

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

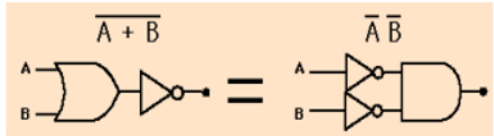
Module-1

Chapter 1: Boolean Algebra and Logic Gates

1.1 Basic Theorems and Properties of Boolean Algebra

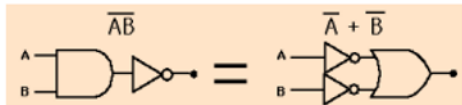
De Morgan's First Theorem: The complement of a sum equals the product of the complements.

A	B	A+B	$\overline{A+B}$	$\bar{A}$	$\bar{B}$	$\bar{A} \cdot \bar{B}$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0



De Morgan's Second Theorem: The complement of a product equals the sum of the complements.

A	B	AB	$\overline{AB}$	$\bar{A}$	$\bar{B}$	$\bar{A} + \bar{B}$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0



NAND Gate

Bubbled OR Gate

Duality Theorem: Starting with a Boolean relation, can derive another Boolean relation by

1. Changing each OR sign to an AND sign
2. Changing each AND sign to an OR sign
3. Complementing any 0 or 1 appearing in the expression.

Example: 1. We say that, $A+0 = A$; the dual is, $A \cdot 1 = A$

2. Consider, $A(B+C) = AB + AC$

By changing the OR and AND operation, we get the dual relation:

$$A + BC = (A+B)(A+C)$$

Laws of Boolean Algebra

The following laws are of immense use in the simplification of Boolean expressions. Note that, if A is a variable, then either $A = 0$ or $A = 1$. Also, when $A = 0$, $A \neq 1$; and when $A = 1$, $A \neq 0$.

- 1) Commutative Law: $A + B = B + A$ and $A \cdot B = B \cdot A$
- 2) Associative Law: $A + (B + C) = (A + B) + C$ and $A \cdot (BC) = (AB) \cdot C$
- 3) Distributive Law: $A(B + C) = AB + AC$
- 4) In relation to OR operation, the following laws hold good:

$A + 0 = A$
 $A + A = A$
 $A + 1 = 1$ and $A + A' = 1$

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

5) In relation to AND operation, the following laws hold good: $A \cdot 1 = A$

$$A \cdot A = A$$

$$A \cdot 0 = 0$$

$$A \cdot A' = 0 \text{ and } A'' = A$$

6) Some more useful Boolean relations: $A + AB = A$

$$A + A'B = A + B$$

$$A(A + B) = A$$

$$A(A' + B) = AB$$

$$A + (B \cdot C) = (A + B)(A + C)$$

1.2 Boolean Functions

Boolean algebra is an algebra that deals with binary variables and logic operations. A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols. For a given value of the binary variables, the function can be equal to either 1 or 0.

As an example, consider the Boolean function $F1 = x + y'z$.

The function $F1$ is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. $F1$ is equal to 0 otherwise. The complement operation dictates that when $y' = 1$, $y = 0$. Therefore, $F1 = 1$ if $x = 1$ or if $y = 0$ and $z = 1$. A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

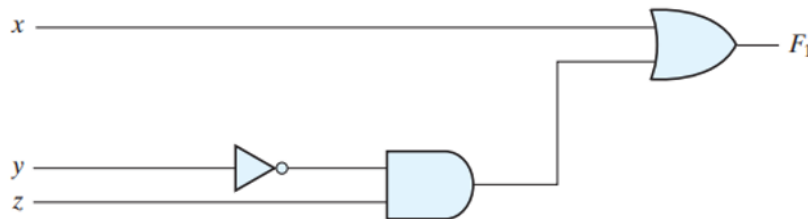
A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n , where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$. Table below shows the truth table for the function $F1$. There are eight possible binary combinations for assigning bits to the three variables x , y , and z . The column labelled $F1$ contains either 0 or 1 for each of these combinations. The table shows that the function is equal to 1 when $x = 1$ or when $yz = 01$ and is equal to 0 otherwise.

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

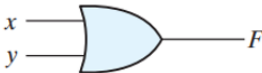
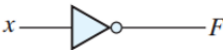
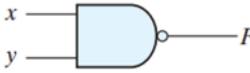
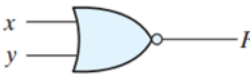
A Boolean function can be transformed from an algebraic expression into a circuit diagram composed of logic gates connected in a particular structure. The logic-circuit diagram (also called a schematic) for $F1$ is shown in Fig. There is an inverter for input y to generate its

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

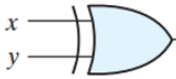
complement. There is an AND gate for the term $y'z$ and an OR gate that combines x with $y'z$. In logic-circuit diagrams, the variables of the function are taken as the inputs of the circuit and the binary variable F_1 is taken as the output of the circuit. The schematic expresses the relationship between the output of the circuit and its inputs.



1.3 Digital Logic Gates

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																

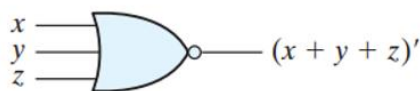
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

			x	y	F
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	0	0	0
			0	1	1
			1	0	1
			1	1	0

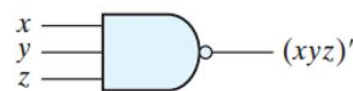
			x	y	F
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	0	0	1
			0	1	0
			1	0	0
			1	1	1

Extension to Multiple Inputs

The graphic symbols for the three-input gates are shown in Fig. below. In writing cascaded NOR and NAND operations, one must use the correct parentheses to signify the proper sequence of the gates.

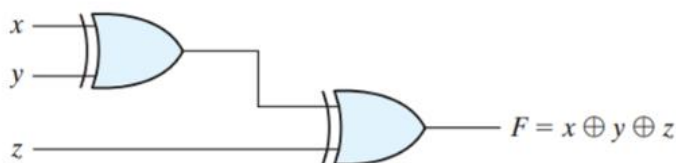


(a) 3-input NOR gate

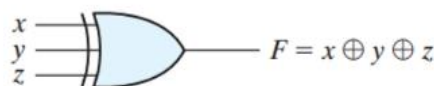


(b) 3-input NAND gate

The construction of a three-input exclusive-OR function is shown in Fig. below. This function is normally implemented by cascading two-input gates, as shown in (a). Graphically, it can be represented with a single three-input gate, as shown in (b). The truth table in (c) clearly indicates that the output F is equal to 1 if only one input is equal to 1 or if all three inputs are equal to 1 (i.e., when the total number of 1's in the input variables is odd).



(a) Using 2-input gates



(b) 3-input gate

x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

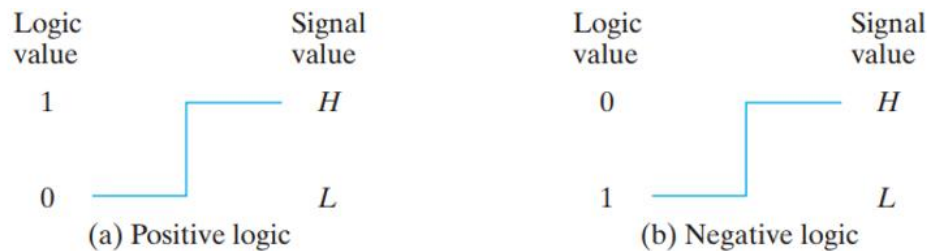
(c) Truth table

Positive and Negative Logic

The binary signal at the inputs and outputs of any gate has one of two values, except during transition. One signal value represents logic 1 and the other logic 0. Since two signal values are assigned to two logic values, there exist two different assignments of signal level to logic value, as shown in Fig. below. The higher signal level is designated by H and the lower signal

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

level by L. Choosing the high-level H to represent logic 1 defines a positive logic system. Choosing the low-level L to represent logic 1 defines a negative logic system. The terms positive and negative are somewhat misleading since both signals may be positive or both may be negative. It is not the actual values of the signals that determine the type of logic, but rather the assignment of logic values to the relative amplitudes of the two signal levels.



1.4 Integrated Circuits

An integrated circuit (IC) is fabricated on a die of a silicon semiconductor crystal, called a chip, containing the electronic components for constructing digital gates. The various gates are interconnected inside the chip to form the required circuit. The chip is mounted in a ceramic or plastic container, and connections are welded to external pins to form the integrated circuit. The number of pins may range from 14 on a small IC package to several thousand on a larger package. Each IC has a numeric designation printed on the surface of the package for identification. Vendors provide data books, catalogs, and Internet websites that contain descriptions and information about the ICs that they manufacture.

Levels of Integration

Digital ICs are often categorized according to the complexity of their circuits, as measured by the number of logic gates in a single package. The differentiation between those chips which have a few internal gates and those having hundreds of thousands of gates is made by customary reference to a package as being either a small-, medium-, large-, or very large-scale integration device.

Small-scale integration (SSI) devices contain several independent gates in a single package. The inputs and outputs of the gates are connected directly to the pins in the package. The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.

Medium-scale integration (MSI) devices have a complexity of approximately 10 to 1,000 gates in a single package. They usually perform specific elementary digital operations. Example: decoders, adders, multiplexers, registers and counters.

Large-scale integration (LSI) devices contain thousands of gates in a single package. They include digital systems such as processors, memory chips, and programmable logic devices.

Very large-scale integration (VLSI) devices now contain millions of gates within a single package. Examples are large memory arrays and complex microcomputer chips. Because of their small size and low cost, VLSI devices have revolutionized the computer system design

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

technology, giving the designer the capability to create structures that were previously uneconomical to build.

Digital Logic Families

Digital integrated circuits are classified not only by their complexity or logical operation, but also by the specific circuit technology to which they belong. The circuit technology is referred to as a digital logic family. Each logic family has its own basic electronic circuit upon which more complex digital circuits and components are developed. The basic circuit in each technology is a NAND, NOR, or inverter gate. The electronic components employed in the construction of the basic circuit are usually used to name the technology. Many different logic families of digital integrated circuits have been introduced commercially. The following are the most popular:

- TTL transistor-transistor logic;
- ECL emitter-coupled logic;
- MOS metal-oxide semiconductor;
- CMOS complementary metal-oxide semiconductor.

TTL is a logic family that has been in use for 50 years and is considered to be standard. ECL has an advantage in systems requiring high-speed operation. MOS is suitable for circuits that need high component density, and CMOS is preferable in systems requiring low power consumption, such as digital cameras, personal media players, and other handheld portable devices. Low power consumption is essential for VLSI design; therefore, CMOS has become the dominant logic family, while TTL and ECL continue to decline in use. The most important parameters distinguishing logic families are listed below; CMOS integrated circuits are discussed briefly in the appendix.

Fan-out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation. A standard load is usually defined as the amount of current needed by an input of another similar gate in the same family.

Fan-in is the number of inputs available in a gate.

Power dissipation is the power consumed by the gate that must be available from the power supply.

Propagation delay is the average transition delay time for a signal to propagate from input to output. For example, if the input of an inverter switches from 0 to 1, the output will switch from 1 to 0, but after a time determined by the propagation delay of the device. The operating speed is inversely proportional to the propagation delay.

Noise margin is the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Chapter 2: Gate-Level Minimization

1.5 Introduction

Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit. This task is well understood but is difficult to execute by manual methods when the logic has more than a few inputs. Fortunately, computer-based logic synthesis tools can minimize a large set of Boolean equations efficiently and quickly. Nevertheless, it is important that a designer understand the underlying mathematical description and solution of the problem.

Minimum Forms of Switching Functions

A minimum sum-of-products expression

A minimum sum-of-products expression for a function is defined as a sum of product terms which (a) has a minimum number of terms and (b) of all those expressions which have the same minimum number of terms, has a minimum number of literals.

The minimum sum of products corresponds directly to a minimum two-level gate circuit which has (a) a minimum number of gates and (b) a minimum number of gate inputs.

Given a minterm expansion, the minimum sum-of-products form can often be obtained by the following procedure:

1. Combine terms by using the uniting theorem $XY' + XY = X$. Do this repeatedly to eliminate as many literals as possible. A given term may be used more than once because $X + X = X$.
2. Eliminate redundant terms by using the consensus theorem or other theorems. Unfortunately, the result of this procedure may depend on the order in which terms are combined or eliminated so that the final expression obtained is not necessarily minimum.

A minimum products-of-sum expression

A minimum product-of-sums expression for a function is defined as a product of sum terms which (a) has a minimum number of terms, and (b) of all those expressions which have the same number of terms, has a minimum number of literals.

Unlike the maxterm expansion, the minimum product-of-sums form of a function is not necessarily unique. Given a maxterm expansion, the minimum product of sums can often be obtained by a procedure similar to that used in the minimum sum of-products case, except that the uniting theorem $(X + Y)(X + Y') = X$ is used to combine terms.

1.6 The Map Method

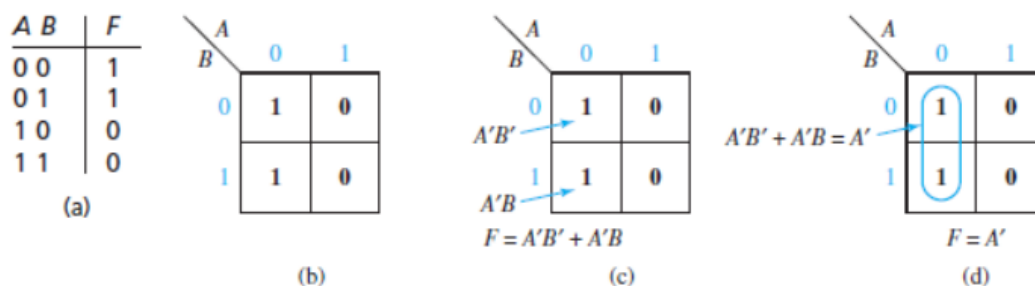
Karnaugh Map-A drawing that shows all the fundamental products and the corresponding output values of a truth table.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Just like a truth table, the Karnaugh map of a function specifies the value of the function for every combination of values of the independent variables.

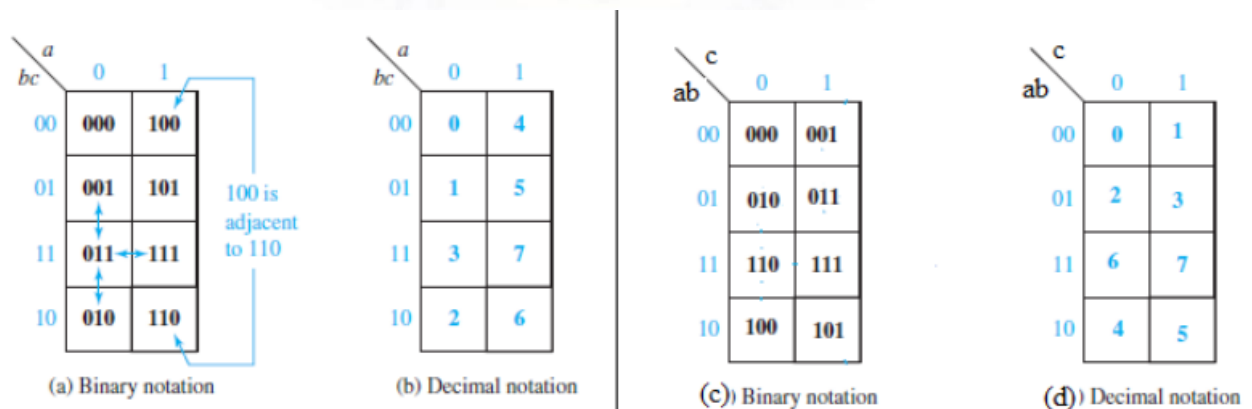
Two variable K-Map

The values of one variable are listed across the top of the map, and the values of the other variable are listed on the left side. Each square of the map corresponds to a pair value for A and B as indicated. The following Figure shows the truth table for a function F and the corresponding Karnaugh map:



Three Variable K-Map

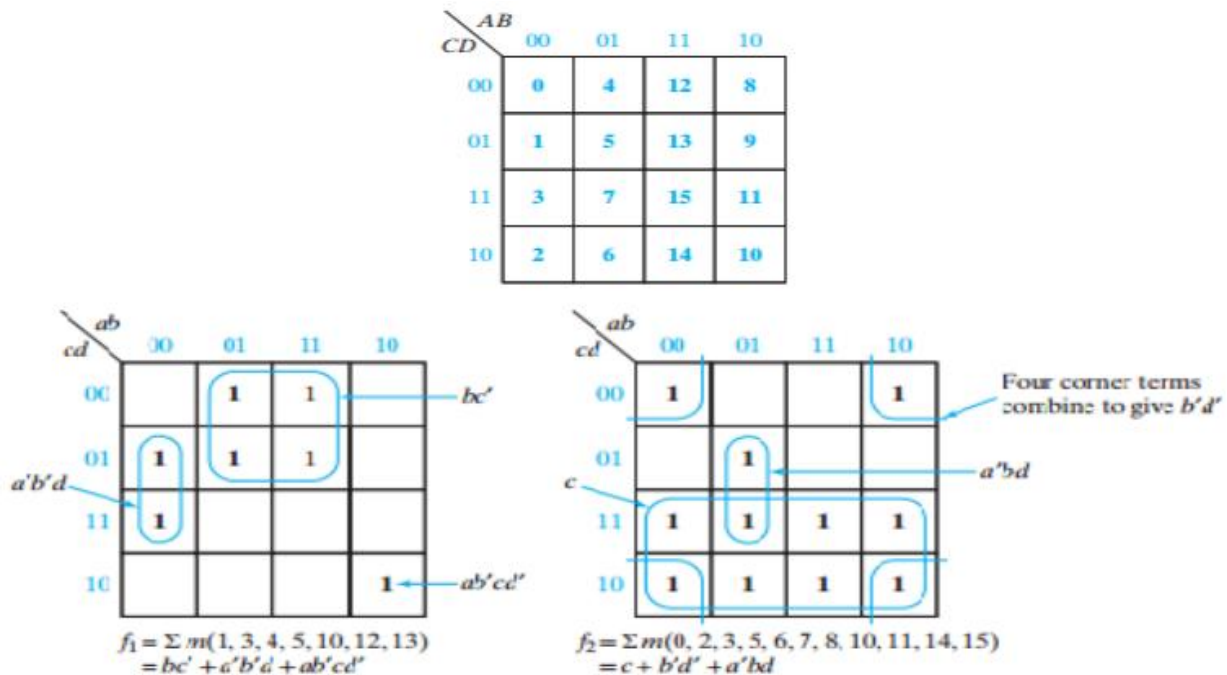
The following Figure shows a three-variable truth table and the corresponding Karnaugh map: The value of one variable (A) is listed across the top of the map, and the values of the other two variables (B, C) are listed along the side of the map. The rows are labelled in the sequence 00, 01, 11, 10 so that values in adjacent rows differ in only one variable.



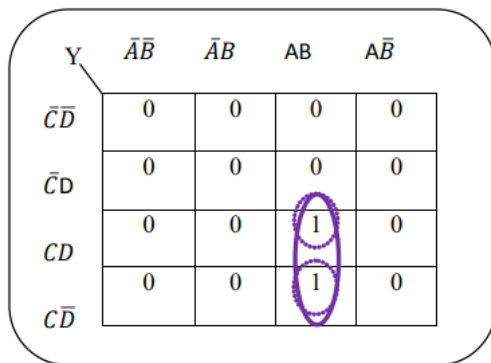
1.7 Four-Variable K-Map

A four -variable Karnaugh map is shown below Each minterm is located adjacent to the four terms with which it can combine.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



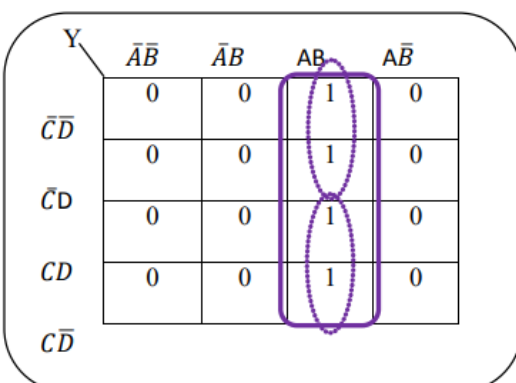
Pairs: The following K-map contains a pair of 1s that are horizontally adjacent. Two adjacent 1s, such as these are called a pair. A pair eliminates one variable and its complement.



The sum-of-product equation is:

$$Y = ABCD + ABCD' = ABC(D + D') = ABC$$

Quad: A quad is a group of four 1s that are horizontally or vertically adjacent. A quad eliminates two variables and their complements.



The sum-of-product equation is:

$$Y = ABC' + ABC = AB(C + C') = AB$$

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The Octet: The octet is a group of eight 1s, as shown in the following Fig. An octet eliminates three variables and their complements.

Y	$\bar{A}\bar{B}$	$\bar{A}B$	AB	$A\bar{B}$
$\bar{C}\bar{D}$	0	0	1	1
$\bar{C}D$	0	0	1	1
CD	0	0	1	1
$C\bar{D}$	0	0	1	1

The sum-of-product equation is:

$$Y = AB + AB' = A(B + B') = A$$

Prime Implicants

In choosing adjacent squares in a map, we must ensure that.

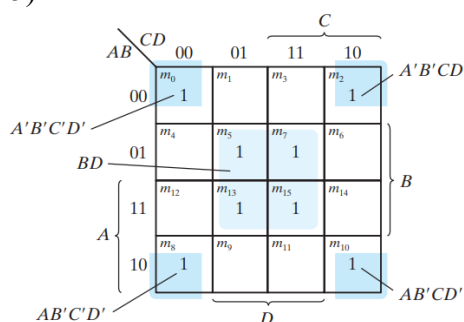
- (1) all the minterms of the function are covered when we combine the squares,
- (2) the number of terms in the expression is minimized, and
- (3) there are no redundant terms (i.e., minterms already covered by other terms).

Sometimes there may be two or more expressions that satisfy the simplification criteria. The procedure for combining squares in the map may be made more systematic if we understand the meaning of two special types of terms. A prime implicant is a product term obtained by combining the maximum possible number of adjacent squares in the map. If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be essential.

The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares. This means that a single 1 on a map represents a prime implicant if it is not adjacent to any other 1's. Two adjacent 1's form a prime implicant, provided that they are not within a group of four adjacent squares. Four adjacent 1's form a prime implicant if they are not within a group of eight adjacent squares, and so on.

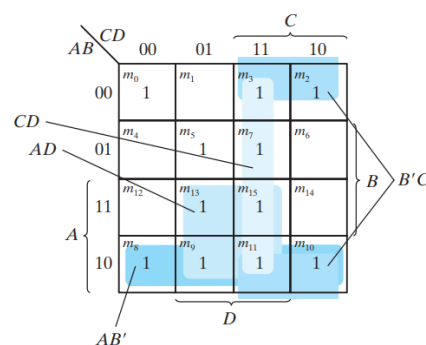
The essential prime implicants are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it. The prime implicant is essential if it is the only prime implicant that covers the minterm.

Consider the following four-variable Boolean function: $F(A, B, C, D) = \sum(0, 2, 3, 5, 7, 8, 9, 10, 11, 13, 15)$



Note: $A'B'C'D' + A'B'CD' = A'B'D'$
 $A'B'C'D + A'B'CD = A'B'D$
 $A'B'D' + A'B'D = B'D'$

(a) Essential prime implicants
 BD and $B'D'$



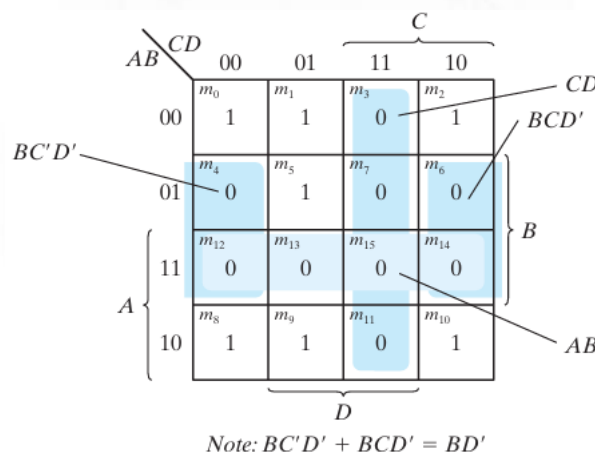
(b) Prime implicants CD , $B'C$,
 AD , and AB'

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

1.8 Product of sums simplification

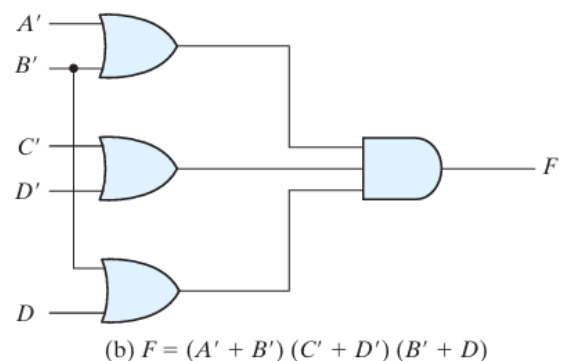
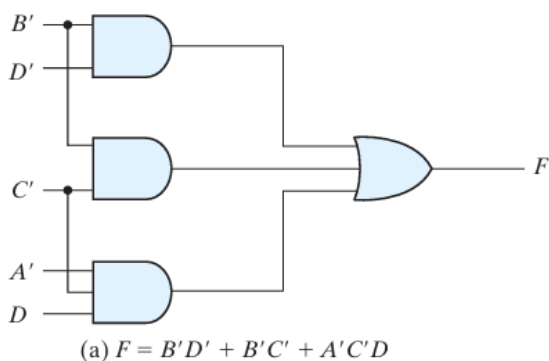
The procedure for obtaining a minimized function in product-of-sums form follows from the basic properties of Boolean functions. The 1's placed in the squares of the map represent the minterms of the function. The minterms not included in the standard sum-of-products form of a function denote the complement of the function. From this observation, we see that the complement of a function is represented in the map by the squares not marked by 1's. If we mark the empty squares by 0's and combine them into valid adjacent squares, we obtain a simplified sum-of-products expression of the complement of the function (i.e., of F'). The complement of F' gives us back the function F in product-of-sums form (a consequence of DeMorgan's theorem). Because of the generalized DeMorgan's theorem, the function so obtained is automatically in product-of-sums form.

Example: Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form: $F(A, B, C, D) = \sum(0, 1, 2, 5, 8, 9, 10)$



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

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

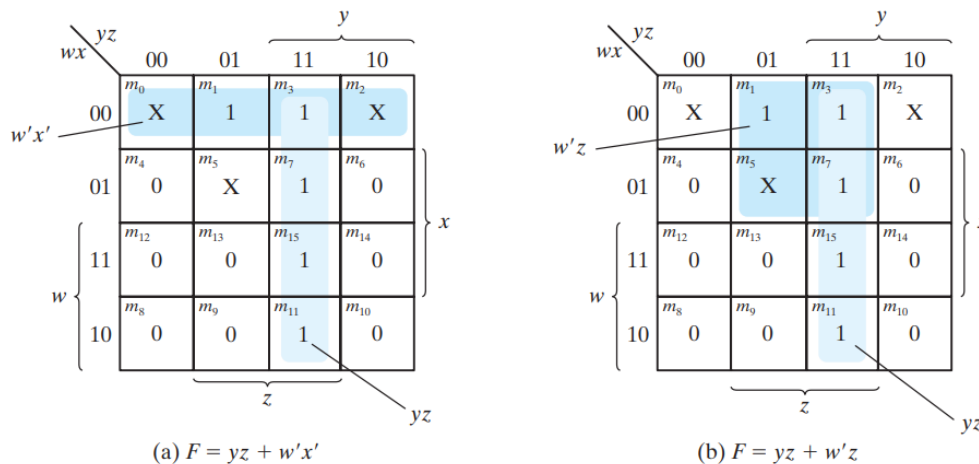


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

1.9 Don't-Care Conditions

A don't-care minterm is a combination of variables whose logical value is not specified. Such a minterm cannot be marked with a 1 in the map, because it would require that the function always be a 1 for such a combination. Likewise, putting a 0 on the square requires the function to be 0. To distinguish the don't-care condition from 1's and 0's, an X is used. Thus, an X inside a square in the map indicates that we don't care whether the value of 0 or 1 is assigned to F for the minterm.

Eg: Simplify the Boolean function $F(w, x, y, z) = \sum(1, 3, 7, 11, 15)$ which has the don't-care conditions $d(w, x, y, z) = \sum(0, 2, 5)$



In Fig. (a), don't-care minterms 0 and 2 are included with the 1's, resulting in the simplified function

$$F = yz + w'x'.$$

In Fig. (b), don't-care minterm 5 is included with the 1's, and the simplified function is now

$$F = yz + w'z$$

Either one of the preceding two expressions satisfies the conditions stated for this example.

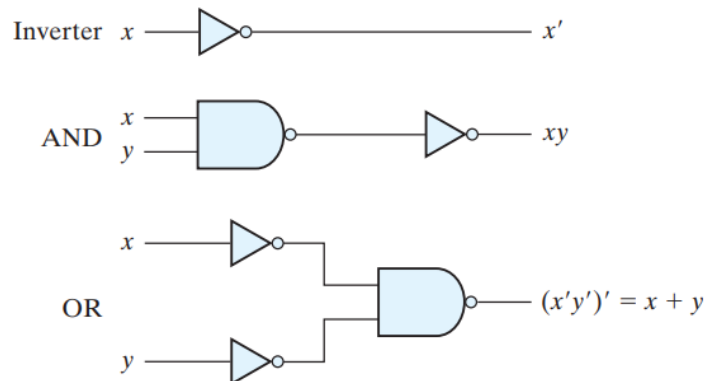
1.10 NAND and NOR Implementation

Digital circuits are frequently constructed with NAND or NOR gates rather than with AND and OR gates. NAND and NOR gates are easier to fabricate with electronic components and are the basic gates used in all IC digital logic families. Because of the prominence of NAND and NOR gates in the design of digital circuits, rules and procedures have been developed for the conversion from Boolean functions given in terms of AND, OR, and NOT into equivalent NAND and NOR logic diagrams.

NAND Circuits

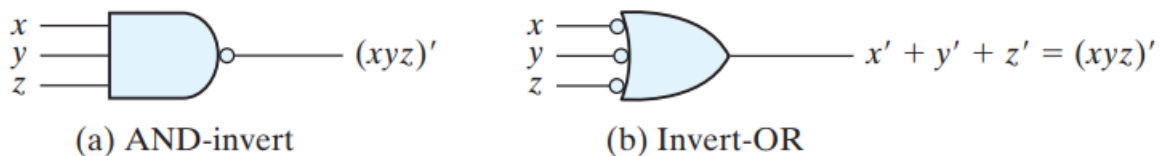
The NAND gate is said to be a universal gate because any logic circuit can be implemented with it. To show that any Boolean function can be implemented with NAND gates, we need only show that the logical operations of AND, OR, and complement can be obtained with NAND gates alone. This is indeed shown in Fig below.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



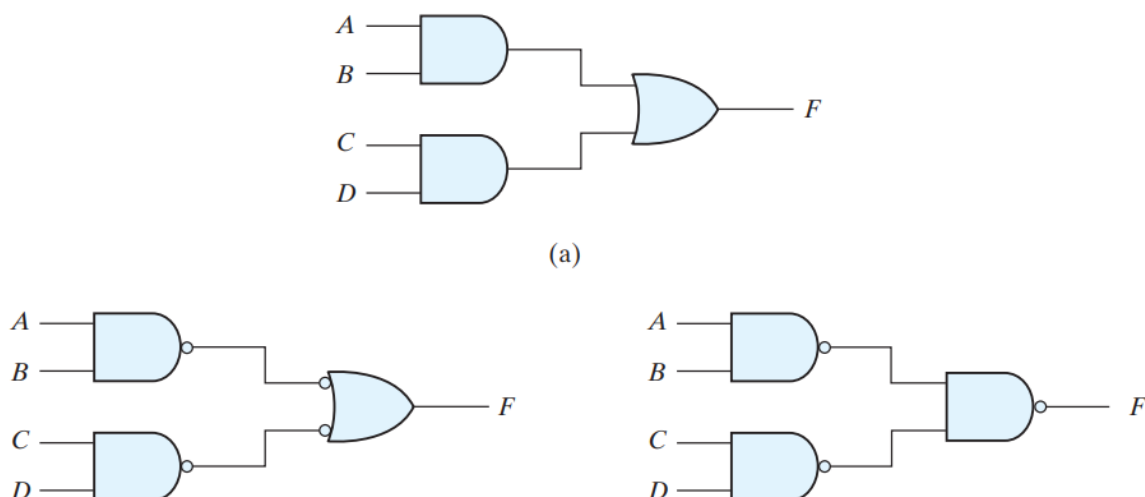
A convenient way to implement a Boolean function with NAND gates is to obtain the simplified Boolean function in terms of Boolean operators and then convert the function to NAND logic. The conversion of an algebraic expression from AND, OR, and complement to NAND can be done by simple circuit manipulation techniques that change AND-OR diagrams to NAND diagrams.

To facilitate the conversion to NAND logic, it is convenient to define an alternative graphic symbol for the gate. Two equivalent graphic symbols for the NAND gate are shown in Fig.



Two-Level Implementation

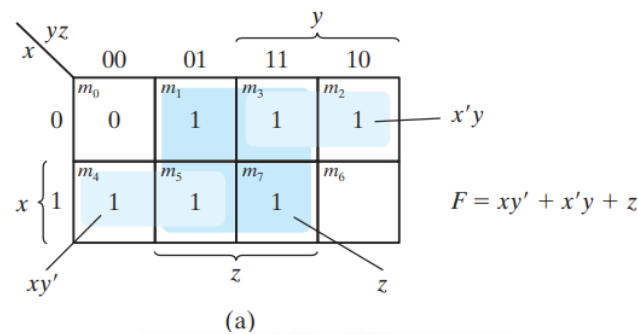
The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form. To see the relationship between a sum-of-products expression and its equivalent NAND implementation, consider the logic diagrams drawn in Fig. All three diagrams are equivalent and implement the function $F = AB + CD$



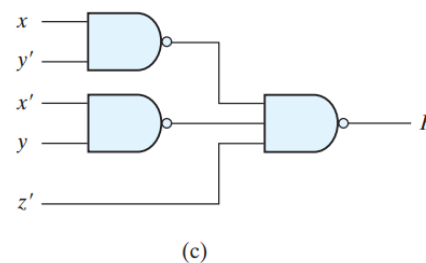
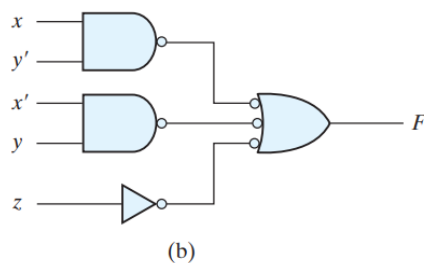
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Eg: Implement the following Boolean function with NAND gates: $F(x, y, z) = (1, 2, 3, 4, 5, 7)$

The first step is to simplify the function into sum-of-products form. This is done by means of the map of Fig. below, from which the simplified function is obtained: $F = xy' + x'y + z$

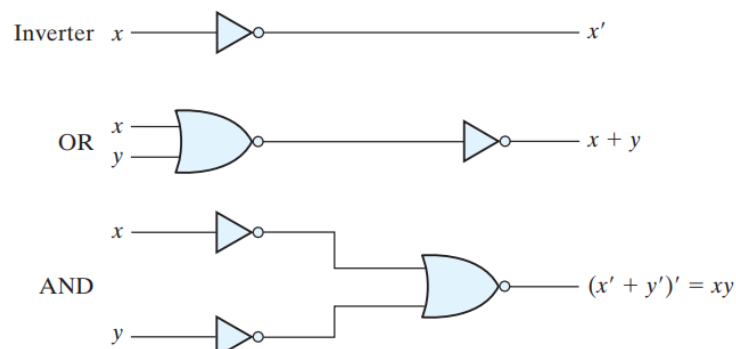


The two-level NAND implementation is shown in Fig. (b) in mixed notation. Note that input z must have a one-input NAND gate (an inverter) to compensate for the bubble in the second-level gate. An alternative way of drawing the logic diagram is given in Fig. (c). Here, all the NAND gates are drawn with the same graphic symbol. The inverter with input z has been removed, but the input variable is complemented and denoted by z' .



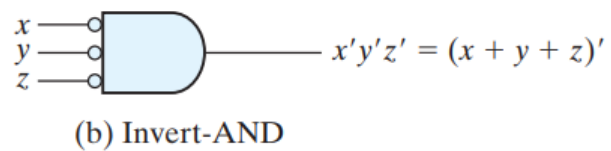
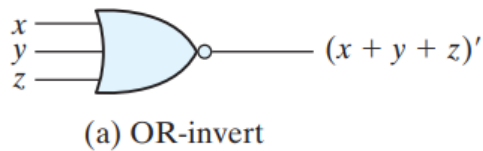
NOR Implementation

The NOR operation is the dual of the NAND operation. Therefore, all procedures and rules for NOR logic are the duals of the corresponding procedures and rules developed for NAND logic. The NOR gate is another universal gate that can be used to implement any Boolean function. The implementation of the complement, OR, and AND operations with NOR gates is shown in Fig below.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The two graphic symbols for the mixed notation are shown in below Fig. The OR-invert symbol defines the NOR operation as an OR followed by a complement. The invert-AND symbol complements each input and then performs an AND operation. The two symbols designate the same NOR operation and are logically identical because of DeMorgan's theorem.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Module-2

Chapter 1: Combinational Logic

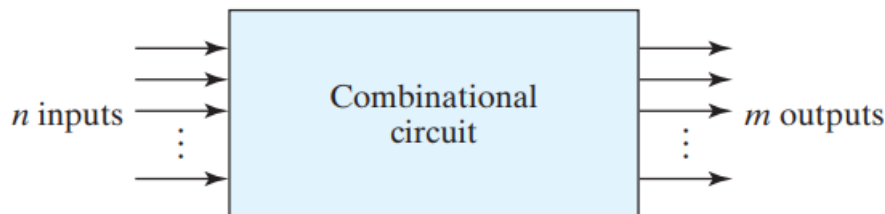
2.1 Introduction

Logic circuits for digital systems may be combinational or sequential. A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs. A combinational circuit performs an operation that can be specified logically by a set of Boolean functions.

In contrast, sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements. Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behavior must be specified by a time sequence of inputs and internal states.

2.2 Combinational Circuits

A combinational circuit consists of an interconnection of logic gates. Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data. A block diagram of a combinational circuit is shown in Fig. below.



The n input binary variables come from an external source; the m output variables are produced by the internal combinational logic circuit and go to an external destination. Each input and output variable exists physically as an analog signal whose values are interpreted to be a binary signal that represents logic 1 and logic 0. In many applications, the source and destination are storage registers. If the registers are included with the combinational gates, then the total circuit must be a sequential circuit.

For n input variables, there are 2^n possible combinations of the binary inputs. For each possible input combination, there is one possible value for each output variable. Thus, a combinational circuit can be specified with a truth table that lists the output values for each combination of input variables.

A combinational circuit also can be described by m Boolean functions, one for each output variable. Each output function is expressed in terms of the n input variables.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2.3 Analysis of Combinational Circuits

The analysis of a combinational circuit requires that determine the function that the circuit implements. This task starts with a given logic diagram and culminates with a set of Boolean functions, a truth table, or, possibly, an explanation of the circuit operation.

If the logic diagram to be analyzed is accompanied by a function name or an explanation of what it is assumed to accomplish, then the analysis problem reduces to a verification of the stated function. The analysis can be performed manually by finding the Boolean functions or truth table or by using a computer simulation program.

The first step in the analysis is to make sure that the given circuit is combinational and not sequential. The diagram of a combinational circuit has logic gates with no feedback paths or memory elements. A feedback path is a connection from the output of one gate to the input of a second gate whose output forms part of the input to the first gate. Feedback paths in a digital circuit define a sequential circuit and must be analyzed by special methods and will not be considered here.

Once the logic diagram is verified to be that of a combinational circuit, proceed to obtain the output Boolean functions or the truth table. If the function of the circuit is under investigation, then it is necessary to interpret the operation of the circuit from the derived Boolean functions or truth table.

To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols—but with meaningful names. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

The analysis of the combinational circuit of below figure illustrates the proposed procedure. We note that the circuit has three binary inputs - A, B, and C and two binary outputs - F_1 and F_2 . The outputs of various gates are labeled with intermediate symbols. The outputs of gates that are a function only of input variables are T_1 and T_2 . Output F_2 can easily be derived from the input variables. The Boolean functions for these three outputs are

$$F_2 = AB + AC + BC$$

$$T_1 = A + B + C$$

$$T_2 = ABC$$

Next, we consider outputs of gates that are a function of already defined symbols:

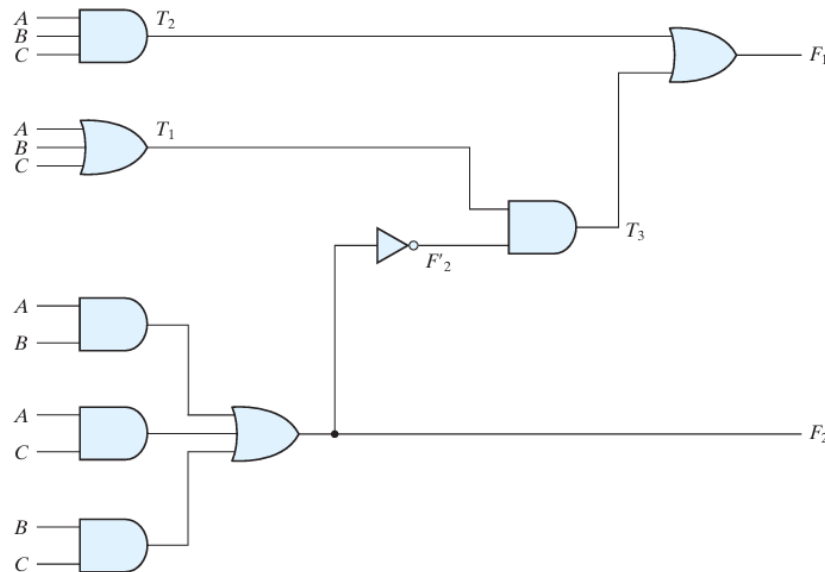
$$T_3 = F_2 T_1$$

$$F_1 = T_3 + T_2$$

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

To obtain F_1 as a function of A , B , and C , we form a series of substitutions as follows:

$$\begin{aligned} F_1 &= T_3 + T_2 = F'_2 T_1 + ABC = (AB + AC + BC)' (A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$



The derivation of the truth table for a circuit is a straightforward process once the output Boolean functions are known. To obtain the truth table directly from the logic diagram without going through the derivations of the Boolean functions, we proceed as follows:

1. Determine the number of input variables in the circuit. For n inputs, form the 2^n possible input combinations and list the binary numbers from 0 to $(2^n - 1)$ in a table.
2. Label the outputs of selected gates with arbitrary symbols.
3. Obtain the truth table for the outputs of those gates which are a function of the input variables only.
4. Proceed to obtain the truth table for the outputs of those gates which are a function of previously defined values until the columns for all outputs are determined.

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

This process is illustrated with the above circuit. In Table above, we form the eight possible combinations for the three input variables. The truth table for F_2 is determined directly from

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

the values of A, B, and C, with F_2 equal to 1 for any combination that has two or three inputs equal to 1. The truth table for F_2 is the complement of F_2 . The truth tables for T_1 and T_2 are the OR and AND functions of the input variables, respectively. The values for T_3 are derived from T_1 and F_2 : T_3 is equal to 1 when both T_1 and F_2 are equal to 1, and T_3 is equal to 0 otherwise. Finally, F_1 is equal to 1 for those combinations in which either T_2 or T_3 or both are equal to 1.

2.4 Design Procedure

The design of combinational circuits starts from the specification of the design objective and culminates in a logic circuit diagram or a set of Boolean functions from which the logic diagram can be obtained. The procedure involves the following steps:

1. From the specifications of the circuit, determine the required number of inputs and outputs and assign a symbol to each.
2. Derive the truth table that defines the required relationship between inputs and outputs.
3. Obtain the simplified Boolean functions for each output as a function of the input variables.
4. Draw the logic diagram and verify the correctness of the design (manually or by simulation).

A truth table for a combinational circuit consists of input columns and output columns. The input columns are obtained from the 2^n binary numbers for the n input variables. The binary values for the outputs are determined from the stated specifications. The output functions specified in the truth table give the exact definition of the combinational circuit. The output binary functions listed in the truth table are simplified by any available method, such as algebraic manipulation, the map method, or a computer-based simplification program.

Frequently, there is a variety of simplified expressions from which to choose. In a particular application, certain criteria will serve as a guide in the process of choosing an implementation. A practical design must consider such constraints as the number of gates, number of inputs to a gate, propagation time of the signal through the gates, number of interconnections, limitations of the driving capability of each gate (i.e., the number of gates to which the output of the circuit may be connected), and various other criteria that must be taken into consideration when designing integrated circuits. Since the importance of each constraint is dictated by the application, it is difficult to make a general statement about what constitutes an acceptable implementation.

In most cases, the simplification begins by satisfying an elementary objective, such as producing the simplified Boolean functions in a standard form. Then the simplification proceeds with further steps to meet other performance criteria.

Code Conversion Example

The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus, a code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

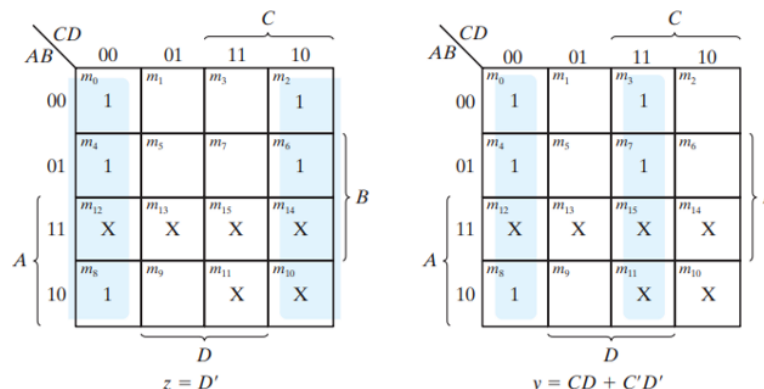
The design procedure will be illustrated by an example that converts binary coded decimal (BCD) to the excess-3 code for the decimal digits. Since each code uses four bits to represent a decimal digit, there must be four input variables and four output variables. Let the four input binary variables be the symbols A, B, C, and D, and the four output variables be w, x, y, and z. The truth table relating the input and output variables is shown in Table below. Note that four binary variables may have 16-bit combinations, but only 10 are listed in the truth table.

Truth Table for Code Conversion Example

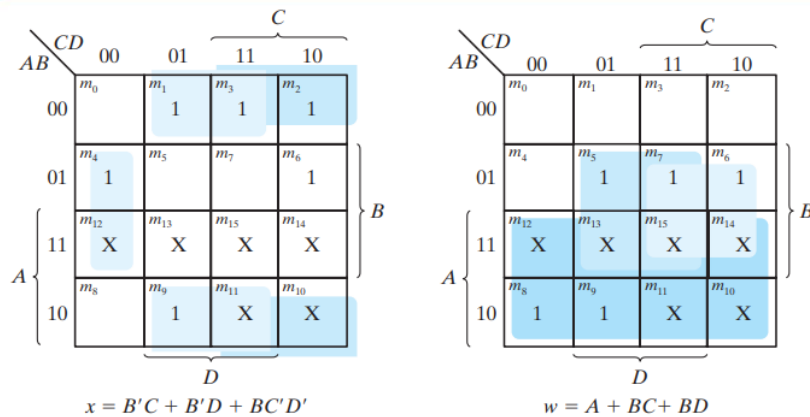
Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

The six-bit combinations not listed for the input variables are don't-care combinations. These values have no meaning in BCD and we assume that they will never occur in actual operation of the circuit. Therefore, we are at liberty to assign to the output variables either a 1 or a 0, whichever gives a simpler circuit.

The maps in below Fig. are plotted to obtain simplified Boolean functions for the outputs. Each one of the four maps represents one of the four outputs of the circuit as a function of the four input variables. The 1's marked inside the squares are obtained from the minterms that make the output equal to 1. The 1's are obtained from the truth table by going over the output columns one at a time. For example, the column under output z has five 1's; therefore, the map for z has five 1's, each being in a square corresponding to the minterm that makes z equal to 1. The six don't-care minterms 10 through 15 are marked with an X.



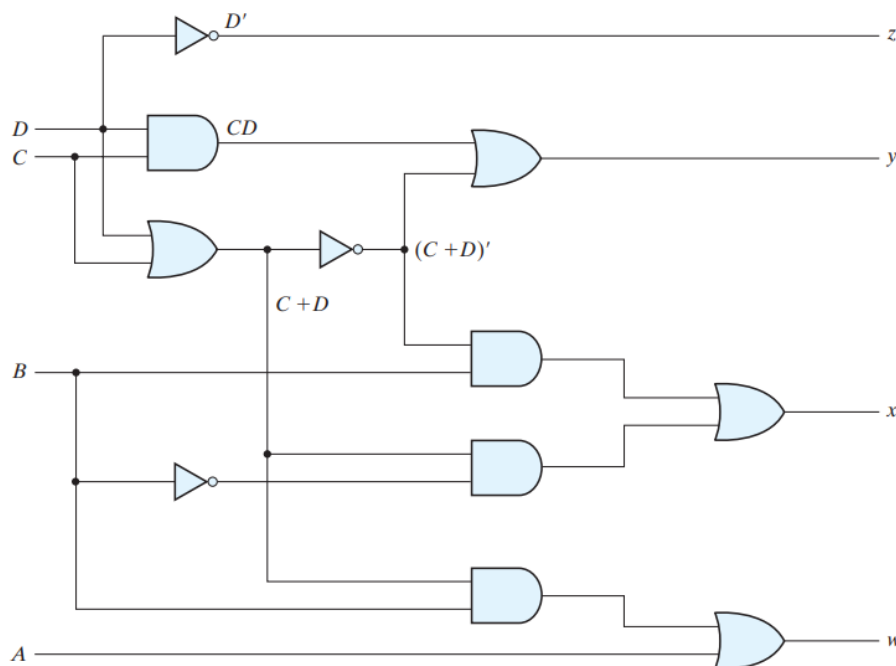
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



A two-level logic diagram for each output may be obtained directly from the Boolean expressions derived from the maps. There are various other possibilities for a logic diagram that implements this circuit. The expressions obtained in Fig. may be manipulated algebraically for the purpose of using common gates for two or more outputs. This manipulation, shown next, illustrates the flexibility obtained with multiple-output systems when implemented with three or more levels of gates:

$$\begin{aligned}
 z &= D' \\
 y &= CD + C'D' = CD + (C + D)' \\
 x &= B'C + B'D + BC'D' = B'(C + D) + BC'D' \\
 &= B'(C + D) + B(C + D)' \\
 w &= A + BC + BD = A + B(C + D)
 \end{aligned}$$

The logic diagram that implements these expressions is shown in Fig. below.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

2.5 Binary Adder – Subtractor

Digital computers perform a variety of information-processing tasks. Among the functions encountered are the various arithmetic operations. The most basic arithmetic operation is the addition of two binary digits. This simple addition consists of four possible elementary operations: $0 + 0 = 0$, $0 + 1 = 1$, $1 + 0 = 1$, and $1 + 1 = 10$. The first three operations produce a sum of one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a carry. When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher order pair of significant bits.

A combinational circuit that performs the addition of two bits is called a half adder. One that performs the addition of three bits (two significant bits and a previous carry) is a full adder. The names of the circuits stem from the fact that two half adders can be employed to implement a full adder. A binary adder–subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.

Half Adder

From the verbal explanation, half adder circuit needs two binary inputs and two binary outputs. The input variables designate the augend and addend bits; the output variables produce the sum and carry. Let symbols x and y be the two inputs and S (for sum) and C (for carry) to the outputs. The truth table for the half adder is listed in Table.

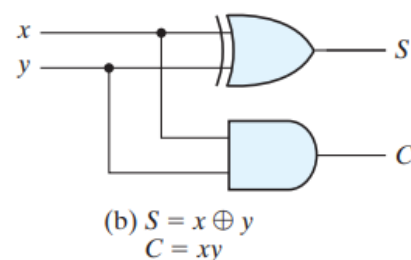
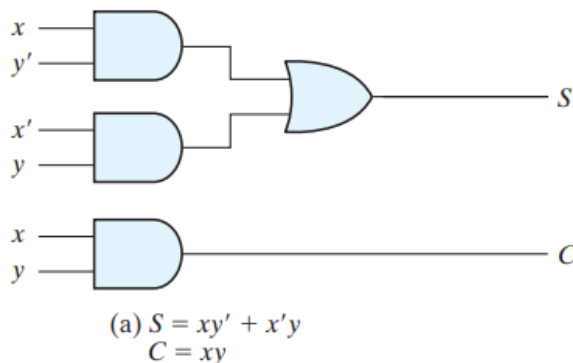
x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The C output is 1 only when both inputs are 1. The S output represents the least significant bit of the sum. The simplified Boolean functions for the two outputs can be obtained directly from the truth table. The simplified sum-of-products expressions are

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

$$C = xy$$

The logic diagram of the half adder implemented in sum of products is shown in Fig. (a). It can be also implemented with an exclusive-OR and an AND gate as shown in Fig.(b). This form is used to show that two half adders can be used to construct a full adder.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Full Adder

A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be added. The third input, z , represents the carry from the previous lower significant position. Two outputs are necessary because the arithmetic sum of three binary digits ranges in value from 0 to 3, and binary representation of 2 or 3 needs two bits. The two outputs are designated by the symbols S for sum and C for carry. The binary variable S gives the value of the least significant bit of the sum. The binary variable C gives the output carry formed by adding the input carry and the bits of the words. The truth table of the full adder is listed in Table below.

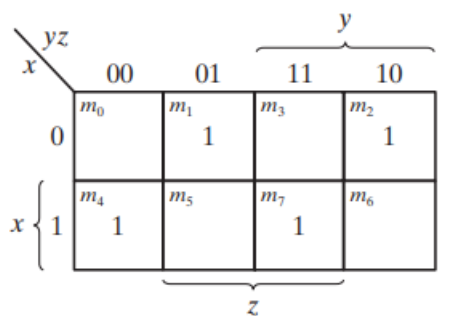
x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

The eight rows under the input variables designate all possible combinations of the three variables. The output variables are determined from the arithmetic sum of the input bits. When all input bits are 0, the output is 0. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output has a carry of 1 if two or three inputs are equal to 1.

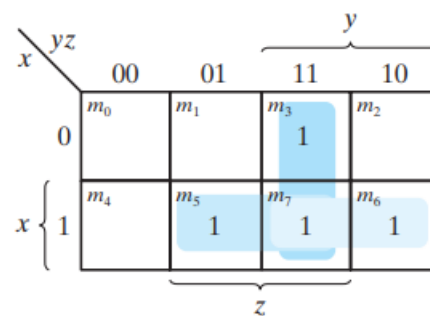
The maps for the outputs of the full adder are shown in Fig. below. The simplified expressions are,

$$S = x'y'z + x'yz' + xy'z' + xyz$$

$$C = xy + xz + yz$$



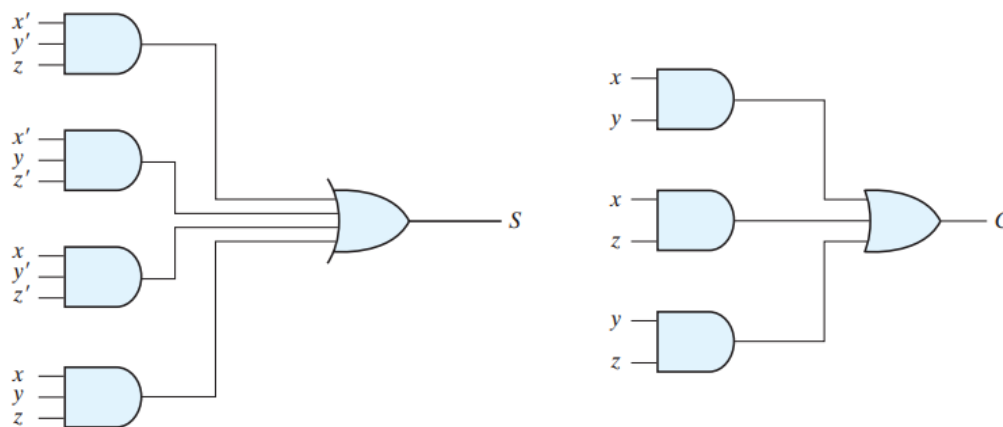
(a) $S = x'y'z + x'yz' + xy'z' + xyz$



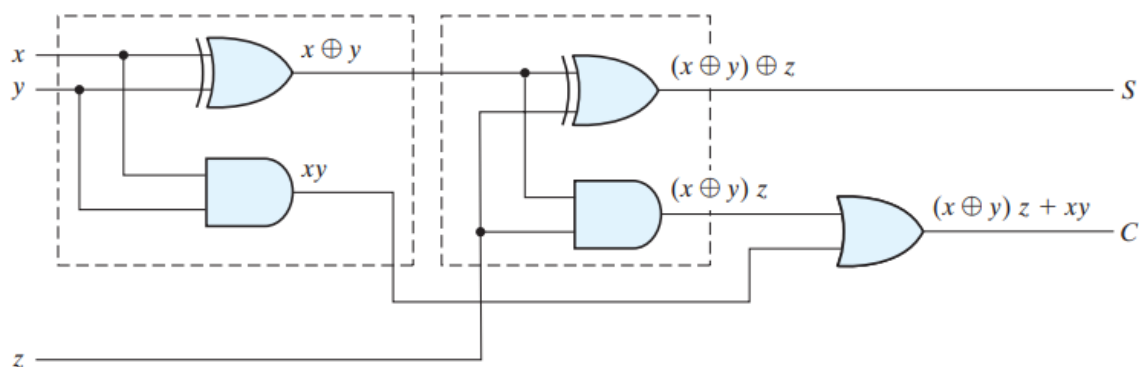
(b) $C = xy + xz + yz$

The logic diagram for the full adder implemented in sum-of-products form is shown in Fig. below.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



It can also be implemented with two half adders and one OR gate, as shown below.



The S output from the second half adder is the exclusive-OR of z and the output of the first half adder, giving

$$\begin{aligned}
 S &= z \oplus (x \oplus y) \\
 &= z'(xy' + x'y) + z(xy' + x'y)' \\
 &= z'(xy' + x'y) + z(xy + x'y') \\
 &= xy'z' + x'yz' + xyz + x'y'z
 \end{aligned}$$

The carry output is

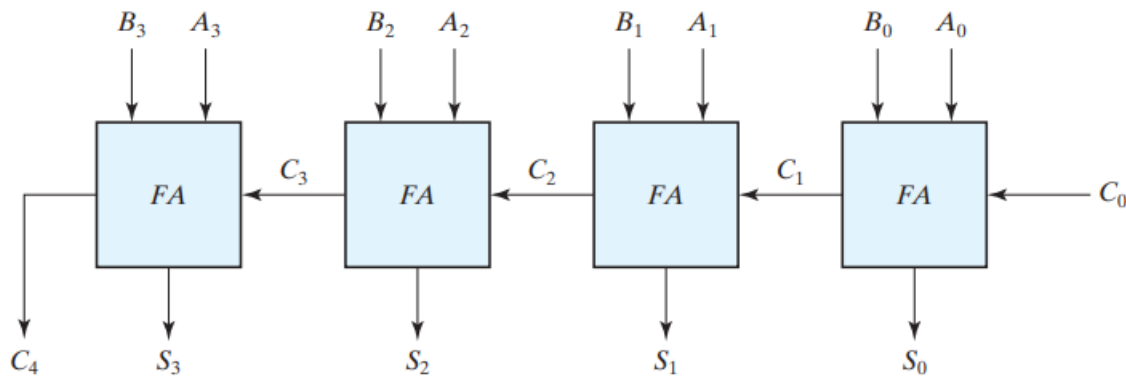
$$C = z(xy' + x'y) + xy = xy'z + x'yz + xy$$

Binary Adder

A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.

Addition of n-bit numbers requires a chain of n full adders or a chain of one-half adder and n-1 full adders. In the former case, the input carry to the least significant position is fixed at 0. Figure below shows the interconnection of four full-adder (FA) circuits to provide a four-bit binary ripple carry adder.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript 0 denoting the least significant bit. The carries are connected in a chain through the full adders. The input carry to the adder is C_0 , and it ripples through the full adders to the output carry C_4 . The S outputs generate the required sum bits. An n-bit adder requires n full adders, with each output carry connected to the input carry of the next higher order full adder. To demonstrate with a specific example, consider the two binary numbers $A = 1011$ and $B = 0011$. Their sum $S = 1110$ is formed with the four-bit adder as follows:

Subscript i:	3	2	1	0	
Input carry	0	1	1	0	C_i
Augend	1	0	1	1	A_i
Addend	0	0	1	1	B_i
Sum	1	1	1	0	S_i
Output carry	0	0	1	1	C_{i+1}

The bits are added with full adders, starting from the least significant position (subscript 0), to form the sum bit and carry bit. The input carry C_0 in the least significant position must be 0. The value of C_{i+1} in a given significant position is the output carry of the full adder. This value is transferred into the input carry of the full adder that adds the bits one higher significant position to the left. The sum bits are thus generated starting from the rightmost position and are available as soon as the corresponding previous carry bit is generated. All the carries must be generated for the correct sum bits to appear at the outputs.

Binary Subtractor

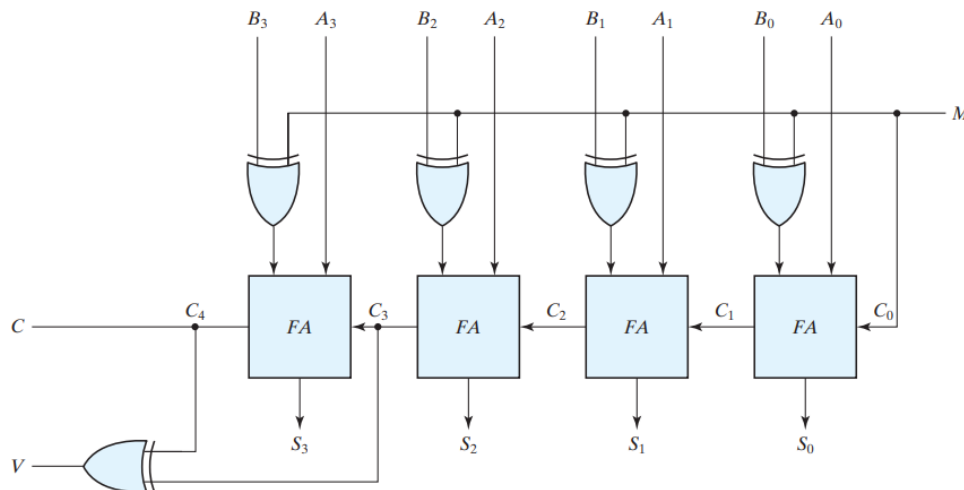
The subtraction of unsigned binary numbers can be done most conveniently by means of complements. Remember that the subtraction $A - B$ can be done by taking the 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

The circuit for subtracting $A - B$ consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when subtraction is performed. The operation thus performed becomes A, plus the 1's complement of B, plus 1. This is equal to A plus the 2's complement of B. For unsigned

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

numbers, that gives $A - B$ if $A \geq B$ or the 2's complement of $(B - A)$ if $A < B$. For signed numbers, the result is $A - B$, provided that there is no overflow.

The addition and subtraction operations can be combined into one circuit with one common binary adder by including an exclusive-OR gate with each full adder. A four-bit adder-subtractor circuit is shown in Fig. below. The mode input M controls the operation. When $M = 0$, the circuit is an adder, and when $M = 1$, the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B . When $M = 0$, we have $B \text{ ex-or } 0 = B$. The full adders receive the value of B , the input carry is 0, and the circuit performs $A \text{ plus } B$. When $M = 1$, we have $B \text{ ex-or } 1 = B'$ and $C_0 = 1$. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation $A \text{ plus the 2's complement of } B$.

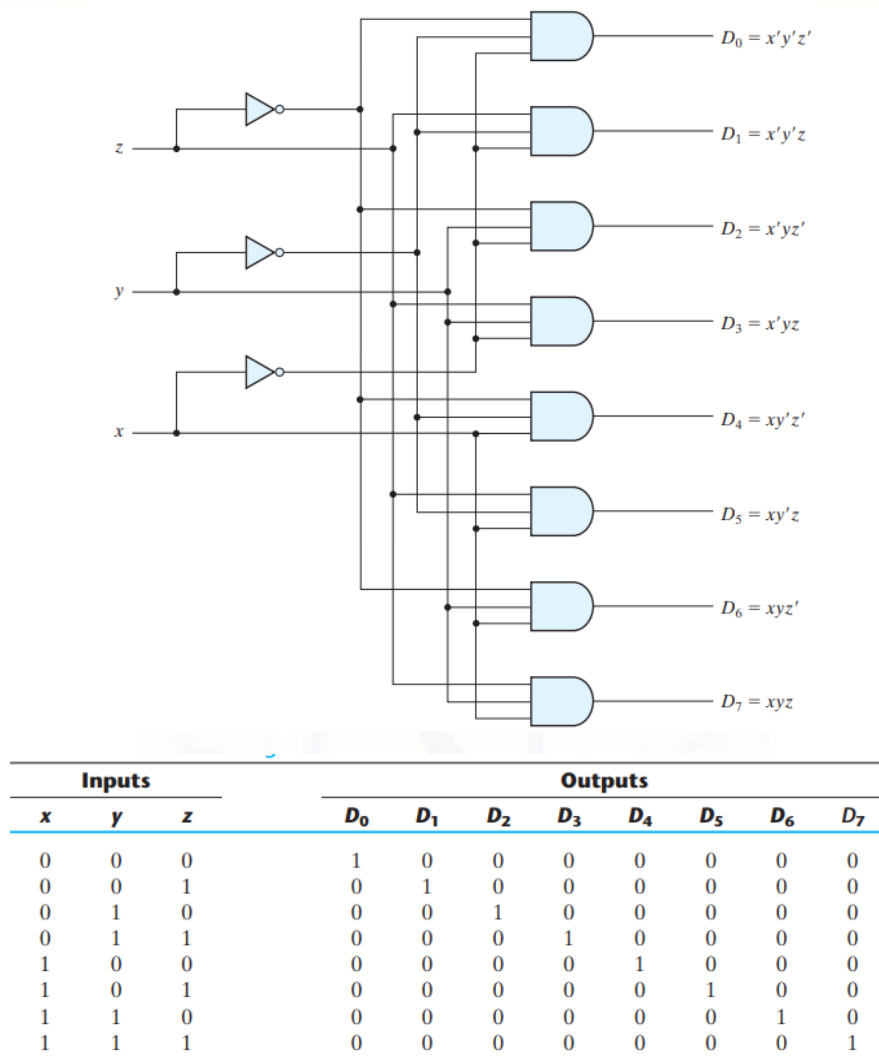


2.6 Decoders

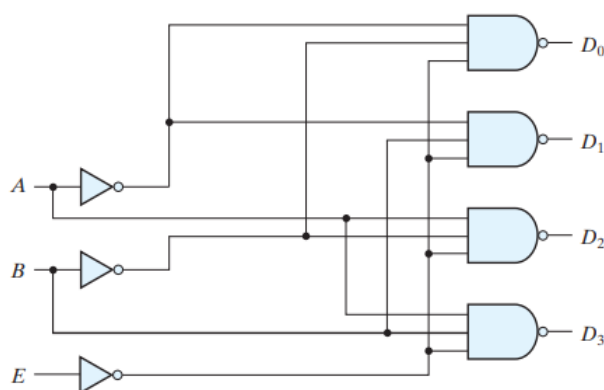
A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit coded information has unused combinations, the decoder may have fewer than 2^n outputs. The decoders presented here are called n -to- m -line decoders, where $m \leq 2^n$. Their purpose is to generate the 2^n (or fewer) minterms of n input variables.

Each combination of inputs will assert a unique output. The name decoder is also used in conjunction with other code converters, such as a BCD-to-seven-segment decoder. As an example, consider the three-to-eight-line decoder circuit of Fig. below. The three inputs are decoded into eight outputs, each representing one of the minterms of the three input variables. The three inverters provide the complement of the inputs, and each one of the eight AND gates generate one of the minterms. A particular application of this decoder is binary-to-octal conversion. The input variables represent a binary number, and the outputs represent the eight digits of a number in the octal number system. However, a three-to-eight-line decoder can be used for decoding any three-bit code to provide eight outputs, one for each element of the code.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Furthermore, decoders include one or more enable inputs to control the circuit operation. A two-to-four-line decoder with an enable input constructed with NAND gates is shown in Fig. below. The circuit operates with complemented outputs and a complement enable input. The decoder is enabled when E is equal to 0 (i.e., active-low enable).



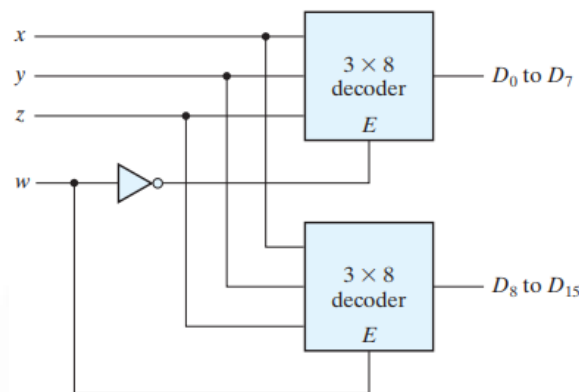
(a) Logic diagram

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

(b) Truth table

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Decoders with enable inputs can be connected together to form a larger decoder circuit. Figure below shows two 3-to-8-line decoders with enable inputs connected to form a 4-to-16-line decoder. When $w=0$, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0111. When $w=1$, the enable conditions are reversed: The bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's. This example demonstrates the usefulness of enable inputs in decoders and other combinational logic components. In general, enable inputs are a convenient feature for interconnecting two or more standard components for the purpose of combining them into a similar function with more inputs and outputs.



2.7 Encoders

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

An example of an encoder is the octal-to-binary encoder whose truth table is given in Table. It has eight inputs (one for each of the octal digits) and three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time. The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7. Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following Boolean output functions:

$$\begin{aligned} z &= D_1 + D_3 + D_5 + D_7 \\ y &= D_2 + D_3 + D_6 + D_7 \\ x &= D_4 + D_5 + D_6 + D_7 \end{aligned}$$

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

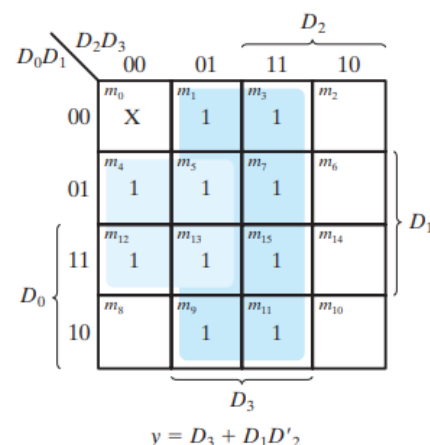
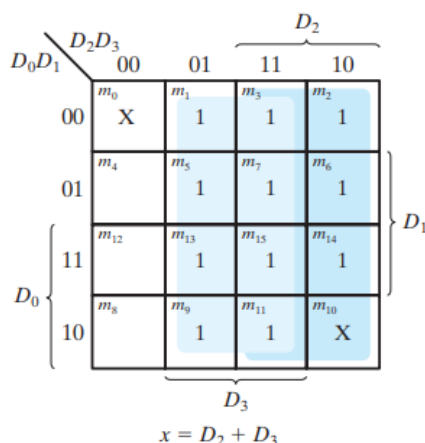
The encoder defined in Table has the limitation that only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination. For example, if D_3 and D_6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. The output 111 does not represent either binary 3 or binary 6. To resolve this ambiguity, encoder circuits must establish an input priority to ensure that only one input is encoded.

Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; but this output is the same as when D_0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equal to 1 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. In addition to the two outputs x and y , the circuit has a third output designated by V ; this is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't-care conditions. Note that whereas X's in output columns represent don't-care conditions, the X's in the input columns are useful for representing a truth table in condensed form. Instead of listing all 16 minterms of four variables, the truth table uses an X to represent either 1 or 0. For example, X 100 represents the two minterms 0100 and 1100.

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

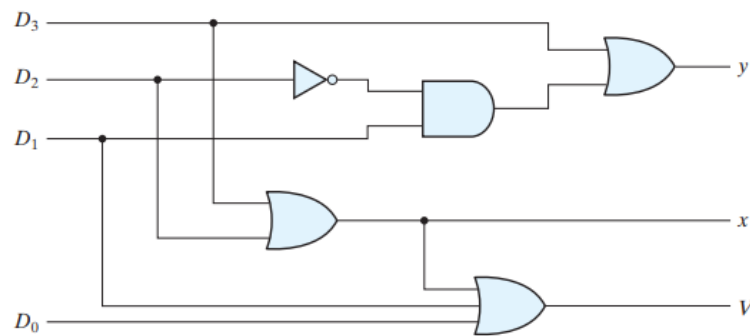


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

$$x = D_2 + D_3$$

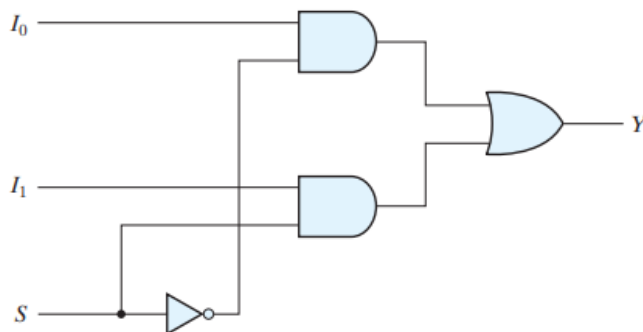
$$y = D_3 + D_1 D_2'$$

$$V = D_0 + D_1 + D_2 + D_3$$

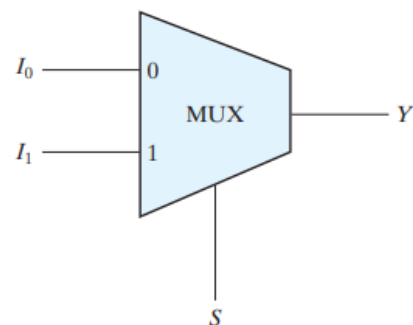


2.8 Multiplexers

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.



(a) Logic diagram



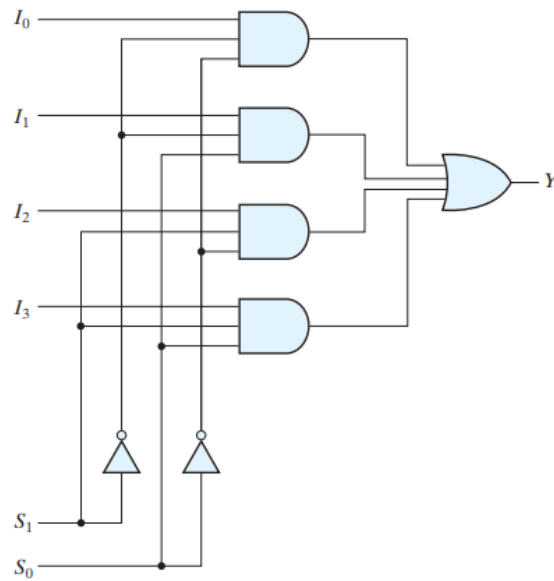
(b) Block diagram

A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination, as shown in Fig. The circuit has two data input lines, one output line, and one selection line S. When $S = 0$, the upper AND gate is enabled and I_0 has a path to the output. When $S = 1$, the lower AND gate is enabled and I_1 has a path to the output. The multiplexer acts like an electronic switch that selects one of two sources. The block diagram of a multiplexer is sometimes depicted by a wedge-shaped symbol, as shown in Fig.(b). It suggests visually how a selected one of multiple data sources is directed into a single destination. The multiplexer is often labeled “MUX” in block diagrams.

A four-to-one-line multiplexer is shown in figure below. Each of the four inputs, I_0 through I_3 , is applied to one input of an AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gates are applied to a single OR gate that provides the one-line output. The function table lists the input that is passed to the output for each combination of the binary selection values. To demonstrate the operation of the circuit, consider

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

the case when $S_1S_0 = 10$. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2 . The other three AND gates have at least one input equal to 0, which makes their outputs equal to 0. The output of the OR gate is now equal to the value of I_2 , providing a path from the selected input to the output. A multiplexer is also called a data selector, since it selects one of many inputs and steers the binary information to the output line.



(a) Logic diagram

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

(b) Function table

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

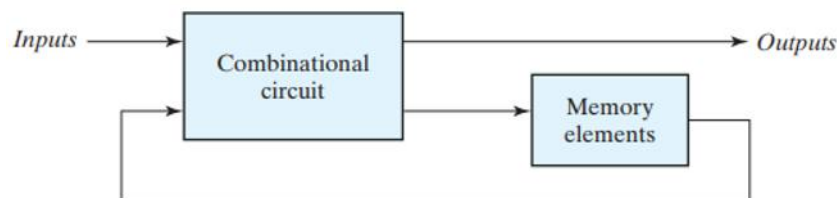
Chapter 2: Synchronous Sequential Logic

2.9 Introduction

Hand-held devices, cell phones, navigation receivers, personal computers, digital cameras, personal media players, and virtually all electronic consumer products can send, receive, store, retrieve, and process information represented in a binary format. The technology enabling and supporting these devices is critically dependent on electronic components that can store information, i.e., have memory. The digital circuits considered thus far have been combinational—their output depends only and immediately on their inputs—they have no memory, i.e., dependence on past values of their inputs. Sequential circuits, however, act as storage elements and have memory. They can store, retain, and then retrieve information when needed later.

2.10 Sequential Circuits

A block diagram of a sequential circuit is shown in Fig. below. It consists of a combinational circuit to which storage elements are connected to form a feedback path. The storage elements are devices capable of storing binary information. The binary information stored in these elements at any given time defines the state of the sequential circuit at that time. The sequential circuit receives binary information from external inputs that, together with the present state of the storage elements, determine the binary value of the outputs. These external inputs also determine the condition for changing the state in the storage elements.



The block diagram demonstrates that the outputs in a sequential circuit are a function not only of the inputs, but also of the present state of the storage elements. The next state of the storage elements is also a function of external inputs and the present state. Thus, a sequential circuit is specified by a time sequence of inputs, outputs, and internal states. In contrast, the outputs of combinational logic depend only on the present values of the inputs. There are two main types of sequential circuits, and their classification is a function of the timing of their signals. A synchronous sequential circuit is a system whose behavior can be defined from the knowledge of its signals at discrete instants of time. The behavior of an asynchronous sequential circuit depends upon the input signals at any instant of time and the order in which the inputs change. The storage elements commonly used in asynchronous sequential circuits are time-delay devices.

The storage capability of a time-delay device varies with the time it takes for the signal to propagate through the device. In practice, the internal propagation delay of logic gates is of sufficient duration to produce the needed delay, so that actual delay units may not be necessary.

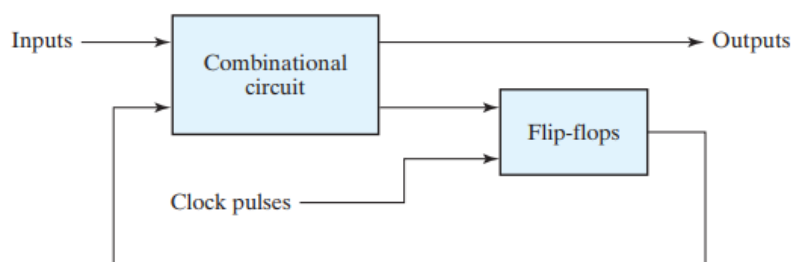
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

In gate-type asynchronous systems, the storage elements consist of logic gates whose propagation delay provides the required storage. Thus, an asynchronous sequential circuit may be regarded as a combinational circuit with feedback. Because of the feedback among logic gates, an asynchronous sequential circuit may become unstable at times.

A synchronous sequential circuit employs signals that affect the storage elements at only discrete instants of time. Synchronization is achieved by a timing device called a clock generator, which provides a clock signal having the form of a periodic train of clock pulses. The clock signal is commonly denoted by the identifiers clock and clk. The clock pulses are distributed throughout the system in such a way that storage elements are affected only with the arrival of each pulse. In practice, the clock pulses determine when computational activity will occur within the circuit, and other signals (external inputs and otherwise) determine what changes will take place affecting the storage elements and the outputs.

For example, a circuit that is to add and store two binary numbers would compute their sum from the values of the numbers and store the sum at the occurrence of a clock pulse. Synchronous sequential circuits that use clock pulses to control storage elements are called clocked sequential circuits and are the type most frequently encountered in practice. They are called synchronous circuits because the activity within the circuit and the resulting updating of stored values is synchronized to the occurrence of clock pulses. The design of synchronous circuits is feasible because they seldom manifest instability problems and their timing is easily broken down into independent discrete steps, each of which can be considered separately.

The storage elements (memory) used in clocked sequential circuits are called flipflops. A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1. A sequential circuit may use many flip-flops to store as many bits as necessary. The block diagram of a synchronous clocked sequential circuit is shown in Fig. below. The outputs are formed by a combinational logic function of the inputs to the circuit, or the values stored in the flip-flops (or both). The value that is stored in a flip-flop when the clock pulse occurs is also determined by the inputs to the circuit or the values presently stored in the flip-flop (or both). The new value is stored (i.e., the flip-flop is updated) when a pulse of the clock signal occurs.



(a) Block diagram



(b) Timing diagram of clock pulses

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Prior to the occurrence of the clock pulse, the combinational logic forming the next value of the flip-flop must have reached a stable value. Consequently, the speed at which the combinational logic circuits operate is critical. If the clock (synchronizing) pulses arrive at a regular interval, as shown in the timing diagram, the combinational logic must respond to a change in the state of the flip-flop in time to be updated before the next pulse arrives.

Propagation delays play an important role in determining the minimum interval between clock pulses that will allow the circuit to operate correctly. A change in state of the flip-flops is initiated only by a clock pulse transition—for example, when the value of the clock signals changes from 0 to 1. When a clock pulse is not active, the feedback loop between the value stored in the flip-flop and the value formed at the input to the flip-flop is effectively broken because the flip-flop outputs cannot change even if the outputs of the combinational circuit driving their inputs change in value. Thus, the transition from one state to the next occurs only at predetermined intervals dictated by the clock pulses.

2.11 Storage Elements: Latches, Flip-Flops

Latches

A storage element in a digital circuit can maintain a binary state indefinitely, until directed by an input signal to switch states. The major differences among various types of storage elements are in the number of inputs they possess and in the way the inputs affect the binary state.

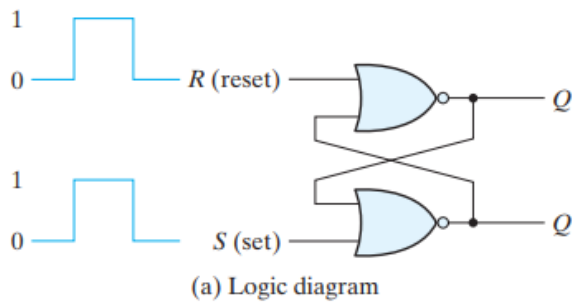
Storage elements that operate with signal levels (rather than signal transitions) are referred to as latches; those controlled by a clock transition are flip-flops. Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices. The two types of storage elements are related because latches are the basic circuits from which all flip-flops are constructed. Although latches are useful for storing binary information and for the design of asynchronous sequential circuits, they are not practical for use as storage elements in synchronous sequential circuits.

SR Latch

The SR latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled S for set and R for reset. The SR latch constructed with two cross-coupled NOR gates is shown in Fig. below. The latch has two useful states.

When output $Q = 1$ and $Q' = 0$, the latch is said to be in the set state. When $Q = 0$ and $Q' = 1$, it is in the reset state. Outputs Q and Q' are normally the complement of each other. However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs. If both inputs are then switched to 0 simultaneously, the device will enter an unpredictable or undefined state or a meta-stable state. Consequently, in practical applications, setting both inputs to 1 is forbidden. Under normal conditions, both inputs of the latch remain at 0 unless the state has to be changed. The application of a momentary 1 to the S input causes the latch to go to the set state. The S input must go back to 0 before any other changes take place, in order to avoid the occurrence of an undefined next state that results from the forbidden input condition.

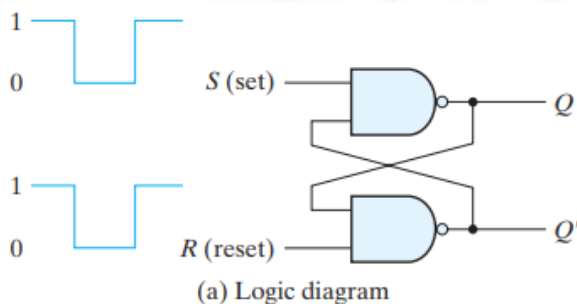
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

(b) Function table

As shown in the function table of Fig. (b), two input conditions cause the circuit to be in the set state. The first condition ($S = 1, R = 0$) is the action that must be taken by input S to bring the circuit to the set state. Removing the active input from S leaves the circuit in the same state. After both inputs return to 0, it is then possible to shift to the reset state by momentarily applying a 1 to the R input. The 1 can then be removed from R, whereupon the circuit remains in the reset state. Thus, when both inputs S and R are equal to 0, the latch can be in either the set or the reset state, depending on which input was most recently a 1. If a 1 is applied to both the S and R inputs of the latch, both outputs go to 0. This action produces an undefined next state because the state that results from the input transitions depends on the order in which they return to 0. It also violates the requirement that outputs be the complement of each other. In normal operation, this condition is avoided by making sure that 1's are not applied to both inputs simultaneously.



S	R	Q	Q'
1	0	0	1
1	1	0	1
0	1	1	0
1	1	1	0
0	0	1	1

(b) Function table

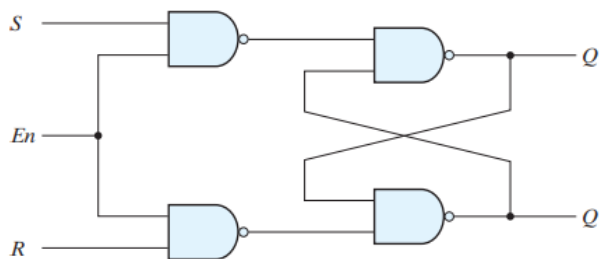
The SR latch with two cross-coupled NAND gates is shown in Fig. above. It operates with both inputs normally at 1 unless the state of the latch has to be changed. The application of 0 to the S input causes output Q to go to 1, putting the latch in the set state. When the S input goes back to 1, the circuit remains in the set state. After both inputs go back to 1, we are allowed to change the state of the latch by placing a 0 in the R input. This action causes the circuit to go to the reset state and stay there even after both inputs return to 1. The condition that is forbidden for the NAND latch is both inputs being equal to 0 at the same time, an input combination that should be avoided.

In comparing the NAND with the NOR latch, note that the input signals for the NAND require the complement of those values used for the NOR latch. Because the NAND latch requires a 0 signal to change its state, it is sometimes referred to as an $S'R'$ latch. The primes (or, sometimes, bars over the letters) designate the fact that the inputs must be in their complement form to activate the circuit. The operation of the basic SR latch can be modified by providing an

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

additional input signal that determines (controls) when the state of the latch can be changed by determining whether S and R (or S' and R') can affect the circuit.

An SR latch with a control input is shown in Fig. below. It consists of the basic SR latch and two additional NAND gates. The control input En acts as an enable signal for the other two inputs. The outputs of the NAND gates stay at the logic-1 level as long as the enable signal remains at 0. This is the quiescent condition for the SR latch. When the enable input goes to 1, information from the S or R input is allowed to affect the latch. The set state is reached with $S = 1$, $R = 0$, and $En = 1$ (active-high enabled). To change to the reset state, the inputs must be $S = 0$, $R = 1$, and $En = 1$. In either case, when En returns to 0, the circuit remains in its current state. The control input disables the circuit by applying 0 to En , so that the state of the output does not change regardless of the values of S and R. Moreover, when $En = 1$ and both the S and R inputs are equal to 0, the state of the circuit does not change. These conditions are listed in the function table accompanying the diagram. An indeterminate condition occurs when all three inputs are equal to 1. This condition places 0's on both inputs of the basic SR latch, which puts it in the undefined state. When the enable input goes back to 0, one cannot conclusively determine the next state, because it depends on whether the S or R input goes to 0 first. This indeterminate condition makes this circuit difficult to manage, and it is seldom used in practice.



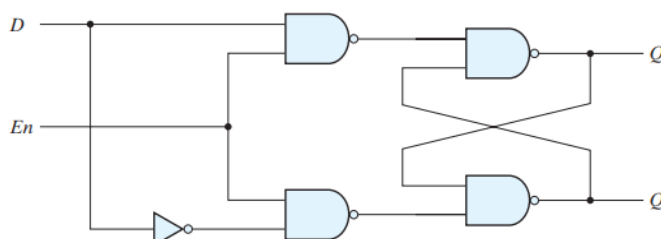
(a) Logic diagram

En	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	$Q = 0$; reset state
1	1	0	$Q = 1$; set state
1	1	1	Indeterminate

(b) Function table

D Latch (Transparent Latch)

One way to eliminate the undesirable condition of the indeterminate state in the SR latch is to ensure that inputs S and R are never equal to 1 at the same time. This is done in the D latch, shown in Fig. below. This latch has only two inputs: D (data) and En (enable). The D input goes directly to the S input, and its complement is applied to the R input. As long as the enable input is at 0, the cross-coupled SR latch has both inputs at the 1 level and the circuit cannot change state regardless of the value of D. The D input is sampled when $En = 1$. If $D = 1$, the Q output goes to 1, placing the circuit in the set state. If $D = 0$, output Q goes to 0, placing the circuit in the reset state.



(a) Logic diagram

En	D	Next state of Q
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

(b) Function table

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The D latch receives that designation from its ability to hold data in its internal storage. It is suited for use as a temporary storage for binary information between a unit and its environment. The binary information present at the data input of the D latch is transferred to the Q output when the enable input is asserted. The output follows changes in the data input as long as the enable input is asserted. This situation provides a path from input D to the output, and for this reason, the circuit is often called a transparent latch. When the enable input signal is deasserted, the binary information that was present at the data input at the time the transition occurred is retained (i.e., stored) at the Q output until the enable input is asserted again. Note that an inverter could be placed at the enable input. Then, depending on the physical circuit, the external enabling signal will be a value of 0 (active low) or 1 (active high).

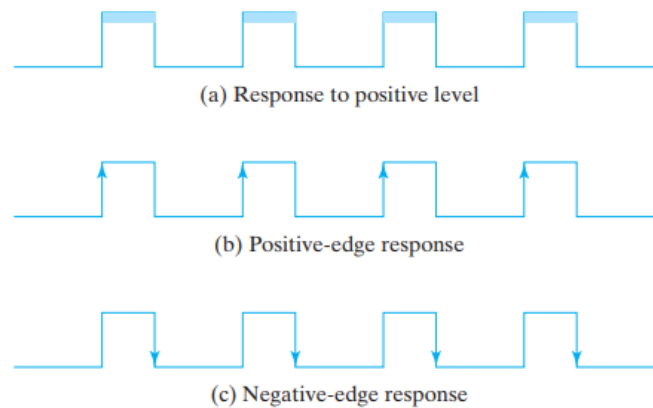
Flip-flops

The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a trigger, and the transition it causes is said to trigger the flip-flop. The D latch with pulses in its control input is essentially a flip-flop that is triggered every time the pulse goes to the logic-1 level. As long as the pulse input remains at this level, any changes in the data input will change the output and the state of the latch.

When latches are used for the storage elements, a serious difficulty arises. The state transitions of the latches start as soon as the clock pulse changes to the logic-1 level. The new state of a latch appears at the output while the pulse is still active. This output is connected to the inputs of the latches through the combinational circuit. If the inputs applied to the latches change while the clock pulse is still at the logic-1 level, the latches will respond to new values and a new output state may occur. The result is an unpredictable situation, since the state of the latches may keep changing for as long as the clock pulse stays at the active level. Because of this unreliable operation, the output of a latch cannot be applied directly or through combinational logic to the input of the same or another latch when all the latches are triggered by a common clock source.

Flip-flop circuits are constructed in such a way as to make them operate properly when they are part of a sequential circuit that employs a common clock. The problem with the latch is that it responds to a change in the level of a clock pulse. As shown in Fig. (a), a positive level response in the enable input allows changes in the output when the D input changes while the clock pulse stays at logic 1. The key to the proper operation of a flip-flop is to trigger it only during a signal transition. This can be accomplished by eliminating the feedback path that is inherent in the operation of the sequential circuit using latches. A clock pulse goes through two transitions: from 0 to 1 and the return from 1 to 0. As shown in Fig. below, the positive transition is defined as the positive edge and the negative transition as the negative edge. There are two ways that a latch can be modified to form a flip-flop. One way is to employ two latches in a special configuration that isolates the output of the flip-flop and prevents it from being affected while the input to the flip-flop is changing.

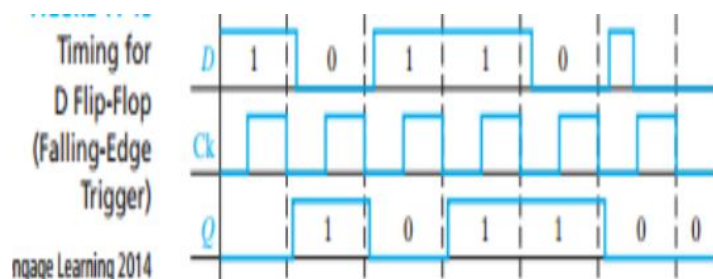
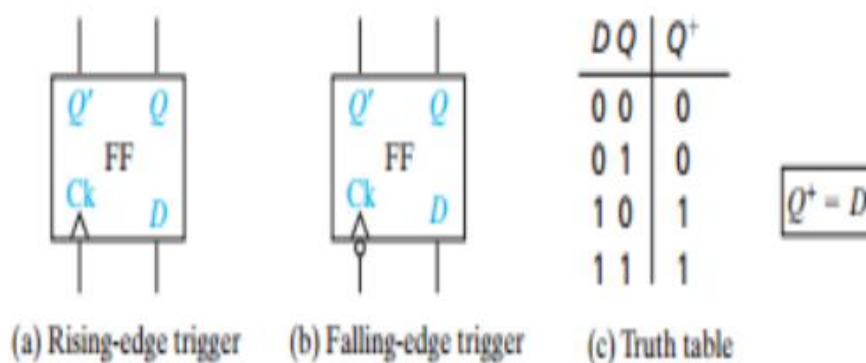
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Another way is to produce a flip-flop that triggers only during a signal transition (from 0 to 1 or from 1 to 0) of the synchronizing signal (clock) and is disabled during the rest of the clock pulse. We will now proceed to show the implementation of both types of flip-flops.

Edge-Triggered D Flip Flop

A D flip-flop has two inputs, D (data) and Ck (clock). The small arrowhead on the flip-flop symbol identifies the clock input. Unlike the D latch, the flip-flop output changes only in response to the clock, not to a change in D. If the output can change in response to a 0 to 1 transition on the clock input, we say that the flip-flop is triggered on the rising edge (or positive edge) of the clock. If the output can change in response to a 1 to 0 transition on the clock input, we say that the flip-flop is triggered on the falling edge (or negative edge) of the clock. An inversion bubble on the clock input indicates a falling-edge trigger (Figure (b)), and no bubble indicates a rising-edge trigger (Figure (a)). The term active edge refers to the clock edge (rising or falling) that triggers the flip-flop state change.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

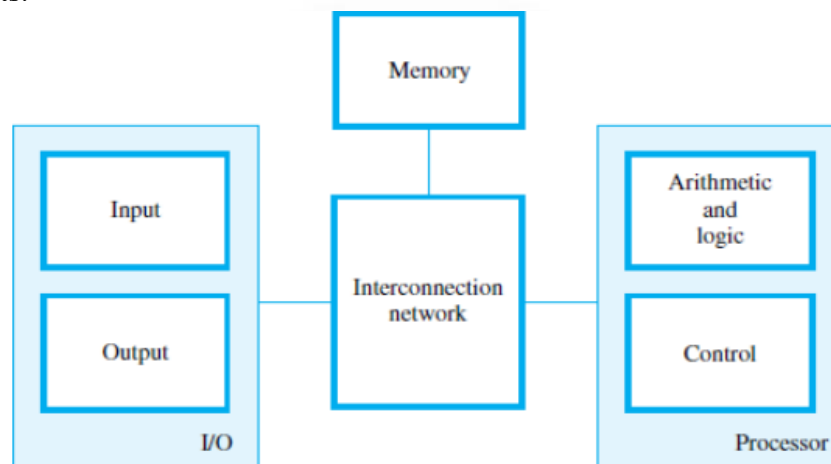
Module-3

Chapter 1: Basic Structure of Computers

3.1 Functional Units

A computer consists of 5 functionally independent main parts:

- 1) Input
- 2) Memory
- 3) ALU
- 4) Output &
- 5) Control units.



Functional unit of computer

3.2 Basic Operational Concepts

An Instruction consists of 2 parts, 1) Operation code (Opcode) and 2) Operands.

The data/operands are stored in memory. The individual instruction is brought from the memory to the processor. Then, the processor performs the specified operation.

Let us see a typical instruction ADD LOCA, R0

This instruction is an addition operation. The following are the steps to execute the instruction:

Step 1: Fetch the instruction from main-memory into the processor.

Step 2: Fetch the operand at location LOCA from main-memory into the processor.

Step 3: Add the memory operand (i.e. fetched contents of LOCA) to the contents of register R0.

Step 4: Store the result (sum) in R0.

The same instruction can be realized using 2 instructions as: Load LOCA, R1

Add R1, R0

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The following are the steps to execute the instruction:

Step 1: Fetch the instruction from main memory into the processor.

Step 2: Fetch the operand at location LOCA from main memory into the register R1.

Step 3: Add the content of Register R1 and the contents of register R0.

Step 4: Store the result (sum) in R0.

Main Parts of Processor

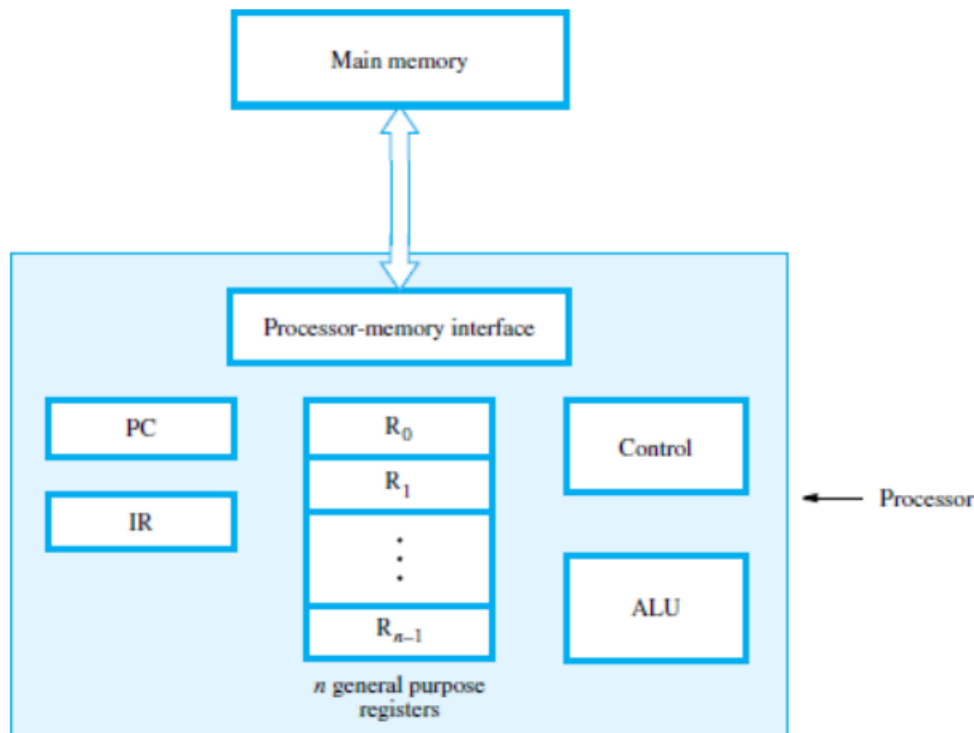
The processor contains ALU, control-circuitry and many registers. The processor contains 'n' general-purpose registers R0 through Rn-1. The IR holds the instruction that is currently being executed. The control-unit generates the timing-signals that determine when a given action is to take place. The PC contains the memory-address of the next-instruction to be fetched & executed. During the execution of an instruction, the contents of PC are updated to point to next instruction. The MAR holds the address of the memory-location to be accessed. The MDR contains the data to be written into or read out of the addressed location. MAR and MDR facilitates the communication with memory.

(Where, IR Instruction-Register, PC-Program Counter, MAR-Memory Address Register, MDR-Memory Data Register)

Steps To Execute an Instruction:

- 1) The address of first instruction (to be executed) gets loaded into PC.
- 2) The contents of PC (i.e. address) are transferred to the MAR & control-unit issues Read signal to memory.
- 3) After certain amount of elapsed time, the first instruction is read out of memory and placed into MDR.
- 4) Next, the contents of MDR are transferred to IR. At this point, the instruction can be decoded & executed.
- 5) To fetch an operand, it's address is placed into MAR & control-unit issues Read signal. As a result, the operand is transferred from memory into MDR, and then it is transferred from MDR to ALU.
- 6) Likewise required number of operands is fetched into processor.
- 7) Finally, ALU performs the desired operation.
- 8) If the result of this operation is to be stored in the memory, then the result is sent to the MDR.
- 9) The address of the location where the result is to be stored is sent to the MAR and a Write cycle is initiated.
- 10) At some point during execution, contents of PC are incremented to point to next instruction in the program.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

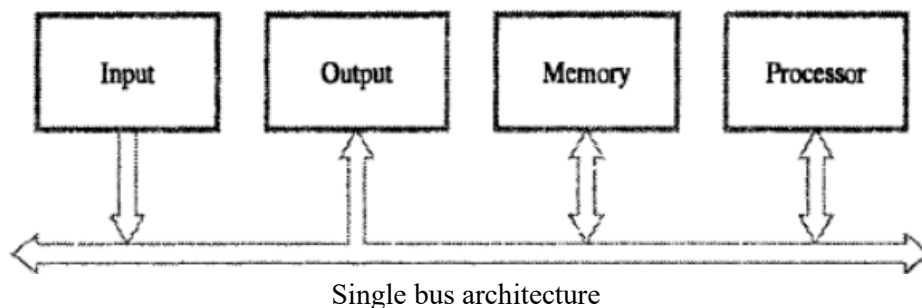


Connection between processor and the main memory

3.3 Bus structure

A bus is a group of lines that serves as a connecting path for several devices. A bus may be lines or wires. The lines carry data or address or control signal. There are 2 types of Bus structures:

1) Single Bus Structure: Because the bus can be used for only one transfer at a time, only 2 units can actively use the bus at any given time. Bus control lines are used to arbitrate multiple requests for use of the bus. Advantages: 1. Low cost & 2. Flexibility for attaching peripheral devices.



2) Multiple Bus Structure: Systems that contain multiple buses achieve more concurrency in operations. Two or more transfers can be carried out at the same time. Advantage: Better performance. Disadvantage: Increased cost.

The devices connected to a bus vary widely in their speed of operation. To synchronize their operational-speed, buffer-registers can be used. Buffer Registers are included with the devices

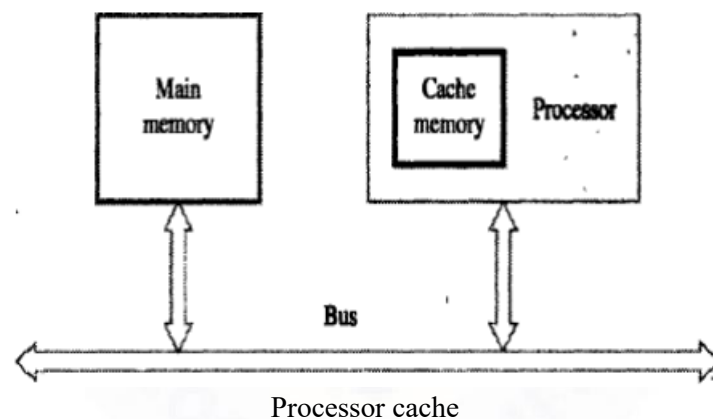
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

to hold the information during transfers. prevent a high-speed processor from being locked to a slow I/O device during data transfers.

3.4 Performance – Processor Clock

The most important measure of performance of a computer is how quickly it can execute programs. The speed of a computer is affected by the design of 1) Instruction-set. 2) Hardware & the technology in which the hardware is implemented. 3) Software including the operating system.

Because programs are usually written in an HLL, performance is also affected by the compiler that translates programs into machine language. (HLL High Level Language). For best performance, it is necessary to design the compiler, machine instruction set and hardware in a co-ordinated way.



Let us examine the flow of program instructions and data between the memory & the processor.

- At the start of execution, all program instructions are stored in the main memory.
- As execution proceeds, instructions are fetched into the processor, and a copy is placed in the cache.
- Later, if the same instruction is needed a second time, it is read directly from the cache.
- A program will be executed faster if movement of instruction/data between the main-memory and the processor is minimized which is achieved by using the cache.

Processor circuits are controlled by a timing signal called a Clock. The clock defines regular time intervals called Clock Cycles. To execute a machine instruction, the processor divides the action to be performed into a sequence of basic steps such that each step can be completed in one clock cycle.

Let P = Length of one clock cycle R = Clock rate. Relation between P and R is given by $R = (1/P)$, R is measured in cycles per second. Cycles per second is also called Hertz (Hz).

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

3.5 Basic Performance Equation

Let T = Processor time required to execute a program.

N = Actual number of instruction executions.

S = Average number of basic steps needed to execute one machine instruction.

R = Clock rate in cycles per second.

The program execution time is given by $T = (N * S) / R$ ------(1)

Equ1 is referred to as the basic performance equation. To achieve high performance, the computer designer must reduce the value of T , which means reducing N and S , and increasing R .

The value of N is reduced if source program is compiled into fewer machine instructions. The value of S is reduced if instructions have a smaller number of basic steps to perform. The value of R can be increased by using a higher frequency clock. Care has to be taken while modifying values since changes in one parameter may affect the other.

3.6 Clock Rate

There are 2 possibilities for increasing the clock rate R :

1) Improving the IC technology makes logic-circuits faster. This reduces the time needed to compute a basic step. (IC integrated circuits). This allows the clock period P to be reduced and the clock rate R to be increased.

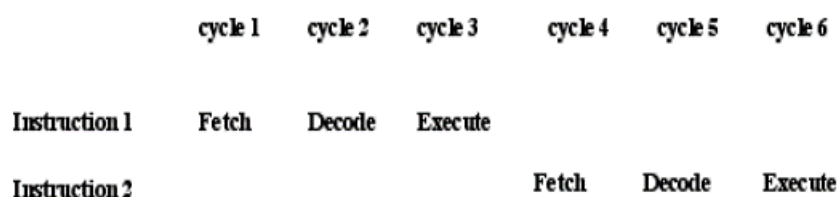
2) Reducing the amount of processing done in one basic step also reduces the clock period P . In presence of a cache, the percentage of accesses to the main memory is small. Hence, much of performance-gain expected from the use of faster technology can be realized. The value of T will be reduced by same factor as R is increased. S & N are not affected.

3.7 Pipelining & Superscalar Operation

Normally, the computer executes the instruction in sequence one by one. An improvement in performance can be achieved by overlapping the execution of successive instructions using a technique called Pipelining.

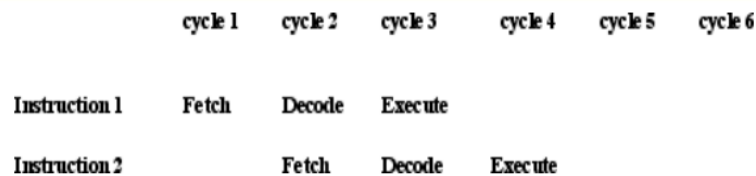
Consider the instruction Add $R1, R2, R3$;
Move $R4, R5$;

Let us assume that both operations take 3 clock cycles each for completion.



As shown in above figure, 6 clock cycles are required to complete two operations.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



As shown in above figure, if we use pipelining & pre-fetching, only 4 cycles are required to complete same two operations. While executing the Add instruction, the processor can read the Move instruction from memory. In the ideal case, if all instructions are overlapped to the maximum degree possible, execution proceeds at the rate of one instruction completed in each clock cycle.

A higher degree of concurrency can be achieved if multiple instruction pipelines are implemented in the processor i.e. multiple functional units can be used to execute different instructions parallelly. This mode of operation is known as Superscalar Execution. With Superscalar arrangement, it is possible to complete the execution of more than one instruction in every clock cycle.

3.8 Performance Measurement

Benchmark refers to standard task used to measure how well a processor operates. The Performance Measure is the time taken by a computer to execute a given benchmark. SPEC selects & publishes the standard programs along with their test results for different application domains. (SPEC System Performance Evaluation Corporation).

SPEC Rating is given by $\text{SPEC rating} = \frac{\text{Running time on the reference computer}}{\text{Running time on the computer under test}}$

SPEC rating = 50 The computer under test is 50 times as fast as reference-computer. The test is repeated for all the programs in the SPEC suite. Then, the geometric mean of the results is computed.

Let SPEC_i = Rating for program 'i' in the suite. Overall SPEC rating for the computer is given by,

$$\text{SPEC rating} = \frac{\text{running time on reference computer}}{\text{running time on computer undertest}}$$

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

where n = no. of programs in the suite.

3.9 Instruction Set: CISC an RISC

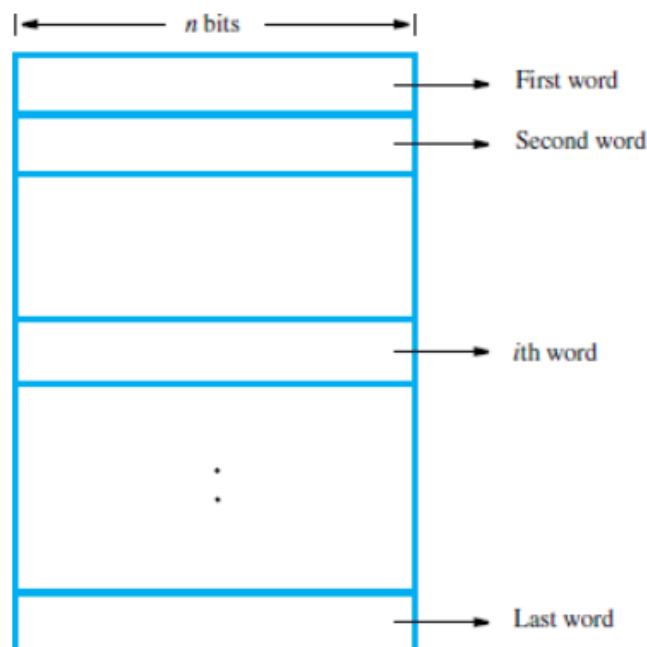
Simple instructions require a small number of basic steps to execute. Complex instructions involve a large number of steps. For a processor that has only simple instructions, a large number of instructions may be needed to perform a given programming task. This could lead to a large value for N and a small value for S. On the other hand, if individual instructions perform more complex operations, fewer instructions will be needed, leading to a lower value of N and a larger value of S. It is not obvious if one choice is better than the other.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

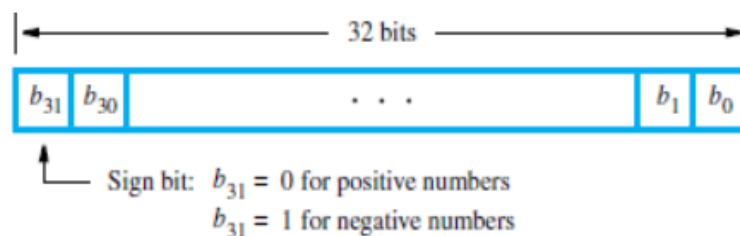
Chapter 2: Machine Instructions and Programs

3.10 Memory Location and Addresses

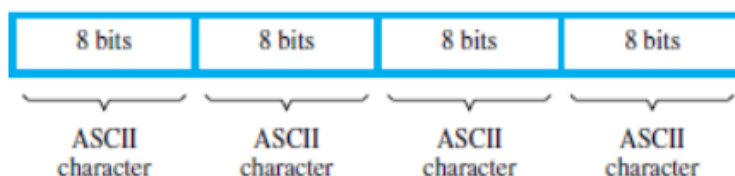
The memory consists of many millions of storage cells (flip-flops). Each cell can store a bit of information i.e., 0 or 1. Each group of n bits is referred to as a word of information, and n is called the word length. The word length can vary from 8 to 64 bits. A unit of 8 bits is called a byte. Accessing the memory to store or retrieve a single item of information (word/byte) requires distinct addresses for each item location. (It is customary to use numbers from 0 through $2^k - 1$ as the addresses of successive locations in the memory). If $2^k = \text{no. of addressable locations}$, then 2^k addresses constitute the address-space of the computer. For example, a 24-bit address generates an address-space of 2^{24} locations (16 MB).



Memory words



(a) A signed integer



(b) Four characters

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Byte-Addressability

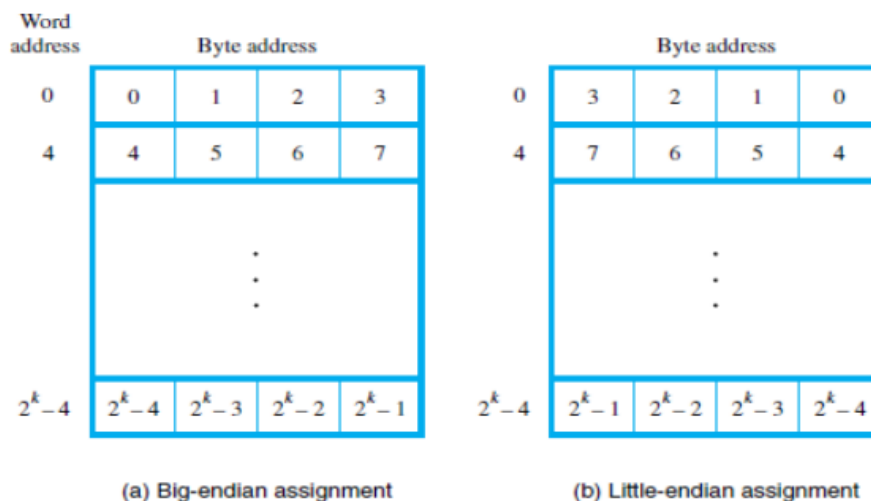
In byte-addressable memory, successive addresses refer to successive byte locations in the memory. Byte locations have addresses 0, 1, 2, ... If the word-length is 32 bits, successive words are located at addresses 0, 4, 8, . . . with each word having 4bytes.

Big-Endian & Little-Endian Assignments

There are two ways in which byte-addresses are arranged.

- 1) Big-Endian: Lower byte-addresses are used for the more significant bytes of the word.
- 2) Little-Endian: Lower byte-addresses are used for the less significant bytes of the word

In both cases, byte-addresses 0, 4, 8, ... are taken as the addresses of successive words in the memory. Consider a 32-bit integer (in hex): 0x12345678 which consists of 4 bytes: 12, 34, 56, and 78. Hence this integer will occupy 4 bytes in memory. Assume, we store it at memory address starting 1000.



Word Alignment

Words are said to be Aligned in memory if they begin at a byte-address that is a multiple of the number of bytes in a word. For example, If the word length is 16(2 bytes), aligned words begin at byte-addresses 0, 2, 4, ... If the word length is 64(2 bytes), aligned words begin at byte-addresses 0, 8, 16, ... Words are said to have Unaligned Addresses, if they begin at an arbitrary byte-address.

Accessing Numbers, Characters & Characters Strings

A number usually occupies one word. It can be accessed in the memory by specifying its word address. Similarly, individual characters can be accessed by their byte-address. There are two ways to indicate the length of the string:

- 1) A special control character with the meaning "end of string" can be used as the last character in the string.
- 2) A separate memory word location or register can contain a number indicating the length of the string in byte

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

3.11 Memory Operations

Two memory operations are: 1) Load (Read/Fetch) & 2) Store (Write).

The Load operation transfers a copy of the contents of a specific memory-location to the processor. The memory contents remain unchanged. Steps for Load operation:

- 1) Processor sends the address of the desired location to the memory.
- 2) Processor issues 'read' signal to memory to fetch the data.
- 3) Memory reads the data stored at that address.
- 4) Memory sends the read data to the processor.

The Store operation transfers the information from the register to the specified memory-location. This will destroy the original contents of that memory-location. Steps for Store operation is:

- 1) Processor sends the address of the memory-location where it wants to store data.
- 2) Processor issues 'write' signal to memory to store the data.
- 3) Content of register (MDR) is written into the specified memory-location.

3.12 Instruction and Instruction sequencing

A computer must have instructions capable of performing 4 types of operations:

- 1) Data transfers between the memory and the registers (MOV, PUSH, POP, XCHG).
- 2) Arithmetic and logic operations on data (ADD, SUB, MUL, DIV, AND, OR, NOT).
- 3) Program sequencing and control (CALL, RET, LOOP, INT).
- 4) I/O transfers (IN, OUT).

Register Transfer Notation (RTN)

The possible locations in which transfer of information occurs are: 1) Memory-location 2) Processor register & 3) Registers in I/O device.

Location	Hardware Binary Address	Example	Description
Memory	LOC, PLACE, NUM	$R1 \leftarrow [LOC]$	Contents of memory-location LOC are transferred into register R1.
Processor	R0, R1, R2	$[R3] \leftarrow [R1] + [R2]$	Add the contents of register R1 & R2 and places their sum into R3.
I/O Registers	DATAIN, DATAOUT	$R1 \leftarrow DATAIN$	Contents of I/O register DATAIN are transferred into register R1.

Assembly Language Notation

To represent machine instructions and programs, assembly language format is used.

Assembly Language Format	Description
Move LOC, R1	Transfer data from memory-location LOC to register R1. The contents of LOC are unchanged by the execution of this instruction, but the old contents of register R1 are overwritten.
Add R1, R2, R3	Add the contents of registers R1 and R2, and places their sum into register R3.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Basic Instruction Types

Instruction Type	Syntax	Example	Description	Instructions for Operation
Three Address	Opcode Source1,Source2, Destination	Add A,B,C	Add the contents of memory-locations A & B. Then, place the result into location C.	
Two Address	Opcode Source, Destination	Add A,B	Add the contents of memory-locations A & B. Then, place the result into location B, replacing the original contents of this location. Operand B is both a source and a destination.	Move B, C Add A, C
One Address	Opcode Source/Destination	Load A	Copy contents of memory-location A into accumulator.	Load A Add B Store C
		Add B	Add contents of memory-location B to contents of accumulator register & place sum back into accumulator.	
		Store C	Copy the contents of the accumulator into location C.	
Zero Address	Opcode [no Source/Destination]	Push	Locations of all operands are defined implicitly. The operands are stored in a pushdown stack.	Not possible

Access to data in the registers is much faster than to data stored in memory-locations.

Let R_i represent a general-purpose register. The instructions: Load A, R_i
Store R_i, A
Add A, R_i

are generalizations of the Load, Store and Add Instructions for the single-accumulator case, in which register R_i performs the function of the accumulator. In processors, where arithmetic operations as allowed only on operands that are in registers, the task $C \leftarrow [A] + [B]$ can be performed by the instruction sequence:

Move A, R_i
Move B, R_j
Add R_i, R_j
Move R_j, C

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

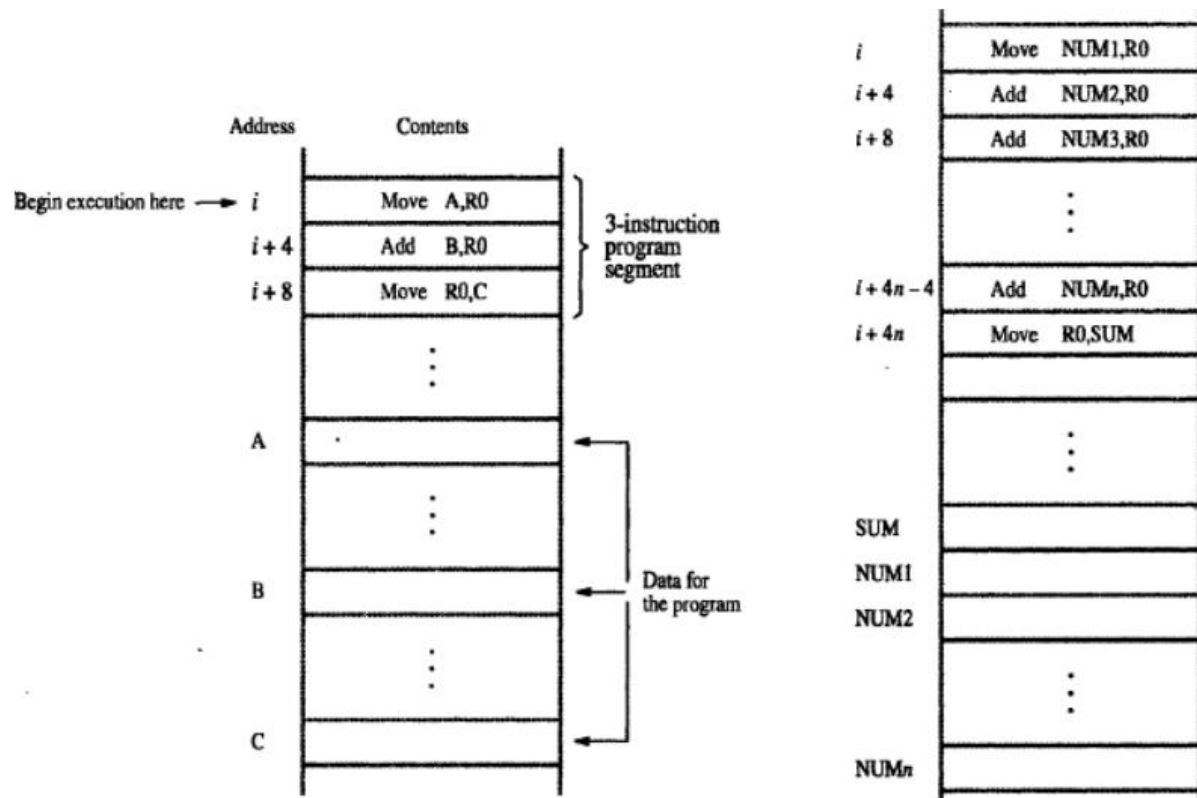
Instruction Execution & Straight-Line Sequencing

The program is executed as follows: 1) Initially, the address of the first instruction is loaded into PC.

2) Then, the processor control circuits use the information in the PC to fetch and execute instructions, one at a time, in the order of increasing addresses. This is called Straight-Line sequencing.

3) During the execution of each instruction, PC is incremented by 4 to point to next instruction. There are 2 phases for Instruction Execution:

- 1) Fetch Phase: The instruction is fetched from the memory-location and placed in the IR.
- 2) Execute Phase: The contents of IR are examined to determine which operation is to be performed. The specified operation is then performed by the processor.



Program Explanation

Consider the program for adding a list of n numbers.

The Address of the memory-locations containing the n numbers are symbolically given as NUM1, NUM2.....NUM n . Separate Add instruction is used to add each number to the contents of register R0. After all the numbers have been added, the result is placed in memory-location SUM.

Branching

Consider the task of adding a list of ' n ' numbers. Number of entries in the list ' n ' is stored in memory-location N. Register R1 is used as a counter to determine the number of times the loop is executed. Content-location N is loaded into register R1 at the beginning of the program. The

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Loop is a straight-line sequence of instructions executed as many times as needed. The loop starts at location LOOP and ends at the instruction Branch>0.

During each pass, address of the next list entry is determined, and that entry is fetched and added to R0. The instruction Decrement R1 reduces the contents of R1 by 1 each time through the loop. Then Branch Instruction loads a new value into the program counter. As a result, the processor fetches and executes the instruction at this new address called the Branch Target. A Conditional Branch Instruction causes a branch only if a specified condition is satisfied. If the condition is not satisfied, the PC is incremented in the normal way, and the next instruction in sequential address order is fetched and executed.

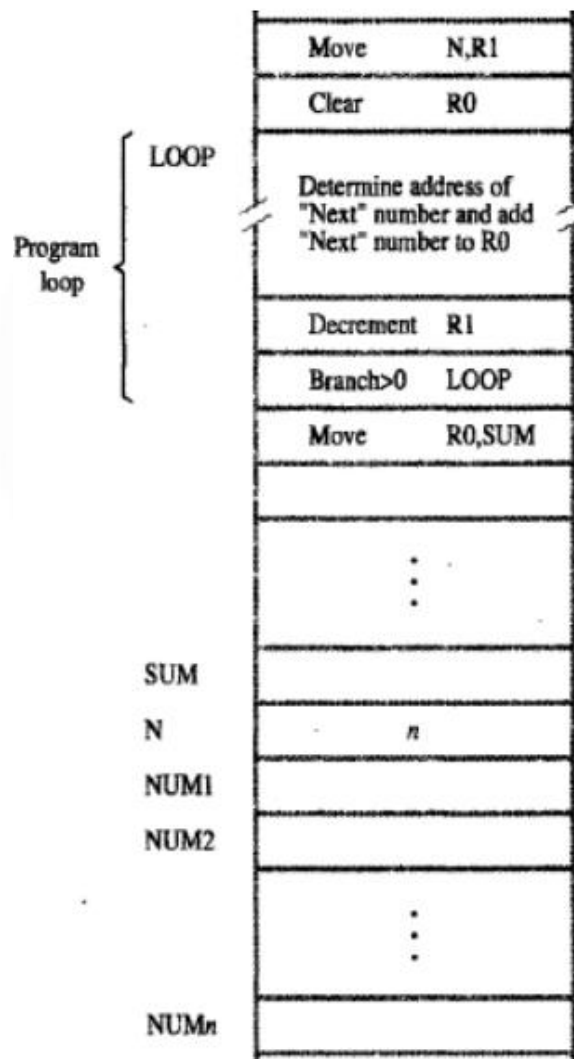


Fig: Loop to add n numbers.

Condition Codes

The processor keeps track of information about the results of various operations. This is accomplished by recording the required information in individual bits, called condition code flags. These flags are grouped together in a special processor-register called the condition code register (or statue register). Four commonly used flags are:

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

- 1) N (negative) set to 1 if the result is negative, otherwise cleared to 0.
- 2) Z (zero) set to 1 if the result is 0; otherwise, cleared to 0.
- 3) V (overflow) set to 1 if arithmetic overflow occurs; otherwise, cleared to 0.
- 4) C (carry) set to 1 if a carry-out results from the operation; otherwise cleared to 0.

3.13 Addressing Modes

The different ways in which the location of an operand is specified in an instruction are referred to as Addressing Modes.

Implementation of Variable and Constants

A variable is represented by allocating a memory-location to hold its value. Thus, the value can be changed as needed using appropriate instructions. There are 2 accessing modes to access the variables:

1. Register Mode: The operand is the contents of a register. The name (or address) of the register is given in the instruction. Registers are used as temporary storage locations where the data in a register are accessed. For example, the instruction Move R1, R2; Copy content of register R1 into register R2.
2. Absolute (Direct) Mode: The operand is in a memory-location. The address of memory-location is given explicitly in the instruction. The absolute mode can represent global variables in the program. For example, the instruction Move LOC, R2; Copy content of memory-location LOC into register R2.

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

Immediate Mode: The operand is given explicitly in the instruction. For example, the instruction Move #200, R0; Place the value 200 in register R0. Clearly, the immediate mode is only used to specify the value of a source-operand.

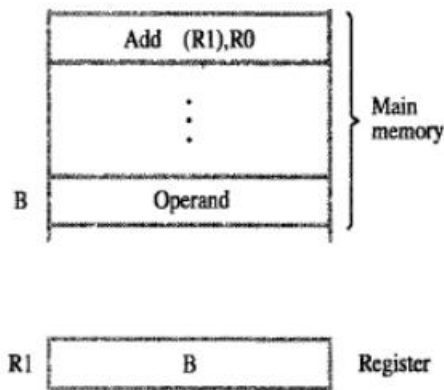
Indirection And Pointers

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

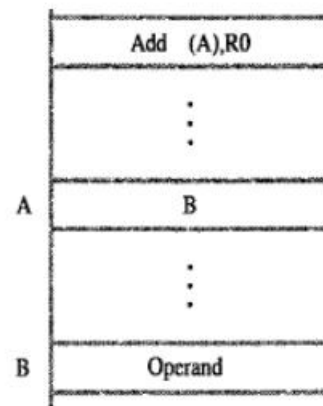
Indirect Mode: The EA of the operand is the contents of a register (or memory-location). The register (or memory-location) that contains the address of an operand is called a Pointer.

We denote the indirection by $\rightarrow$ name of the register or $\rightarrow$ new address given in the instruction. E.g.: Add (R1), R0; The operand is in memory. Register R1 gives the effective address (B) of the operand. The data is read from location B and added to contents of register R0.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



(a) Through a general-purpose register



(b) Through a memory location

To execute the Add instruction in fig (a), the processor uses the value which is in register R1, as the EA of the operand. It requests a read operation from the memory to read the contents of location B. The value read is the desired operand, which the processor adds to the contents of register R0. Indirect addressing through a memory-location is also possible as shown in fig (b). In this case, the processor first reads the contents of memory-location A, then requests a second read operation using the value B as an address to obtain the operand.

Address	Contents	
	Move N,R1	} Initialization
	Move #NUM1,R2	
	Clear R0	
→ LOOP	Add (R2),R0	
	Add #4,R2	
	Decrement R1	
	Branch>0 LOOP	
	Move R0,SUM	

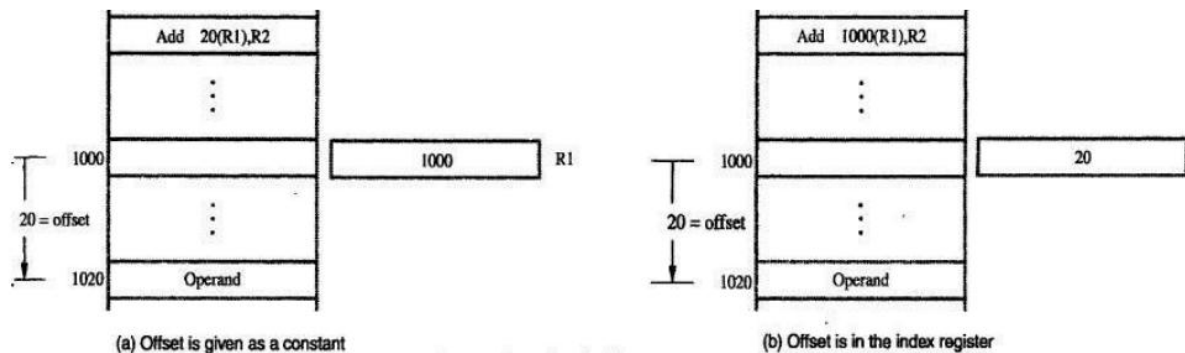
Explanation: In above program, Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectly through R2. The initialization-section of the program loads the counter-value n from memory-location N into R1 and uses the immediate addressing-mode to place the address value NUM1, which is the address of the first number in the list, into R2. Then it clears R0 to 0. The first two instructions in the loop implement the unspecified instruction block starting at LOOP. The first time through the loop, the instruction Add (R2), R0 fetches the operand at location NUM1 and adds it to R0. The second Add instruction adds 4 to the contents of the pointer R2, so that it will contain the address value NUM2 when the above instruction is executed in the second pass through the loop.

Indexing And Arrays

A different kind of flexibility for accessing operands is useful in dealing with lists and arrays. Index mode: the operation is indicated as X(Ri) where X=the constant value which defines an offset (also called a displacement). Ri=the name of the index registers which contains address

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

of a new location. The effective address of the operand is given by $EA = X + [R_i]$. The contents of the index-register are not changed in the process of generating the effective-address. The constant X may be given either $\rightarrow$ as an explicit number or $\rightarrow$ as a symbolic name representing a numerical value.



Fig(a) illustrates two ways of using the Index mode. In fig(a), the index register, R_1 , contains the address of a memory-location, and the value X defines an offset (also called a displacement) from this address to the location where the operand is found.

To find EA of operand: Eg: Add 20(R_1), R_2

$$EA \Rightarrow 1000 + 20 = 1020$$

An alternative use is illustrated in fig(b). Here, the constant X corresponds to a memory address, and the contents of the index register define the offset to the operand. In either case, the effective address is the sum of two values; one is given explicitly in the instruction, and the other is stored in a register.

Base with Index Mode: Another version of the Index mode uses 2 registers which can be denoted as (R_i, R_j). Here, a second register may be used to contain the offset X . The second register is usually called the base register. The effective address of the operand is given by $EA = [R_i] + [R_j]$. This form of indexed addressing provides more flexibility in accessing operands because both components of the effective address can be changed.

Base with Index & Offset Mode: Another version of the Index mode uses 2 registers plus a constant, which can be denoted as $X(R_i, R_j)$. The effective address of the operand is given by $EA = X + [R_i] + [R_j]$. This added flexibility is useful in accessing multiple components inside each item in a record, where the beginning of an item is specified by the (R_i, R_j) part of the addressing-mode. In other words, this mode implements a 3-dimensional array.

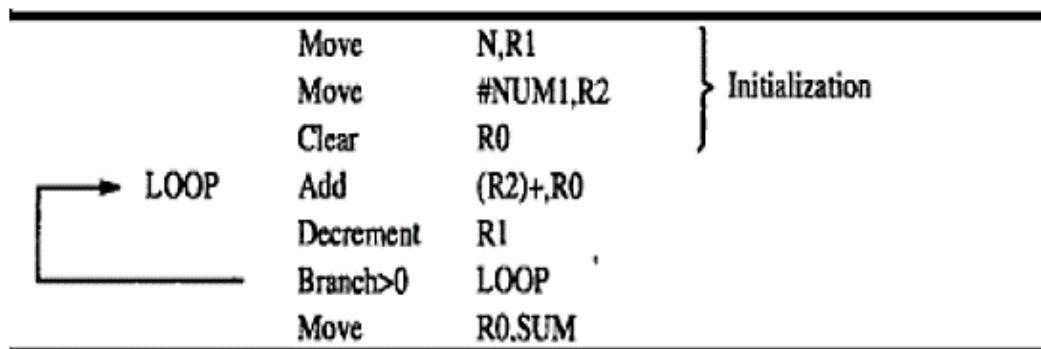
Relative Mode: This is similar to index-mode with one difference: The effective-address is determined using the PC in place of the general-purpose register R_i . The operation is indicated as $X(PC)$. $X(PC)$ denotes an effective address of the operand which is X locations above or below the current contents of PC. Since the addressed location is identified "relative" to the PC, the name Relative mode is associated with this type of addressing. This mode is used commonly in conditional branch instructions.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Additional Addressing Modes

1) Auto Increment Mode: Effective address of operand is contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list. Implicitly, the increment amount is 1. This mode is denoted as $(Ri)+$; where Ri =pointer-register.

2) Auto Decrement Mode: The contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand. This mode is denoted as $-(Ri)$; where Ri =pointer-register. These 2 modes can be used together to implement an important data structure called a stack.



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Module-4

Chapter 1: Input/Output Organization

Input/output architecture is an interface to the outside world that provide systematic means of controlling interaction with outside world. Input device such as keyboard, scanner, digital camera, etc. Output devices such as display, printer etc. Input/output devices cannot directly connect to the system bus because Data transfer rate of input/output devices is slower than (memory & processor) and verse versa. Input/output devices are used different data format and word length.

4.1 Accessing I/O Devices

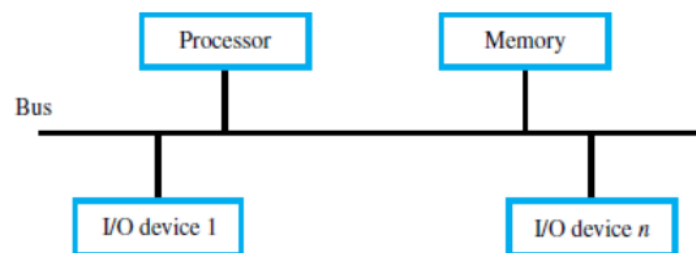
A single bus-structure can be used for connecting I/O-devices to a computer. Each I/O device is assigned a unique set of address. Bus consists of 3 sets of lines to carry address, data & control signals. Each IO devices are assigned with unique set of addresses. When processor places an address on address-lines, the intended-device responds to the command using control signal line. The processor requests either a read or write-operation. The requested-data are transferred over the data-lines.

There are 2 ways to deal with I/O-devices:

1) Memory-Mapped I/O: Memory and I/O-devices share a common address-space. Any data-transfer instruction (like Move, Load) can be used to exchange information.

For example, Move DATAIN, R0; This instruction sends the contents of location DATAIN to register R0. Here, DATAIN address of the input-buffer of the keyboard.

2) I/O-Mapped I/O: Memory and I/O address-spaces are different. A special instruction named IN and OUT are used for data-transfer. Ex: Processor in Intel family. Advantage of separate I/O space: I/O-devices deal with fewer address-lines.



Single bus structure

I/O Interface for an Input Device

1) Address Decoder: enables the device to recognize its address when this address appears on the address-lines.

2) Status Register: contains information relevant to operation of I/O-device.

3) Data Register: holds data being transferred to or from processor. There are 2 types:

i) DATAIN Input-buffer associated with keyboard.

ii) DATAOUT Output data buffer of a display/printer. Both data and status register are connected to the data bus and assigned unique address. The address decoder, the data and status

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

register and the control circuitry required to coordinate I/O transfers constitute the device's interface circuit.

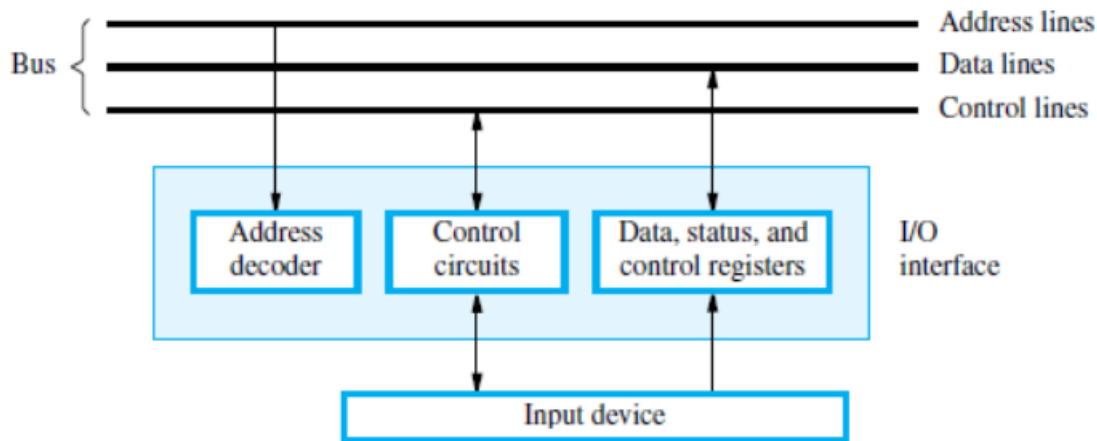


Figure 7.2 I/O interface for an input device.

Mechanisms Used for Interfacing I/O-Devices

- 1) Program Controlled I/O: Processor repeatedly checks status-flag to achieve required synchronization b/w processor & I/O device. Main drawback: The processor wastes time in checking status of device before actual data-transfer takes place.
- 2) Interrupt I/O: I/O-device initiates the action instead of the processor. I/O-device sends an INTR signal over bus whenever it is ready for a data-transfer operation. Like this, required synchronization is done between processor & I/O device.
- 3) Direct Memory Access (DMA): Device-interface transfer data directly to/from the memory w/o continuous involvement by the processor. DMA is a technique used for high speed I/O-device.

4.2 Interrupts

There are many situations where other tasks can be performed while waiting for an I/O device to become ready. A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready. Interrupt-signal is sent on the interrupt-request line. The processor can be performing its own task without the need to continuously check the I/O-device. The routine executed in response to an interrupt-request is called ISR. The processor must inform the device that its request has been recognized by sending INTA signal. (INTR Interrupt Request, INTA Interrupt Acknowledge, ISR Interrupt Service Routine)

For example, consider COMPUTE and PRINT routines. Assume that an interrupt request arrives during execution of instruction i in Fig below.

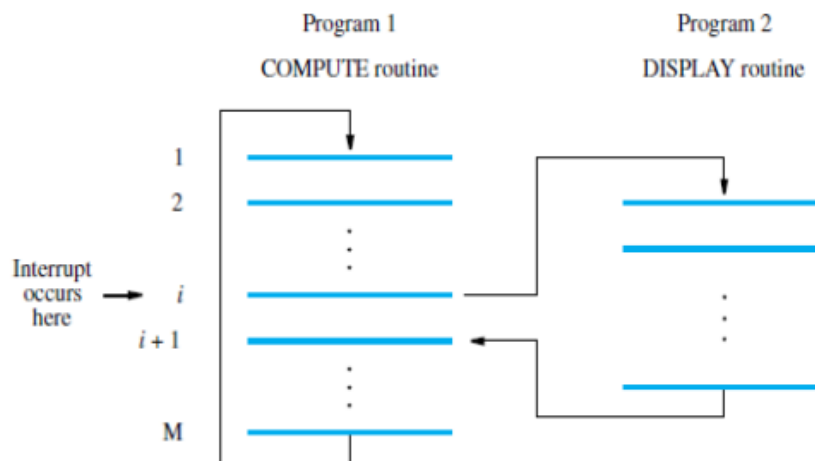
The processor first completes the execution of instruction i. Then, processor loads the PC with the address of the first instruction of the ISR (interrupt service routine). After the execution of ISR, the processor must come back to instruction i+1. Therefore, when an interrupt occurs, the current content of PC is put in temporary storage location. A return at the end of ISR reloads

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

the PC from that temporary storage location. This causes the execution to resume at instruction $i+1$.

When processor is handling interrupts, it must inform device that its request has been recognized. This may be accomplished by special control signal on bus. An interrupt-acknowledge signal is used. The task of saving and restoring the information can be done automatically by the processor or by program instruction. Saving and restoring registers involves memory transfers: Increases the total execution time. Increases the delay between the time an interrupt request is received, and the start of execution of the interrupt-service routine. This delay is called interrupt latency.

To reduce the interrupt latency, most processors save only the minimal amount of information. This minimal amount of information includes Program Counter and processor status registers. Any additional information that must be saved, must be saved explicitly by the program instructions at the beginning of the interrupt service routine. Interrupt Latency is a delay between time an interrupt-request is received and start of the execution of the ISR.



Transfer of control through the use of interrupts

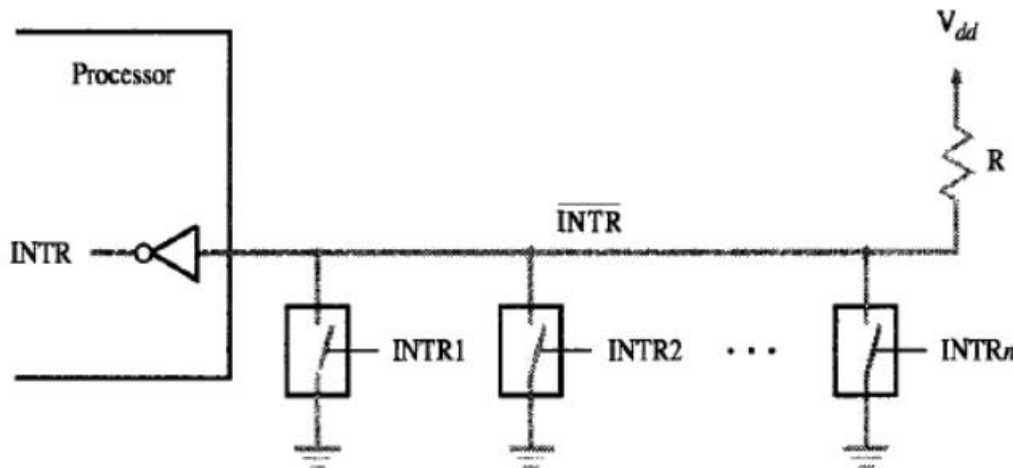
4.3 Interrupt Hardware

Most computers have several I/O devices that can request an interrupt. A single interrupt-request (IR) line may be used to serve n devices. All devices are connected to IR line via switches to ground. To request an interrupt, a device closes its associated switch. Thus, if all IR signals are inactive, the voltage on the IR line will be equal to V_{dd} . When a device requests an interrupt, the voltage on the line drops to 0. This causes the INTR received by the processor to go to 1. The value of INTR is the logical OR of the requests from individual devices.

$$\text{INTR} = \text{INTR}_1 + \text{INTR}_2 + \dots + \text{INTR}_n$$

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

A special gate known as open-collector or open-drain are used to drive the INTR line. The Output of the open collector control is equal to a switch to the ground that is open when gates input is in "0" state and closed when the gates input is in "1" state. Resistor R is called a Pull-up Resistor because it pulls the line voltage up to the high-voltage state when the switches are open.



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

4.4 Enabling and Disabling Interrupts

All computers fundamentally should be able to enable and disable interruptions as desired. Processors generally provide the ability to enable and disable such interruptions as desired. The problem of infinite loop occurs due to successive interruptions of active INTR signals. There are 3 mechanisms to solve problem of infinite loop:

- 1) Processor should ignore the interrupts until execution of first instruction of the ISR by the processor hardware. By using an Interrupt-disable instruction as the first instruction in the interrupt-service routine, the programmer can ensure that no further interruptions will occur until an Interrupt-enable instruction is executed. Typically, the Interrupt-enable instruction will be the last instruction in the interrupt-service routine before the Return from-interrupt instruction.
- 2) Processor should automatically disable interrupts before starting the execution of the ISR. The second option, which is suitable for a simple processor with only one interrupt request line, is to have the processor automatically disable interrupts before starting the execution of the interrupt-service routine. After saving the contents of the PC and the processor status register (PS) on the stack, the processor performs the equivalent of executing an Interrupt-disable instruction. It is often the case that one bit in the PS register, called Interrupt-enable, indicates whether interrupts are enabled.
- 3) In the third option, the processor has a special interrupt-request line for which the interrupt handling circuit responds only to the leading edge of the signal. Such a line is said to be edge triggered.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Sequence of events involved in handling an interrupt-request:

- 1) The device raises an interrupt-request.
- 2) The processor interrupts the program currently being executed.
- 3) Interrupts are disabled by changing the control bits in the processor status register (PS).
- 4) The device is informed that its request has been recognized. In response, the device deactivates the interrupt-request signal.
- 5) The action requested by the interrupt is performed by the interrupt-service routine.
- 6) Interrupts are enabled, and execution of the interrupted program is resumed.

4.5 Handling Multiple Devices

While handling multiple devices, the issues concerned are:

- 1) How can the processor recognize the device requesting an interrupt?
- 2) How can the processor obtain the starting address of the appropriate ISR?
- 3) Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- 4) How should 2 or more simultaneous interrupt-requests be handled?

When a request is received over the common interrupt-request line, additional information is needed to identify the device that activated the line. The information needed to determine whether a device is requesting an interrupt is available in its status register. When a device raises an interrupt request, it sets to 1 one of the bits in its status register, which we will call the IRQ bit. The two methods to handle multiple devices are: Polling and Vectored Interrupts

Polling

Information needed to determine whether device is requesting interrupt is available in status-register. Following condition-codes are used:

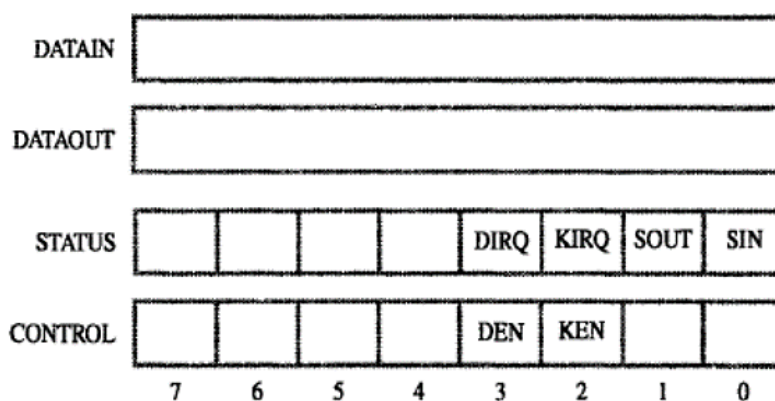
DIRQ Interrupt-request for display.

KIRQ Interrupt-request for keyboard.

KEN keyboard enable.

DEN Display Enable.

SIN, SOUT status flags.



Registers in keyboard and display interfaces

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

For an input device, SIN status flag is used. $SIN = 1$ when a character is entered at the keyboard. $SIN = 0$ when the character is read by processor. $IRQ = 1$ when a device raises an interrupt-requests. Simplest way to identify interrupting-device is to have ISR poll all devices connected to bus. The first device encountered with its IRQ bit set is serviced. After servicing first device, next requests may be serviced. Advantage: Simple & easy to implement. Disadvantage: More time spent polling IRQ bits of all devices.

Vectored Interrupts

A device requesting an interrupt identifies itself by sending a special code to processor over bus. Then, the processor starts executing the ISR. The special-code indicates starting-address of ISR. The special-code length ranges from 4 to 8 bits. The location pointed to by the interrupting-device is used to store the starting address to ISR. The starting address to ISR is called the interrupt vector. Processor loads interrupt-vector into PC & executes appropriate ISR. When processor is ready to receive interrupt-vector code, it activates INTA line. Then, I/O-device responds by sending its interrupt-vector code & turning off the INTR signal. The interrupt vector also includes a new value for the Processor Status Register.

Interrupt Nesting

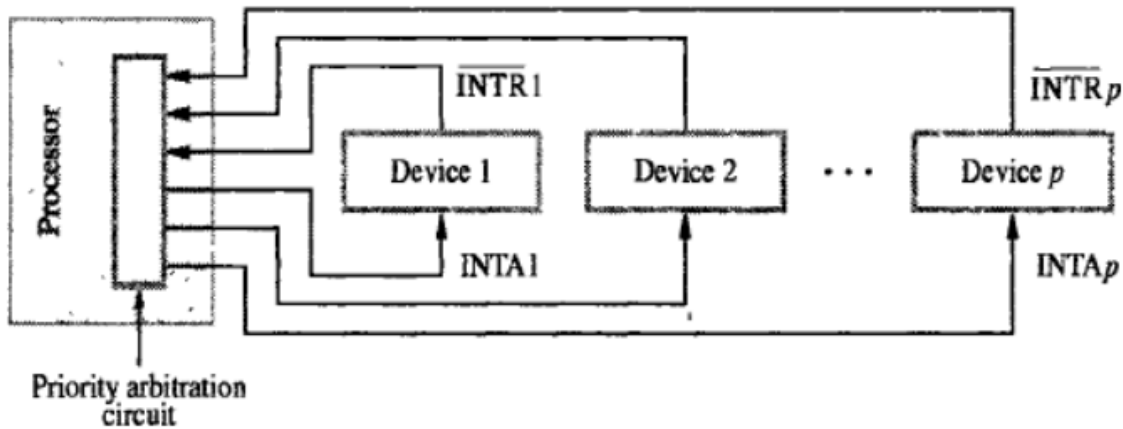
Interrupts should be disabled during the execution of an interrupt-service routine, to ensure that a request from one device will not cause more than one interruption. The same arrangement is often used when several devices are involved, in which case execution of a given interrupt-service routine, once started, always continues to completion before the processor accepts an interrupt request from a second device.

Interrupt-service routines are typically short, and the delay they may cause is acceptable for most simple devices. A multiple-priority scheme is implemented by using separate INTR & INTA lines for each device. Each INTR line is assigned a different priority-level. Priority-level of processor is the priority of program that is currently being executed. Processor accepts interrupts only from devices that have higher priority than its own. At the time of execution of ISR for some device, priority of processor is raised to that of the device. Thus, interrupts from devices at the same level of priority or lower are disabled.

Privileged Instruction: Processor's priority is encoded in a few bits of PS word. (PS Processor-Status). Encoded-bits can be changed by Privileged Instructions that write into PS. Privileged-instructions can be executed only while processor is running in Supervisor Mode. Processor is in supervisor-mode only when executing operating-system routines.

Privileged Exception: User program cannot accidentally or intentionally change the priority of the processor & disrupt the system-operation. An attempt to execute a privileged-instruction while in user-mode leads to a Privileged Exception.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Implementation of interrupt priority using individual interrupt-request and acknowledge lines

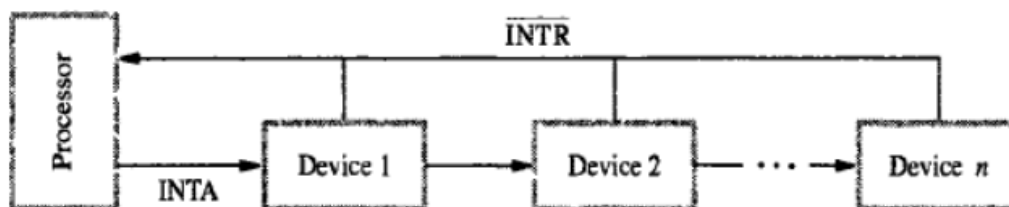
Simultaneous Requests

The processor must have some mechanisms to decide which request to service when simultaneous requests arrive. A widely used scheme is to connect the devices to form a daisy chain, as shown in fig. a. INTR line is common to all devices. The interrupt-acknowledge line, INTA, is connected in a daisy-chain fashion, such that the INTA signal propagates serially through the devices. INTA signal propagates serially through devices.

When several devices raise an interrupt-request, INTR line is activated. Processor responds by setting INTA line to 1. This signal is received by device 1. Device-1 passes signal on to device 2 only if it does not require any service. If device-1 has a pending-request for interrupt, the device-1 blocks INTA signal & proceeds to put its identifying-code on data-lines. Device that is electrically closest to processor has highest priority.

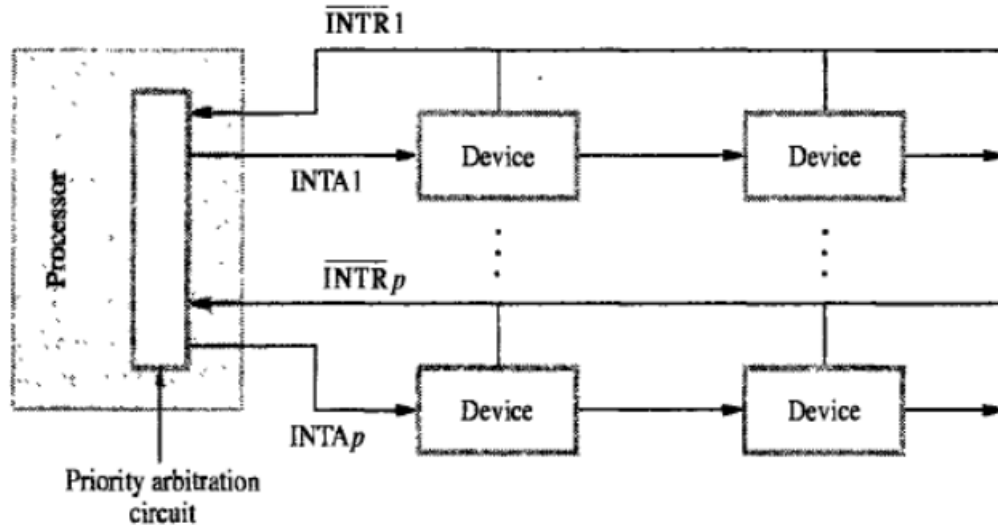
Advantage: It requires fewer wires than the individual connections.

Arrangement of Priority Groups: Here, the devices are organized in groups & each group is connected at a different priority level. Within a group, devices are connected in a daisy chain.



(a) Daisy chain

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

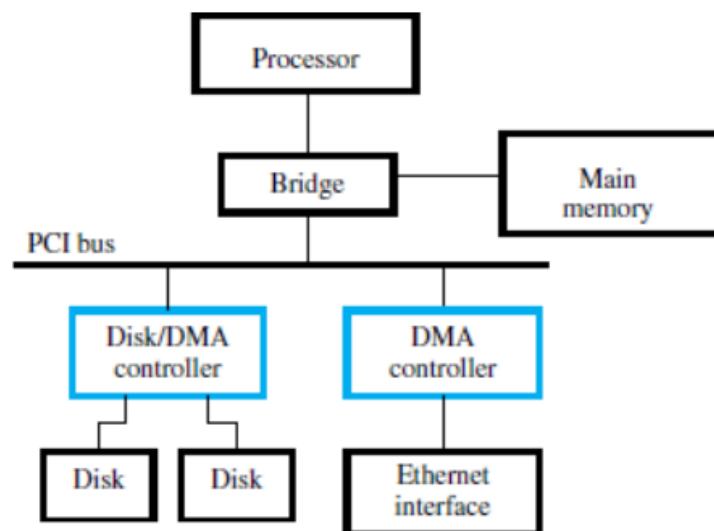


(b) Arrangement of priority groups

Interrupt priority schemes

4.6 Direct Memory Access

The transfer of a block of data directly b/w an external device & main-memory w/o continuous involvement by processor is called DMA. DMA controller is a control circuit that performs DMA transfers. it is a part of the I/O device interface, performs the functions that would normally be carried out by processor. While a DMA transfer is taking place, the processor can be used to execute another program.



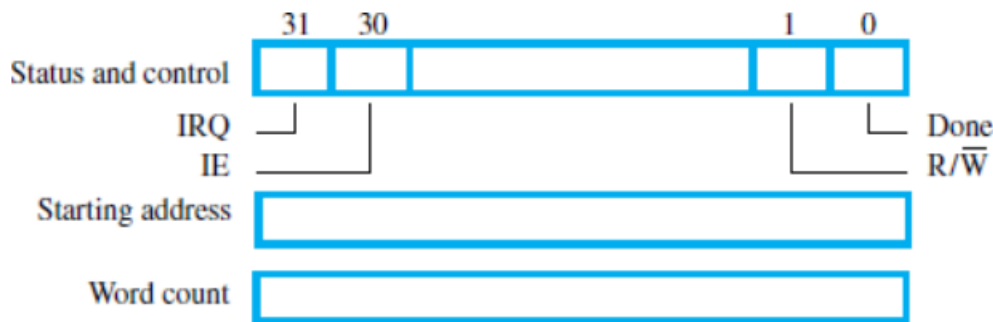
Use of DMA controllers in a computer system

DMA interface has three registers:

- 1) First register is used for storing starting address.
- 2) Second register is used for storing word-count.
- 3) Third register contains status- & control-flags.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The R/W bit determines direction of transfer. If R/W=1, controller performs a read-operation (i.e., it transfers data from memory to I/O), Otherwise, controller performs a write-operation (i.e., it transfers data from I/O to memory). If Done=1, the controller has completed transferring a block of data and is ready to receive another command. (IE Interrupt Enable). If IE=1, controller raises an interrupt after it has completed transferring a block of data. If IRQ=1, controller requests an interrupt. Requests by DMA devices for using the bus are always given higher priority than processor requests.



Typical registers in a DMA controller

There are 2 ways in which the DMA operation can be carried out:

- 1) Processor originates most memory-access cycles. DMA controller is said to "steal" memory cycles from processor. Hence, this technique is usually called Cycle Stealing.
- 2) DMA controller is given exclusive access to main memory to transfer a block of data without any interruption. This is known as Block Mode (or burst mode).

Bus Arbitration

The device that is allowed to initiate data-transfers on bus at any given time is called bus-master. There can be only one bus-master at any given time. Bus Arbitration is the process by which next device to become the bus-master is selected & bus-mastership is transferred to that device.

The two approaches are: 1) Centralized Arbitration: A single bus-arbiter performs the required arbitration. 2) Distributed Arbitration: All devices participate in selection of next bus-master.

A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve this, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers. The bus arbiter may be the processor, or a separate unit connected to the bus.

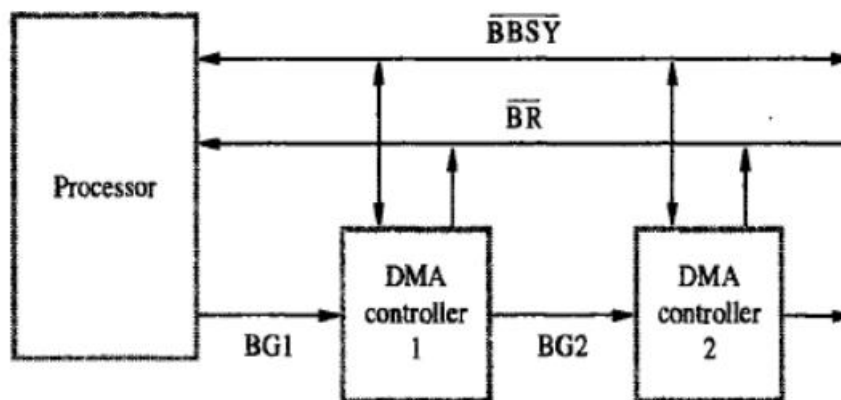
Centralized Arbitration

A single bus-arbiter performs the required arbitration. Normally, processor is the bus-master. Processor may grant bus-mastership to one of the DMA controllers. A DMA controller indicates that it needs to become bus-master by activating BR line. The signal on the BR line is the

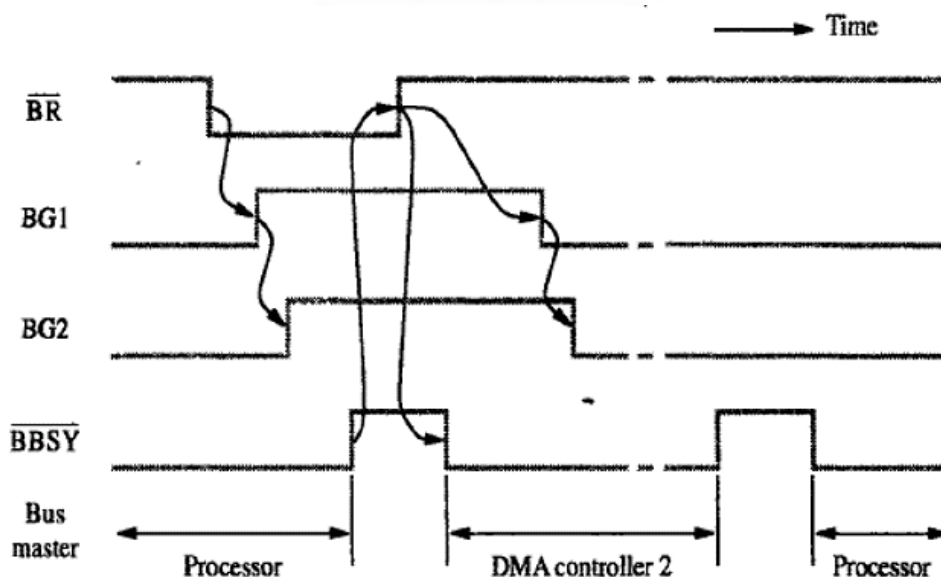
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

logical OR of bus-requests from all devices connected to it. Then, processor activates BG1 signal indicating to DMA controllers to use bus when it becomes free. BG1 signal is connected to all DMA controllers using a daisy-chain arrangement.

If DMA controller-1 is requesting the bus, Then, DMA controller-1 blocks propagation of grant-signal to other devices. Otherwise, DMA controller-1 passes the grant downstream by asserting BG2. Current bus-master indicates to all devices that it is using bus by activating BBSY line. The bus-arbiter is used to coordinate the activities of all devices requesting memory transfers. Arbiter ensures that only 1 request is granted at any given time according to a priority scheme. (BR Bus-Request, BG Bus-Grant, BBSY Bus Busy). The timing diagram shows the sequence of events for the devices connected to the processor. DMAcontroller-2 requests and acquires bus-mastership and later releases the bus. After DMA controller-2 releases the bus, the processor resumes bus-mastership.



A simple arrangement for bus arbitration using a daisy chain



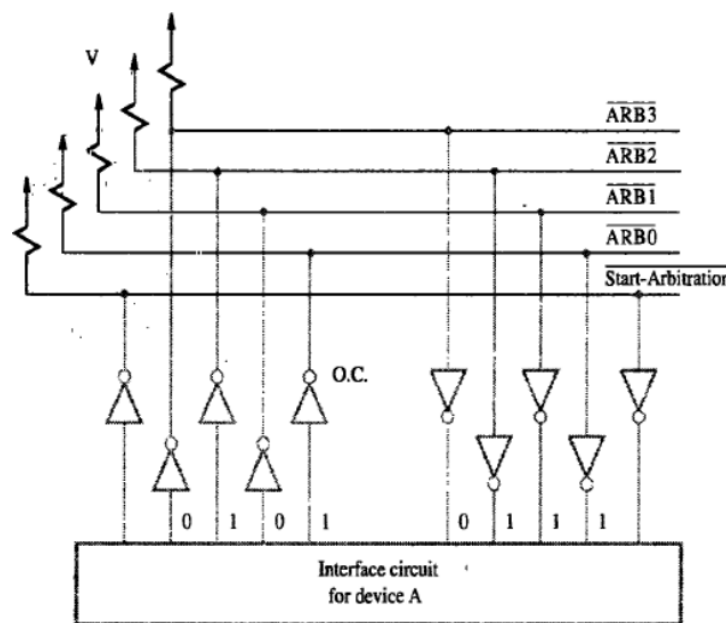
Sequence of signals during transfer of bus master ship for the devices in previous figure.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Distributed Arbitration

All devices participate in the selection of next bus-master. Each device on bus is assigned a 4-bit identification number (ID). When 1 or more devices request bus, they assert Start-Arbitration signal & place their 4-bit ID numbers on four open-collector lines ARB 0 through ARB 3. A winner is selected as a result of interaction among signals transmitted over these lines. Net-outcome is that the code on 4 lines represents request that has the highest ID number.

Advantage: This approach offers higher reliability since operation of bus is not dependent on any single device.



Distributed arbitration scheme

For E.g.: Assume 2 devices A & B have their ID 5 (0101), 6 (0110) and their code is 0111. Each device compares the pattern on the arbitration line to its own ID starting from MSB. If the device detects a difference at any bit position, it disables the drivers at that bit position. Driver is disabled by placing "0" at the input of the driver.

In e.g. "A" detects a difference in line ARB1, hence it disables the drivers on lines ARB1 & ARB0. This causes pattern on arbitration-line to change to 0110. This means that "B" has won contention.

4.7 Buses

The processor, main memory, and I/O devices can be interconnected by means of a common bus whose primary function is to provide a communications path for the transfer of data. The bus includes the lines needed to support interrupts and arbitration.

A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on. After describing bus protocols, we will present examples of interface circuits that use these protocols.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The bus lines used for transferring data may be grouped into three types: data, address, and control lines. The control signals specify whether a read or a write operation is to be performed. Usually, a single $R/\bar{W}$ line is used. It specifies Read when set to 1 and write when set to 0. When several operand sizes are possible, such as byte, word, or long word, the required size of data is indicated.

The bus control signals also carry timing information. They specify the times at which the processor and the I/O devices may place data on the bus or receive data from the bus. A variety of schemes have been devised for the timing of data transfers over a bus. These can be broadly classified as either synchronous or asynchronous schemes.

In any data transfer operation, one device plays the role of a master. This is the device that initiates data transfers by issuing read or write commands on the bus; hence, it may be called an initiator. Normally, the processor acts as the master, but other devices with DMA capability may also become bus masters. The device addressed by the master is referred to as a slave or target.

1. Synchronous Bus

In a synchronous bus, all devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals. In the simplest form of a synchronous bus, each of these intervals constitutes a bus cycle during which one data transfer can take place. Such a scheme is illustrated in Figure below. The address and data lines in this and subsequent figures are shown as high and low at the same time. This is a common convention indicating that some lines are high and some low, depending on the particular address or data pattern being transmitted. The crossing points indicate the times at which these patterns change. A signal line in an indeterminate or high impedance state is represented by an intermediate level half-way between the low and high signal levels.

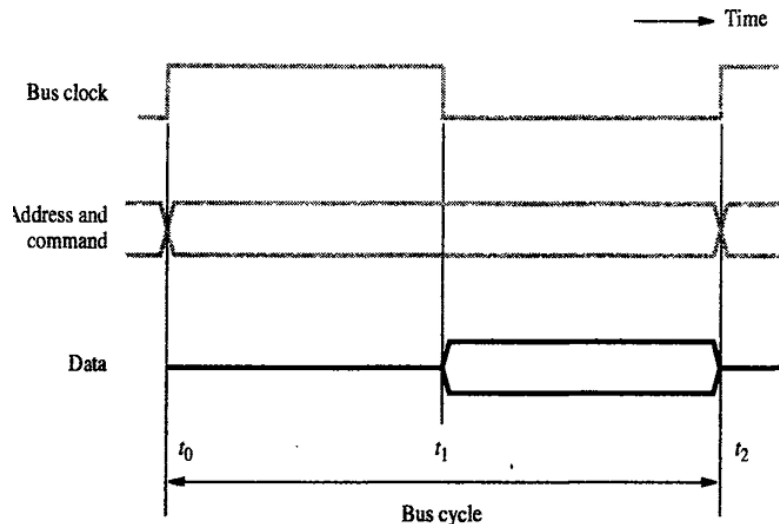
Let us consider the sequence of events during an input (read) operation. At time t_0 , the master places the device address on the address lines and sends an appropriate command on the control lines. In this case, the command will indicate an input operation and specify the length of the operand to be read, if necessary. Information travels over the bus at a speed determined by its physical and electrical characteristics. The clock pulse width, $t_1 - t_0$, must be longer than the maximum propagation delay between two devices connected to the bus. It also has to be long enough to allow all devices to decode the address and control signals so that the addressed device (the slave) can respond at time t_1 . It is important that slaves take no action or place any data on the bus before t_1 . The information on the bus is unreliable during the period t_0 to t_1 because signals are changing state. The addressed slave places the requested input data on the data lines at time t_1 .

At the end of the clock cycle, at time t_2 , the master strobes the data on the data lines into its input buffer. In this context, "strobe" means to capture the values of the data at a given instant and store them into a buffer. For data to be loaded correctly into any storage device, such as a register built with flip-flops, the data must be available at the input of that device for a period greater than the setup time of the device. Hence, the period $t_2 - t_1$ must be greater than the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

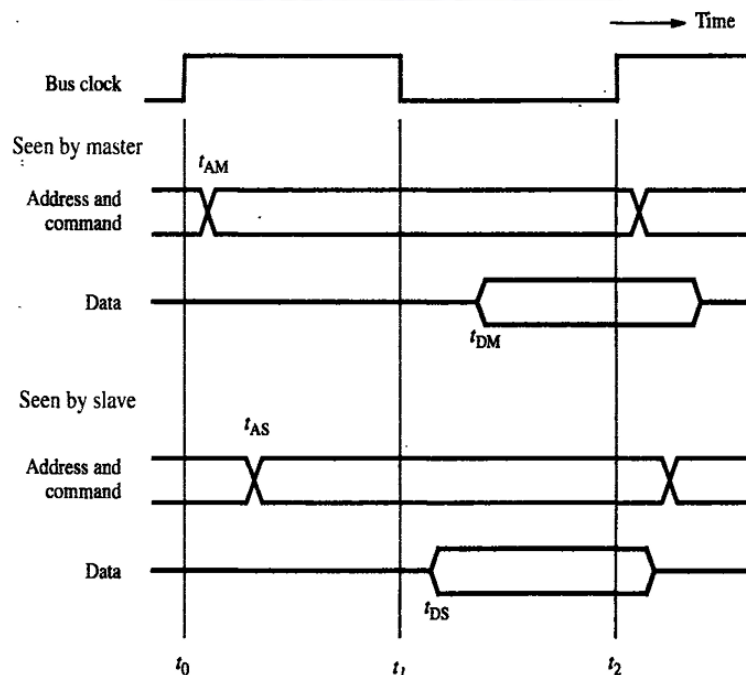
maximum propagation time on the bus plus the setup time of the input buffer register of the master.

A similar procedure is followed for an output operation. The master places the output data on the data lines when it transmits the address and command information. At time t_2 , the addressed device strobbs the data lines and loads the data into its data buffer.



Timing of an input transfer on a synchronous bus.

The timing diagram in Figure above is an idealized representation of the actions that take place on the bus lines. The exact times at which signals actually change state are somewhat different from those shown because of propagation delays on bus wires and in the circuits of the devices. Figure below gives a more realistic picture of what happens in practice.



A detailed timing diagram for the input transfer

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

It shows two views of each signal, except the clock. Because signals take time to travel from one device to another, a given signal transition is seen by different devices at different times. One view shows the signal as seen by the master and the other as seen by the slave. We assume that the clock changes are seen at the same time by all devices on the bus. System designers spend considerable effort to ensure that the clock signal satisfies this condition.

The master sends the address and command signals on the rising edge at the beginning of clock period 1 (t_0). However, these signals do not actually appear on the bus until t_{AM} , largely due to the delay in the bus driver circuit. A while later, at t_{AS} , the signals reach the slave. The slave decodes the address and at t_1 sends the requested data. Here again, the data signals do not appear on the bus until t_{DS} . They travel toward the master and arrive at t_{DM} . At t_2 , the master loads the data into its input buffer. Hence the period $t_2 - t_{DM}$ is the setup time for the master's input buffer. The data must continue to be valid after t_2 for a period equal to the hold time of that buffer.

Multiple-Cycle Transfers

The scheme described above results in a simple design for the device interface. However, it has some limitations. Because a transfer has to be completed within one clock cycle, the clock period, $t_2 - t_0$, must be chosen to accommodate the longest delays on the bus and the slowest device interface. This forces all devices to operate at the speed of the slowest device.

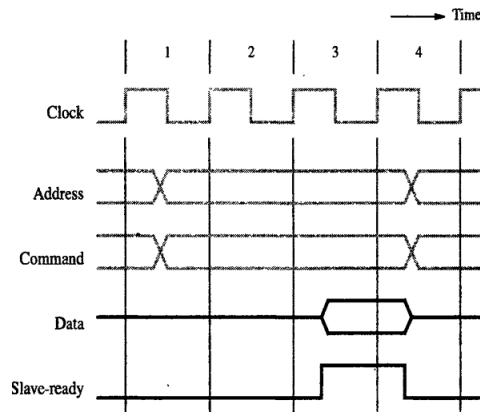
Also, the processor has no way of determining whether the addressed device has actually responded. It simply assumes that, at t_2 , the output data have been received by the I/O device or the input data are available on the data lines. If, because of a malfunction, the device does not respond, the error will not be detected.

To overcome these limitations, most buses incorporate control signals that represent a response from the device. These signals inform the master that the slave has recognized its address and that it is ready to participate in a data-transfer operation. They also make it possible to adjust the duration of the data-transfer period to suit the needs of the participating devices. To simplify this process, a high-frequency clock signal is used such that a complete data transfer cycle would span several clock cycles. Then, the number of clock cycles involved can vary from one device to another.

An example of this approach is shown in Figure below. During clock cycle 1, the master sends address and command information on the bus, requesting a read operation. The slave receives this information and decodes it. On the following active edge of the clock, that is, at the beginning of clock cycle 2, it makes a decision to respond and begins to access the requested data. We have assumed that some delay is involved in getting the data, and hence the slave cannot respond immediately. The data become ready and are placed on the bus in clock cycle 3. At the same time, the slave asserts a control signal called Slave-ready. The master, which has been waiting for this signal, strobes the data into its input buffer at the end of clock cycle 3. The bus transfer operation is now complete, and the master may send a new address to start a new transfer in clock cycle 4.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The Slave-ready signal acts as an acknowledgment that valid data is available. Because different devices require varying response times, this signal allows transfer durations to dynamically adjust. If a device fails to respond within a predefined maximum number of clock cycles—often due to an invalid address or malfunction—the master will abort the operation.

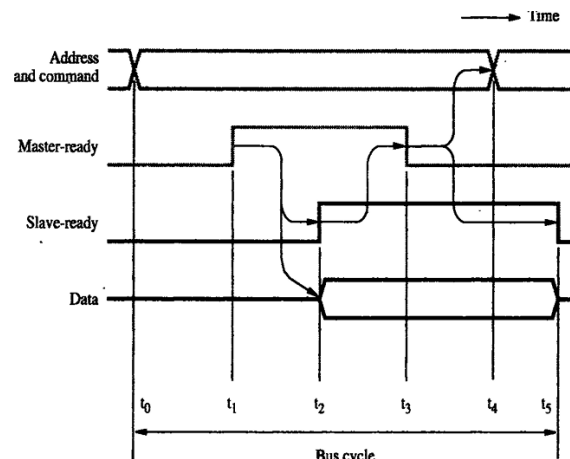


An input transfer using multiple clock cycles

2. Asynchronous Bus

An alternative scheme for controlling data transfers on the bus is based on the use of a handshake between the master and the slave. The concept of a handshake is a generalization of the idea of the Slave-ready signal in Figure above. The common clock is replaced by two timing control lines, Master-ready and Slave-ready. The first is asserted by the master to indicate that it is ready for a transaction, and the second is a response from the slave.

In principle, a data transfer controlled by a handshake protocol proceeds as follows. The master places the address and command information on the bus. Then it indicates to all devices that it has done so by activating the Master-ready line. This causes all devices on the bus to decode the address. The selected slave performs the required operation and informs the processor it has done so by activating the slave ready line. The master waits for slave-ready to become asserted before it removes its signals from the bus. In the case of a read operation, it also strobes the data into its input buffer. An example of timing of an input data transfer using the handshake scheme is given below.



Handshake control of data transfer during an input operation

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Chapter 2: The Memory System

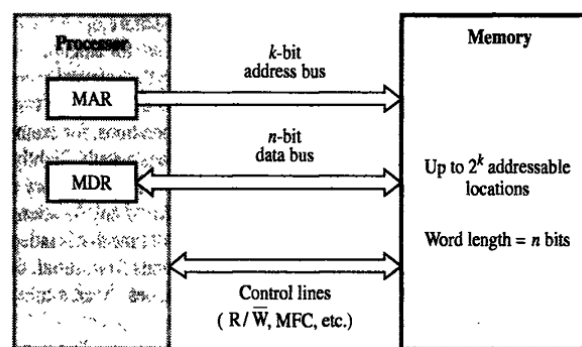
4.8 Some Basic Concepts

The maximum size of the memory that can be used in any computer is determined by the addressing scheme. For example, a 16-bit computer that generates 16-bit addresses is capable of addressing up to $2^{16} = 64K$ memory locations. Similarly, machines whose instructions generate 32-bit addresses can utilize a memory that contains up to $2^{32} = 4G$ (giga) memory locations, whereas machines with 40-bit addresses can access up to $2^{40} = 1T$ (tera) locations. The number of locations represents the size of the address space of the computer.

Most modern computers are byte addressable. The big-endian arrangement is used in the 68000 processor. The little-endian arrangement is used in Intel processors. The ARM architecture can be configured to use either arrangement. As far as the memory structure is concerned, there is no substantial difference between the two schemes.

The memory is usually designed to store and retrieve data in word-length quantities. In fact, the number of bits actually stored or retrieved in one memory access is the most common definition of the word length of a computer. Consider, for example, a byte-addressable computer whose instructions generate 32-bit addresses. When a 32-bit address is sent from the processor to the memory unit, the high-order 30 bits determine which word will be accessed. If a byte quantity is specified, the low-order 2 bits of the address specify which byte location is involved. In a Read operation, other bytes may be fetched from the memory, but they are ignored by the processor. If the byte operation is a Write, however, the control circuitry of the memory must ensure that the contents of other bytes of the same word are not changed.

From the system standpoint, we can view the memory unit as a black box. Data transfer between the memory and the processor takes place through the use of two processor registers, usually called MAR (memory address register) and MDR (memory data register). If MAR is k bits long and MDR is n bits long, then the memory unit may contain up to 2^k addressable locations. During a memory cycle, n bits of data are transferred between the memory and the processor. This transfer takes place over the processor bus, which has k address lines and n data lines. The bus also includes the control lines Read/write' (R/W') and Memory Function Completed (MFC) for coordinating data transfers. Other control lines may be added to indicate the number of bytes to be transferred. The connection between the processor and the memory is shown schematically in Figure.



Connection of the memory to the processor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The processor reads data from the memory by loading the address of the required memory location into the MAR register and setting the R/W' line to 1. The memory responds by placing the data from the addressed location onto the data lines and confirms this action by asserting the MFC signal. Upon receipt of the MFC signal, the processor loads the data on the data lines into the MDR register.

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

If read or write operations involve consecutive address locations in the main memory, then a "block transfer" operation can be performed in which the only address sent to the memory is the one that identifies the first location.

Memory accesses may be synchronized using a clock, or they may be controlled using special signals that control transfers on the bus, using the bus signalling schemes. Memory read and write operations are controlled as input and output bus transfers, respectively.

A useful measure of the speed of memory units is the time that elapses between the initiation of an operation and the completion of that operation, for example, the time between the Read and the MFC signals. This is referred to as the memory access time. Another important measure is the memory cycle time, which is the minimum time delay required between the initiation of two successive memory operations, for example, the time between two successive Read operations. The cycle time is usually slightly longer than the access time, depending on the implementation details of the memory unit.

A memory unit is called random-access memory (RAM) if any location can be accessed for a Read or Write operation in some fixed amount of time that is independent of the location's address. This distinguishes such memory units from serial, or partly serial, access storage devices such as magnetic disks and tapes. Access time on the latter devices depends on the address or position of the data. The basic technology for implementing the memory uses semiconductor integrated circuits.

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

Virtual memory is another important concept related to memory organization. So far, we have assumed that the addresses generated by the processor directly specify physical locations in the memory. This may not always be the case. For reasons that will become apparent later in this chapter, data may be stored in physical memory locations that have addresses different from those specified by the program. The memory control circuitry translates the address specified by the program into an address that can be used to access the physical memory. In such a case, an address generated by the processor is referred to as a virtual or logical address. The virtual address space is mapped onto the physical memory where data are actually stored. The mapping

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

function is implemented by a special memory control circuit, often called the memory management unit. This mapping function can be changed during program execution according to system requirements.

Virtual memory is used to increase the apparent size of the physical memory. Data are addressed in a virtual address space that can be as large as the addressing capability of the processor. But at any given time, only the active portion of this space is mapped onto locations in the physical memory. The remaining virtual addresses are mapped onto the bulk storage devices used, which are usually magnetic disks. As the active portion of the virtual address space changes during program execution, the memory management unit changes the mapping function and transfers data between the disk and the memory. Thus, during every memory cycle, an address-processing mechanism determines whether the addressed information is in the physical memory unit. If it is, then the proper word is accessed and execution proceeds. If it is not, a page of words containing the desired word is transferred from the disk to the memory. This page displaces some page in the memory that is currently inactive.

4.9 Semiconductor RAM Memories

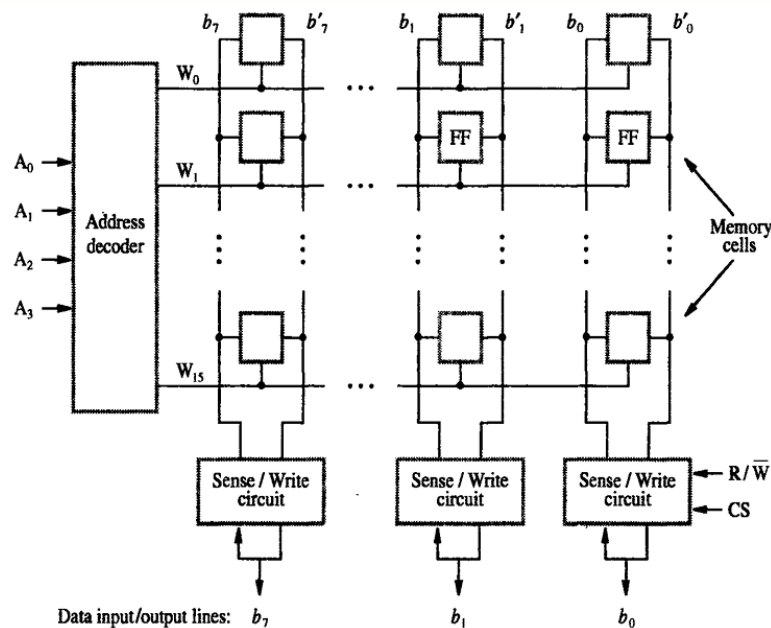
Semiconductor memories are available in a wide range of speeds. Their cycle times range from 100 ns to less than 10 ns. When first introduced in the late 1960s, they were much more expensive than the magnetic-core memories they replaced. Because of rapid advances in VLSI (Very Large-Scale Integration) technology, the cost of semiconductor memories has dropped dramatically. As a result, they are now used almost exclusively in implementing memories.

Internal Organization of Memory Chips

Memory cells are usually organized in the form of an array, in which each cell is capable of storing one bit of information. A possible organization is illustrated in Figure below. Each row of cells constitutes a memory word, and all cells of a row are connected to a common line referred to as the word line, which is driven by the address decoder on the chip. The cells in each column are connected to a Sense/Write circuit by two-bit lines. The Sense/Write circuits are connected to the data input/output lines of the chip. During a Read operation, these circuits sense, or read, the information stored in the cells selected by a word line and transmit this information to the output data lines. During a Write operation, the Sense/Write circuits receive input information and store it in the cells of the selected word.

Figure below is an example of a very small memory chip consisting of 16 words of 8 bits each. This is referred to as a 16×8 organization. The data input and the data output of each Sense/Write circuit are connected to a single bidirectional data line that can be connected to the data bus of a computer. Two control lines, $R/\bar{W}$ and CS, are provided in addition to address and data lines. The $R/\bar{W}$ (Read/Write) input specifies the required operation, and the CS (Chip Select) input selects a given chip in a multichip memory system.

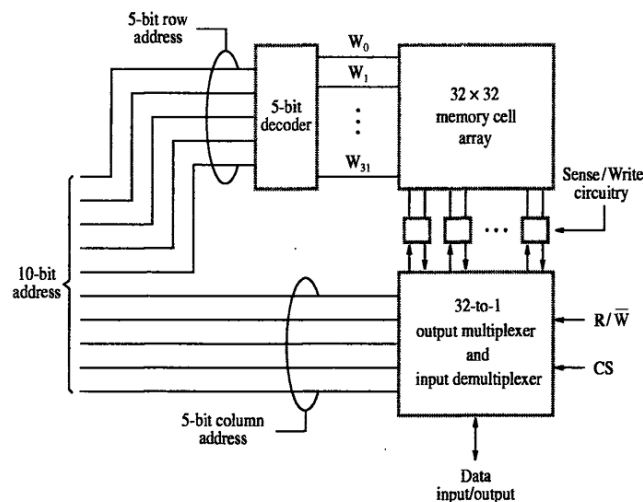
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Organization of bit cells in a memory chip.

The memory circuit in Figure stores 128 bits and requires 14 external connections for address, data, and control lines. Of course, it also needs two lines for power supply and ground connections. Consider now a slightly larger memory circuit, one that has 1K (1024) memory cells. This circuit can be organized as a 128×8 memory, requiring a total of 19 external connections. Alternatively, the same number of cells can be organized into a $1K \times 1$ format. In this case, a 10-bit address is needed, but there is only one data line, resulting in 15 external connections.

Figure below shows such an organization. The required 10-bit address is divided into two groups of 5 bits each to form the row and column addresses for the cell array. A row address selects a row of 32 cells, all of which are accessed in parallel. However, according to the column address, only one of these cells is connected to the external data line by the output multiplexer and input demultiplexer.



Organization of a 1k x 1 memory chip

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Large chips essentially have the same organization as Figure below but use a larger memory cell array and have more external connections. For example, a 4M-bit chip may have a $512K \times 8$ organization, in which case 19 address and 8 data input/output pins are needed. Chips with a capacity of hundreds of megabits are now available.

Static Memories

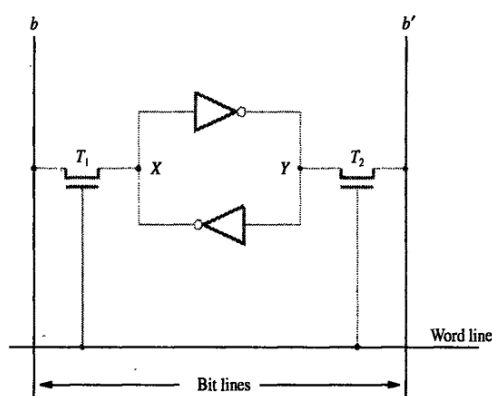
Memories that consist of circuits capable of retaining their state as long as power is applied are known as static memories. Figure below illustrates how a static RAM (SRAM) cell may be implemented. Two inverters are cross connected to form a latch. The latch is connected to two-bit lines by transistors T_1 and T_2 . These transistors act as switches that can be opened or closed under control of the word line. When the word line is at ground level, the transistors are turned off and the latch retains its state. For example, let us assume that the cell is in state 1 if the logic value at point X is 1 and at point Y is 0. This state is maintained as long as the signal on the word line is at ground level.

Read Operation

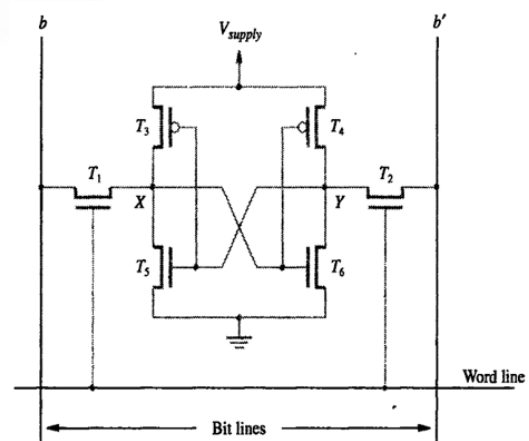
In order to read the state of the SRAM cell, the word line is activated to close switches T_1 and T_2 . If the cell is in state 1, the signal on bit line b is high and the signal on bit line b' is low. The opposite is true if the cell is in state 0. Thus, b and b' are complements of each other. Sense/Write circuits at the end of the bit lines monitor the state of b and b' and set the output accordingly.

Write Operation

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



A Static RAM Cell



An Example of a CMOS memory cell

CMOS cell

A CMOS realization of the cell is given above. Transistor pairs (T_3, T_5) and (T_4, T_6) form the inverters in the latch. The power supply voltage, V_{supply} , is 5 V in older CMOS SRAMs or 3.3

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

V in new low-voltage versions. Note that continuous power is needed for the cell to retain its state. If power is interrupted, the cell's contents will be lost. When power is restored, the latch will settle into a stable state, but it will not necessarily be the same state the cell was in before the interruption. Hence, SRAMs are said to be volatile memories because their contents are lost when power is interrupted.

A major advantage of CMOS SRAMs is their very low power consumption because current flows in the cell only when the cell is being accessed. Otherwise, T_1 , T_2 , and one transistor in each inverter are turned off, ensuring that there is no active path between V_{supply} and ground.

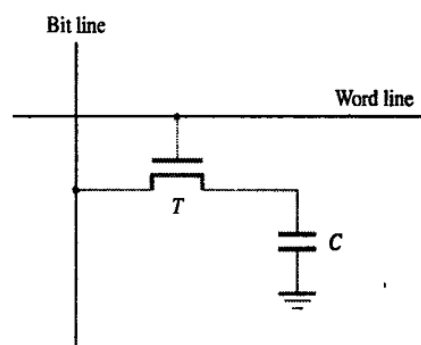
Static RAMs can be accessed very quickly. Access times of just a few nanoseconds are found in commercially available chips. SRAMs are used in applications where speed is of critical concern.

Asynchronous DRAMs

Static RAMs are fast, but they come at a high cost because their cells require several transistors. Less expensive RAMs can be implemented if simpler cells are used. However, such cells do not retain their state indefinitely; hence, they are called dynamic RAMs (DRAMs).

Information is stored in a dynamic memory cell in the form of a charge on a capacitor, and this charge can be maintained for only tens of milliseconds. Since the cell is required to store information for a much longer time, its contents must be periodically refreshed by restoring the capacitor charge to its full value.

An example of a dynamic memory cell that consists of a capacitor, C , and a transistor, T , is shown in Figure below. In order to store information in this cell, transistor T is turned on and an appropriate voltage is applied to the bit line. This causes a known amount of charge to be stored in the capacitor.



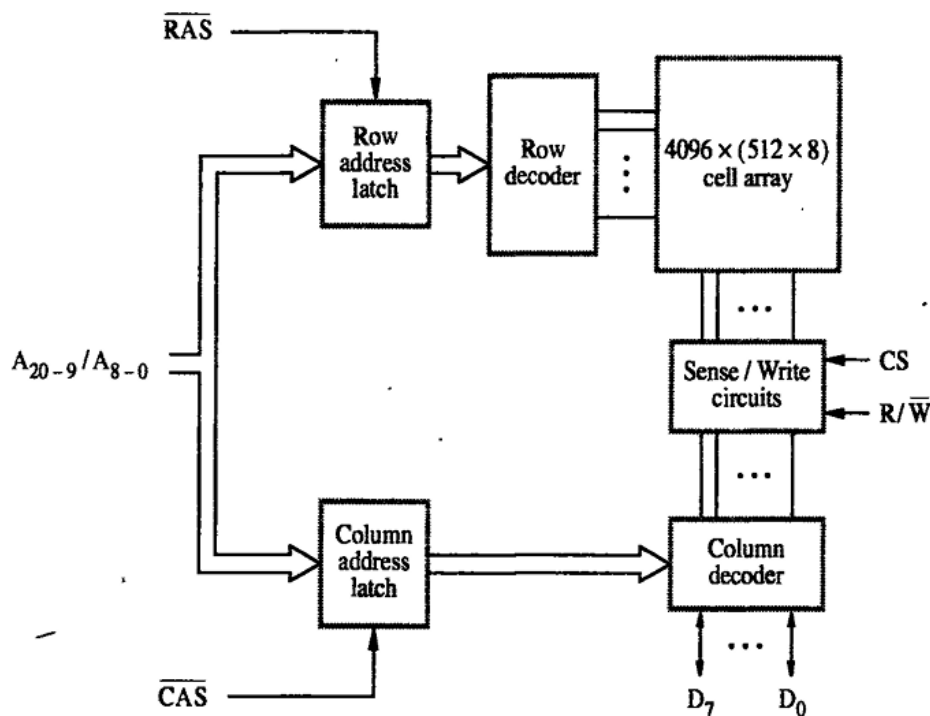
A single-transistor dynamic memory cell

After the transistor is turned off, the capacitor begins to discharge. This is caused by the capacitor's own leakage resistance and by the fact that the transistor continues to conduct a tiny amount of current, measured in picoamperes, after it is turned off. Hence, the information stored in the cell can be retrieved correctly only if it is read before the charge on the capacitor drops below some threshold value. During a Read operation, the transistor in a selected cell is turned on. A sense amplifier connected to the bit line detects whether the charge stored on the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

capacitor is above the threshold value. If so, it drives the bit line to a full voltage that represents logic value 1. This voltage recharges the capacitor to the full charge that corresponds to logic value 1. If the sense amplifier detects that the charge on the capacitor is below the threshold value, it pulls the bit line to ground level, which ensures that the capacitor will have no charge, representing logic value 0. Thus, reading the contents of the cell automatically refreshes its contents. All cells in a selected row are read at the same time, which refreshes the contents of the entire row.

A 16-megabit DRAM chip, configured as $2M \times 8$, is shown in Figure below. The cells are organized in the form of a $4K \times 4K$ array. The 4096 cells in each row are divided into 512 groups of 8, so that a row can store 512 bytes of data. Therefore, 12 address bits are needed to select a row. Another 9 bits are needed to specify a group of 8 bits in the selected row. Thus, a 21-bit address is needed to access a byte in this memory. The high-order 12 bits and the low-order 9 bits of the address constitute the row and column addresses of a byte, respectively.



Internal organization of a $2M \times 8$ dynamic memory chip

To reduce the number of pins needed for external connections, the row and column addresses are multiplexed on 12 pins. During a Read or a Write operation, the row address is applied first. It is loaded into the row address latch in response to a signal pulse on the Row Address Strobe (RAS) input of the chip. Then a Read operation is initiated, in which all cells on the selected row are read and refreshed. Shortly after the row address is loaded, the column address is applied to the address pins and loaded into the column address latch under control of the Column Address Strobe (CAS) signal. The information in this latch is decoded, and the appropriate group of 8 Sense/Write circuits are selected. If the R/W control signal indicates a Read operation, the output values of the selected circuits are transferred to the data lines, D_{7-0} .

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

For a Write operation, the information on the D_{7-0} lines is transferred to the selected circuits. This information is then used to overwrite the contents of the selected cells in the corresponding 8 columns. Note that in commercial DRAM chips, the RAS and CAS control signals are active low so that they cause the latching of addresses when they change from high to low. To indicate this fact, these signals are shown on diagrams as RAS and CAS.

Applying a row address causes all cells on the corresponding row to be read and refreshed during both Read and Write operations. To ensure that the contents of a DRAM are maintained, each row of cells must be accessed periodically. A refresh circuit usually performs this function automatically. Many dynamic memory chips incorporate a refresh facility within the chips themselves. In this case, the dynamic nature of these memory chips is almost invisible to the user.

In the DRAM described in this section, the timing of the memory device is controlled asynchronously. A specialized memory controller circuit provides the necessary control signals, RAS and CAS, that govern the timing. The processor must take into account the delay in the response of the memory. Such memories are referred to as asynchronous DRAMs.

Because of their high density and low cost, DRAMs are widely used in the memory units of computers. Available chips range in size from 1M to 256M bits, and even larger chips are being developed. To reduce the number of memory chips needed in a given computer, a DRAM chip is organized to read or write a number of bits in parallel. To provide flexibility in designing memory systems, these chips are manufactured in different organizations. For example, a 64-Mbit chip may be organized as $16M \times 4$, $8M \times 8$, or $4M \times 16$.

Fast Page Mode

When the DRAM is accessed, the contents of all 4096 cells in the selected row are sensed, but only 8 bits are placed on the data lines D_{7-0} . This byte is selected by the column address bits A_{8-0} . A simple modification can make it possible to access the other bytes in the same row without having to reselect the row. A latch can be added at the output of the sense amplifier in each column. The application of a row address will load the latches corresponding to all bits in the selected row. Then, it is only necessary to apply different column addresses to place the different bytes on the data lines.

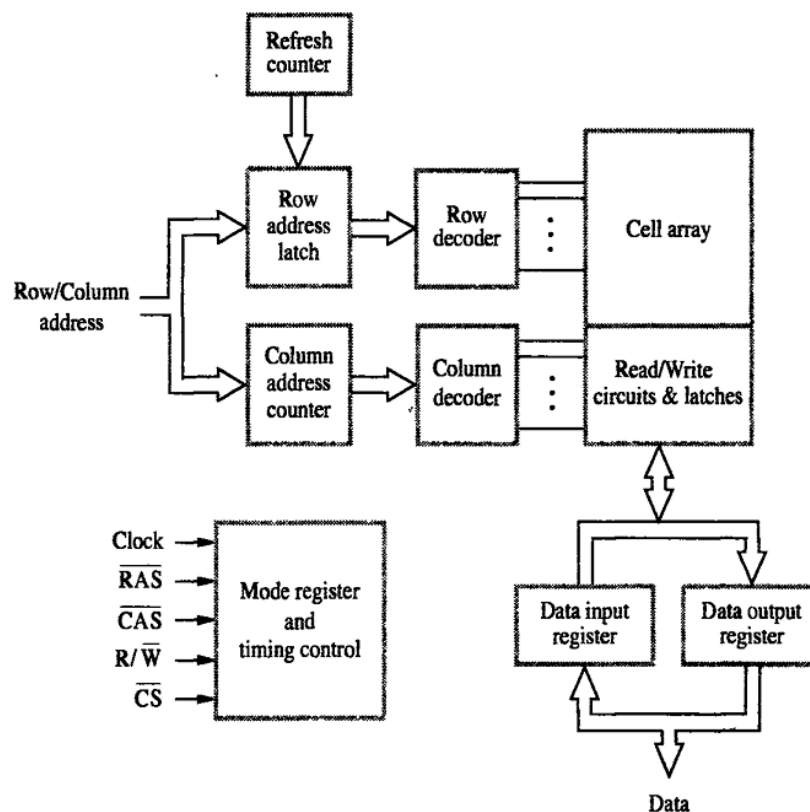
The most useful arrangement is to transfer the bytes in sequential order, which is achieved by applying a consecutive sequence of column addresses under the control of CAS Signals. This scheme allows transferring a block of data at a much faster rate than can be achieved for transfers involving random addresses. The block transfer capability is referred to as the fast page mode feature. (Popular jargon refers to small groups of bytes as blocks, and larger groups as pages.)

The faster rate attainable in block transfers can be exploited in applications in which memory accesses follow regular patterns, such as in graphics terminals. This feature is also beneficial in general-purpose computers for transferring data blocks between the main memory and a cache.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Synchronous Drams

More recent developments in memory technology have resulted in DRAMs whose operation is directly synchronized with a clock signal. Such memories are known as synchronous DRAMS (SDRAMs). Figure below indicates the structure of an SDRAM. The cell array is the same as in asynchronous DRAMs. The address and data connections are buffered by means of registers. We should particularly note that the output of each sense amplifier is connected to a latch.



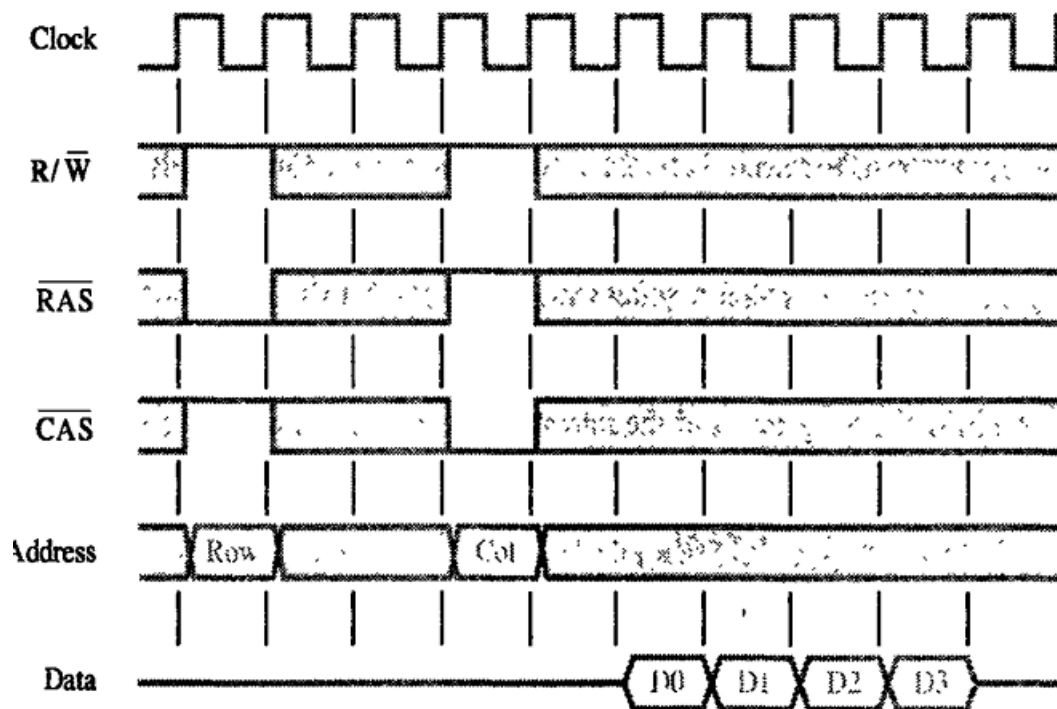
Synchronous DRAM

A Read operation causes the contents of all cells in the selected row to be loaded into these latches. But, if an access is made for refreshing purposes only, it will not change the contents of these latches; it will merely refresh the contents of the cells. Data held in the latches that correspond to the selected column(s) are transferred into the data output register, thus becoming available on the data output pins.

SDRAMs have several different modes of operation, which can be selected by writing control information into a mode register. For example, burst operations of different lengths can be specified. The burst operations use the block transfer capability described above as the fast page mode feature. In SDRAMs, it is not necessary to provide externally generated pulses on the CAS line to select successive columns. The necessary control signals are provided internally using a column counter and the clock signal. New data can be placed on the data lines in each clock cycle. All actions are triggered by the rising edge of the clock.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Figure below shows a timing diagram for a typical burst read of length 4. First, the row address is latched under control of the RAS' signal. The memory typically takes 2 or 3 clock cycles to activate the selected row. Then, the column address is latched under control of the CAS' signal. After a delay of one clock cycle, the first set of data bits is placed on the data lines. The SDRAM automatically increments the column address to access the next three sets of bits in the selected row, which are placed on the data lines in the next 3 clock cycles.



Burst read of length 4 in an SDRAM

SDRAMs have built-in refresh circuitry. A part of this circuitry is a refresh counter, which provides the addresses of the rows that are selected for refreshing. In a typical SDRAM, each row must be refreshed at least every 64 ms.

Commercial SDRAMs can be used with clock speeds above 100 MHz. These chips are designed to meet the requirements of commercially available processors that are used in large volume. For example, Intel has defined PC100 and PC133 bus specifications in which the system bus (to which the main memory is connected) is controlled by a 100 or 133 MHz clock, respectively. Therefore, major manufacturers of memory chips produce 100 and 133 MHz SDRAM chips.

Latency and Bandwidth

Transfers between the memory and the processor involve single words of data or small blocks of words. Large blocks, constituting a page of data, are transferred between the memory and the disks. The speed and efficiency of these transfers have a large impact on the performance of a computer system. A good indication of the performance is given by two parameters: latency and bandwidth.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The term memory latency is used to refer to the amount of time it takes to transfer a word of data to or from the memory. In the case of reading or writing a single word of data, the latency provides a complete indication of memory performance. But, in the case of burst operations that transfer a block of data, the time needed to complete the operation depends also on the rate at which successive words can be transferred and on the size of the block. In block transfers, the term latency is used to denote the time it takes to transfer the first word of data. This time is usually substantially longer than the time needed to transfer each subsequent word of a block. For instance, in the timing diagram in Figure above, the access cycle begins with the assertion of the RAS' signal. The first word of data is transferred five clock cycles later. Thus, the latency is five clock cycles. If the clock rate is 100 MHz, then the latency is 50 ns. The remaining three words are transferred in consecutive clock cycles.

When transferring blocks of data, it is of interest to know how much time is needed to transfer an entire block. Since blocks can be variable in size, it is useful to define a performance measure in terms of the number of bits or bytes that can be transferred in one second. This measure is often referred to as the memory bandwidth. The bandwidth of a memory unit (consisting of one or more memory chips) depends on the speed of access to the stored data and on the number of bits that can be accessed in parallel. However, the effective bandwidth in a computer system (involving data transfers between the memory and the processor) is not determined solely by the speed of the memory; it also depends on the transfer capability of the links that connect the memory and the processor, typically the speed of the bus. Memory chips are usually designed to meet the speed requirements of popular buses. The bandwidth clearly depends on the speed of access and transmission along a single wire, as well as on the number of bits that can be transferred in parallel, namely the number of wires. Thus, the bandwidth is the product of the rate at which data are transferred (and accessed) and the width of the data bus.

Double-Data-Rate SDRAM

In the continuous quest for improved performance, a faster version of SDRAM has been developed. The standard SDRAM performs all actions on the rising edge of the clock signal. A similar memory device is available, which accesses the cell array in the same way, but transfers data on both edges of the clock. The latency of these devices is the same as for standard SDRAMs. But, since they transfer data on both edges of the clock, their bandwidth is essentially doubled for long burst transfers. Such devices are known as double-data-rate SDRAMs (DDR SDRAMs).

To make it possible to access the data at a high enough rate, the cell array is organized in two banks. Each bank can be accessed separately. Consecutive words of a given block are stored in different banks. Such interleaving of words allows simultaneous access to two words that are transferred on successive edges of the clock.

DDR SDRAMs and standard SDRAMs are most efficiently used in applications where block transfers are prevalent. This is the case in general-purpose computers in which main memory transfers are primarily to and from processor caches. Block transfers are also done in high-quality video displays.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Structure of Larger Memories

Static Memory System

Consider a memory consisting of 2M (2,097,152) words of 32 bits each. Figure shows how we can implement this memory using 512K x 8 static memory chips. Each column in the figure consists of four chips, which implement one byte position. Four of these sets provide the required 2M x 32 memory. Each chip has a control input called Chip Select. When this input is set to 1, it enables the chip to accept data from or to place data on its data lines. The data output for each chip is of the three-state type. Only the selected chip places data on the data output line, while all other outputs are in the high-impedance state. Twenty-one address bits are needed to select a 32-bit word in this memory. The high-order 2 bits of the address are decoded to determine which of the four Chip Select control signals should be activated, and the remaining 19 address bits are used to access specific byte locations inside each chip of the selected row. The R/W inputs of all chips are tied together to provide a common Read/Write control.

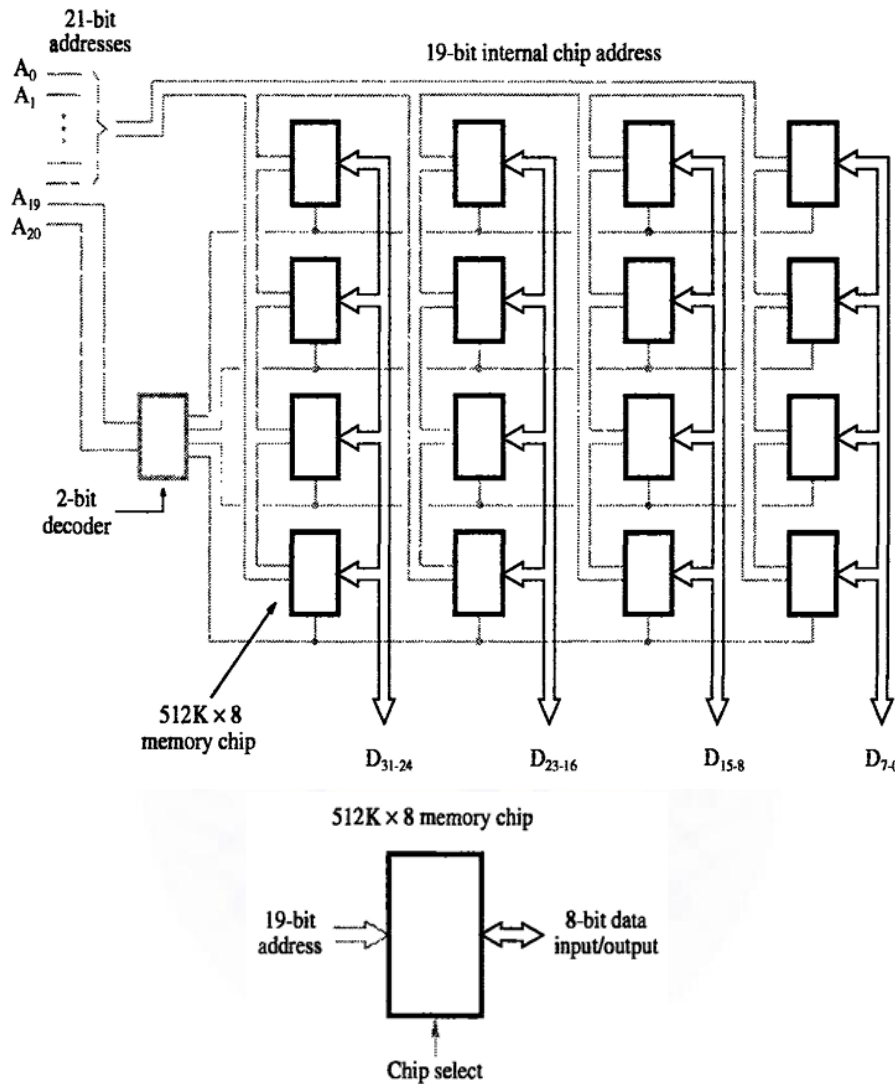
Dynamic Memory System

The organization of large dynamic memory systems is essentially the same as the memory shown in Figure. However, physical implementation is often done more conveniently in the form of memory modules.

Modern computers use very large memories; even a small personal computer is likely to have at least 32M bytes of memory. Typical workstations have at least 128M bytes of memory. A large memory leads to better performance because more of the programs and data used in processing can be held in the memory, thus reducing the frequency of accessing the information in secondary storage. However, if a large memory is built by placing DRAM chips directly on the main system printed-circuit board that contains the processor, often referred to as a motherboard, it will occupy an unacceptably large amount of space on the board. Also, it is awkward to provide for future expansion of the memory, because space must be allocated and wiring provided for the maximum expected size. These packaging considerations have led to the development of larger memory units known as SIMMs (Single In-line Memory Modules) and DIMMs (Dual In-line Memory Modules). Such a module is an assembly of several memory chips on a separate small board that plugs vertically into a single socket on the motherboard.

SIMMs and DIMMs of different sizes are designed to use the same size socket. For example, 4M x 32, 16M x 32, and 32M x 32-bit DIMMs all use the same 100-pin socket. Similarly, 8M x 64, 16M x 64, 32M x 64, and 64M x 72 DIMMs use a 168-pin socket. Such modules occupy a smaller amount of space on a motherboard, and they allow easy expansion by replacement if a larger module uses the same socket as the smaller one.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Organization of a 2M x 32 memory module using 512K x 8 static memory chips

Memory System Considerations

The choice of a RAM chip for a given application depends on several factors. Foremost among these factors are the cost, speed, power dissipation, and size of the chip.

Static RAMs are generally used only when very fast operation is the primary requirement. Their cost and size are adversely affected by the complexity of the circuit that realizes the basic cell. They are used mostly in cache memories. Dynamic RAMs are the predominant choice for implementing computer main memories. The high densities achievable in these chips make large memories economically feasible.

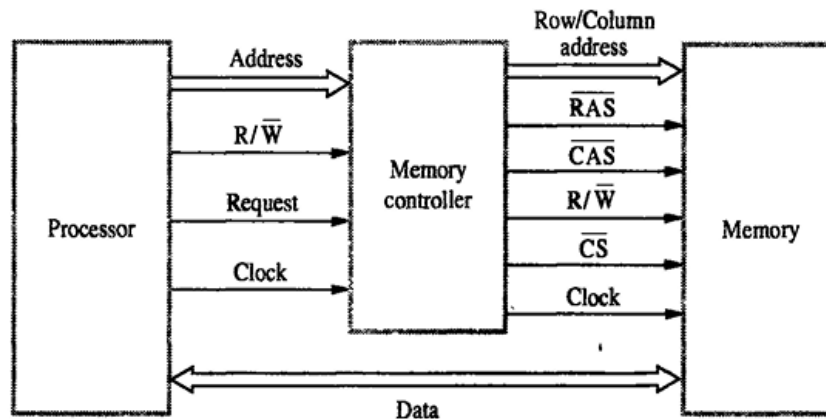
Memory Controller

To reduce the number of pins, the dynamic memory chips use multiplexed address inputs. The address is divided into two parts. The high-order address bits, which select a row in the cell array, are provided first and latched into the memory chip under control of the RAS signal.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Then, the low-order address bits, which select a column, are provided on the same address pins and latched using the CAS signal.

A typical processor issues all bits of an address at the same time. The required multiplexing of address bits is usually performed by a memory controller circuit, which is interposed between the processor and the dynamic memory as shown in following Figure.



Use of a memory controller

The controller accepts a complete address and the R/W signal from the processor, under control of a Request signal which indicates that a memory access operation is needed. The controller then forwards the row and column portions of the address to the memory and generates the RAS' and CAS' signals. Thus, the controller provides the RAS-CAS timing, in addition to its address multiplexing function. It also sends the R/W' and CS signals to the memory. The CS signal is usually active low, hence it is shown as CS' in Figure. Data lines are connected directly between the processor and the memory. Note that the clock signal is needed in SDRAM chips.

When used with DRAM chips, which do not have self-refreshing capability, the memory controller has to provide all the information needed to control the refreshing process. It contains a refresh counter that provides successive row addresses. Its function is to cause the refreshing of all rows to be done within the period specified for a particular device.

Refresh Overhead

All dynamic memories have to be refreshed. In older DRAMs, a typical period for refreshing all rows was 16 ms. In typical SDRAMs, a typical period is 64 ms.

Consider an SDRAM whose cells are arranged in 8K (8192) rows. Suppose that it takes four clock cycles to access (read) each row. Then, it takes $8192 \times 4 = 32,768$ cycles to refresh all rows. At a clock rate of 133 MHz, the time needed to refresh all rows is $32,768 / (133 \times 10^6) = 246 \times 10^{-9}$ seconds. Thus, the refreshing process occupies 0.246 ms in each 64-ms time interval. Therefore, the refresh overhead is $0.246 / 64 = 0.0038$, which is less than 0.4 percent of the total time available for accessing the memory.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

RAMBUS Memory

The performance of a dynamic memory is characterized by its latency and bandwidth. Since all dynamic memory chips use similar organizations for their cell arrays, their latencies tend to be similar if the chips are produced using the same manufacturing process. On the other hand, the effective bandwidth of a memory system depends not only on the structure of the memory chips, but also on the nature of the connecting path to the processor. DDR SDRAMs and standard SDRAMs are connected to the processor bus. Thus, the speed of transfers is not just a function of the speed of the memory device- it also depends on the speed of the bus. A bus clocked at 133 MHz allows at most one transfer every 7.5 ns, or two transfers if both edges of the clock are used. The only way to increase the amount of data that can be transferred on a speed-limited bus is to increase the width of the bus by providing more data lines, thus widening the bus.

A very wide bus is expensive and requires a lot of space on a motherboard. An alternative approach is to implement a narrow bus that is much faster. This approach was used by Rambus Inc. to develop a proprietary design known as Rambus. The key feature of Rambus technology is a fast-signaling method used to transfer information between chips. Instead of using signals that have voltage levels of either 0 or V_{supply} to represent the logic values, the signals consist of much smaller voltage swings around a reference voltage, V_{ref} . The reference voltage is about 2 V, and the two logic values are represented by 0.3 V swings above and below V_{ref} . This type of signaling is generally known as differential signaling. Small voltage swings make it possible to have short transition times, which allows for a high speed of transmission.

Differential signaling and high transmission rates require special techniques for the design of wire connections that serve as communication links. These requirements make it difficult to make the bus wide. It is also necessary to design special circuit interfaces to deal with the differential signals. Rambus provides a complete specification for the design of such communication links, called the Rambus channel. Present designs of Rambus allow for a clock frequency of 400 MHz. Moreover, data are transmitted on both edges of the clock, so that the effective data transfer rate is 800 MHz.

Rambus requires specially designed memory chips. These chips use cell arrays based on the standard DRAM technology. Multiple banks of cell arrays are used to access more than one word at a time. Circuitry needed to interface to the Rambus channel is included on the chip. Such chips are known as Rambus DRAMS (RDRAMs).

The original specification of Rambus provided for a channel consisting of 9 data lines and a number of control and power supply lines. Eight of the data lines are intended for transferring a byte of data. The ninth data line can be used for purposes such as parity checking. Subsequent specifications allow for additional channels. A two-channel Rambus, also known as Direct RDRAM, has 18 data lines intended to transfer two bytes of data at a time. There are no separate address lines.

Communication between the processor, or some other device that can serve as a master, and RDRAM modules, which serve as slaves, is carried out by means of packets transmitted on the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

data lines. There are three types of packets: request, acknowledge, and data. A request packet issued by the master indicates the type of operation that is to be performed. It contains the address of the desired memory location and includes an 8-bit count that specifies the number of bytes involved in the transfer. The operation types include memory reads and writes, as well as reading and writing of various control registers in the RDRAM chips. When the master issues a request packet, the addressed slave responds by returning a positive acknowledgement packet if it can immediately satisfy the request. Otherwise, the slave indicates that it is "busy" by returning a negative acknowledgement packet, in which case the master will try again.

The number of bits in a request packet exceeds the number of data lines, which means that several clock cycles are needed to transmit the entire packet. Using a narrow communication link is compensated by the very high rate of transmission.

RDRAM chips can be assembled into larger modules, similar to SIMMs and DIMMs. One such module, called RIMM, can hold up to 16 RDRAMs.

Rambus technology competes directly with the DDR SDRAM technology. Each has certain advantages and disadvantages. A nontechnical consideration is that the specification of DDR SDRAM is an open standard, while RDRAM is a proprietary design of Rambus Inc. for which the manufacturers of chips have to pay a royalty. Finally, we should note that in the memory market, assuming that the performance is adequate, the decisive factor is often the price of components.

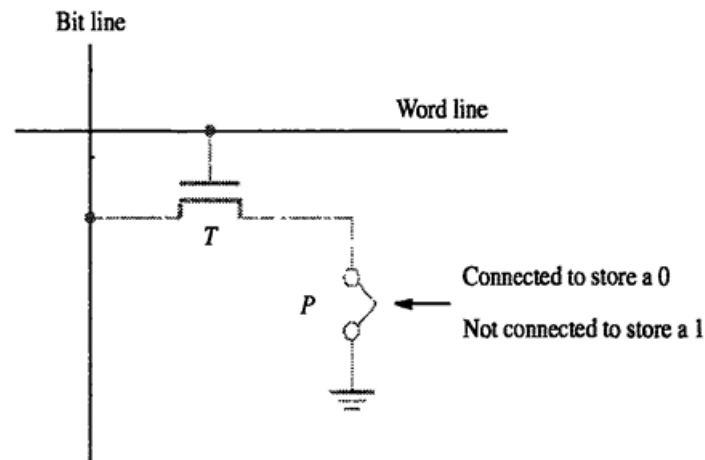
4.10 Read Only Memory

Both SRAM and DRAM chips are volatile, which means that they lose the stored information if power is turned off. There are many applications that need memory devices which retain the stored information if power is turned off. For example, in a typical computer a hard disk drive is used to store a large amount of information, including the operating system software. When a computer is turned on, the operating system software has to be loaded from the disk into the memory. This requires execution of a program that "boots" the operating system. Since the boot program is quite large, most of it is stored on the disk. The processor must execute some instructions that load the boot program into the memory. If the entire memory consisted of only volatile memory chips, the processor would have no means of accessing these instructions. A practical solution is to provide a small amount of non-volatile memory that holds the instructions whose execution results in loading the boot program from the disk.

ROM

Following figure shows a possible configuration for a ROM cell. A logic value 0 is stored in the cell if the transistor is connected to ground at point P; otherwise, a 1 is stored. The bit line is connected through a resistor to the power supply. To read the state of the cell, the word line is activated. Thus, the transistor switch is closed and the voltage on the bit line drops to near zero if there is a connection between the transistor and ground. If there is no connection to ground, the bit line remains at the high voltage, indicating a 1. A sense circuit at the end of the bit line generates the proper output value. Data are written into a ROM when it is manufactured.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



A ROM cell

PROM

Some ROM designs allow the data to be loaded by the user, thus providing a programmable ROM (PROM). Programmability is achieved by inserting a fuse at point P in Figure above. Before it is programmed, the memory contains all 0s. The user can insert 1s at the required locations by burning out the fuses at these locations using high-current pulses. Of course, this process is irreversible.

PROMs provide flexibility and convenience not available with ROMs. The latter are economically attractive for storing fixed programs and data when high volumes of ROMs are produced. However, the cost of preparing the masks needed for storing a particular information pattern in ROMs makes them very expensive when only a small number are required. In this case, PROMs provide a faster and considerably less expensive approach because they can be programmed directly by the user.

EPROM

Another type of ROM chip allows the stored data to be erased and new data to be loaded. Such an erasable, reprogrammable ROM is usually called an EPROM. It provides considerable flexibility during the development phase of digital systems. Since EPROMs are capable of retaining stored information for a long time, they can be used in place of ROMs while software is being developed. In this way, memory changes and updates can be easily made.

An EPROM cell has a structure similar to the ROM cell in Figure. In an EPROM cell, however, the connection to ground is always made at point P and a special transistor is used, which has the ability to function either as a normal transistor or as a disabled transistor that is always turned off. This transistor can be programmed to behave as a permanently open switch, by injecting charge into it that becomes trapped inside. Thus, an EPROM cell can be used to construct a memory in the same way as the previously discussed ROM cell.

The important advantage of EPROM chips is that their contents can be erased and reprogrammed. Erasure requires dissipating the charges trapped in the transistors of memory

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

cells; this can be done by exposing the chip to ultraviolet light. For this reason, EPROM chips are mounted in packages that have transparent windows.

EEPROM

A significant disadvantage of EPROMs is that a chip must be physically removed from the circuit for reprogramming and that its entire contents are erased by the ultraviolet light. It is possible to implement another version of erasable PROMs that can be both programmed and erased electrically. Such chips, called EEPROMs, do not have to be removed for erasure. Moreover, it is possible to erase the cell contents selectively. The only disadvantage of EEPROMs is that different voltages are needed for erasing, writing, and reading the stored data.

Flash Memory

An approach similar to EEPROM technology has more recently given rise to flash memory devices. A flash cell is based on a single transistor controlled by trapped charge, just like an EEPROM cell. While similar in some respects, there are also substantial differences between flash and EEPROM devices. In EEPROM it is possible to read and write the contents of a single cell. In a flash device it is possible to read the contents of a single cell, but it is only possible to write an entire block of cells. Prior to writing, the previous contents of the block are erased. Flash devices have greater density, which leads to higher capacity and a lower cost per bit. They require a single power supply voltage and consume less power in their operation.

The low power consumption of flash memory makes it attractive for use in portable equipment that is battery driven. Typical applications include hand-held computers, cell phones, digital cameras, and MP3 music players. In hand-held computers and cell phones, flash memory holds the software needed to operate the equipment, thus obviating the need for a disk drive. In digital cameras, flash memory is used to store picture image data. In MP3 players, flash memory stores the data that represent sound.

Single flash chips do not provide sufficient storage capacity for the applications mentioned above. Larger memory modules consisting of a number of chips are needed. There are two popular choices for the implementation of such modules: flash cards and flash drives.

Flash Cards

One way of constructing a larger module is to mount flash chips on a small card. Such flash cards have a standard interface that makes them usable in a variety of products. A card is simply plugged into a conveniently accessible slot. Flash cards come in a variety of memory sizes. Typical sizes are 8, 32, and 64 Mbytes. A minute of music can be stored in about 1 Mbyte of memory, using the MP3 encoding format. Hence, a 64-MB flash card can store an hour of music.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Flash Drives

Larger flash memory modules have been developed to replace hard disk drives. These flash drives are designed to fully emulate the hard disks, to the point that they can be fitted into standard disk drive bays. However, the storage capacity of flash drives is significantly lower. Currently, the capacity of flash drives is less than one gigabyte. In contrast, hard disks can store many gigabytes.

The fact that flash drives are solid state electronic devices that have no movable parts provides some important advantages. They have shorter seek and access times, which results in faster response. They have lower power consumption, which makes them attractive for battery driven applications, and they are also insensitive to vibration.

The disadvantages of flash drives vis-a-vis hard disk drives are their smaller capacity and higher cost per bit. Disks provide an extremely low cost per bit. Another disadvantage is that the flash memory will deteriorate after it has been written a number of times.

4.11 Speed, size and Cost of memory systems

A very fast memory can be implemented if SRAM chips are used. But these chips are expensive because their basic cells have six transistors, which precludes packing a very large number of cells onto a single chip. Thus, for cost reasons, it is impractical to build a large memory using SRAM chips. The alternative is to use Dynamic RAM chips, which have much simpler basic cells and thus are much less expensive. But such memories are significantly slower.

Although dynamic memory units in the range of hundreds of megabytes can be implemented at a reasonable cost, the affordable size is still small compared to the demands of large programs with voluminous data. A solution is provided by using secondary storage, mainly magnetic disks, to implement large memory spaces. Very large disks are available at a reasonable price, and they are used extensively in computer systems. However, they are much slower than the semiconductor memory units. A huge amount of cost-effective storage can be provided by magnetic disks. A large, yet affordable, main memory can be built with dynamic RAM technology. This leaves SRAMs to be used in smaller units where speed is of the essence, such as in cache memories.

All of these different types of memory units are employed effectively in a computer. The entire computer memory can be viewed as the hierarchy depicted in Figure. The fastest access is to data held in processor registers. Therefore, if we consider the registers to be part of the memory hierarchy, then the processor registers are at the top in terms of the speed of access. Of course, the registers provide only a minuscule portion of the required memory.

At the next level of the hierarchy is a relatively small amount of memory that can be implemented directly on the processor chip. This memory, called a processor cache, holds copies of instructions and data stored in a much larger memory that is provided externally. There are often two levels of caches. A primary cache is always located on the processor chip. This cache is small because it competes for space on the processor chip, which must implement many other functions. The primary cache is referred to as level 1 (L1) cache. A larger, secondary

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

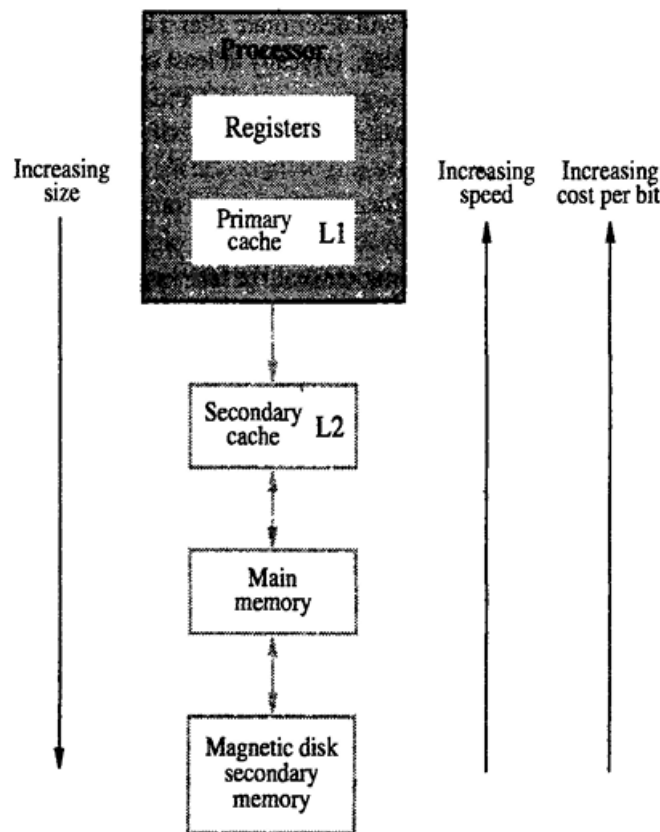
cache is placed between the primary cache and the rest of the memory. It is referred to as level 2 (L2) cache. It is usually implemented using SRAM chips.

Including a primary cache on the processor chip and using a larger, off-chip, secondary cache is currently the most common way of designing computers. However, other arrangements can be found in practice. It is possible not to have a cache on the processor chip at all. Also, it is possible to have both L1 and L2 caches on the processor chip.

The next level in the hierarchy is called the main memory. This rather large memory is implemented using dynamic memory components, typically in the form of SIMMs, DIMMs, or RIMMs. The main memory is much larger but significantly slower than the cache memory. In a typical computer, the access time for the main memory is about ten times longer than the access time for the L1 cache.

Disk devices provide a huge amount of inexpensive storage. They are very slow compared to the semiconductor devices used to implement the main memory.

During program execution, the speed of memory access is of utmost importance. The key to managing the operation of the hierarchical memory system in Figure is to bring the instructions and data that will be used in the near future as close to the processor as possible. This can be done by using the mechanisms presented in the sections that follow.



Memory hierarchy

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

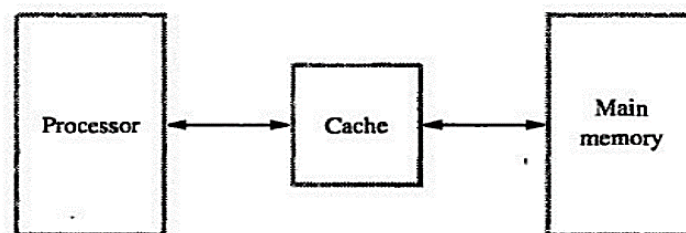
4.12 Cache Memories

Processor is much faster than the main memory. As a result, the processor has to spend much of its time waiting while instructions and data are being fetched from the main memory. This serves as a major obstacle towards achieving good performance. Speed of the main memory cannot be increased beyond a certain point. So, we use Cache memories. Cache memory is an architectural arrangement which makes the main memory appear faster to the processor than it really is. Cache memory is based on the property of computer programs known as “locality of reference”.

Analysis of programs indicates that many instructions in localized areas of a program are executed repeatedly during some period of time, while the others are accessed relatively less frequently. These instructions may be the ones in a loop, nested loop or few procedures calling each other repeatedly. This is called “locality of reference”. Its types are:

Temporal locality of reference: Recently executed instruction is likely to be executed again very soon.

Spatial locality of reference: Instructions with addresses close to a recently instruction are likely to be executed soon.



Use of a cache memory

A simple arrangement of cache memory is as shown above.

- Processor issues a Read request, a block of words is transferred from the main memory to the cache, one word at a time.
- Subsequent references to the data in this block of words are found in the cache.
- At any given time, only some blocks in the main memory are held in the cache. Which blocks in the main memory are in the cache is determined by a “mapping function”.
- When the cache is full, and a block of words needs to be transferred from the main memory, some block of words in the cache must be replaced. This is determined by a “replacement algorithm”.

Cache hit: Existence of a cache is transparent to the processor. The processor issues Read and Write requests in the same manner. If the data is in the cache it is called a Read or Write hit.

- **Read hit:** The data is obtained from the cache.
- **Write hit:** Cache has a replica of the contents of the main memory. Contents of the cache and the main memory may be updated simultaneously. This is the write-through protocol.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Update the contents of the cache, and mark it as updated by setting a bit known as the dirty bit or modified bit. The contents of the main memory are updated when this block is replaced. This is write-back or copy-back protocol.

Cache miss: If the data is not present in the cache, then a Read miss or Write miss occurs.

- **Read miss:** Block of words containing this requested word is transferred from the memory. After the block is transferred, the desired word is forwarded to the processor. The desired word may also be forwarded to the processor as soon as it is transferred without waiting for the entire block to be transferred. This is called load-through or early-restart.
- **Write-miss:** Write-through protocol is used, then the contents of the main memory are updated directly. If write-back protocol is used, the block containing the addressed word is first brought into the cache. The desired word is overwritten with new information.

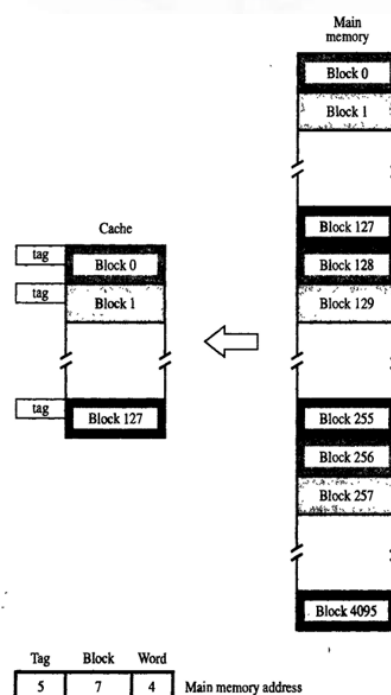
4.13 Mapping Functions

Mapping functions determine how memory blocks are placed in the cache. Three mapping functions can be used.

A simple processor example:

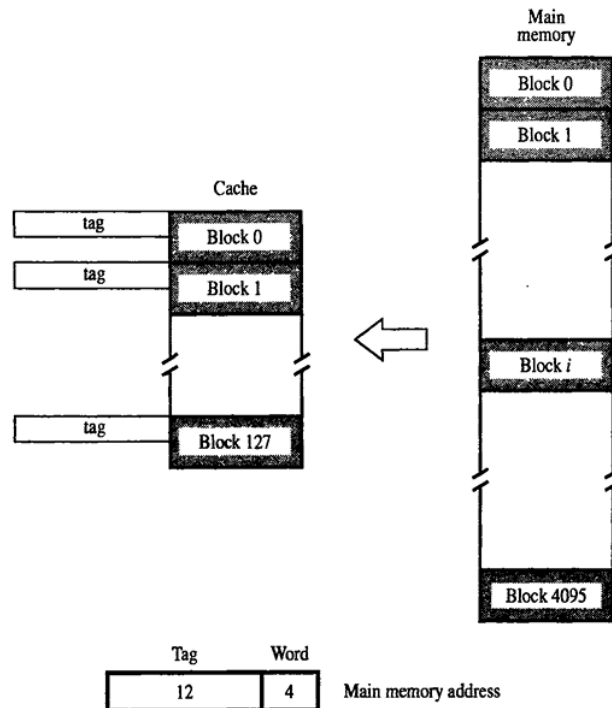
- Cache consisting of 128 blocks of 16 words each.
- Total size of cache is 2048 (2K) words.
- Main memory is addressable by a 16-bit address.
- Main memory has 64K words.
- Main memory has 4K blocks of 16 words each.

Direct mapping

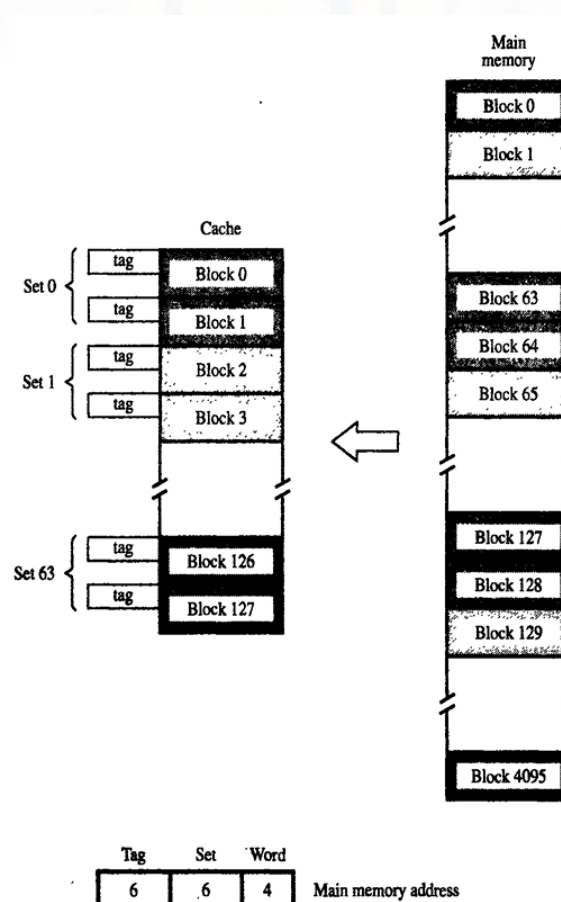


DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Associative mapping



Set-associative mapping



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Module-5

Chapter 1: Basic Processing Unit

5.1 Some Fundamental Concepts

The processing unit which executes machine instructions and coordinates the activities of other units of computer is called the Instruction Set Processor (ISP) or processor or Central Processing Unit (CPU). The primary function of a processor is to execute the instructions stored in memory. Instructions are fetched from successive memory locations and executed in processor, until a branch instruction occurs.

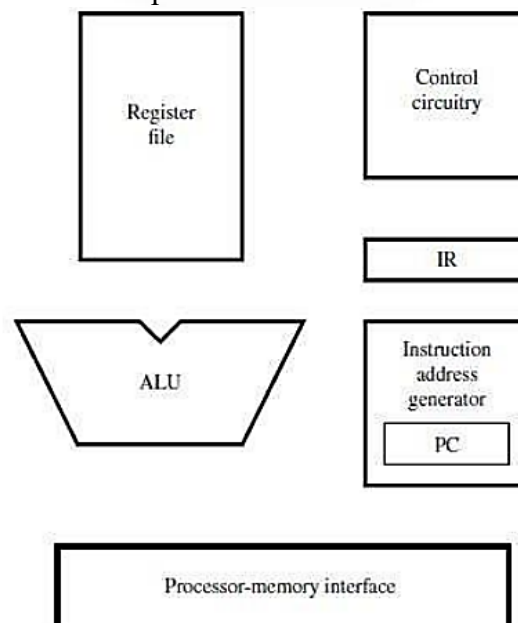
To execute an instruction, processor has to perform following 3 steps:

1. Fetch contents of memory-location pointed to by PC. Content of this location is an instruction to be executed. The instructions are loaded into IR, Symbolically, this operation is written as: $IR \leftarrow [PC]$
2. Increment PC by 4. $PC \leftarrow [PC] + 4$
3. Carry out the actions specified by instruction (in the IR). The steps 1 and 2 are referred to as Fetch Phase. Step 3 is referred to as Execution Phase.

The operation specified by an instruction can be carried out by performing one or more of the following actions:

- 1) Read the contents of a given memory-location and load them into a register.
- 2) Read data from registers or memory location.
- 3) Perform an arithmetic or logic operation and place the result into a register.
- 4) Store data from a register into a given memory-location.

The hardware-components needed to perform these actions are shown in Figure.



Main hardware components of a processor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Single Bus Organization

Here the processor contains only a single bus for the movement of data, address and instructions. ALU and all the registers are interconnected via a Single Common Bus. Data & address lines of the external memory-bus is connected to the internal processor-bus via MDR & MAR respectively. (MDR - Memory Data Register, MAR - Memory Address Register).

MDR has 2 inputs and 2 outputs. Data may be loaded into MDR either from memory-bus (external) or from processor-bus (internal).

MAR's input is connected to internal bus; MAR's output is connected to external bus. (address sent from processor to memory only)

Instruction Decoder & Control Unit is responsible for Decoding the instruction and issuing the control-signals to all the units inside the processor, implementing the actions specified by the instruction (loaded in the IR).

Processor Registers - Register R0 through R(n-1) are also called as General-Purpose Register. The programmer can access these registers for general-purpose use.

Temporary Registers – There are 3 temporary registers in the processor. Registers - Y, Z & Temp are used for temporary storage during program-execution. The programmer cannot access these 3 registers.

In ALU,

- 1) "A" input gets the operand from the output of the multiplexer (MUX).
- 2) "B" input gets the operand directly from the processor-bus. There are 2 options provided for "A" input of the ALU. MUX is used to select one of the 2 inputs. MUX selects either output of Y or constant-value 4 (which is used to increment PC content).

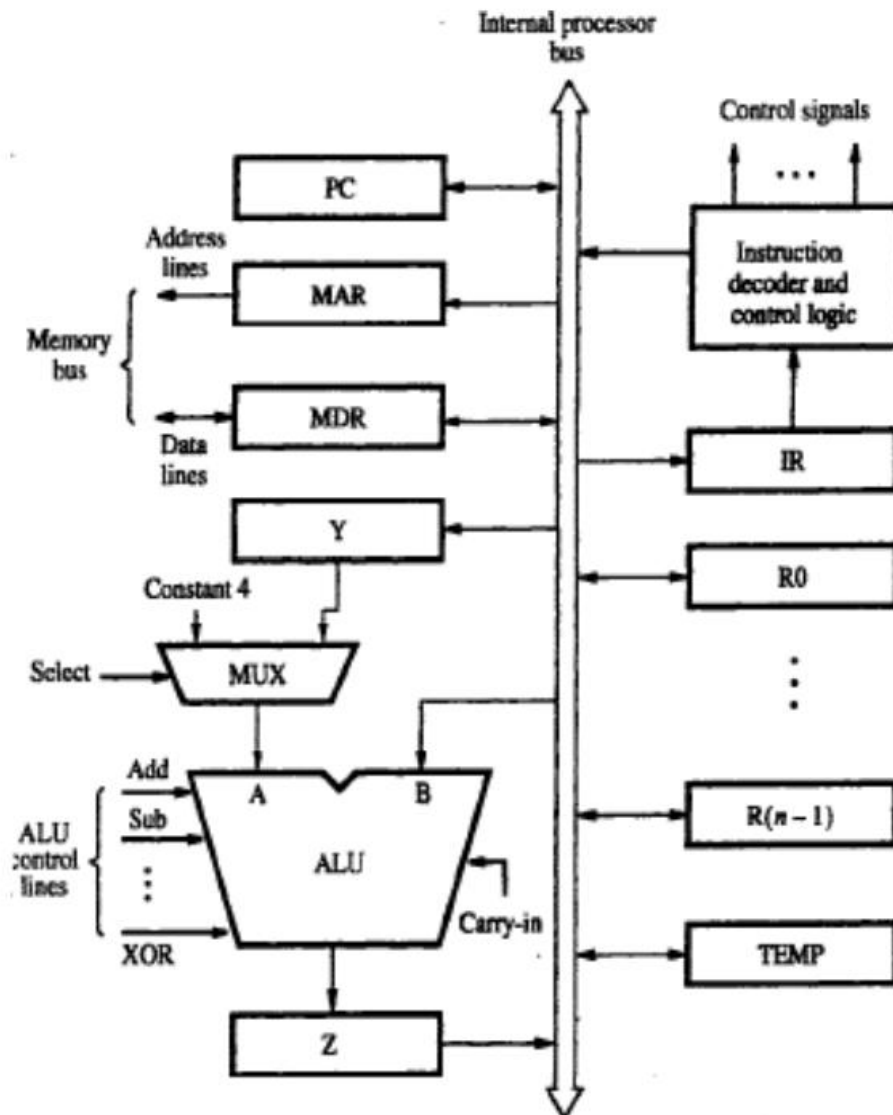
An instruction is executed by performing one or more of the following operations:

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

Disadvantage: Only one data-word can be transferred over the bus in a clock cycle.

Solution: Provide multiple internal paths. Multiple paths allow several data transfers to take place in parallel.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Single bus organization of the datapath inside a processor

5.1.1 Register Transfers

Instruction execution involves a sequence of steps in which data are transferred from one register to another. For each register, two control-signals are used: R_{in} & R_{out} . These are called Gating Signals. $R_{in}=1$ data on bus is loaded into R_i . $R_{out}=1$ content of R_i is placed on bus. $R_{out}=0$, bus can be used for transferring data from other registers.

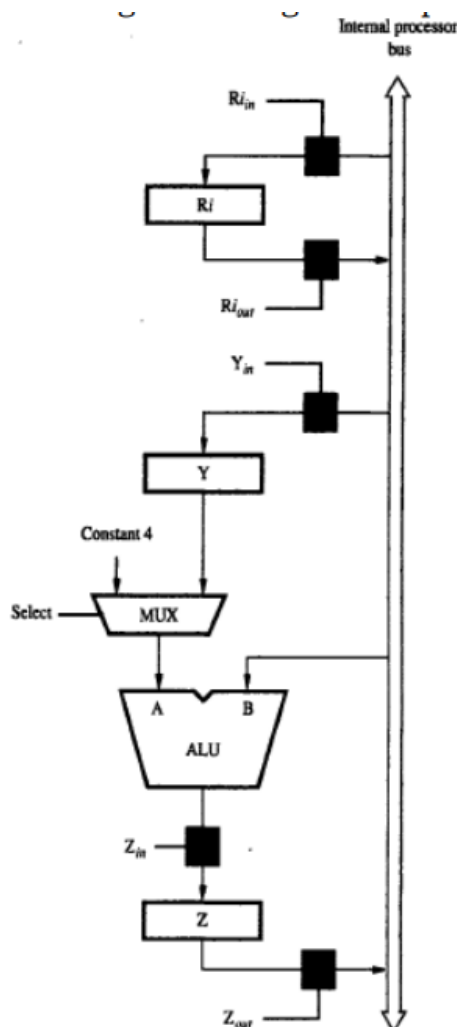
For example, Move R1, R2; This transfers the contents of register R1 to register R2. This can be accomplished as follows:

- 1) Enable the output of registers R1 by setting $R1_{out}$ to 1. This place the contents of R1 on processor-bus.
- 2) Enable the input of register R2 by setting $R2_{in}$ to 1.

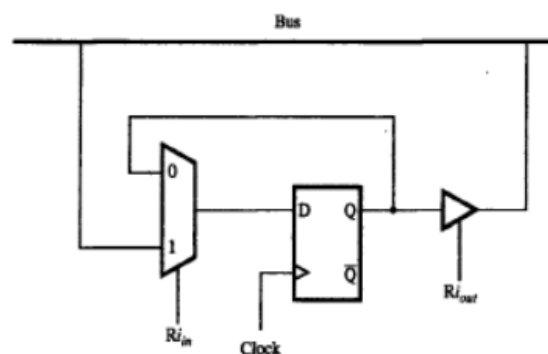
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

This loads data from processor-bus into register R4. All operations and data transfers within the processor take place within time periods defined by the processor-clock. The control-signals that govern a particular transfer are asserted at the start of the clock cycle.

Input & Output Gating for one Register Bit: A 2-input multiplexer is used to select the data applied to the input of an edge triggered D flip-flop. $R_{i_{in}}=1$ mux selects data on bus. This data will be loaded into flip-flop at rising-edge of clock. $R_{i_{in}}=0$ mux feeds back the value currently stored in flip-flop. Q output of flip-flop is connected to bus via a tri-state gate. $R_{i_{out}}=0$ gate's output is in the high-impedance state. $R_{i_{out}}=1$ the gate drives the bus to 0 or 1, depending on the value of Q.



Input Output gating for the registers



Input and output gating for one register bit

5.1.2 Performing ALU operations

The ALU performs arithmetic operations on the 2 operands applied to its A and B inputs. One of the operands is output of MUX; And the other operand is obtained directly from processor-bus. The result (produced by the ALU) is stored temporarily in register Z.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The sequence of operations for $[R3] = [R1] + [R2]$ is as follows: 1) R1out, Yin 2) R2out, SelectY, Add, Zin 3) Zout, R3in

Instruction execution proceeds as follows:

Step 1: Contents from register R1 are loaded into register Y.

Step 2: Contents from Y and from register R2 are applied to the A and B inputs of ALU; Addition is performed & Result is stored in the Z register.

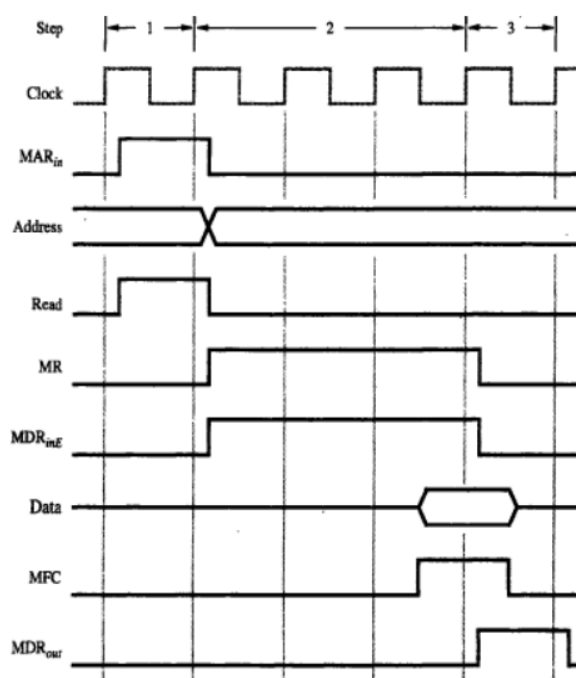
Step 3: The contents of Z register is stored in the R3 register. The signals are activated for the duration of the clock cycle corresponding to that step. All other signals are inactive.

5.1.3 Fetching a word from Memory

To fetch instruction/data from memory, processor transfers required address to MAR. At the same time, processor issues Read signal on control-lines of memory-bus. When requested data are received from memory, they are stored in MDR. From MDR, they are transferred to other registers. The response time of each memory access varies (based on cache miss, memory mapped I/O).

To accommodate this, MFC is used. (MFC Memory Function Completed). MFC is a signal sent from addressed device to the processor. MFC informs the processor that the requested operation has been completed by addressed device. Consider the instruction Move (R1),R2. The sequence of steps is:

- 1) R1out, MARin, Read; desired address is loaded into MAR & Read command is issued.
- 2) MDRinE, WMFC; load MDR from memory-bus & Wait for MFC response from memory.
- 3) MDRout, R2in; load R2 from MDR. where WMFC=control-signal that causes processor's control. circuitry to wait for arrival of MFC signal.



Timing of a memory read operation

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

5.1.4 Storing a word in memory

Consider the instruction Move R2, (R1). This requires the following sequence:

- 1) R1_{out}, MAR_{in}; desired address is loaded into MAR.
- 2) R2_{out}, MDR_{in}, Write ;data to be written are loaded into MDR & Write command is issued.
- 3) MDR_{out}E, WMFC ;load data into memory-location pointed by R1 from MDR.

5.2 Execution of a Complete Instruction

Consider the instruction Add (R3), R1 which adds the contents of a memory-location pointed by R3 to register R1. Executing this instruction requires the following actions:

- 1) Fetch the instruction.
- 2) Fetch the first operand.
- 3) Perform the addition &
- 4) Load the result into R1.

Step	Action
1	PC _{out} , MAR _{in} , Read, Select4, Add, Z _{in}
2	Z _{out} , PC _{in} , Y _{in} , WMFC
3	MDR _{out} , IR _{in}
4	R3 _{out} , MAR _{in} , Read
5	R1 _{out} , Y _{in} , WMFC
6	MDR _{out} , SelectY, Add, Z _{in}
7	Z _{out} , R1 _{in} , End

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

Instruction execution proceeds as follows:

Step1: The instruction-fetch operation is initiated by loading contents of PC into MAR & sending a Read request to memory. The Select signal is set to Select4, which causes the Mux to select constant 4. This value is added to operand at input B (PC's content), and the result is stored in Z.

Step2: Updated value in Z is moved to PC. This completes the PC increment operation and PC will now point to next instruction.

Step3: Fetched instruction is moved into MDR and then to IR. The step 1 through 3 constitutes the Fetch Phase. At the beginning of step 4, the instruction decoder interprets the contents of the IR. This enables the control circuitry to activate the control-signals for steps 4 through 7. The step 4 through 7 constitutes the Execution Phase.

Step4: Contents of R3 are loaded into MAR & a memory read signal is issued.

Step5: Contents of R1 are transferred to Y to prepare for addition.

Step6: When Read operation is completed, memory-operand is available in MDR, and the addition is performed.

Step7: Sum is stored in Z, then transferred to R1. The End signal causes a new instruction fetch cycle to begin by returning to step1.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

5.2.1 Branching Instructions

Instruction execution proceeds as follows:

Step 1-3: The processing starts & the fetch phase ends in step 3.

Step 4: The offset-value is extracted from IR by instruction-decoding circuit. Since the updated value of PC is already available in register Y, the offset X is gated onto the bus, and an addition operation is performed.

Step 5: the result, which is the branch-address, is loaded into the PC. The branch instruction loads the branch target address in PC so that PC will fetch the next instruction from the branch target address. The branch target address is usually obtained by adding the offset in the contents of PC. The offset X is usually the difference between the branch target-address and the address immediately following the branch instruction.

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

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

Control sequence for an unconditional branch instruction.

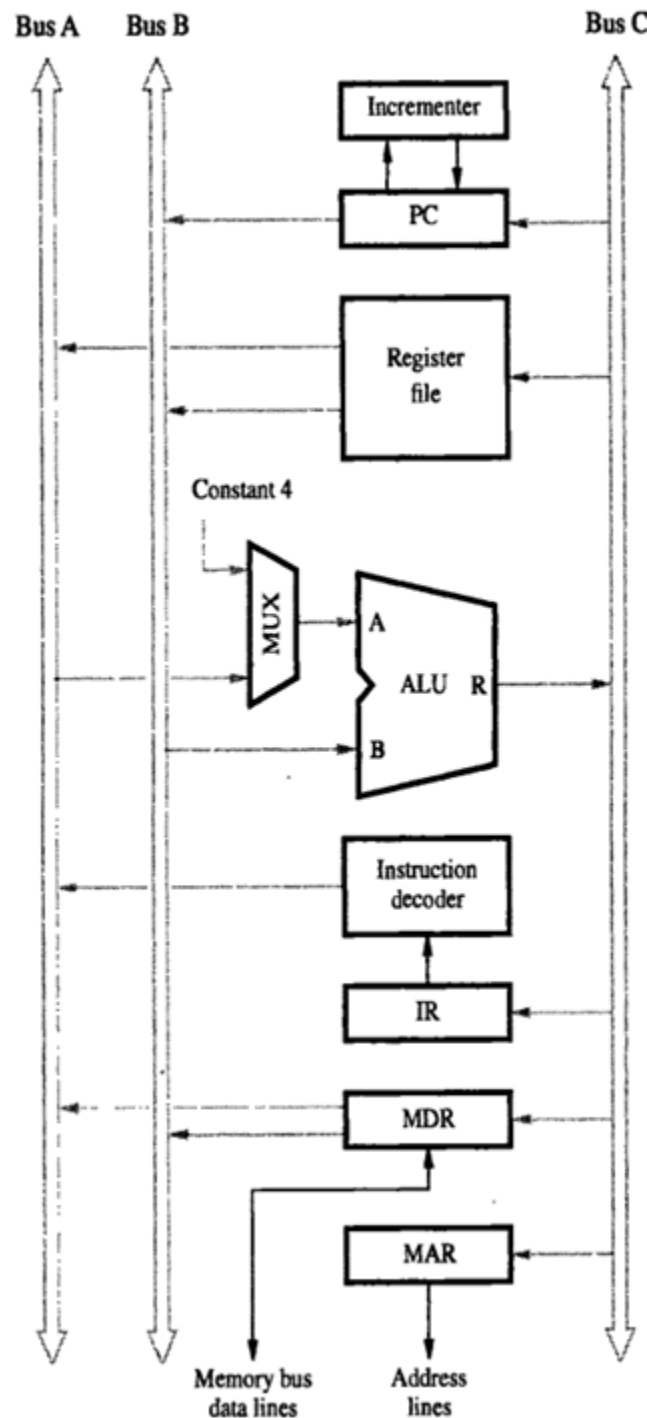
In case of conditional branch, we must check the status of the condition-codes before loading a new value into the PC. e.g.: Offset-field-of-IR_{out}, Add, Z_{in}, If N=0 then End If N=0, processor returns to step 1 immediately after step 4. If N=1, step 5 is performed to load a new value into PC.

5.3 Multiple Bus Organization

Following figure depicts a three-bus structure used to connect the registers and the ALU of a processor. All general-purpose registers are combined into a single block called the register file. In VLSI technology, the most efficient way to implement a number of registers is in the form of an array of memory cells similar to those used in the implementation of random-access memories (RAMs). The register file in Figure is said to have three ports. There are two outputs, allowing the contents of two different registers to be accessed simultaneously and have their contents placed on buses A and B. The third port allows the data on bus C to be loaded into a third register during the same clock cycle.

Buses A and B are used to transfer the source operands to the A and B inputs of the ALU, where an arithmetic or logic operation may be performed. The result is transferred to the destination over bus C. If needed, the ALU may simply pass one of its two input operands unmodified to bus C. We will call the ALU control signals for such an operation $R=A$ or $R=B$.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Three-bus organization of the data path

A second feature in Figure is the introduction of the Incrementer unit, which is used to increment the PC by 4. Using the Incrementer eliminates the need to add 4 to the PC using the main ALU. The source for the constant 4 at the ALU input multiplexer is still useful. It can be used to increment other addresses, such as the memory addresses in LoadMultiple and StoreMultiple instructions.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

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

Control sequence for the instruction Add R4, R5, R6 for the three-bus organization

Consider the three-operand instruction

Add R4,R5,R6

In step 1, the contents of the PC are passed through the ALU, using the R-B control signal, and loaded into the MAR to start a memory read operation. At the same time the PC is incremented by 4. Note that the value loaded into MAR is the original contents of the PC. The incremented value is loaded into the PC at the end of the clock cycle and will not affect the contents of MAR. In step 2, the processor waits for MFC and loads the data received into MDR, then transfers them to IR in step 3. Finally, the execution phase of the instruction requires only one control step to complete, step 4.

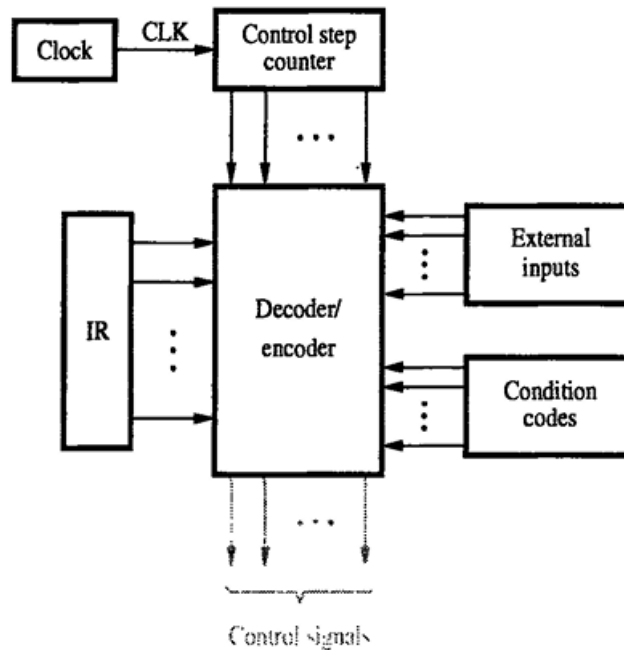
5.4 Hardwired Control

To execute instructions, the processor must have some means of generating the control signals needed in the proper sequence. Computer designers use a wide variety of techniques to solve this problem. The approaches used fall into one of two categories: hardwired control and microprogrammed control.

Consider the sequence of control signals given for instruction Add (R3),R1. Each step in this sequence is completed in one clock period. A counter may be used to keep track of the control steps, as shown in Figure below. Each state, or count, of this counter corresponds to one control step. The required control signals are determined by the following information:

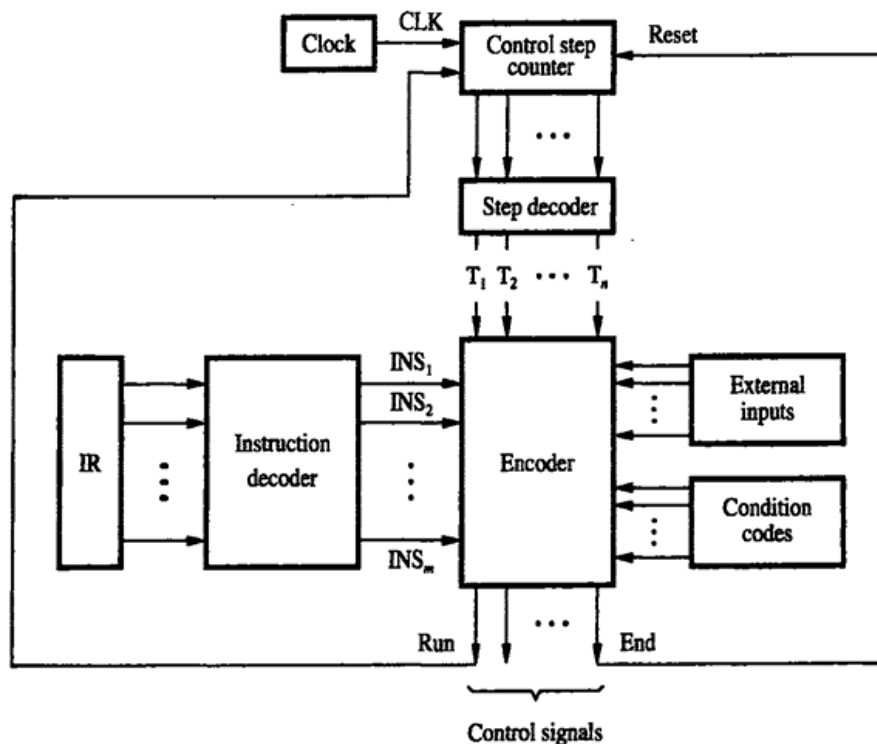
- Contents of the control step counter
- Contents of the instruction register
- Contents of the condition code flags
- External input signals, such as MFC and interrupt requests

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



Control unit organization

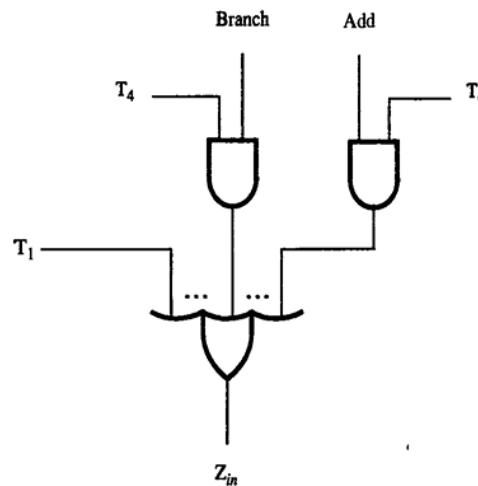
To gain insight into the structure of the control unit, we start with a simplified view of the hardware involved. The decoder/encoder block in Figure is a combinational circuit that generates the required control outputs, depending on the state of all its inputs. By separating the decoding and encoding functions, we obtain the more detailed block diagram.



Separation of the decoding and encoding functions

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

The step decoder provides a separate signal line for each step, or time slot, in the control sequence. Similarly, the output of the instruction decoder consists of a separate line for each machine instruction. For any instruction loaded in the IR, one of the output lines INS, through INS, is set to 1, and all other lines are set to 0. (For design details of decoders, refer to Appendix A.) The input signals to the encoder block in following figure are combined to generate the individual control signals Yin, PCout, Add, End, and so on. An example of how the encoder generates the Zin control signal for the processor organization is given in following figure.



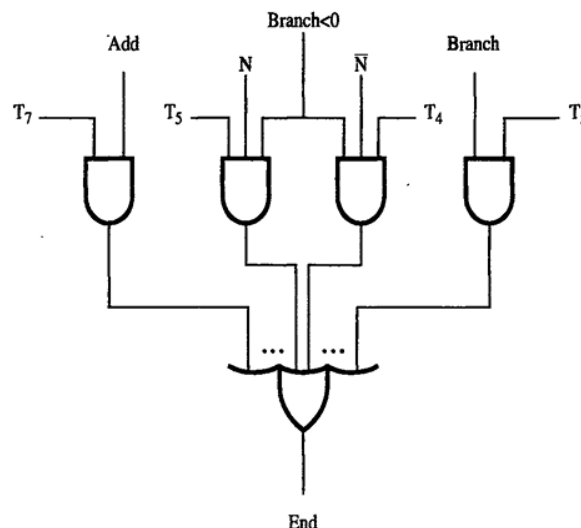
Generation of the Zin control signal for the processor

This circuit implements the logic function

$$Zin = T1 + T6 * ADD + T4 * BR + \dots \quad [1]$$

This signal is asserted during time slot T1 for all instructions, during T6 for an Add instruction, during T4 for an unconditional branch instruction, and so on. As another example, Figure below gives a circuit that generates the End control signal from the logic function

$$End = T7 * ADD + T5 * BR + (T5 * N + T4 * \bar{N}) * BRN + \dots \quad [2]$$

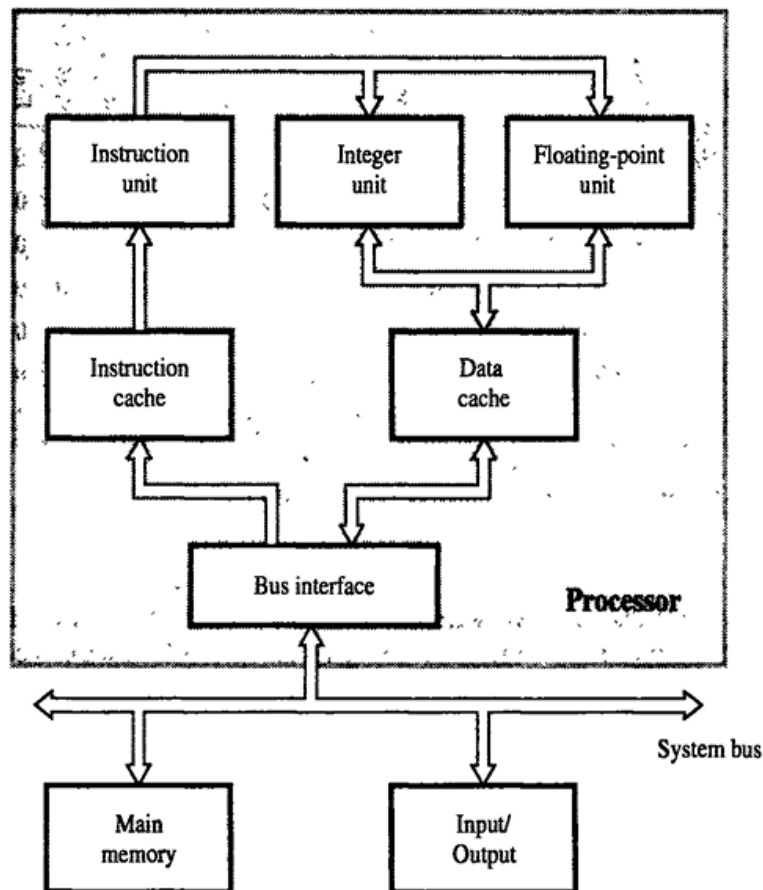


Generation of the Zin control signal for the processor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

5.4.1 A Complete Processor

A complete processor can be designed using the structure shown in following figure. This structure has an instruction unit that fetches instructions from an instruction cache or from the main memory when the desired instructions are not already in the cache. It has separate processing units to deal with integer data and floating-point data. A data cache is inserted between these units and the main memory. Using separate caches for instructions and data is common practice in many processors today. Other processors use a single cache that stores both instructions and data. The processor is connected to the system bus and hence, to the rest of the computer, by means of a bus interface.



Block diagram of a complete processor

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Chapter 2: Pipelining

5.5 Basic concepts

The speed of execution of programs is influenced by many factors. One way to improve performance is to use faster circuit technology to build the processor and the main memory. Another possibility is to arrange the hardware so that more than one operation can be performed at the same time. In this way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.

Pipelining is a particularly effective way of organizing concurrent activity in a computer system. The basic idea is very simple. It is frequently encountered in manufacturing plants, where pipelining is commonly known as an assembly-line operation. Readers are undoubtedly familiar with the assembly line used in car manufacturing. The first station in an assembly line may prepare the chassis of a car, the next station adds the body, the next one installs the engine, and so on. While one group of workers is installing the engine on one car, another group is fitting a car body on the chassis of another car, and yet another group is preparing a new chassis for a third car. It may take days to complete work on a given car, but it is possible to have a new car rolling off the end of the assembly line every few minutes.

The processor executes a program by fetching and executing instructions, one after the other. Let F_i and E_i refer to the fetch and execute steps for instruction I_i . Execution of a program consists of a sequence of fetch and execute steps, as shown in Figure a.

Now consider a computer that has two separate hardware units, one for fetching instructions and another for executing them, as shown in Figure b. The instruction fetched by the fetch unit is deposited in an intermediate storage buffer, B_1 . This buffer is needed to enable the execution unit to execute the instruction while the fetch unit is fetching the next instruction. The results of execution are deposited in the destination location specified by the instruction.

The computer is controlled by a clock whose period is such that the fetch and execute steps of any instruction can each be completed in one clock cycle. Operation of the computer proceeds as in Figure 8.1c. In the first clock cycle, the fetch unit fetches an instruction I_1 (step F_1) and stores it in buffer B_1 at the end of the clock cycle. In the second clock cycle, the instruction fetch unit proceeds with the fetch operation for instruction I_2 (step F_2). Meanwhile, the execution unit performs the operation specified by instruction I_1 , which is available to it in buffer B_1 (step E_1). By the end of the second clock cycle, the execution of instruction I_1 is completed and instruction I_2 is available. Instruction I_2 is stored in B_1 , replacing I_1 which is no longer needed. Step E_2 is performed by the execution unit during the third clock cycle, while instruction I_3 is being fetched by the fetch unit. In this manner, both the fetch and execute units are kept busy all the time. If the pattern in Figure c can be sustained for a long time, the completion rate of instruction execution will be twice that achievable by the sequential operation depicted in Figure a.

In summary, the fetch and execute units in Figure b constitute a two-stage pipeline in which each stage performs one step in processing an instruction. An inter-stage storage buffer, B_1 , is

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

needed to hold the information being passed from one stage to the next. New information is loaded into this buffer at the end of each clock cycle.

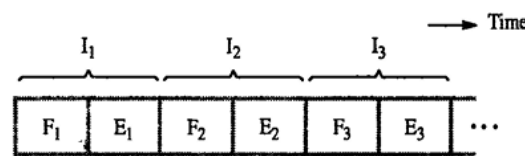
The processing of an instruction need not be divided into only two steps. For example, a pipelined processor may process each instruction in four steps, as follows:

F Fetch: read the instruction from the memory.

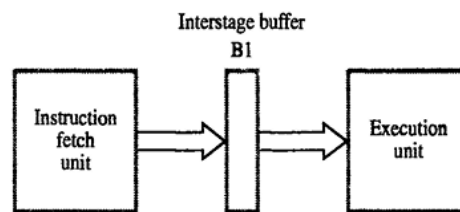
D Decode: decode the instruction and fetch the source operand(s).

E Execute: perform the operation specified by the instruction.

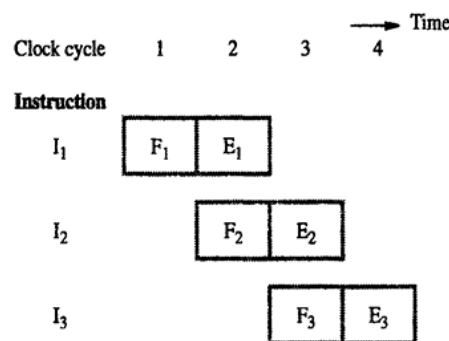
W Write: store the result in the destination location.



(a) Sequential execution



(b) Hardware organization



(c) Pipelined execution

Basic idea of instruction pipelining

The sequence of events for this case is shown in Figure a. Four instructions are in progress at any given time. This means that four distinct hardware units are needed, as shown in Figure b. These units must be capable of performing their tasks simultaneously and without interfering with one another. Information is passed from one unit to the next through a storage buffer. As an instruction progresses through the pipeline, all the information needed by the stages downstream must be passed along.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

For example, during clock cycle 4, the information in the buffers is as follows:

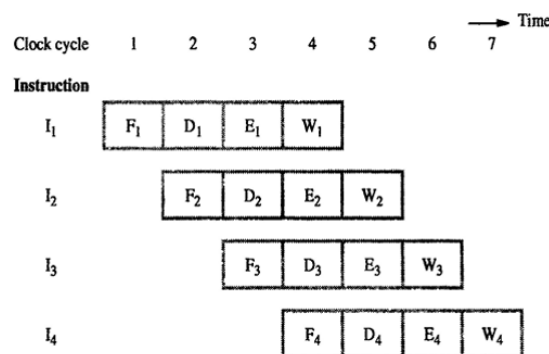
- Buffer B1 holds instruction 13, which was fetched in cycle 3 and is being decoded by the instruction-decoding unit.
- Buffer B2 holds both the source operands for instruction 12 and the specification of the operation to be performed. This is the information produced by the decoding hardware in cycle 3. The buffer also holds the information needed for the write step of instruction 12 (step W₂). Even though it is not needed by stage E, this information must be passed on to stage W in the following clock cycle to enable that stage to perform the required Write operation.
- Buffer B3 holds the results produced by the execution unit and the destination information for instruction I.

5.5.1 Role of Cache memory

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

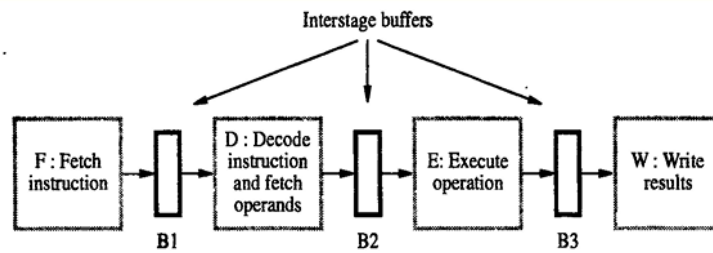
This consideration is particularly important for the instruction fetch step, which is assigned one clock period in Figure a. The clock cycle has to be equal to or greater than the time needed to complete a fetch operation. However, the access time of the main memory may be as much as ten times greater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

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



(a) Instruction execution divided into four steps

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING



(b) Hardware organization

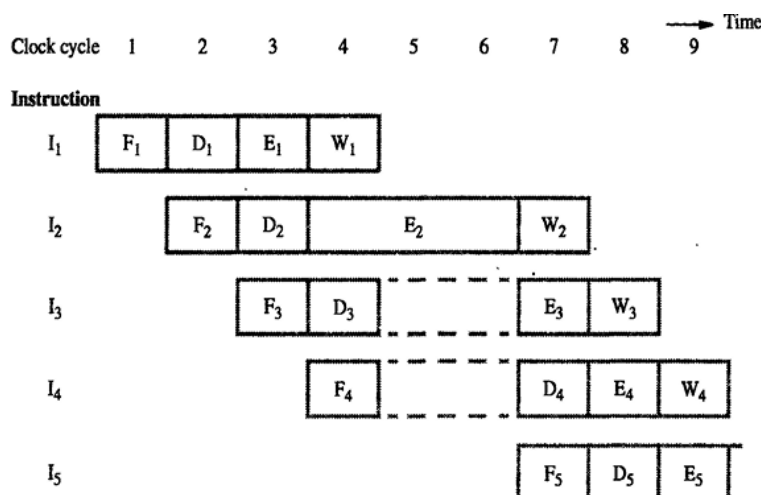
A 4-stage pipeline

5.6 Pipeline Performance

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

For a variety of reasons, one of the pipeline stages may not be able to complete its processing task for a given instruction in the time allotted. For example, stage E in the four-stage pipeline of Figure b is responsible for arithmetic and logic operations, and one clock cycle is assigned for this task. Although this may be sufficient for most operations, some operations, such as divide, may require more time to complete.

Figure below shows an example in which the operation specified in instruction 12 requires three cycles to complete, from cycle 4 through cycle 6. Thus, in cycles 5 and 6, the Write stage must be told to do nothing, because it has no data to work with. Meanwhile, the information in buffer B2 must remain intact until the Execute stage has completed its operation. This means that stage 2 and, in turn, stage 1 are blocked from accepting new instructions because the information in B1 cannot be overwritten. Thus, steps D4 and F5 must be postponed as shown.

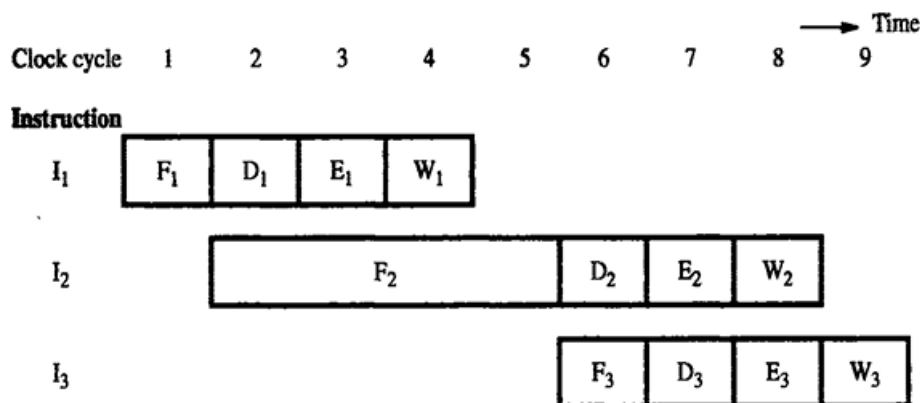


Effect of an execution operation taking more than one clock cycle

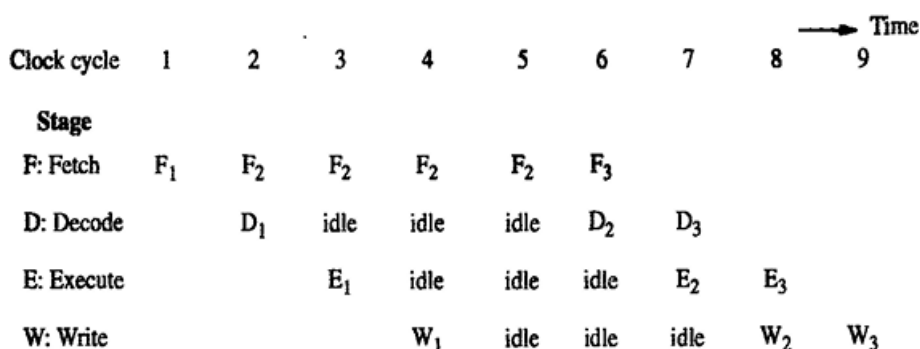
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Pipelined operation here is said to have been stalled for two clock cycles. Normal pipelined operation resumes in cycle 7. Any condition that causes the pipeline to stall is called a hazard. We have just seen an example of a data hazard. A data hazard is any condition in which either the source or the destination operands of an instruction are not available at the time expected in the pipeline. As a result some operation has to be delayed, and the pipeline stalls.

The pipeline may also be stalled because of a delay in the availability of an instruction. For example, this may be a result of a miss in the cache, requiring the instruction to be fetched from the main memory. Such hazards are often called control hazards or instruction hazards. The effect of a cache miss on pipelined operation is illustrated in Figure below. Instruction I₁ is fetched from the cache in cycle 1, and its execution proceeds normally. However, the fetch operation for instruction I₂, which is started in cycle 2, results in a cache miss. The instruction fetch unit must now suspend any further fetch re-requests and wait for I₂ to arrive. We assume that instruction I₂ is received and loaded into buffer B1 at the end of cycle 5. The pipeline resumes its normal operation at that point.



(a) Instruction execution steps in successive clock cycles



(b) Function performed by each processor stage in successive clock cycles

Pipeline stall caused by a cache miss in F₂

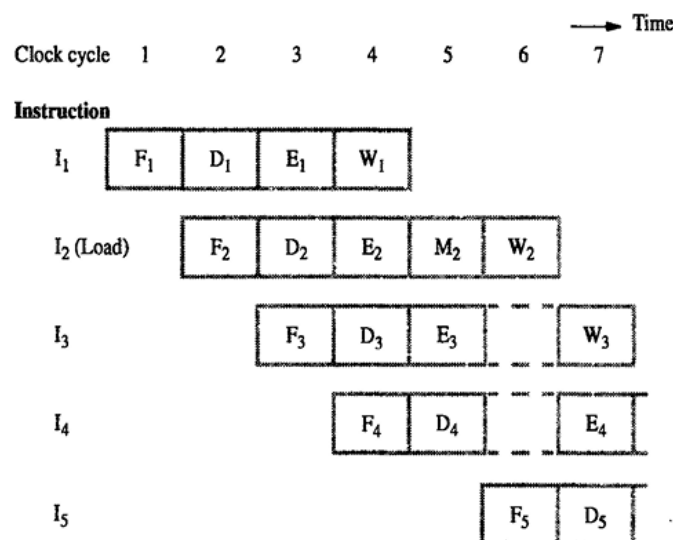
An alternative representation of the operation of a pipeline in the case of a cache miss is shown in Figure b. This figure gives the function performed by each pipeline stage in each clock cycle.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

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

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

An example of a structural hazard is shown in Following figure. This figure shows how the load instruction: Load X(R1),R2 can be accommodated in our example 4-stage pipeline. The memory address, $X+[R1]$, is computed in step E_2 in cycle 4, then memory access takes place in cycle 5. The operand read from memory is written into register R2 in cycle 6. This means that the execution step of this instruction takes two clock cycles (cycles 4 and 5). It causes the pipeline to stall for one cycle, because both instructions 12 and 13 require access to the register file in cycle 6. Even though the instructions and their data are all available, the pipeline is stalled because one hardware resource, the register file, cannot handle two operations at once. If the register file had two input ports, that is, if it allowed two simultaneous write operations, the pipeline would not be stalled. In general, structural hazards are avoided by providing sufficient hardware resources on the processor chip.



Effect of a load instruction on pipeline timing

It is important to understand that pipelining does not result in individual instructions being executed faster, rather, it is the throughput that increases, where throughput is measured by the rate at which instruction execution is completed. Any time one of the stages in the pipeline cannot complete its operation in one clock cycle, the pipeline stalls, and some degradation in performance occurs.