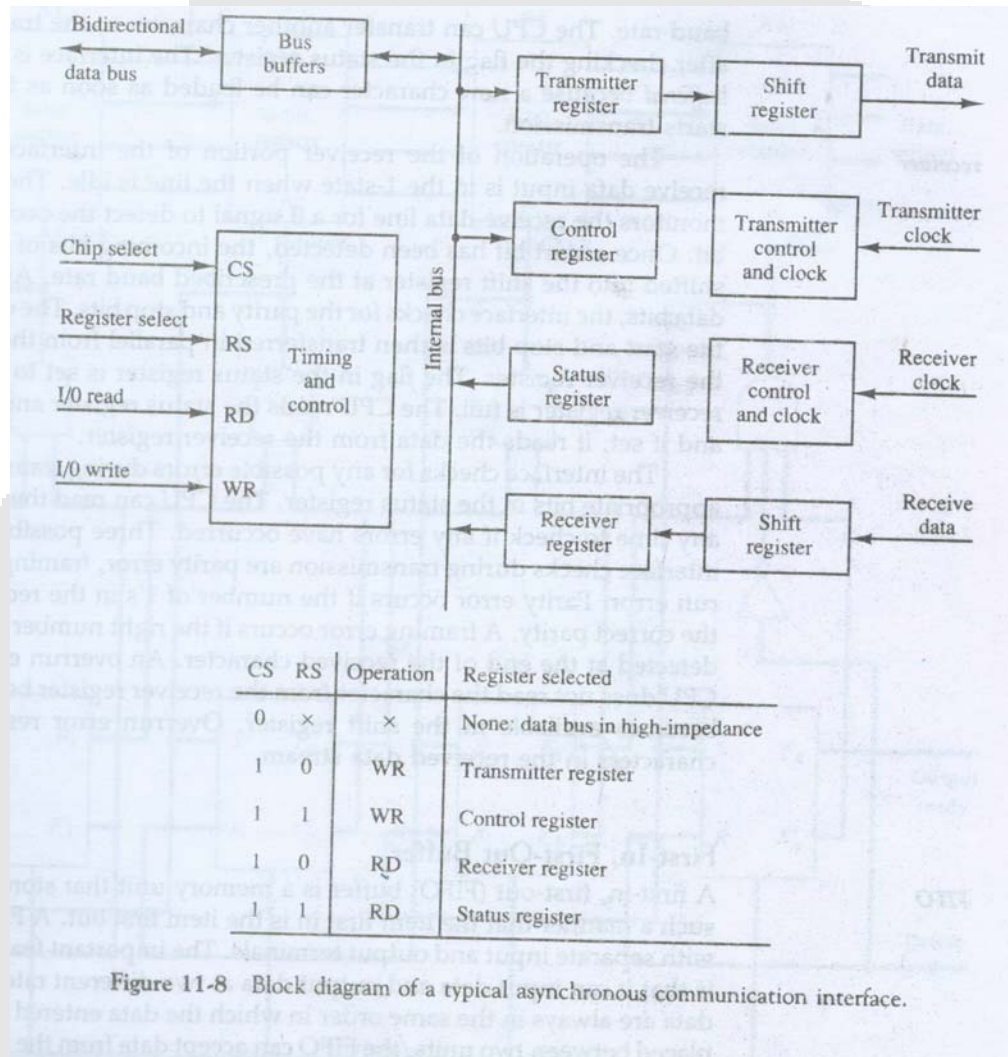


Figure 11-8 Block diagram of a typical asynchronous communication interface.

bits in the status register are used for input and output flags and for recording certain errors that may occur during the transmission. The CPU can read the status register to check the status of the flag bits and to determine if any errors have occurred.

- The chip select and the read and write control lines communicate with the CPU. The chip select (CS) input is used to select the interface through the address bus. The register select (RS) is associated with the read (RD) and write (WR) controls. Two registers are write-only and two are read-only. The register selected is a function of the RS value and the RD and WR status, as listed in the table accompanying the diagram.
- The CPU initializes the operation of the asynchronous communication interface by sending a byte to the control register.
- The initialization procedure places the interface in a specific mode of operation as it defines-certain parameters such as the baud rate to use, how many bits are in each character, whether to generate and check parity, and how many stop bits are appended to each character.
- Two bits in the status register are used as flags.

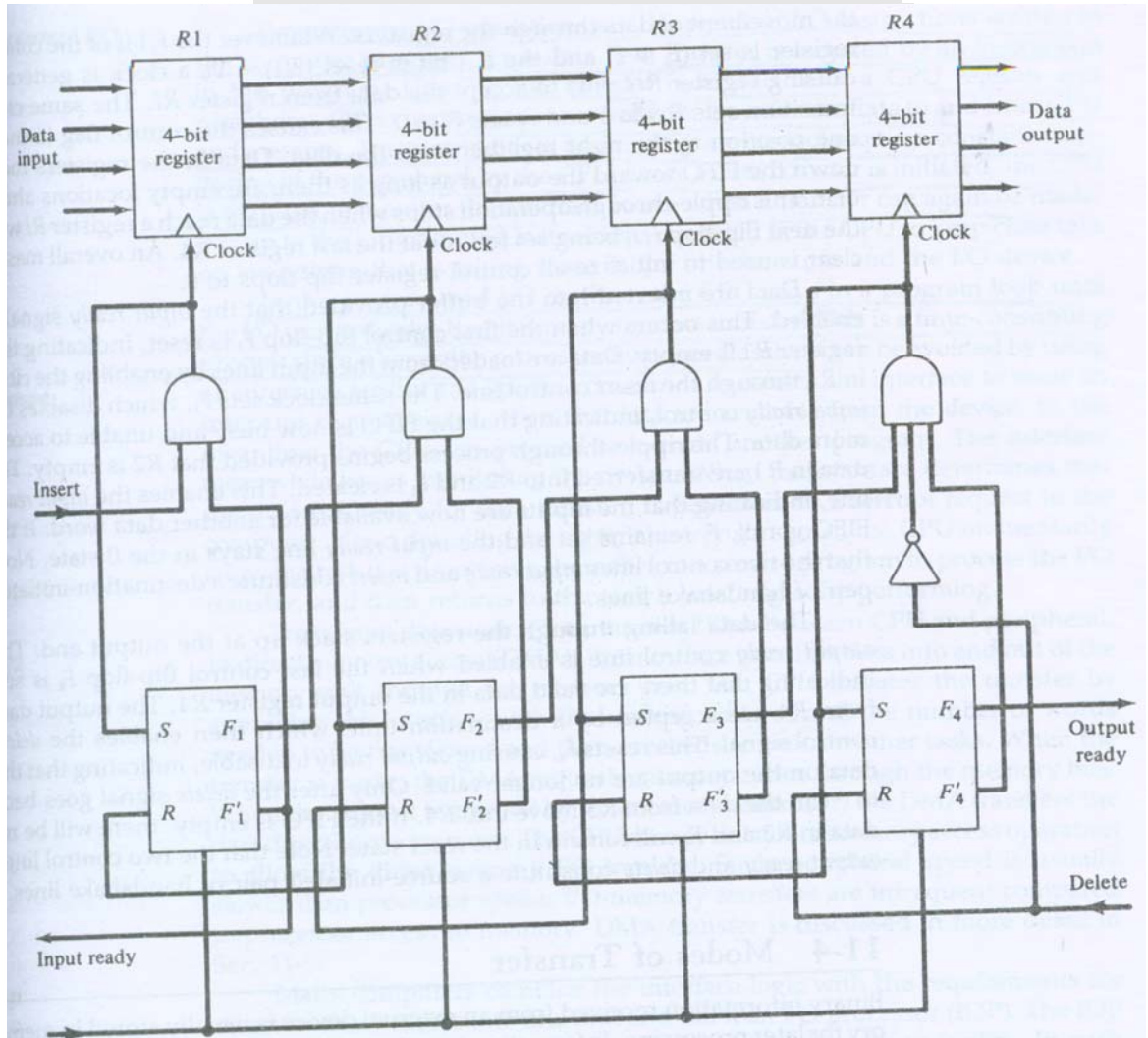
- One bit is used to indicate whether the transmitter register is empty and another bit is used to indicate whether the receiver register is full.
- The operation of the transmitter portion of the interface is as follows. The CPU reads the status register and checks the flag to see if the transmitter register is empty.
- If it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full. The first bit in the transmitter shift register is set to 0 to generate a start bit.
- The character is transferred in parallel from the transmitter register to the shift register and the appropriate number of stop bits is appended into the shift register.
- The transmitter register is then marked empty.
- The character can now be transmitted one bit at a time by shifting the data in the shift register at the specified baud rate.
- The CPU can transfer another character to the transmitter register after checking the flag in the status register.
- The interface is said to be *double buffered* because a new character can be loaded as soon as the previous one starts transmission.

First In First Out Buffer

- A first-in, first-out (FIFO) buffer is a memory unit that stores information in such a manner that the item first in is the item first out. A FIFO buffer comes with separate input and output terminals.
- The important feature of this buffers that it can input data and output data at two different rates and the output data are always in the same order in which the data entered the buffer.
- When placed between two units, the FIFO can accept data from the source unit at one rate of transfer and deliver the data to the destination unit at another rate.
- If the source unit is slower than the destination unit, the buffer can be filled with data at a slow rate and later emptied at the higher rate.

ATMIYA

- If the source is faster than the destination, the FIFO is useful for those



cases where the source data arrive in bursts that fill out the buffer but the time between bursts is long enough for the destination unit to empty some or all the information from the buffer. Thus a FIFO buffer can be useful in some applications when data are transferred asynchronously.

- The logic diagram of a typical 4 x 4 FIFO buffer is shown in Fig. It consists of four 4-bit registers R_i , $i = 1, 2, 3, 4$, and a control register with flip-flops F_i , $i = 1, 2, 3, 4$, one for each register.
- The FIFO can store four words of four bits each. Increasing the number of bits in each register can increase the number of bits per word and increasing the number of registers can increase the number of words.
- A flip-flop F_i in the control register that is set to 1 indicates that a 4-bit data word is stored in the corresponding register R_i . A 0 in F_i indicates that the corresponding register does not contain valid data. The control register directs the movement of data through the registers.

- Whenever the F_i bit of the control register is set ($F_i = 1$) and the F_{i+1} bit is reset ($F_{i+1} = 0$), a clock is generated causing register R_{i+1} to accept the data from register R_i .
- The same clock transition sets F_{i+1} to 1 and resets F_i to 0. This causes the control flag to move one position to the right together with the data. Data in the registers move down the FIFO toward the output as long as there are empty locations ahead of it.
- This ripple-through operation stops when the data reach a register R_i with the next flip-flop F_{i+1} being set to 1, or at the last register R_4 . An overall master clear is used to initialize all control register flip-flops to 0.
- Data are inserted into the buffer provided that the *input ready* signal is enabled. This occurs when the first control flip-flop F_1 is reset, indicating that register R_1 is empty.
- Data are loaded from the input lines by enabling the clock in R_1 through the *insert* control line.
- The same clock sets F_1 , which disables the *input ready* control, indicating that the FIFO is now busy and unable to accept more data. The ripple-through process begins provided that R_1 is empty.
- The data in R_1 are transferred into R_2 and F_1 is cleared. This enables the *input ready* line, indicating that the inputs are now available for another data word. If the FIFO is full, F_1 remains set and the *input ready* line stays in the 0 state.
- Note that the two control lines *input ready* and *insert* constitute a destination-initiated pair of handshake lines.
- The data falling through the registers stack up at the output end. The *output ready* control line is enabled when the last control flip-flop F_4 is set, indicating that there are valid data in the output register R_4 .
- The output data from R_4 are accepted by a destination unit, which then enables the *delete* control signal. This resets F_4 , causing *output ready* to disable, indicating that the data on the output are no longer valid.
- Only after the *delete* signal goes back to 0 can the data from R_3 move into R_4 . If the FIFO is empty, there will be no data in R_3 and F_4 will remain in the reset state. Note that the two control lines *output ready* and *delete* constitute a source-initiated pair of handshake lines.

Modes of transfer

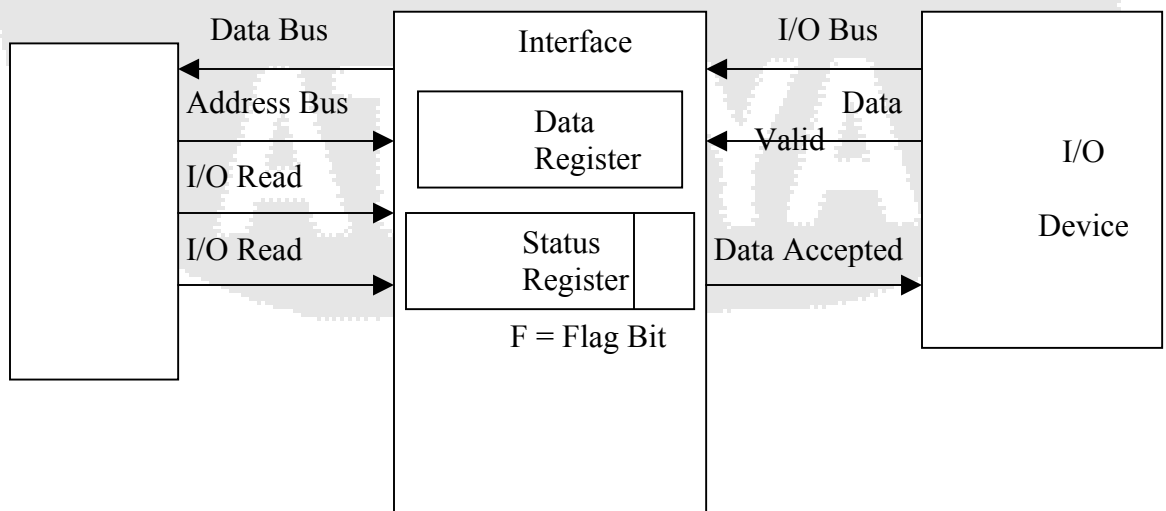
- Data transfer to and from peripherals may be handled in one of three possible modes:
- Programmed I/O
- Programmed I/O operations are the result of I/O instructions written in the computer program.
- Each data item transfer is initiated by an instruction in the program.
- Usually, the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU and memory.
- Transferring data under program control requires constant monitoring of the peripheral by the CPU.
- Once a data transfer are needed to transfer is initiated, the CPU is required to monitor the interface to see when transfer can be made. It is up to the

programmed instructions executed in the CPU to keep close tabs on everything that is taking place in the interface unit and the I/O device. In

- In the programmed I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time-consuming process since it keeps the processor busy needlessly. It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device.
- Transfer of data under programmed I/O is between CPU and peripheral. In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks. When the transfer is made, the DMA requests memory cycles through the memory bus.
- Many computers combine the interface logic with the requirements for direct memory access into one unit and call it an I/O processor (IOP). The IOP can handle many peripherals through a DMA and interrupt facility.

Example of Programmed I/O

- In the programmed I/O method, the I/O device does not have direct access to memory.
- A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory.
- Other instructions may be needed to verify that the data are available from the device and to count the numbers of words transferred.
- An example of data transfer from an I/O device through an interface into the CPU is shown in Fig.



Interrupt-initiated I/O

- An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility.
- While the CPU is running a program, it does not check the flag.
- However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set.
- The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.

Priority Interrupt

- A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously.
- The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced.
- Higher-priority interrupt levels are assigned to requests, which, if delayed or interrupted, could have serious consequences. Devices with high-speed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority.
- When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first.
- How the priority is determined????
- A polling procedure is used to identify the highest-priority source by software means.
- In this method there is one common branch address for all interrupts.
- The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence.
- The order in which they are tested determines the priority of each interrupt.
- The highest-priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise, the next-lower-priority source is tested, and so on.
- A hardware priority-interrupt unit functions as an overall manager in an interrupt system environment.
- It accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination.
- To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required because the hardware priority-interrupt unit establishes all the decisions.

Daisy Chaining Priority

- Either a serial or a parallel connection of interrupt lines can establish the hardware priority function. The serial connection is known as the daisy-chaining method.
- The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt.
- The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain. This method of connection between three devices and the CPU is shown in Fig.
- The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU.
- When no interrupts are pending, the interrupt line stays in the high-level state and the CPU recognizes no interrupts. This is equivalent to a negative logic OR operation.
- The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device I at its *PI* (priority input).
- The acknowledge signal passes on to the next device through the *PC* (priority out) output only if device I is not requesting an interrupt.
- If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output.
- It then proceeds to insert its own interrupt. *Vector address (VAD)* vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.
- A device with a 0 in its *PI* input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked.
- A device that is requesting an interrupt and has a 1 in its *PI* input will intercept the acknowledge signal by placing a 0 in its PO output.
- If the device does not have pending interrupts, it transmits the acknowledge signal to the next device placing a 1 in its PO output. Thus the device with $PI = 1$ and $PO = 0$ is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus.
- The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position, the lower is its priority.
- Figure shows the internal logic that must be included within each device when connected in the daisy-chaining scheme.
- The device sets its *RF* flip-flop when it wants to interrupt the CPU. The output of the *RF* flip-flop goes through an open-collector inverter; a circuit that provides the wired logic for the common interrupts line. If $PI = 0$, both PO and the enable line to VAD are equal to 0, irrespective of the value of *RF*. If $PI = 1$ and $RF = 0$, then $PO = 1$ and the vector address is disabled.

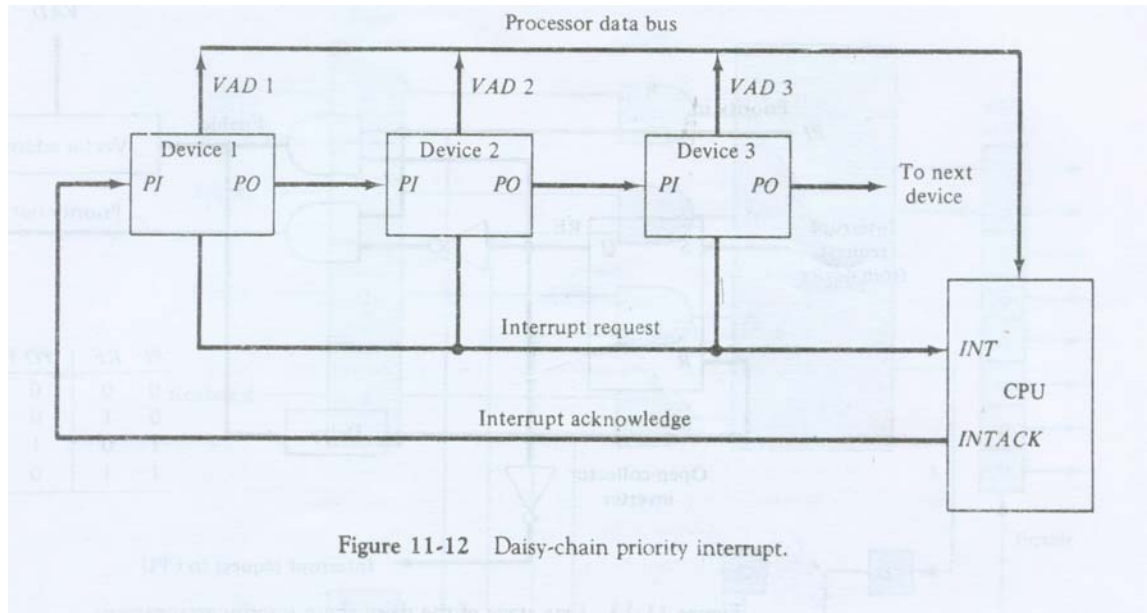


Figure 11-12 Daisy-chain priority interrupt.

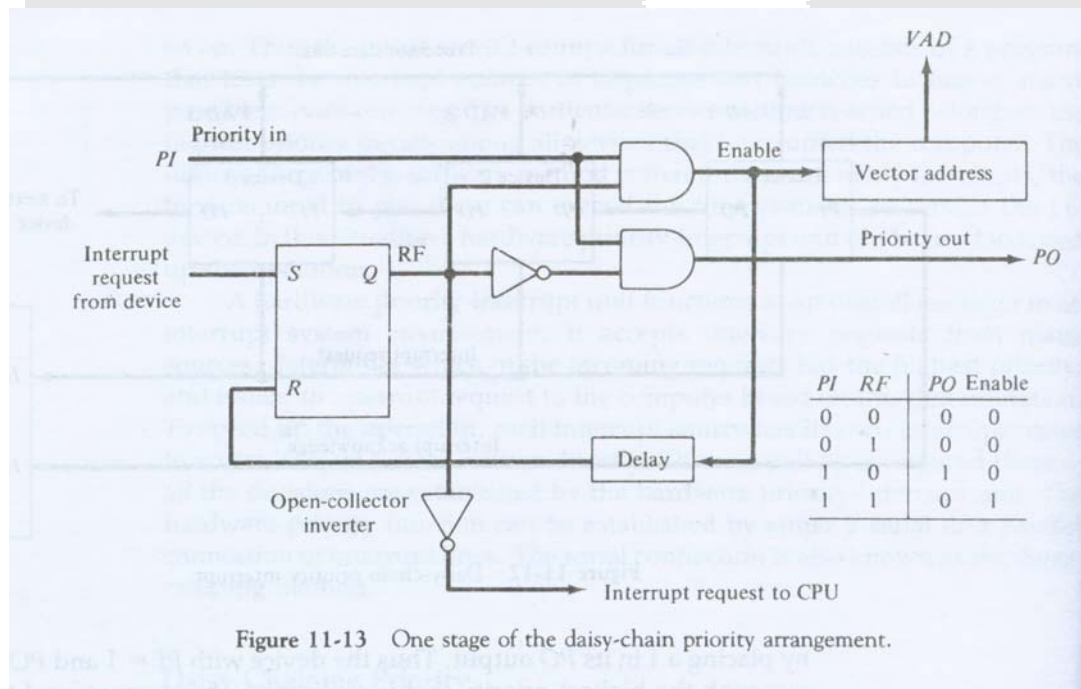
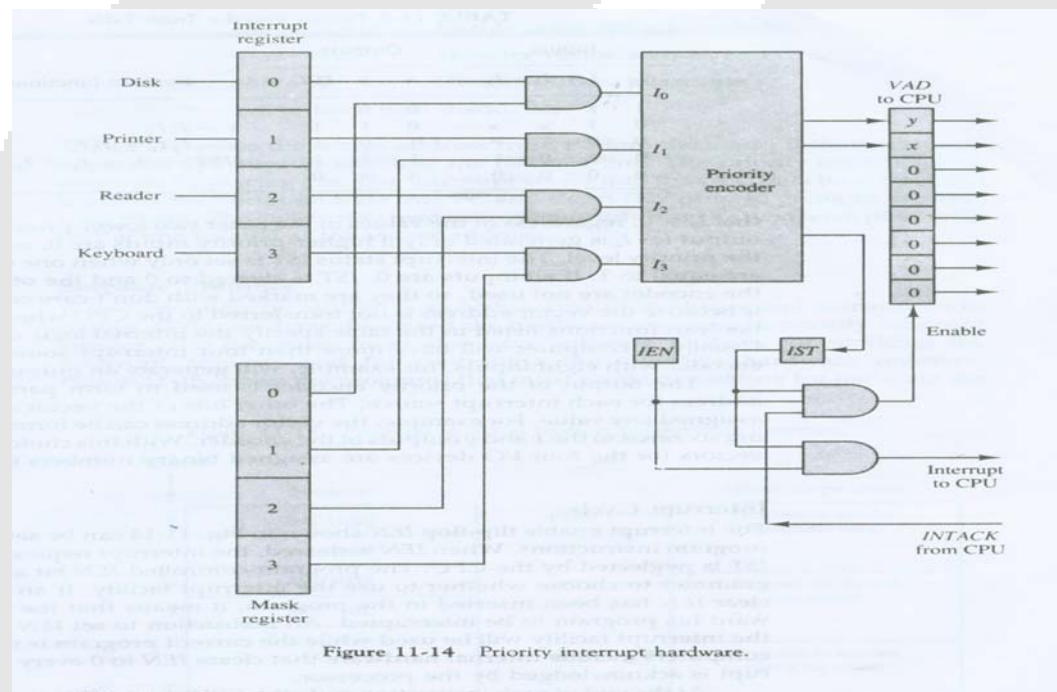


Figure 11-13 One stage of the daisy-chain priority arrangement.

- This condition passes the acknowledge signal to the next device through PO.
- The device is active when $PI = 1$ and $RF = 1$. This condition places a 0 in PO and enables the vector address for the data bus. It is assumed that each device has its own distinct vector address.
- The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

Parallel Priority Interrupt

- The parallel priority interrupt method *uses* a register whose bits are set separately by the interrupt signal from each device.
- Priority is established according to the position of the bits in the register. In addition to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request.
- One stage of the daisy-chain priority arrangement. *Priority logic* lower-priority interrupts while a higher-priority device is being serviced.
- The priority logic for a system of four interrupt sources is shown in Fig.11.14



- It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions.
- The magnetic disk, being a high-speed device, is given the highest priority.
- The printer has the next priority, followed by a character reader and a keyboard.
- The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register.
- Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder.
- In this way an interrupt is recognized only if the program sets its corresponding mask bit to 1.

- The priority encoder generates two bits of the vector address, which is transferred to the CPU. Another output from the encoder sets an interrupt status flip-flop *IST* when an interrupt that is not masked occurs.
- The interrupt enable flip-flop *IEN* can be set or cleared by the program to provide an overall control over the interrupt system.
- The outputs of *IST* Ended with *IEN* provide a common interrupt signal. For the CPU.
- The interrupt acknowledge *INTACK* signal from the CPU enables the bus buffers in the output register and a vector address *VAD* is placed into the data bus.
- We will now explain the priority encoder circuit and then discuss the interaction between the priority interrupt controller and the CPU.

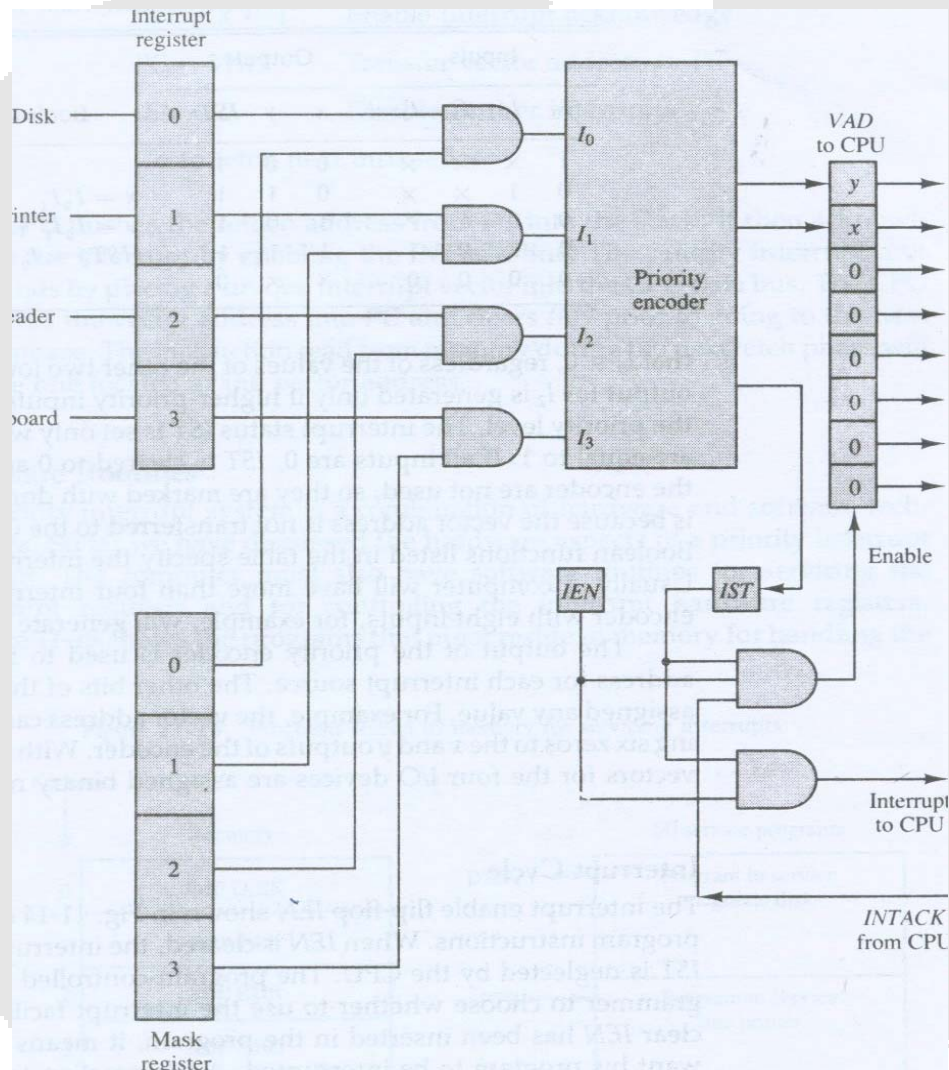


Figure 11.14 Priority Interrupt hardware

Priority Encoder

The priority encoder is a circuit that implements the priority function. The logic of the priority encoder is such that if two or more inputs arrive at the same time, the input having the highest priority will take precedence. The truth table of a four-input priority encoder is given in Table

Priority Encoder Truth Table

The X'S in the table designate don't-care conditions.

Input I_0 has the highest priority; so regardless of the values of other inputs, when this input is 1, the output generates an output $xy = 00$. I_1 has the next priority level. The output is 01 if $I_1 = 1$ provided that $I_0 = 0$, regardless of the values of the other two lower-priority inputs.

The output for I_2 is generated only if higher-priority inputs are 0, and so on down the priority level.

The interrupt status IST is set only when one or more inputs are equal to 1. If all inputs are 0, IST is cleared to 0 and the other outputs of the encoder are not used, so they are marked with don't-care conditions. This is because the vector address is not transferred to the CPU when $IST = 0$.

The Boolean functions listed in the table specify the internal logic of the encoder. Usually, a computer will have more than four interrupt sources. A priority encoder with eight inputs, for example, will generate an output of three bits.

The output of the priority encoder is used to form part of the vector address for each interrupt source.

The other bits of the vector address can be assigned any value. For example, the vector address can be formed by appending six zeros to the x and y outputs of the encoder.

With this choice the interrupt vectors for the four I/O devices are assigned binary numbers 0, 1, 2, and 3.

Interrupt Cycle

The interrupt enable flip-flop JEW shown in Fig. 11-14 can be set or cleared by program instructions. When IEN is cleared, the interrupt request coming from IST is neglected by the CPU.

The program-controlled IEN bit allows the programmer to choose whether to use the interrupt facility. If an instruction to clear IEN has been inserted in the program, it means that the user does not want his program to be interrupted.

An instruction to set IEN indicates that the interrupt facility will be used while the current program is running. Most computers include internal hardware that clears IEN to 0 every time an interrupt is acknowledged by the processor.

At the end of each instruction cycle the CPU checks IEN and the interrupt signal from IST . If either is equal to 0, control continues with the next instruction. If both IEN and IST are equal to 1, the CPU goes to an interrupt cycle.

During the interrupt cycle the CPU performs the following sequence of micro-operations:

- $SP \leftarrow SP - 1$ Decrement stack pointer
- $M[SP] \leftarrow PC$ Push PC into stack
- $INTACK \leftarrow 1$ Enable interrupt acknowledge
- $PC \leftarrow VAD$ Transfer vector address to PC
- $IEN \leftarrow 0$ Disable further interrupts
- Go to fetch next instruction

The CPU pushes the return address from PC into the stack.

It then acknowledges the interrupt by enabling the *INTACK* line.

The priority interrupt unit responds by placing a unique interrupt vector into the CPU data bus.

The CPU transfers the vector address into PC and clears *IEN* prior to going to the next fetch phase.

The instruction read from memory during the next fetch phase would be the one located at the vector address.

Software Routines

A priority interrupt system is a combination of hardware and software techniques. So far we have discussed the hardware aspects of a priority interrupt system. The computer must also have software routines for servicing the interrupt requests and for controlling the interrupt hardware registers.

Figure 11-15 shows the programs that must reside in memory for handling the

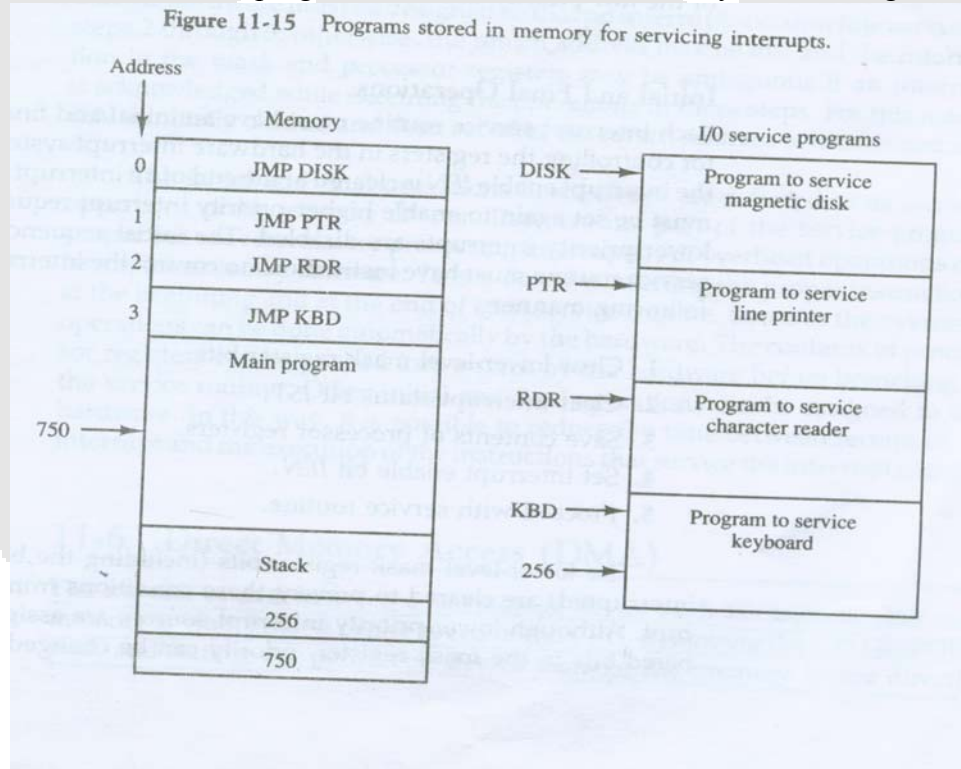


Figure. Programs stored in memory for servicing interrupts.

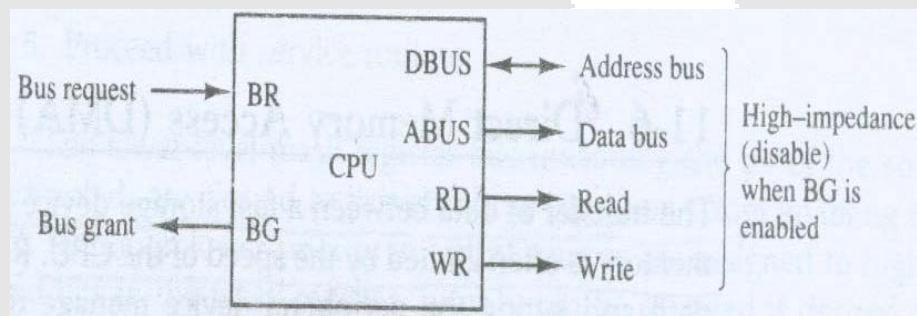
- Each device has its own service program that can be reached through a jump (JMP) instruction stored at the assigned vector address.
- The symbolic name of each routine represents the starting address of the service program.
- The stack shown in the diagram is used for storing the return address after each interrupt.
- To illustrate with a specific example assume that the keyboard sets its interrupt bit while the CPU is executing the instruction in location 749 of the main program.
- At the end of the instruction cycle, the computer goes to an interrupt cycle. It stores the return address 750 in the stack and then accepts the vector address 0000011 from the bus and transfers it to *PC*.
- The instruction in location 3 is executed next, resulting in transfer of control to the KBD routine. Now suppose that the disk sets its interrupt bit when the CPU is executing the instruction at address 255 in the KBD program.
- Address 256 is pushed into the stack and control is transferred to the DISK service program.
- The last instruction in each routine is a return from interrupts instruction.
- When the disk service program is completed, the return instruction pops the stack and places 256 into *PC*. This returns control to the KBD routine to continue servicing the keyboard. At the end of the KBD program, the last instruction pops the stack and returns control to the main program at address 750.
- Thus, a higher-priority device can interrupt a lower-priority device. It is assumed that the time spent in servicing the high-priority interrupt is short compared to the transfer rate of the low-priority device so that no loss of information takes place.

Initial And Final Operations

- The initial sequence of each interrupt service routine must have instructions to control the interrupt hardware in the following manner:
 - Clear lower-level mask register bits.
 - Clear interrupt status bit *IST*.
 - Save contents of processor registers.
 - Set interrupt enable bit *IEN*.
 - Proceed with service routine.
- The final sequence in each interrupt service routine must have instructions to control the interrupt hardware in the following manner:
 - Clear interrupt enable bit *JEN*.
 - Restore contents of processor registers.
 - Clear the bit in the interrupt register belonging to the source that has been serviced.
 - Set lower-level priority bits in the mask register.
 - Restore return address into *PC* and set *IEN*.

Direct Memory Access

- The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU.
- Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access.
- During DMA transfer, the CPU is idle and has no control of the memory buses.
- A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.
- The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals.
- Figure below shows two control signals in the CPU that facilitate the DMA transfer.



The *bus request (BR)* input is used by the DMA controller to request the CPU to relinquish control of the buses.

- When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus, and the read and writes lines into a high-impedance state.
- The high-impedance state behaves like an open circuit, which means that the output is disconnected and does not have logic significance.
- The CPU activates the *bus grant (BG)* output to inform the external DMA that the buses are in the high-impedance state.
- The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention. When the DMA terminates the transfer, it disables the bus request line.
- The CPU disables the bus grant, takes control of the buses, and returns to its normal operation. When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways.
- In *DMA burst transfer*, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses. This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred

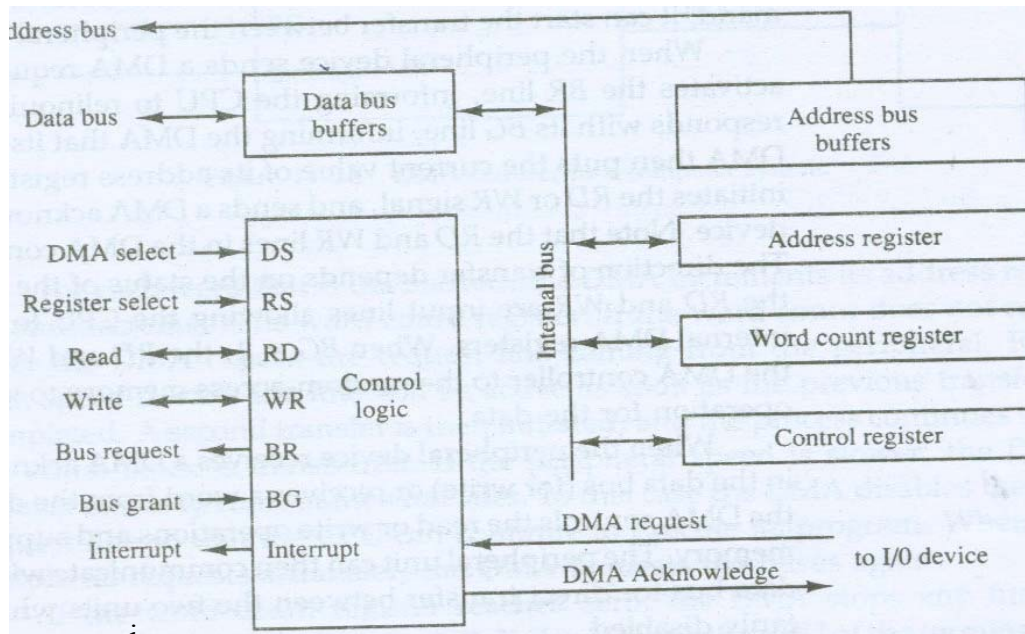
- An alternative technique called *cycle stealing* allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to "steal" one memory cycle.

DMA Controller

- The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. In addition, it needs an address register, a word count register, and a set of address lines are used for direct communication with the memory.
- The word count register specifies the number of words that must be transferred.
- The data transfer may be done directly between the device and memory under control of the DMA. Figure shows the block diagram of a typical DMA controller.
- The unit communicates with the CPU via the data bus and control lines.
- The CPU through the address bus selects the registers in the DMA by enabling the DS (DMA select) and RS (register select) inputs.
- The RD (read; and WR (write) inputs are bi-directional. When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.
- When BG = 1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.
- The DMA communicates with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure.
- The DMA controller has three registers: an address register, a word count register, and a control register.
- The address register contains an address to specify the desired location in memory. The address bits go through bus buffers into the address bus.
- The address register is incremented after each word that is transferred to memory.
- The word count register holds the number of words to be transferred. This register is decremented by one after each word transfer and internally tested for zero.

The control register specifies the mode of transfer. All registers in the DMA appear to the CPU as I/O interface registers.

Thus the CPU can read from or write into the DMA registers under program control via the data bus.



realizes the DMA by sending the following information through the data bus.

- The starting addresses of the memory block where data are available (for read) or where data are to be stored (for write).
- The word count, which is the number of words in the memory block
- Control to specify the mode of transfer such as read or write
- A control to start the DMA transfers.

DMA Transfer

- The position of the DMA controller among the other components in a computer system is illustrated in Fig.

ATMIYA

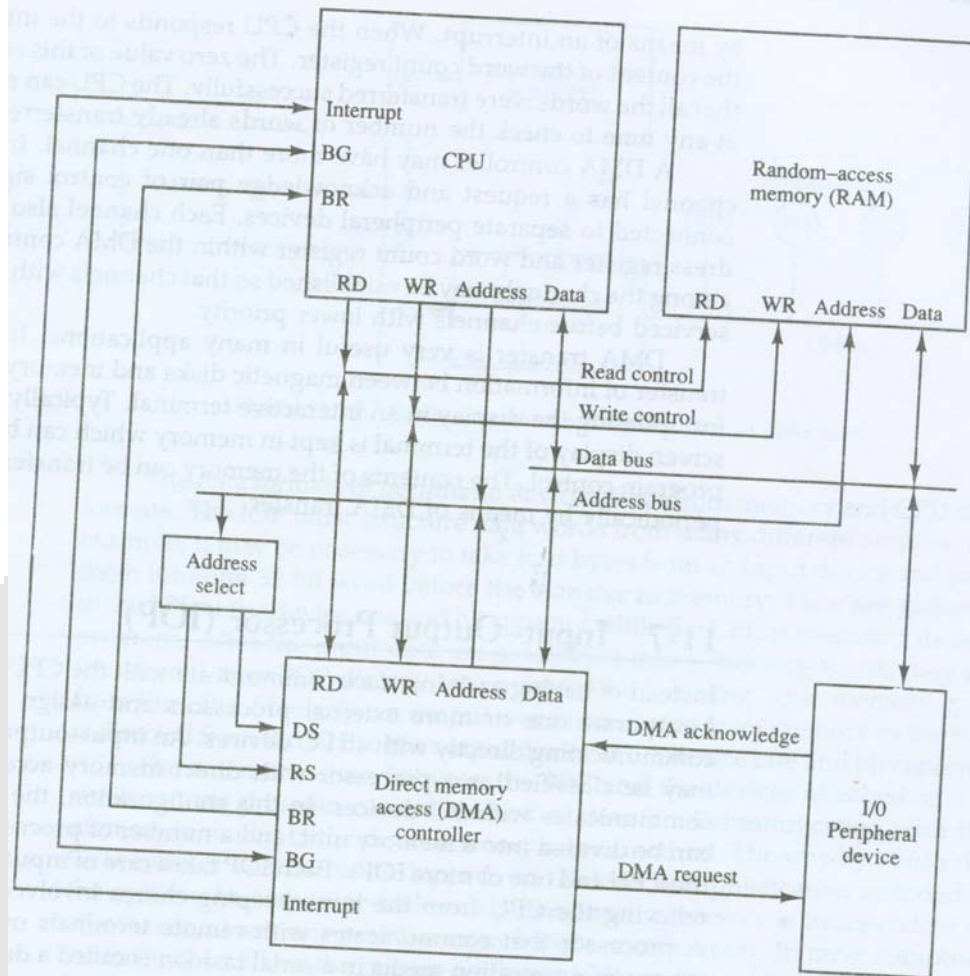


Figure 11-18 DMA transfer in a computer system.

- The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.
- When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses.
- The CPU responds with its BG line, informing the DMA that its buses are disabled.
- The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device. Note that the RD and WR lines in the DMA controller are bi-directional.
- The direction of transfer depends on the status of the BG line. When BG = 0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers.
- When BG = 1, the RD and WR are output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.

- When the peripheral device receives a DMA acknowledge, it puts a word in the data bus (for write) or receives a word from the data bus (for read).
- Thus the DMA controls the read or write operations and supplies the address for the memory
- The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.
- For each word that is transferred, the DMA increments its address registers and decrements its word count registers.
- If *the* word count does not reach zero, the DMA checks the request line coming from the peripheral. For a high-speed device, the line will be active as soon as the previous transfer is completed.
- A second transfer is then initiated, and the process continues until the entire block is transferred. If the peripheral speed is slower, the DMA request line may come somewhat later. In this case the DMA disables the bus request line so that the CPU can continue to execute its program.
- When the peripheral requests a transfer, the DMA requests the buses again. If the word count registers reaches zero, the DMA stops any further transfer and removes its bus request. It also informs the CPU of the termination by means of an interrupt
- A DMA controller may have more than one channel. In this case, each channel has a request and acknowledges pair of control signals, which are connected to separate peripheral devices.

Input-Output Processor

- The IOP is similar to a CPU except that it is designed to handle the details of I/O processing.
- Unlike the DMA controller that must be set up entirely by the CPU, the IOP can fetch and execute its own instructions.
- IOP instructions are specifically designed to facilitate I/O transfers. In addition, the IOP can perform other processing tasks, such as arithmetic, logic, branching, and code translation.
- The block diagram of a computer with two processors is shown in Fig.
- The memory unit occupies a central position and can communicate with each processor by means of direct memory access.
- The CPU is responsible for processing data needed in the solution of computational tasks.
- The IOP provides a path for transfer of data between various peripheral devices and the memory unit.
- The CPU is usually assigned the task of initiating the I/O program. From then on the IOP operates independent of the CPU and continues to transfer data from external devices and memory.
- The communication between the IOP and the devices attached to it is similar to the program control method of transfer.
- Communication with the memory is similar to the direct memory access method. The way by which the CPU and IOP communicate depends on the level of sophistication included in the system.

- In very-large-scale computers, each processor is independent of all others and any one processor can initiate an operation. In most computer systems, the CPU is the master while the IOP is a slave processor.
- The CPU is assigned the task of initiating all operations, but I/O instructions are executed in the IOP. CPU instructions provide operations to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities.

Block diagram of a computer with I/O processor

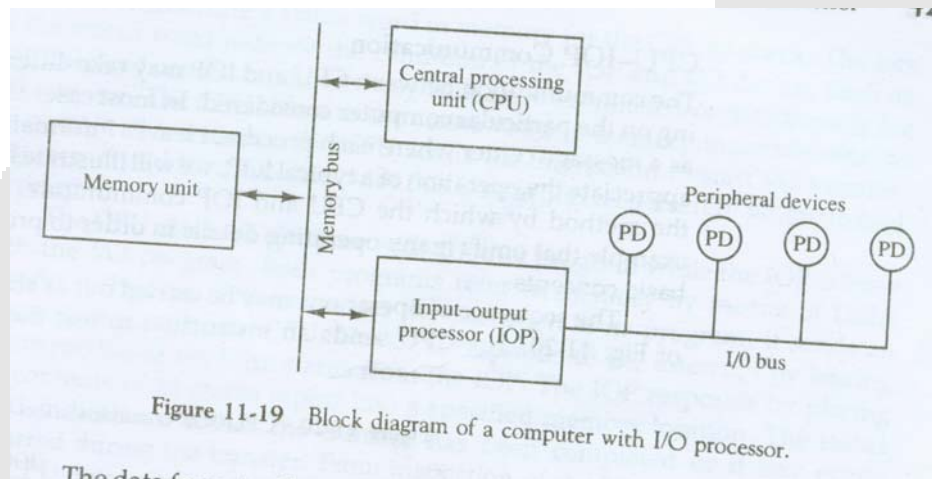


Figure 11-19 Block diagram of a computer with I/O processor.

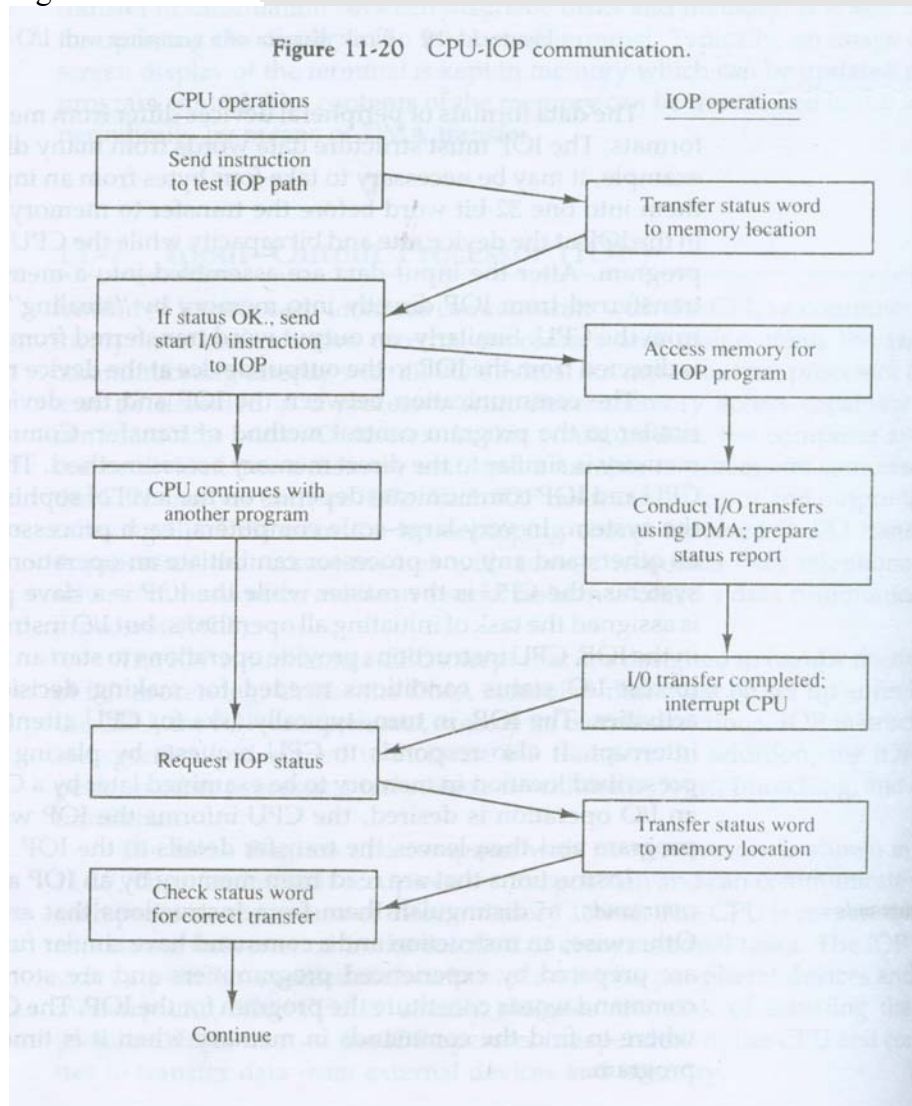
- The IOP, in turn, typically asks for CPU attention by means of an interrupt. It also responds to CPU requests by placing a status word in a prescribed location in memory to be examined later by a CPU program.
- When an I/O operation is desired, the CPU informs the IOP where to find the I/O program and then leaves the transfer details to the IOP.
- Instructions that are read from memory by an IOP are sometimes called *commands*, to distinguish them from instructions that are read by the CPU. Otherwise, an instruction and a command have similar functions. Commands are prepared by experienced programmers and are stored in memory.
- The command words constitute the program for the IOP. The CPU informs the IOP where to find the commands in memory when it is time to execute the I/O program.

CPU-IOP Communication

- The communication between CPU and IOP may take different forms, depending on the particular computer considered.
- In most cases the memory unit acts as a message center where each processor leaves information for the other.
- To appreciate the operation of a typical IOP, we will illustrate by a specific example the method by which the CPU and IOP communicate.

- This is a simplified example that omits many operating details in order to provide an overview of basic concepts
- . The sequence of operations may be carried out as shown in the flowchart of Fig. The CPU sends an instruction to test the IOP path.

Fig. CPU- IOP communication



- The IOP responds by inserting a status word in memory for the CPU to check.
- The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer.
- The CPU refers to the status word in memory to decide what to do next. If all is in order, the CPU sends the instruction to start I/O transfer.

- The memory address received with this instruction tells the IOP where to find its program.
- The CPU can now continue with another program while the IOP is busy with the I/O program. Both programs refer to memory by means of DMA transfer.
- When the IOP terminates the execution of its program, it sends an interrupt request to the CPU.
- The CPU responds to the interrupt by issuing an instruction to read the status from the IOP.
- The IOP responds by placing the contents of its status report into a specified memory location. The status word indicates whether the transfer has been completed or if any errors occurred during the transfer. From inspection of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.
- The IOP takes care of all data transfers between several I/O units and the memory while the CPU is processing another program.
- The IOP and CPU are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory.
- It is not possible to saturate the memory by I/O devices in most systems, as the speed of most devices is much slower than the CPU. However, some very fast units, such as magnetic disks, can use an appreciable number of the available memory cycles.
- In that case, the speed of the CPU may deteriorate because it will often have to wait for the IOP to conduct memory transfers.

Serial Communication*

- A data communication processor is an I/O processor that distributes and collects data from many remote terminals connected through telephone and other communication lines.
- It is a specialized I/O processor designed to communicate directly with data communication networks.
- A communication network may consist of any of a wide variety of devices, such as printers, interactive display devices, digital sensors, or a remote computing facility.
- With the use of a data communication processor, the computer can service fragments of each network demand in an interspersed manner and thus have the apparent behavior of serving many users at once. In this way the computer is able to operate efficiently in a time-sharing environment.
- The most striking difference between an I/O processor and a data communication processor is in the way the processor communicates with the I/O devices.
- An I/O processor communicates with the peripherals through a common I/O bus that is comprised of many data and control lines.
- All peripherals share the common bus and use it to transfer information to and from the I/O processor.

Atmiya Infotech

- A data communication processor communicates with each terminal through a single pair of wires. Both data and control information are transferred in a serial fashion with the result that the transfer rate is much slower.
- The task of the data communication processor is to transmit and collect digital information to and from each terminal, determine if the information is data or control and respond to all requests according to predetermined established procedures.
- The processor, obviously, must also communicate with the CPU and memory in the same manner as any I/O processor.
- The way that remote terminals are connected to a data communication processor is via telephone lines or other public or private communication facilities.
- Since telephone lines were originally designed for voice communication and computers communicate in terms of digital signals, some form of conversion must be used.
- The converters are called *data sets /acoustic couplers*, or *modems* (from "modulator-demodulator").
- A modem converts digital signals into audio tones to be transmitted over telephone lines and also converts audio tones from the line to digital signals for machine use.
- Various modulation schemes as well as different grades of communication media and transmission speeds are used.
- A communication line may be connected to a synchronous or asynchronous interface, depending on the transmission method of the remote terminal. An asynchronous interface receives serial data with start and stop bits in each character. This type of interface is similar to the asynchronous communication interface unit presented in Fig. 11-8.
- Synchronous transmission does not use start-stop bits to frame characters and therefore makes more efficient use of the communication link. High-speed devices use synchronous transmission to realize this efficiency.
- The modems used in synchronous transmission have internal clocks that are set to the frequency that bits are being transmitted in the communication line.
- For proper operation, it is required that the clocks in the transmitter and receiver modems remain synchronized at all times.
- The communication line, however, contains only the data bits from which the clock information must be extracted. Frequency synchronization is achieved by the receiving modem from the signal transitions that occur in the received data.
- Any frequency shift that may occur between the transmitter and receiver clocks is continuously adjusted by maintaining the receiver clock at the frequency of the incoming bit stream.
- The modem transfers the received data together with the clock to the interface unit.

- The interface or terminal on the transmitter side also uses the clock information from its modem. In this way, the same bit rate is maintained in both transmitter and receiver.
- A *half-duplex* transmission system is one that is capable of transmitting in both directions but data can be transmitted in only one direction at a time
- A *full-duplex* transmission can send and receive data in both directions simultaneously. This can be achieved by means of a four-wire link, with a different pair of wires dedicated to each direction of transmission.
- The communication lines, modems, and other equipment used in the transmission of information between two or more stations are called a *data link*.
- The orderly transfer of information in a data link is accomplished by means of a *protocol*. A data link control protocol is a set of rules that are followed by interconnecting computers and terminals to ensure the orderly transfer of information.

Character-Oriented Protocol

- The character-oriented protocol is based on the binary code of a character set the code most commonly used is ASCII (American Standard Code for Information Interchange).
- It is a 7-bit code with an eighth bit used for parity. The (code has 128 characters, of which 95 are graphic characters and 33 are control characters.
- The graphic characters include the upper- and lowercase letters, the (ten numerals, and a variety of special symbols. A list of the ASCII character; can be found in Table 11-1.
- The control characters are used for the purpose o routing data, arranging the test in a desired format, and for the layout of the printed page.

The characters that control the transmission are called *communication control* characters.

These characters are listed in Table 11-4. Each character has a 7-bit code and is referred to by a three-letter symbol. The role of each character in the control of data transmission is stated briefly in the function column of the table.

The SYN character serves as synchronizing agent between the transmitter and receiver.

When the 7-bit ASCII code is used with an odd-parity bit in the most significant p

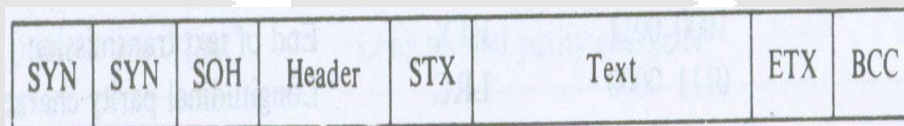
00010110	SYN	Synchronous idle	Establishes synchronism
00000001	SOH	Start of heading	Heading of block message
00000010	STX	Start of text	Precedes block of text
00000011	ETX	End of text	Terminates block of text
00001000	EOT	End of transmission	Concludes transmission
00001100	ACK	Acknowledge	Affirmative acknowledgement
00101010	NAK	Negative acknowledge	Negative acknowledgement
00001010	ENQ	Inquiry	Inquire if terminal is on
00101111	ETB	End of transmission block	End of block of data
00100000	DLE	Data link escape	Special control character

the assigned SYN character has the 8-bit cod 00010110, which has the property that, upon circular shifting, it repeats itself only after a full 8-bit cycle.

- When the transmitter starts sending 8-bit characters it sends a few characters first and then sends the actual message.
- If they do not match the bits of the SYN character, the receiver accepts the next bit, rejects the previous high-order bit, and again checks the last eight bits received for a SYN character
- . This is repeated after each clock pulse and bit received until a SYN character is recognized. Once a SYN character is detected, the receiver has framed a character. From here on the receiver counts every eight bits and accepts them as a single character.
- Usually, the receiver checks two consecutive SYN characters to remove any doubt that the first did not occur as a result of a noise signal on the line. Moreover, when the transmitter is idle and does not have any message characters to send, it sends a continuous string of SYN characters.
- The receiver recognizes these characters as a condition for synchronizing the line and goes into a synchronous idle state. In this state, the two units maintain bit and character synchronism even though no meaningful information is communicated.

Transmission Example

- The communication with the memory unit and CPU is similar to any I/O processor.
- Figure 11-25 Typical message format for character-oriented protocol



- A typical message that might be sent from a terminal to the processor is listed in Table 11-5. A look at this message reveals that there is a number of control characters used for message formation.
- Each character, including the control characters, is transmitted serially as an 8-bit binary code, which consists of the 7-bit ASCII code plus an odd parity bit in the eighth most significant position.
- The two SYN characters are used to synchronize the receiver and transmitter. The heading starts with the SOH character and continues with two characters that specify the address of the terminal.

Typical Transmission from a Terminal to Processor

- In this particular example, the address is T4, but in general it can have any set of two or more graphic characters. The STX character terminates the heading and signifies the beginning of the text transmission. The text data of concern here is "request balance of account number 1234."
- The individual characters for this message are not listed in the table because they will take too much space. It must be realized, however, that each character in the message has an 8-bit code and that each bit is transmitted serially.
- The ETX control character signifies the termination of the text characters. The next character following ETX is a longitudinal redundancy check (LRC).
- Each bit in this character is a parity bit calculated from all the bits in the same column in the code section of the table.

ATMIYA

Memory Organization

This chapter introduces the concept of memory hierarchy, composed of cache memory, main memory and auxiliary memory such as magnetic disk. The organization and operation of associative memories is explained in detail. The concept of memory management is introduced through the presentation of the hardware requirement for cache memory and a virtual memory system.

Main Memory

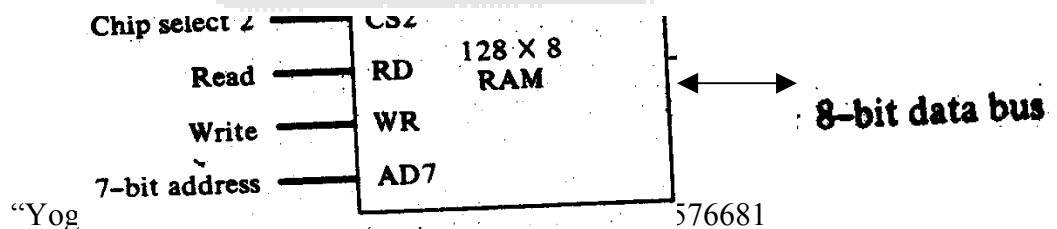
- The main memory' is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data during the computer operation.
- The principal technology used for the main memory is based on semiconductor integrated circuits. Integrated circuit RAM chips are available in two possible operating modes, *static* and *dynamic*.
- The static RAM consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit.
- The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors.
- The stored charge on the capacitors tends to discharge with time and refreshing the dynamic memory must periodically recharge the capacitors.
- Cycling through the words every few milliseconds to restore the decaying charge does refreshing.
- The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write cycles.
- Most of the main memory in a general-purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips. Originally, RAM was used to refer to a random-access memory, but now it is used to designate a read/write memory to distinguish it from a read-only memory, although ROM is also random access.
- RAM is used for storing the bulk of the programs and data that are subject to change.

- ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.
- Among other things, the ROM portion of main memory is needed for storing an initial program called a *bootstrap loader*. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on.
- Since RAM is volatile, its contents are destroyed when power is turned off.
- The contents of ROM remain unchanged after power is turned off and on again.
- The startup of a computer consists of turning the power on and starting the execution of an initial program.
- Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory' and control is then transferred to the operating system, which prepares the computer for general use.
- RAM and ROM chips are available in variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size.

RAM and ROM Chips(Bit –may01) (Bit-Mar02)

- A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed.
- Another common feature is a bi-directional data bus that allows the transfer of data either from memory to CPU during a read operation, or from CPU to memory during a write operation.
- A bi-directional bus can be constructed with three-state buffers.
- A three-state buffer output can be placed in one of three possible states: a signal equivalent to logic 1, a signal equivalent to logic 0, or a high-impedance state.
- The logic 1 and 0 are normal digital signals.
- The high-impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.

Diagram of RAM chip is shown below



a)Block Diagram

CS1	$\overline{CS2}$	RD	WR	Memory function	State of data bus
0	0	x	x	Inhibit	High-impedance
0	1	x	x	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	x	Read	Output data from RAM
1	1	x	x	Inhibit	High-impedance

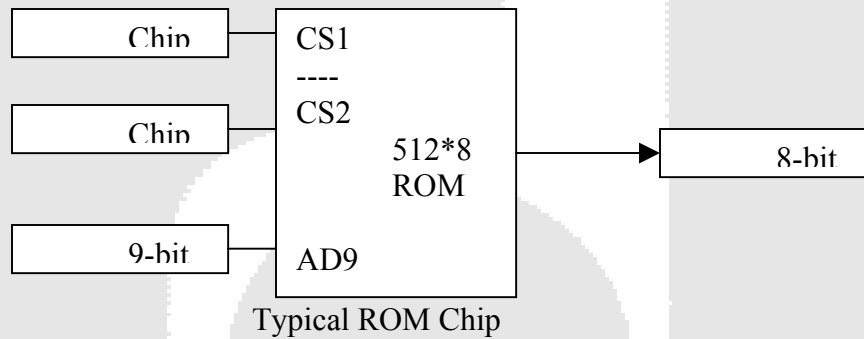
b)Function Table

- The memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bi-directional data bus.
- The read and write inputs specify the memory operation and the two chips select (CS) control inputs are for enabling the chip only when it is selected by the microprocessor.
- The availability of more than one control input to select the chip facilitates the decoding of the address lines when multiple chips are used in the microcomputer.
- The read and write inputs are sometimes combined into one line labeled R/W.
- When the chip is selected, the two binary states in this line specify the two operations of read or write.
- The function table listed in Fig. 12-2(b) specifies the operation of the RAM chip.
- The unit is in operation only when $CS1 = 1$ and $CS2 = 0$.
- The bar on top of the second select variable indicates that this input is enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state.
- When $CS1 = 1$ and $CS2 = 0$, the memory can be placed in a write or read mode. When the WR input is enabled, the memory stores a byte from the data bus into a location specified by the address input lines.
- When the RD input is enabled, the content of the selected byte is placed into the data bus. The RD and WR signals control the memory operation as well as the bus buffers associated with the bi-directional data bus.
- A ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode.
- The block diagram of a ROM chip is shown in Fig. 12-3. For the same-size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM. For this reason, the diagram specifies a 512-byte ROM, while the RAM has only 128 bytes.

- The nine address lines in the ROM chip specify any one of the 512 bytes stored in it. The two chip select inputs must be CS1 = 1 and CS2 = 0 for the unit to operate. Otherwise, the data bus is in a high-impedance state.
- There is no need for a read or write control because the unit can only read.
- Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

Memory Address Map

- The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM.
- The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available.



The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a *memory address map*, is a pictorial representation of assigned address space for each chip in the system. To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The memory address map for this configuration is shown in Table 12-1.

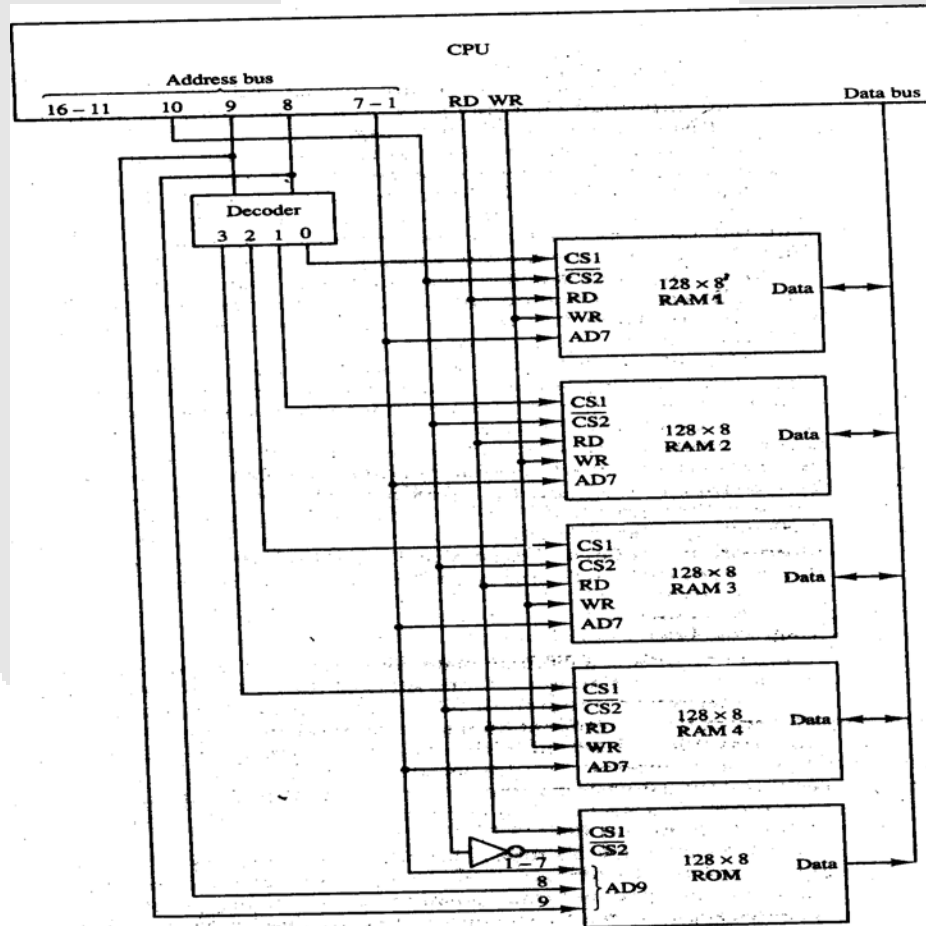
Component	Hexadecimal Address	Address Bus									
		10	9	8	7	6	5	4	3	2	1
RAM 1	0000-007F	0	0	0	x	x	x	x	x	x	x
RAM 2	0080-00FF	0	0	1	x	x	x	x	x	x	x
RAM 3	0100-017F	0	1	0	x	x	x	x	x	x	x
RAM 4	0180-01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200-03FF	1	x	x	x	x	x	x	x	x	x

- The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip.

- The address bus lines are listed in the third column. Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero.
- The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip.
- The RAM chips have 128 bytes and need seven address lines. The ROM chip has 512 bytes and needs 9 address lines.
- The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM.

Memory Connection to CPU

- RAM and ROM chips are connected to a CPU through the data and address buses.
- The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs.



- The connection of memory chips to the CPU is shown in Fig. 12-4. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM.

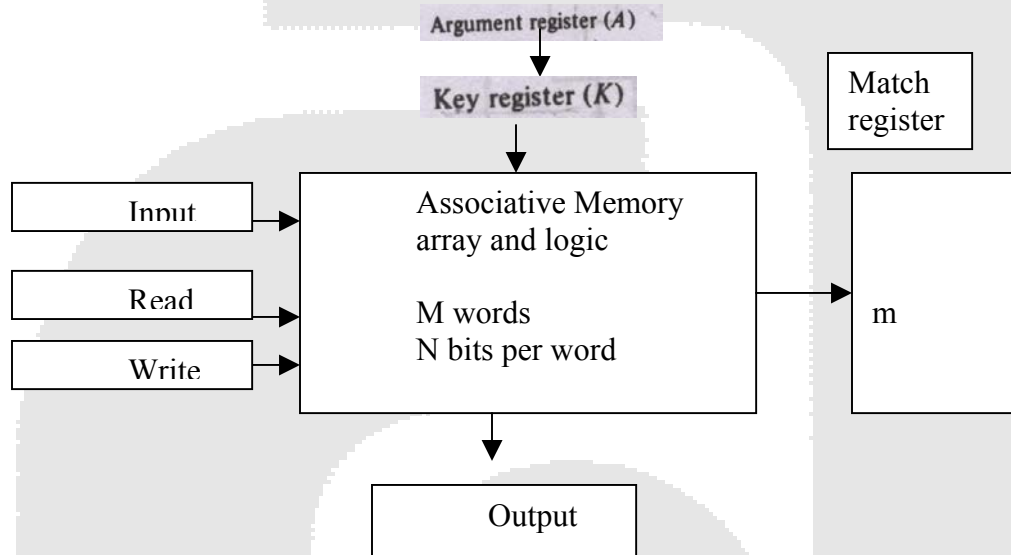
It implements the memory map of Table 12-1.

- Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes.
- The particular RAM chip selected is determined from lines 8 and 9 in the address bus.
- This is done through a 2x4 decoder whose outputs go to the CS1 inputs in each RAM chip.
- Thus, when address lines 8 and 9 are equal to 00, the first RAM chip is selected.
- When 01, the second RAM chip is selected, and so on. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip.
- The selection between RAM and ROM is achieved through bus line 10.
- The RAMs are selected when the bit in this line is 0, and the ROM when the bit is 1.
- The other chip select input in the ROM is connected to the RD control line for the ROM chip to be enabled only during a read operation.
- Address bus lines 1 to 9 are applied to the input address of ROM without going through the decoder.
- This assigns addresses 0 to 511 to RAM and 512 to 1023 to ROM. The data bus of the ROM has only an output capability, whereas the data bus connected to the RAMs can transfer information in both directions.
- The example just shown gives an indication of the interconnection complexity that can exist between memory chips and the CPU.
- The more chips that are connected, the more external decoders are required for selection among the chips.
- The designer must establish a memory map that assigns addresses to the various chips from which the required connections are determined.
- The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks.
- To understand fully the physical mechanism of auxiliary memory devices one must have knowledge of magnetic, electronics, and electromechanical systems.

Associative Memory

- Many-data-processing applications require the search of items in a table stored in memory. An assembler program searches the symbol address table in order to extract the symbol's binary equivalent.
- An account number may be searched in a file to determine the holder's name and account status.

- The established way to search a table is to store all items where they can be addressed in sequence.



- The search procedure is a strategy for choosing a sequence of addresses, reading the content of memory at each address, and comparing the information read with the item being searched until a match occurs.
- The number of accesses to memory depends on the location of the item and the efficiency of the each algorithm. Many search algorithms have been developed to minimize the number of accesses while searching for an item in a random or sequential access memory.
- The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
- A memory unit accessed by content is called *content addressable unjassociatiix memory* or *content addressable memory (CAM)*. When a word is written in an associative memory, no address is given.
- The memory is capable of finding an empty unused location to store the word.
- When a word is to be read from an associative memory, the content of the word, or part of the word, is specified. The memory locates all words, which match the specified content, and marks them for reading.

Hardware Organization

The block diagram of an associative memory is shown 0in Fig. 12-6. It consists of a memory array and logic for m words with n bits per word.

The argument register A and key register K each have n bits, one for each bit of a word.

The match register M has m bits, one for each memory word. Each word in memory is compared in parallel with the content of the argument register.

The words that match the bits of the argument register set a corresponding bit in the match register. After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched. Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.

The key register provides a mask for choosing a particular field or key in the argument word. The entire argument is compared with each memory word if the key register contains all 1's.

Cache Memory

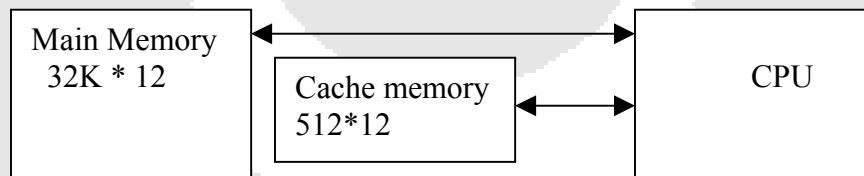
(Bit –may01) (Bit-Mar02)

If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a *cache memory*. It is placed between the CPU and main memory as illustrated in Fig given below. The cache memory access time is less than the access time of main memory by a factor of 5 to 10.

The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.

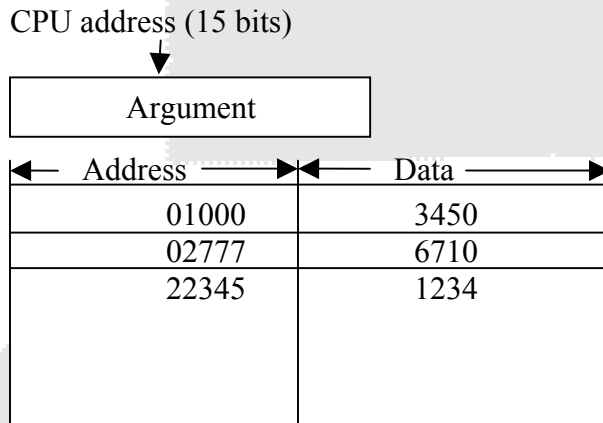
The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory. The average memory access time will approach the access time of the cache.

Figure



Associative Mapping

The fastest and most flexible cache organization uses an associative this organization is illustrated in Fig. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory.

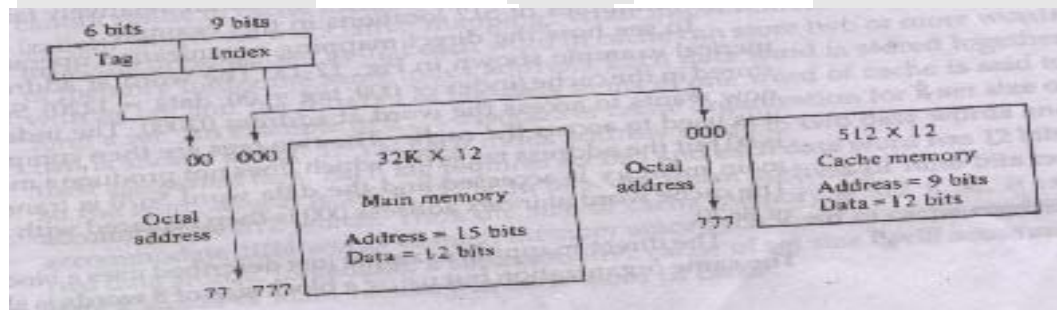


The diagram shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number.

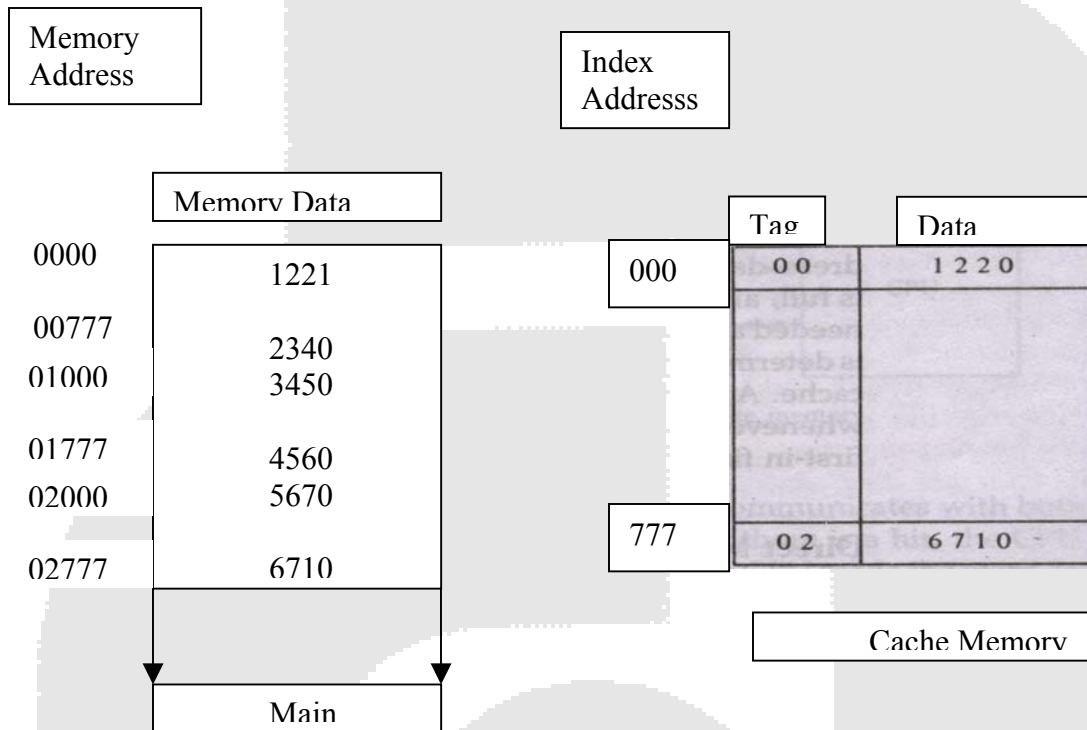
A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address.

Direct Mapping

Associative memories are expensive compared to random-access memory because of the added logic associated with each cell. The possibility of using a random-access memory for the cache is investigated in Fig.



The CPU address of 15 bits is divided into two fields. The nine least significant bits constitute the *index* field and the remaining six bits form the *tag* field. The figure shows that main memory needs an address that includes both the tag and the index bits.



- The number of bits in the index field is equal to the number of address bits required to access the cache memory.

Virtual Memory ❖

- *Virtual memory* is a concept used in some large computer systems that permits the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory.
- Each address that is referenced by the CPU goes through an address mapping from the so-called virtual address- to a physical address in main memory.
- Virtual memory is used give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory.
- A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations.
- This is done dynamically, while programs are being executed in the *CPU*.
- The translation or mapping is handled automatically by the hardware by means of a mapping table.

Address Space and Memory Space

- An address used by a programmer will be called *virtual addresses*, and the set of such addresses the *address space*.
- An address in main memory is called a *memory space location or physical address*.

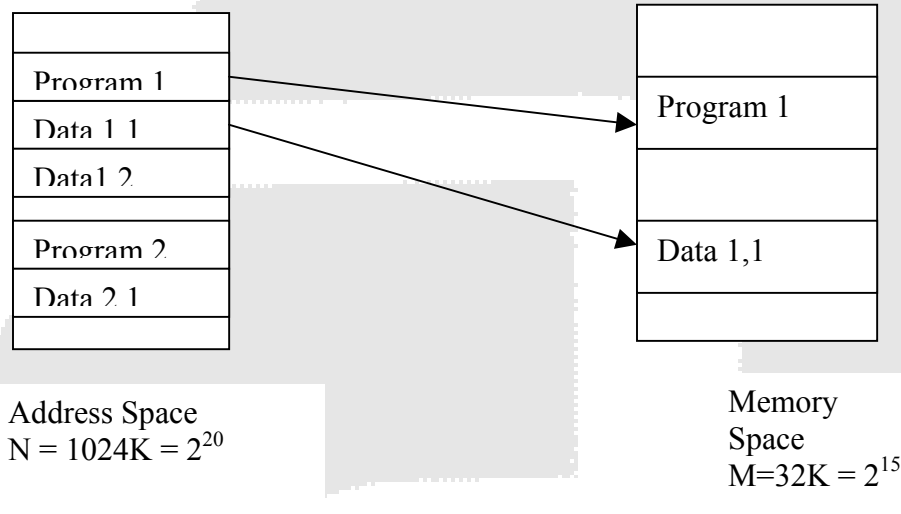


Figure Relation between address and memory space in a virtual memory system

- The set of such locations is called the *memory space*.
- Thus the address space is the set of addresses generated by programs as the reference instructions and data; the memory space consists of the actual main memory locations directly addressable for processing.

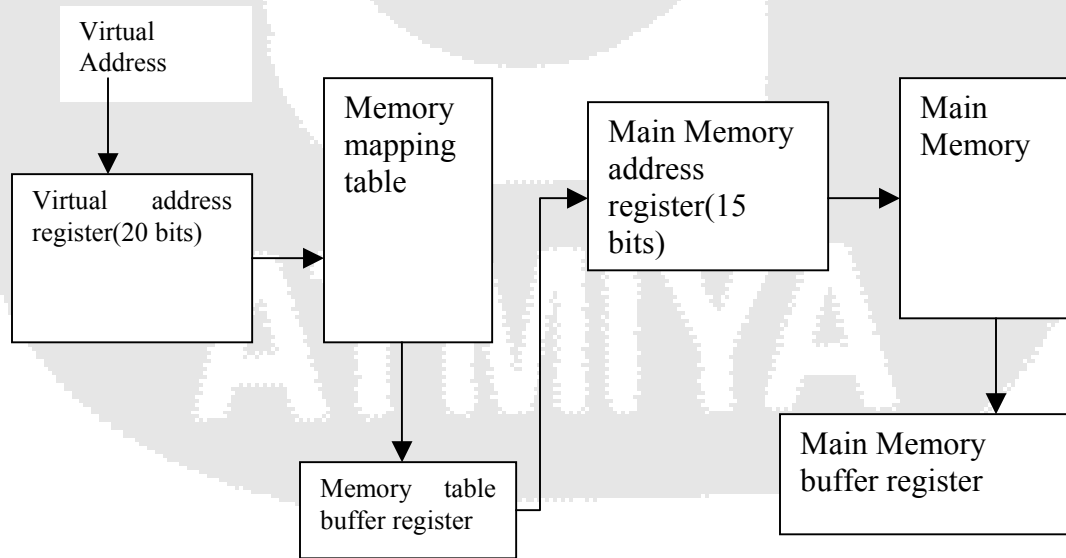
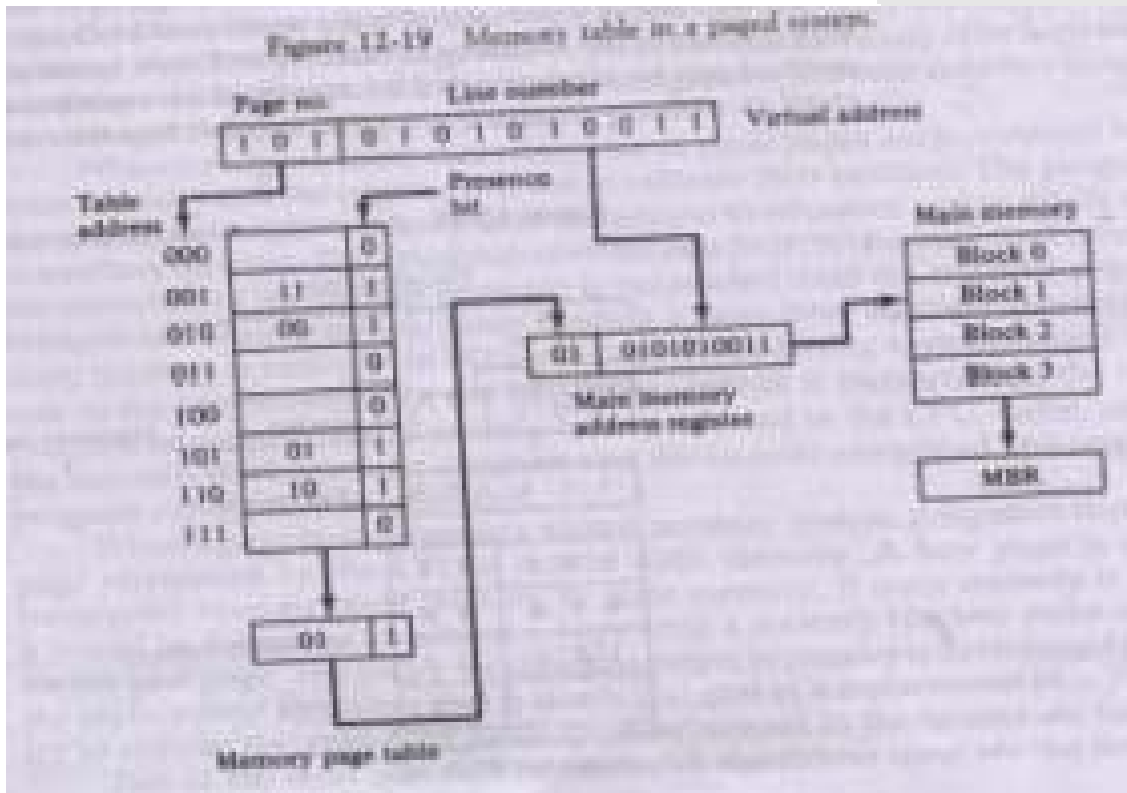


Fig 12.17 memory table for mapping a virtual address

In most computers the address and memory' spaces are identical. The address space is allowed to be larger than the memory space in computers with virtual memory.

Associative Memory Page Table

- A random-access memory page table is inefficient with respect to storage utilization.
- In the example of Fig. We observe that eight words of memory are needed; one for each page, but at least four words will always be marked empty because main memory cannot accommodate more than four blocks.



- In general, a system with n pages and m blocks would require a memory-page table of n locations of which up to m blocks will be marked with block numbers and all others will be empty.
- As a second numerical example, consider an address space of 1024K words and memory space of 32K words. If each page or block contains 1K words, the number of pages is 1024 and the number of blocks 32.
- The capacity of the memory-page table must be 1024 words and only 32 locations may have a presence bit equal to 1. At any given time, at least 992 locations will be empty and not in use.

Page Replacement

- A virtual memory system is a combination of hardware and software techniques.

- The memory management software system handles all the software operations for the efficient utilization of memory space. It must decide
- Which page in main memory ought to be removed to make room for a new page
- When a new page is to be transferred from auxiliary memory to main memory
- Where the page is to be placed in. main memory.
- The hardware mapping mechanism and the memory management software together constitute the architecture of a virtual memory.
- When a program starts execution, one or more pages are transferred into main memory and the page table is set to indicate their position.
- The program is executed from main memory until it attempts to reference a page that is still in auxiliary memory. This condition is called *page fault*.
- When page fault occurs the execution of the present program is suspended until the required page is brought into main memory.
- Since loading a page from the auxiliary memory to main memory is basically an I/O operation, the operating system assigns this task to the I/O processor.
- In the meantime, control is transferred to the next program in memory that is waiting to be processed in the CPU. Later, when the memory block has been assigned and the transfer completed, the original program can resume its operation.
- When a page fault occurs in a virtual memory system, it signifies that the page referenced by the CPU is not in main memory.
- A new page is then transferred from auxiliary memory to main memory. If main memory is full, it would be necessary to remove a page from a memory block to make room for the new page.
- The policy for choosing pages to remove is determined from the replacement algorithm that is used.
- The goal of a replacement policy is to try to remove the page least likely to be referenced in the immediate future.
- Two of the most common replacement algorithms used are *the first-in first out(FIFO)* and the *least recently used(LRU)*.

ATMIYA

Bibliography

Computer System Architecture

M. Morris Mano

<http://www.ece.uvic.ca/>

<http://www.cs.rit.edu/>

<http://www.ece.northwestern.edu/>

<http://www.icg.harvard.edu/>

<http://www.cs.jcu.edu.au/>

<http://www.csif.cs.ucdavis.edu/>

<http://www.williamstallings.com/>

<http://www.pcs.cnu.edu/>

<http://www.engineering.dartmouth.edu/>

<http://www.cs.berkeley.edu/~pattrsn>

<http://www.cs.rit.edu/~wrc/documents/mano-mcu/>

ATMIYA

BCA-Question Papers

Bca-Apr/May2001

1. (a) Explain the term 'Logic gates'. Explain following gates : 6
 - (1) AND
 - (2) OR
 (b) Explain the functional block diagram of computer. 6
OR
 (b) Explain the historical perspective of computer. 6
 (C) Determine by means of truth table the validity of associate law. 4
2. (a) Simplify the following boolean expression using laws of boolean algebra : (any two) 4
 - (1) $[(CD)' + A]' + A + CD + AB$
 - (2) $ABC A'B + ABC'$
 - (3) $(BC' + A'D) (AB' + CD')$
 (b) Draw the logic diagram and give the truth table for the following boolean expression : 4
 $F + xy + xy' + y'z$
- (c) NAND and NOR gates are known as universal gates - why? 3
 (d) Simplify the following Boolean function using K'map : (any one) 6
 - (1) $F(w,x,y,z) = E (2,3,12,13,14,15)$
 $d(w,x,y,z) = E (0,1,4)$
 - (2) $F(w,x,y,z) = E (1,3,7,11,15)$
 $d(w,x,y,z) = E (0,2,5)$
3. (a) Design and develop the circuit of Decoder. 6
OR
 (a) Design and develop the circuit of Encoder 6
 (b) Construct a full adder using two half adders 4
 (c) Explain JK Flip Flop 6
OR
 (c) Explain SR Flip Flop
4. (a) Convert the following expression into reverse polish notation : (any two) 4
 - (1) $A*B/C + (E+F+G) - H*I$
 - (2) $A*[B+(C*D-E)-F]/G$
 - (3) $A*(B=G(D+E)) * F*(G+H)$
 (b) Explain the stack organization in detail. 6
 (c) Explain the following addressing modes: 4
 - (1) Register Indirect Mode
 - (2) Autoincrement Mode.
 (d) Write two address instruction for evaluating following expression : 4
 $X = A-B+C*D+E$
OR
 (d) Write one address instruction for evaluating following expression. 4

$$X = A - B + C * D + E$$

- (e) Define the following terms : (any two) 2
- (1) Control memory
 - (2) Micro program
 - (3) Hit ratio.
5. (a) Explain in detail DMA operation.
- (b) A computer having 16 address lines and 8 data lines 4
employs RAM chips of 540X8 and ROM chips of 1024X8.
The computer system needs three such RAM chips and two
such ROM chips. Each chip is having two chips select line
namely CS1 and CS2. Draw memory address map this
configuration and show the connection of memory chips to CPU.
- (c) Distinguish : Isolate I/O Vs. Memory mapped I/O. 4
OR
- (c) Explain various I/O commands. 4
6. (a) Sketch and explain the construction of magnetic tape 6
OR
Magnetic disk
- (b) Explain in details the I/O Interface unit. 8
- (c) Explain programmed I/O mode of data transfer between 3
I/O and processor.
OR
- (c) Explain interrupt-initiated I/O mode of data transfer 3
between I/O and processor.

ATMIYA

BCA-AprMay2000

1. (a) Using block diagram, Explain bus structure of computer system. 8
(b) Design and develop full adder circuit. 8
OR
(b) Sketch and explain construction of magnetic drum/disk. 8
2. (a) Draw circuit diagram of 3 to 8 decoder and write truth table. 8
(b) What is universal gate ? Construction AND, OR gate using NAND gates. 8
OR
(b) Distinguish between combination and sequential circuit using example.
(c) Write truth table and derive Boolean equation for EX-OR gate. 4
3. (a) Explain types of memory in computer system. 8
(b) What is cache memory ? Explain it in detail
Compare speed, size and cost/bit for main memory, auxiliary memory and cash memory.
OR
(b) Explain direct and associate mapping for cache memory.
4. (a) State whether following are true or false :
(1) In case of boolean algebra $A+A+A = 3A$.
(2) EX-OR gate is complement of OR gate.
(3) Two's complement of 1000 is 1000 in binary.
(4) In memory mapped I/O separate I/O instructions are used.
(5) Synchronous data transfer is faster than Asynchronous.
(6) Cycle stealing principle can't be used for DMA data transfer.
(b) Explain various data transfer schemes in detail. 8
(c) What is micro operation? Explain in brief. 2
5. (a) Using flow chart, Explain CPU IOP communication. 8
(b) Using block diagram, explain microprocessor and I/O Interface unit.
OR
5. (a) Describe various types of CPU organizations.
(b) Explain memory mapped I/O and I/O mapped I/O.
6. (a) List various types of instructions formats and explain any one of them with example. 8
(b) List various types of addressing modes and explain any two of them. 8
OR
(b) What is stack? Explain PUSH and POP operation.
(c) Convert following in reverse polish notation : 4
(1) $(A+B) * [C * (D+E) + F]$

(2) $A*B + C*D + E*F$.



BIT Question Papers
Bit-MarApr2002

Q-1

A State True/False [05]

1. CPU is a part of ALU.
2. Decoder and demultiplexers are same.
3. Nor gate is universal gate
4. ROM and RAM are permanent storage.

B Answer in short : [15]

1. How many buses are there on motherboard?
2. What is Truth table?
3. Calculate the 2's complement of 11000110
4. What is Compiler? Differentiate it from interpreter.
5. What is flow Chart?

Q-2

Attempt any four : [20]

- a. What is Encoder? What is the difference between encoder & Decoder? Explain Decoder in detail.
- b. What is microprocessor?
- c. Explain parallel data Transmission. Differentiate it From serial data transmission.
- d. Explain EX-OR, NAND and NOR gate with symbols.
- e. What is Asynchronous Data Transmission?

Q-3 Attempt any five: [20]

- A. Draw the block diagram of digital Computer & explain it.
- b. What is Stack? Explain in brief.
- c. What is RISC? - Discuss it.
- D. How many types of address modes are there? Discuss them.
- e. What is fetch Cycle? Discuss Error Detection Codes.
- f. What is memory? Explain cache memory.

Q-4

a. What is Flip Flop? Explain J-K Flip Flop with advantages [08] & Disadvantages. Also discuss the don't care conditions.

b. Simplify following Boolean Expressions: [08]

1. $F(x,y,z) = \sum (0,2,3,4,5)$
2. $F(x,y,z) = \sum (3,5,6,7)$

c. Attempt any one : [04]

1. What is Full Adder? Explain it.
2. What is the Hard Disk? How it is created?

Q-5 Attempt any two : [20]

1. Explain DMA.
2. What is instruction? Explain different instruction formats.
3. Explain Binary Counters.

BIT-MarApr2001

Q-1 (a) Explain different functional unit of a computer with block diagram. [15]

- (b) Discuss difference bus structures?
- (c) Discuss first and second generation of computer with example.

Q-2 (any four) [20]

- (a) What is gate? Discuss different logic gates available in Digital logic circuit.
- (b) Prove the following expression by perfect induction method
 $XY + YZ + YZ = XY + Z$
- (c) Simplify the following expression.
 $ABC + ABC + ABC + ABC + ABC + ABC + ABC$
- (D) What is flip flop? Explain S-R flip flop.
- (e) Write short notes on : (any two)
 - (1) Decoder
 - (2) Universal gate
 - (3) Sequential circuit.
- (f) Draw the gate implementation of the following Boolean Expression:
 - (1) $AB + AB + AC + AC$
 - (2) $ABD + ACD + ABC$

Q-3 (a) Explain magnetic disk or magnetic drum [15]

- (b) Explain the concept of cache memory? What is hit ratio.

OR

- (b) What are different types of main memory ? Discuss
- (c) Explain : (any two) :
 - (1) Access time
 - (2) Alterability
 - (3) Cycle time

Q-4 (any three) [15]

- (a) Explain briefly direct memory access operation.
- (b) What is serial and parallel interface.
- (c) What is input and output processor?
What are the function of IOP?
- (d) What is input output interface? What are function of input output interface

Q-5 (any two) [15]

- (a) Explain different addressing modes with an example.
- (b) Explain different address instruction, Give examples.
- (c) Write short notes on : (any three)
 - (1) Micro operation
 - (2) Micro instruction
 - (3) PUSH operation
 - (4) Micro. program.

Q-6 (a) _____ was the first general purpose electronics computer.

- (b) The section of the CPU that selects, interprets and sees the execution of the program instruction is _____
- (c) Light pen is an _____ device.

- (d) A bus that connects major computer components is called _____
- (e) _____ command is used to test various status condition of the interfave and the peripheral.
- (f) A sequence of micro instruction constitute a _____
- (g) Write the following expression in reverse polish notation.
 $A * B + C * D$
- (h) In boolean algebra $X + X.Y = X + Y$ (TRUE/FALSE)
- (i) During the DMA transfer the CPU is ideal and has no control of the memory bus. (TRUE/FALSE)
- (J) Which is secondary memory device :
- (1) CPU
 - (2) ALU
 - (3) Read only memory
 - (4) Mouse
 - (5) None of the above.



ATMIYA

MCA-Question Paper
M.C.A. November – 1999

Computer Organization & Assembly Language

SECTION – I

Q1. Answer any two of the following:

Explain contemporary multilevel machines in detail.

Explain milestones in Computer Architecture.

Explain Printer, Block diagram? What are the characteristics of a printer? And list out the different types of printers.

Q2. Answer any four of the following:

What is a Multiplexer? Explain the functioning of a multiplexer circuit.

Explain a general chip used for forming sums of products.

What is the SR latch's ambiguity? How can you resolve that ambiguity?

Draw the circuit diagram for a full adder and explain its operation in brief with a truth table.

What is a Comparator? Taking suitable example explain its operation in brief with the circuit diagram.

Q3. Answer any two of the following:

What is Hamming distance? Taking suitable example, explain how hamming algorithm can be used to construct single but error correcting code.

Explain the working operation of CRT-display monitor. Also describe one technique for generating image on the screen.

Write a note on 'Parallel Instruction Execution'.

SECTION II

Q4. Answer any two of the following:

Explain instruction formats. What are the design criteria for instruction formats?

Explain trade off between short instructions and long instructions with appropriate example.

Explain different types of addressing techniques with suitable examples.

What are RISC machines? Explain design principles for RISC machines.

Q5.

(a) Answer any two of the following:

Asynchronous Buses

Microprocessor Chips

Bus Arbitration

(b) Complete the following blanks:

NOT gates are often called _____.

A flip-flop is _____ triggered and a latch is _____ triggered.

EEPROM means _____.

_____ and _____ signals are used to cause CPU interrupts in the Intel 8088 microprocessor chip.

PEACK is not present in the 80386 chip because _____.

(c) Answer any two of the following:
Explain architecture of pipeline machine.
What are the main functions of a device controller?
Synchronous and asynchronous transmission.

Q6. Answer any two of the following:
Explain CPU registers in detail.
What is the task of a loader? Explain dynamic relocation in detail.
Write assembly language code to do the following:
Set the cursor at row 11 and column 40
Clearing the screen
Display a string on screen.



ATMIYA