DIGITAL PRINCIPLES AND COMPUTER ORGANIZATION

UNIT – 1

NUMBER REPRESENTATION: NUMBER SYSTEM

A number system is a system representing numbers. It is also called the system of numeration and it defines a set of values to represent a quantity. These numbers are used as digits and the most common ones are 0 and 1, that are used to represent binary numbers. Digits from 0 to 9 are used to represent other types of number systems.

A number system is defined as the representation of numbers by using digits or other symbols in a consistent manner. The value of any digit in a number can be determined by a digit, its position in the number, and the base of the number system. The numbers are represented in a unique manner and allow us to operate arithmetic operations like addition, subtraction, and division.

There are different types of number systems in which the four main types are:

- Binary number system (Base - 2)
- Octal number system (Base - 8)
- Decimal number system (Base - 10)
- Hexadecimal number system (Base - 16)

Binary Number System

The binary number system uses only two digits: 0 and 1. The numbers in this system have a base of 2. Digits 0 and 1 are called bits and 8 bits together make a byte. The data in computers is stored in terms of bits and bytes. The binary number system does not deal with other numbers such as 2,3,4,5 and so on. For example: 10001_2, 111101_2, 1010101_2 are some examples of numbers in the binary number system.

Octal Number System

The octal number system uses eight digits: 0,1,2,3,4,5,6 and 7 with the base of 8. The advantage of this system is that it has lesser digits when compared to several other systems, hence, there would be fewer

computational errors. Digits like 8 and 9 are not included in the octal number system. Just as the binary, the octal number system is used in minicomputers but with digits from 0 to 7. For example: 35_8, 23_8, 141_8 are some examples of numbers in the octal number system.

Decimal Number System

The decimal number system uses ten digits: 0,1,2,3,4,5,6,7,8 and 9 with the base number as 10. The decimal number system is the system that we generally use to represent numbers in real life. If any number is represented without a base, it means that its base is 10. For example: 723_{10}, 32_{10}, 4257_{10} are some examples of numbers in the decimal number system.

Hexadecimal Number System

The hexadecimal number system uses sixteen digits/alphabets: 0,1,2,3,4,5,6,7,8,9 and A,B,C,D,E,F with the base number as 16. Here, A-F of the hexadecimal system means the numbers 10-15 of the decimal number system respectively. This system is used in computers to reduce the large-sized strings of the binary system. For example: $7B3_{16}$, $6F_{16}$, $4B2A_{16}$ are some examples of numbers in the hexadecimal number system.

Conversion Rules of Number Systems

A number can be converted from one number system to another number system. Like binary numbers can be converted to octal numbers and vice versa, octal numbers can be converted to decimal numbers and vice versa and so on. Let us see the steps required in converting number systems.

Conversion of Binary / Octal / Hexadecimal Number Systems to Decimal Number System

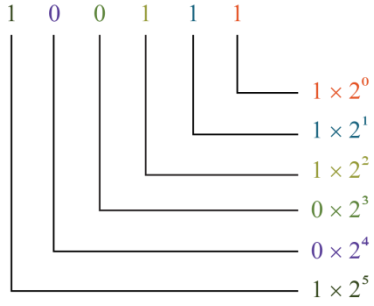
To convert a number from the binary/octal/hexadecimal system to the decimal system, we use the following steps. The steps are shown by an example of a number in the binary system.

Example: Convert 100111_2 into the decimal system.

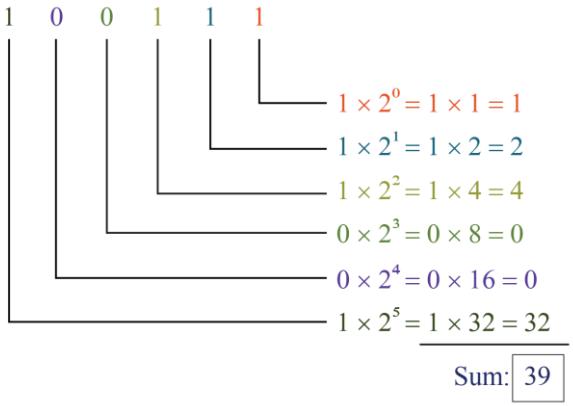
Solution:

Step 1: Identify the base of the given number. Here, the base of 100111_2 is 2.

Step 2: Multiply each digit of the given number, starting from the rightmost digit, with the exponents of the base. The exponents should start with 0 and increase by 1 every time as we move from right to left. Since the base is 2 here, we multiply the digits of the given number by 2^0, 2^1, 2^2 , and so on from right to left.



Step 3: We just simplify each of the above products and add them.



Here, the sum is the equivalent number in the decimal number system of the given number. Or, we can use the following steps to make this process simplified.

$100111 = (1\times2^5) + (0\times2^4) + (0\times2^3) + (1\times2^2) + (1\times2^1) + (1\times2^0)$

$= (1\times32) + (0\times16) + (0\times8) + (1\times4) + (1\times2) + (1\times1)$

$= 32 + 0 + 0 + 4 + 2 + 1$

$= 39$

Thus, $100111_2 = 39_{10}$.

Conversion of Decimal Number System to Binary / Octal / Hexadecimal Number System

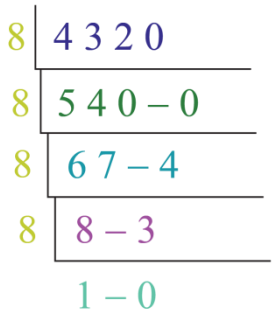
To convert a number from the decimal number system to binary/octal/hexadecimal number system, we use the following steps. The steps are shown on how to convert a number from the decimal system to the octal system.

Example: Convert 4320_{10} into the octal system.

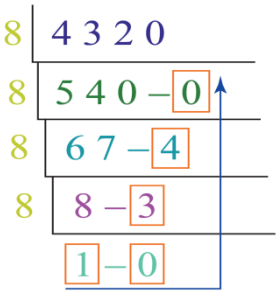
Solution:

Step 1: Identify the base of the required number. Since we have to convert the given number into the octal system, the base of the required number is 8.

Step 2: Divide the given number by the base of the required number and note down the quotient and the remainder in the quotient-remainder form. Repeat this process (dividing the quotient again by the base) until we get the quotient less than the base.



Step 3: The given number in the octal number system is obtained just by reading all the remainders and the last quotient from bottom to top.



Therefore, $4320_{10} = 10340_{8}$.

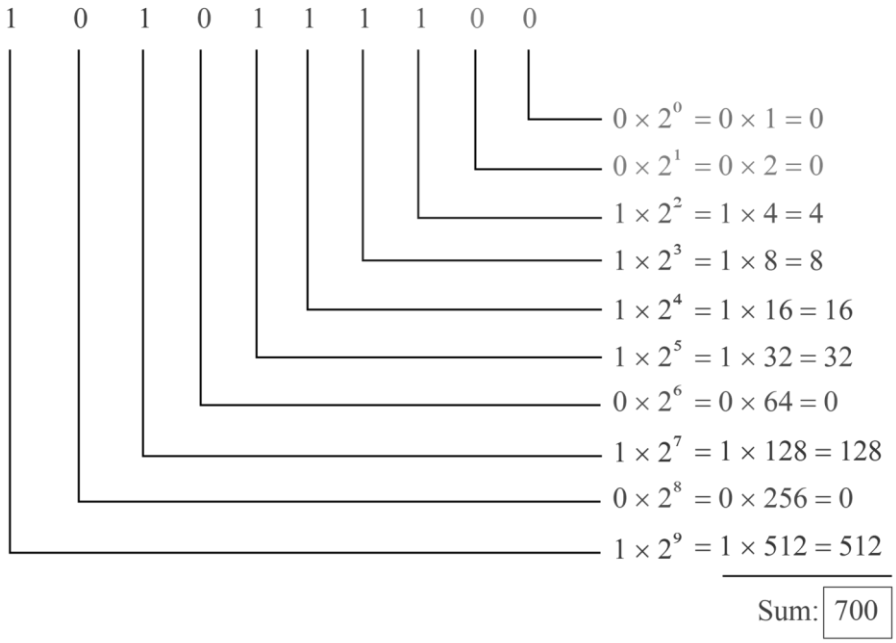
Conversion from One Number System to Another Number System

To convert a number from one of the binary/octal/hexadecimal systems to one of the other systems, we first convert it into the decimal system, and then we convert it to the required systems by using the above-mentioned processes.

Example: Convert 1010111100_2 to the hexadecimal system.

Solution:

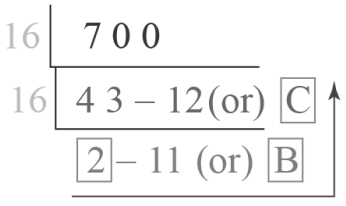
Step 1: Convert this number to the decimal number system as explained in the above process.



Thus, $1010111100_2 = 700_{10} \rightarrow (1)$.

Step 2: Convert the above number (which is in the decimal system), into the required number system.

Here, we have to convert 700_{10} into the hexadecimal system using the above-mentioned process. It should be noted that in the hexadecimal system, the numbers 11 and 12 are written as B and C respectively.

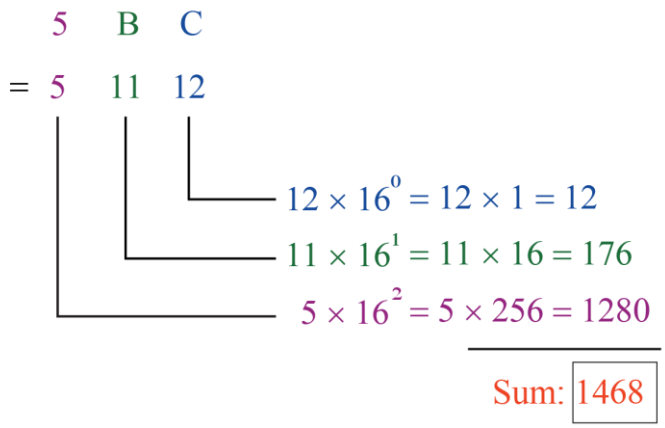


Thus, $700_{10} = 2BC_{16} \rightarrow (2)$.

From the equations (1) and (2), $1010111100_2 = 2BC_{16}$.

Example 2: Convert $5BC_{16}$ into the decimal system.

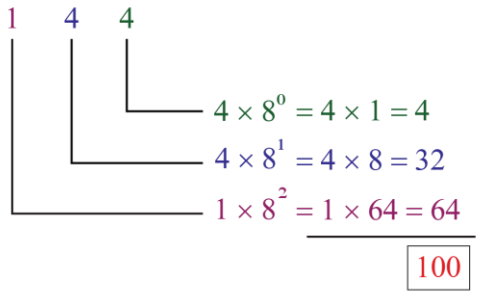
Solution: $5BC_{16}$ is in the hexadecimal system. We know that B=11 and C= 12 in the hexadecimal system. So we get the equivalent number in the decimal system using the following process:



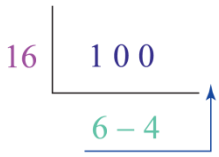
Thus, $5BC_{16} = 1468_{10}$.

Example 3: Convert 144_8 into the hexadecimal system.

Solution: The base of 144_8 is 8. First, we will convert this number into the decimal system as follows:



Thus, $144_8 = 100_{10} \rightarrow (1)$. Now we will convert this into the hexadecimal system as follows:



Thus, $100_{10} = 64_{16} \rightarrow (2)$.

From the equations (1) and (2), we can conclude that: $144_8 = 64_{16}$.

BCD or Binary Coded Decimal

Binary Coded Decimal, or **BCD**, is another process for converting decimal numbers into their binary equivalents.

It is a form of binary encoding where each digit in a decimal number is represented in the form of bits.

- This encoding can be done in either 4-bit or 8-bit (usually 4-bit is preferred).
- It is a fast and efficient system that converts the decimal numbers into binary numbers as compared to the existing binary system.
- These are generally used in digital displays where is the manipulation of data is quite a task.
- Thus BCD plays an important role here because the manipulation is done treating each digit as a separate single sub-circuit.

Many decimal values, have an infinite place-value representation in binary but have a finite place-value in binary-coded decimal. For example, 0.2 in binary is .001100… and in BCD is 0.0010. It avoids fractional errors and is also used in huge financial calculations.

Truth Table for Binary Coded Decimal

DECIMAL NUMBER	BCD
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

In the **BCD numbering system**, the given decimal number is segregated into chunks of four bits for each decimal digit within the number. Each decimal digit is converted into its direct binary form (usually represented in 4-bits).

- It is noticeable that the BCD is nothing more than a binary representation of each digit of a decimal number.
- It cannot be ignored that the BCD representation of the given decimal number uses extra bits, which makes it heavy-weighted.

Example :

Convert the following binary numbers: 1001_2, 1010_2, 1000111_2 and 10100111000.101_2 into their decimal equivalents.

$1001_2 = 1001_{BCD} = 9_{10}$

1010_2 = this will produce an error as it is decimal 10_{10} and not a valid BCD number

$1000111_2 = 0100\ 0111_{BCD} = 47_{10}$

$10100111000.101_2 = 0101\ 0011\ 0001.1010_{BCD} = 538.625_{10}$

The conversion of BCD-to-decimal or decimal-to-BCD is a relatively straight forward task but we need to remember that BCD numbers are decimal numbers and not binary numbers, even though they are represented using bits. The BCD representation of a decimal number is important to understand, because microprocessor based systems used by most people needs to be in the decimal system.

While BCD is easy to code and decode, it is not an efficient way to store numbers. In the standard 8421 BCD encoding of decimal numbers, the number of individual data bits needed to represent a given decimal number will always be greater than the number of bits required for an equivalent binary encoding.

For example, in binary a three digit decimal number from 0-to-999 requires only 10-bits (1111100111_2), whereas in binary coded decimal, the same number requires a minimum of 12-bits ($0011\ 1110\ 0111_{BCD}$) for the same representation.

The main advantage of the **Binary Coded Decimal** system is that it is a fast and efficient system to convert the decimal numbers into binary numbers as compared to the pure binary system. But the BCD code is wasteful as many of the 4-bit states (10-to-16) are not used but decimal displays have important applications.

EXCESS-3 CODE

The excess-3 code (or XS3) is a non-weighted code used to express code used to express decimal numbers. It is a self-complementary binary coded decimal (BCD) code and numerical system which has biased representation. Excess-3 arithmetic uses different algorithm than normal non-biased BCD or binary positional number system.

Representation of Excess-3 Code

Excess-3 codes are unweighted and can be obtained by adding 3 to each decimal digit then it can be represented by using 4 bit binary number for each digit. An Excess-3 equivalent of a given binary binary number is obtained using the following steps:

- Find the decimal equivalent of the given binary number.
- Add +3 to each digit of decimal number.
- Convert the newly obtained decimal number back to binary number to get required excess-3 equivalent.

You can add 0011 to each four-bit group in binary coded decimal number (BCD) to get desired excess-3 equivalent.

These are following excess-3 codes for decimal digits −

Decimal Digit	BCD Code	Excess-3 Code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000

Decimal Digit	BCD Code	Excess-3 Code
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

The codes 0000 and 1111 are not used for any digit.

Example-1 −Convert decimal number 23 to Excess-3 code.

So, according to excess-3 code we need to add 3 to both digit in the decimal number then convert into 4-bit binary number for result of each digit. Therefore,

= 23+33=56 =0101 0110 which is required excess-3 code for given decimal number 23.

Example-2 −Convert decimal number 15.46 into Excess-3 code.

According to excess-3 code we need to add 3 to both digit in the decimal number then convert into 4-bit binary number for result of each digit. Therefore,

= 15.46+33.33=48.79 =0100 1000.0111 1001 which is required excess-3 code for given decimal number 15.46.

Converting into Binary Coded Decimal (BCD) codes

One should note that to given Excess-3 code, the equivalent decimal number can be determined by splitting number into 4-bit group starting from least significant for integer part and from leftmost digit for fractional part. Then subtract 0011 (=3) from each four-bit group that will be binary decimal digit (BCD) form of that number. Now you can also convert this BCD code into decimal number by converting each 4-bit group into decimal digit.

Example − Convert Excess-3 code 1001001 into BCD and decimal number.

So, grouping 4-bit for each group, i.e., 0100 1001 and subtract 0011 0011 from given number. Therefore,

= 0100 1001 -0011 0011 =0001 0110

So, binary coded decimal number is 0001 0110 and decimal number will be 16.

Self-complementary property

Excess-3 code is non-weighted and self complementary code. A self complementary binary codes are always compliment themselves. The complement of a binary number can be obtained from that number by replacing 0's with 1's and 1's with 0's. The sum of binary number and its complement is always equal to decimal 9. In other words, the 1's complement of an excess-3 code is the excess-3 code for the 9's complement of the corresponding decimal number. For example, the excess-3 code for decimal number 5 is 1000 and 1's complement of 1000 is 0111, which is excess-3 code for decimal number 4, and it is 9's complement of number 5.

Advantages of Excess-3 Codes

These are following advantages of Excess-3 codes,

- These are unweighted binary decimal codes.
- These are self-complementary codes.
- These use biased representation.
- The codes 0000 and 1111 are not used for any digit which is an advantage for memory organization as these codes can cause fault in transmission line.
- It has no limitation, and it considerably simplifies arithmetic operations.

Example 1: Decimal number 31

1. We find the BCD code of each digit of the decimal number.

Digit	**BCD**
3	0011
1	0001

2) Then, we add 0011 in both of the BCD code.

Decimal	**BCD**	**Excess-3**
3	0011+0011	0110
1	0001+0011	0100

3. So, the excess-3 code of the decimal number 31 is **0110 0100**

Example 2: Decimal number 81.61

1. We find the BCD code of each digit of the decimal number.

Digit	BCD
8	1000
1	0001
6	0110
1	0001

2) Then, we add 0011 in both of the BCD code.

Decimal	BCD	Excess-3
8	1000+0011	1011
1	0001+0011	0100
6	0110+0011	1001

3) So, the excess-3 code of the decimal number 81.61 is **1011 0100.1001 0100**

GRAY CODE

The reflected binary code or Gray code is an ordering of the binary numeral system such that two successive values differ in only one bit (binary digit). Gray codes are very useful in the normal sequence of binary numbers generated by the hardware that may cause an error or ambiguity during the transition from one number to the next. So, the Gray code can eliminate this problem easily since only one bit changes its value during any transition between two numbers.

Gray code is not weighted that means it does not depends on positional value of digit. This cyclic variable code that means every transition from one value to the next value involves only one bit change.

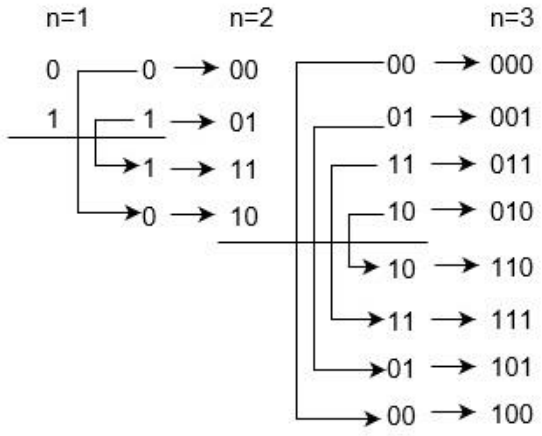
Gray code also known as reflected binary code, because the first (n/2) values compare with those of the last (n/2) values, but in reverse order.

Constructing an *n*-bit Gray code

n-bit Gray code can be generated recursively using reflect and prefix method which is explained as following below.

- Generate code for n=1: 0 and 1 code.
- Take previous code in sequence: 0 and 1.
- Add reversed codes in the following list: 0, 1, 1 and 0.
- Now add prefix 0 for original previous code and prefix 1 for new generated code: 00, 01, 11, and 10.

Therefore, Gray code 0 and 1 are for Binary number 0 and 1 respectively. Gray codes: 00. 01, 11, and 10 are for Binary numbers: 00, 01, 10, and 11 respectively. Similarly you can construct Gray code for 3 bit binary numbers:



Therefore, Gray codes are as following below,

For n = 1 bit		**For n = 2 bit**		**For n = 3 bit**	
Binary	Gray	Binary	Gray	Binary	Gray
0	0	00	00	000	000
1	1	01	01	001	001
		10	11	010	011
		11	10	011	010

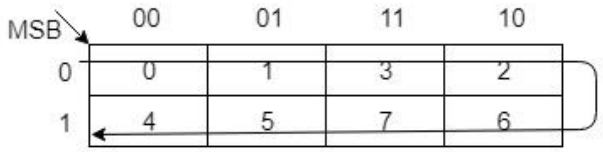
For n = 1 bit	For n = 2 bit	For n = 3 bit	
		100	110
		101	111
		110	101
		111	100

Iterative method of generating $G_{(n+1)}$ from G_n are given below. This is simpler method to contract Gray code of n-bit Binary numbers. Each bit is inverted if the next higher bit of the input value is set to one. The nth Gray code is obtained by computing $n \oplus (floor(n/2))$.

- G_n is unique numbers for the permutation from 0 to (2^n-1).
- G_n is embedded as the first half of $G_{(n+1)}$ and second half as the reverse order of $G_{(n+1)}$.
- Prefix 0 in each digit of first half and 1 in each digit of second half.

The hamming distance of two neighbours Gray codes is always 1 and also first Gray code and last Gray code also has Hamming distance is always 1, so it is also called *Cyclic codes*.

You can construct Gray codes using other methods but they may not be performed in parallel like given above method. For example, 3 bit Gray codes can be contracted using K-map which is given as following below:



Decimal	Binary	Gray Code
0	000	000
1	001	001
2	010	011

Decimal	Binary	Gray Code
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Types of Gray Codes

There are also other types of Gray codes, like Beckett-Gray code, Single track Gray codes etc.

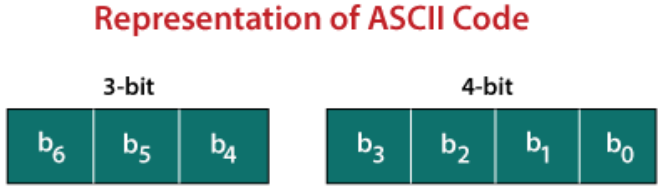
- N-ary Gray code, where non-Boolean values are included like sequences of 1, 2, 3.
- Two dimensional (n,k) Gray codes are used for error correction.
- Balanced Gray codes has equal transition counts.

Uses of Gray codes

Gray codes are used in rotary and optical encoders, Karnaugh maps, and error detection.

ASCII Code

The ASCII stands for American Standard Code for Information Interchange. The ASCII code is an alphanumeric code used for data communication in digital computers. The ASCII is a 7-bit code capable of representing 2^7 or 128 number of different characters. The ASCII code is made up of a three-bit group, which is followed by a four-bit code.



- The ASCII Code is a 7 or 8-bit alphanumeric code.

- This code can represent 127 unique characters.
- The ASCII code starts from 00h to 7Fh. In this, the code from 00h to 1Fh is used for control characters, and the code from 20h to 7Fh is used for graphic symbols.
- The 8-bit code holds ASCII, which supports 256 symbols where math and graphic symbols are added.
- The range of the extended ASCII is 80h to FFh.

The ASCII characters are classified into the following groups:

Additional Features of the ASCII code

· Each code word consists of seven symbols. Hence, we have a total of 2^7(=128) possible combinations, to represent alphabets, punctuation marks, and numbers.

· It is divided into two parts. The first part consists of three bits and the second part consists of four bits. This is illustrated in Table 1.16.

Table 1.16 Organization of ASCII code

First part (3 bits)	Second part (4 bits)
xxx	xxxx

In Table 1.16, x's represent bit **0** or **1**. The 3-bit first part is used to identify whether the 4-bits following these 3-bits represent letters, numerals, or punctuation marks. For example the 3-bit combinations of **100** and **101** represent upper-case (capital) letters (A, B, C, etc.). Similarly, bit 3-bit combinations of **110**, and **111** represent lower-case (small) letters (a, b, c, etc.). Table 1.17 shows a few examples of ASCII code.

Table 1.17 A few examples of ASCII code

First Part (3 bits)	Second Part (4 bits)	Letter/ Punctuation / Number concerned
0 0 0	0 0 0 1	Start of Heading
0 0 0	0 0 1 0	Start of text
0 0 0	0 0 1 1	End of text
0 0 0	0 1 0 0	End of transmission

Since there are 128 possible combinations of **1**s and **0**s, all letters, numerals and most of the punctuation marks can be represented by using the ASCII code. Since the length of each code word is much larger than that of the Morse code, its efficiency is much less.

Advantages of ASCII code

- In ASCII coding scheme, $2^7 = 128$ code words are possible. Hence, a large number of symbols, alphabets etc. can be easily represented.
- There is a definite order in which the alphabets, etc. are assigned to each code word.
- Parity bits can be added for error-detection and correction.

Disadvantages ASCII code

- The length of the code is larger; hence, more bandwidth is required for transmission.
- Number of code words in ASCII is insufficient to represent newly invented and added characters and symbols.

EBCDIC

EBCDIC is short for extended binary coded decimal interchange code is eight bits, or one byte, wide. This is a coding system used to represent characters-letters, numerals, punctuation marks, and other symbols in computerized text. A character is represented in EBCDIC by eight bit. EBCDIC mainly used on IBM mainframe and IBM mid-range computer operating systems. Each byte consists of two nibbles, each four bits wide. The first four bits define the class of character, while the second nibble defines the specific character inside that class.

EBCDIC is different from, and incompatible with, the ASCII character set used by all other computers. The EBCDIC code allows for 256 different characters. For personal computers, however, ASCII is the standard. If you want to move text between your computer and a mainframe, you can get a file conversion utility that will convert between EBCDIC and ASCII.

EBCDIC was adapted from the character codes used in IBM's per-electronic PUNCHED CARD machines, which made it less than ideal for modern computers. Among its many inconveniences were the use of non-contiguous codes for the alphabetic characters, and the absence of several punctuation characters such as the square brackets [] used by much modern software.

For example, setting the first nibble to all-ones,1111, defines the character as a number, and the second nibble defines which number is encoded. EBCDIC can code up to 256 different characters.

There have been six or more incompatible versions of EBCDIC, the latest of which do include all the ASCII characters, but also contain characters that are not supported in ASCII.

BINARY ARITHMETIC

Binary arithmetic is essential part of all the digital computers and many other digital system.

Binary Addition

It is a key for binary subtraction, multiplication, division. There are four rules of binary addition.

Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

In fourth case, a binary addition is creating a sum of (1 + 1 = 10) i.e. 0 is written in the given column and a carry of 1 over to the next column.

Example − Addition

0011010 + 001100 = 00100110

```
   1 1          carry
 0 0 1 1 0 1 0  = 26(10)
+0 0 0 1 1 0 0  = 12(10)
---------------
 0 1 0 0 1 1 0  = 38(10)
```

Binary Subtraction

Subtraction and Borrow, these two words will be used very frequently for the binary subtraction. There are four rules of binary subtraction.

Case	A - B	Subtract	Borrow
1	0 - 0	0	0
2	1 - 0	1	0
3	1 - 1	0	0
4	0 - 1	0	1

Example − Subtraction

0011010 - 001100 = 00001110

```
        1 1        borrow
  0 0 1 1 0 1 0  = 26₁₀
- 0 0 0 1 1 0 0  = 12₁₀
-----------------
  0 0 0 1 1 1 0  = 14₁₀
```

Binary Multiplication

Binary multiplication is similar to decimal multiplication. It is simpler than decimal multiplication because only 0s and 1s are involved. There are four rules of binary multiplication.

Case	A x B	Multiplication
1	0 x 0	0
2	0 x 1	0
3	1 x 0	0
4	1 x 1	1

Example − Multiplication

Example:

0011010 x 001100 = 100111000

```
      0 0 1 1 0 1 0  = 26₁₀
    x 0 0 0 1 1 0 0  = 12₁₀
-----------------------
      0 0 0 0 0 0 0
    0 0 0 0 0 0 0
  0 0 1 1 0 1 0
0 0 1 1 0 1 0
-----------------------
0 1 0 0 1 1 1 0 0 0  = 312₁₀
```

Binary Division

Binary division is similar to decimal division. It is called as the long division procedure.

Example − Division

101010 / 000110 = 000111

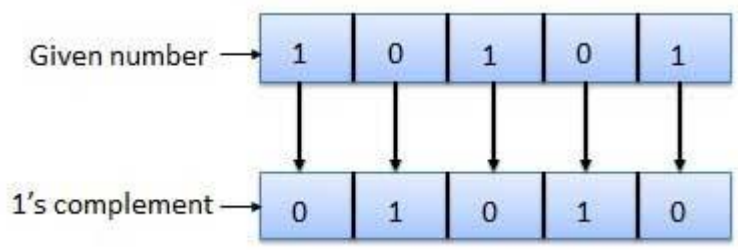
```
              1 1 1      = 7₁₀
          ------------
000110 ) 1 0 1 0 1 0     = 42₁₀
       - 1 1 0           = 6₁₀
       -------
         1 0 0 1
       - 1 1 0
       -------
           1 1 0
         - 1 1 0
         -------
               0
```

1’S COMPLEMENT

s the binary system has base r = 2. So the two types of complements for the binary system are 2's complement and 1's complement.

1's complement

The 1's complement of a number is found by changing all 1's to 0's and all 0's to 1's. This is called as taking complement or 1's complement. Example of 1's Complement is as follows.

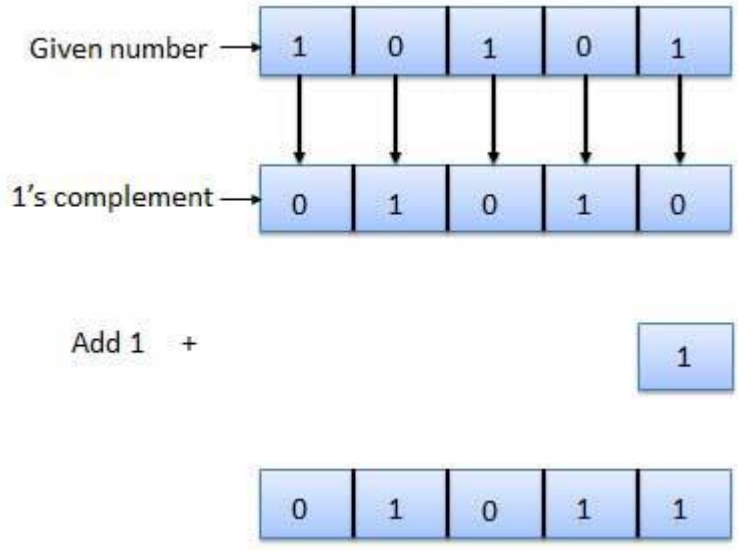


2'S COMPLEMENT

The 2's complement of binary number is obtained by adding 1 to the Least Significant Bit (LSB) of 1's complement of the number.

2's complement = 1's complement + 1

Example of 2's Complement is as follows.



ERROR DETECTING CODES

We know that the bits 0 and 1 corresponding to two different range of analog voltages. So, during transmission of binary data from one system to the other, the noise may also be added. Due to this, there may be errors in the received data at other system.

That means a bit 0 may change to 1 or a bit 1 may change to 0. We can't avoid the interference of noise. But, we can get back the original data first by detecting whether any errors present and then correcting those errors. For this purpose, we can use the following codes.

- Error detection codes
- Error correction codes

Error detection codes − are used to detect the errorss present in the received data bitstreambitstream. These codes contain some bitss, which are included appendedappended to the original bit stream. These codes detect the error, if it is occurred during transmission of the original data bitstream**Example** − Parity code, Hamming code.

Error correction codes − are used to correct the errorss present in the received data bitstream so that, we will get the original data. Error correction codes also use the similar strategy of error detection codes.**Example** − Hamming code.

Therefore, to detect and correct the errors, additional bits are appended to the data bits at the time of transmission.

Parity Code

It is easy to include appendone parity bit either to the left of MSB or to the right of LSB of original bit stream. There are two types of parity codes, namely even parity code and odd parity code based on the type of parity being chosen.

Even Parity Code

The value of even parity bit should be zero, if even number of ones present in the binary code. Otherwise, it should be one. So that, even number of ones present in **even parity code**. Even parity code contains the data bits and even parity bit.

The following table shows the **even parity codes** corresponding to each 3-bit binary code. Here, the even parity bit is included to the right of LSB of binary code.

Binary Code	Even Parity bit	Even Parity Code

000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

Here, the number of bits present in the even parity codes is 4. So, the possible even number of ones in these even parity codes are 0, 2 & 4.

- If the other system receives one of these even parity codes, then there is no error in the received data. The bits other than even parity bit are same as that of binary code.
- If the other system receives other than even parity codes, then there will be an errors in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, even parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

Odd Parity Code

The value of odd parity bit should be zero, if odd number of ones present in the binary code. Otherwise, it should be one. So that, odd number of ones present in **odd parity code**. Odd parity code contains the data bits and odd parity bit.

The following table shows the **odd parity codes** corresponding to each 3-bit binary code. Here, the odd parity bit is included to the right of LSB of binary code.

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

Here, the number of bits present in the odd parity codes is 4. So, the possible odd number of ones in these odd parity codes are 1 & 3.

- If the other system receives one of these odd parity codes, then there is no error in the received data. The bits other than odd parity bit are same as that of binary code.
- If the other system receives other than odd parity codes, then there is an errorss in the received data. In this case, we can't predict the original binary code because we don't know the bit positions of error.

Therefore, odd parity bit is useful only for detection of error in the received parity code. But, it is not sufficient to correct the error.

HAMMING CODE

Hamming codes are error-detecting codes that are designed to correct a single-bit error in digital data transmission. Hamming codes are more popular due to their ability to detect errors up to a single bit. The error correction is done based on the generation of a number of parity bits. The parity bits are generated based on the data transmitted between receiver and sender. For every data, we need to add parity bits in order to generate the Hamming code. All the code is represented in binary form.

Parity Bits

For generating Hamming code, we need to add parity codes to the data. For obtaining the parity bits, the following relations must hold true.

$$2^p \geq p + m + 1$$

In the above expression, p is the number of parity bits to be generated, m is the number of bits or number of message bits. For example, let us say we want to determine the number of parity bits for 4-bit data. So the m value is 4. The above equation becomes

$$2^p \geq p + 5$$

For a value of p=3, the above equation is satisfied. This means that, for 4-bit data, the number of parity bits required is 3. These parity bits are added for security purposes to the data. After adding the parity bits, the total size of data transmission now becomes 7 bits. Four bits as message bits and three bits as parity bits.

Placement of Parity Bits

Placement of parity bits is important since the receiver must know out of the seven bits received which are parity bits and which are message bits. Else, it would again cause an error while interpreting the data. So a protocol is followed for placing the parity bits which is explained as follows. Now, we have a total of seven bits, out of which four bits are message bits and three bits are parity bits. In the seven-bit data, the parity bits are added in the place of powers of two. Consider the following table.

Power of Two	2^0	2^1		2^2			
Bit Location	1	2	3	4	5	6	7
Bits	P1	P2	M1	P3	M2	M3	M4

Placement of Parity Bits

In the table above, P1, P2, and P3 are parity bits. M1, M2, M3, and M4 are message bits. As mentioned, the location of parity bits is the powers of two, i.e. two to the power zero, one, and two. Hence parity bit location is 1, 2, and 4. The rest of locations 3,5,6 and 7 are allotted to the four message bits. The overall bits are seven. The value of parity bits is evaluated based on the message which is transmitted. This code is also known as (7,4) code out of which total bits are seven, consisting of 4 message bits and 3 parity bits.

Hamming Code Equation

For developing the complete hamming code, we need to first determine the parity bit values. For determining the parity bit, again we have two types, the first one is called Hamming code with even parity and then Hamming code with odd parity. Let's see the Hamming code with even parity first.

Hamming Code with Even Parity

Let us determine Hamming code for the message to be transmitted 1110. Once we get the message to be transmitted, we can fix the places for M1, M2, M3, and M4, as shown below.

o determine the value of P1, since we are following even parity, we need to check for an even number of ones in (P1, M1, M2, M3, M4). To determine the exact sequence of bits to be checked along with the parity, the following rule is followed.

Rules to Determine the Parity Sequence

- Rule 1. The value of the parity bit is determined by the sequence of bits that is alternatively checked and skipped. For P1, check one, skip one. So The pattern becomes, start with location one, skip 2, check 3, skip 4, check 5 skips 6 and check 7. So it becomes (P1,3,5,7)
- Rule 2. To determine the value of parity bit P2, the rule is to check two bits and skip two bits. So, since the parity bit, P2 starts from location two, check locations 2 and 3, and then skip locations 4 and 5. Then check 6 and 7, and skip 8 and 9 (if available). The resultant sequence now becomes (P2,3,6,7).
- Rule 3. To determine the value of parity bit P3, the rule is to check four bits, skip four bits. Starting from P3 location, check location 4,5,6,7 and skip location 8,9,10,11 (if available). The resultant sequence now becomes, (P3,5,6,7).

BOOLEAN ALGEBRA

Boolean algebra is used to analyze and simplify the digital (logic) circuits. It uses only the binary numbers i.e. 0 and 1. It is also called as **Binary Algebra** or **logical Algebra**. Boolean algebra was invented by **George Boole** in 1854.

Rule in Boolean algebra

Following are the important rules used in Boolean algebra.

- Variable used can have only two values. Binary 1 for HIGH and Binary 0 for LOW.
- Complement of a variable is represented by an overbar (-). Thus, complement of variable B is represented as . $\overline{B}$ Thus if B = 0 then $\overline{B}$ = 1 and B = 1 then $\overline{B}$ = 0.
- ORing of the variables is represented by a plus (+) sign between them. For example ORing of A, B, C is represented as A + B + C.
- Logical ANDing of the two or more variable is represented by writing a dot between them such as A.B.C. Sometime the dot may be omitted like ABC.

Boolean Laws

There are six types of Boolean Laws.

Commutative law

Any binary operation which satisfies the following expression is referred to as commutative operation.

(i) $A.B = B.A$ (ii) $A + B = B + A$

Commutative law states that changing the sequence of the variables does not have any effect on the output of a logic circuit.

Associative law

This law states that the order in which the logic operations are performed is irrelevant as their effect is the same.

(i) $(A.B).C = A.(B.C)$ (ii) $(A + B) + C = A + (B + C)$

Distributive law

Distributive law states the following condition.

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

AND law

These laws use the AND operation. Therefore they are called as **AND** laws.

(i) $A.0 = 0$ (ii) $A.1 = A$

(iii) $A.A = A$ (iv) $A.\overline{A} = 0$

OR law

These laws use the OR operation. Therefore they are called as **OR** laws.

(i) $A + 0 = A$ (ii) $A + 1 = 1$

(iii) $A + A = A$ (iv) $A + \overline{A} = 1$

INVERSION law

This law uses the NOT operation. The inversion law states that double inversion of a variable results in the original variable itself.

$\overline{\overline{A}} = A$

De-Morgan's Theorem

A famous mathematician **DeMorgan** invented the two most important theorems of boolean algebra. The DeMorgan's theorems are used for mathematical verification of the equivalency of the NOR and negative-AND gates and the negative-OR and NAND gates. These theorems play an important role in solving various boolean algebra expressions. In the below table, the logical operation for each combination of the input variable is defined.

The rules of De-Morgan's theorem are produced from the Boolean expressions for OR, AND, and NOT using two input variables x and y. The first theorem of Demorgan's says that if we perform the AND operation of two input variables and then perform the NOT operation of the result, the result will be the same as the OR operation of the complement of that variable. The second theorem of DeMorgan says that if we perform the OR operation of two input variables and then perform the NOT operation of the result, the result will be the same as the AND operation of the complement of that variable.

Input variables		Output Condition			
A	**B**	**AND**	**NAND**	**OR**	**NOR**
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	0

De-Morgan's First Theorem

According to the first theorem, the complement result of the AND operation is equal to the OR operation of the complement of that variable. Thus, it is equivalent to the NAND function and is a negative-OR function proving that (A.B)' = A'+B' and we can show this using the following table.

Inputs		Output For Each Term				
A	**B**	**A.B**	**(A.B)'**	**A'**	**B'**	**A'A+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

De-Morgan's Second Theorem

According to the second theorem, the complement result of the OR operation is equal to the AND operation of the complement of that variable. Thus, it is the equivalent of the NOR function and is a

negative-AND function proving that (A+B)' = A'.B' and we can show this using the following truth table.

Inputs		**Output For Each Term**				
A	**B**	**A+B**	**(A+B)'**	**A'**	**B'**	**A'.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

Let's take some examples in which we take some expressions and apply DeMorgan's theorems.

Example 1: (A.B.C)'

(A.B.C)'=A'+B'+C'

Example 2: (A+B+C)'

(A+B+C)'=A'.B'.C

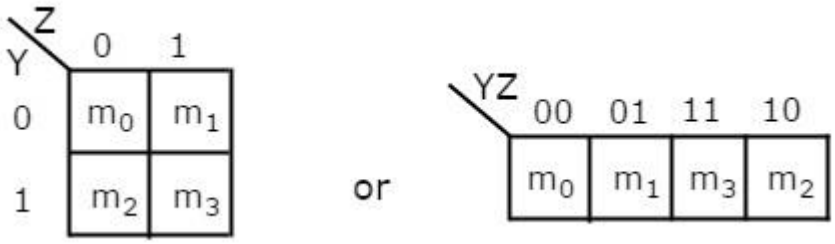
Karnaugh-map or K-map

We have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.

To overcome this difficulty, **Karnaugh** introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of 2^n cells for ‘n’ variables. The adjacent cells are differed only in single bit position.

2 Variable K-Map

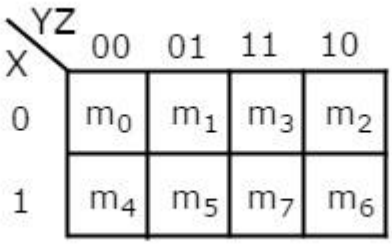
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

3 Variable K-Map

The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.



- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are {(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) and (m_2, m_0, m_6, m_4)}.
- The possible combinations of grouping 2 adjacent min terms are {(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) and (m_2, m_6)}.
- If x=0, then 3 variable K-map becomes 2 variable K-map.

Sum of Products (SOP) Form

A canonical sum of products is a boolean expression that entirely consists of minterms. The Boolean function F is defined on two variables X and Y. The X and Y are the inputs of the boolean function F

whose output is true when any one of the inputs is set to true. The truth table for Boolean expression F is as follows:

Inputs		**Output**
X	**Y**	**F**
0	0	0
0	1	1
1	0	1
1	1	1

Now, a column will be added for the minterm in the above table. The complement of the variables is taken whose value is 0, and the variables whose value is 1 will remain the same.

Inputs		**Output**	**Minterm**
X	**Y**	**F**	**M**
0	0	0	X'Y'
0	1	1	X'Y
1	0	1	XY'
1	1	1	XY

Now, we will add all the minterms for which the output is true to find the desired canonical SOP(Sum of Product) expression.

F=X' Y+XY'+XY

Product of Sums (POS) Form

A canonical product of sum is a boolean expression that entirely consists of maxterms. The Boolean function F is defined on two variables X and Y. The X and Y are the inputs of the boolean function F whose output is true when only one of the inputs is set to true. The truth table for Boolean expression F is as follows:

Inputs		**Output**
X	**Y**	**F**
0	0	0
0	1	1
1	0	1
1	1	0

In our minterm and maxterm section, we learned about how we can form the maxterm from the variable's value. A column will be added for the maxterm in the above table. The complement of the variables is taken whose value is 0, and the variables whose value is 1 will remain the same.

Inputs		**Output**	**Minterm**
X	**Y**	**F**	**M**
0	0	0	X'+Y'
0	1	1	X'+Y
1	0	1	X+Y'
1	1	1	X+Y

Now, we will multiply all the minterms for which the output is false to find the desired canonical POS(Product of sum) expression.

F=(X'+Y').(X+Y)

UNITIV

I/O ORGANIZATION

Peripheral Devices

Peripheral devices are those devices that are linked either internally or externally to a computer. These devices are commonly used to transfer data. The most common processes that are carried out in a computer are entering data and displaying processed data. Several devices can be used to receive data and display processed data. The devices used to perform these functions are called peripherals or I/O devices.

Peripherals read information from or write in the memory unit on receiving a command from the CPU. They are considered to be a part of the total computer system. As they require a conversion of signal values, these devices can be referred to as electromechanical and electromagnetic devices. The most common peripherals are a printer, scanner, keyboard, mouse, tape device, microphone, and external modem that are externally connected to the computer.

The following are some of the commonly used peripherals −

Keyboard

The keyboard is the most commonly used input device. It is used to provide commands to the computer. The commands are usually in the form of text. The keyboard consists of many keys such as function keys, numeric keypad, character keys, and various symbols.

Monitor

The most commonly used output device is the monitor. A cable connects the monitor to the video adapters in the computer's motherboard. These video adapters convert the electrical signals to the text and images that are displayed. The images on the monitor are made of thousands of pixels. The cursor is the characteristic feature of display devices. It marks the position on the screen where the next character will be inserted.

Printer

Printers provide a permanent record of computer data or text on paper. We can classify printers as impact and non-impact printers. Impact printers print characters due to the physical contact of the print head with the paper. In non-impact printers, there is no physical contact.

Magnetic Tape

Magnetic tapes are used in most companies to store data files. Magnetic tapes use a read-write mechanism. The read-write mechanism refers to writing data on or reading data from a magnetic tape. The tapes sequentially store the data manner. In this sequential processing, the computer must begin searching at the beginning and check each record until the desired data is available.

Magnetic tape is the cheapest medium for storage because it can store a large number of binary digits, bytes, or frames on every inch of the tape. The advantages of using magnetic tape include unlimited storage, low cost, high data density, rapid transfer rate, portability, and ease of use.

Magnetic Disk

There is another medium for storing data is magnetic disks. Magnetic disks have high-speed rotational surfaces coated with magnetic material. A read-write mechanism is used to achieve access to write on or read from the magnetic disk. Magnetic disks are generally used for the volume storage of programs and information.

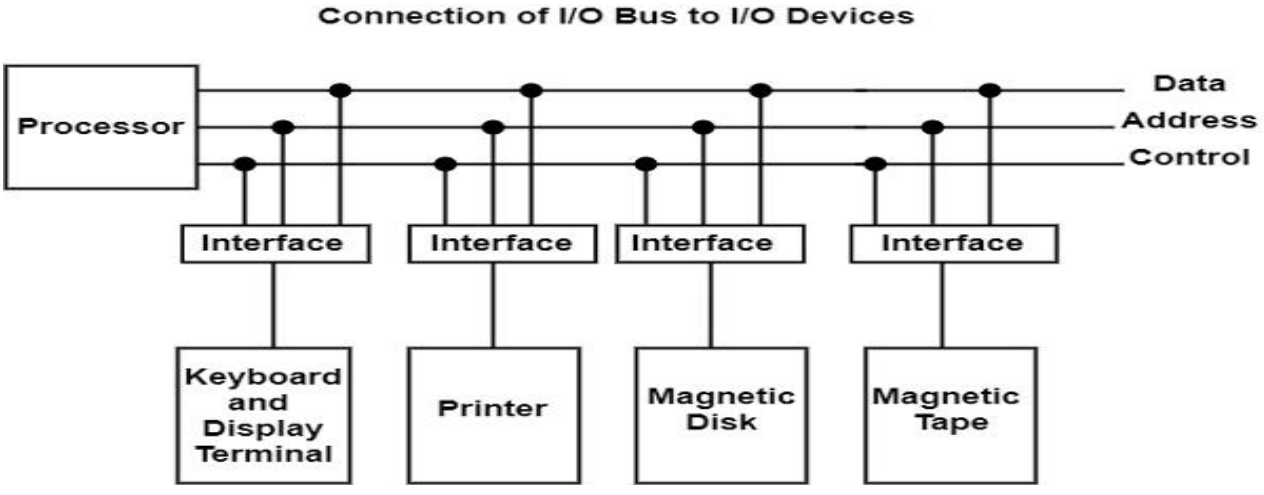
There are some peripheral devices found in computer systems including digital incremental plotters, optical and magnetic readers, analog to digital converters, and several data acquisition equipment.

I/O Interface

The I/O interface supports a method by which data is transferred between internal storage and external I/O devices. All the peripherals connected to a computer require special communication connections for interfacing them with the CPU.

I/O Bus and Interface Modules

The I/O bus is the route used for peripheral devices to interact with the computer processor. A typical connection of the I/O bus to I/O devices is shown in the figure.



The I/O bus includes data lines, address lines, and control lines. In any general-purpose computer, the magnetic disk, printer, and keyboard, and display terminal are commonly employed. Each peripheral unit has an interface unit associated with it. Each interface decodes the control and address received from the I/O bus.

It can describe the address and control received from the peripheral and supports signals for the peripheral controller. It also conducts the transfer of information between peripheral and processor and also integrates the data flow.

The I/O bus is linked to all peripheral interfaces from the processor. The processor locates a device address on the address line to interact with a specific device. Each interface contains an address decoder attached to the I/O bus that monitors the address lines.

When the address is recognized by the interface, it activates the direction between the bus lines and the device that it controls. The interface disables the peripherals whose address does not equivalent to the address in the bus.

An interface receives any of the following four commands −

- **Control** − A command control is given to activate the peripheral and to inform its next task. This control command depends on the peripheral, and each peripheral receives its sequence of control commands, depending on its mode of operation.
- **Status** − A status command can test multiple test conditions in the interface and the peripheral.
- **Data Output** − A data output command creates the interface counter to the command by sending data from the bus to one of its registers.
- **Data Input** − The data input command is opposite to the data output command. In data input, the interface gets an element of data from the peripheral and places it in its buffer register.

Mode of Transfers

We store the binary information received through an external device in the memory unit. The information transferred from the CPU to external devices originates from the memory unit. Although the CPU processes the data, the target and source are always the memory unit. We can transfer this information using three different modes of transfer.

1. **Programmed I/O**
2. **Interrupt- initiated I/O**
3. **Direct memory access(DMA)**

Programmed I/O

Programmed I/O uses the I/O instructions written in the computer program. The instructions in the program initiate every data item transfer. Usually, the data transfer is from a memory and CPU register. This case requires constant monitoring by the peripheral device's CPU.

Advantages:

- Programmed I/O is simple to implement.
- It requires very little hardware support.
- CPU checks status bits periodically.

Disadvantages:

- The processor has to wait for a long time for the I/O module to be ready for either transmission or reception of data.
- The performance of the entire system is severely degraded.

Interrupt-initiated I/O

In the above section, we saw that the CPU is kept busy unnecessarily. We can avoid this situation by using an interrupt-driven method for data transfer. The interrupt facilities and special commands inform the interface for issuing an interrupt request signal as soon as the data is available from any device. In the meantime, the CPU can execute other programs, and the interface will keep monitoring the i/O device. Whenever it determines that the device is ready for transferring data interface initiates an interrupt request signal to the CPU. As soon as the CPU detects an external interrupt signal, it stops the program it was already executing, branches to the service program to process the I/O transfer, and returns to the program it was initially running.

Working of CPU in terms of interrupts:

- CPU issues read command.
- It starts executing other programs.
- Check for interruptions at the end of each instruction cycle.
- On interruptions:-
    - Process interrupt by fetching data and storing it.
    - See operation system notes.
- Starts working on the program it was executing.

Advantages:

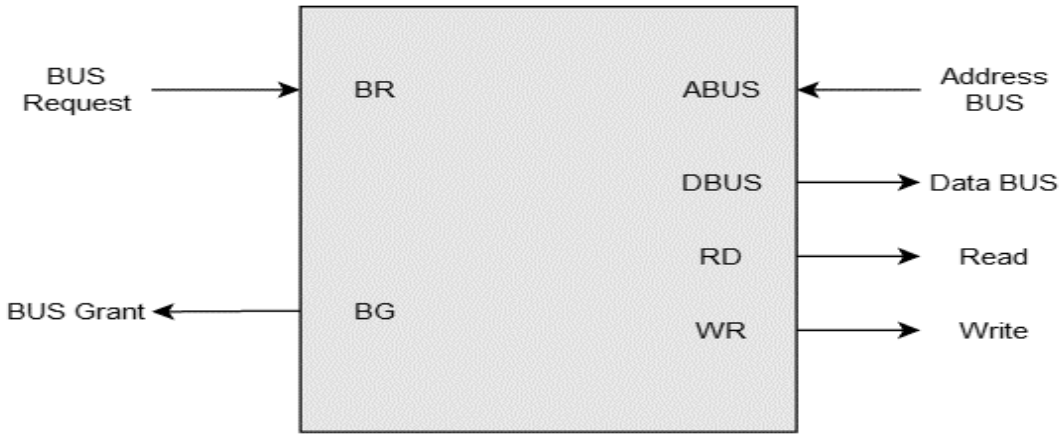
- It is faster and more efficient than Programmed I/O.
- It requires very little hardware support.
- CPU does not check status bits periodically.

Disadvantages:

- It can be tricky to implement if using a low-level language.
- It can be tough to get various pieces of work well together.
- The hardware manufacturer / OS maker usually implements it, e.g., Microsoft.

Direct Memory Access (DMA)

The data transfer between any fast storage media like a memory unit and a magnetic disk gets limited with the speed of the CPU. Thus it will be best to allow the peripherals to directly communicate with the storage using the memory buses by removing the intervention of the CPU. This mode of transfer of data technique is known as Direct Memory Access (DMA). During Direct Memory Access, the CPU is idle and has no control over the memory buses. The DMA controller takes over the buses and directly manages data transfer between the memory unit and I/O devices.



CPU Bus Signal for DMA transfer

Bus Request - We use bus requests in the DMA controller to ask the CPU to relinquish the control buses.

Bus Grant - CPU activates bus grant to inform DMA controller that DMA can take control of the control buses. Once the control is taken, it can transfer data in many ways.

Types of DMA transfer using DMA controller:

- **Burst Transfer:** In this transfer, DMA will return the bus control after the complete data transfer. A register is used as a byte count, which decrements for every byte transfer, and once it becomes zero, the DMA Controller will release the control bus. When the DMA Controller operates in burst mode, the CPU is halted for the duration of the data transfer.
- **Cyclic Stealing:** It is an alternative method for data transfer in which the DMA controller will transfer one word at a time. After that, it will return the control of the buses to the CPU. The CPU operation is only delayed for one memory cycle to allow the data transfer to “steal” one memory cycle.

RAM

RAM (Random Access Memory) is the internal memory of the CPU for storing data, program, and program result. It is a read/write memory which stores data until the machine is working. As soon as the machine is switched off, data is erased.

Access time in RAM is independent of the address, that is, each storage location inside the memory is as easy to reach as other locations and takes the same amount of time. Data in the RAM can be accessed randomly but it is very expensive.

RAM is volatile, i.e. data stored in it is lost when we switch off the computer or if there is a power failure. Hence, a backup Uninterruptible Power System (UPS) is often used with computers. RAM is small, both in terms of its physical size and in the amount of data it can hold.

RAM is of two types −

- Static RAM (SRAM)
- Dynamic RAM (DRAM)

Static RAM (SRAM)

The word **static** indicates that the memory retains its contents as long as power is being supplied. However, data is lost when the power gets down due to volatile nature. SRAM chips use a matrix of 6-transistors and no capacitors. Transistors do not require power to prevent leakage, so SRAM need not be refreshed on a regular basis.

There is extra space in the matrix, hence SRAM uses more chips than DRAM for the same amount of storage space, making the manufacturing costs higher. SRAM is thus used as cache memory and has very fast access.

Characteristic of Static RAM

- Long life
- No need to refresh
- Faster
- Used as cache memory
- Large size
- Expensive
- High power consumption

Dynamic RAM (DRAM)

DRAM, unlike SRAM, must be continually **refreshed** in order to maintain the data. This is done by placing the memory on a refresh circuit that rewrites the data several hundred times per second. DRAM is used for most system memory as it is cheap and small. All DRAMs are made up of memory cells, which are composed of one capacitor and one transistor.

Characteristics of Dynamic RAM

- Short data lifetime
- Needs to be refreshed continuously
- Slower as compared to SRAM
- Used as RAM
- Smaller in size
- Less expensive
- Less power consumption

ROM

ROM stands for **Read Only Memory**. The memory from which we can only read but cannot write on it. This type of memory is non-volatile. The information is stored permanently in such memories during manufacture. A ROM stores such instructions that are required to start a computer. This operation is referred to as **bootstrap**. ROM chips are not only used in the computer but also in other electronic items like washing machine and microwave oven.

MROM (Masked ROM)

The very first ROMs were hard-wired devices that contained a pre-programmed set of data or instructions. These kind of ROMs are known as masked ROMs, which are inexpensive.

PROM (Programmable Read Only Memory)

PROM is read-only memory that can be modified only once by a user. The user buys a blank PROM and enters the desired contents using a PROM program. Inside the PROM chip, there are small fuses which are burnt open during programming. It can be programmed only once and is not erasable.

EPROM (Erasable and Programmable Read Only Memory)

EPROM can be erased by exposing it to ultra-violet light for a duration of up to 40 minutes. Usually, an EPROM eraser achieves this function. During programming, an electrical charge is trapped in an insulated gate region. The charge is retained for more than 10 years because the charge has no leakage path. For erasing this charge, ultra-violet light is passed through a quartz crystal window (lid). This exposure to ultra-violet light dissipates the charge. During normal use, the quartz lid is sealed with a sticker.

EEPROM (Electrically Erasable and Programmable Read Only Memory)

EEPROM is programmed and erased electrically. It can be erased and reprogrammed about ten thousand times. Both erasing and programming take about 4 to 10 ms (millisecond). In EEPROM, any location can be selectively erased and programmed. EEPROMs can be erased one byte at a time, rather than erasing the entire chip. Hence, the process of reprogramming is flexible but slow.

Advantages of ROM

The advantages of ROM are as follows −

- Non-volatile in nature
- Cannot be accidentally changed
- Cheaper than RAMs
- Easy to test
- More reliable than RAMs
- Static and do not require refreshing
- Contents are always known and can be verified

Memory Decoding

In addition to requiring storage components in a memory unit, there is a need for decoding circuits to select the memory word specified by the input address.

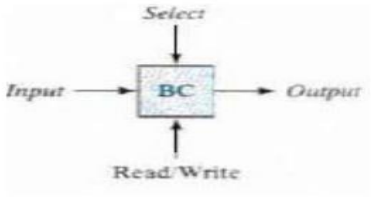


Fig: Block diagram of a memory cell

The storage part of the cell is modeled by an SR latch with associated gate s to form a D latch. Actually, the cell is an electronic circuit with four to six transistors. The select input enables the cell for reading or writing and the read/write input determines the operation of the cell when it is selected. A 1 in the read/write input provides the read operation by fanning a path from the latch to the output terminal. A 0 in the read/write input provides the write operation by forming a path from the input terminal to the latch.

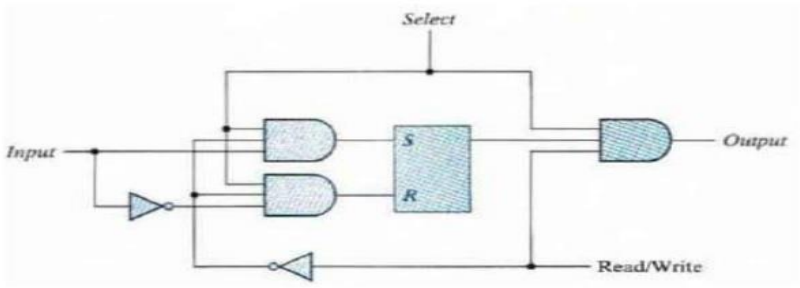


Fig: Logic diagram of a memory cell

The logical construction of a small RAM consists of four words of four bits each and bas a$_{\times}$total of 16 binary cells. The small blocks labeled BC represent the binary cell with its three inputs and one output. A memory with four words needs two address lines. The two address inputs go through a 2 4 decoder to select one of the four words. The decoder is enabled with the memory-enable input.

When the memory enable is 0, all outputs of the decoder are 0 and none of the memory words are selected. With the memory select at 1, one of the four words is selected, dictated by the value in the two address lines.

Once a word has been selected, the read/write input determines the operation. During the read operation the four bits of the selected word go through OR gates to the output terminals.

During the write operation, the data available in the input lines arc transferred into the four binary cells of the selected word. The binary cells that are not selected are disabled and their previous binary values remain unchanged.When the memory select input that goes into the decoder is equal to 0 none of the words are selected and the contents of all cells remain unchanged regardless of the value of the read/write input.

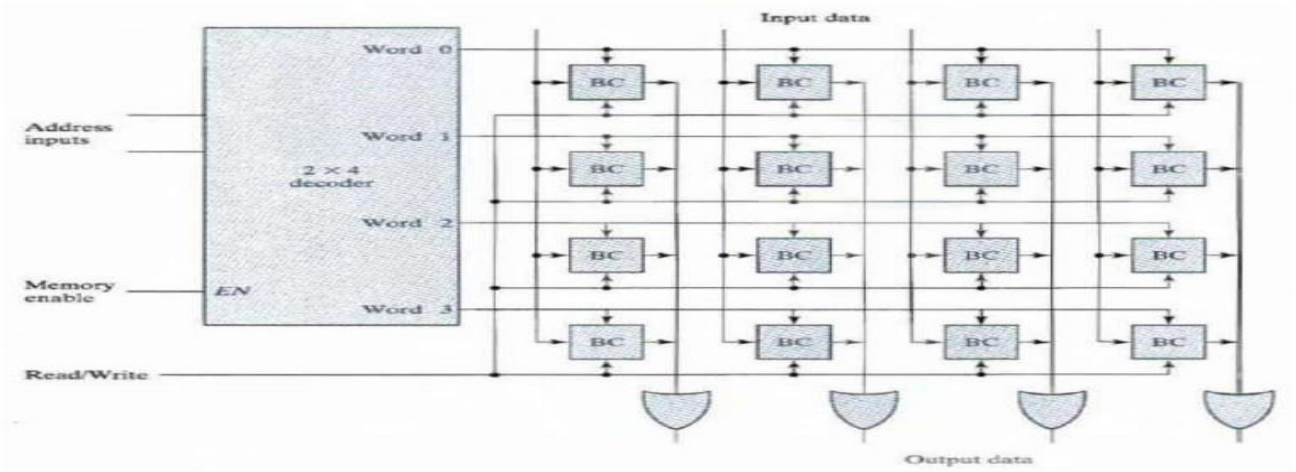


Fig: Diagram of a 4×4 RAM

- **Coincident Decoding**

A decoder with *k* inputs and 2^k outputs requires 2^k AND gates with *k* inputs per gate. The total number of gates and the number of inputs per gate can be reduced by employing two decoders in a two - dimensional selection scheme.

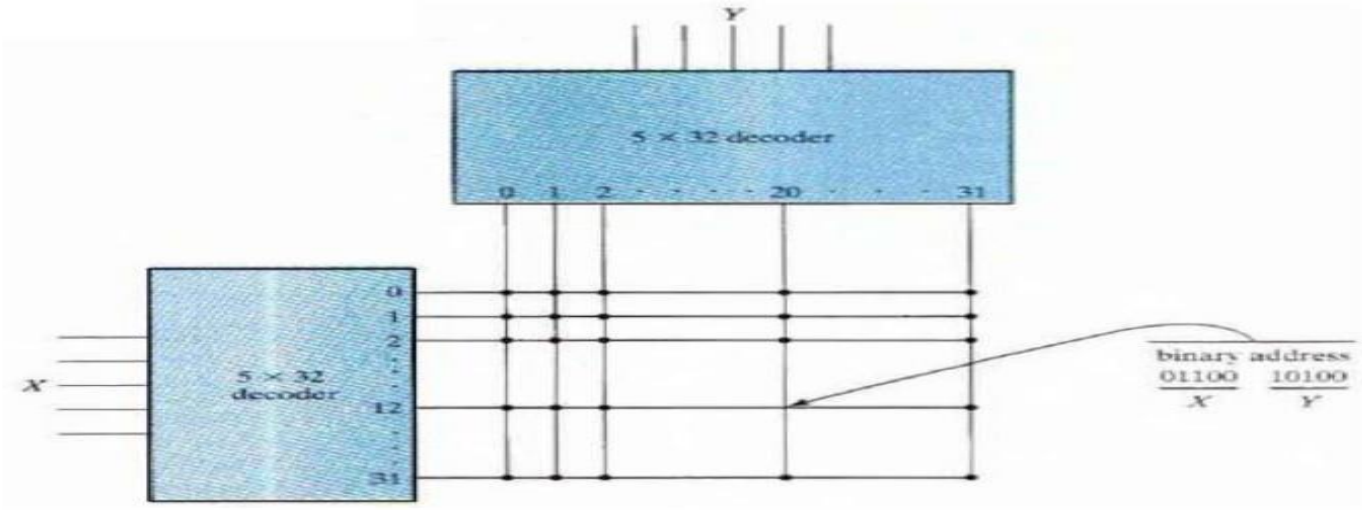


Fig: Two dimensional decoding structure for a 1K-word memory

The basic idea in two -dimensional decoding is to arrange the memory cells in an array, (i.e.) two $k/2$-

input decoders are used instead of one k-input decoder. One decoder performs the row selection and the other the column selection in a two-dimensional matrix configuration.

For example, instead of using a single 10 x 1,024 decoder, we use two 5 x 32 decoders. With the single decoder, we would need 1,024 AND gates with 10 inputs in each. The five most significant bits of the address go to input X and the five least significant bits go to input Y. Each word within the

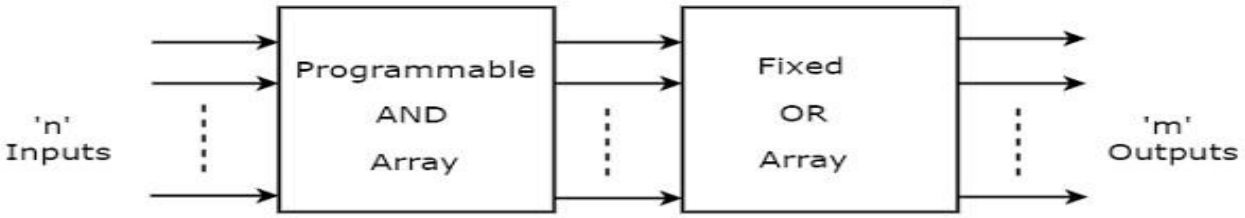
memory array is selected by the coincidence of one X line and one Y line. Thus each word in memory is selected by the coincidence between 1 of 32 rows and 1 of 32 columns, for a total of 1,024 words.

- **Address Multiplexing**

Because of large capacity, the address decoding of DRAM is arranged in a two dimensional array and larger memories often have multiple arrays. To reduce the number of pins in the IC package, designers utilize address multiplexing whereby one set of address input pins accommodates the address components.

Programmable Array Logic PAL

PAL is a programmable logic device that has Programmable AND array & fixed OR array. The advantage of PAL is that we can generate only the required product terms of Boolean function instead of generating all the min terms by using programmable AND gates. The **block diagram** of PAL is shown in the following figure.



Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

Here, the inputs of OR gates are not of programmable type. So, the number of inputs to each OR gate will be of fixed type. Hence, apply those required product terms to each OR gate as inputs. Therefore, the outputs of PAL will be in the form of **sum of products form**.

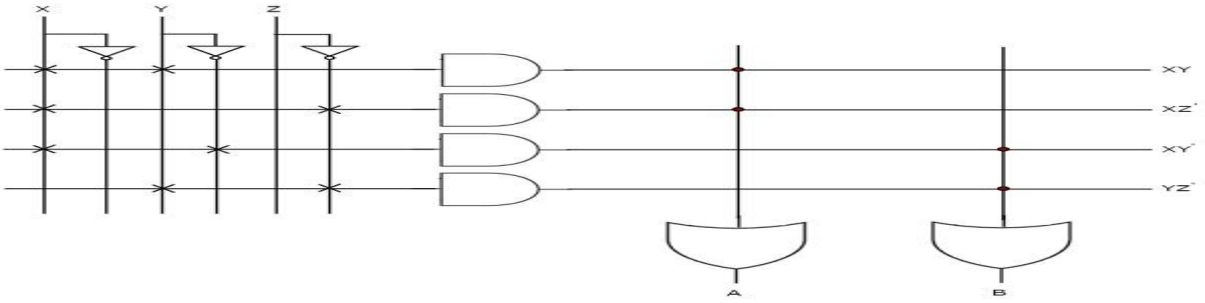
Example

Let us implement the following **Boolean functions** using PAL.

$$A=XY+XZ'A=XY+XZ'$$

$$A=XY'+YZ'A=XY'+YZ'$$

The given two functions are in sum of products form. There are two product terms present in each Boolean function. So, we require four programmable AND gates & two fixed OR gates for producing those two functions. The corresponding **PAL** is shown in the following figure.

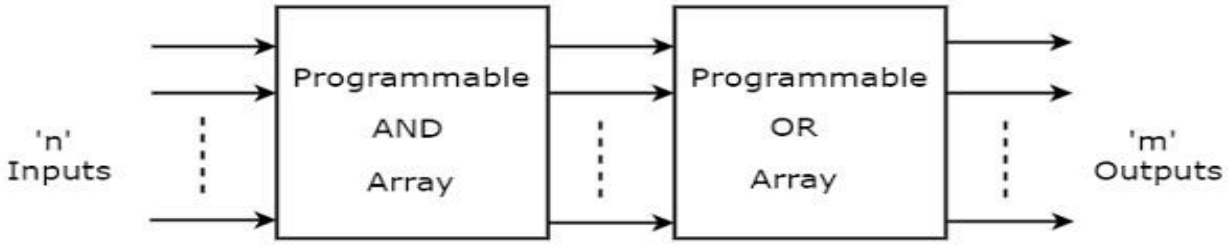


The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X′X′, Y, Y′Y′, Z & Z′Z′, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate. The symbol 'X' is used for programmable connections.

Here, the inputs of OR gates are of fixed type. So, the necessary product terms are connected to inputs of each **OR gate**. So that the OR gates produce the respective Boolean functions. The symbol '.' is used for fixed connections.

Programmable Logic Array PLA

PLA is a programmable logic device that has both Programmable AND array & Programmable OR array. Hence, it is the most flexible PLD. The **block diagram** of PLA is shown in the following figure.



Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of **sum of products form**.

Example

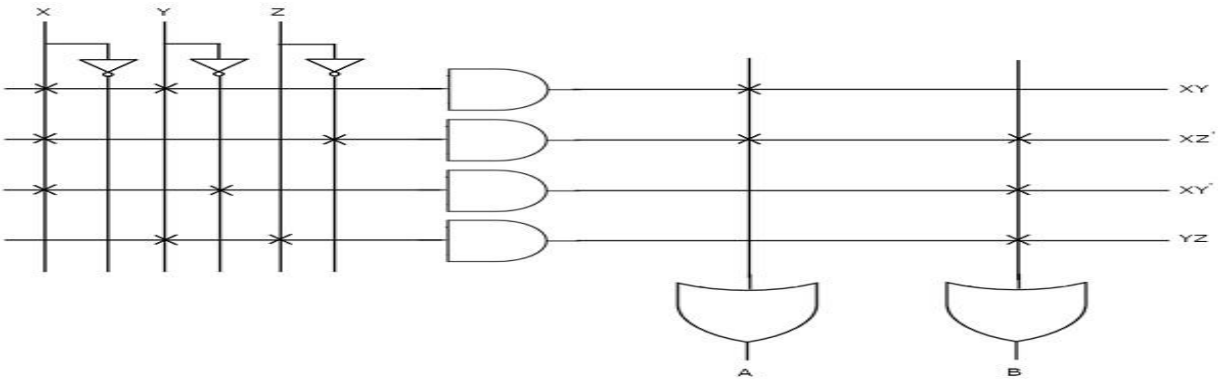
Let us implement the following **Boolean functions** using PLA.

$$A=XY+XZ'A=XY+XZ'$$

$$B=XY'+YZ+XZ'B=XY'+YZ+XZ'$$

The given two functions are in sum of products form. The number of product terms present in the given Boolean functions A & B are two and three respectively. One product term, Z′XZ′X is common in each function.

So, we require four programmable AND gates & two programmable OR gates for producing those two functions. The corresponding **PLA** is shown in the following figure.



The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X′X′, Y, Y′Y′, Z & Z′Z′, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate.

All these product terms are available at the inputs of each **programmable OR gate**. But, only program the required product terms in order to produce the respective Boolean functions by each OR gate. The symbol ‘X’ is used for programmable connections.

UNIT V

Memory Organization

The memory is organized in the form of a cell, each cell is able to be identified with a unique number called address. Each cell is able to recognize control signals such as “read” and “write”, generated by CPU when it wants to read or write address. Whenever CPU executes the program there is a need to transfer the instruction from the memory to CPU because the program is available in memory. To access the instruction CPU generates the memory request.

Memory Request:

Memory request contains the address along with the control signals. For Example, When inserting data into the stack, each block consumes memory (RAM) and the number of memory cells can be determined by the capacity of a memory chip.

Example: Find the total number of cells in 64k*8 memory chip.

Size of each cell = 8

Number of bytes in 64k = (2^6)*(2^10)

Therefore, the total number of cells = 2^16 cells

With the number of cells, the number of address lines required to enable one cell can be determined.

Word Size:

It is the maximum number of bits that a CPU can process at a time and it depends upon the processor. Word size is a fixed size piece of data handled as a unit by the instruction set or the hardware of a processor.

Word size varies as per the processor architectures because of generation and the present technology, it could be low as 4-bits or high as 64-bits depending on what a particular processor can handle. Word size is used for a number of concepts like Addresses, Registers, Fixed-point numbers,Floating-point numbers.

MemoryHierarchy

The Computer memory hierarchy looks like a pyramid structure which is used to describe the differences among memory types. It separates the computer storage based on hierarchy.

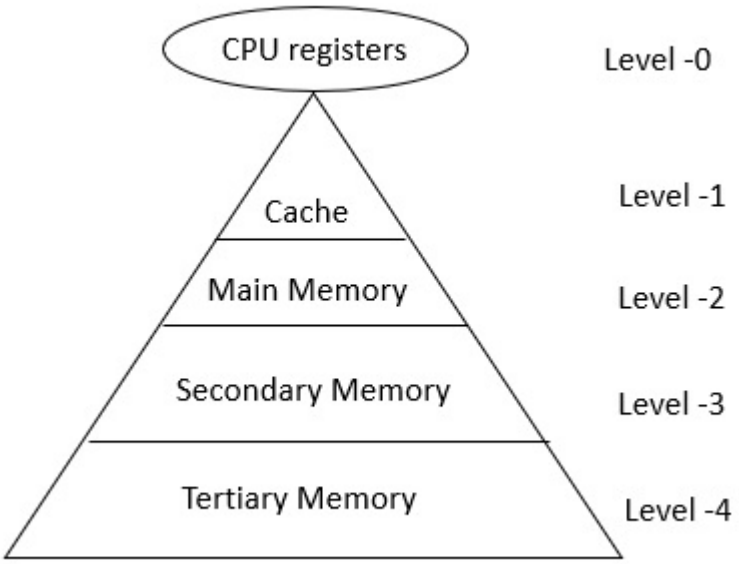
Level 0: CPU registers

Level 1: Cache memory

Level 2: Main memory or primary memory

Level 3: Magnetic disks or secondary memory

Level 4: Optical disks or magnetic types or tertiary Memory



In Memory Hierarchy the cost of memory, capacity is inversely proportional to speed. Here the devices are arranged in a manner Fast to slow that is form register to Tertiary memory.

Level-0 − Registers

The registers are present inside the CPU. As they are present inside the CPU, they have least access time. Registers are most expensive and smallest in size generally in kilobytes. They are implemented by using Flip-Flops.

Level-1 − Cache

Cache memory is used to store the segments of a program that are frequently accessed by the processor. It is expensive and smaller in size generally in Megabytes and is implemented by using static RAM.

Level-2 − Primary or Main Memory

It directly communicates with the CPU and with auxiliary memory devices through an I/O processor. Main memory is less expensive than cache memory and larger in size generally in Gigabytes. This memory is implemented by using dynamic RAM.

Level-3 − Secondary storage

Secondary storage devices like Magnetic Disk are present at level 3. They are used as backup storage. They are cheaper than main memory and larger in size generally in a few TB.

Level-4 − Tertiary storage

Tertiary storage devices like magnetic tape are present at level 4. They are used to store removable files and are the cheapest and largest in size (1-20 TB).

Let us see the memory levels in terms of size, access time, bandwidth.

Level	**Register**	**Cache**	**Primary memory**	**Secondary memory**
Bandwidth	4k to 32k MB/sec	800 to 5k MB/sec	400 to 2k MB/sec	4 to 32 MB/sec
Size	Less than 1KB	Less than 4MB	Less than 2 GB	Greater than 2 GB
Access time	2 to 5nsec	3 to 10 nsec	80 to 400 nsec	5ms
Managed by	Compiler	Hardware	Operating system	OS or user

MainMemory

Primary Memory (Main Memory)

Primary memory holds only those data and instructions on which the computer is currently working. It has a limited capacity and data is lost when power is switched off. It is generally made up of semiconductor device. These memories are not as fast as registers. The data and instruction required to be processed resides in the main memory. It is divided into two subcategories RAM and ROM.

Characteristics of Main Memory

- These are semiconductor memories.
- It is known as the main memory.
- Usually volatile memory.
- Data is lost in case power is switched off.
- It is the working memory of the computer.
- Faster than secondary memories.
- A computer cannot run without the primary memory.

AuxiliaryMemory

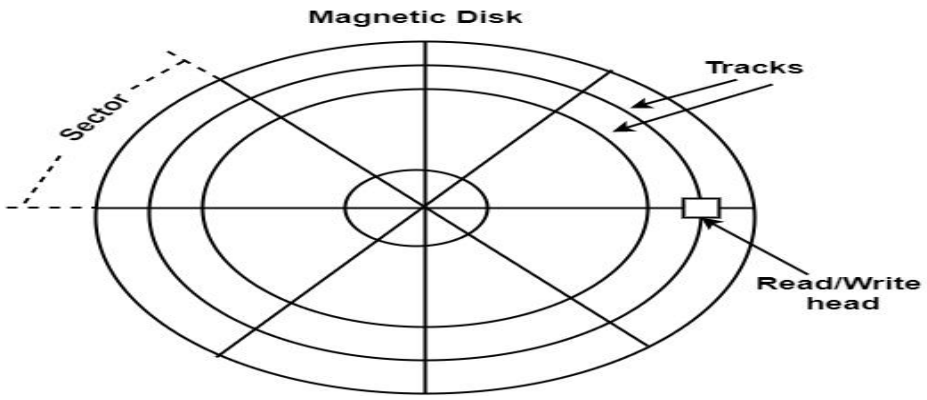
An Auxiliary memory is referred to as the lowest-cost, highest-space, and slowest-approach storage in a computer system. It is where programs and information are preserved for long-term storage or when not in direct use. The most typical auxiliary memory devices used in computer systems are magnetic disks and tapes.

Magnetic Disks

A magnetic disk is a round plate generated of metal or plastic coated with magnetized material. There are both sides of the disk are used and multiple disks can be stacked on one spindle with read/write heads accessible on each surface.

All disks revolve together at high speed and are not stopped or initiated for access purposes. Bits are saved in the magnetized surface in marks along concentric circles known as tracks. The tracks are frequently divided into areas known as sectors.

In this system, the lowest quantity of data that can be sent is a sector. The subdivision of one disk surface into tracks and sectors is displayed in the figure.



Magnetic Tape

Magnetic tape transport includes the robotic, mechanical, and electronic components to support the methods and control structure for a magnetic tape unit. The tape is a layer of plastic coated with a magnetic documentation medium.

Bits are listed as a magnetic stain on the tape along various tracks. There are seven or nine bits are recorded together to form a character together with a parity bit. Read/write heads are mounted one in each track therefore that information can be recorded and read as a series of characters.

Magnetic tape units can be stopped, initiated to move forward, or in the opposite, or it can be reversed. However, they cannot be initiated or stopped fast enough between single characters. For this reason, data is recorded in blocks defined as records. Gaps of unrecorded tape are added between records where the tape can be stopped.

The tape begins affecting while in a gap and achieves its permanent speed by the time it arrives at the next record. Each record on tape has a recognition bit design at the starting and end. By reading the bit design at the starting, the tape control recognizes the data number.

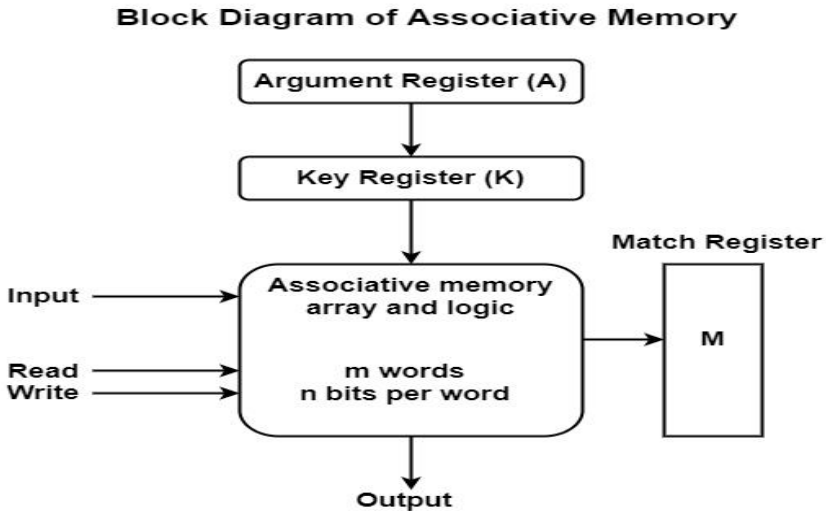
AssociativeMemory

An associative memory can be treated as a memory unit whose saved information can be recognized for approach by the content of the information itself instead of by an address or memory location. Associative memory is also known as **Content Addressable Memory (CAM)**.

The block diagram of associative memory is shown in the figure. It includes a memory array and logic for m words with n bits per word. The argument register A and key register K each have n bits, one for each bit of a word.

The match register M has m bits, one for each memory word. Each word in memory is related in parallel with the content of the argument register.The words that connect the bits of the argument register set an equivalent bit in the match register. After the matching process, those bits in the match register that have been set denote the fact that their equivalent words have been connected.

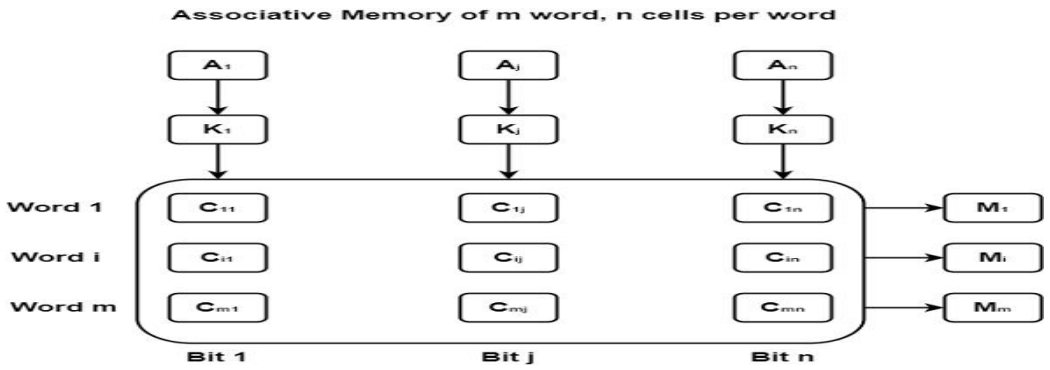
Reading is proficient through sequential access to memory for those words whose equivalent bits in the match register have been set.



The key register supports a mask for selecting a specific field or key in the argument word. The whole argument is distinguished with each memory word if the key register includes all 1's.

Hence, there are only those bits in the argument that have 1's in their equivalent position of the key register are compared. Therefore, the key gives a mask or recognizing a piece of data that determines how the reference to memory is created.

The following figure can define the relation between the memory array and the external registers in associative memory.



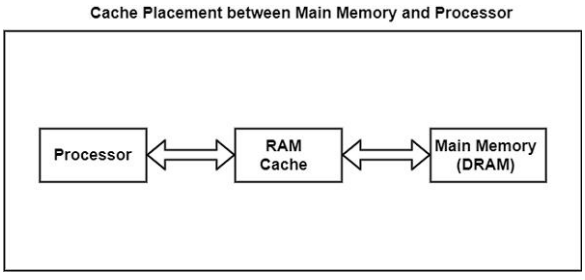
The cells in the array are considered by the letter C with two subscripts. The first subscript provides the word number and the second determines the bit position in the word. Therefore cell C_{ij} is the cell for bit j in word i.

A bit in the argument register is compared with all the bits in column j of the array supported that K_j = 1. This is completed for all columns j = 1, 2 . . . , n.If a match appears between all the unmasked bits of the argument and the bits in word i, the equivalent bit M_i in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, M_i is cleared to 0.

Cache Memory

The data or contents of the main memory that are used generally by the CPU are saved in the cache memory so that the processor can simply create that information in a shorter time. Whenever the CPU requires to create memory, it first tests the cache memory. If the data is not established in cache memory, so the CPU transfers into the main memory.

Cache memory is located between the CPU and the main memory. The block diagram for a cache memory can be represented as −

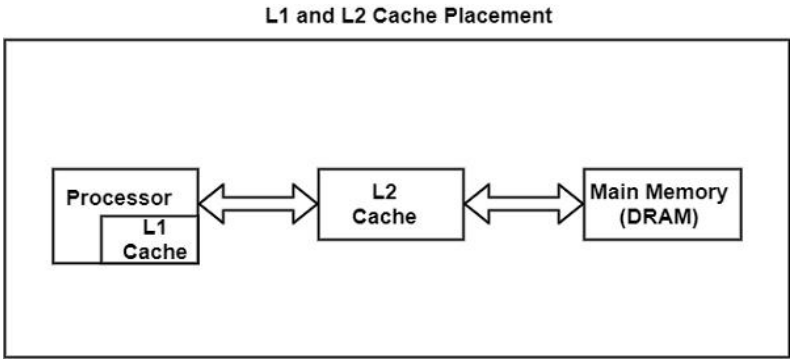


The concept of reducing the size of memory can be optimized by placing an even smaller SRAM between the cache and the processor, thereby creating two levels of cache. This new cache is usually contained inside the processor. As the new cache is put inside the processor, the wires connecting the

two become very short, and the interface circuitry becomes more closely integrated with that of the processor.

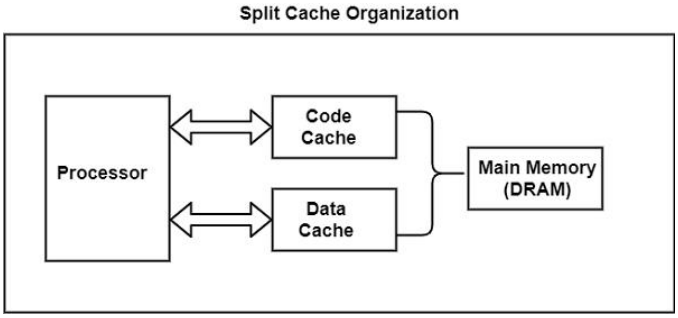
These two conditions together with the smaller decoder circuit facilitate faster data access. When two caches are present, the cache within the processor is referred to as a level 1 or L1 cache. The cache between the L1 cache and memory is referred to as a level 2 or L2 cache.

The figure shows the placement of L1 and L2 cache in memory.



The split cache, another cache organization, is shown in the figure. Split cache requires two caches. In this case, a processor uses one cache to store code/instructions and a second cache to store data.

This cache organization is typically used to support an advanced type of processor architecture such as pipelining. Here, the mechanisms used by the processor to handle the code are so distinct from those used for data that it does not make sense to put both types of information into the same cache.



The success of caches depends upon the principle of locality. The principle proposes that when one data item is loaded into a cache, the items close to it in memory should be loaded too.

If a program enters a loop, most of the instructions that are part of that loop are executed multiple times. Therefore, when the first instruction of a loop is being loaded into the cache, it loads its bordering instructions simultaneously to save time. In this way, the processor does not have to wait for the main memory to provide subsequent instructions.

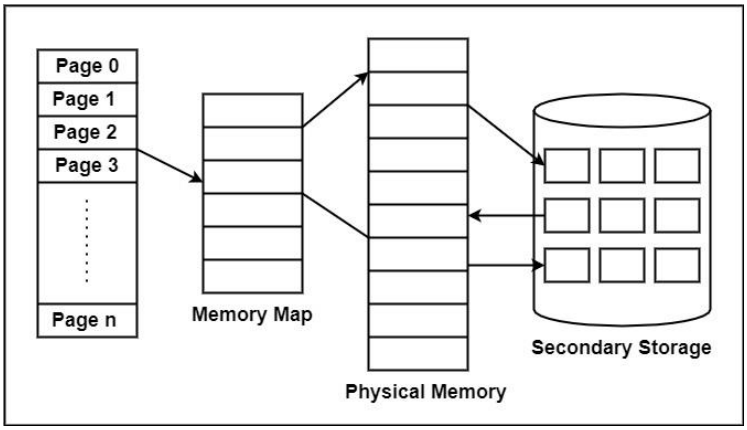
As a result of this, caches are organized in such a way that when one piece of data or code is loaded, the block of neighbouring items is loaded too. Each block loaded into the cache is identified with a number known as a tag.

This tag can be used to find the original addresses of the data in the main memory. Therefore, when the processor is in search of a piece of data or code (hereafter referred to as a word), it only needs to check the tags to see if the word is contained in the cache.

Virtual Memory

Virtual memory is the partition of logical memory from physical memory. This partition supports large virtual memory for programmers when only limited physical memory is available.

Virtual memory can give programmers the deception that they have a very high memory although the computer has a small main memory. It creates the function of programming easier because the programmer no longer requires to worry about the multiple physical memory available.



Virtual memory works similarly, but at one level up in the memory hierarchy. A memory management unit (MMU) transfers data between physical memory and some gradual storage device, generally a disk. This storage area can be defined as a swap disk or swap file, based on its execution. Retrieving data from physical memory is much faster than accessing data from the swap disk.

There are two primary methods for implementing virtual memory are as follows −

- **Paging**

Paging is a technique of memory management where small fixed-length pages are allocated instead of a single large variable-length contiguous block in the case of the dynamic allocation technique. In a paged system, each process is divided into several fixed-size ‘chunks’ called pages, typically 4k bytes in length. The memory space is also divided into blocks of the equal size known as frames.

Advantages of Paging

There are the following advantages of Paging are −

- In Paging, there is no requirement for external fragmentation.
- In Paging, the swapping among equal-size pages and page frames is clear.

- Paging is a simple approach that it can use for memory management.

Disadvantage of Paging

There are the following disadvantages of Paging are −

- In Paging, there can be a chance of Internal Fragmentation.
- In Paging, the page table employs more memory.
- Because of Multi-level Paging, there can be a chance of memory reference overhead.
- **Segmentation**

The partition of memory into logical units called segments, according to the user's perspective is called segmentation. Segmentation allows each segment to grow independently, and share. In other words, segmentation is a technique that partition memory into logically related units called a segment. It means that the program is a collection of the segment.

Unlike pages, segments can vary in size. This requires the MMU to manage segmented memory somewhat differently than it would manage paged memory. A segmented MMU contains a segment table to maintain track of the segments resident in memory.

A segment can initiate at one of several addresses and can be of any size, each segment table entry should contain the start address and segment size. Some system allows a segment to start at any address, while other limits the start address. One such limit is found in the Intel X86 architecture, which requires a segment to start at an address that has 6000 as its four low-order bits.

Dynamic Storage management

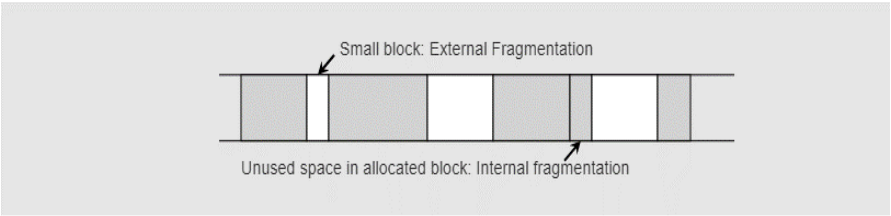
For the purpose of dynamic storage allocation, we view memory as a single array broken into a series of variable-size blocks, where some of the blocks are **free blocks** and some are **reserved blocks** or already allocated. The free blocks are linked together to form a **freelist** used for servicing future **memory requests**. This figure illustrates the situation that can arise after a series of memory allocations and deallocations.

Figure 11.2.1: The results from a series of memory allocations and deallocations. Memory is made up of a series of variable-size blocks, some allocated and some free. In this example, shaded areas represent memory currently allocated and unshaded areas represent unused memory available for future allocation.

When a memory request is received by the memory manager, some block on the freelist must be found that is large enough to service the request. If no such block is found, then the memory manager must resort to a **failure policy** such as **garbage collection**.

If there is a request for mm words, and no block exists of exactly size mm, then a larger block must be used instead. One possibility in this case is that the entire block is given away to the memory allocation request. This might be desirable when the size of the block is only slightly larger than the request. This is because saving a tiny block that is too small to be useful for a future memory request might not be worthwhile. Alternatively, for a free block of size kk, with $k>mk>m$, up to $k-mk-m$ space may be retained by the memory manager to form a new free block, while the rest is used to service the request.

Memory managers can suffer from two types of fragmentation. **External fragmentation** occurs when a series of memory requests result in lots of small free blocks, no one of which is useful for servicing typical requests. **Internal fragmentation** occurs when more than mm words are allocated to a request for mm words, wasting free storage. The difference between internal and external fragmentation is illustrated by this figure. The small white block labeled “External fragmentation” is too small to satisfy typical memory requests. The small grey block labeled “Internal fragmentation” was allocated as part of the grey block to its left, but it does not actually store information.



An illustration of internal and external fragmentation.

Some memory management schemes sacrifice space to internal fragmentation to make memory management easier (and perhaps reduce external fragmentation). For example, external fragmentation does not happen in file management systems that allocate file space in clusters. Another example of sacrificing space to internal fragmentation so as to simplify memory management is the **buddy method** described later in this chapter.

The process of searching the **memory pool** for a block large enough to service the request, possibly reserving the remaining space as a free block, is referred to as a **sequential fit** method.

Data ManagementConcepts

The term data management collectively describes those operating system functions that control the flow of data from an input/output (I/O) device to the processing program's data buffer and vice versa. It describes the functions that enforce data storage conventions.

Figure: Relation of Data Management to Devices and Application Program

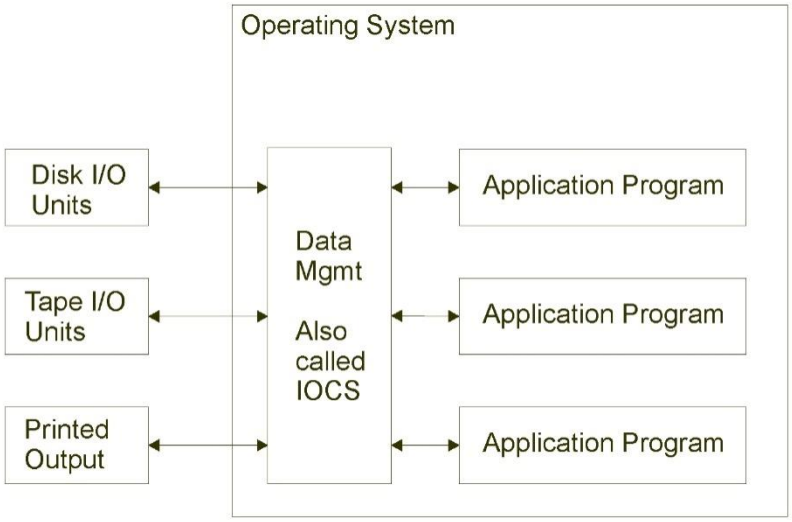


Figure shows that data management, a set of operating system routines called IOCS, is primarily concerned with:

- **Reading data**
  This involves the transfer of data from a device to an area in virtual storage. It involves locating logical records, one after the other, if the unit of data transfer comprises two or more logical records.
- **Writing data**
  This involves the transfer of data from virtual storage to an output device. It may involve the grouping logical records into blocks if the unit of data transfer comprises two or more logical records.

ProgrammableLogicdevices

Programmable Logic Devices PLDsPLDs are the integrated circuits. They contain an array of AND gates & another array of OR gates. There are three kinds of PLDs based on the type of arrayss, which has programmable feature.

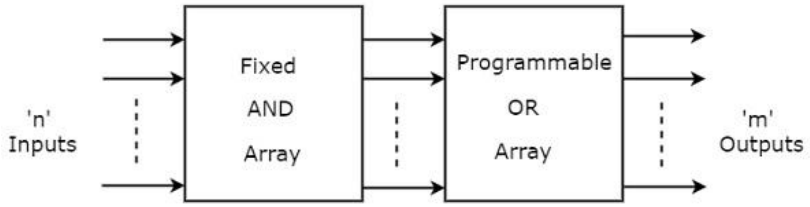
- Programmable Read Only Memory
- Programmable Array Logic
- Programmable Logic Array

The process of entering the information into these devices is known as **programming**. Basically, users can program these devices or ICs electrically in order to implement the Boolean functions based on the requirement. Here, the term programming refers to hardware programming but not software programming.

Programmable Read Only Memory PROM

Read Only Memory ROM is a memory device, which stores the binary information permanently. That means, we can't change that stored information by any means later. If the ROM has programmable feature, then it is called as **Programmable ROM (**PROM). The user has the flexibility to program the binary information electrically once by using PROM programmer.

PROM is a programmable logic device that has fixed AND array & Programmable OR array. The **block diagram** of PROM is shown in the following figure.

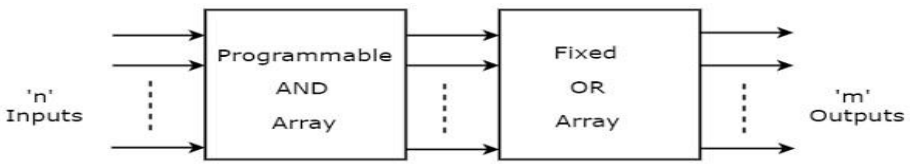


Here, the inputs of AND gates are not of programmable type. So, we have to generate 2^n product terms by using 2^n AND gates having n inputs each. We can implement these product terms by using $nx2^n$ decoder. So, this decoder generates 'n' **min terms**.

Here, the inputs of OR gates are programmable. That means, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PROM will be in the form of **sum of min terms**.

Programmable Array Logic PAL

PAL is a programmable logic device that has Programmable AND array & fixed OR array. The advantage of PAL is that we can generate only the required product terms of Boolean function instead of generating all the min terms by using programmable AND gates. The **block diagram** of PAL is shown in the following figure.



Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

Here, the inputs of OR gates are not of programmable type. So, the number of inputs to each OR gate will be of fixed type. Hence, apply those required product terms to each OR gate as inputs. Therefore, the outputs of PAL will be in the form of **sum of products form**.

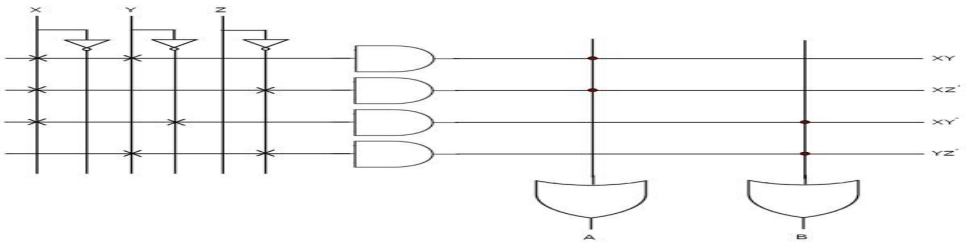
Example

Let us implement the following **Boolean functions** using PAL.

$$A=XY+XZ'A=XY+XZ'$$

$$A=XY'+YZ'A=XY'+YZ'$$

The given two functions are in sum of products form. There are two product terms present in each Boolean function. So, we require four programmable AND gates & two fixed OR gates for producing those two functions. The corresponding **PAL** is shown in the following figure.

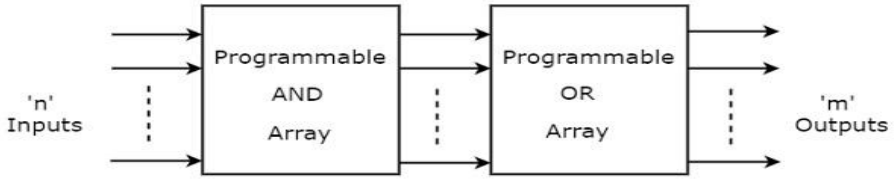


The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X′X′, Y, Y′Y′, Z & Z′Z′, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate. The symbol 'X' is used for programmable connections.

Here, the inputs of OR gates are of fixed type. So, the necessary product terms are connected to inputs of each **OR gate**. So that the OR gates produce the respective Boolean functions. The symbol '.' is used for fixed connections.

Programmable Logic Array PLA

PLA is a programmable logic device that has both Programmable AND array & Programmable OR array. Hence, it is the most flexible PLD. The **block diagram** of PLA is shown in the following figure.



Here, the inputs of AND gates are programmable. That means each AND gate has both normal and complemented inputs of variables. So, based on the requirement, we can program any of those inputs. So, we can generate only the required **product terms** by using these AND gates.

Here, the inputs of OR gates are also programmable. So, we can program any number of required product terms, since all the outputs of AND gates are applied as inputs to each OR gate. Therefore, the outputs of PAL will be in the form of **sum of products form**.

Example

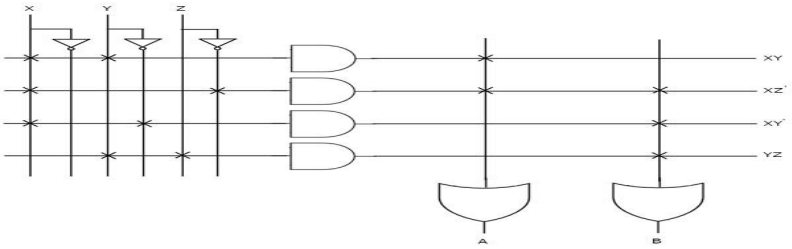
Let us implement the following **Boolean functions** using PLA.

$$A=XY+XZ'A=XY+XZ'$$

$$B=XY'+YZ+XZ'B=XY'+YZ+XZ'$$

The given two functions are in sum of products form. The number of product terms present in the given Boolean functions A & B are two and three respectively. One product term, Z′XZ′X is common in each function.

So, we require four programmable AND gates & two programmable OR gates for producing those two functions. The corresponding **PLA** is shown in the following figure.



The **programmable AND gates** have the access of both normal and complemented inputs of variables. In the above figure, the inputs X, X′X′, Y, Y′Y′, Z & Z′Z′, are available at the inputs of each AND gate. So, program only the required literals in order to generate one product term by each AND gate.

All these product terms are available at the inputs of each **programmable OR gate**. But, only program the required product terms in order to produce the respective Boolean functions by each OR gate. The symbol ‘X’ is used for programmable connections.