UCID - 16462
Rev. 1 5/74

LAWRENCE LIVERMORE LABORATORY
University of California/Livermore, California

LECTURE NOTES
For a Course in the
FUNDAMENTALS OF DIGITAL SYSTEMS

Dennis K. Fisher, Instructor

Winter 1974

Prepared for U. S. Atomic Energy Commission under contract no. W-7405-Eng-48

DISCLAIMER

UCID - 16462
Rev. 1 5/74

LAWRENCE LIVERMORE LABORATORY
University of California/Livermore, California

LECTURE NOTES
For a Course in the
FUNDAMENTALS OF DIGITAL SYSTEMS

Dennis K. Fisher, Instructor

Winter 1974

Prepared for U. S. Atomic Energy Commission under contract no. W-7405-Eng-48

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

PREFACE

OBJECTIVES

This course provides a systematic introduction to digital systems and computers. It is intended for mechanical engineers involved in the design and utilization of instrumentation and control systems involving digital logic or dedicated computers. It is also appropriate for the individual who has had high-level language programming experience and would like greater in-depth knowledge of computer operations.

The course begins with the basic concepts of the switching element and develops from that the principles involved in the design of complex logic systems. This leads to the ultimate digital system, the computer. Both the hardware and software aspects of computer technology will be covered.

PREREQUISITES

Undergraduate introductory course in electronics or equivalent experience.

TEXTS

H. V. Malmstadt and C. G. Enke, "Digital Electronics for Scientists," and Soucek, B., "Minicomputers in Data Processing and Simulation."

ACKNOWLEDGMENTS

These notes were written to accompany the above texts for a single semester course (25 one-hour lectures) in the Continuing Education Program at the Lawrence Livermore Laboratory. The notes follow the basic organization of the course texts and amount to a condensation of the original material contained therein. Some additional material has been added, and the basic sources are referenced where appropriate.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Course Policy, Objectives, Outline

1.2 Introduction to Digital Systems

1.2.1 Digital/Analog Comparison

TYPICAL DATA ACQUISITION SYSTEM CHARACTERISTICS

Attribute	System Type	
	Analog	Digital
Accuracy	0.1% of F.S.	0.1% of F.S. → Unlim.
Speed	Fast	Medium
Resolution	1 in. 10^4	1 in. 10^4 → Unlim.
Dynamic Range	90 db	90 db → Unlim.
Reliability	Good	Good → Excel.
Noise Immunity	Fair	Excel.
Data Processing Compatibility	Must be digitized	Inherently Compatible
Cost	Competitive	

1.2 Number Systems [1]

$Binary_2$	$Decimal_{10}$	BCD_{10}	$Octal_8$	$Hexadecimal_{16}$
0000	0	0000	00	0
0001	1	0001	01	1
0010	2	0010	02	2
0011	3	0011	03	3
0100	4	0100	04	4
0101	5	0101	05	5
0110	6	0110	06	6
0111	7	0111	07	7
1000	8	1000	10	8
1001	9	1001	11	9
1010	10	0001 0000	12	A
1011	11	0001 0001	13	B
1100	12	0001 0010	14	C
1101	13	0001 0011	15	D
1110	14	0001 0100	16	E
1111	15	0001 0101	17	F
1 0000	16	0001 0110	20	10

CONVERSION TO BASE 10

To convert any number to base 10, multiply each digit by its base raised to the power N where N is the digits position

BINARY: $0\ 1\ 1\ 1 \quad 1\ 0\ 1\ 1_2$

$1 \times 2^0 = 1$

$1 \times 2^1 = 2$

$0 \times 2^2 = 0$

$1 \times 2^3 = 8$

$1 \times 2^4 = 16$

$1 \times 2^5 = 32$

$1 \times 2^6 = 64$

123_{10}

OCTAL: $1\ 7\ 3_8$

$3 \times 8^0 = 3$

$7 \times 8^1 = 56$

$1 \times 8^2 = 64$

123_{10}

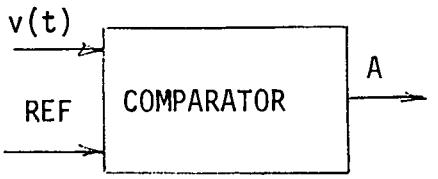
BCD: $0\ 0\ 0\ 1 \quad 0\ 0\ 1\ 0 \quad 0\ 0\ 1\ 1$ BCD

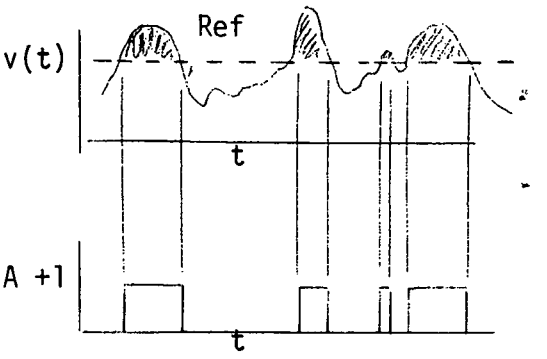
$1\ 2\ 3_{10}$

1.2.3 Digital Measurement Concepts [1]

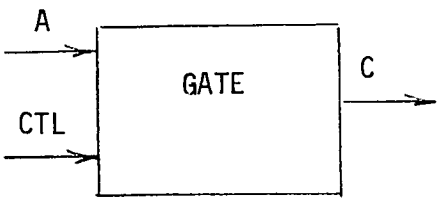
System Building Blocks

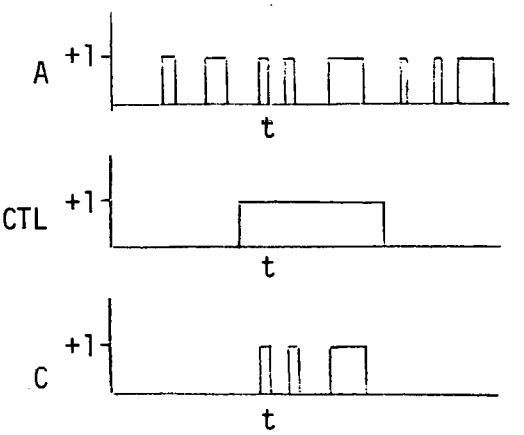
(a)



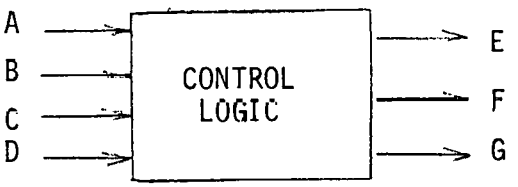


(b)





(c)

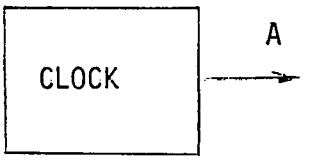


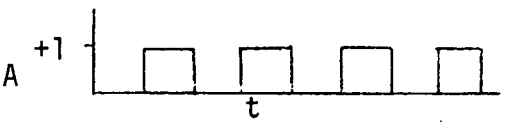
$E = f_1(t, A, B, C, D)$

$F = f_2(t, A, B, C, D)$

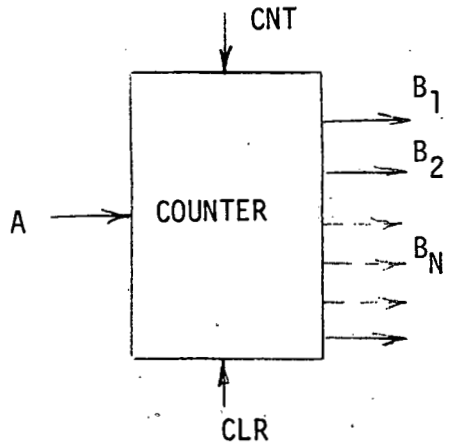
$G = f_3(t, A, B, C, D)$

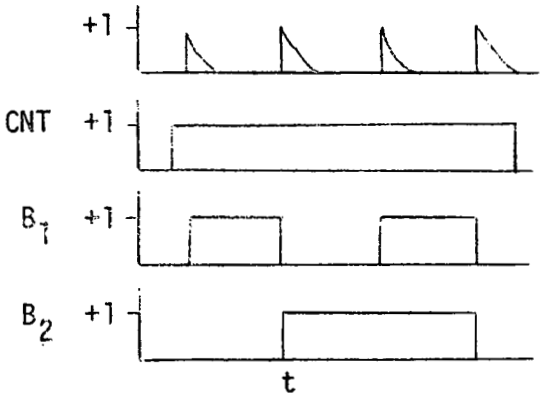
(d)



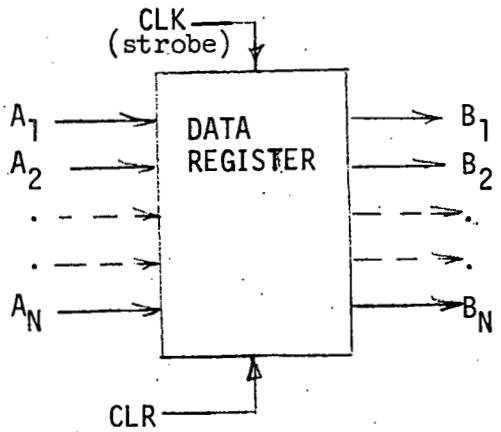


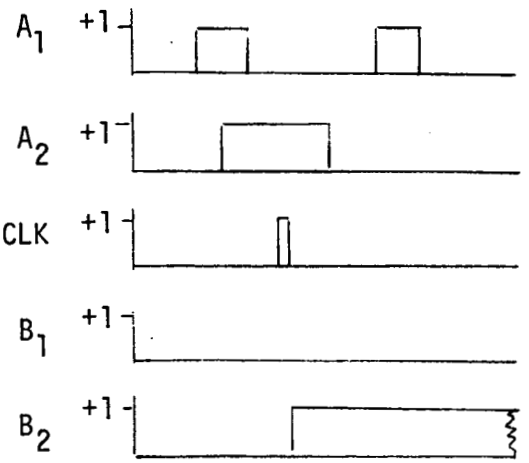
(e)


CNT
A
COUNTER
B_1
B_2
B_N
CLR

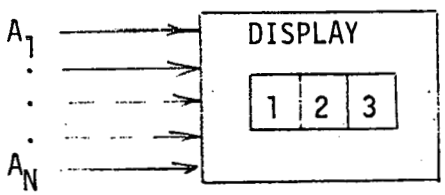

+1
CNT
+1
B_1
+1
B_2
+1
t

(f)

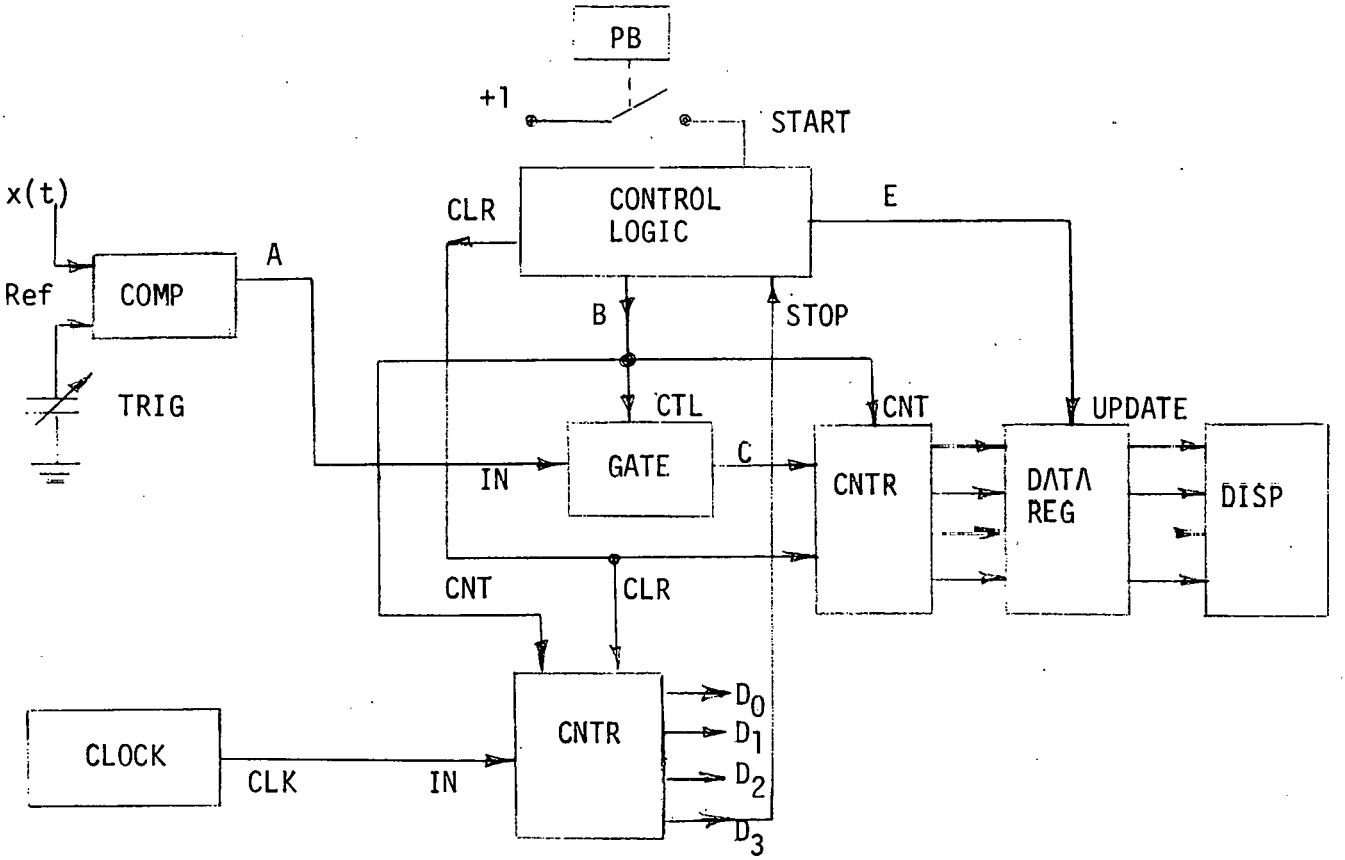

CLK
(strobe)
A_1
A_2
A_N
DATA
REGISTER
B_1
B_2
B_N
CLR

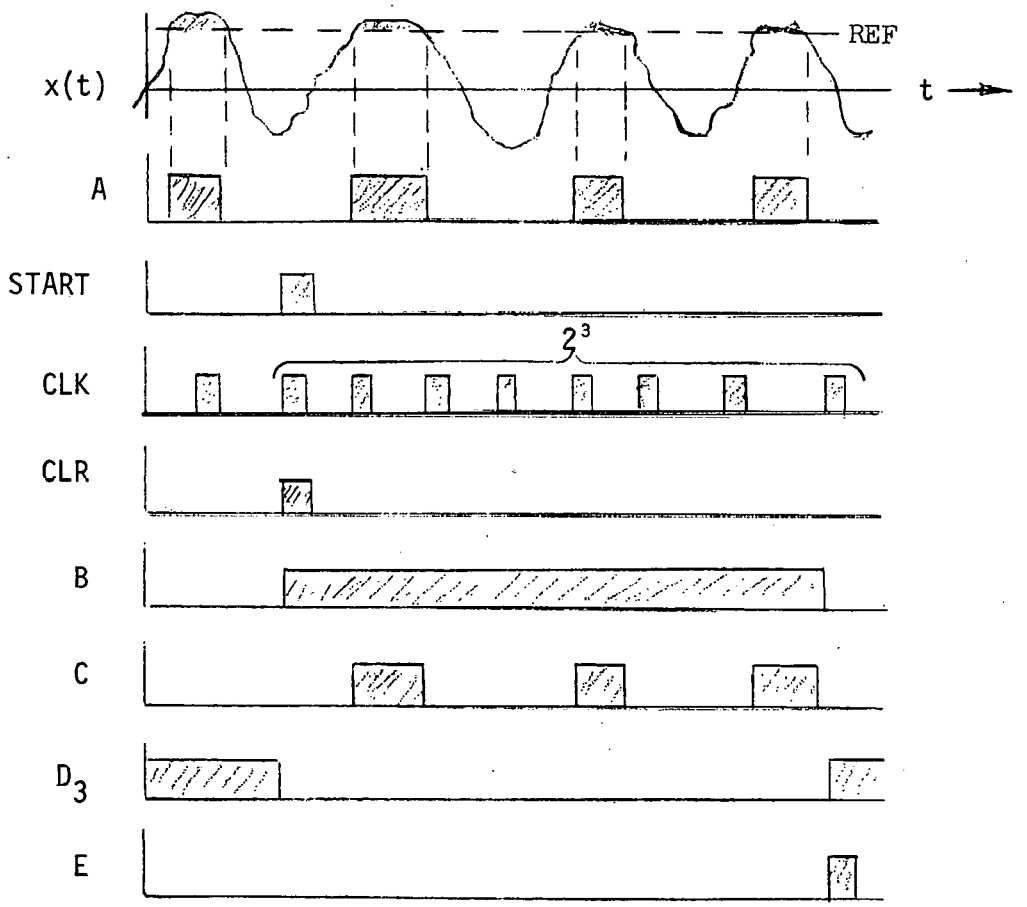

A_1
+1
A_2
+1
CLK
+1
B_1
+1
B_2
+1

(g)

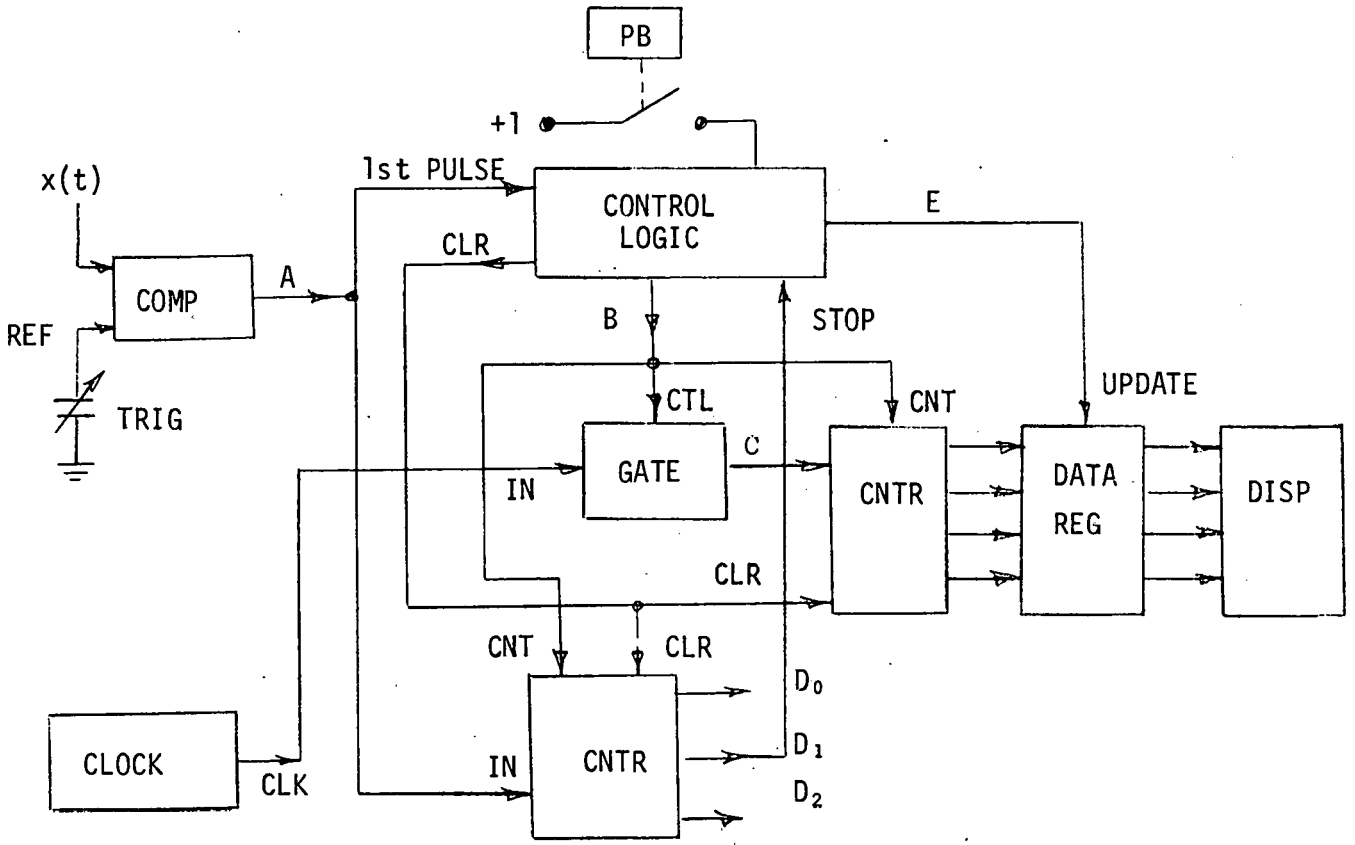

A_1
A_N
DISPLAY
1
2
3

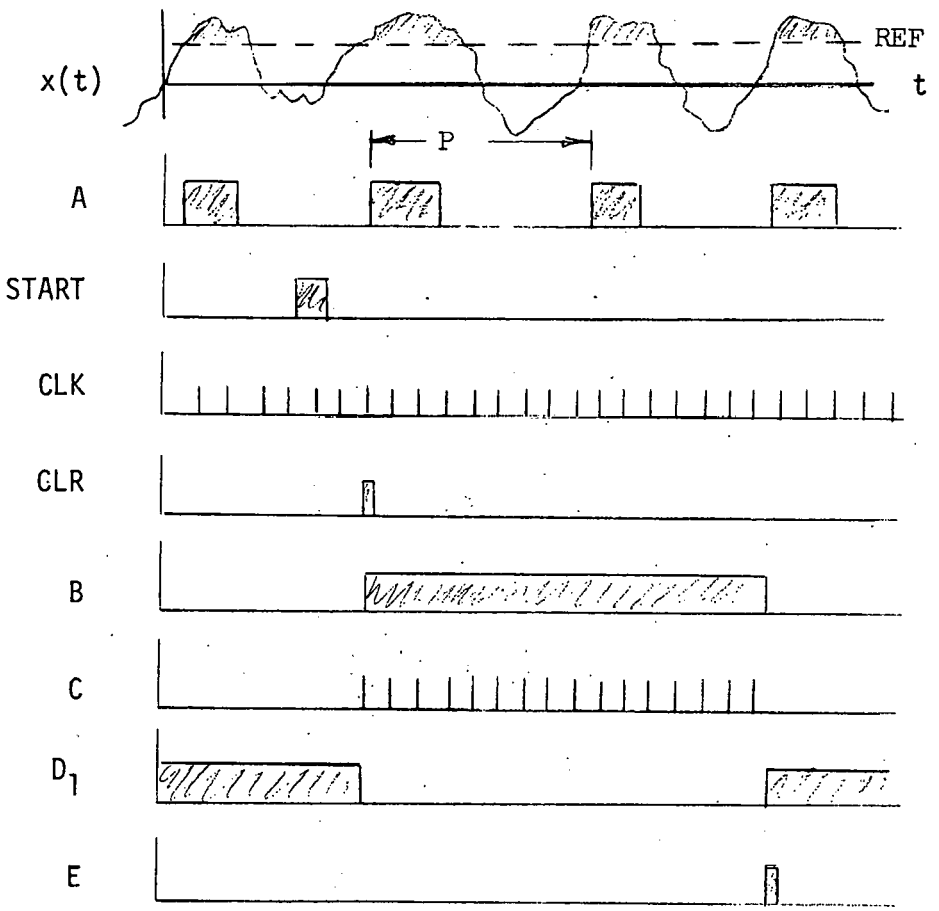
EXAMPLE 1-1 Frequency Meter (Freq α Counts/ΔT)





EX 1-2 Interval Timer (Period α ΔT/Count)

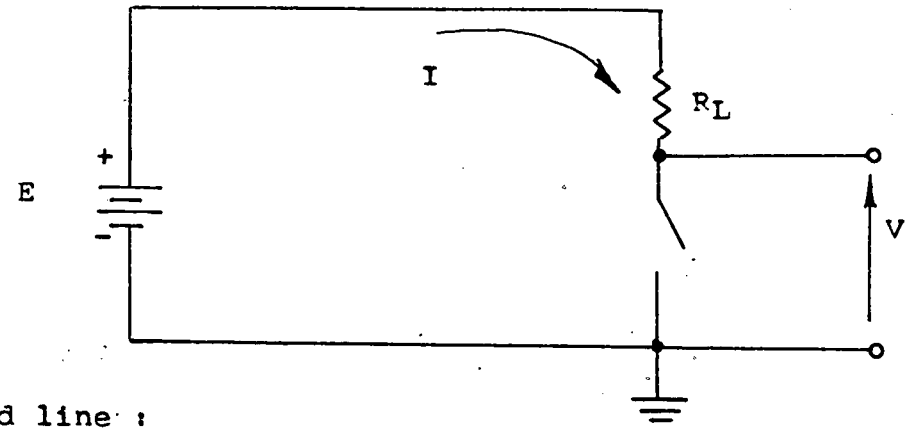




2. SWITCHING CONCEPTS[1]

2.1 Ideal and Real Switches

Ideal



Load line :

$$\lesssim V = 0$$

$$E - IR_L - V = 0$$

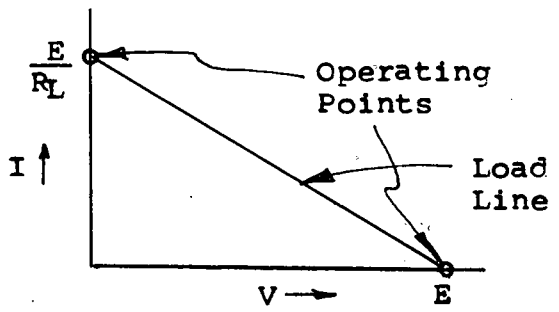
$$I = \frac{E - V}{R_L}$$

Operating points :

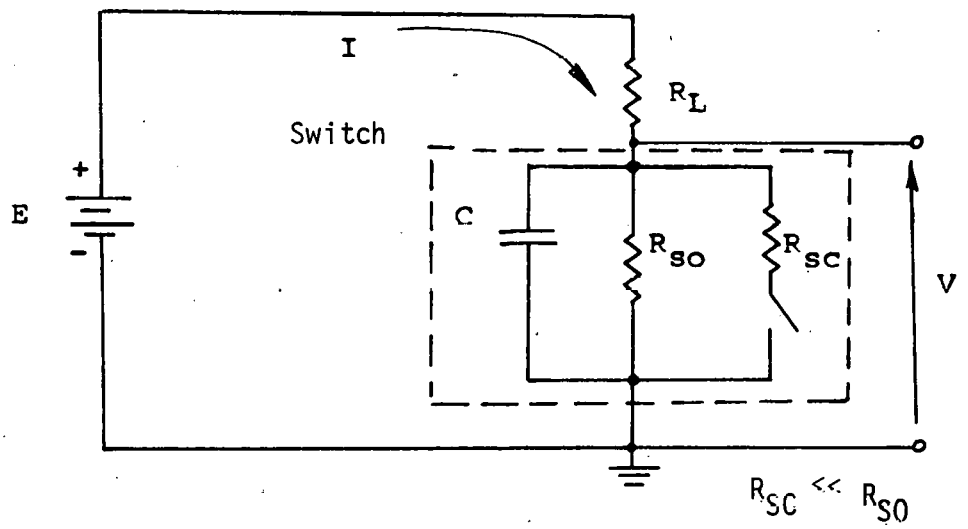
V = 0, Switch closed

and

V = E, Switch open



Real



Static Analysis -

Load line :

$$I = \frac{E - V}{R_L}$$

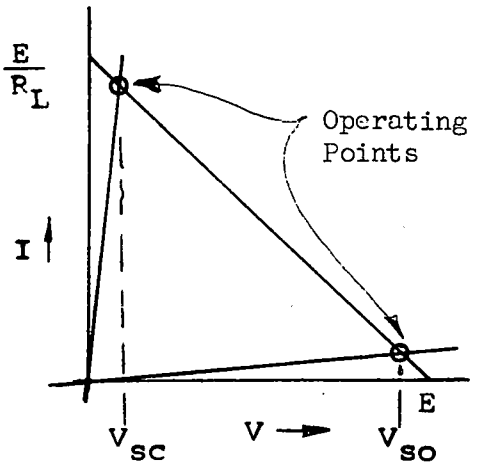
Operating Points :

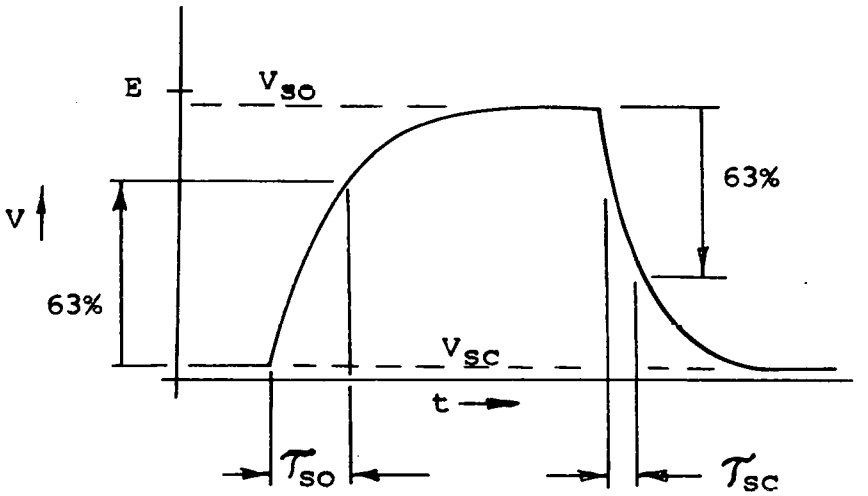
$V = IR_{so}$, switch open

and

$$V = I\left(\frac{R_{so}R_{sc}}{R_{so} + R_{sc}}\right)$$

$\approx I R_{sc}$, switch closed





Dynamic Analysis -

Opening transient :

$$V = V_{sc} + (V_{so} - V_{sc})\,(1 - e^{-t/\tau_{so}})$$

where

$$V_{so} = \left(\frac{R_{so}}{R_L + R_{so}}\right)E$$

$$V_{sc} \approx \left(\frac{R_{sc}}{R_L + R_{sc}}\right)E$$

$$\tau_{so} = \left(\frac{R_{so}R_L}{R_{so} + R_L}\right)C \approx R_L C,\ R_{so} >> R_L$$

Closing transient :

$$V = V_{sc} + (V_{so} - V_{sc})\,e^{-t/\tau_{sc}}$$

where $\tau_{sc} = \left(\frac{R_{sc}\,R_L}{R_{sc} + R_L}\right)C \approx R_{sc}C,\ R_{sc} << R_L$

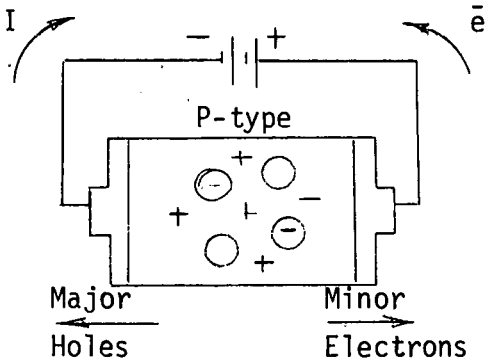
2.2 Switching Devices

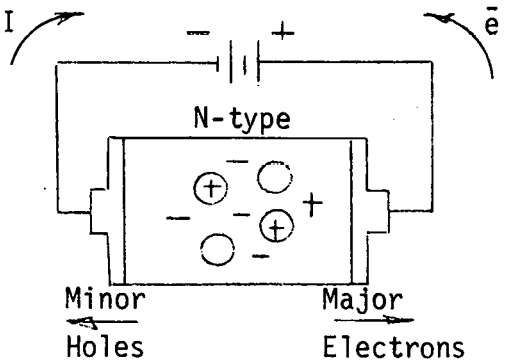
2.2.1 Diodes[6]

Diodes are a semiconductor device formed by the junction between N- and P-type materials

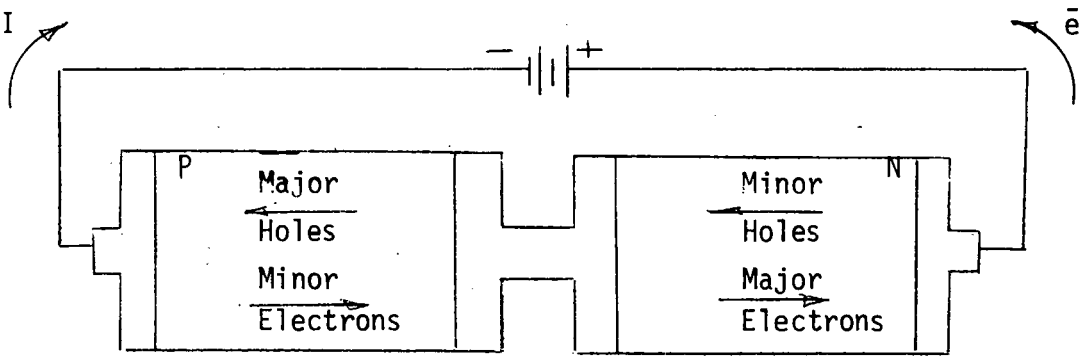
Semiconductors Ge, Si (IV) + Donor atoms P, As, Sb (V) = N Type

Semiconductors Ge, Si (IV) + Acceptor atoms Ga, In (III) = P Type

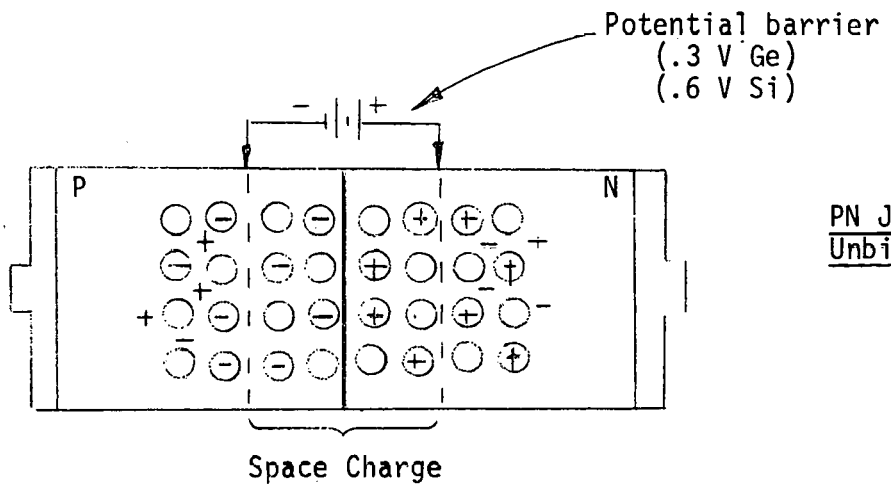




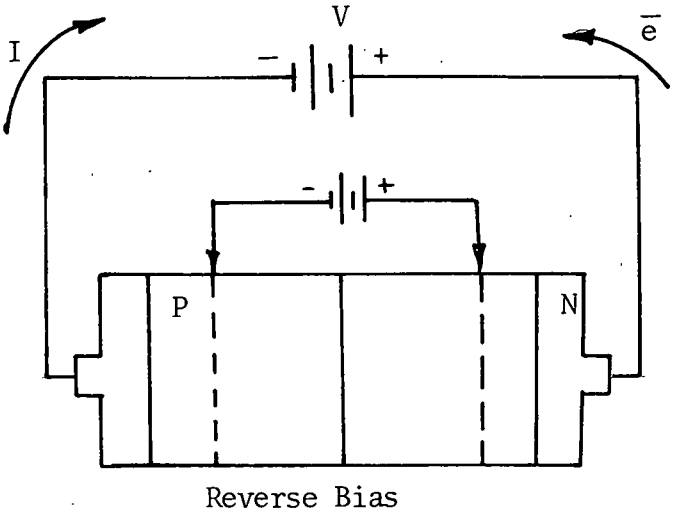
P & N Materials with Bias



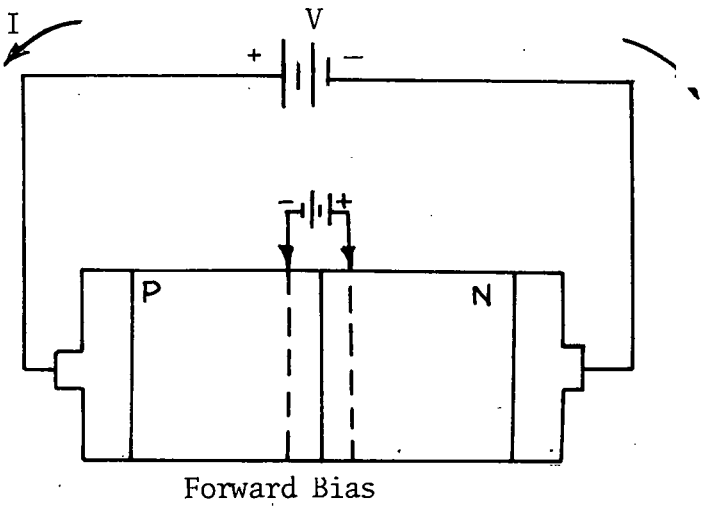
P & N Materials in Series with Bias



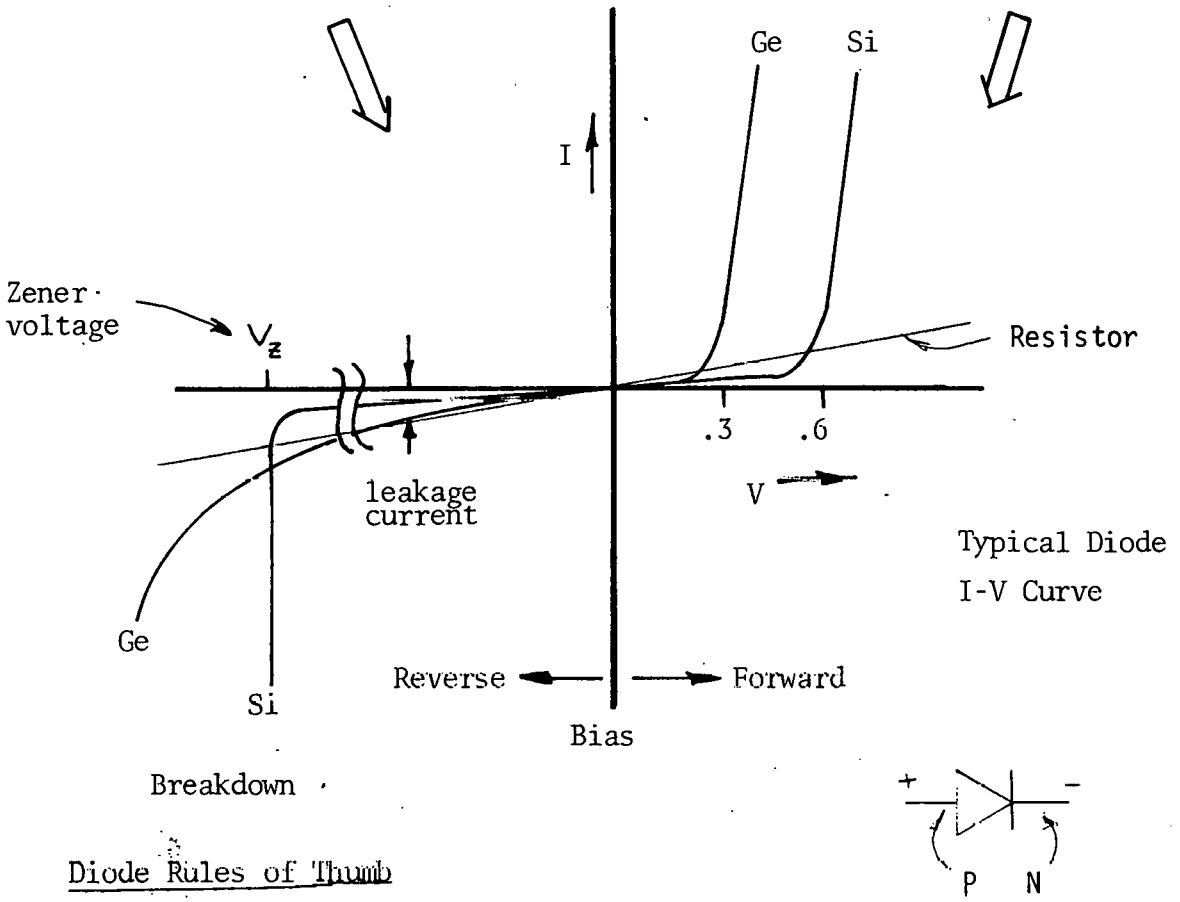
PN Junction Unbiased



Reverse Bias



Forward Bias

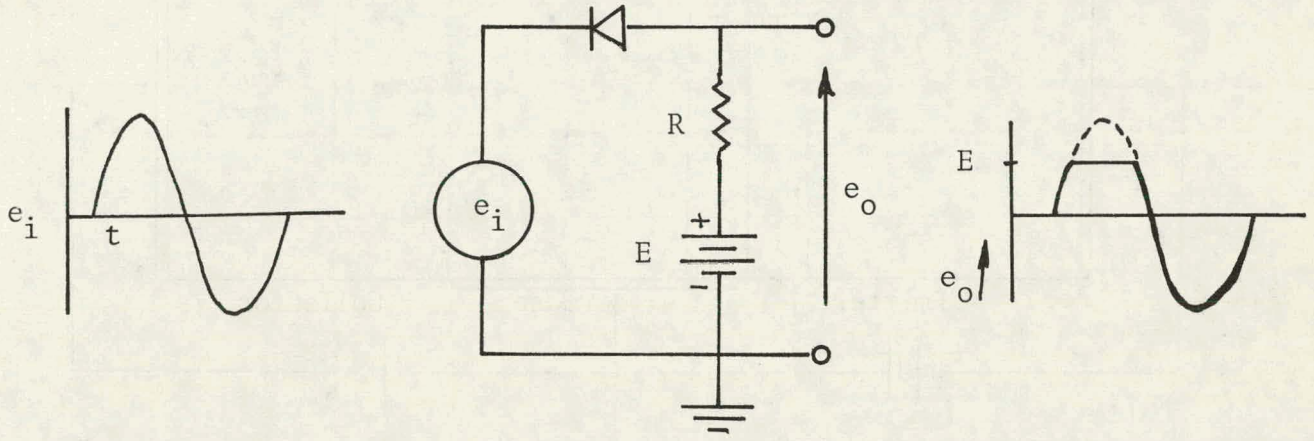


Typical Diode I-V Curve

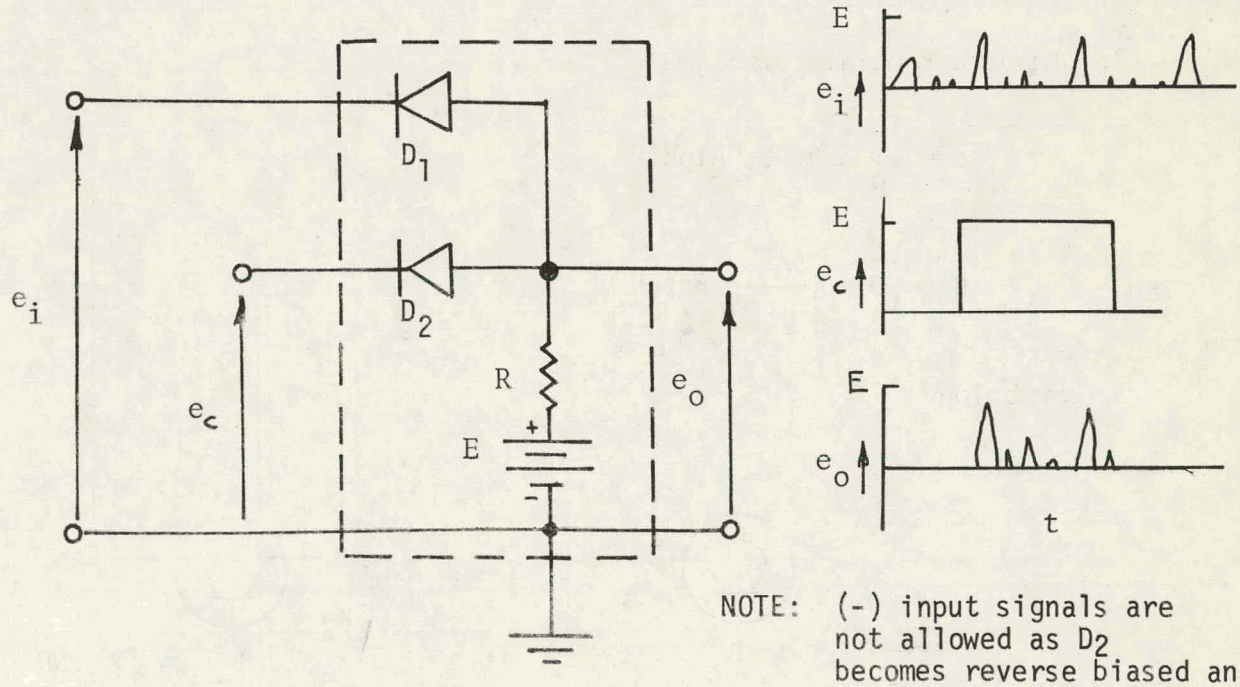
Diode Rules of Thumb

1. Anode is + (P) material
2. Cathode is - (N) material
3. Diode conducts whenever P is biased more positive with respect to N than barrier potential.

Diode Clipping Circuit



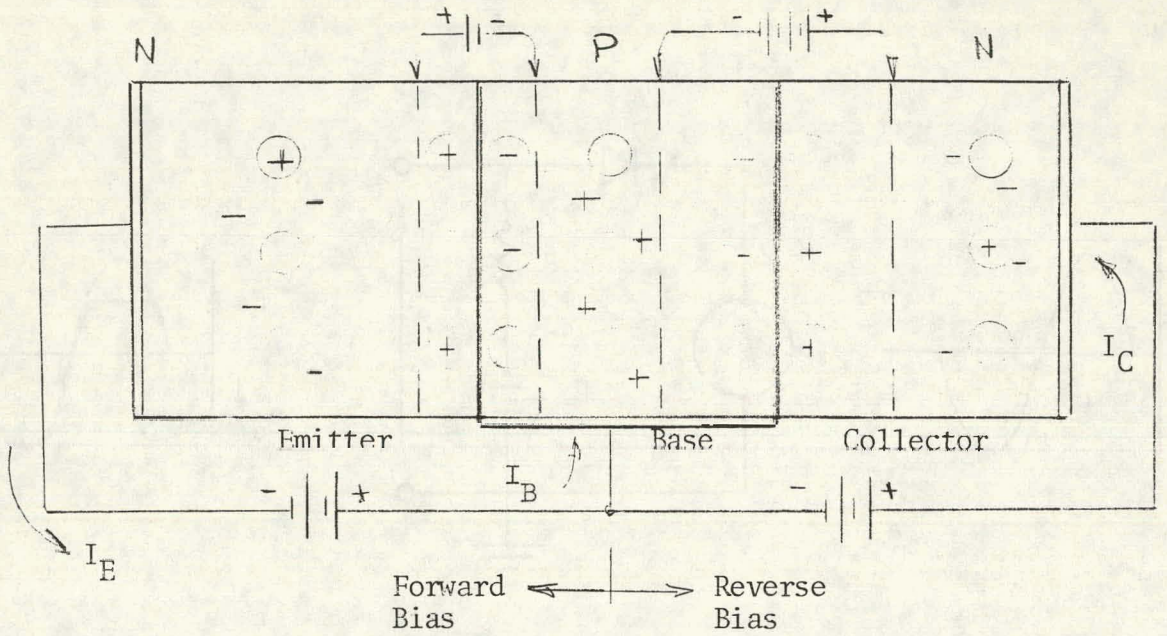
Diode Switch



NOTE: (-) input signals are not allowed as D_2 becomes reverse biased and e_o can not be switched off.

2.2.2 Transistor [6]

Junction Transistor Operation:



NPN Transistor, Common Base Configuration

NOTE:

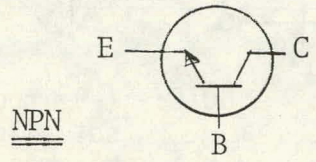
1. Emitter-base junction is forward biased and collector-base junction is reverse biased in normal operation. Bias polarity depends on transistor type (NPN or PNP).

2. Important parameters:

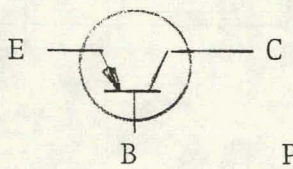
$$\alpha = \frac{I_C}{I_E} \approx 0.98, \text{"Alpha"}$$

$$\beta = \frac{I_C}{I_B} = \frac{\alpha}{1 - \alpha}, \text{"Beta"}$$

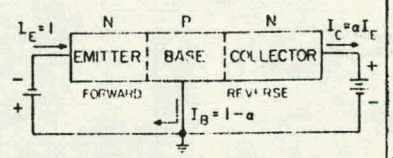
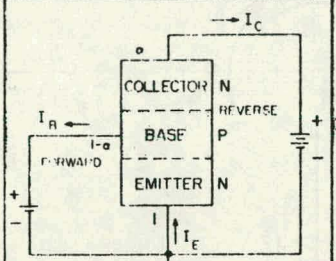
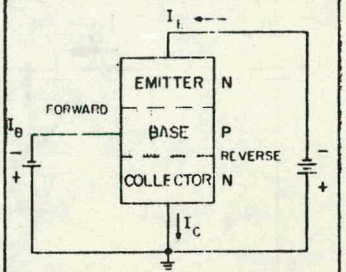
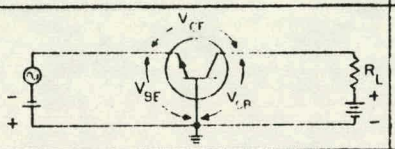
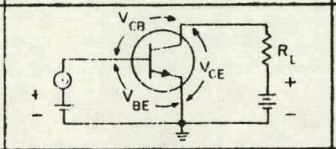
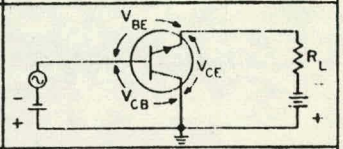
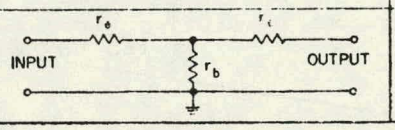
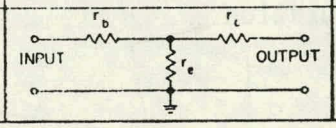
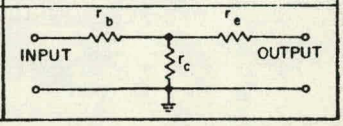
3. Symbology



NPN



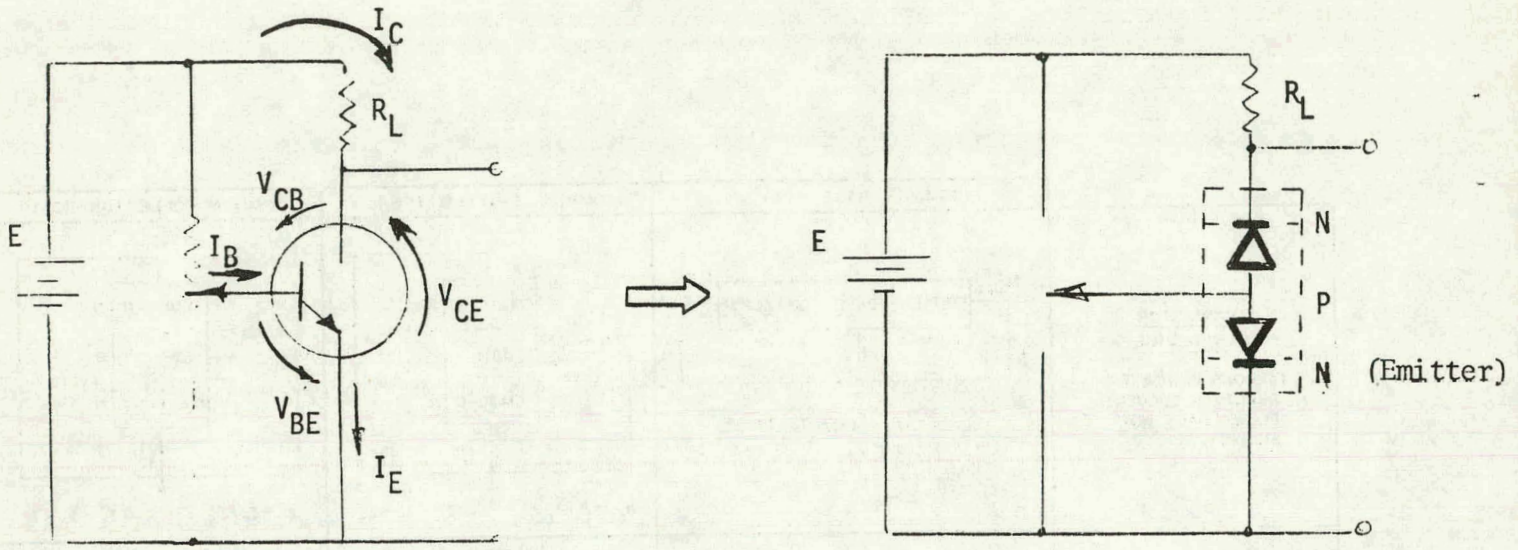
PNP

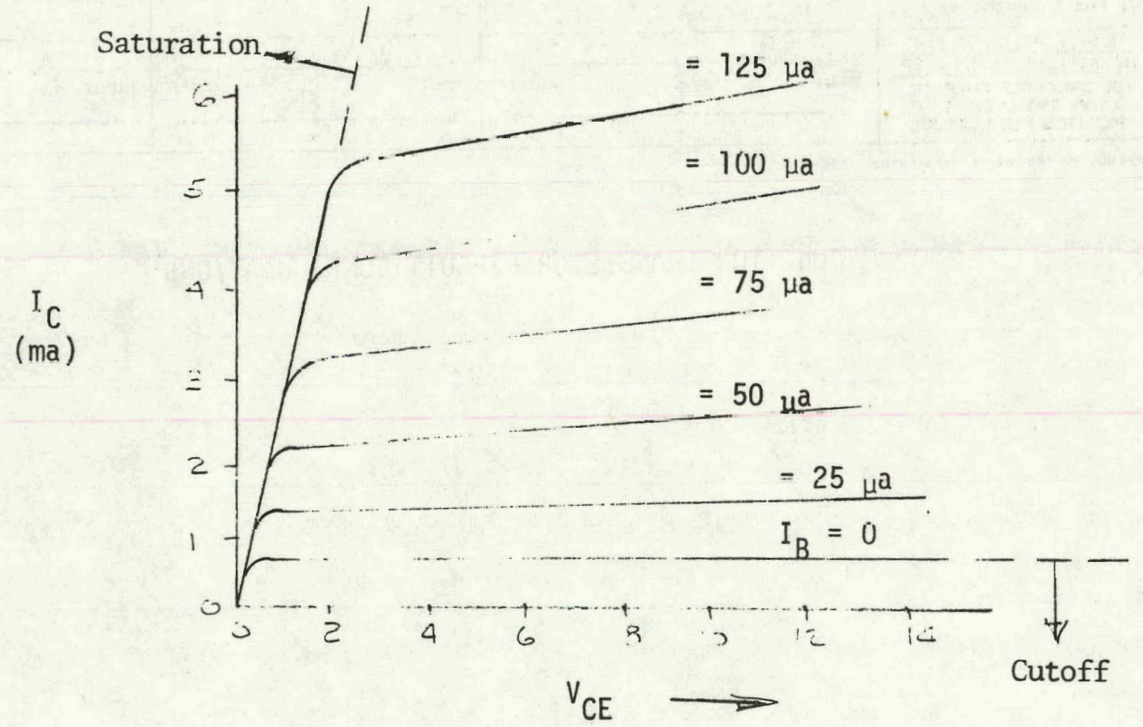
	COMMON BASE (CB)	COMMON EMITTER (CE)	COMMON COLLECTOR (CC)
TRANSISTOR AS A DEVICE (ARROWS INDICATE ELECTRON CURRENT FLOW. LOADS NOT SHOWN)	 $\alpha = \gamma\beta^{*}\alpha^{*}$ (SEE TEXT) $h_{FB} = I_C / I_E$ $I_C = I_E - I_B$	 $h_{FE} = \beta = I_C / I_B$ $I_C = I_E - I_B$	 $h_{FC} = I_E / I_B$ $I_C = I_E - I_B$
BASIC TRANSISTOR CIRCUITS SHOWING SIGNAL SOURCE AND LOAD (R_L)			
CHARACTERISTICS			
POWER GAIN *	YES	YES (HIGHEST)	YES
VOLTAGE GAIN *	YES (APPROX. SAME CE)	YES	NO (LESS THAN UNITY)
CURRENT GAIN *	NO (LESS THAN UNITY)	YES	YES
INPUT IMPEDANCE *	LOWEST ($\cong$ 50Ω)	INTERMEDIATE ($\cong$ 1.0K)	HIGHEST ($\cong$ 300K)
OUTPUT IMPEDANCE *	HIGHEST ($\cong$ 1.0 MEG.)	INTERMEDIATE ($\cong$ 50K)	LOWEST ($\cong$ 300Ω)
PHASE INVERSION	NO	YES	NO
SIMPLE T-EQUIVALENT NETWORK OF TRANSISTOR (DC ONLY - SEE CHAPTER 2 FOR SMALL-SIGNAL & HIGH FREQ. EQUIV. CIRCUITS)			

*DEPENDS ON TRANSISTOR, TERMINATIONS, ETC

JUNCTION TRANSISTOR CIRCUIT CONFIGURATIONS(6)

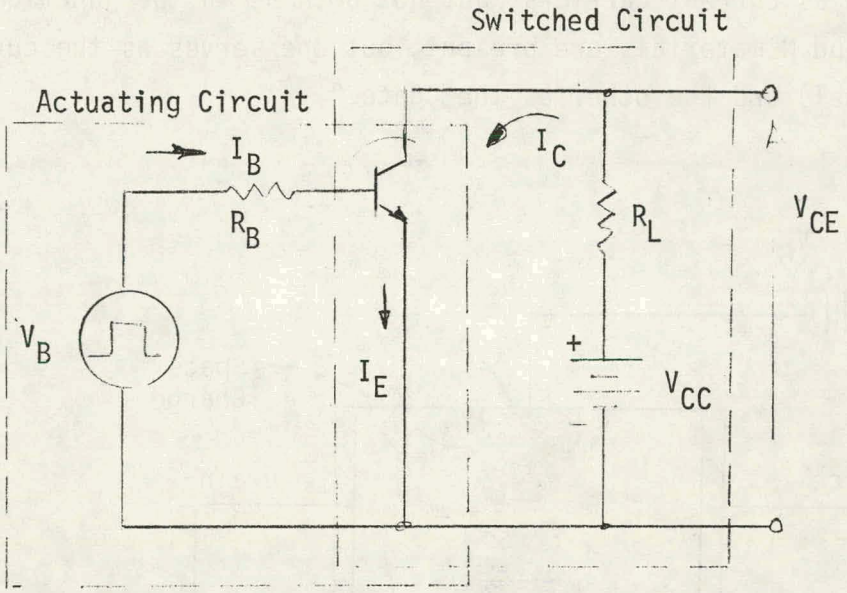
Common Emitter Configuration:





Typical Characteristic Curve, Common-Emitter Configuration.

Common Emitter NPN Switch:

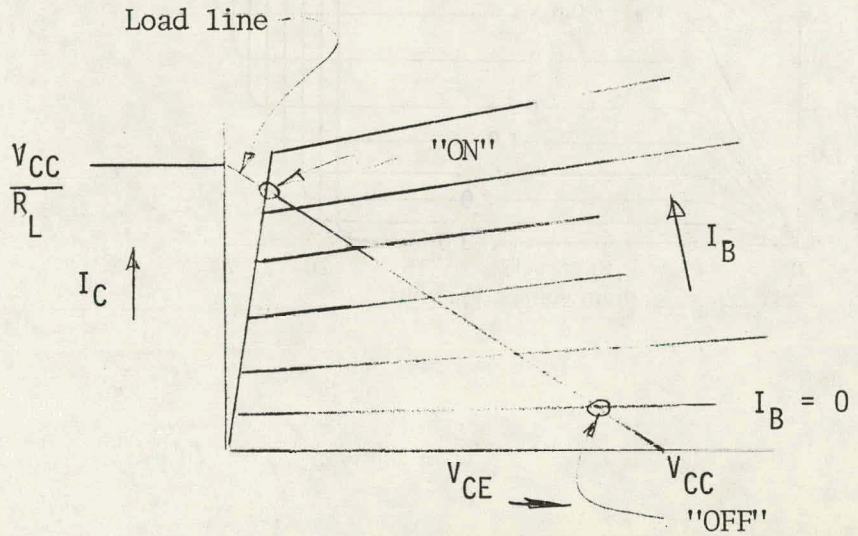


$$\Sigma v = 0$$

$$V_{cc} - I_C R_L - V_{CE} = 0$$

$$\therefore \quad I_C = -\frac{V_{CE}}{R_L} + \frac{V_{CC}}{R_L}$$

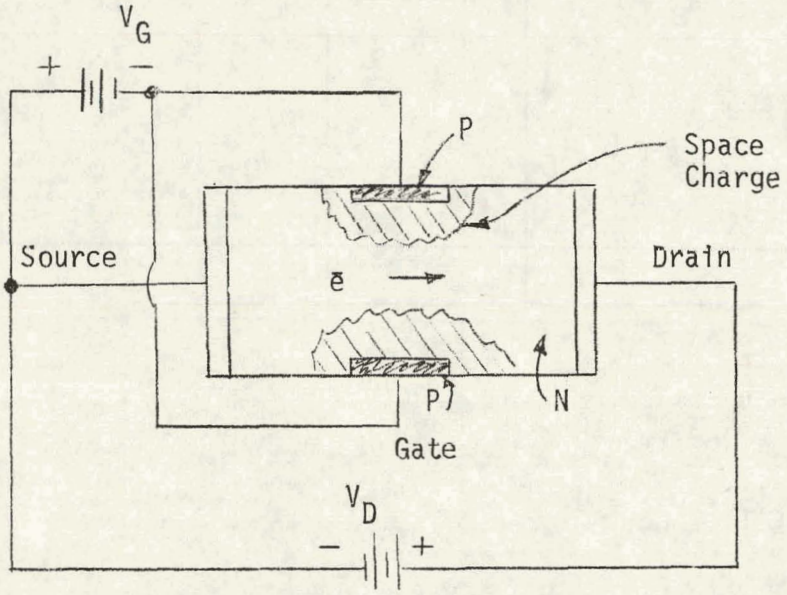
Also $$I_B = \frac{V_B - V_{BE\ (on)}}{R_B}$$



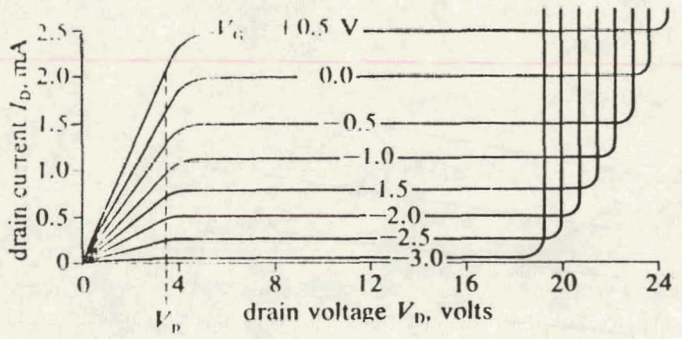
Load line superimposed on CE characteristic curves.

Field Effect Transistor (1)

The field effect transistor (FET) is a unipolar device in that either holes or $\bar{e}$ act as current carriers, but not both as in the junction transistor. Both P and N materials are present, but one serves as the current carrier (channel) and the other as the "gate."



In the N-channel FET device shown here, the voltage V_G controls the space charge and therefore the effective channel size. The characteristics of this device are as follows:



2.2.3 Electro-optical and electro-magnetic switches.[1]

Electro-optical:

Light sources

Source	Characteristics
Incandescent lamp	~ 100 ms response, broad spectrum.
Neon lamp	~ μs response, limited spectrum.
LED *	~ ns response, line spectrum.

Light detectors

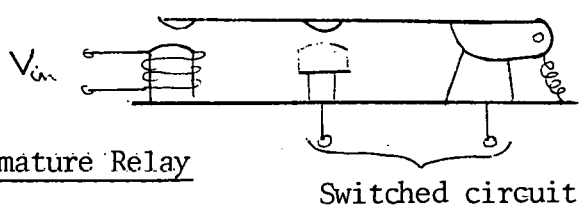
Detector	Characteristics
CdS, CdSe	~ 1 ms response, inexpensive.
Photovoltaic	No voltage supply needed.
Photodiode *	~ 10^{-3} ns response

* Most commonly found combined in digital system in the form of photo isolators.

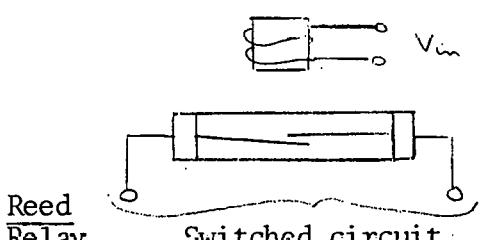
Electro-magnetic:

(a) Limited in application in digital circuits due to slow speed, contact bounce.

(b) Reed relays most often used for switching low-level analog signals, armature relay for high power applications.



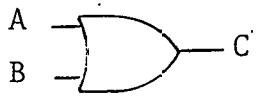
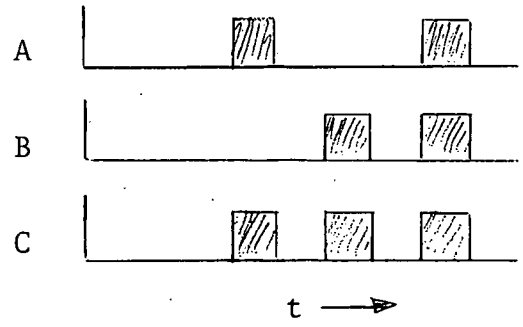
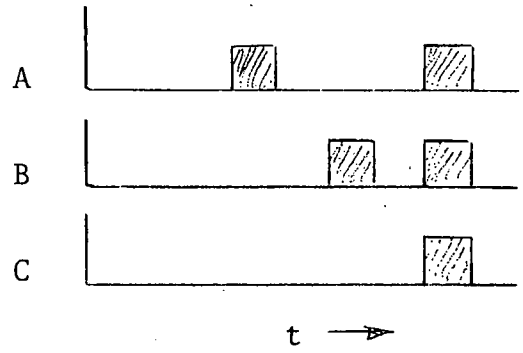
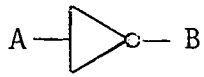
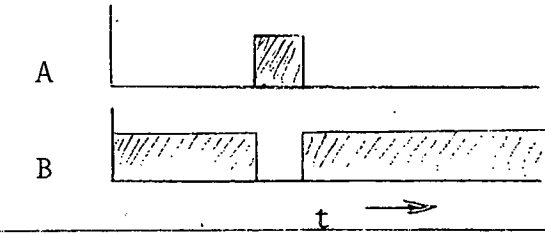
Armature Relay



Reed Relay

3. Combinatorial Logic

3.1 Basic Logic Functions

FUNCTION	SYMBOL	TRUTH TABLE	TIMING DIAGRAM
OR	 C = A + B	A B C 0 0 0 0 1 1 1 0 1 1 1 1	
AND	 C = A · B = AB	A B C 0 0 0 0 1 0 1 0 0 1 1 1	
NOT	 $B = \overline{A}$	A B 0 1 1 0	

3.2 Design of Combinational Circuitry

3.2.1 Boolean Algebra

Table of Basic Theorems

No.	Name	OR Operation (a)	AND Operation (b)
1	Single-Variable Theorems	$\overline{\overline{A}} = A$	
2		$A + 0 = A$	$A \cdot 0 = 0$
3		$A + 1 = 1$	$A \cdot 1 = A$
4		$A + A = A$	$A \cdot A = A$
5		$A + \overline{A} = 1$	$A \cdot \overline{A} = 0$
	Multiple-Variable Theorems		
6	Commutation	$A + B = B + A$	$AB = BA$
7	Absorption	$A + AB = A$	$A(A + B) = A$
8	DeMorgan's	$\overline{A + B} = \overline{A} \cdot \overline{B}$	$\overline{AB} = \overline{A} + \overline{B}$
9	Association	$A + (B + C) = (A + B) - C$	$A(BC) = (AB)C$
10	Distribution	$A + BC = (A + B)(A + C)$	$A(B + C) = AB + AC$
11	Auxillary	$A + \overline{A}B = A + B$	$A(\overline{A} + B) = AB$

APPLICATIONS OF BOOLEAN ALGEBRA TO CIRCUIT SIMPLIFICATION

Ex 3-1

What's the simplest implementation of the function $F_1 = A(\overline{A} + B)$?

Boolean manipulation:

$F_1 = A(\overline{A} + B)$

$= A\overline{A} + AB$ (theorem 10 b)

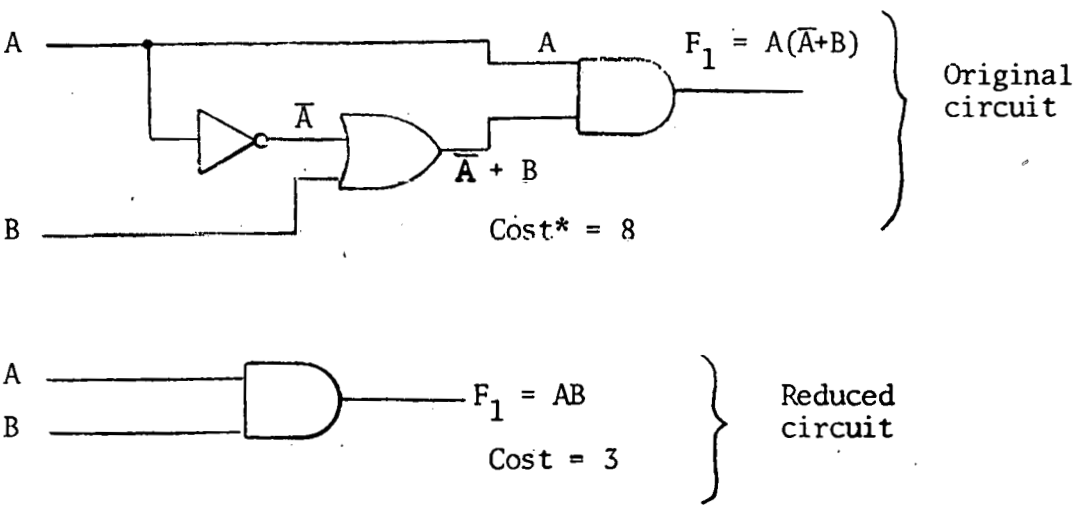
$= AB$ (theorem 5 b)

Check result using truth table:

A	B	AB	$A(\overline{A} + B)$
0	0	0	0
0	1	0	0
1	0	0	0
1	1	1	1

OK

Implementation:



* Cost is approximately proportional to the number of gate leads (input and output).

Ex 3-2

What's the simplest implementation of the function $F_1 = A + \overline{A}B$?

Boolean manipulation:

$F_1 = A + \overline{A}B$

$= A + AB + \overline{A}B$ (Theorem 7a)

$= A + B(A + \overline{A})$ (Theorem 10b)

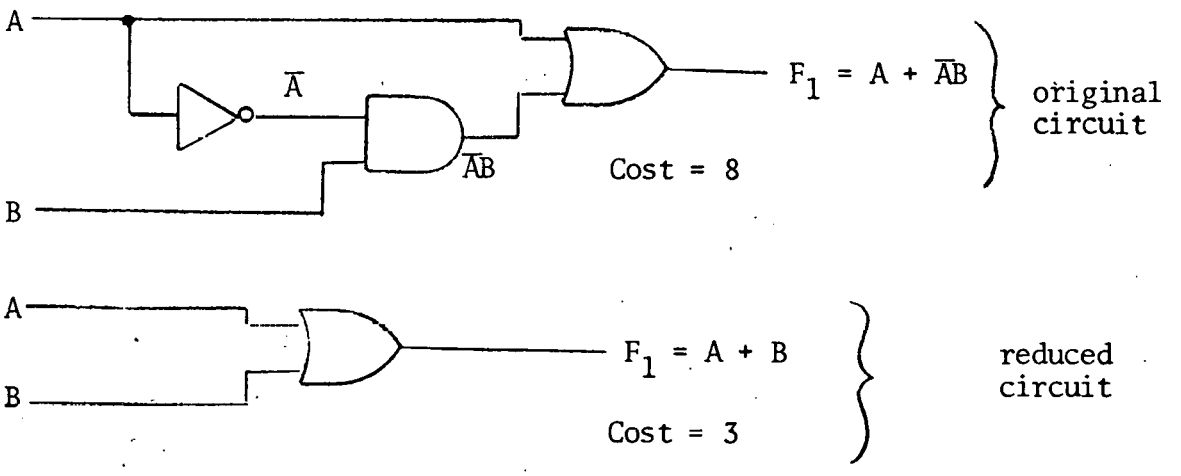
$= A + B$ (Theorem 5a, 3b)

Check result using truth table:

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

OK

Implementation:



Ex 3-3

You are an observer on a street corner. A car approaches the intersection signaling a left-hand turn. Design a circuit to predict whether or not the driver will come to a stop before making the turn. As input data, you have information to indicate

(1) whether the traffic signal is red or green

(2) whether or not the intersection is clear

(3) whether the driver is sober or intoxicated.

(a) Assign variables

Inputs

A = 1, green light
  = 0, red light
B = 1, intersection clear
  = 0, oncoming traffic
C = 1, intoxicated (always turns)
  = 0, sober

Output

D = 1, driver turns immediately
  = 0, driver stops

(b) Generate truth table:

A	B	C	D	
0	0	0	0	
0	0	1	1	$= \bar{A}\bar{B}C$
0	1	0	0	
0	1	1	1	$= \bar{A}B\bar{C}$
1	0	0	0	
1	0	1	1	$= A\bar{B}C$
1	1	0	1	$= AB\bar{C}$
1	1	1	1	$= ABC$

(c)

$D = \bar{A}\bar{B}C + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$= C[\bar{A}\bar{B} + \bar{A}B + A\bar{B}] + AB(\bar{C} + C)$ (Theorem 10 b)

$= C[A(\bar{B} + B) + A\bar{B}] + AB$ (Theorem 5a, 10 b)

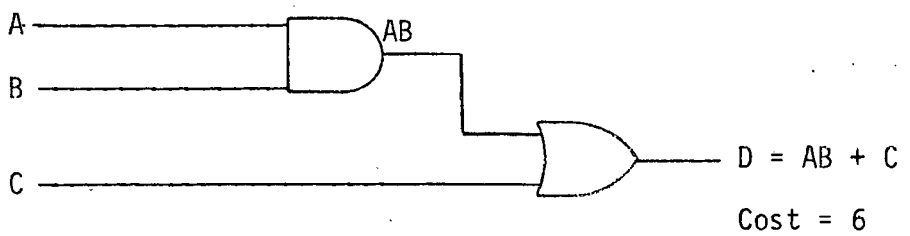
$= C[\bar{A} + A\bar{B}] + AB$ (Theorem 5a)

$= C[\bar{A} + \bar{B}] + AB$ (Theorem 11a)

$= C[\overline{AB}] + AB$ (Theorem 8b)

$= C + AB$ (Theorem 11a)

(d) Implement:



Ex. 3-4

Derive a circuit for a half adder (adds two binary digits, provides for carry-out but not carry-in).

(a) Assign variables:

all possible combinations

A	0	0	1	1
B	0	1	0	1
C_o, D	00	01	01	10

(b) Generate truth table:

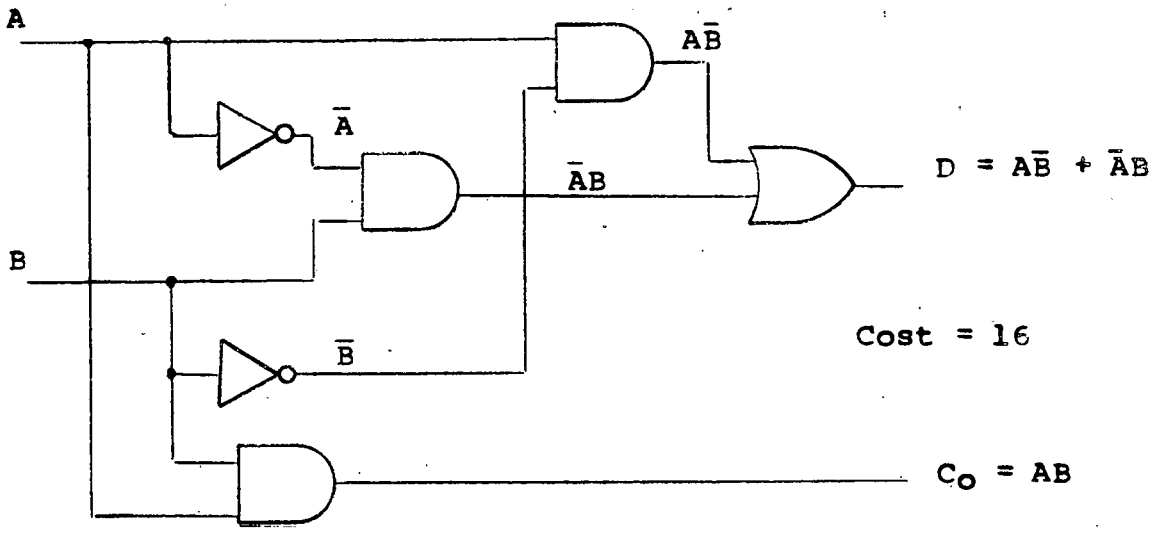
A	B	C_o	D
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

(c) Derive logical expression:

$C_o = AB$

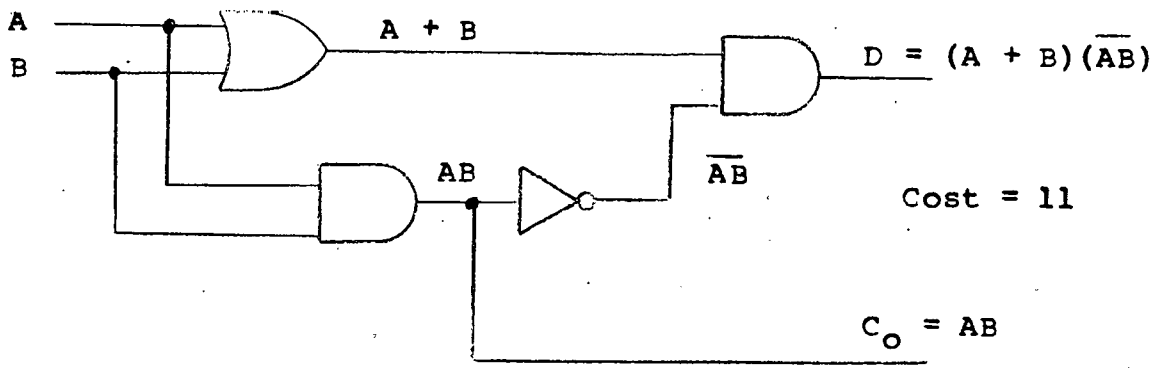
$D = \bar{A}B + A\bar{B}$

(d) Implement:



(e) Minimize circuit:

$$
\begin{aligned}
D &= \overline{A}B + A\overline{B} \\
&= \overline{A}(A + B) + \overline{B}(B + A) && \text{(Theorem 11b)} \\
&= (A + B)(\overline{A} + \overline{B}) && \text{(Theorem 10b)} \\
&= (A + B)(\overline{AB}) && \text{(Theorem 8b)}
\end{aligned}
$$



3.2.2 Karnaugh Map[3]

The Karnaugh map is a circuit reduction procedure based on simplifying recognition of the Boolean identity

$$AB + A\bar{B} = A$$

Whenever we have two ANDings of the same group of variables, and a single variable appears in the NOTed form in just one of them (such as B above), the technique applies. For instance, the technique would prove useful in analyzing the function

$$F_1 = \bar{X}YZ + \bar{X}\bar{Y}Z$$
$$= \bar{X}Z$$

but not the function

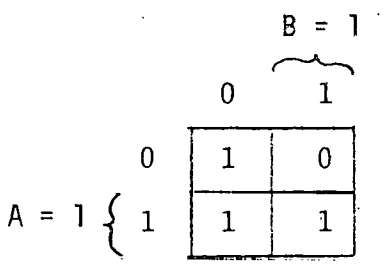
$$F_2 = \bar{X}YZ + \bar{X}\bar{Y}\bar{Z}$$

The Karnaugh map is a graphical procedure for recognizing these combinations in functions of up to four variables.

The map is constructed by taking data from a truth table and "mapping" it onto a grid, each square of which represents one row in the truth table. The square contains the value of the output variable for that row. Thus:

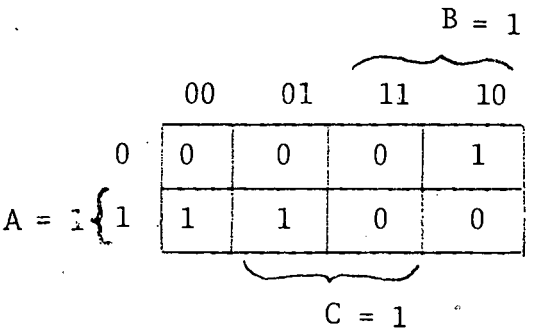
2 Variables

A	B	C
0	0	1
0	1	0
1	0	1
1	1	1



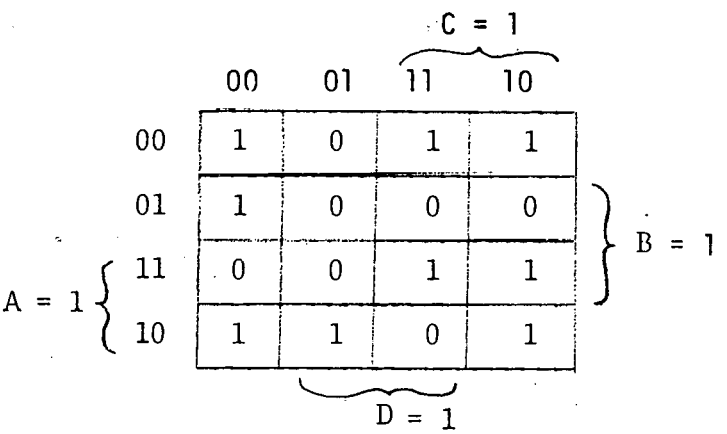
3 Variables

A	B	C	D
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0



4 Variables

A	B	C	D	E
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



Any two adjacent squares in the table (up-down or across) correspond to a change in just one variable. Therefore adjacent logic 1's imply that the identity discussed above can be applied to the reduction of the Boolean expression for the function. Adjacent logic 1's in groups of 2, 4, or 8 are referred to as two-square, four-square, or eight-square implicants. Because of the way the map is constructed, the extreme left-hand row of squares is "adjacent" to the right, as is the top row "adjacent" to the bottom. The variables which differ between the squares drops out of the Boolean expression for the square involved. Consequently greatly simplified Boolean expressions can be derived directly from the Karnaugh map.

Ex. 3-5

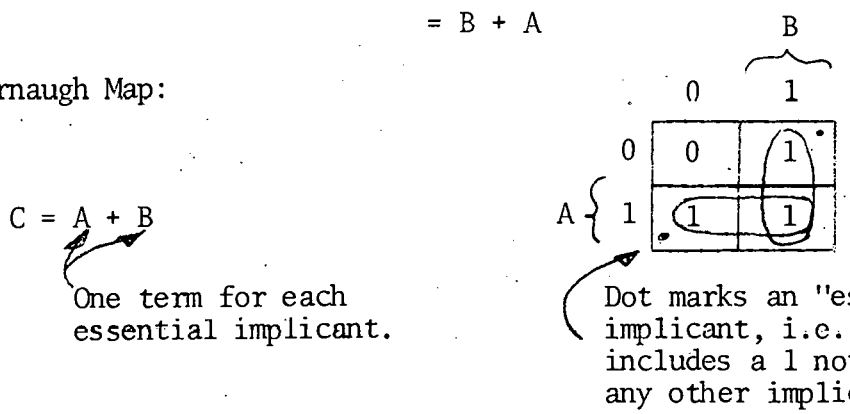
Derive the Boolean expression for the following truth table:

A	B	C	
0	0	0	
0	1	1	$= \bar{A}B$
1	0	1	$= A\bar{B}$
1	1	1	$= AB$

By Boolean Reduction:

$$C = \bar{A}B + A\bar{B} + AB$$
$$= \bar{A}B + A(\bar{B} + B)$$
$$= \bar{A}B + A$$
$$= B + A$$

By Karnaugh Map:



Ex. 3-6

Derive the Boolean expression for the following truth table:

A	B	C	D	
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$= \bar{A}BC$
1	0	0	1	$= A\bar{B}\bar{C}$
1	0	1	1	$= A\bar{B}C$
1	1	0	1	$= AB\bar{C}$
1	1	1	1	$= ABC$

By Boolean reduction:

$$D = \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

$$= BC(\bar{A} + A) + A\bar{C}(\bar{B} + B) + A\bar{B}C$$

$$= BC + AC + ABC$$

$$= C(B + A\bar{B}) + A\bar{C}$$

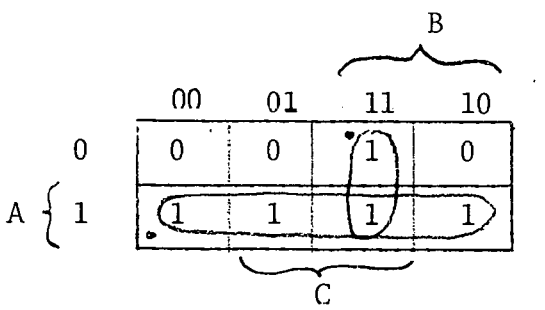
$$= C(B + A) + A\bar{C}$$

$$= CB + CA + A\bar{C}$$

$$= CB + A$$

By Karnaugh map:

$D = A + BC$

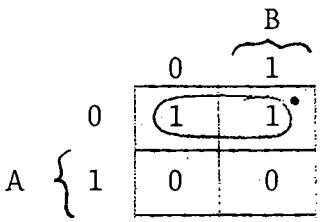


Ex. 3-7

The following truth tables and maps further illustrate the technique:

(a)

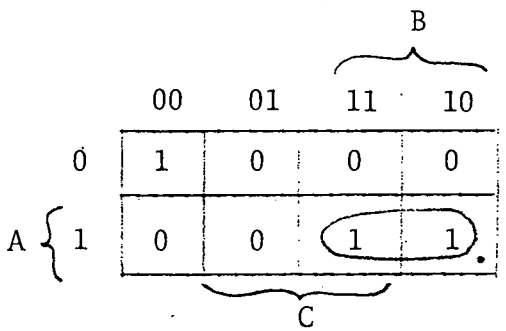
A	B	C
0	0	1
0	1	1
1	0	0
1	1	0



$C = \bar{A}$

(b)

A	B	C	D
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

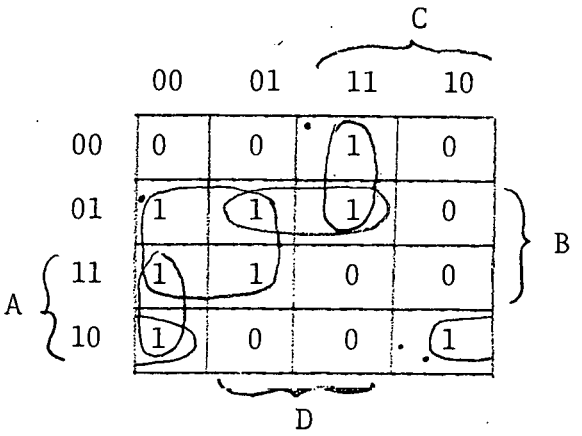


$F = AB + \bar{A}\bar{B}\bar{C}$

Completely isolated logic one's must be included as usual.

(c)

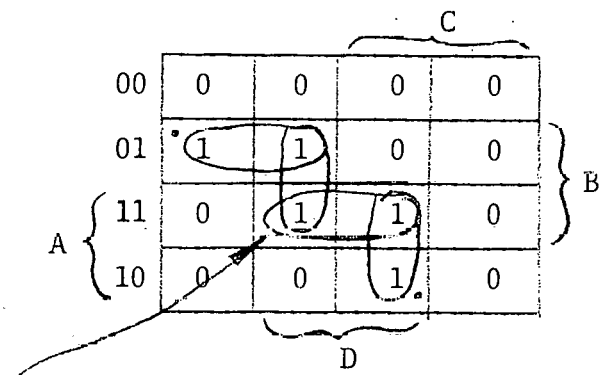
A	B	C	D	E
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	0
1	1	1	1	0



$E = \bar{A}CD + B\bar{C} + A\bar{B}\bar{D}$

(d)

A	B	C	D	E
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	0
1	1	1	1	1



In some instances the essential implicants do not include one or more logic 1's. Terms should be added to the final function for the nonessential implicants required to cover each logic one just once.

Therefore, for the above truth table and map,

$E = \bar{A}B\bar{C} + ACD + ABD$

or

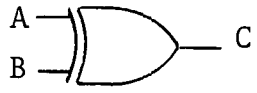
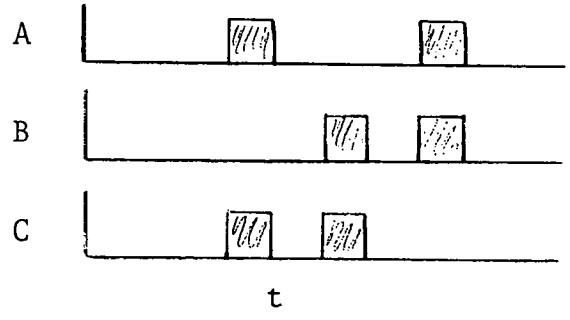
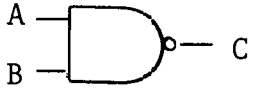
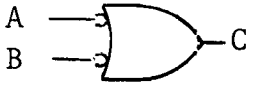
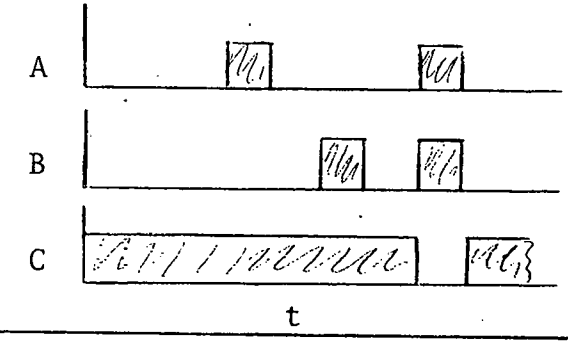
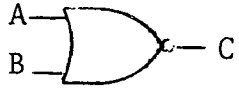
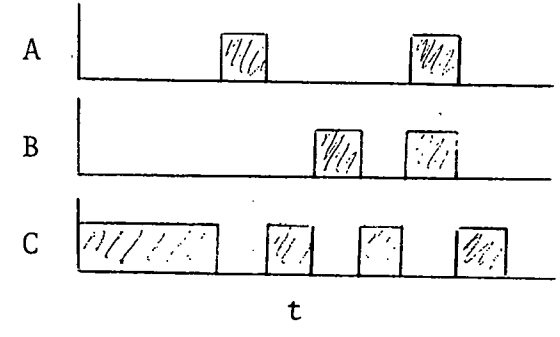
$E = \bar{A}B\bar{C} + ACD + B\bar{C}D$

Nonessential implicants.

To summarize the Karnaugh map procedure:

1. Map the truth table onto the Karnaugh matrix.
2. Circle all implicants of 2, 4, or 8 blocks.
3. Mark all essential implicants and completely isolated logic 1's.
4. Write the Boolean expression for the function as the OR'ed expressions for each of the essential implicants and any isolated logic 1's.
5. OR to this function expressions for as many nonessential implicants as necessary to cover all remaining logic 1's just once.

3.3 XOR, NAND, NOR Gates

Function	Symbol	Truth Table	Timing Diagram
XOR	 $C = A\bar{B} + B\bar{A}$ $= A \oplus B$	A B C 0 0 0 0 1 1 1 0 1 1 1 0	
NAND	 Also  $C = \overline{AB} = \bar{A} + \bar{B}$	A B C 0 0 1 0 1 1 1 0 1 1 1 0	
NOR	 Also  $C = \overline{A + B} = \bar{A}\bar{B}$	A B C 0 0 1 0 1 0 1 0 0 1 1 0	

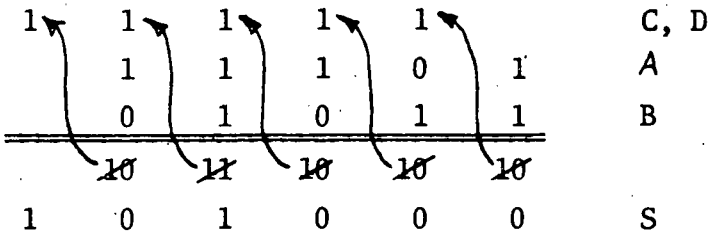
3.3.1 Use of XOR Gates

The exclusive OR function (XOR) has the basic form

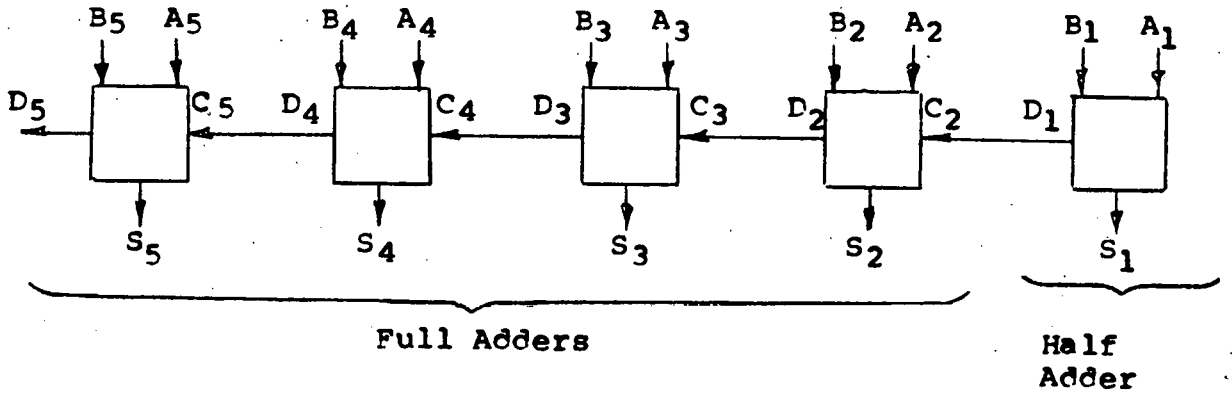
$$\bar{A}B + A\bar{B} = A \oplus B$$

As the truth table for this function shows, $A \oplus B$ is logic one when A or B is logic one, but not both.

Ex. 3-8 - Full Adder - Previously we studied the half-adder for summing two single-digit binary numbers. For multiple digit numbers, we must consider the possibility of a carry-in from a preceeding column:



Functionally, this binary addition process is equivalent to the circuit



The full adder, therefore, performs the function of adding together three single-digit binary numbers, one of which is the carry-out from the previous column. The truth table for this operation is as follows:

A	B	C	D	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

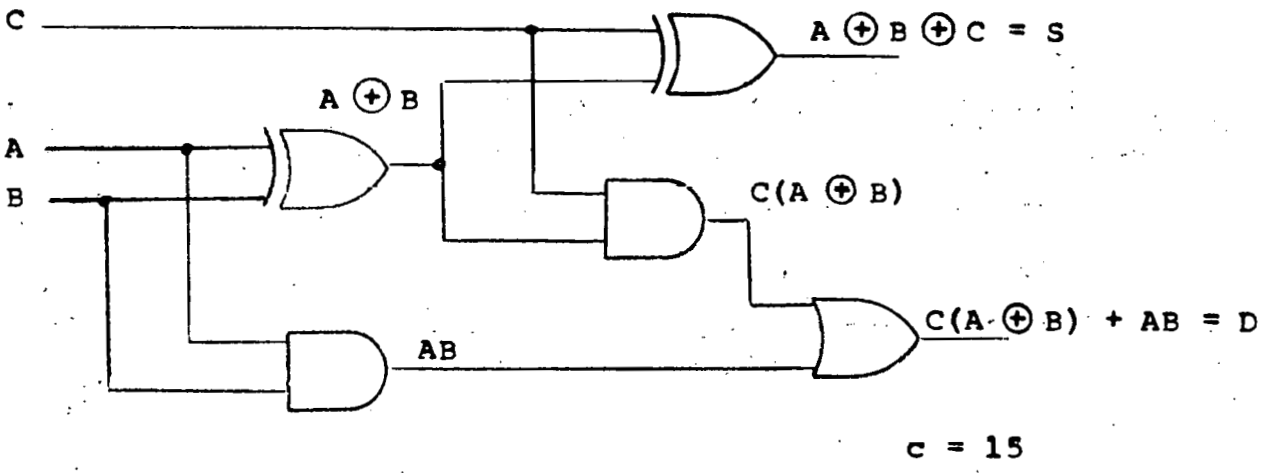
In terms of XOR gates:

$$
\begin{aligned}
S &= \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC \\
&= C(\bar{A}\bar{B} + AB) + \bar{C}(\bar{A}B + A\bar{B}) \\
&= C(\bar{A} + B)(\bar{B} + A) + \bar{C}(\bar{A}B + A\bar{B}) \\
&= C\overline{(A\bar{B})}\,\overline{(B\bar{A})} + \bar{C}(\bar{A}B + A\bar{B}) \\
&= C\overline{(A\bar{B} + B\bar{A})} + \bar{C}(\bar{A}B + A\bar{B}) \\
&= C \oplus (A \oplus B) \\
&= C \oplus A \oplus B
\end{aligned}
$$

and

$$
\begin{aligned}
D &= \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC \\
&= C(\bar{A}B + A\bar{B}) + AB(\bar{C} + C) \\
&= C(A \oplus B) + AB
\end{aligned}
$$

Implementing in hardware:



Karnaugh Map: From the Karnaugh map for D

"S" Map

A \ B	00	01	11	10
0	0	1	0	1
1	1	0	1	0

C

"D" Map

A \ B	00	01	11	10
0	0	0	1	0
1	0	1	1	1

C

we get the reduced equation

$$D = AC + BC + AB$$

Although the "S" map contains no implicants, it can be shown that the pattern of 1's present (equal number of 1's and 0's in a checkerboard pattern) is characteristic of the XOR function and the final equation for S can also be written directly fron the Karnaugh map

$$S = A \oplus B \oplus C$$

The use of the Kanaugh map in recognizing the XOR function with larger numbers of variables is not very straightforward, however, and we will not go into it any deeper. It's sufficient to say that the XOR function is available, the Karnaugh map can be useful in applying it, and when the XOR function is used it can result in considerable circuit simplification. For the above full adder, implementation using AND/OR logic has a relative cost c = 43 versus the c = 16 for XOR logic.

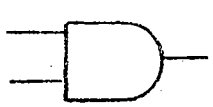
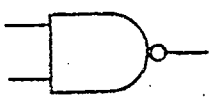
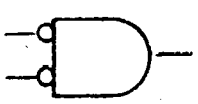
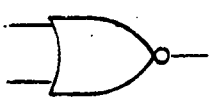
3.3.2 Use of NAND/NOR Logic [3]

Our techniques to date have dealt primarily with AND and OR gates. However, the gates most commonly available for implementing digital systems (see section 3.4) are of the NAND, NOR type. To implement a circuit in NAND or NOR logic, the procedure is as follows:

a) Obtain a minimized Boolean expression for the function in terms of AND's and OR's.

b) Draw the circuit using AND's, OR's and NOT functions.

c) Substitute for each AND gate the symbol for the "AND"-type representation of NAND or NOR gates to be used, and for each OR gate substitute the symbol for the "OR"-type representation.

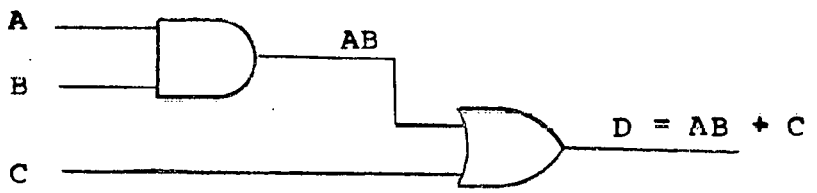
	Symbol to be Substituted	
Original Gate	NAND Logic	NOR Logic
		
		

d) Now check the resulting inversion and add whatever inverters are necessary to make the circuit functionally equivalent to its AND/OR counterpart. Remember, $\overline{AB} \neq \bar{A}\bar{B}$ and $\bar{A} + \bar{B} \neq \overline{A + B}$.

Ex. 3-9

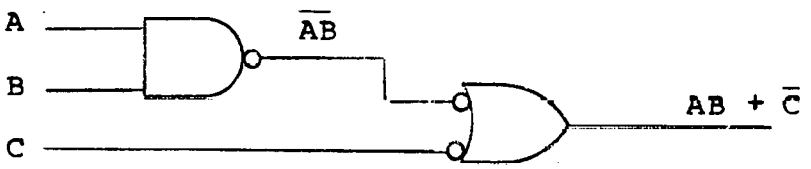
Referring back to Ex. 3-3, the final circuit implementation using AND/OR logic was

$$D = AB + C$$

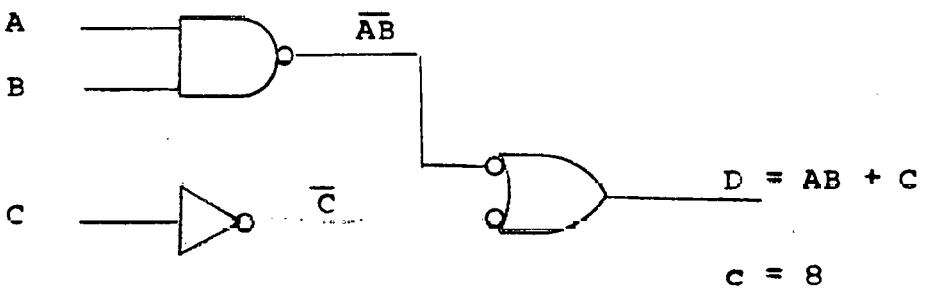


c = 6

a) To convert this circuit to NAND logic, we first substitute the appropriate "AND-" and "OR-" type representations

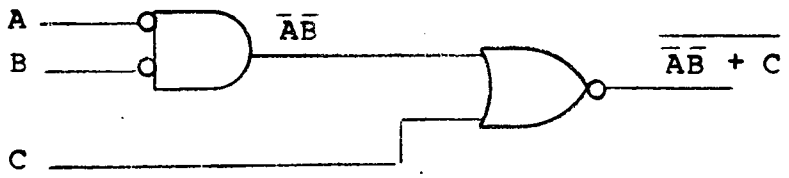


and add whatever inverters are necessary to get the correct final result.

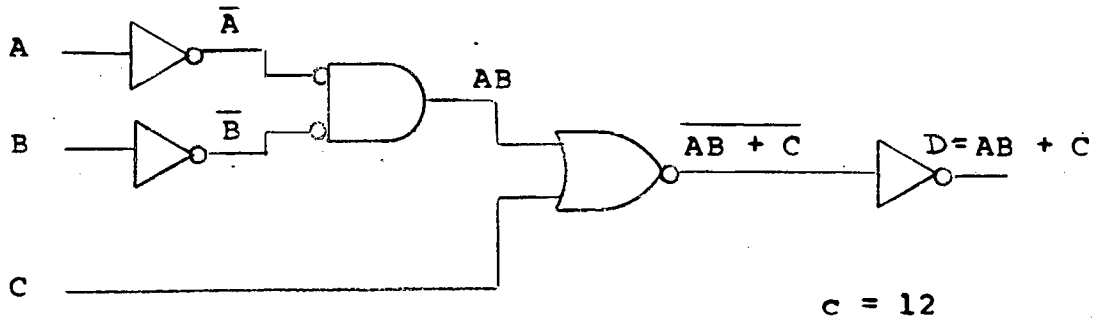


c = 8

b) To convert the original circuit to NOR logic, we first substitute the appropriate "AND-" and "OR-" type representations.



and then add inverters as necessary.

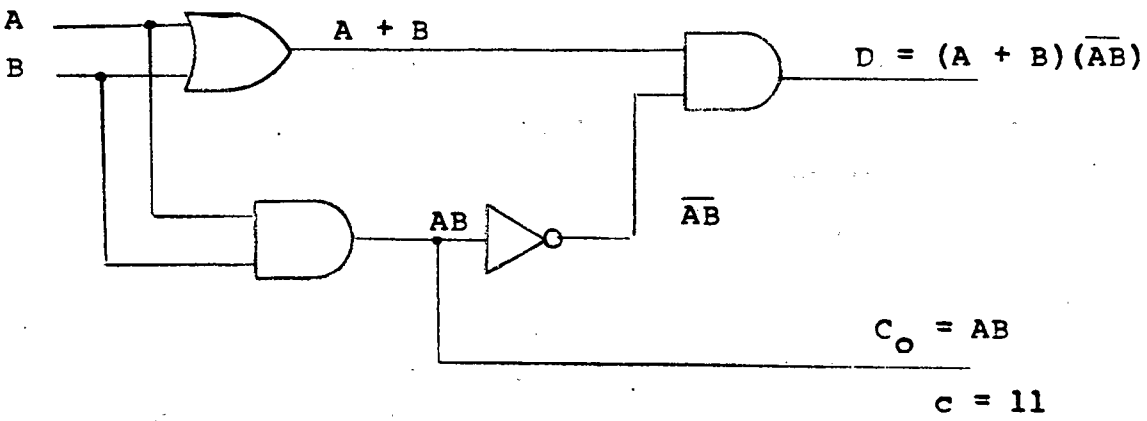


Ex. 3-10

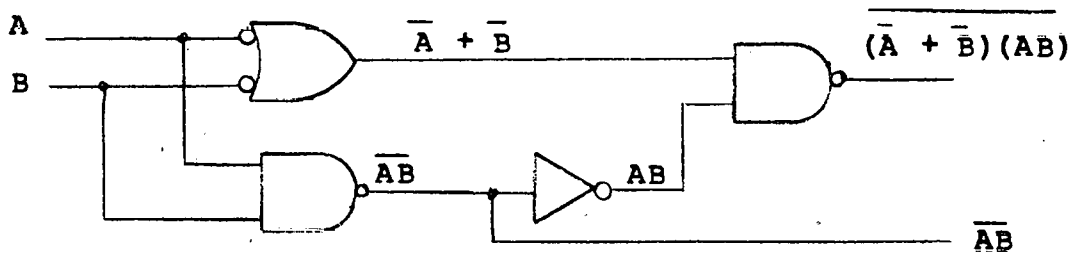
For the half-adder of Ex. 3-4, the final circuit implementation was

$$D = (A + B)(\overline{AB})$$

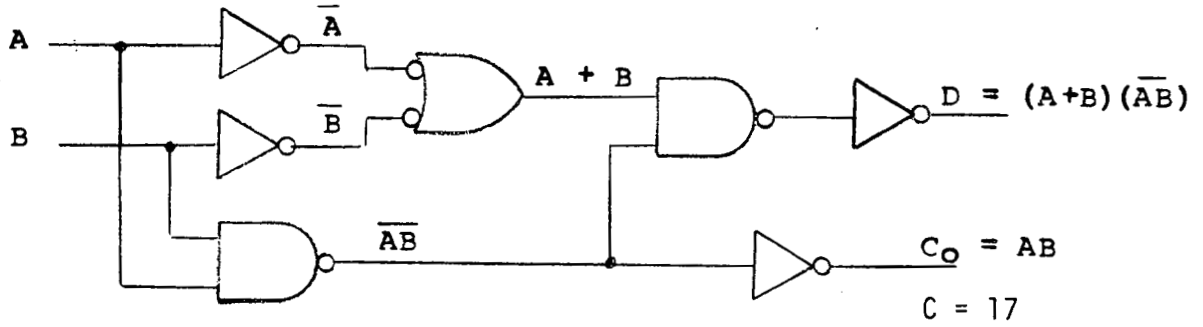
$$C_o = AB$$



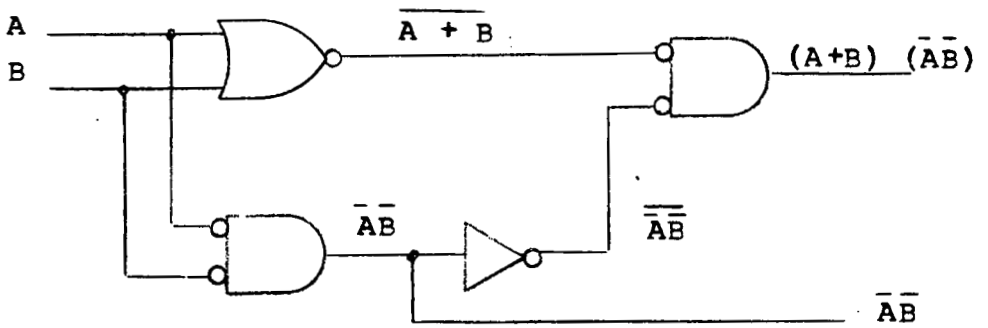
a) By substituting the NAND logic "AND-" and "OR-" type representations we get



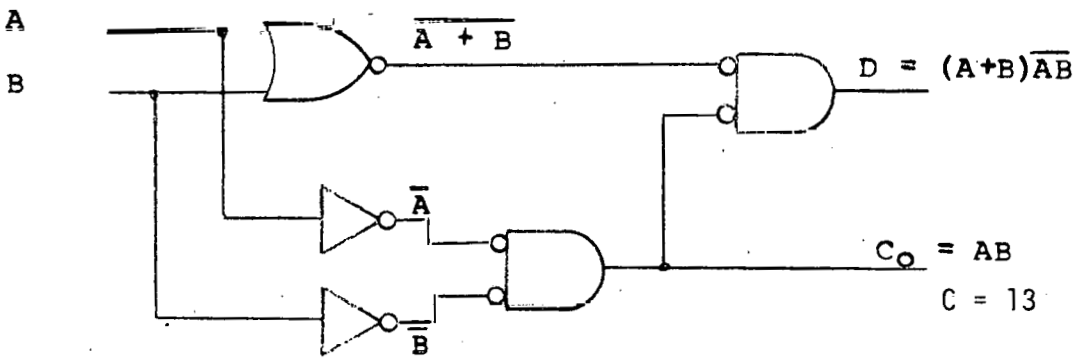
and after correcting the inversions we get



b) By substituting the NOR logic "AND-" and "OR-type" representations we get



and with the inversion corrected the circuit becomes



3.4 Hardware Logic Families

3.4.1 Gate Structures [3]

The designer has a number of logic families to choose from in building a system. The family names refer to the general structure of the devices -

RTL : Resistor-transistor logic

DTL : Diode-transistor logic

HTL : High threshold logic

TTL : Transistor-transistor logic

ECL : Emitter-coupled logic

CMOS : Complementary metal-oxide semiconductor

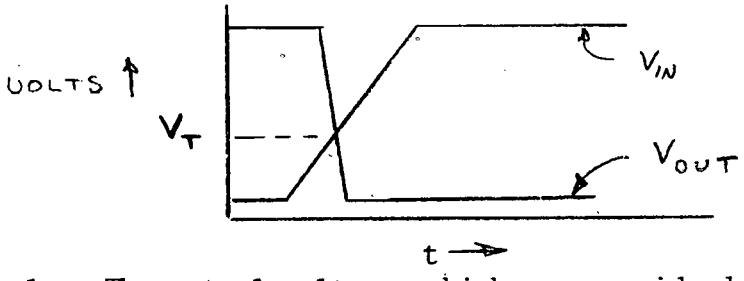
Generally speaking, RTL and DTL represent old technology, HTL and TTL represent current technology, and ECL and CMOS represent more advanced technology. Although TTL is currently by far the most available logic type, the CMOS family is expanding rapidly and may soon overtake TTL. The basic gate structure for all six logic types is illustrated on the following page.

3.4.2 Performance Characteristics [7]

The basic performance characteristics by which logic devices are specified are as follows :

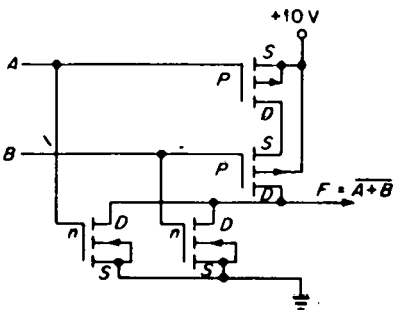
Speed - The frequency that a flip-flop (see section 4.0) can be toggled)

Threshold voltage - That voltage which when applied to all the inputs of a NAND, NOR, or NOT gate produces the same output voltage. For Texas Instruments (TI) 54/74 TTL logic, V_T = 1.5 v

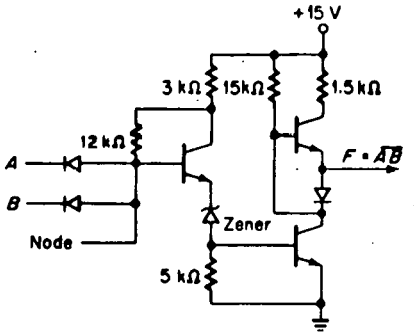


Logic levels - The actual voltages which are considerd to be "logic 1" and "logic 0". These are generally related to input voltages required to produce these voltages at the outputs. For instance for TI's 54/74 TTL logic

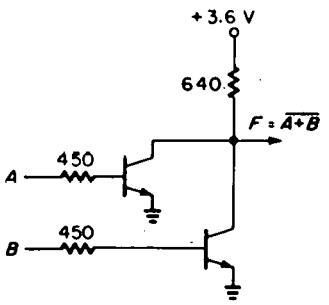
CMOS



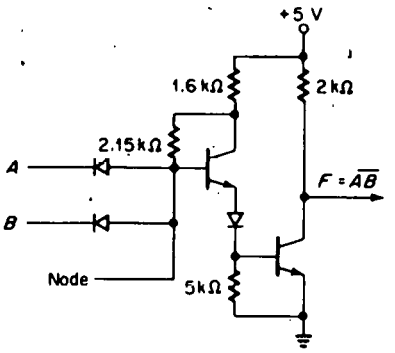
HTL



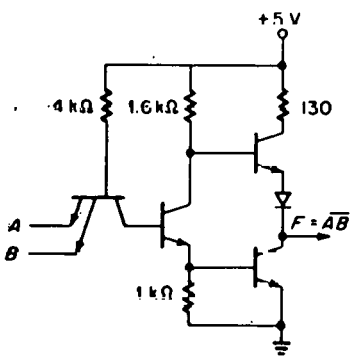
RTL



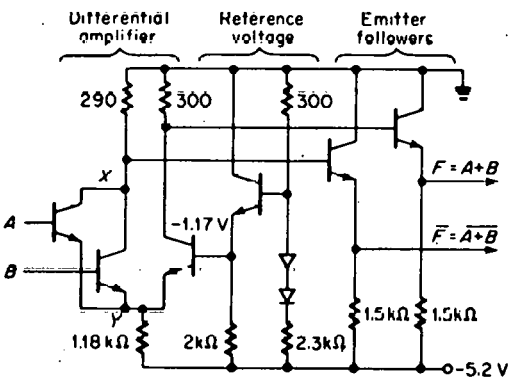
DTL



TTL



ECL



Typical Gate Structures (3)

	Output Voltages	
Gate Input	Guaranteed Min/Max	Typical Value
< .8 V	> 2.4 V	3.3 V, logic 1
> 2.0 V	< .4 V	.2 V, logic 0

Noise immunity - Refers to the amount of noise that can be picked up on the wire connecting the output of one gate to the input of a second without causing a state transition on the second. This value is therefore different depending on whether the second gate is at logic 1 or 0, and can be calculated at dc conditions as

Logic 1 dc noise immunity = logic 1 output - V_T

Logic 0 dc noise immunity = V_T - logic 0 output

Noise immunity is also a dynamic consideration, and as shown in the accompanying charts, larger noise pulses can be tolerated if they are of shorter duration.

Fanout - The number of gates inputs that can be driven by a single gate and still maintain the "guaranteed" logic outputs. The fanout for a gate is often stated in terms of arbitrary load units, as are the loadings imposed by a driven gate input on the driving gate. That is, a NAND gate may have a fanout of 10 load units, whereas each NAND input may be 2 load units. Therefore the output of one such gate could be used to drive the inputs of 5 others.

Propagation delay - In a gate, the delay between transition of the input to transition of the output. Typical propagation delays are in terms of nanoseconds. For comparison purposes, a bare wire delays a signal approximately 1 ns per foot of length.

Power consumption - The different logic lines have widely varying power consumptions, from the minimum of CMOS to the maximum of ECL. Power consumptions are generally stated for each IC device.

Temperature Range - Specifications generally state the recommended operating temperature. For example, in TI's 54/74 TTL logic the operating temperatures are:

5400 Series - -55°C to 125°C

7400 Series - 0°C to 70°C

Logic type		Specific logic line (second-sourced by many companies)		Relative speed (worst-case flip-flop toggling rate) (MHz)	Noise immunity		Logical flexibility		Gate power dissipation at 50% duty cycle (mW)	Power-supply voltage (V)
					Internal	External	Gate fanout capability	Dot AND/OR capability		
CMOS	Complementary metal-oxide semiconductor	RCA COS/MOS		2	Excellent	Good	50	No	3* (at 1 MHz)	+10
HTL	High-threshold logic	Motorola MC660 series		2	Excellent	Excellent	10	For some gates	28	+15
RTL	Resistor-transistor logic	Fairchild 900 series		4	Fair	Fair	5	No	12	+3.6
DTL	Diode-transistor logic	Fairchild 930 series		8	Good	Good	7	Yes	14	+5
TTL or I²L	Transistor-transistor logic	Texas Instruments	54L / 74L (low power)	3	Good*	Good	10	For some gates	1	+5
			54 / 74	15			10	For some gates	10	+5
			54H / 74H (high speed)	25			10	For some gates	23	+5
ECL	Emitter-coupled logic	Motorola	MECL II	70	Excellent	Fair	25	Yes	32	−5.2
			MECL III	300			7	Yes	240	−5.2

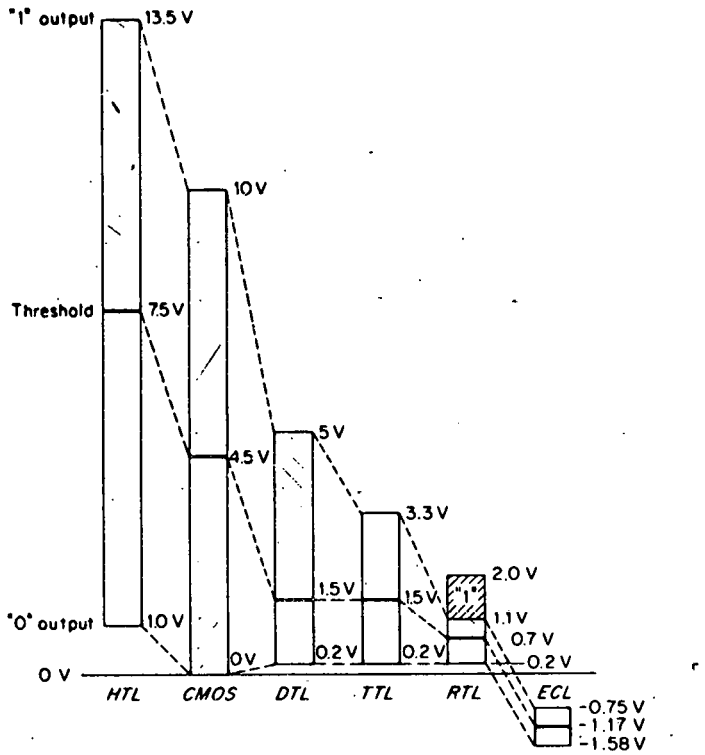
* TTL logic achieves good internal noise immunity only if switching transients on the power supply line are locally suppressed with one 0.01-μF capacitor per eight integrated circuit packages.

*Power dissipation proportional to frequency down to 2 μW.

Characteristics of different logic lines.(7)

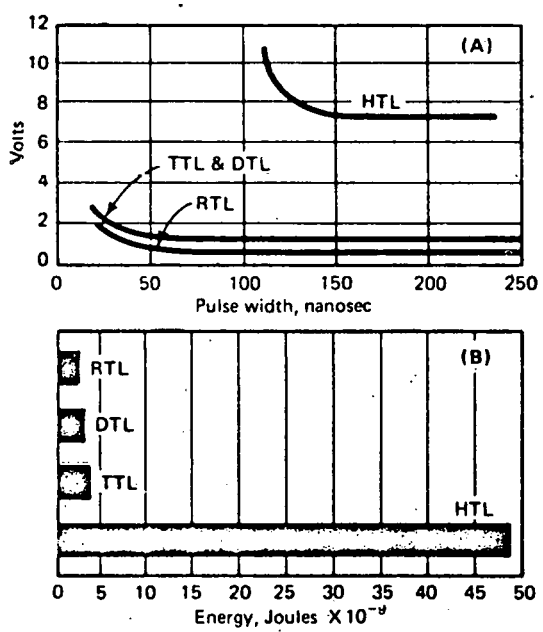
		RTL	DTL	HTL	TTL
NAND gates	Quad 2-input	x	x	x	x
	Triple 3-input		x	x	x
	Dual 4-input		x	x	x
	Dual 5-input		x		x
	Single 8-input		x		x
	Single 10-input		x		x
NOR gates	Quad 2-input	x	x		x
	Triple 3-input	x			
	Dual 4-input	x			
Inverters	Hex	x	x	x	x
	Hex & strobe			x	
	Hex & exp. inputs		x		
	Hex & h.v. outputs		x	x	x
Exclusive-OR	Quad 2-input	x	x	x	x
Exclusive-NOR	Quad 2-input				x
AND-OR invert	Single				x
	Dual			x	x
Line devices	Quad 2-input		x		x
	Dual 4-input		x	x	x
Buffers	Quad 2-input		x		x
	Dual 4-input	x	x	x	x
AND gates	Quad 2-input	x	x		x
	Triple 3-input				x
	Dual 4-input				x
OR gates	Quad 2-input	x	x		x
Majority logic	Dual 3-input				x

Types of Logic Gates(7)

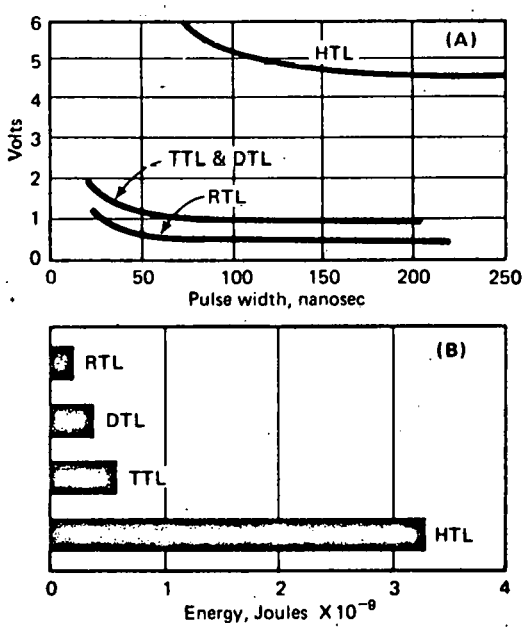


Typical output and threshold voltages for six logic types.(7)

Signal line noise immunity. The threshold noise level as a function of noise pulse width on signal lines is plotted in A for each logic family. The energy measured at the knee of each voltage curve is plotted in B. (7)



Ground line noise immunity. Voltage plot of noise threshold vs. pulse width for noise injected into the ground terminal is shown in A. The worst-case conditions of ground line noise energy for each given family is graphed in B. (7)



Some Typical IC Costs - LLL Stock

TTL Chips :

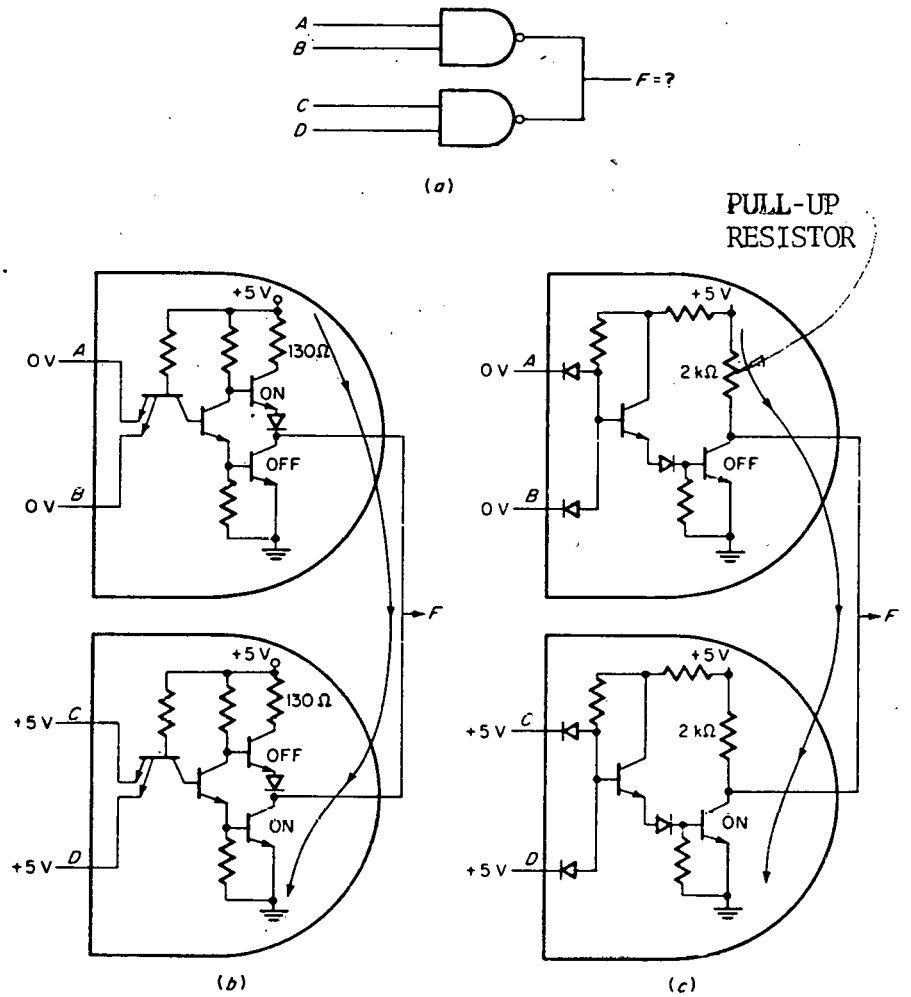
IC Description	Approx. Cost
Quad 2-input NAND	$.17
Quad 2-input NOR	.22
Quad 2-input AND	.25
Quad 2-input OR	.27
Quad 2-input XOR	.44
Dual j-K Master/Slave F-F	.49
4-Bit Shift Register	.82
Decade counter	.92
Full adder	2.33

Non-TTL Chips :

DTL Quad 2-input NAND	.45
RTL Dual 2-input NOR	1.20

3.5 DOT-AND/ORing[3]

Since they are active devices, the outputs of gates are not normally connected together. In the case of the TTL family, tying the outputs together would result in a low impedance path to ground whenever one output is low and the other gate is trying to reach logic one:

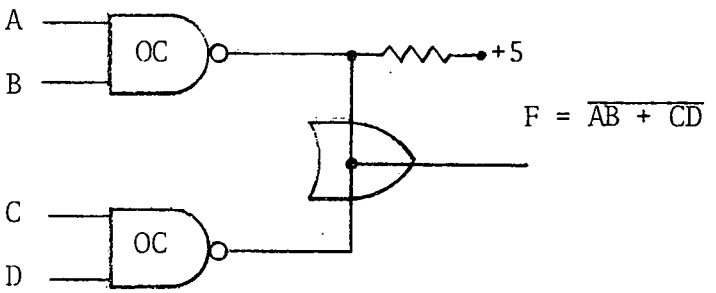


Effect of tying gate outputs together. (*a*) Tying outputs of two gates together; (*b*) not recommended for TTL gates; (*c*) fine for DTL gates. [3]

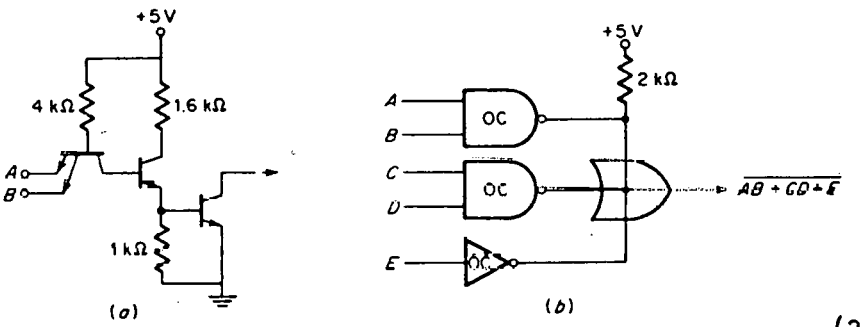
For a DTL gate, however, the same situation results in a 2 k-ohm impedance to ground, and no harm is done, although the output stays low. Since the output can come high only when the outputs of both gates are attempting to come high, the connected gates perform the function:

$$F = \overline{AB + CD}$$

and are represented symbolically as



This is referred to as the DOT-AND/OR function. When gates are used in this fashion, they are modified by leaving out the internal 2 k-ohm pull-up resistor of each gate. A single external resistor serves the entire group. Such gates are said to be in an open collector configuration, as indicated by the label, "OC". Specially made open collector TTL gates for use in DOT-AND/ORing are also available:

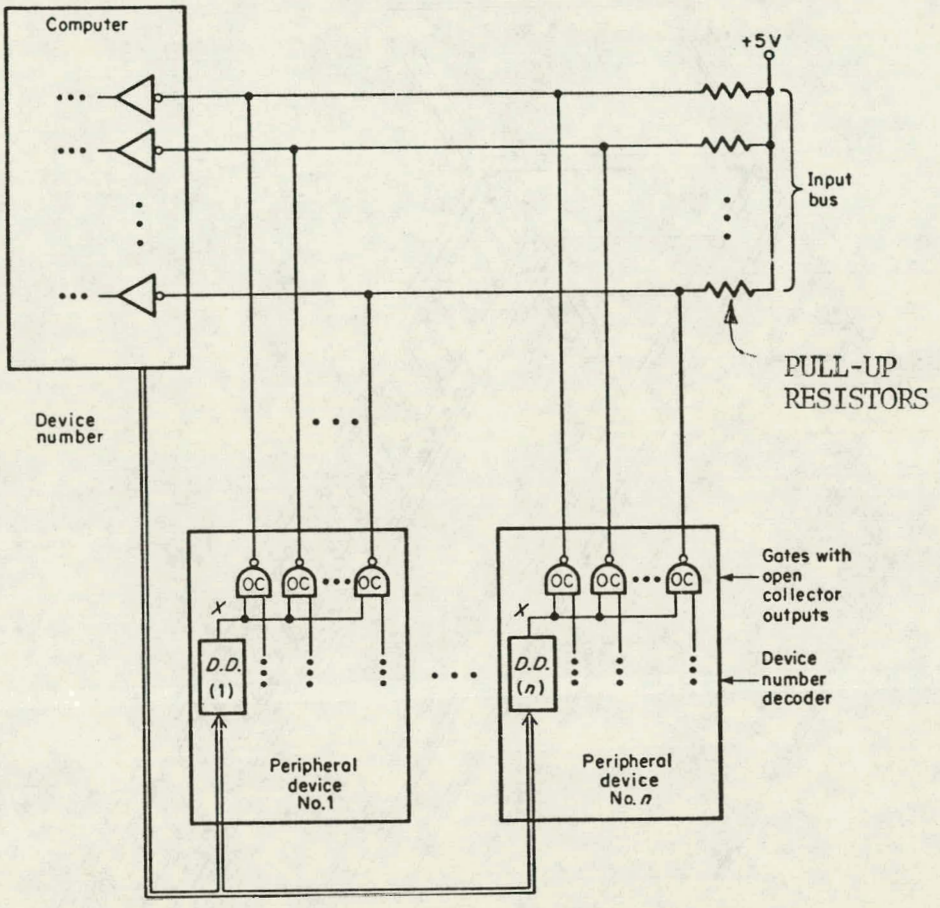


(3)

Modified TTL gate with open-collector output. (*a*) Its structure; (*b*) its use.

The DOT-AND/ORing technique is almost universally used in small computer systems for communication between peripheral devices and the central processing unit (CPU) over a data bus. When a computer wishes to get data from a particular device, it sends out the same device number and control command to every peripheral. Only the peripheral corresponding to that device code responds by setting X to logic one. The signal X is NANDed with each data bit to be input by that device, and if the data bit is high, the corresponding input bus line is pulled low. Because only one peripheral device at a time ever responds, the output data lines of each device can

be DOT-AND/ORed onto a common data bus, thus eliminating the need for a separate set of input lines for each peripheral.



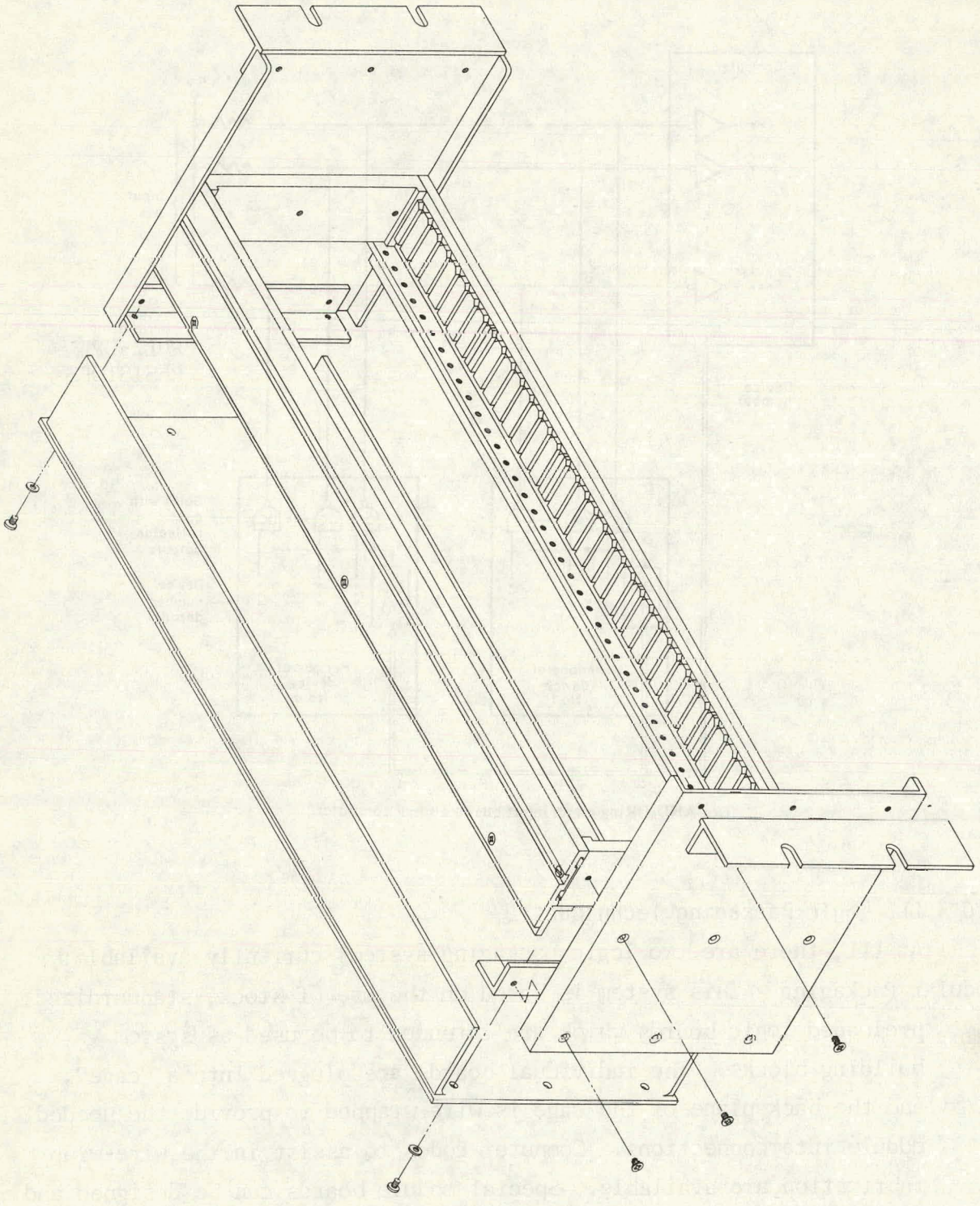
Dot-AND/ORing on an input bus to a small computer.

3.6 LLL Logic Packaging Techniques

At LLL, there are two logic packaging systems currently available:

Modular Packaging - This system is based on the use of stock, standardized, preloaded logic boards which are intended to be used as system building blocks. The individual boards are plugged into a "cage", and the back plane of the cage is wire-wrapped to provide the needed module interconnections. Computer codes to assist in the wire-wrap fabrication are available. Special module boards can be designed and analog components are easily accommodated. The basic vendor for this system is Control Logic Corp.

40-CONNECTOR CAGE ASSEMBLY CGB-540



Modular Packaging Card Cage

EIGHT 4-INPUT NAND GATES CNG-154T
LRL STOCK NO.: 5975-56518
DECAL NO.: 5975-56567

Description

The CNG-154T NAND gate card supplies eight 4-input gates for control and logical gating. They operate from dc to 10 Hz with pulse or level inputs.

The NAND gate is equivalent to a diode AND network followed by an inverting amplifier. Logically this gate performs the AND-NOT function with positive true inputs (+3 V = logic 1) and the OR-NOT function with negative true inputs (0 V = logic 1).

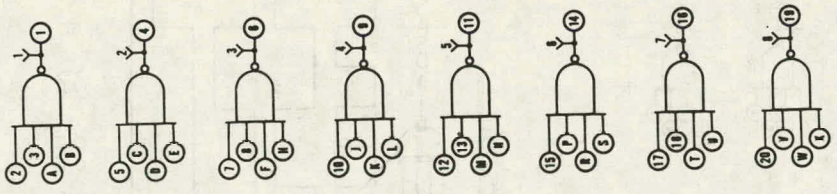
Specifications

Frequency	dc to 10 MHz
Output Drive	10 loads
Input Loading	1 load
Delay (single gate)	45 ns maximum
(gate pair)	75 ns maximum
Power Requirements	30 mA maximum @ +5 V

IC Specifications

EE Standards Manual, Section 11, "Microcircuits"
SN7420N See LED68-902113

Logic Diagram

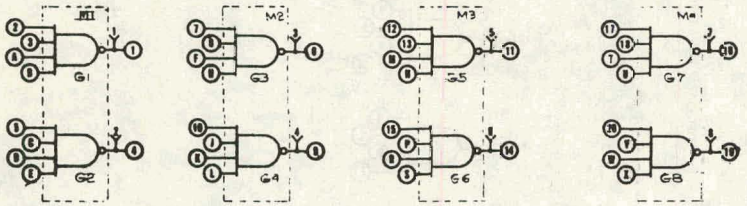


SIZE 1X

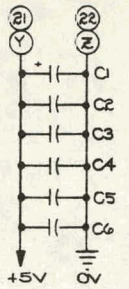
RL-3741-3

EIGHT 4-INPUT NAND GATES CNG-154T
(Continued)

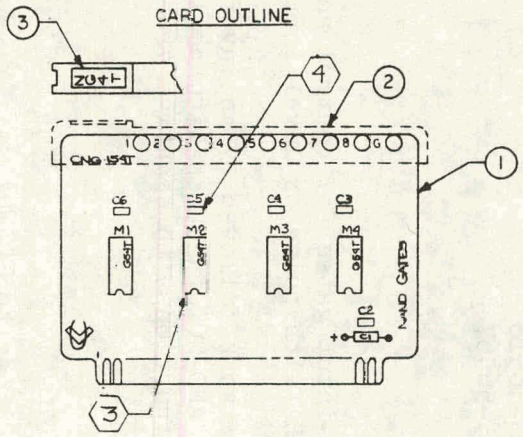
CIRCUIT DIAGRAM



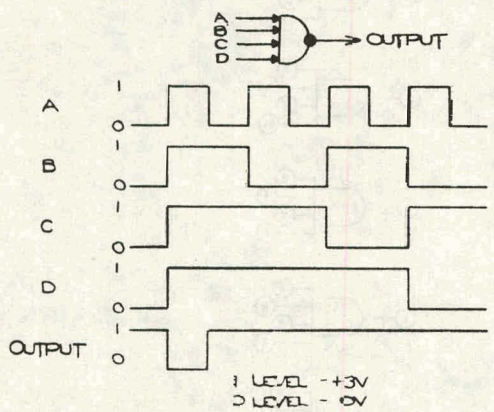
VOLTAGE CONNECTIONS



CARD OUTLINE



APPLICATION



G54T = SN7420N

PIN CONNECTIONS					
1	OUTPUT G1	A	INPUT G1
2	INPUT G1	B	INPUT G1
3	INPUT G1	C	INPUT G2
4	OUTPUT G2	D	INPUT G2
5	INPUT G2	E	INPUT G2
6	OUTPUT G3	F	INPUT G3
7	INPUT G3	H	INPUT G3
8	INPUT G3	J	INPUT G4
9	OUTPUT G4	K	INPUT G4
10	INPUT G4	L	INPUT G5
11	OUTPUT G5	M	INPUT G5
12	INPUT G5	N	INPUT G5
13	INPUT G5	P	INPUT 36
14	OUTPUT G6	R	INPUT G6
15	INPUT G6	S	INPUT G6
16	OUTPUT G7	T	INPUT G7
17	INPUT G7	U	INPUT G7
18	INPUT G7	V	INPUT G8
19	OUTPUT G8	W	INPUT G8
20	INPUT G8	X	INPUT G8
21	+5V	Y	+5V
22	0V	Z	0V

TEST TERMINALS			
1	OUTPUT G1	5	OUTPUT G5
2	OUTPUT G2	6	OUTPUT G6
3	OUTPUT G3	7	OUTPUT G7
4	OUTPUT G4	8	OUTPUT G8

KEY POSITIONS
8-11-20

REQ'D	ITEM	PART NO.	CIRCUIT SYM	DESCRIPTION	MAT'L SPEC.
1	6	112-233	C1	CAPACITOR, 1μF ±10% 35V	C.L
4	5	112-037	M1-M4	IC MODULE, G54T	C.L
5	4	MC51C103EM	C2-C5	CAPACITOR, .01μF ± 20% 100V	RETO YDX
1	3	179-052-01		LABLE, HANDLE NG4T	C.L
1	2	183-096		KIT, HANDLE, MICRO CARD	C.L
1	1	179-308 M		CARD, PRINTED WIRING, CNG-154T	C.L
-1				LIST OF MATERIAL	

DO NOT SCALE DRAWING
REMOVE ALL BURRS AND SHARP EDGES

CONTROL LOGIC, INC.
NATICK, MASSACHUSETTS

UNLESS OTHERWISE SPECIFIED: DIMENSIONS ARE IN INCHES
TOLERANCES ON FRACTIONS DECIMALS ANGLES
± ½ .XX ± .030 ±1°
.XXX ± .010

DRAWN Wayne Kelly DATE 7-3-68
CHECKED DATE 7-10-68
APPROVED DATE
APPROVED DATE

TITLE
EIGHT 4 INPUT NAND GATES
CNG-154T

SIZE C | 179-308 | REV. F
SCALE | ACCOUNT NO. | CODE | SHEET 1 OF 1

1. TEST TERMINAL SYMBOL
2. PIN CONNECTION SYMBOL

NOTES:

(4) CAPACITORS C2-C6 (ITEM 4) TO BE INSERTED PER CONTROL LOGIC, INC. DRAWING NO. 10000-52

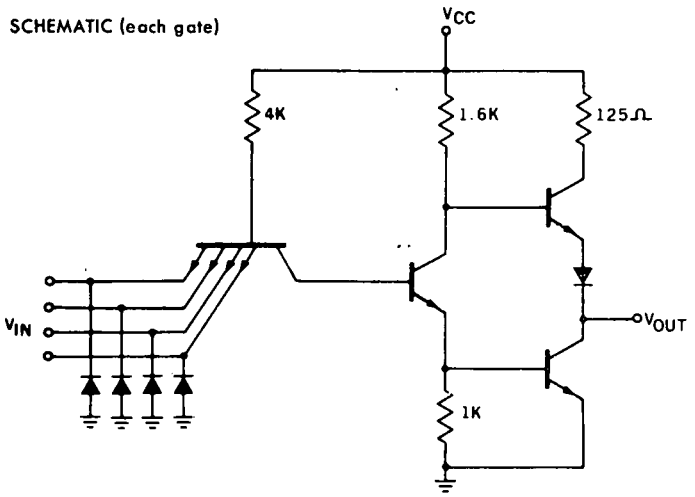
(3) I.C. MODULES (ITEM 5) TO BE INSERTED PER CONTROL LOGIC, INC. DRAWING NO. 10000-51

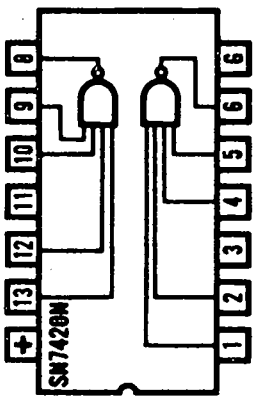
SN7420N

DUAL 4-INPUT NAND GATE

LRL PRIME TYPE: SN7420N
LRL STOCK NO.: 5961-53222
DECAL NO.: 7510-58043

SCHEMATIC (each gate)





RECOMMENDED OPERATING CONDITIONS	MIN.	NOM.	MAX.	UNIT
Supply Voltage (V_{CC}):	4.75	5.0	5.25	V
Operating Temperature Range:	0	25	+70	°C
Fan-Out from each output (N):	1 to 10			

ELECTRICAL CHARACTERISTICS: (over operating temperature range unless otherwise noted)

Characteristic	Symbol	Test Conditions						Limits				Notes
		Test Fig.	Temp.	V_{CC}	Driven Input	Other Input	Output	Min.	Typ.	Max.	Units	
"1" Input Voltage	$V_{in(1)}$	1A		MIN				2.0			V	
"0" Input Voltage	$V_{in(0)}$	1B		MIN						0.8	V	
"1" Output Voltage	$V_{out(1)}$	1B		MIN	0.8V	V_{CC}	−400μA	2.4	3.3		V	1
"0" Output Voltage	$V_{out(0)}$	1A		MIN	2.0V	2.0V	16mA		0.22	0.4	V	1
"0" Input Current	$I_{in(0)}$	1C		MAX	0.4V	4.5V				−1.6	mA	2
"1" Input Current	$I_{in(1)}$	1D		MAX	2.4V	0V				40	μA	2
				MAX	5.5V	0V				1	mA	
Output Short Circuit Current	I_{OS}	1E		MAX	0V	0V	0V	−20		−55	mA	3
"0" Level Supply Current	$I_{CC(0)}$	1F	NOM	NOM	5.0V	5.0V			3	4.8	mA	1, 4
"1" Level Supply Current	$I_{CC(1)}$	1G	NOM	NOM	0V	0V			1	1.6	mA	1, 4
Input Clamp Voltage	V_{Clamp}	1J	NOM	MIN	−20mA				1	1.5	V	1, 2

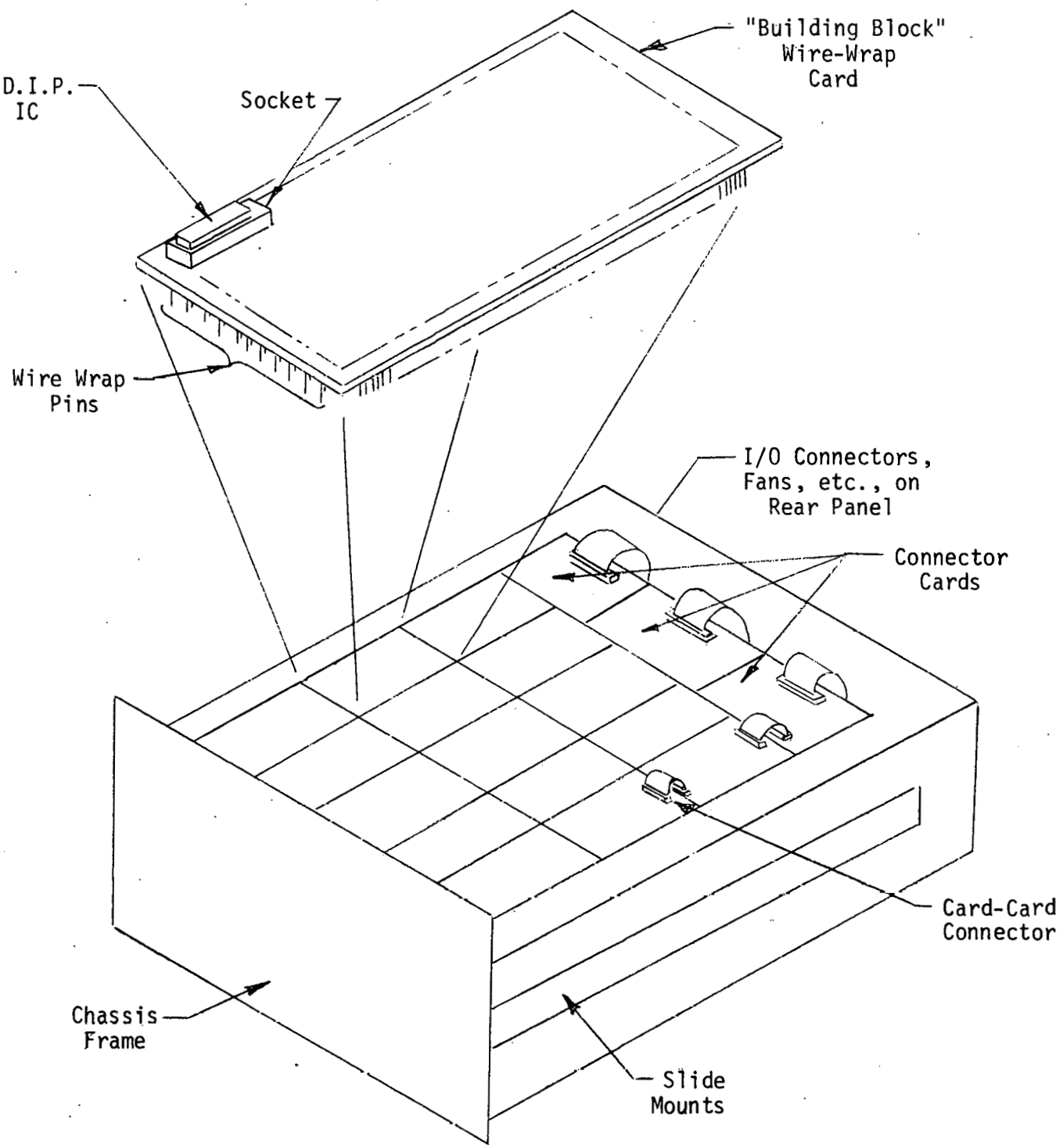
SWITCHING CHARACTERISTICS: V_{CC} = 5.0V, T_A = 25°C, N = 10

Characteristic	Symbol	Test Fig.	Test Conditions	Limits			Units	Notes
				Min.	Typ.	Max.		
Turn-On Delay Time	t_{pd0}	13	C_L = 15 pF, R_L = 400Ω	3	8	12	ns	
Turn-Off Delay Time	t_{pd1}	13	C_L = 15 pF, R_L = 400Ω	3	12	20	ns	
Output Rise Time	t_r	13	C_L = 15 pF, R_L = 400Ω	3	12	18	ns	
Output Fall Time	t_f	13	C_L = 15 pF, R_L = 400Ω	1	5	8	ns	

NOTES:
1. Typical values are at V_{CC} = 5.0V, T_A = 25°C.
2. Each input tested separately.
3. Not more than one output should be shorted at one time.
4. Each gate.

Planar Packaging - This system is just now being introduced at LLL, although it has been in use by computer manufacturers for a few years. In this packaging system various styles of blank, general-purpose DIP (dual in-line packaging) socket cards are combined in a frame to form a logic plane. This plane is then wire-wrapped as a whole. Frames may be stacked up and hinged so that they can be "paged" like a book for easy maintenance. Analog components are less easily accommodated than in the modular packaging system. Computer codes for semi-automatic wire-wrapping the frames are available. The vendor for this system is Standard Logic, Inc.

Specification sheets and drawings illustrating the two packaging systems are contained on the following pages.



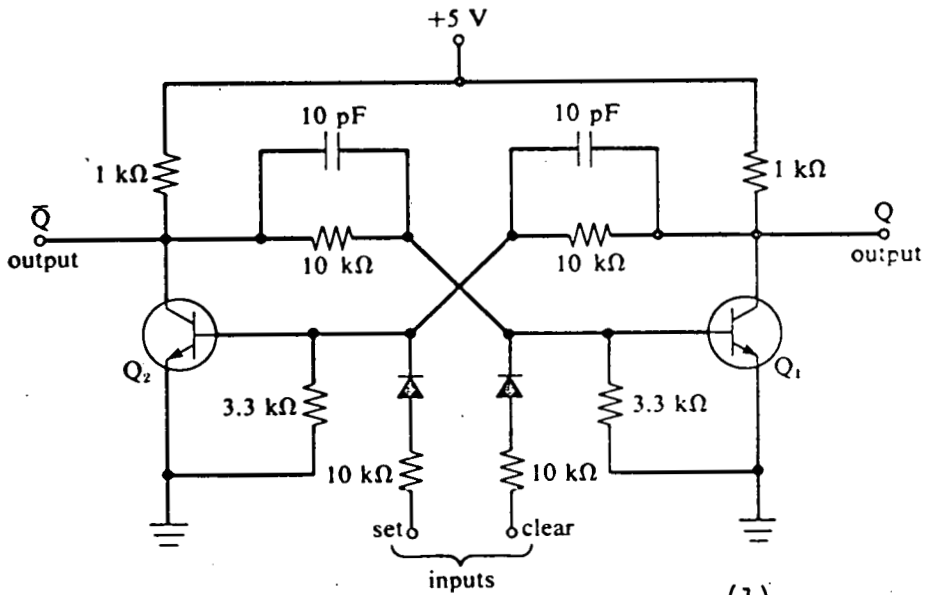
Planar Packaging

4. FLIP-FLOPS and MULTI-VIBRATORS

m 4.1 Flip-Flops[1]

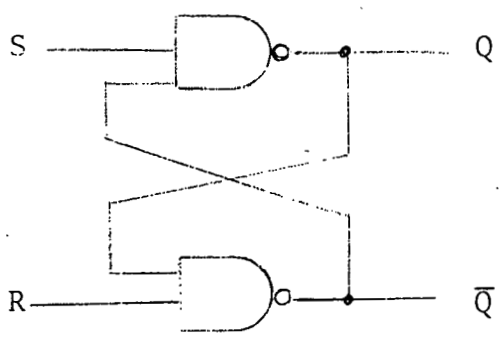
4.1.1 Transistor and logic-gate memories

A bistable device is one which has two stable states. In the case of the transistor bistable memory shown below, these states correspond to outputs of logic "1" or "0". The output state Q is set by pulses on the "set" or "clear" inputs, whichever occurred most recently.



Basic transistor bistable memory.[1]

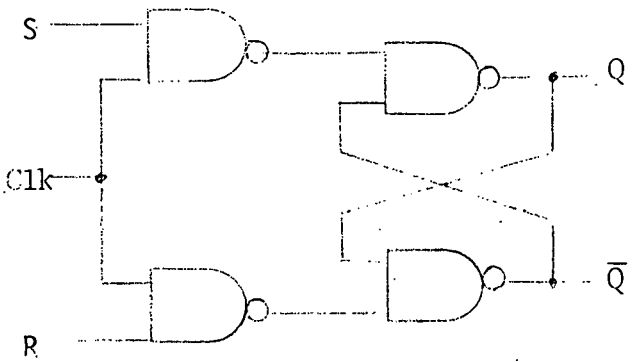
As an alternative to discrete components, a memory circuit can also be constructed using logic gates as illustrated below:



R	S	Q	
0	1	0	
1	0	1	
0	0	1	
1	1	0	(R last 0)
1	1	1	(S last 0)

Logic-Gate Memory ("Latch")

For timing purposes it is generally desirable that a flip-flop respond to its inputs only during a specific interval such as a clock "on" pulse. To accomplish this, a clock signal is NANDED with the R and S inputs to get a clocked R-S flip-flop:

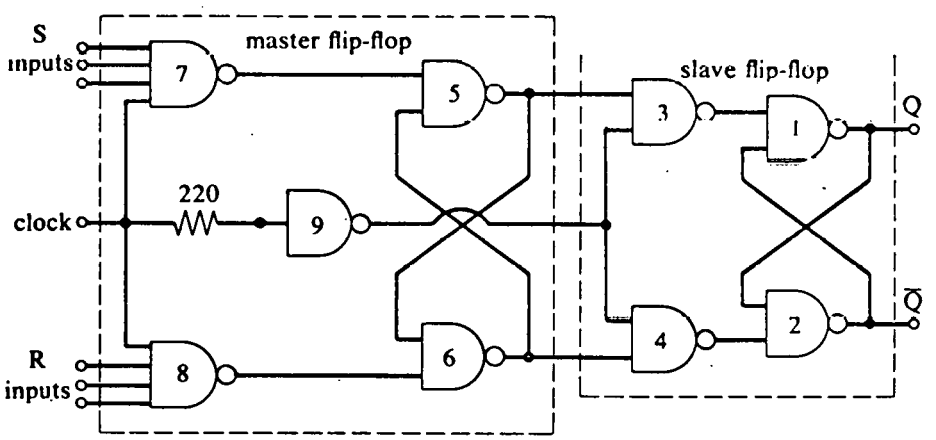


Inputs, t_n		$Q_{t_{n+1}}$
R	S	
0	0	Q_n
0	1	1
1	0	0
1	1	I

R-S Clocked Flip-Flop

4.1.2 R-S Master Slave Flip-flop (1)

The clocked flip-flop responds to its R-S inputs during the clock ON period. In a complex circuit it is often possible for the outputs of a flip-flop to feed back so as to affect its own inputs during the clock "on" time. In order to prevent any ambiguity in the operation of the flip-flop, it is broken up into two stages--a master stage to hold the input data at the start of the clock pulse, and a slave stage which holds the current output during the clock pulse and is updated with the master-stage data at the end of the pulse.



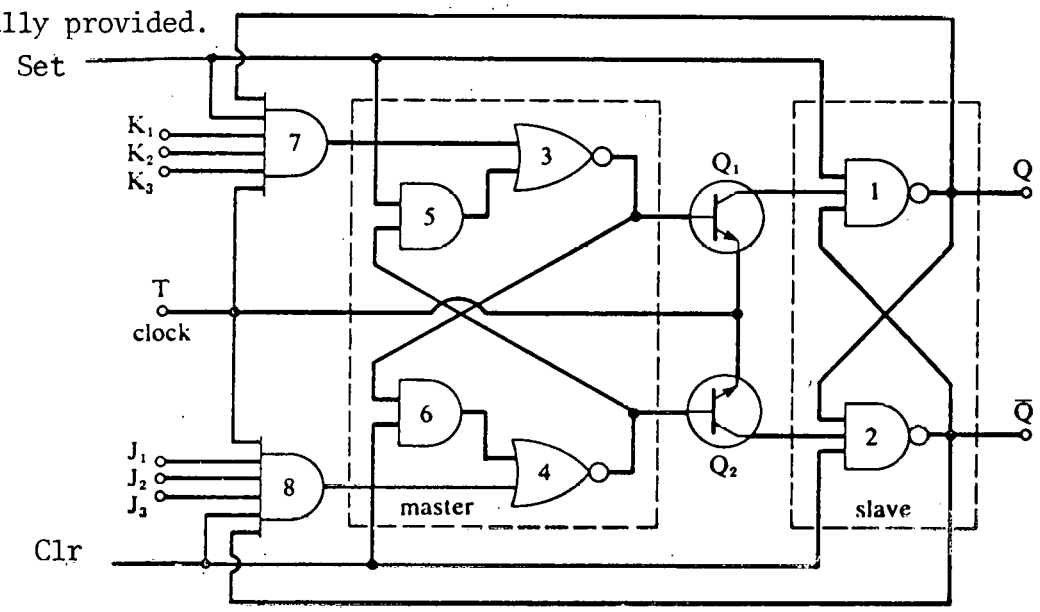
The operating sequence and truth table for the flip-flop is as follows:

Inputs, t_n		Q_{t_n+1}
S	R	
0	0	Q_{t_n}
0	1	0
1	0	1
1	1	Ind.

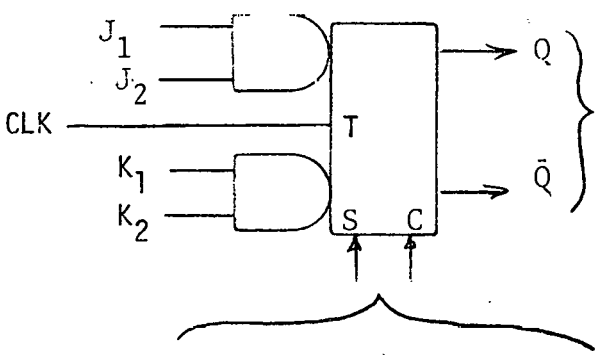
1) Gates 3 and 4 close, isolating slave stage.
2) Gates 7 and 8 open master stage to inputs.
3) Gates 7 and 8 close, isolating master stage from inputs.
4) Gates 3 and 4 open slave stage to master stage.

4.1.3 J-K Master-Slave Flip-flop[1]

The J-K master-slave flip-flop is similar to the R-S flip-flop except that the outputs are connected back to the inputs to provide a toggle action whenever both inputs are 1 simultaneously. Direct set and clear inputs are also generally provided.



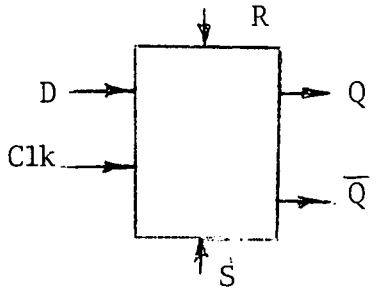
The symbol and truth table for the J-K flip-flop are as follows:



Inputs, t_n		$Q_{t_{n+1}}$
J	K	
0	0	Q_{tn}
0	1	0
1	0	1
1	1	$\overline{Q}_{tn}$

4.1.4 D-type Master Slave Flip-flop[1]

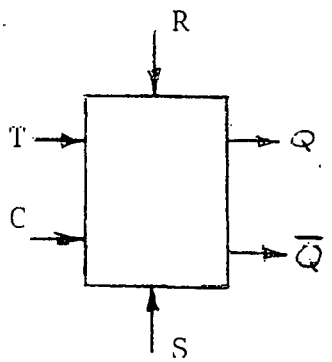
The D-type flip-flop is similar to the R-S except that the single D input is inverted to drive the R input while the D drives the S directly. As a result, whatever data is on the input line at the clock pulse is simply transferred to the output.



D_{t_n}	$Q_{t_{n+1}}$
0	0
1	1

4.1.5 T-type Master Slave[1]

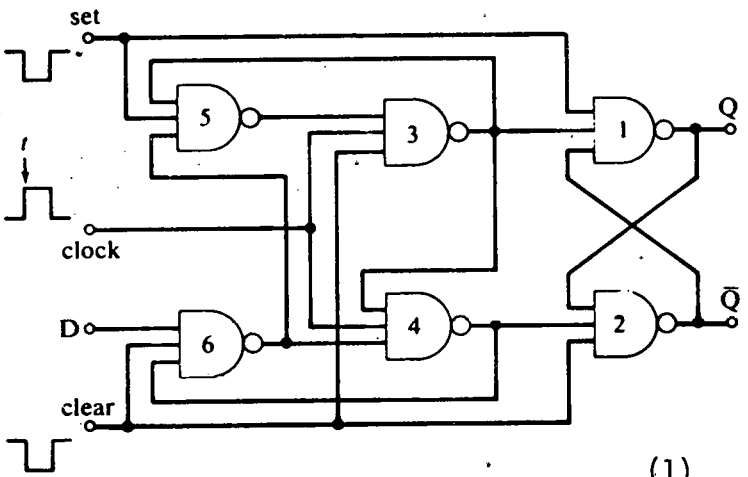
A toggle or T-type flip-flop simply complements its output whenever T is high during a clock pulse.



T_{tn}	$Q_{t_{n+1}}$
0	Q
1	$\overline{Q}$

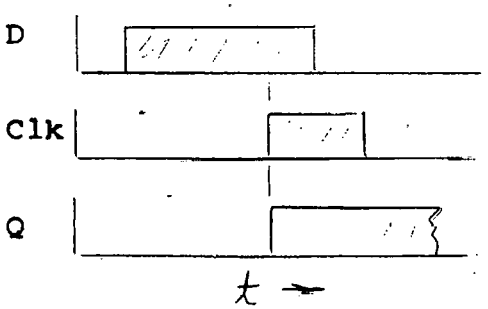
4.1.6 Edge Triggering[1]

A second solution to the flip-flop timing ambiguity problem is the use of edge triggering. In this type of device the input signals are "sampled" at a particular instant of time and then locked out until the next clock pulse. The following D-type edge-triggered flip-flop is typical of these devices:



D-type edge-triggered flip-flop.[1]

This flip-flop has a timing diagram which appears as follows:



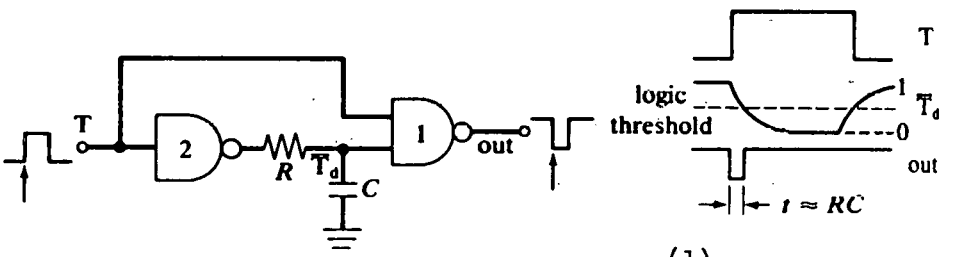
It should be noted that the data D was sampled on the leading edge of the clock pulse and the result transferred to the

outputs immediately (almost). The output is unaffected by the 1-0 transition of D while the clock input is still high.

4.2 Multi-vibrators (MV) [1]

4.2.1 Monostable Multivibrators (one-shot)

A monostable multivibrators is a device which has only one stable state; when forced into another state, it automatically returns to its only stable condition. A monostable MV can be simply constructed from logic gates and an RC network :



RC delay monostable. [1]

This device is generally useful for generating clear or update pulses, clocking, and other applications requiring pulses of short and relatively imprecise duration.

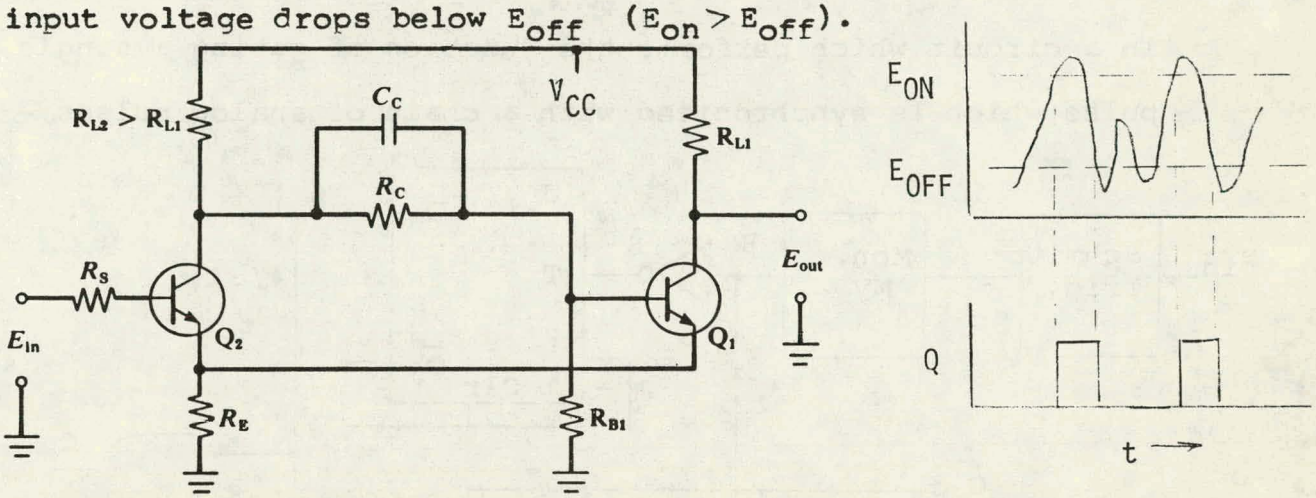
4.2.2 Astable MV

The astable MV has no stable output - it continuously oscillates back and forth between the 0 and 1 logic states at a rate fixed by its internal RC components. It can be used in any application where an oscillator of moderate precision is required.

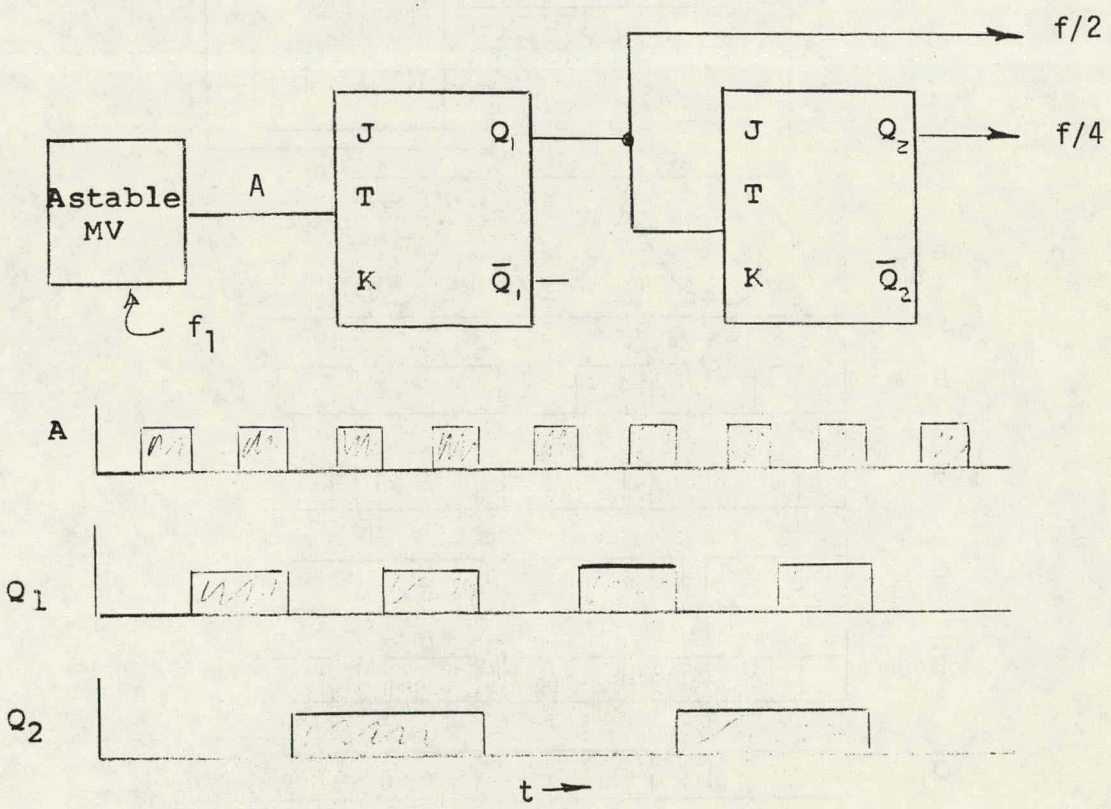
4.2.3 Schmitt Trigger

The Schmitt trigger is a bistable device in which the logic level output switches from 0 to 1 whenever the analog input

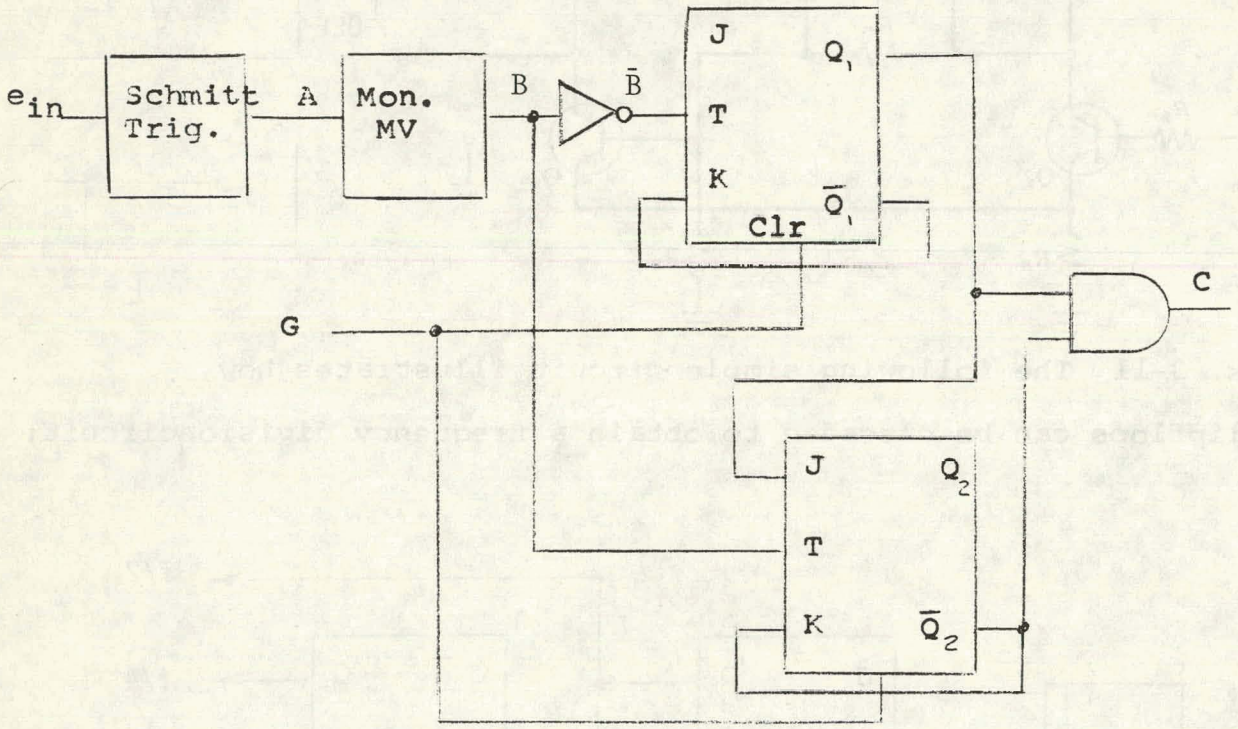
voltage exceeds a level E_{on}, and switches back to 0 when the input voltage drops below E_{off} ($E_{on} > E_{off}$).

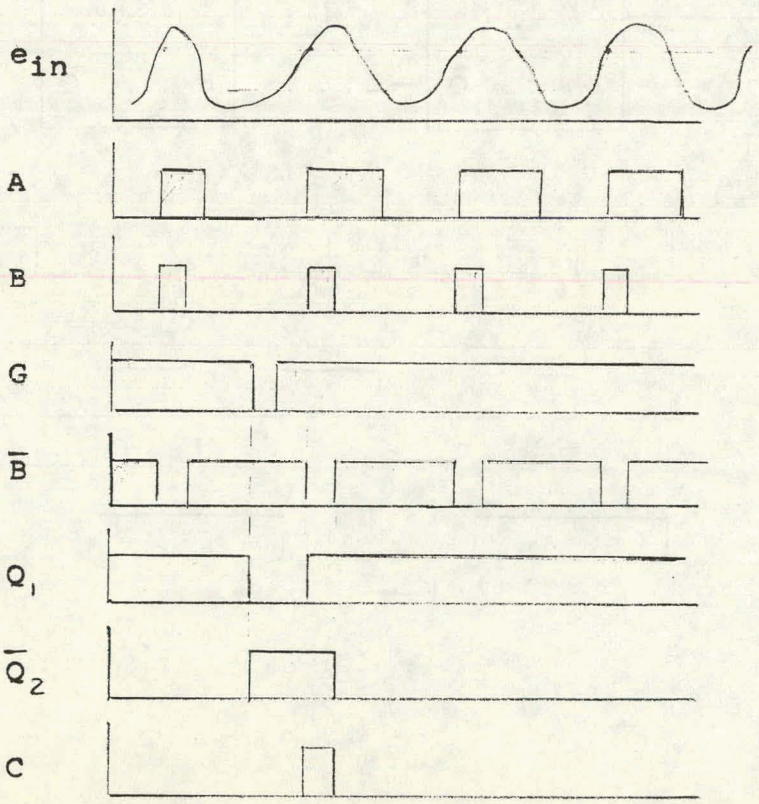


Ex. 3-11 The following simple circuit illustrates how flipflops can be cascaded to obtain a frequency division circuit:



Ex. 4-1. The following circuit illustrates the combination of a Schmitt trigger, monostable MV, J-K flip-flop, and some gates in a circuit which performs the function of gating a single logic pulse which is synchronized with a chain of analog pulses -





5. Sequential Circuitry

5.1 Clock

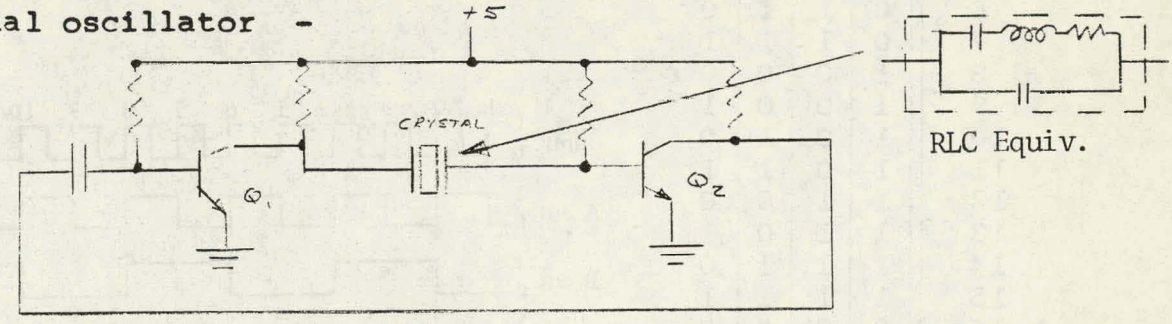
A "clock" is any circuit which serves as a relative time base for a digital system by emitting a continuous stream of pulses. Clocking pulses are characterized by their frequency, duration, and rise time. Maximum frequencies, minimum pulse duration, and required waveshape are specified by each manufacturer for his logic line. For TI 54/74 logic,

Max. frequency - 15 MHz

Duration - 20 ns, min.

Rise time - dc → ?

The clock signal may be derived from an external source such as shaft rotation or AC power line frequency, or from some independent internal source. The two most common examples of the latter are the astable multivibrator (see Sec. 4.2.2) and the crystal oscillator -



Crystal Oscillator

5.2 Counters[1]

Counters come in many varieties. They are generally

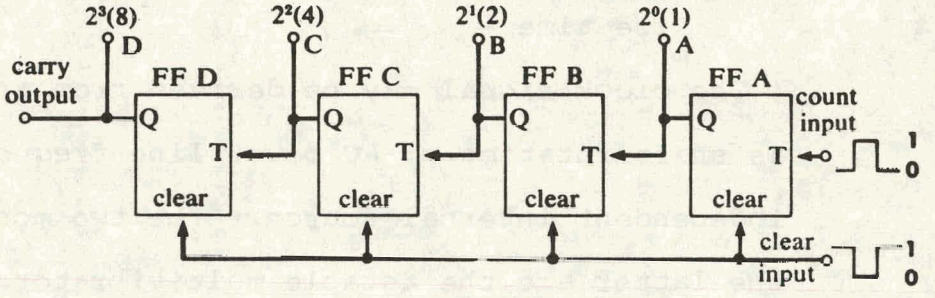
characterized by their (1) output code (binary or BCD), (2) direction of count (up/down), (3) timing (asynchronous or synchronous), and (4) modulus (number of pulses to reach full scale).

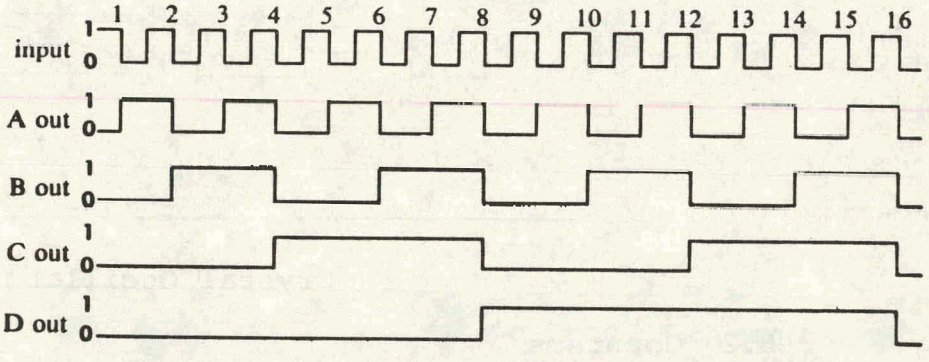
5.2.1 Binary / BCD [1]

A counter circuit counts pulses and accumulates the result in either a binary or BCD number code. The basic binary counter consists of a chain of flip-flops in which the output of each successive stage toggles the next. Such a group of flip-flops is termed a register, and each piece of output binary data is referred to as a bit. Typical binary and BCD counters are as follows :

Binary

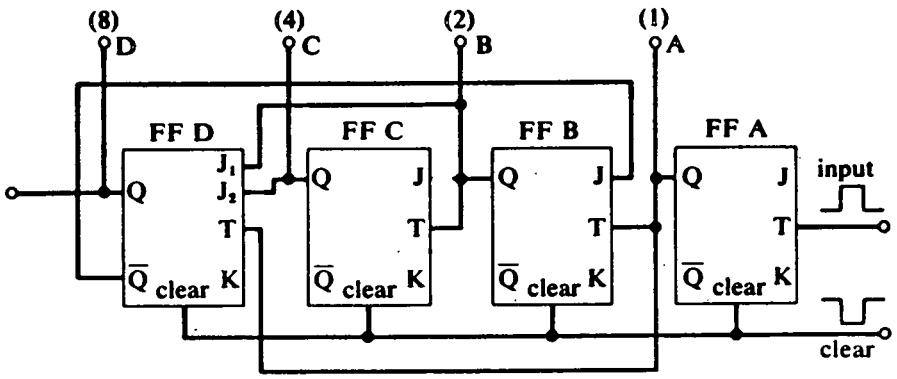
Cnt	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

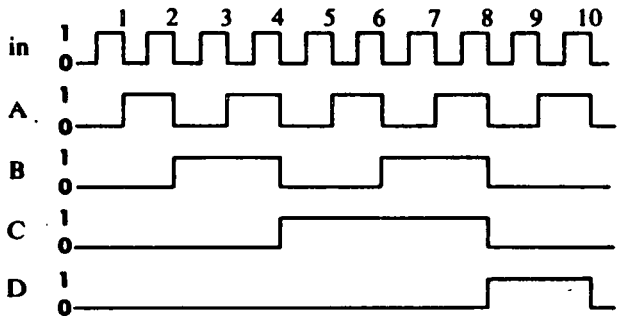




BCD

Cnt	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0
11	0	0	0	1
12	0	0	1	0

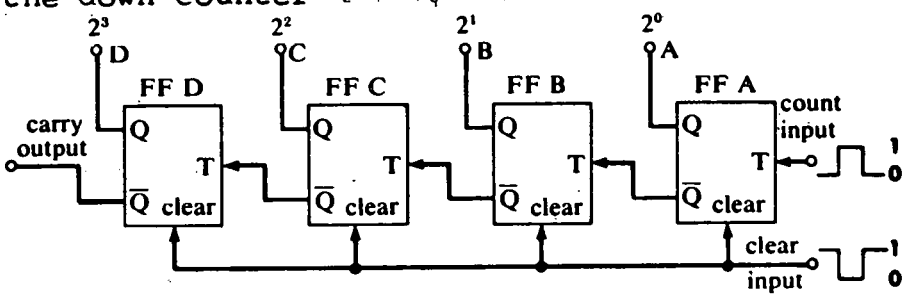


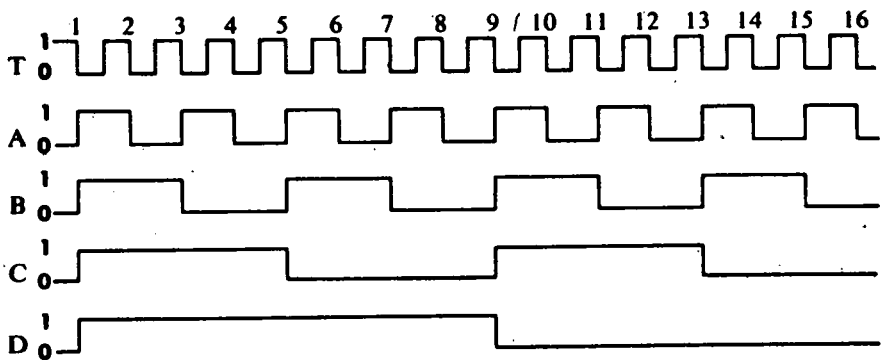


5.2.2 Up/Down (1)

Both of the previous counters were "up" counters. A common variation is the down counter (binary shown here):

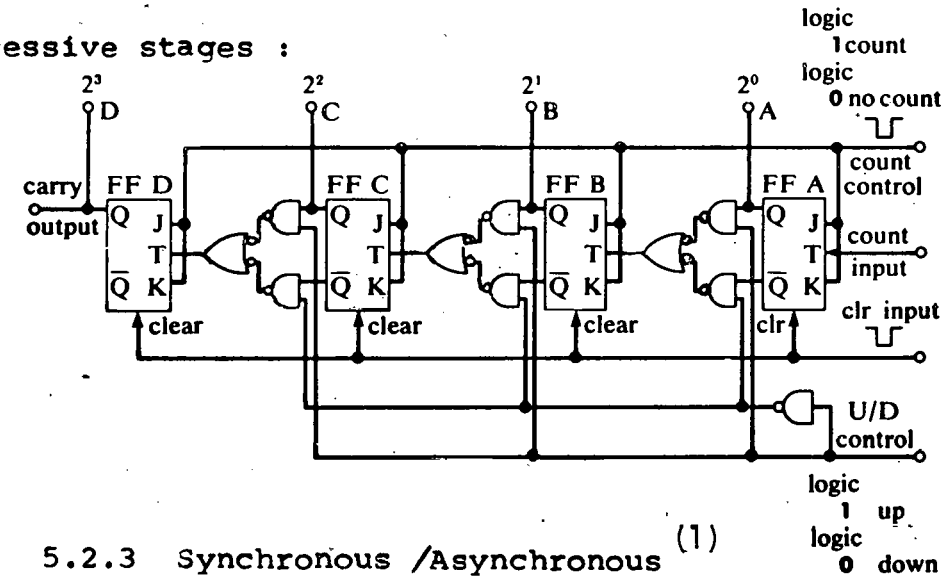
Cnt	D	C	B	A
0	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0





A single binary counter may be made to serve as an "up" or "down" counter by gating either the Q or $\overline{Q}$ outputs between

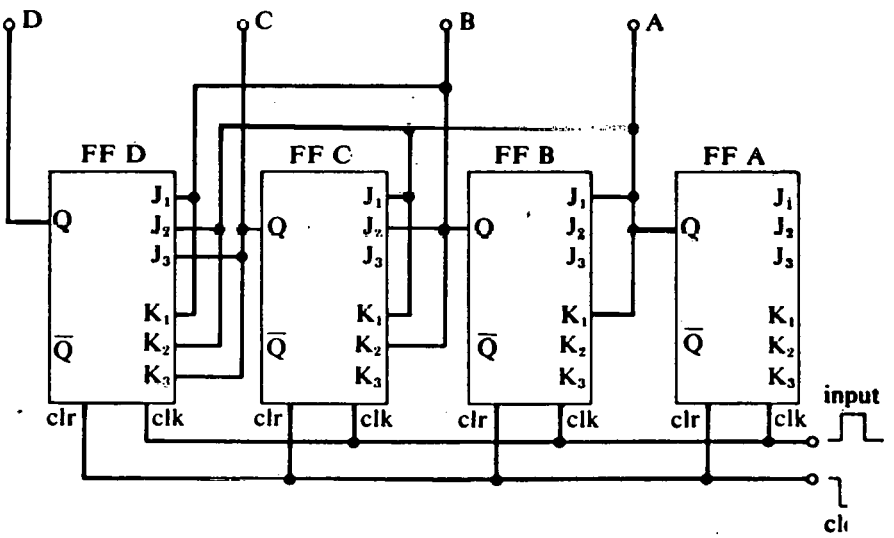
successive stages :



5.2.3 Synchronous /Asynchronous (1)

In all the counters up to this point the counting was asynchronous - i.e., the flip-flops change state for each count at slightly different times as the output changes rippled from one stage to the next (worst case 0111→1000). This can cause count errors at very high data rates due to missed counts, so under these conditions a synchronous counter is required. In such a counter the input conditions to each stage are setup on the J-K inputs before the clock pulse, and all flip-flops respond together at the clock pulse -

Cnt	D	C	B	A
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

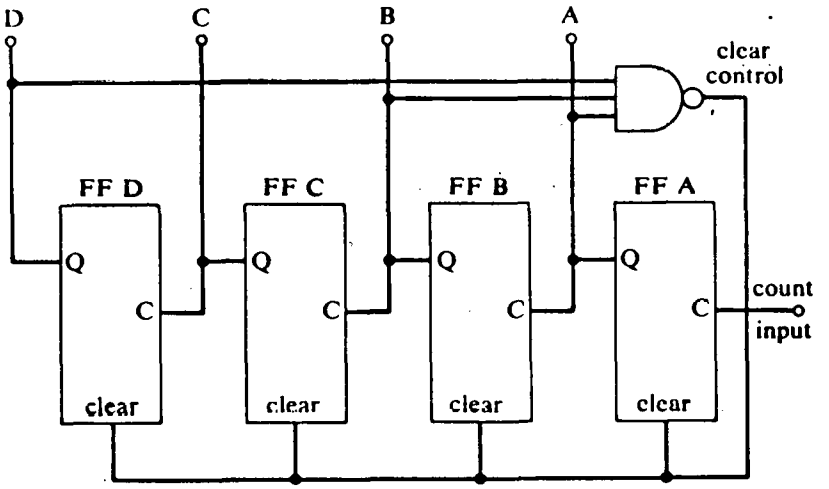


Synchronous binary counter.

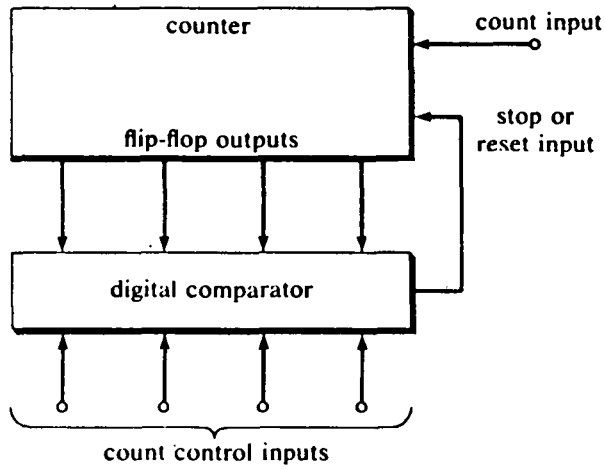
5.2.4 Fixed- and Variable- Modulus Counters[1]

The modulus of a counter is the number of counts to reach full-scale output. A binary counter with n flip-flops has a modulus of 2^n. A counter of modulus N followed by a counter of modulus M results in an overall modulus of NxM.

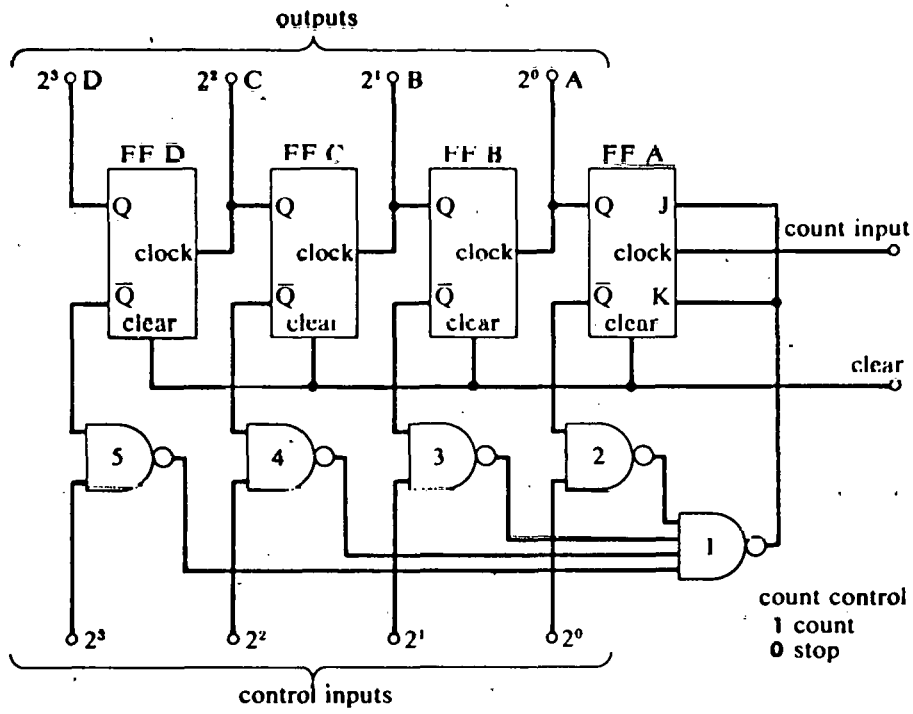
All the counters discussed to date have been fixed-modulus types in which logic gates and the J-K inputs were utilized to control the counting sequence. A counter of virtually any modulus can be obtained by gating the logic one states at the desired maximum count to the direct clear inputs such as in this modulo-11 counter :



Variable modulus counters are generally based on circuit configurations simular to that shown in the block diagram on the following page. A count control from some external source fixes the modulus of the counter by enabling a set



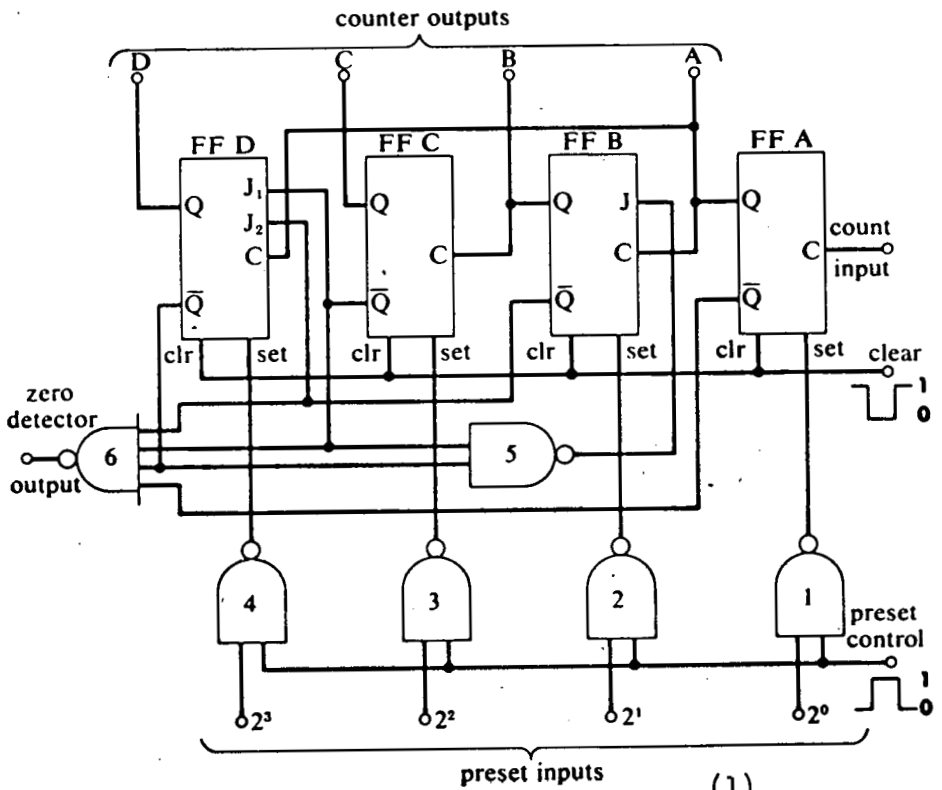
of gates which act as a comparator and detect when the count reaches the preset value. In the counter shown below, the counter is simply stopped when the count is reached. In further sophistications of this technique the comparator circuit output is used to enable the J-K inputs of all stages directly, thereby enabling automatic recycling of the counter.



Preset Binary Up Counter(1)

In preset counters an initial count is set into the flip-flops and the count begins there. A very common application is in an interval timer where the desired period (number of clock

pulses) is initially set into the flip-flops of a down counter. Clock pulses are then counted until all zeros are detected on the counter outputs. A logic zero is then generated, indicating the end of the timed interval.



Preset BCD down-counter. (1)

5.3 Shift Registers (1)

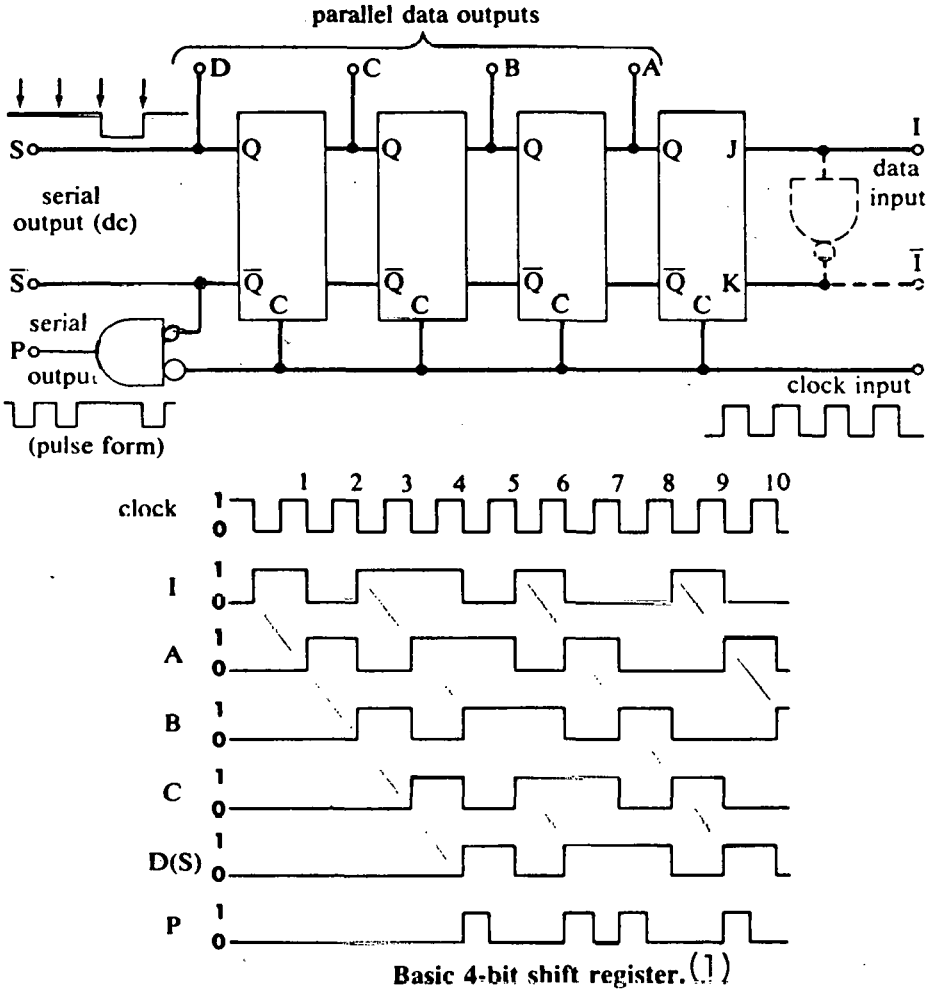
A shift register is a logic device often required for the mathematical or logical manipulation of data. For instance, successive divisions by 2 can be accomplished by simply shifting the binary decimal point to the left one bit for each division. This, of course, is equivalent to shifting the whole binary number to the right one place and discarding the rightmost bit.

$20_{10} =$ | 1 | 0 | 1 | 0 | 0 | $_2$

$10_{10} =$ | 0 | 1 | 0 | 1 | 0 | $_2$

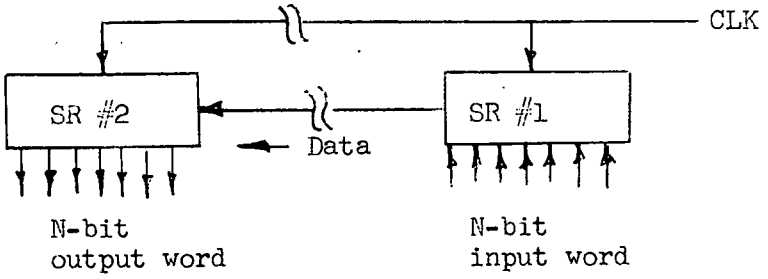
Shift registers also find wide application in parallel/serial/parallel data conversion, data buffers, and delay devices.

The inputs and outputs of a shift register can be virtually any combination of serial (one bit at a time) and/or parallel (all bits at once) data. A basic serial input, serial/parallel output configuration is illustrated by this 4-bit shift register:

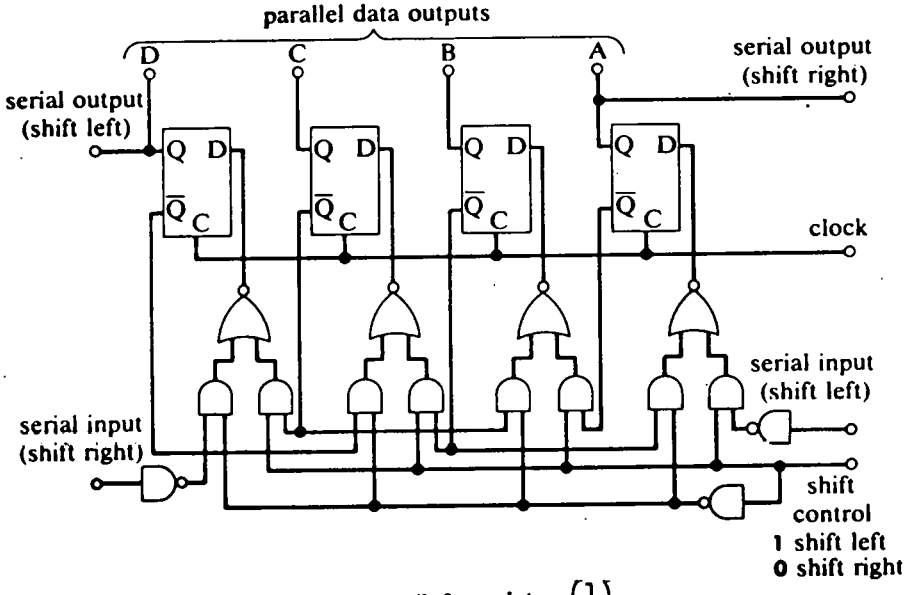


Basic 4-bit shift register.(1)

Two N-bit shift registers are all that is required to transmit an N-bit data word over a 2-wire transmission line :

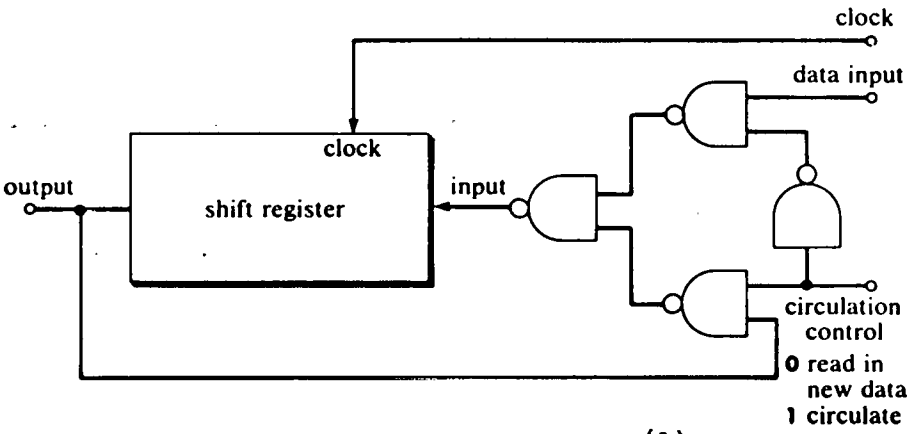


The 4-bit shift register discussed above is a shift-left design. Since left or right shifting is simply a geometric consideration, a combination left/right shift register is obtained by gating the outputs of the preceeding or following flip-flops at any particular stage :



Shift right/left register. (1)

A circulating shift register is simply a shift register which has, optionally, the ability to use as inputs its own outputs or a data input :



Circulating register.(1)

5.4 Readouts

Digital readouts are required in many systems to present the results of calculations or data manipulations to the operator. The principal advantage of digital vs. analog displays is that they require no operator interpretation. The readout consists of two main parts - the alphanumeric display which actually presents the data, and the decoder/driver which interfaces the digital circuit to the display.

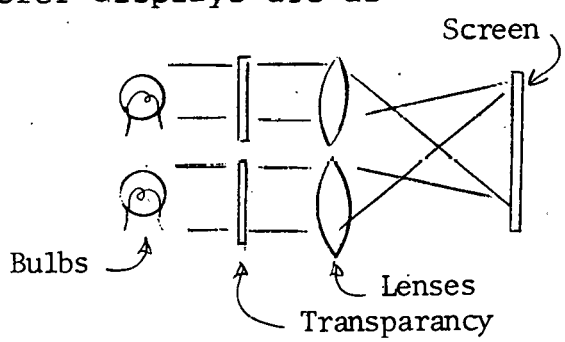
5.4.1 Display

The displays most often used in digital systems fall into three classes :

I. Light transfer displays - In this class of display, light is projected, transmitted, or reflected to form characters. As such, their readability is generally adversely affected by high ambient lighting (LC is exception) The four basic types of light transfer displays are as follows :

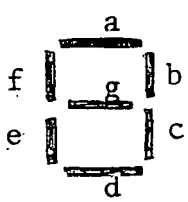
(1) Projection-lighted

- complex displays possible



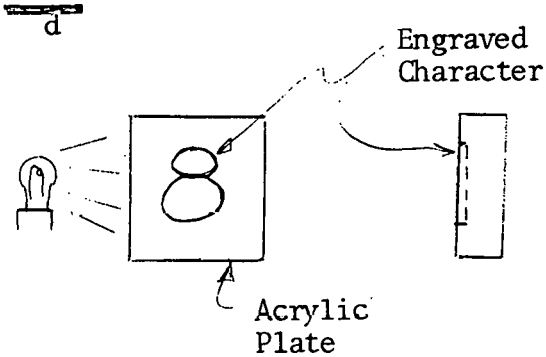
(2) Segmented

- inexpensive

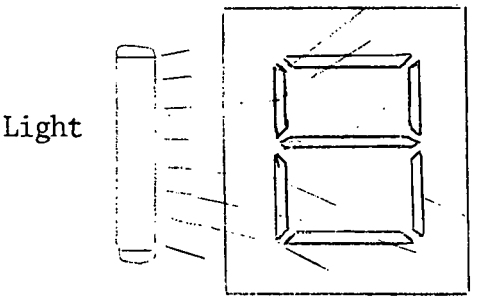


(3) Edge lighted

- esthetic



(4) Liquid crystal (LC)



- low power
- short life
- temperature limited
- slow response

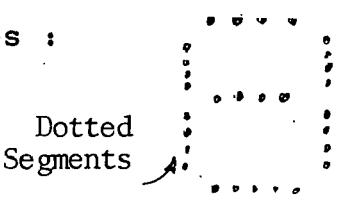
Incadescent bulbs are widely used in the above types. This leads to a relatively short life and lower mechanical ruggedness for light-transfer displays.

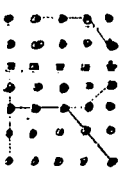
II. Light Emitting - In light emitting displays the characters are generated by luminous emission from preformed characters, character segments, or a dot matrix. These types of displays rely on four phenomenon -

(1) Cold-cathode glow discharge - Luminosity of ionized gas - NIXIE tubes.

(2) Incadescence - Character segments are generated by tungsten filaments which are viewed directly

(3) Light emitting diode (LED) - These devices depend on the light emitting property of certain P-N junctions, primarily GaAsP and GaP. These devices feature long life, low power, high speed, and rugged construction. LED displays are of both the 7-segment and dot matrix types :





Dot Matrix
(5 x 7)

LED displays can be difficult to view over extended periods of time, but newer light-pipe types offer considerable improvement in this area.

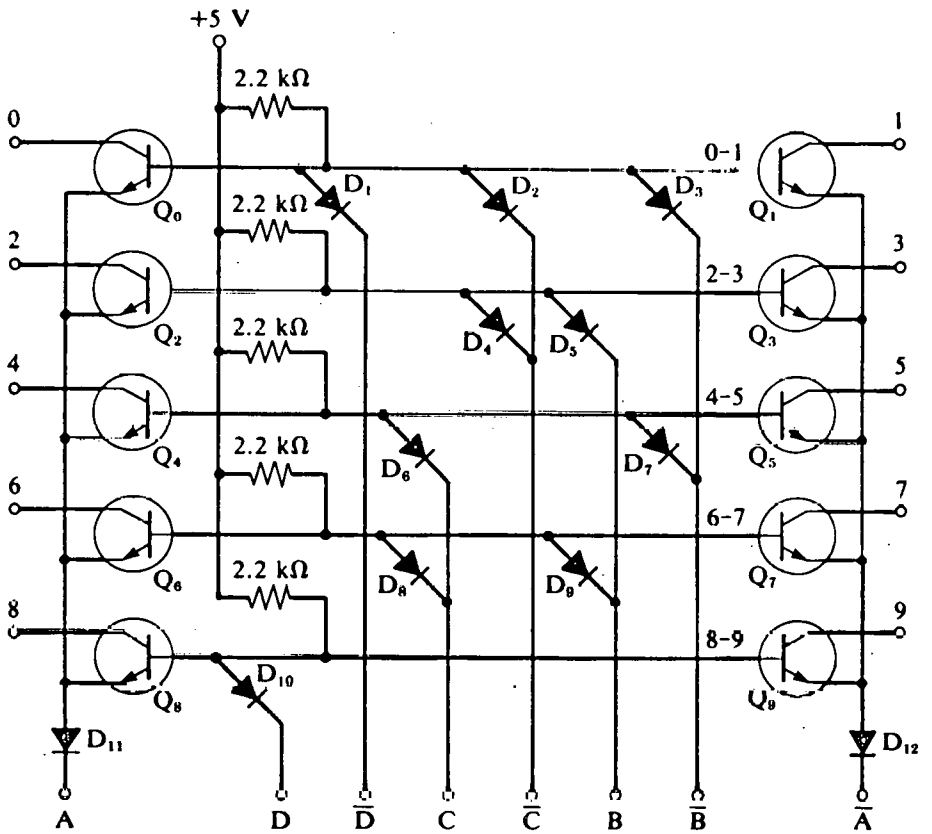
III. Cathode ray tube (CRT) - By far the most powerful, and also the most expensive, type of display is the CRT. Large amounts of information can be displayed in a relatively compact form and at high speeds. The positioning of characters is arbitrary, and alphanumeric data is often combined with a graphic capability to form a very potent man/machine interface. Character generation hardware is often used to increase display rates.

5.4.2 Decoder/Drivers[1]

Most of the displays discussed above require some means of deriving from the digital data the information required to light the proper character segments, dots, full characters, etc. This is referred to as decoding. Since the actual display may require much higher currents or voltages than the logic type (e.g., TTL), an interface is required to transform the digital signals to the desired level. This is referred to as a driver. Manufacturers often combine a decoder and driver into one package which is suitable for their line of displays. A typical decoder, used for driving NIXIE tubes directly from a binary count, is shown below.

Binary-to-Decimal Decoding Table, Natural Code

Decimal No.	Flip-Flop Outputs D	C	B	A	Logic Statement				Minimized Statement			
0	0	0	0	0	$\overline{A}$	$\overline{B}$	$\overline{C}$	$\overline{D}$	$\overline{A}$	$\overline{B}$	$\overline{C}$	$\overline{D}$
1	0	0	0	1	A	$\overline{B}$	$\overline{C}$	$\overline{D}$	A	$\overline{B}$	$\overline{C}$	$\overline{D}$
2	0	0	1	0	$\overline{A}$	B	$\overline{C}$	$\overline{D}$	$\overline{A}$	B	$\overline{C}$	
3	0	0	1	1	A	B	C	$\overline{D}$	A	B	$\overline{C}$	
4	0	1	0	0	$\overline{A}$	$\overline{B}$	C	$\overline{D}$	$\overline{A}$	$\overline{B}$	C	
5	0	1	0	1	A	$\overline{B}$	C	$\overline{D}$	A	$\overline{B}$	C	
6	0	1	1	0	$\overline{A}$	B	C	$\overline{D}$	$\overline{A}$	B	C	
7	0	1	1	1	A	B	C	$\overline{D}$	A	B	C	
8	1	0	0	0	$\overline{A}$	$\overline{B}$	$\overline{C}$	D	$\overline{A}$			D
9	1	0	0	1	A	$\overline{B}$	$\overline{C}$	D	A			D



Biquinary decoder-driver. (1)

6. HYBRID DEVICES

For the purpose of this discussion, the term hybrid refers to the transition between some analog quantity (motion, pressure, and voltage) and a digital signal.

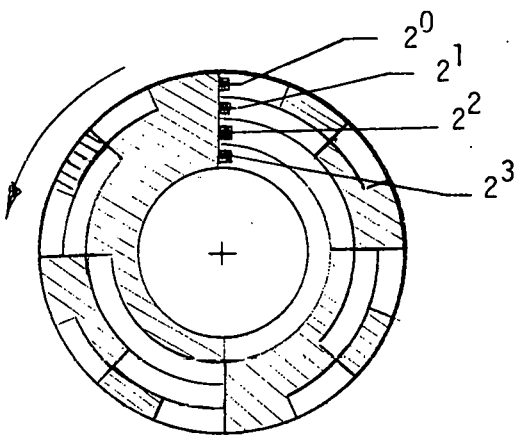
6.1 Direct Encoders[1, 3]

Direct encoders are devices which combine the functions of transduction and analog/digital conversion. Such devices are commonly available for displacement measurements, and less commonly for pressure and temperature. Since most of the schemes for pressure and temperature sensing rely ultimately on displacement measurements, we will concentrate on that type.

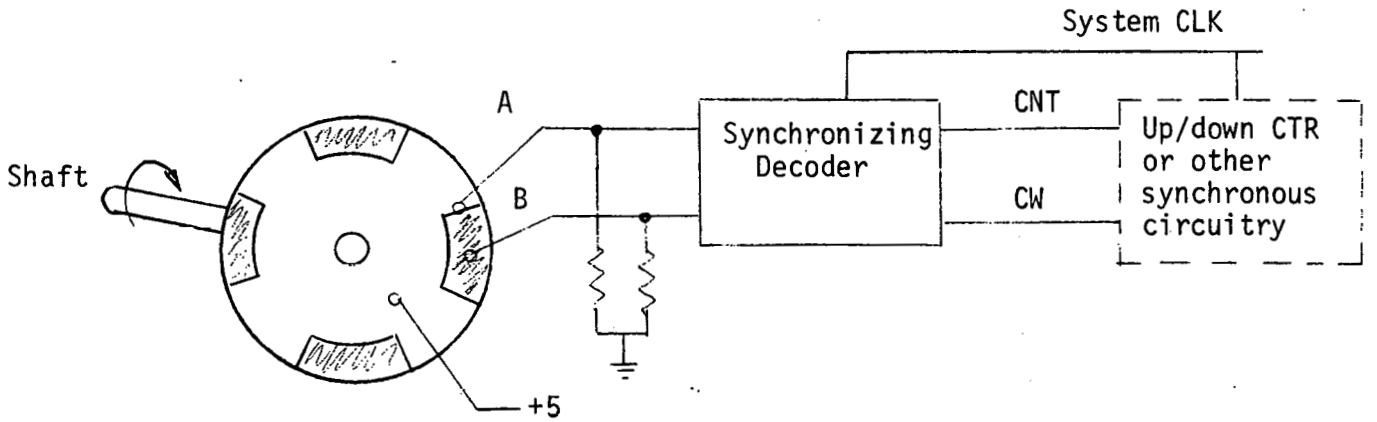
Incremental encoders - In a simple incremental encoder, pulses are generated at precise increments of mechanical translation or rotation. The signal by itself does not indicate position. Position must be determined externally by counting the number of pulses from some mechanical zero in the encoder.

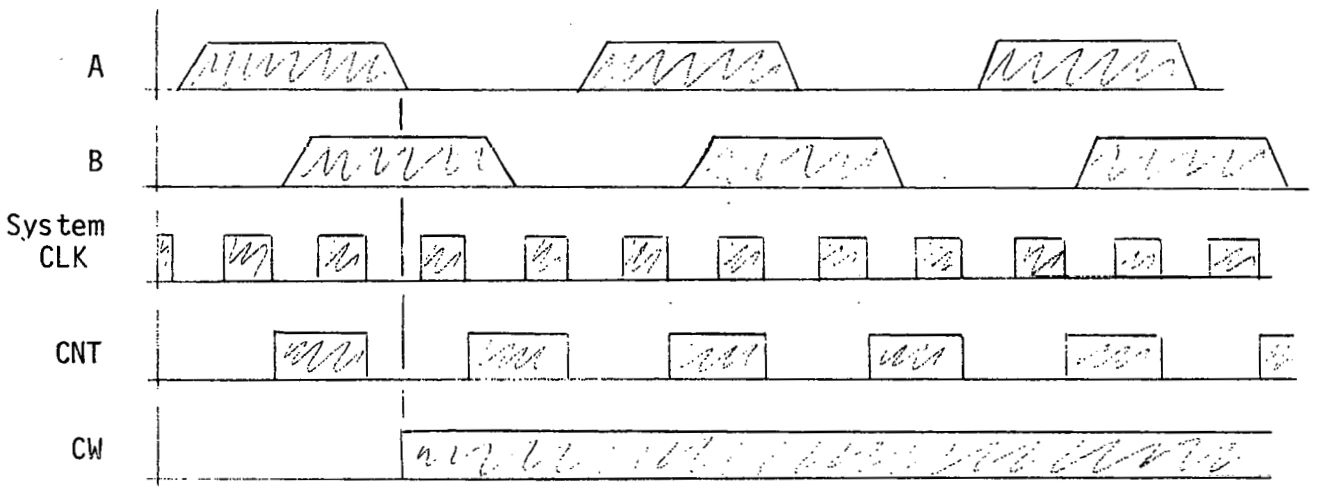
In a bidirectional incremental encoder a second stream of output pulses is available which is phased 90° with respect to the first. By determining which signal leads the other, the direction of motion can be determined. When an encoder is serving as an input to a sequential digital circuit, it is generally desirable to synchronize the count pulse from the encoder with the system clock. This is illustrated by the circuit on the following page.

Absolute encoders - Absolute encoders use a disc or scale with many tracks to indicate the absolute angle of rotation or physical displacement. With N tracks, the resolution can be as high as one part in 2^N of full

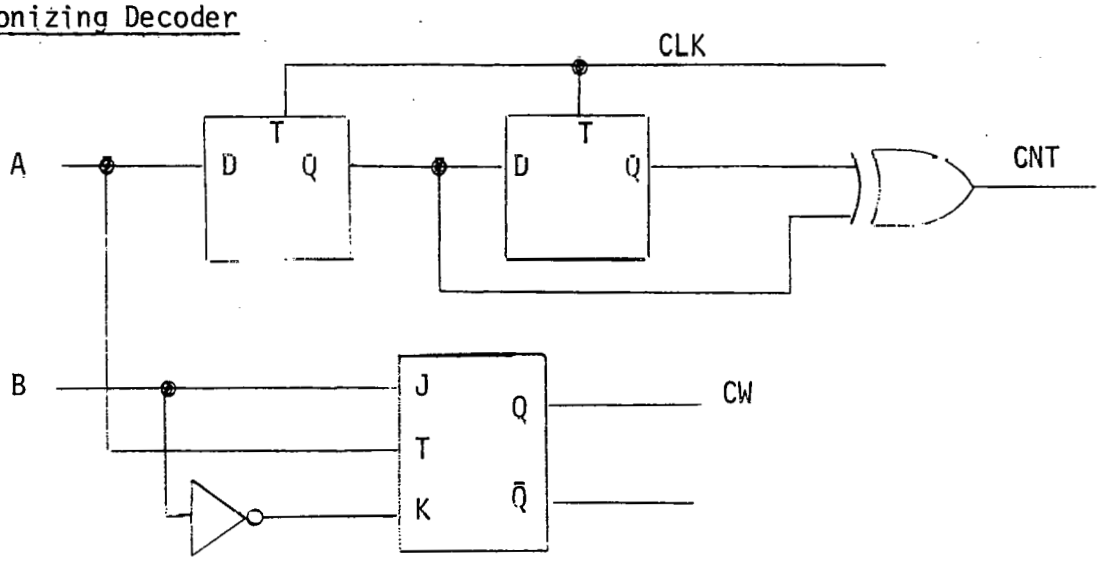


Bidirectional Incremental Encoder





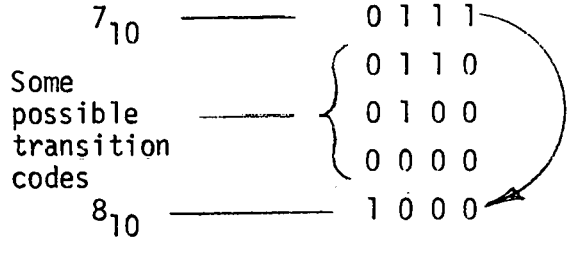
Synchronizing Decoder



scale (no better than an incremental encoder with 2^N pulses per revolution). The output is in the form of N data lines which contain a count in some suitable code, e.g., binary, BCD, gray, etc. The binary or BCD codes represent a potential problem due to the fact that certain transitions, say 7_{10} to

Decimal	Binary	Gray
0	0	0
1	01	01
2	10	11
3	11	10
4	100	110
5	101	111
6	110	101
7	111	100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000
16	10000	11000

8_{10}, result in changes to every binary digit. If the system were to be sampled during this transition, the results may be totally in error, e.g.,



The gray scale eliminates this problem because any transition between two adjacent numbers differs by only one bit. Thus any ambiguity caused by sampling during an encoder transition is limited to a maximum of one count. The gray code, however, is more difficult to decode and is therefore not as popular as binary or BCD.

The advantages of the absolute encoder are that it is less sensitive to noise than the incremental type, and does not "lose its place" during a power failure. However these advantages are offset by the increased cost of the encoder and associated electronics (about 2 x $).

Various types of construction are found in both incremental and absolute encoders. Mechanically, they can be either linear or rotary motion. Geared stages may be present in high resolution rotary encoders. The actual detectors can be magnetic, electro-optical (both LED and filament bulbs), or switch contacts. The task of selecting an encoder must consider all of the above features in terms of the requirements of the application, and no simple selection procedure is possible.

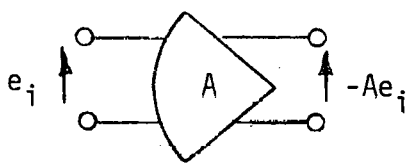
6.2 Op-Amp Review

The term "operational amplifier" was the name coined by analog computer designers to describe the basic amplifier module used to form summers, integrators, and other "operators." Such amplifiers, now found in many applications including analog-to-digital and digital-to-analog devices, are characterized by

(a) High dc voltage gain (10^4 to 10^9)
(b) Wide bandwidth (dc to 100 kC)
(c) Bipolar operation
(d) High input impedance

6.2.1 Idealized Op-Amp Characteristics[1, 8, 9]

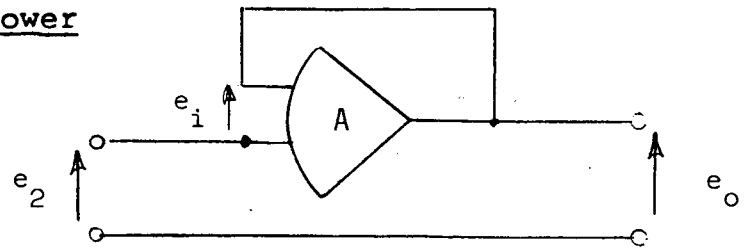
The ideal operational amplifier is represented as a differential-input/differential-output amplifier having ∞ input impedance, ∞ gain, and 0 output impedance. Single-ended inputs and/or outputs are also common. In



practice, op-amps are often used with negative feedback to improve their gain stability and to lower output impedance. Three basic circuits and their output/input relationships are shown below. In analyzing these circuits, the cardinal rules are:

(a) Essentially no current flows into either input terminal of the ideal op-amp, and
(b) when negative feedback is applied, the differential input voltage approaches zero.

Voltage follower



Applying Kirchoffs' voltage law

$$e_2 + e_i - e_o = 0$$

but

$$e_o = -A\, e_i$$

then

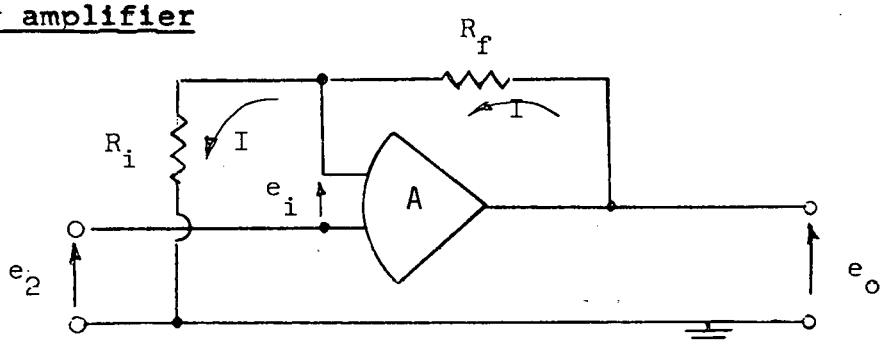
$$e_i = \frac{-e_o}{A}$$

$$e_2 - \frac{e_o}{A} = e_o$$

but as $A \to \infty$,

$$e_2 = e_o$$

Non-inverting amplifier



Since no current flows into the inverting input, the current through $R_f = R_i$ and

$$\frac{e_i + e_2}{R_i} = \frac{e_o - (e_i + e_2)}{R_f}$$

but $e_o = -Ae_i$

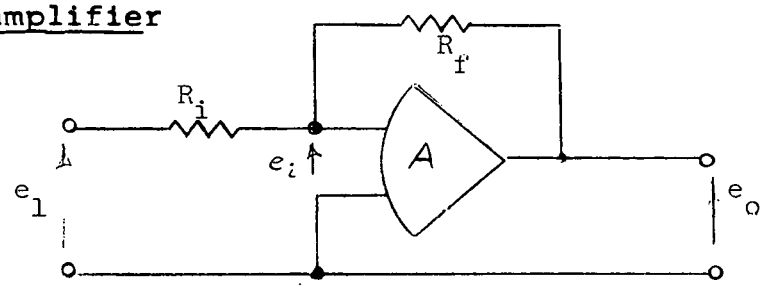
so that

$$\frac{e_i + e_2}{R_i} = -\frac{Ae_i - (e_i + e_2)}{R_f}$$

Then as $A \to \infty$ we have

$$\frac{e_o}{e_2} = \frac{R_f + R_i}{R_i} \qquad \left(\text{Note : } \frac{e_o}{e_2} > 1\right)$$

Inverting amplifier



Again summing currents at the input to zero,

$$\frac{e_1 - e_i}{R_i} + \frac{e_o - e_i}{R_f} = 0$$

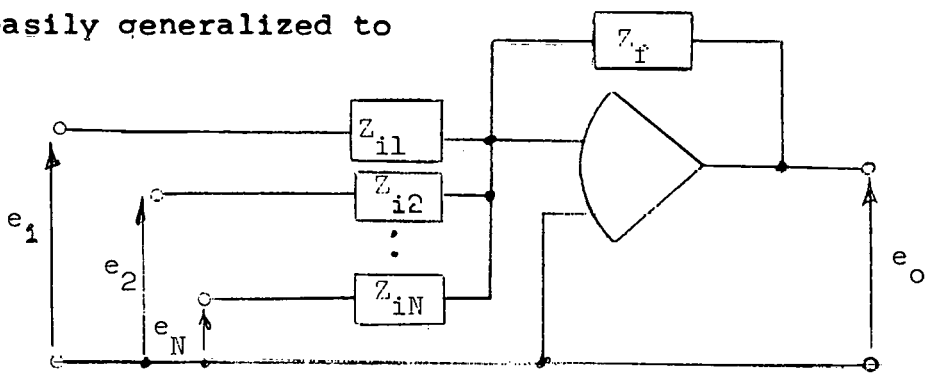
but

$$e_o = -A e_i$$

which leads to (as $A \to \infty$),

$$\frac{e_o}{e_1} = \frac{-R_f}{R_i}$$

For the inverting amplifier, the input/output relationship is easily generalized to



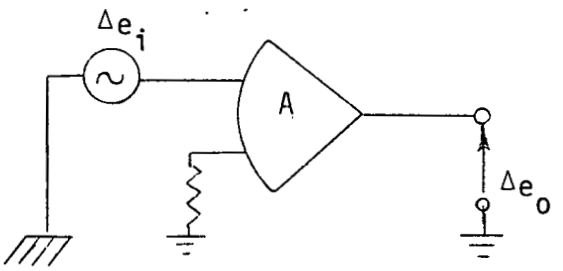
$$e_o = \left(\frac{Z_f}{Z_{i1}}\right)e_1 + \left(\frac{Z_f}{Z_{i2}}\right)e_2 + \ldots + \left(\frac{Z_f}{Z_{iN}}\right)e_N$$

where Z_x represents the dynamic impedance of the input or feedback elements.

6.2.2 Characteristics of Real Operational Amplifiers[1, 8, 9]

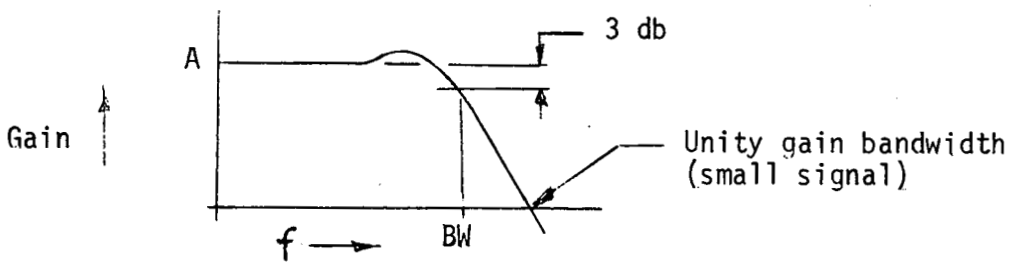
Manufacturers' specification for op-amp performance can be quite difficult to evaluate as each manufacturer has his own interpretation of the various parameters. The principal specifications of concern are as follows:

Open Loop Voltage Gain (A_0)--ratio of change in output voltage to corresponding change in voltage at input terminals. Should be as high as possible since the primary function is to amplify and, in general, the higher the gain, the better the gain accuracy.



$$A_o = \frac{\Delta e_o}{\Delta e_i}$$

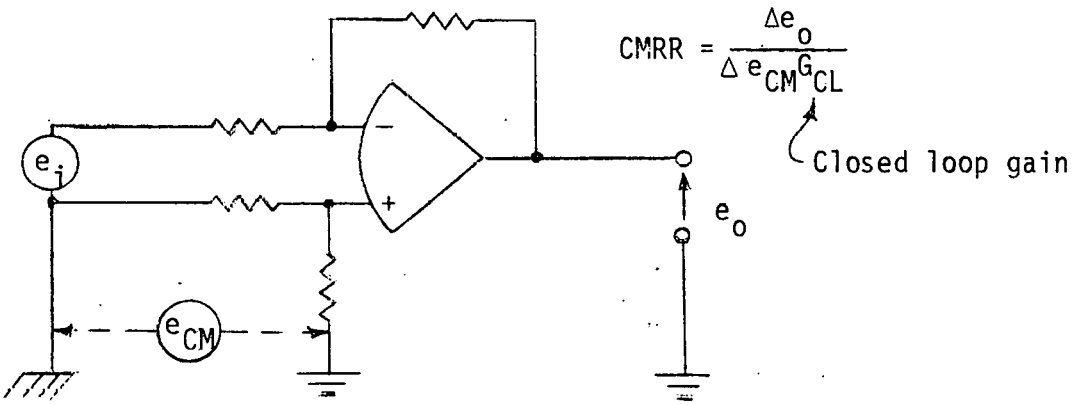
Open Loop Bandwidth (BW_0)--frequency where the high frequency gain is 3 dB less than the gain at a very low frequency. Probably responsible for most operational amplifier design problems. Many applications are designed without allowing for closed-loop gain error due to a lower loop gain at the frequency of interest. Problems are often encountered with amplifier stability due to inadequate or marginal frequency compensation.



Output Impedance (Z_0)--impedance seen by a load at the amplifier output. Excessive output impedance reduces the amplifier's loop gain since, in conjunction with the load and feedback resistors, it forms an attenuator network.

Input Impedance (Z_i)--impedance seen by a source looking into one amplifier input with the other input grounded. Primary effect is to reduce amplifier loop gain, and consequently, alter gain accuracy and stability.

Common-Mode Rejection Ratio (CMRR)--ratio of change in output voltage to change in input common-mode voltage producing it, divided by the closed loop gain. Indicates degree of circuit balance of the amplifier's differential stages, since a common-mode input signal applied to the input terminals should be amplified identically in both sides of the device and produce zero output signal or a numerically infinite decibel rejection ratio.



Input Offset Voltage (V_{IO})--voltage that must be applied at the input terminals to obtain zero output voltage. Indicative of matching tolerances in the differential amplifier stages, it is a major source of offset voltage error in low impedance circuits.

Average Temperature Coefficient of Input Offset Voltage (C_T)--variation in input offset voltage with temperature, divided by temperature change.

Slew Rate (SR)--maximum rate of change of output voltage under large signal conditions.

The performance characteristics of a number of common op-amps are shown in Table 6-1.

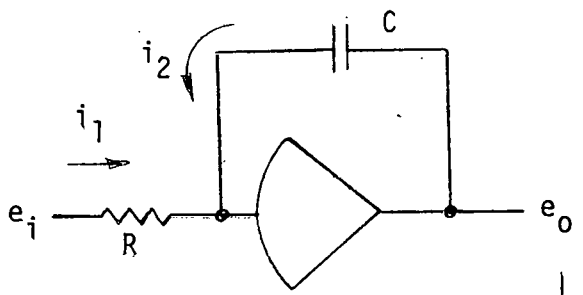
TABLE 6-1 - TYPICAL OP-AMP PARAMETERS[8]

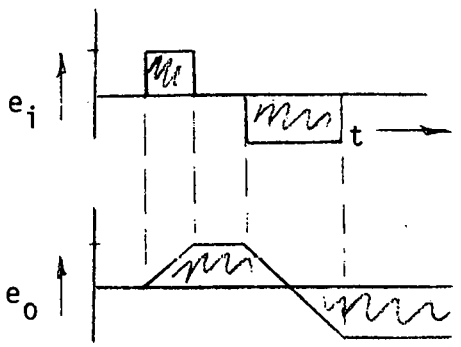
Parameter	Symbol	Units	709	741C	LM101A	747	748	740
Input bias current	I_{IB}	μA max	0.5	0.5	0.075	1.5	1.5	0.004
Input offset voltage	V_{IO}	mV max	5	6	2	6	6	30
Input offset current	I_{IO}	nA max	200	200	10	500	500	0.185
Open loop voltage gain	A_O	V/V min	25,000	20,000	50,000	25,000	25,000	25,000
Output voltage swing	V_O	V_{pk} min	10	10	10	10	10	10
dc Voltage	V_{dc}	V	± 15	± 15	± 15	± 15	± 15	± 15
Unit gain crossover frequency	F_c	MHz typ	0.5	1	1	1	1	1
Slew rate at unit gain	SR	V/μs typ	0.25	0.8	0.5	0.5	0.5	6
Temperature range		°C	-55 to 125	0 to 70	-55 to 125	-55 to 125	-55 to 125	-55 to 125

6.2.3 Op-Amp Functional Circuits

By selecting the appropriate input and feedback elements, various functional circuits can be created:

Integrator





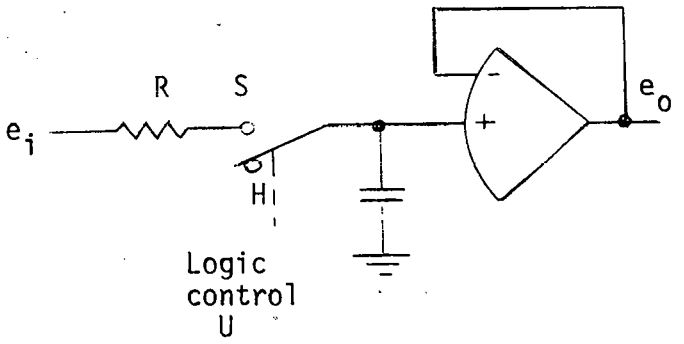
$$i_1 + i_2 = 0$$

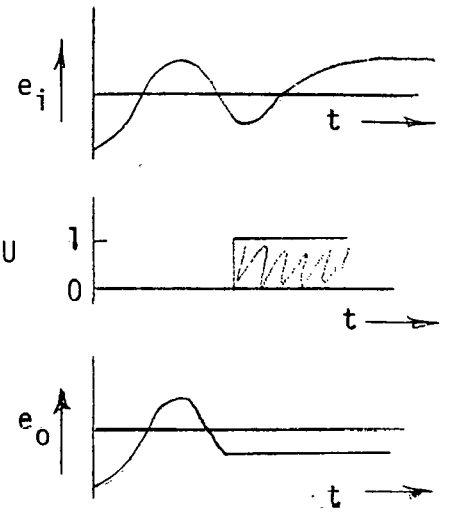
$$\frac{e_i}{R} + C\frac{de_0}{dt} = 0$$

$$\frac{de_0}{dt} = -\frac{1}{RC}\, e_i$$

$$e_0 = -\frac{1}{RC}\int_0^t e_i\, dt$$

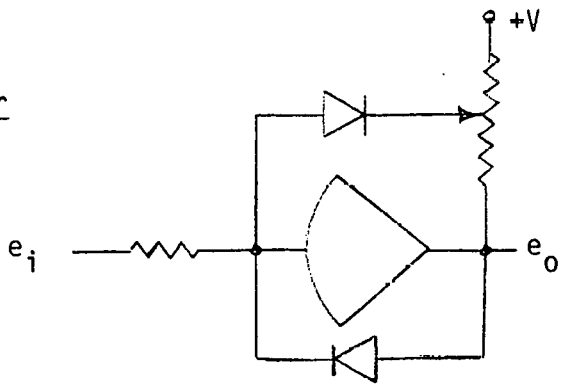
Sample-Hold (Track Store)

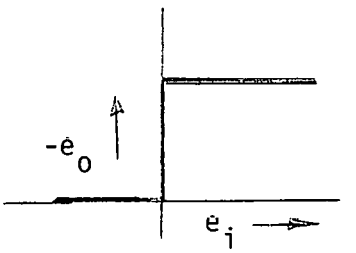




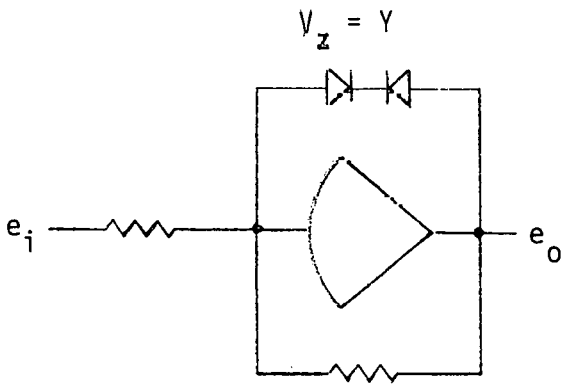
By introducing diodes into the input or feedback path, many limiting, rectifying, and generally nonlinear functions are possible:

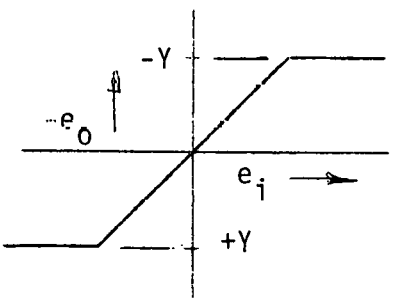
Comparator



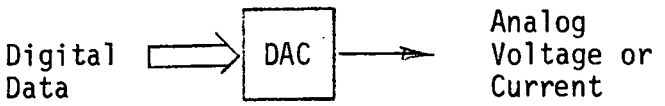


Limiter



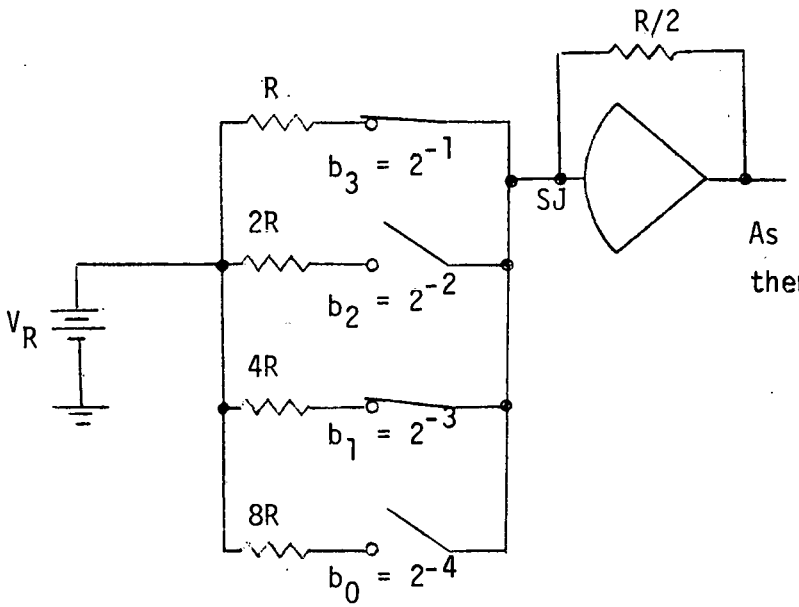


6.3 DAC's and MDAC's



6.3.1 Design Types[1, 9]

Digital-to-analog converters (DAC's) are used to convert digital data to analog voltages or currents. One common configuration is based on the summation of weighted current components:



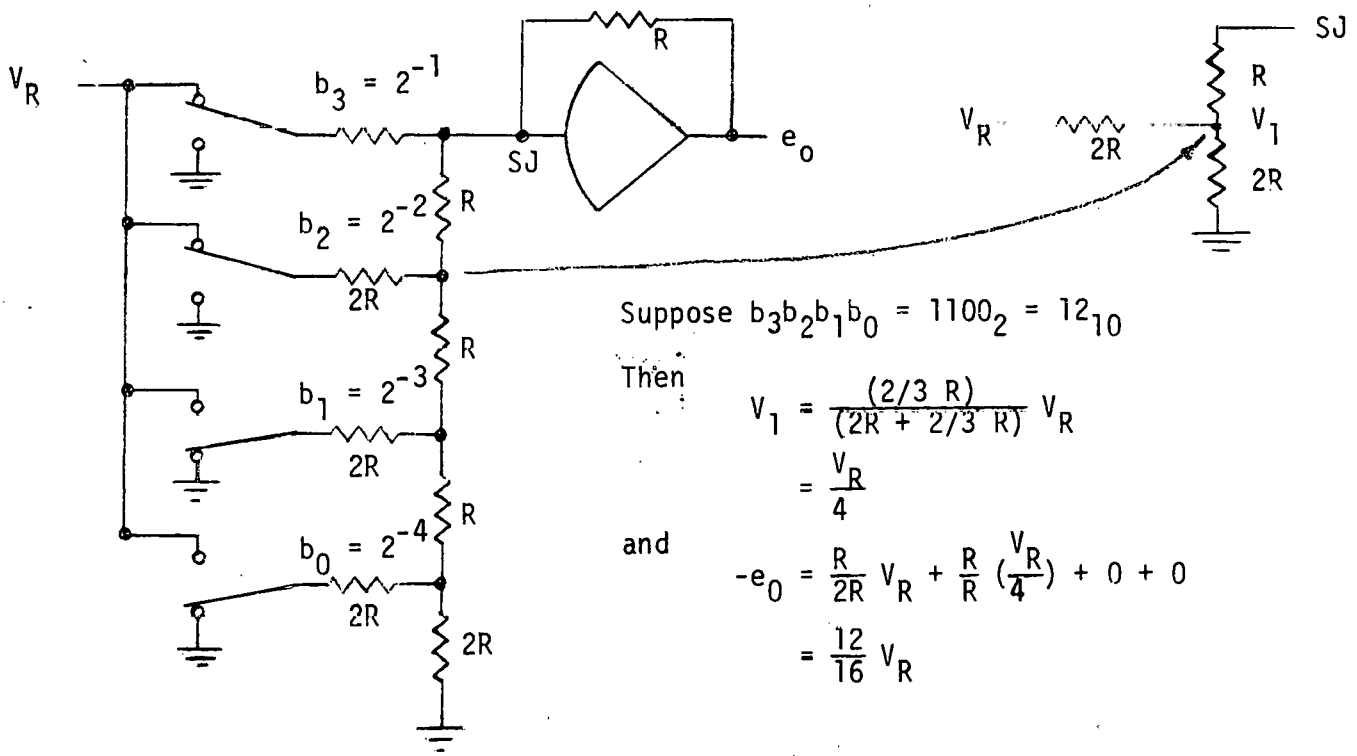
As an example, let $b_3b_2b_1b_0 = 1010_2 = 10_{10}$ then,

$$-e_0 = \left(\frac{R/2}{R} + 0 + \frac{R/2}{4R} + 0\right) V_R$$

$$= (1/2 + 0 + 1/8)\, V_R$$

$$= \frac{10}{16} V_R$$

For a high resolution DAC, a large range of values of precision resistors are required for the above design. This is very difficult to fabricate in an integrated circuit (IC), so a resistor ladder circuit requiring only two different resistor values is often used:



Suppose $b_3b_2b_1b_0 = 1100_2 = 12_{10}$

Then

$$V_1 = \frac{(2/3\ R)}{(2R + 2/3\ R)} V_R$$

$$= \frac{V_R}{4}$$

and

$$-e_0 = \frac{R}{2R} V_R + \frac{R}{R}\left(\frac{V_R}{4}\right) + 0 + 0$$

$$= \frac{12}{16} V_R$$

6.3.2 Specification Parameters

Many different characteristics are used to specify DAC's. The most important of these are as follows:

Digital input code - If the DAC is unipolar, the input digital code is either a binary or BCD fractional number.

Decimal Fraction	Binary Fraction	Code MSB (x1/2)	Bit 2 (x1/4)	Bit 3 (x1/8)	Bit 4 (x1/16)
0	0.0000	0	0	0	0
$1/16 = 2^{-4}$ (LSB)	0.0001	0	0	0	1
2/16 = 1/8	0.0010	0	0	1	0
3/16 = 1/8 + 1/16	0.0011	0	0	1	1
4/16 = 1/4	0.0100	0	1	0	0
5/16 = 1/4 + 1/16	0.0101	0	1	0	1
6/16 = 1/4 + 1/8	0.0110	0	1	1	0
7/16 = 1/4 + 1/8 + 1/16	0.0111	0	1	1	1
8/16 = 1/2 (MSB)	0.1000	1	0	0	0
9/16 = 1/2 + 1/16	0.1001	1	0	0	1
10/16 = 1/2 + 1/8	0.1010	1	0	1	0
11/16 = 1/2 + 1/8 + 1/16	0.1011	1	0	1	1
12/16 = 1/2 + 1/4	0.1100	1	1	0	0
13/16 = 1/2 + 1/4 + 1/16	0.1101	1	1	0	1
14/16 = 1/2 + 1/4 + 1/8	0.1110	1	1	1	0
15/16 = 1/2 + 1/4 + 1/8 + 1/16	0.1111	1	1	1	1

Fractional Binary Codes (9)

When bipolar operation (both positive and negative output voltages) is required, several codes are possible. The most popular are summarized below:

Sign + magnitude = Like binary except that the most significant bit (MSB) is a 1 for negative numbers.

One's Complement = Positive numbers same as binary. Negative numbers obtained by complementing each bit of the positive form.

Two's Complement - Same as binary for positive numbers. Negative numbers obtained by adding one to the one's complement.

Offset binary - Really a natural binary code for four bits with 0 at minus full scale.

The table below shows each of the four codes expressed for a 4-bit data word:

Number	Decimal Fraction: Positive Reference	Decimal Fraction: Negative Reference	Sign + Magnitude	Two's Complement	Offset Binary	One's Complement
+7	+7/8	−7/8	0 1 1 1	0 1 1 1	1 1 1 1	0 1 1 1
+6	+6/8	−6/8	0 1 1 0	0 1 1 0	1 1 1 0	0 1 1 0
+5	+5/8	−5/8	0 1 0 1	0 1 0 1	1 1 0 1	0 1 0 1
+4	+4/8	−4/8	0 1 0 0	0 1 0 0	1 1 0 0	0 1 0 0
+3	+3/8	−3/8	0 0 1 1	0 0 1 1	1 0 1 1	0 0 1 1
+2	+2/8	−2/8	0 0 1 0	0 0 1 0	1 0 1 0	0 0 1 0
+1	+1/8	−1/8	0 0 0 1	0 0 0 1	1 0 0 1	0 0 0 1
0	0+	0−	0 0 0 0	0 0 0 0	1 0 0 0	0 0 0 0
0	0−	0+	1 0 0 0	(0 0 0 0)	(1 0 0 0)	1 1 1 1
−1	−1/8	+1/8	1 0 0 1	1 1 1 1	0 1 1 1	1 1 1 0
−2	−2/8	+2/8	1 0 1 0	1 1 1 0	0 1 1 0	1 1 0 1
−3	−3/8	+3/8	1 0 1 1	1 1 0 1	0 1 0 1	1 1 0 0
−4	−4/8	+4/8	1 1 0 0	1 1 0 0	0 1 0 0	1 0 1 1
−5	−5/8	+5/8	1 1 0 1	1 0 1 1	0 0 1 1	1 0 1 0
−6	−6/8	+6/8	1 1 1 0	1 0 1 0	0 0 1 0	1 0 0 1
−7	−7/8	+7/8	1 1 1 1	1 0 0 1	0 0 0 1	1 0 0 0
−8	−8/8	+8/8		(1 0 0 0)	(0 0 0 0)	

Commonly-Used Bipolar Codes (9)

Output Range - Typically 10 volts full scale (unipolar), or ± 10 volts (bipolar).

Offset Error - The digital input code for 0 may not result in exactly 0 input.

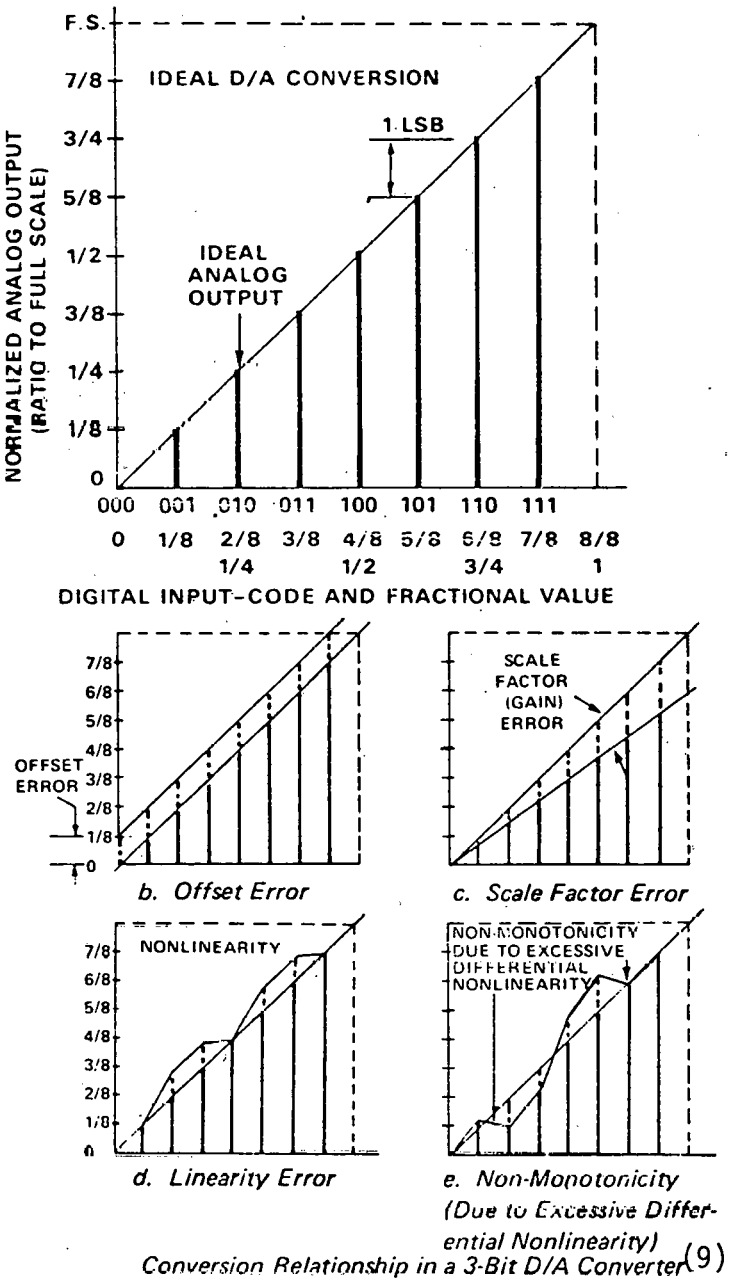
Gain Error - The range from 0 to full scale may be something other than as specified.

Nonlinearity - Unit (incremental) steps in the binary input code may result in voltage steps of varying size.

Nonmonotonicity - A form of nonlinearity in which an increase in value of the digital code actually results in a decrease in output voltage.

Resolution - The output of the DAC is quantitized in steps proportional to the value of the least significant bit (LSB). For an n-bit DAC, the output is quantitized in steps of

$$\left(\frac{1}{2^n}\right) V_R$$



b. Offset Error

c. Scale Factor Error

d. Linearity Error

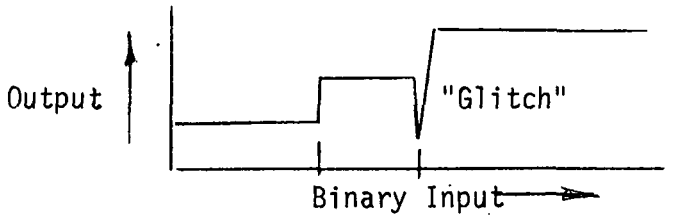
e. Non-Monotonicity (Due to Excessive Differential Nonlinearity)

Conversion Relationship in a 3-Bit D/A Converter (9)

a. Ideal Relationship

b, c, d, e. Typical Sources of Error

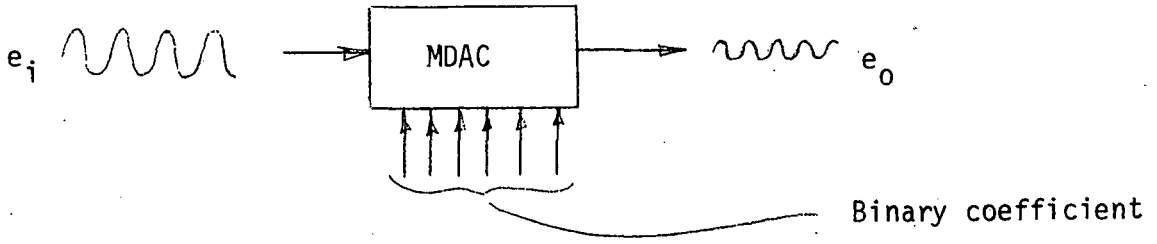
Glitches - At places in the digital input code where several bits change at once, the output may exhibit a large transient swing due to unequal timing of the conversion circuits for each bit. These transient spikes are called "glitches."



Speed - The time to convert a digital data word to an analog output voltage. DAC's are quite fast, many operating in submicrosecond range.

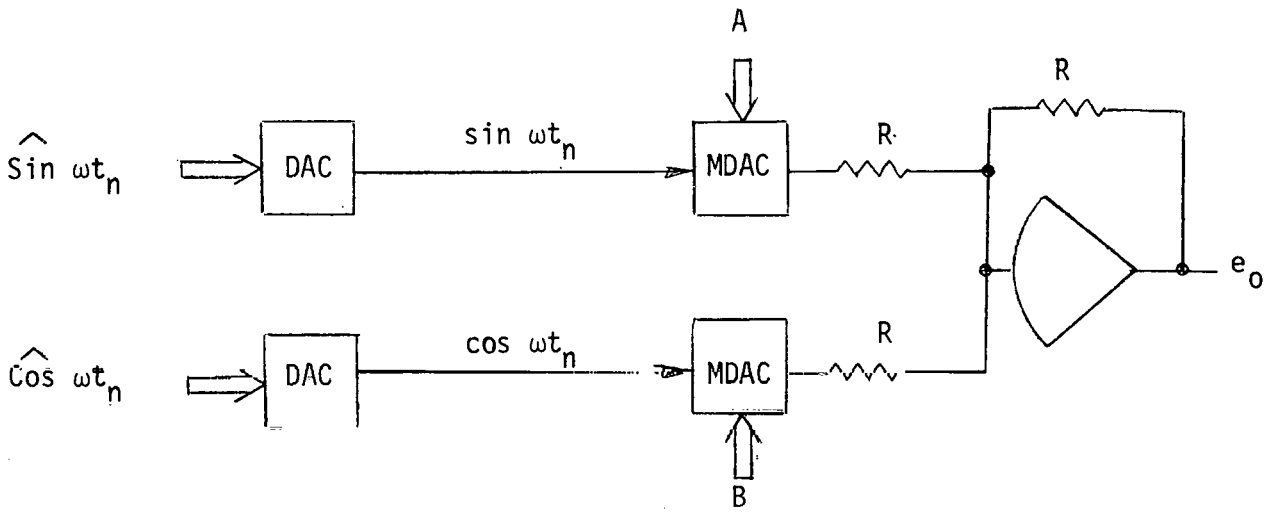
Accuracy - The overall accuracy of a DAC is determined by a combination of the resolution and linearity specifications.

In the preceding discussion of DAC's the reference voltage was considered to be a fixed quantity. In a multiplying DAC (MDAC), an analog input serves as the reference and the device performs similar to a potentiometer.



Depending on the design of the device, it may be able to accept $\pm$ digital coefficients (4-quadrant operation) as well as $\pm$ analog inputs (2-quadrant operation).

EX 6-1 - The following circuit illustrates the application of DAC's and MDAC's in a waveform synthesizer:



$$- e_0 = A \text{ Sin } \omega t_n + B \text{ Cos } \omega t_n$$

$$= X \text{ Sin } (\omega t_n + \phi)$$

where $X = \sqrt{A^2 + B^2}$ $\qquad \phi = \text{Tan}^{-1} \left(\frac{B}{A}\right)$

DIGITAL-TO-ANALOG CONVERTER

MODEL DAC-HI 12B

GENERAL DESCRIPTION

The DAC-HI12B is a 12 bit Digital to Analog converter featuring a state-of-the-art output settling time of 25 nanoseconds combined with a ±1/2LSB linearity and a temperature coefficient of ±20ppm/°C.

Bipolar operation is achieved by externally pin strapping a built-in offsetting reference. Input coding can be straight binary for unipolar output, or a choice of offset binary or two's complement for bipolar output.

The DAC-HI12B is completely self-contained, requiring only ±15 volts D.C. power. Packaged in a 2"x2"x0.375", low profile module, it is readily soldered or plugged directly into P.C. cards or other mother board hardware. Included in each module is digital interface logic, a precision resistor ladder network, high speed electronic switches, and a temperature compensated precision voltage reference source.

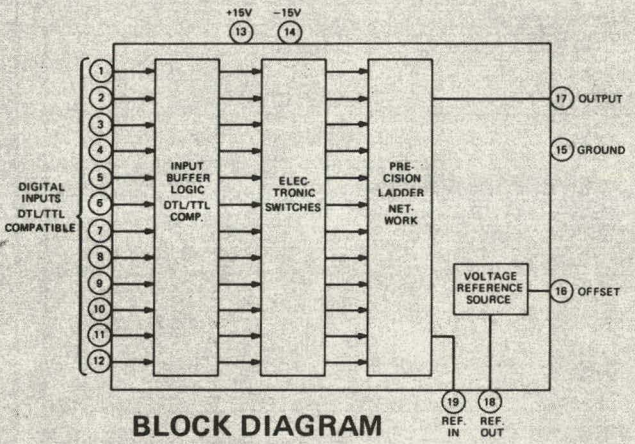
One of the prime features is the unit's output flexibility with a 5 ma current output which can be fed directly into an external resistor to develop a 1.2V maximum output or, by external pin strapping, a bipolar output of ±1.2V maximum can be generated across the output load resistor. The output current can also be fed into an operational amplifier for those who require sign inversion, scaling etc. This amplifier can be selected to suit a particular application.

Applications for the DAC-HI12B include high speed two quadrant multipliers, graphic generators, CRT displays, high speed function generators and programmers.

12 BINARY BIT RESOLUTION
25 NSEC OUTPUT SETTLING TIME

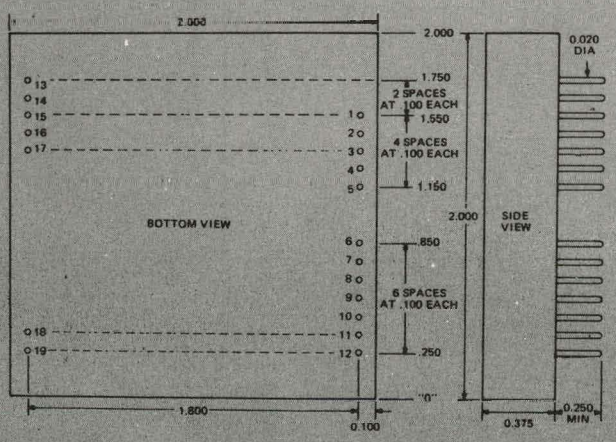
FEATURES

- ☐ ULTRA FAST SETTLING ..... 25nsec to 0.1% of FS
- ☐ HIGH RESOLUTION ..... 12 BINARY BITS
- ☐ SMALL SIZE ..... 2" x 2" x 0.375" MOUNTS ON PCBs WITH 0.5" CENTERS
- ☐ LOW TEMPERATURE COEFFICIENT ..... ± 20 PPM/°C



BLOCK DIAGRAM
MODEL DAC–HI12B

MECHANICAL DIMENSIONS (INCHES)



INPUT/OUTPUT CONNECTIONS

PIN	FUNCTION
1	BIT 1 INPUT (MSB)
2	BIT 2 INPUT
3	BIT 3 INPUT
4	BIT 4 INPUT
5	BIT 5 INPUT
6	BIT 6 INPUT
7	BIT 7 INPUT
8	BIT 8 INPUT
9	BIT 9 INPUT
10	BIT 10 INPUT
11	BIT 11 INPUT
12	BIT 12 INPUT (LSB)
13	+15V POWER INPUT
14	−15V POWER INPUT
15	COMMON GROUND
16	OFFSET
17	ANALOG OUTPUT
18	REFERENCE OUTPUT
19	REFERENCE INPUT

SPECIFICATIONS

MODEL NUMBER	**DAC-HI 12B**
DIGITAL INPUTS	
RESOLUTION	12 Binary Bits
CODING (Parallel Data in the following Formats)	Straight Binary (Unipolar Output) Offset Binary (Bipolar Output) Two's Complement (Note 1) (Bipolar Output)
DATA INPUTS	DTL or TTL Compatible Positive Logic Loading: 2 standard TTL loads

INPUT CODE	VINPUT MIN.	VINPUT MAX.	BIT STATUS
"0"	0V	+0.8V	OFF
"1"	+2.0V	+5.5V	ON

ANALOG OUTPUT (@25°C)	
ACCURACY	Adj. to ±0.01%
TYPE OF OUTPUT	Current
FULL SCALE OUTPUT	5 ma @ +1.2V max. (Unipolar) ±2.5 ma @ ±1.2V max. (Bipolar)
OUTPUT IMPEDANCE	600 Ohms ±1%
OUTPUT ZERO OFFSET	15 na
OUTPUT LOADING	300 ohms for 0 to +1V Output 2.325K for ±1.0V Output
OUTPUT SETTLING TIME	50 nsec to ±0.025% of FS
OUTPUT RESOLUTION	1 LSB (1.25μa for 12 Binary Bits)
LINEARITY	±.625μA (1/2 LSB)
TEMPERATURE COEFFICIENT	±20 ppm/°C of FS
LONG TERM STABILITY	±0.5%/Yr.
REFERENCE SOURCE	Internal
INPUT POWER REQUIREMENTS (2)	+15VDC, ±0.5%V @ 40 ma −15VDC, ±0.5%V @ 20 ma
POWER SUPPLY REJECTION RATIO	.0085%/V
PHYSICAL – ENVIRONMENTAL	
OPERATING TEMPERATURE RANGE	0° to +70°C
STORAGE TEMPERATURE RANGE	−55°C to +85°C
RELATIVE HUMIDITY	Up to 100% Non-Condensing
SIZE	2"L x 2"W x 0.375"H Plug-in Module
PINS	0.020" Round Gold Plated 0.250" Long Min.
CASE MATERIAL	Black Diallyl Phthalate
WEIGHT	2 oz
MATING CONNECTOR	DILS-2 2 per module $4.00 per pair
PRICE (1-9)	$149 ea.

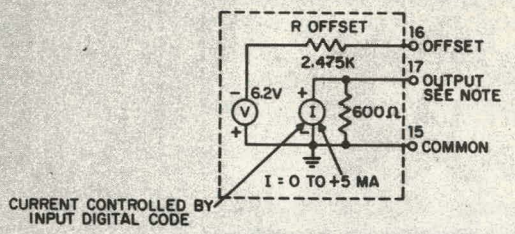
INPUT CODING

ANALOG OUTPUT (+ 2.5mA FS)	OFFSET BINARY	2'S COMPLEMENT (1)
+2.49875mA	111111111111	011111111111
+1.25000mA	110000000000	010000000000
0mA	100000000000	000000000000
−1.25000mA	010000000000	110000000000
−2.50000mA	000000000000	100000000000

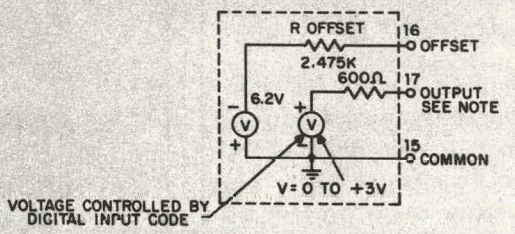
ANALOG OUTPUT (0 to +5mA FS)	STRAIGHT BINARY
+4.99875mA	111111111111
+3.75000mA	110000000000
+2.50000mA	100000000000
+1.25000mA	010000000000
000000mA	000000000000

DAC–HI 12B APPLICATION NOTES

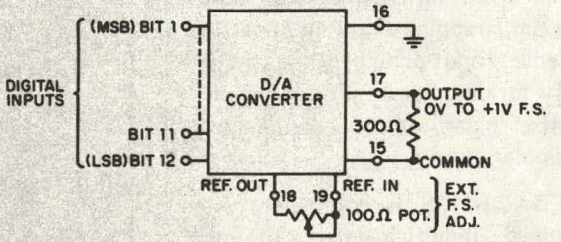
CURRENT EQUIVALENT CIRCUIT



VOLTAGE EQUIVALENT CIRCUIT



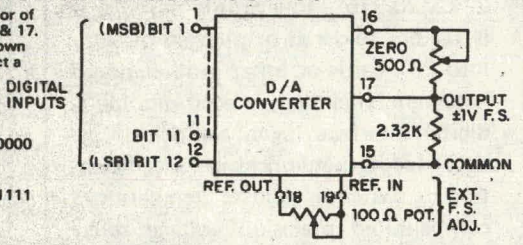
UNIPOLAR CURRENT OUTPUT



BIPOLAR CURRENT OUTPUT

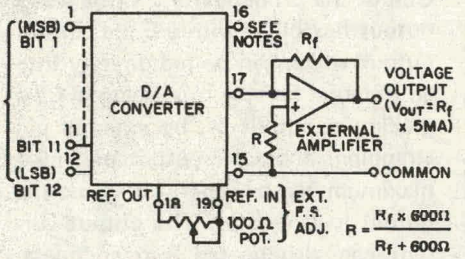
Connect external load resistor of 2.32K Ohms across pins 15 & 17. Connect a 500 Ω pot. as shown across pins 16 & 17. Connect a 100 Ω pot. as shown across pins 18 & 19.

Calibration Procedure
With input code 000000000000 adjust the 500 Ω pot. for −1.0000 volts output.
With input code 111111111111 adjust the 100 Ω pot. for +0.99976 volts output.



UNIPOLAR OR BIPOLAR VOLTAGE OUTPUT

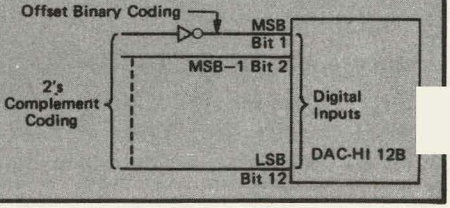
For Unipolar voltage output connect jumper between pins 15 & 16. For Bipolar voltage output connect a 500 Ohm potentiometer between pins 15 & 17 and adjust for zero output with 100000000000 input.



Note: (1) The converter accepts only straight binary or offset binary coding. It does not directly accept 2's complement coding without first complementing the 2's comp. most significant bit before providing it to the converter. After this bit is complemented, the code appears as normal offset binary which the converter can process.

(2) See power supply catalog, #PSC 3-73-1

2'S COMPLEMENT TO OFFSET BINARY CONVERSION (1)

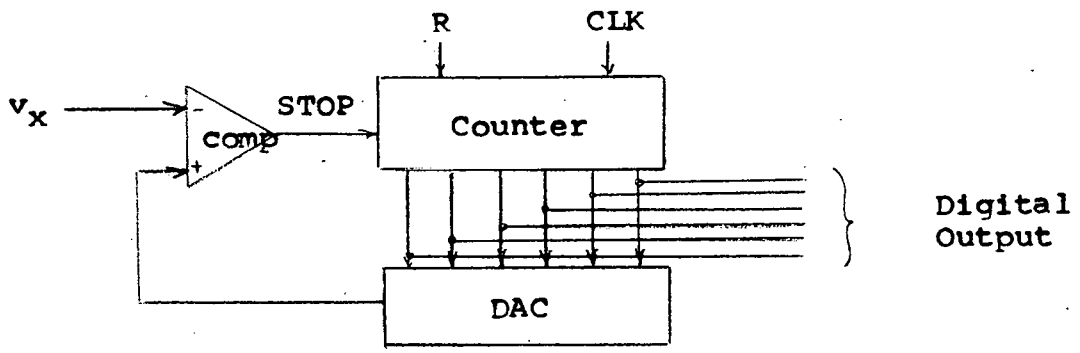


6.4 Analog-to-Digital Converters (ADC's)

6.4.1 Design Types[1, 9]

Analog-to-digital converters are used to convert analog voltages or currents to representative digital values.

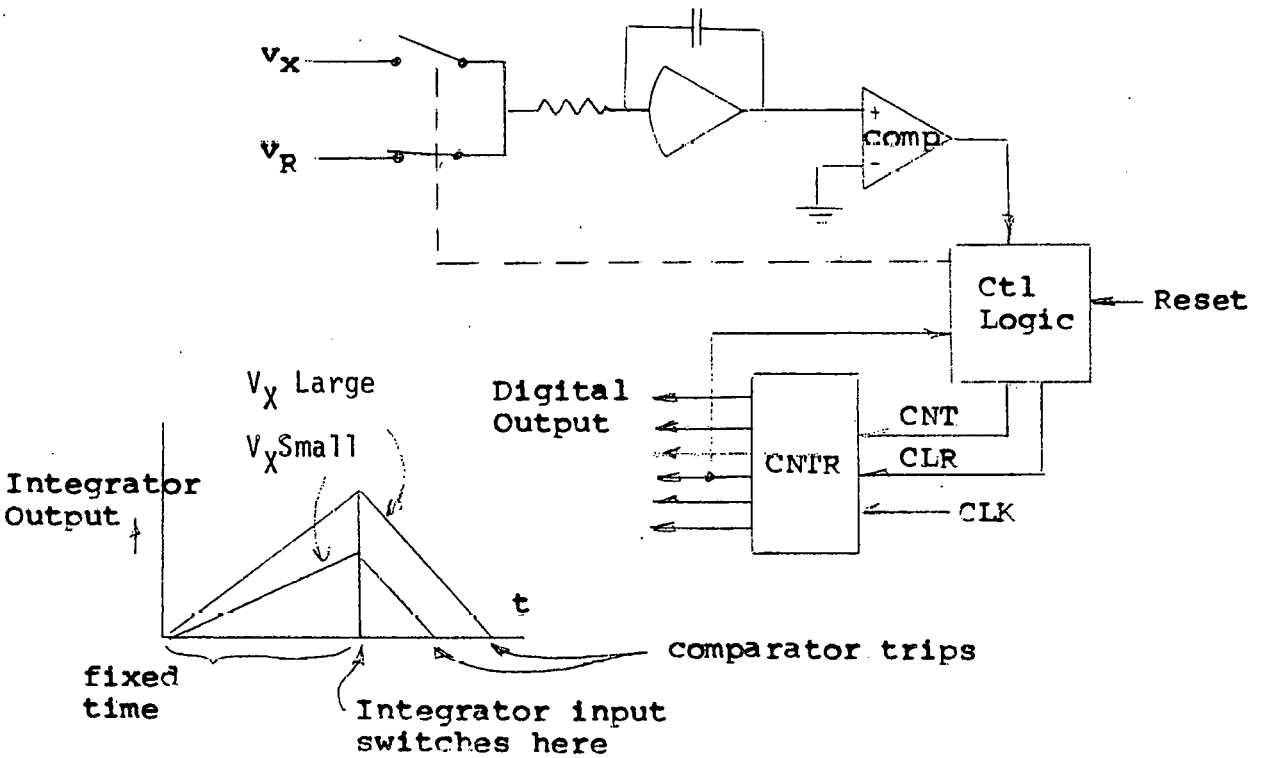
ADC's are often formed by incorporating a DAC in a feedback loop:



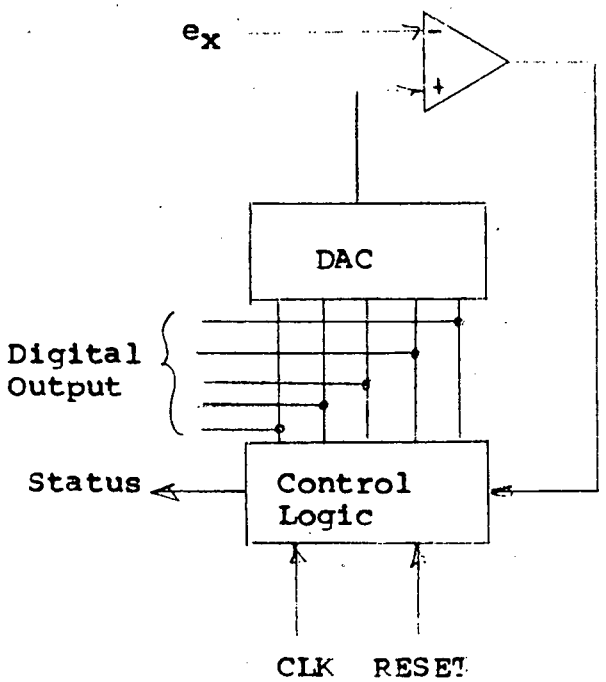
In the above ADC, the DAC output and the unknown analog voltage are compared to control the counter operation. A STOP signal is given to the counter when the comparator trips. The counter output at that point is the digital value corresponding to the unknown voltage. The counter must be reset to get a new reading. Although this technique is slow, it is also inexpensive and is often used in digital volt meters.

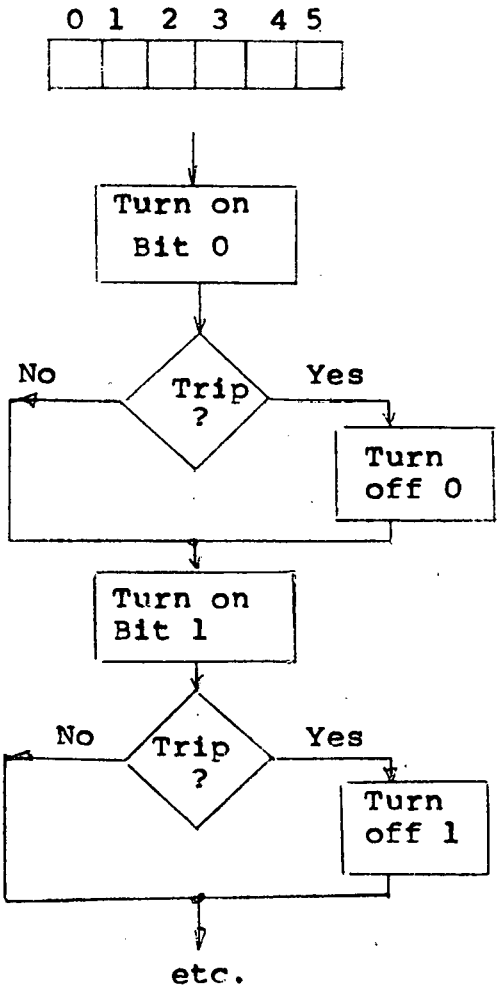
Another type of ADC found in volt meters is the dual-slope integrating ADC. At the start of a conversion cycle the unknown input voltage is integrated for a fixed period of time (usually a multiple of 16 2/3 ms = 1/60 hz). At the end of that time period the integrator input is switched to a negative reference voltage and the digital counter begins counting clock pulses.

When the comparator trips, counting is stopped and the digital counter output at that point is proportional to the unknown voltage. Long term stability effects are eliminated since the same clock and integrator are used for integrating up as well as down.

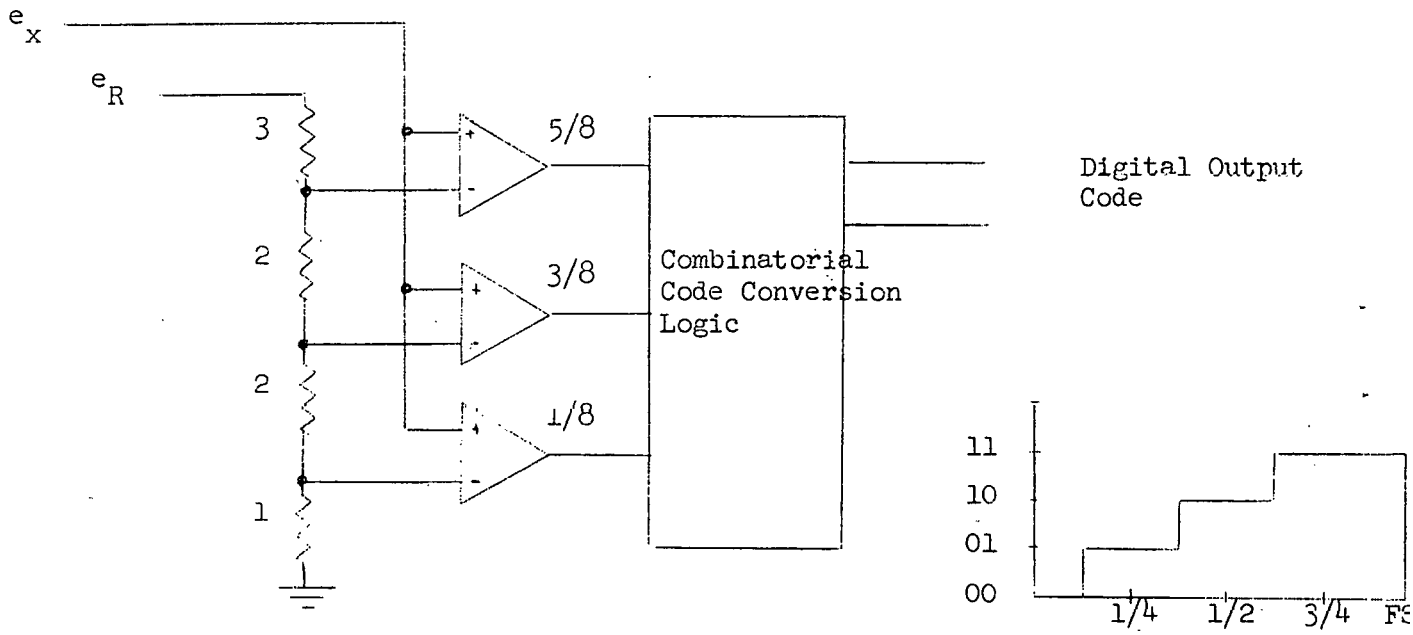


ADC's in computer peripherals are generally of the successive approximation or parallel conversion type. In the successive approximation design each bit of a DAC, starting with the MSB, is turned on in succession. If the comparator does not trip for that bit, it is left on and the next most significant bit is tried. After only n comparisons, the n-bit digital output is available. For the purpose of timing communications with the computer, a status bit is turned on when the conversion is complete.



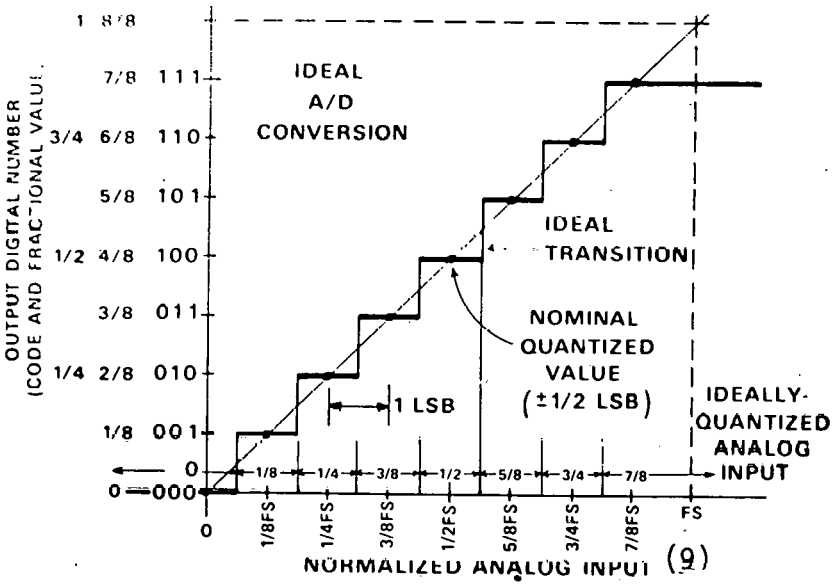


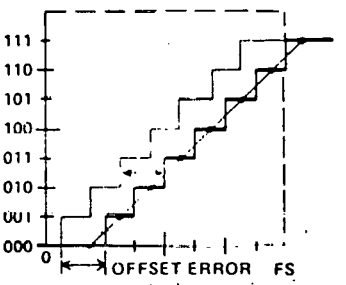
The parallel conversion type of ADC represents the ultimate in speed and cost. Every analog level is detected using a separate comparator and a reference voltage biasing network. For n bits of resolution, 2^n-1 comparators are required. The 2^n-1 comparator outputs must be encoded by combinatorial circuitry to form the n-bit output data word. A 2-bit parallel ADC is illustrated on the following page.



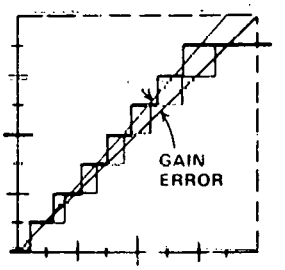
6.4.2 Specification parameters[9]

The definitions of the digital codes, offset error, gain error, nonlinearity, accuracy, etc. used for DAC's also applies to ADC's. Several of these are illustrated below.

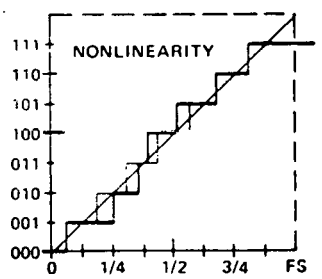




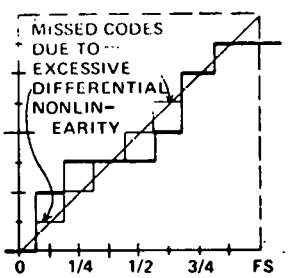
b. Offset Error



c. Scale Factor Error



d. Linearity Error



e. Missed Codes

MAXIMUM AVAILABLE RESOLUTION vs BINARY BITS (9)

Binary Bits (n)	(2^n)	Equivalent Percent or Fraction of Range of Least-Significant Bit*		Residual $1 - \sum_{1}^{n} (\frac{1}{2^n})$
		Percent	ppm	
1	2	50.0	500 000.	0.5
2	4	25.	250 000.	0.25
3	8	12.5	125 000.	0.125
4	16	6.25	62 500.	0.062 5
5	32	3.125	31 250.	0.031 25
6	64	1.562 5	15 625.	0.015 625
7	128	0.781 25	7 812.5	0.007 812 5
8	256	0.390.625	3 906.25	0.003 906 25
9	512	0.195 313	1 953.13	0.001 953 13
10	1 024	0.097 656	976.56	0.000 976 56
11	2 048	0.048 828	488.28	0.000 488 28
12	4 096	0.024 414	244.14	0.000 244 14
13	8 192	0.012 207	122.07	0.000 122 07
14	16 384	0.006 104	61.04	0.000 061 04
15	32 768	0.003 052	30.52	0.000 030 52
16	65 536	0.001 526	15.26	0.000 015 26
17	131 072	0.000 763	7.63	0.000 007 63
18	262 144	0.000 381	3.81	0.000 003 81
19	524 288	0.000 191	1.91	0.000 001 91
20	1 048 576	0.000 095	0.95	0.000 000 95
21	2 097 152	0.000 048	0.48	0.000 000 48
22	4 194 304	0.000 024	0.24	0.000 000 24
23	8 388 608	0.000 012	0.12	0.000 000 12
24	16 777 216	0.000 006	0.06	0.000 000 06

*May be limited by noise and other uncertainties in actual circuit.

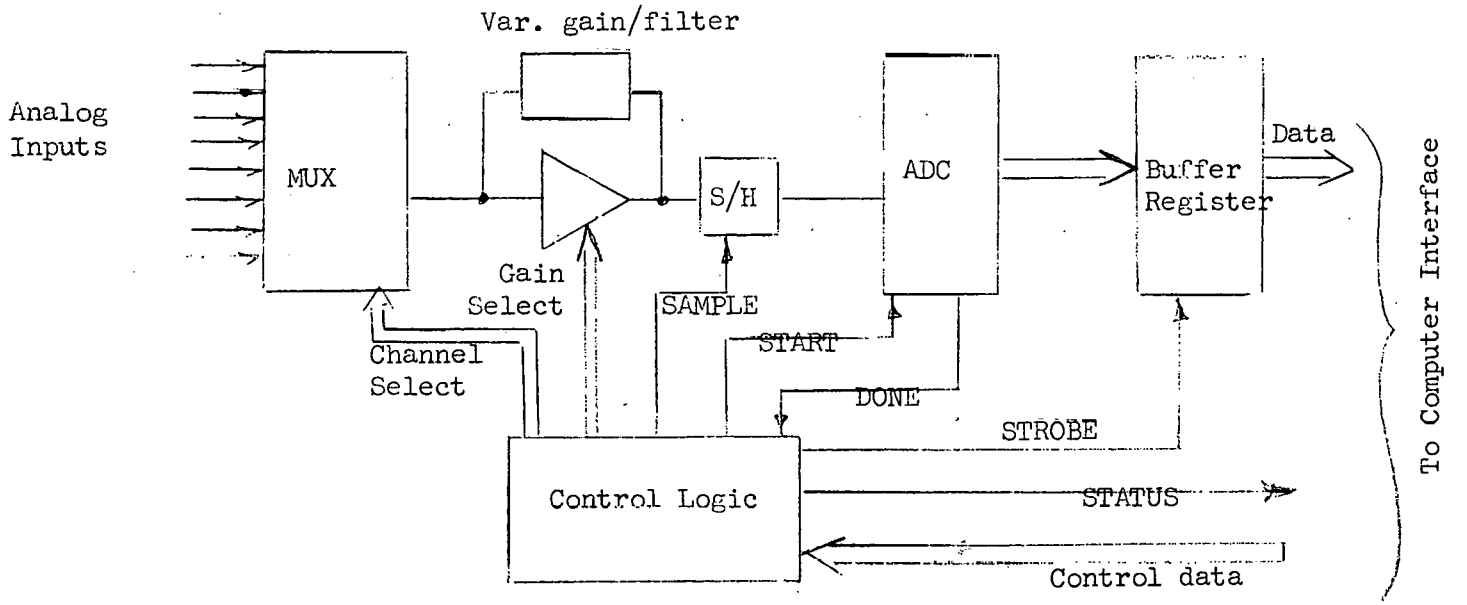
In addition, the following terminology is used -

Quantitization error - Since the A/D process results in output transitions which occur half-way between analog input 'steps', there is an inherent quantitization uncertainty of ± ½ LSB associated with any digital output.

Missing codes - This is similar to the monotonicity error in a DAC. Because of large differential linearity errors (i.e., detection of unequal analog input steps), certain output codes will never appear.

6.4.3 Analog-to-digital subsystems[9]

Most data acquisition and/or control systems are characterized by many channels of analog input. In order to communicate to the computer over one set of data lines, it becomes desirable to multiplex the data. In addition to performing the multiplexing, computer peripheral A/D subsystems also filter, sample, and scale the data. A typical system is shown below :



ANALOG-TO-DIGITAL CONVERTERS

ADC-D SERIES
ADC-K SERIES

GENERAL DESCRIPTION

Moderate accuracy, medium conversion speed and relatively low cost is the theme behind the ADC-D and K series. Both use the successive approximation conversion technique. This encoding method is the most popular of all the A/D conversion techniques because it offers a favorable combination of a full monotonic conversion with excellent linearity over the full scale input range.

ADC-D and K series have a total conversion speed of 20KHz (50 μsec). Voltage input can be unipolar (0 to +10V) or bipolar (±5V) by external pin strapping. Output coding can be straight binary, offset binary or two's complement with word lengths of 8,10, and 12 binary bits.

Specified accuracy is ±0.05% for ADC-D series and ±0.02% for ADC-K series. Model ADC-D and ADC-K feature 50 ppm/°C and 30 ppm/°C temperature coefficients respectively and need not be adjusted over an operating temperature range of 0° to +70°C. Digital outputs include up to 12 parallel lines, serial output and an end of conversion status.

Overall dimensions are 2"W x 4"L x 0.4"H. Input power requirements are ±15VDC and +5VDC and all input control lines and digital outputs are DTL/TTL compatible.

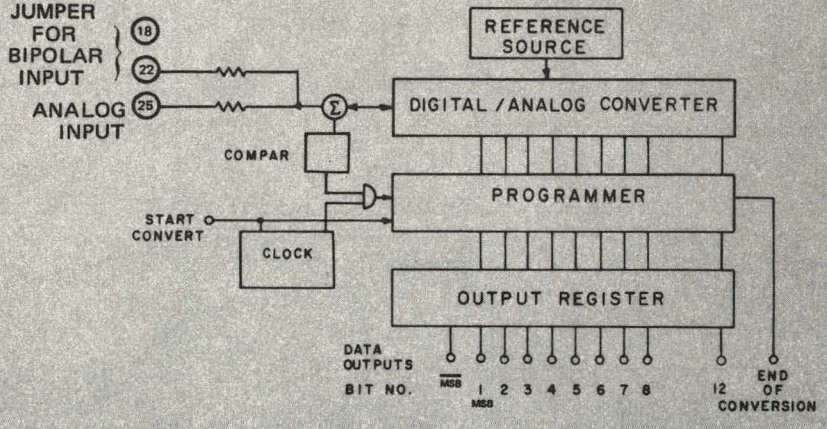
Basic ingredients of each series is a temperature compensated voltage reference source, successive approximation logic, output storage register/programmer, a low noise voltage comparator and a precision digital to analog converter.

All models feature dual-in-line pinning compatibility on .100" grid pin spacing.

MODERATE ACCURACY AND SPEED AT LOW COST
PRICED FROM $69.00

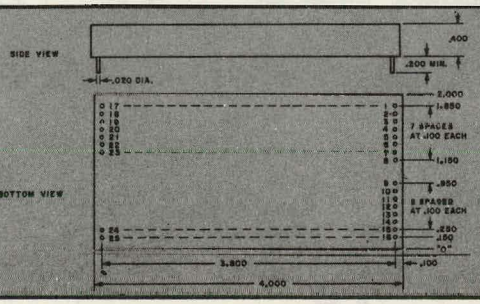
FEATURES

- ☐ Highly accurate . . . . . ±0.025%
- ☐ Moderate conversion rate . . . . 20 KHz word rate
- ☐ High resolution . . . . . Up to 12 binary bits.
- ☐ Ultra-Stable . . . . . ±0.003%/°C
- ☐ Adjustment-Free . . . . . From 0°C to +70°C
- ☐ Small size . . . . . 3.2 cubic inches.
- ☐ Low profile package . . . . . 0.4 inches
- ☐ Completely self-contained . . . . Simply apply D.C. power.
- ☐ Hardware compatible . . . . . Will mount on 0.5" card file centers. Compatible with dual in-line pinning.
- ☐ OEM designed . . . . . Generous discounts available.



MODEL ADC - D and K SERIES BLOCK DIAGRAM

MECHANICAL DIMENSIONS (INCHES)



INPUT/OUTPUT CONNECTIONS

PIN	FUNCTION	PIN	FUNCTION
1	E.O.C. (STATUS)	17	+ 5 VDC POWER IN
2	MSB OUTPUT	18	+15 VDC POWER IN
3	START CONVERT	19	-15 VDC POWER IN
4	SERIAL OUTPUT	20	POWER GROUND
5	BIT 1 OUT (MSB)	21	OFFSET (NOTE 1)
6	BIT 2 OUT	22	OFFSET (NOTE 2)
7	BIT 3 OUT	23	GAIN ADJUST
8	BIT 4 OUT	24	ANALOG GND
9	BIT 5 OUT	25	ANALOG INPUT
10	BIT 6 OUT		
11	BIT 7 OUT		
12	BIT 8 OUT		
13	BIT 9 OUT		
14	BIT 10 OUT		
15	BIT 11 OUT		
16	BIT 12 OUT (LSB)		

SPECIFICATIONS

ELECTRICAL

Inputs:

Analog input voltage range . . Standard ranges of 0V to +10V FS, ±5V FS via ext. pin strapping.

Input Overvoltage . . . . . ±15V DC without damage of unit.

Input Impedance . . . . . Standard 10K ohms ±1% shunted with 10 pf single-ended to ground.

Start of Conversion . . . . . 2V min. to 7V max. positive pulse with duration of 100 nsec min.
"1" resets the converter.
"0" initiates conversion.
Loading of one TTL load.

Outputs:

Parallel output data . . . . . Up to 12 parallel lines of data held until next conversion command.
Vout ("0") ≤ +0.8V
V out ("1") ≥ +2.4V
Each output capable of driving up to 6 TTL loads.

Coding . . . . . Straight Binary (Unipolar Input)
Offset Binary (Dipolar Input)
Two's Complement (Bipolar Input)

Serial Output . . . . . NRZ successive decision pulse output generated during conversion, with MSB first.

End of Conversion . . . . . Conversion Status Signal
Vout ("0")≤ +0.8V conversion complete
Vout ("1") ≥ +2.4V during reset and conversion period.

Performance:

Resolution . . . . . One part in 2^n (max. resolution 12 binary bits or 3 digit BCD). (n= number of binary bits).

Accuracy (@25°C) . . . . . **ADC-D Series**
8 Binary Bits – ±0.05% of FS ±½ LSB
10 Binary Bits – ±0.05% of FS ±½ LSB
12 Binary Bits – ±0.025% of FS ±½ LSB

ADC-K Series
8 Binary Bits, 10 Binary Bits, 12 Binary Bits } ±0.025% of FS ±½ LSB

Total Conversion Time . . . . 100 µsec – ADC-D Series
50 µsec – ADC-K Series

Input Power Requirements . ±15VDC, ±0.5VDC @ ±35 ma
+5VDC, ±0.5VDC @ 300 ma (3)

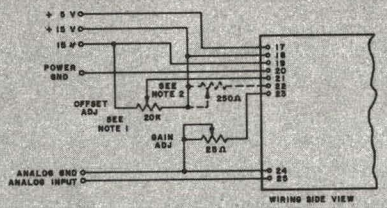
PHYSICAL-ENVIRONMENTAL

Operating temperature range . . . . . 0°C to +70°C
Storage temperature range . . −55°C to +85°C
Relative Humidity . . . . . Up to 100% Non-condensing
Size . . . . . 2"W x 4"L x 0.4"H
Pins . . . . . 0.020" round gold plated 0.250" long min.
Case Material . . . . . Black Diallyl Phthalate Per MIL-M-14.
Weight . . . . . 4 oz.

Model ADC-D and K Series A/D converter modules are fully encapsulated and feature dual in-line pinning compatibility (i.e. 0.100" grid pin spacing and 2.800" or 3.800" between rows of pins) permitting direct plug-in to AUGAT, CAMBION, EECO, etc., circuit boards.

Note 3: See power supply catalog #PSC 3-73-1

GAIN & OFFSET EXTERNAL ADJUSTMENT CONNECTIONS



ALL TRIMPOTS ARE ≤ 100 PPM

NOTES: 1. OFFSET ADJUSTMENT FOR UNIPOLAR 0V TO +10V USE. WHEN NOT IN USE, CONNECT PIN 21 TO GROUND.
2. OFFSET ADJUSTMENT FOR BIPOLAR ±5V FS USE. WHEN NOT IN USE, PIN 22 REMAINS OPEN.

ORDERING INFORMATION

ADC-D
or
ADC-K

NUMBER OF BITS AND CODING
8B = 8 BINARY BITS
10B = 10 BINARY BITS
12B = 12 BINARY BITS

CODING FOR ADC-D AND K SERIES CONVERTERS

Analog Input Range ±5V, FS	Offset Binary	2's Complement	Analog Input Range (0 to +10V, FS)	Straight Binary
+4.9975	111111111111	011111111111	+9.9975	111111111111
+4.3750	111100000000	011100000000	+8.7500	111000000000
+3.7500	111000000000	011000000000	+7.5000	110000000000
+2.5000	110000000000	010000000000	+5.0000	100000000000
0.0000	100000000000	000000000000	+2.5000	010000000000
–2.5000	010000000000	110000000000	+1.2500	001000000000
–3.7500	001000000000	101000000000	0.0000	000000000000
–4.3750	000100000000	100100000000		
–4.9975	000000000001	100000000001		
–5.0000	000000000000	100000000000		

PRICE:

ADC-D8B – $ 69.00 ea.
ADC-D10B – $ 89.00 ea.
ADC-D12B – $109.00 ea.
ADC-K8B – $ 99.00 ea.
ADC-K10B – $109.00 ea.
ADC-K12B – $129.00 ea.

Mating Connector – DILS-2, 2 per module, $4.00/pair.

Multiplexer (MUX)

In the above system, the multiplexer is used to select one of many analog input signals. In high speed, high level systems, the selecting is done with electronic switches, whereas for low-level medium-speed systems it is done with reed relays. The important parameters to be considered in multiplexers are :

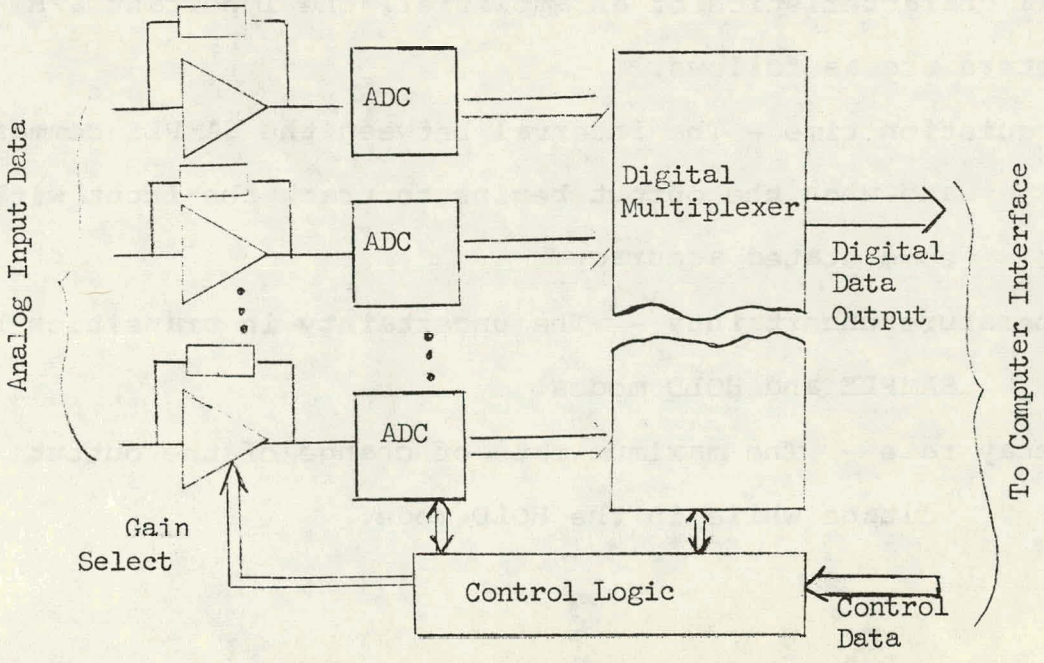
Transfer accuracy - There should be precise agreement between input and output voltages.

Settling time - The time between the start of channel selection and arriving at a stable output voltage.

Throughput rate - How fast can channels be selected

Crosstalk - The individual channels should have no influence on each other.

In high resolution, high-speed systems, the cost of the ADC is the dominating factor and it is desirable to multiplex the analog signals. For less accurate systems, the least expensive design may be to have an ADC per channel and digitally multiplex the data onto a common data bus :



Scaling Amplifier/Filter

ADC's operate over a fixed full-scale (FS) voltage range. Therefore to utilize the available resolution it becomes desirable to gain scale each channel to a level at least 50% of FS. The scaling can be performed automatically within the subsystem, or under remote control by the computer. The filtering operation is desirable to avoid aliasing errors (errors caused by the presence of significant energy in the data at frequencies greater than ½ the sampling frequency) and is often combined with the scaling amplifier. The important characteristics of the scaling amplifier are those typical of any amplifier - gain stability, settling time, maximum slew rate, and small-signal bandwidth.

Sample-Hold

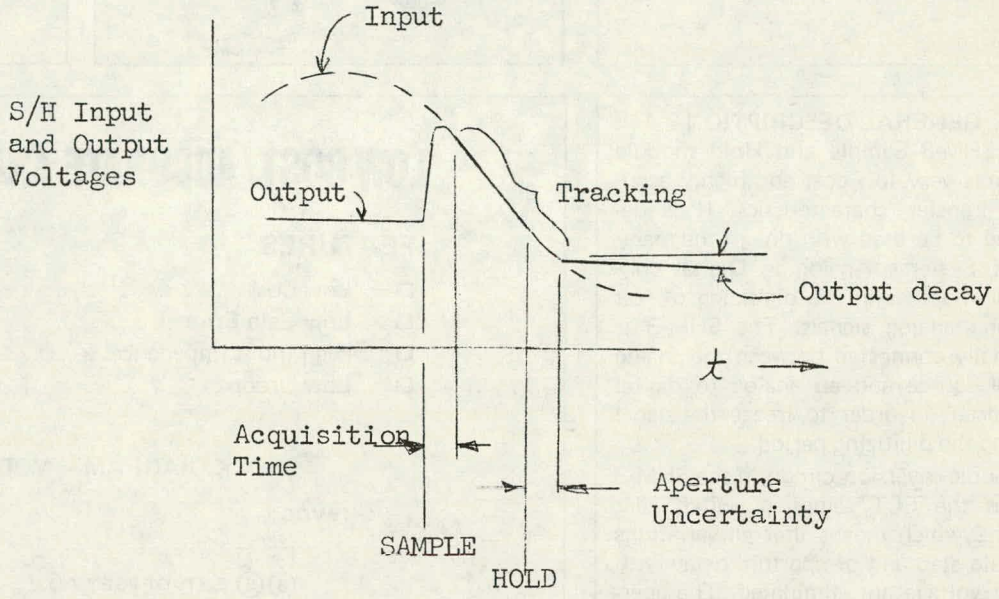
The sample-hold (S/H) module has the job of freezing fast-moving signals and to also hold the previous MUX output for conversion while the next channel is being selected. In addition to the general characteristics of an amplifier, the important S/H parameters are as follows.

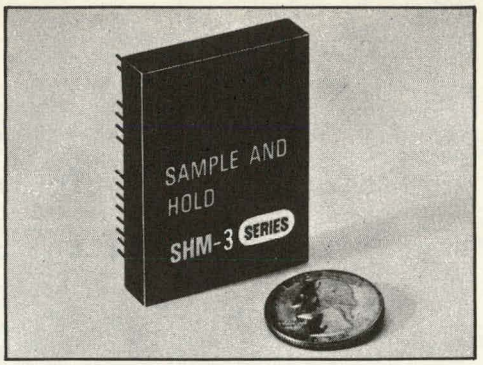
Acquistion time - The interval between the SAMPLE command and when the output begins to track the input within some stated accuracy.

Aperature uncertainty - The uncertainty in transition between SAMPLE and HOLD modes.

Decay rate - The maximum rate of change of the output voltage while in the HOLD mode.

Feedthrough Rejection - The ratio in db of a specific input signal to the resultant output during HOLD.





SAMPLE AND HOLD

MODEL SHM-3

GENERAL DESCRIPTION

The SHM-3 Sample and Hold module features very low cost and highly accurate transfer characteristics. It is designed to be used with one of the many Datel Systems Analog to Digital converter modules in the digitizing of fast moving analog signals. The SHM-3 is normally connected between the analog signal source and an analog to digital converter, in order to freeze the signal during the digitizing period.

A double inversion circuit in the SHM-3 places the FET sampling switch near ground which means that all variations of hold step and of aperture delay with input voltage are eliminated. The aperture delay of 40 nsec can then be compensated for, leaving an uncertainty or jitter of less than a nanosecond.

APPLICATION

When digitizing an analog signal which varies with time and having a frequency spectrum, it is difficult to determine what point of this signal is exactly represented by the resultant digital output. Since the maximum time "uncertainty" of the conversion is the total conversion time of the converter which may be called "aperture time or ambiguity time"; therefore, the maximum error due to this uncertainty is the difference of two points of the analog signal under measurement from T_0 to time, T_1 representing the time required to convert the changing analog signal.

A faster converter will obviously shorten the aperture and the error will be reduced proportionately, but a device such as the SHM-3 with very narrow aperture characteristics, controlled by command, is far more useful in trying to determine the exact point of the changing analog signal when converting. The purpose of SHM-3 is to "hold" upon command at the beginning of the conversion (T_0 time) the analog voltage applied at its input. The "held" value will remain constant during the conversion process.

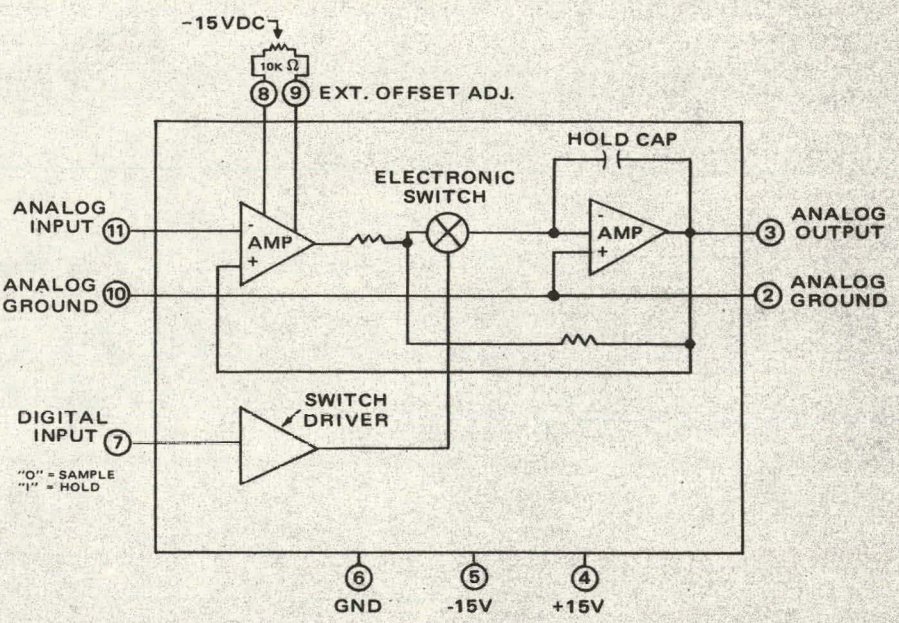
Relationships of error due to time uncertainty versus input frequency is plotted on the reverse side of this sheet.

LOW COST/ACCURATE ANALOG STORAGE

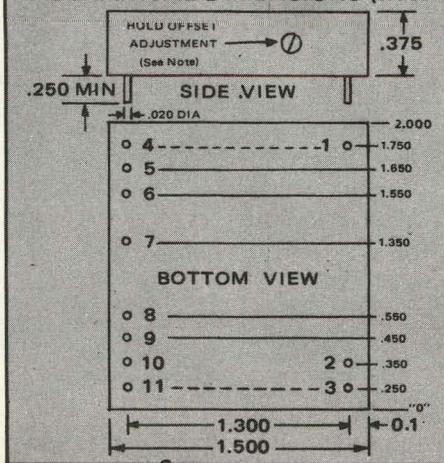
FEATURES

- ☐ Low Cost $39.00
- ☐ Low Gain Error . . . . . ± .005%
- ☐ High Input Impedance . . . . . 100 Megohm
- ☐ Low Droop . . . . . 20 μV/msec

BLOCK DIAGRAM – MODEL SHM-3



MECHANICAL DIMENSIONS (INCHES)



INPUT/OUTPUT CONNECTIONS

1	NO CONNECTION
2	ANALOG GROUND
3	ANALOG OUT
4	+15 VDC
5	-15 VDC
6	GROUND
7	HOLD COMMAND INPUT
8	OFFSET
9	OFFSET
10	ANALOG GROUND
11	ANALOG IN

Note: Adjust for hold offset at Vin = 0 Volts or with analog input at ground.

SPECIFICATIONS (Typical at 25°C, ±15V unless otherwise noted)

ELECTRICAL

Analog Input

Input Voltage Range	±10V
Max. Safe Input	±15V
Impedance	10^8 ohms in parallel with a 200 nA current source

Digital Inputs

(TTL, DTL, C/MOS Compatible)	Nominal	Limits
Sample	0V	0 to +.8V @ -1.5 mA
Hold	+4V	+2 to +15V @ +0.1 mA

Sample

Offset V, (Vout -Vin) @ Vin = 0	Ext. Adj. to 0
Offset V over temp. range (1)	±30 μ V/°C
Offset V vs Supply Voltage	±100 μ V/V
Gain Error (offset V vs Vin)	±.005%
Gain Error over temp. range	±.001%/70° C
Bandwidth, 3db, 20V .p-p (1)	10 KHz
Bandwidth, 3db, 1V .p-p (1)	100 KHz
Slew Rate (1)	0.5 V/μ sec
Settling Time, 20V step, to ±.005% (1)	50 μ sec
Noise, wideband, (100 KHz)	100 μ V rms

Sample to Hold

Hold Step (2)	Int. Adj. to 0
Peak Transient	400 mV
Aperture Delay	40 nsec
Aperture Jitter	1 nsec

Hold

Droop	20 μ V/msec
Droop vs Temp.	x2/10° C
Feedthrough attenuation (within 100 KHz amplifier bandwidth)	-80dB

Hold to Sample

Acquisition Time (1)	50 μ sec to ±.005% for a 20V step

Output

Max. Current	±5 mA
Impedance	.05 ohm
Short Circuit Protection	Output to GND indefinitely
Input Power	+15VDC @ 14 mA -15VDC @ 12 mA

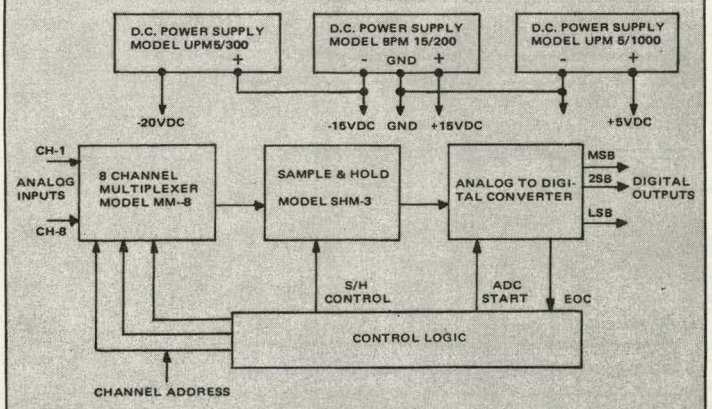
PHYSICAL - ENVIRONMENTAL

Operating Temperature Range	0° C to +70° C
Storage Temperature Range	-55° C to +85° C
Relative Humidity	Up to 100% non-condensing
Size	2"Lx 1.5"Wx .375"H plug-in module
Pins	.020" round gold plated .250" Long Min.
Case Material	Black diallyl phthalate, per MIL-M-14
Weight	3 oz.
Mating Socket	DILS-2 2 req'd per module, $4/Pr.
Price (1-9).	$39.00

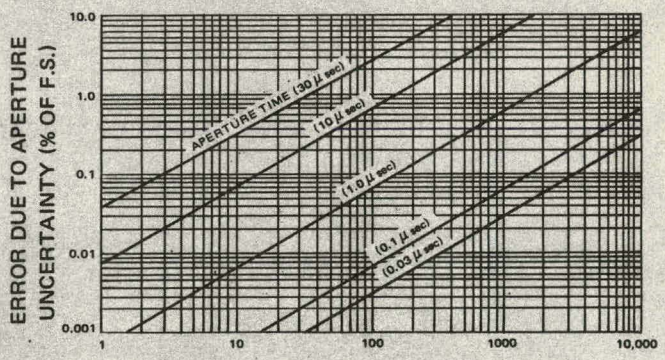
Model SHM-3 sample and hold module is fully encapsulated and features dual in-line pinning compatibility (i.e., 0.100" grid pin spacing and 1.3" between rows of pins), permitting direct plug-in to AUGAT, CAMBION, EECO, etc., circuit boards. Module is fully repairable.

NOTES: (1) Source resistance of 5K ohm or less
(2) Adjust for hold offset at Vin=0 Volts or with analog input at ground.

TYPICAL SYSTEM CONFIGURATION



ERROR DUE TO TIME UNCERTAINTY (APERTURE TIME) AS A FUNCTION OF INPUT SIGNAL FREQUENCY



TYPICAL EXAMPLE

ANALOG/DIGITAL CONV. W/WO SAMPLE & HOLD

FULL SCALE ACCURACY	WITH S & H (50 NANOSEC APERTURE)	WITHOUT S & H (20 μsec CONVERSION TIME)
0.1%	3KHz (MAX. INPUT FREQUENCY)	700Hz (MAX. INPUT FREQUENCY)
.01%	300Hz (MAX. INPUT FREQUENCY)	< 1Hz (MAX. INPUT FREQUENCY)

7. GENERAL PURPOSE COMPUTER HARDWARE

7.1 A Very Simple Computer

Up to now we've been studying the various components that are used in digital systems. By combining these components, we've been able to form various functional circuits or systems for measurement and/or data manipulation. We are now ready to begin the study of the ultimate digital system, the computer. It is important to keep in mind that the computer is simply a collection of hardware that is capable of performing only very mundane manipulations of data, but at very high speeds and with absolute accuracy.

In order to introduce some of the concepts involved in computers, consider the very simple computer (SC-1) illustrated on the following page. Although this computer has very limited capabilities, it embodies many of the principles involved in much larger machines:

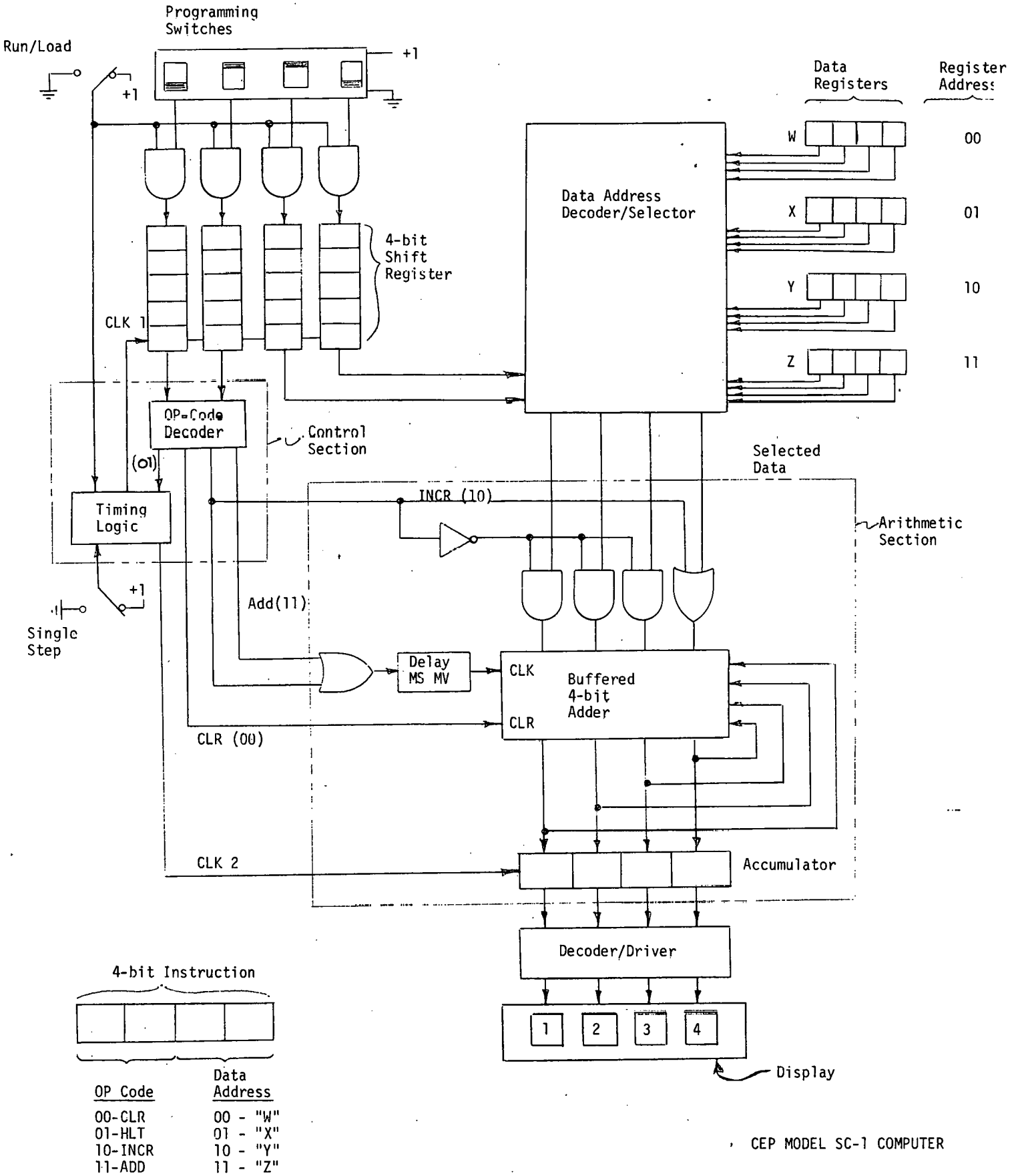
General purpose - Although only a few functions are possible, these functions can be executed in a perfectly arbitrary sequence to satisfy varying computational needs.

Stored program - The sequence of data manipulation to be performed and the locations of the data are "programmed" and "stored" in the hardware. In the SC-1, the program steps (instructions) are defined by a set of four switches and then loaded into a 4-bit-wide parallel-input/parallel-output circulating shift register.

Word size - The number of bits in the instruction word (e.g., four for the SC-1).

Instruction format - The 4-bit instruction word is formatted into a 2-bit group for definition of the function, and a 2-bit group which defines the location (address) of the data variable.

OP Code - The information contained in the 2-bit group of the instruction word which defines the function to be performed is termed the operation code (OP code). The available OP codes for the SC-1 are:


Programming Switches
Run/Load
+1
+1
Data Registers
Register Address
W
00
X
01
Y
10
Z
11
Data Address Decoder/Selector
4-bit Shift Register
CLK 1
OP-Code Decoder
Control Section
(01)
Timing Logic
INCR (10)
Selected Data
Arithmetic Section
+1
Single Step
Add(11)
Delay MS MV
CLK
CLR
Buffered 4-bit Adder
CLR (00)
CLK 2
Accumulator
Decoder/Driver
1
2
3
4
Display
4-bit Instruction
OP Code
Data Address
00-CLR
01-HLT
10-INCR
11-ADD
00 - "W"
01 - "X"
10 - "Y"
11 - "Z"
CEP MODEL SC-1 COMPUTER

OP Code	Function
00	Clear accumulator
01	Halt
10	Increment accumulator
11	Add selected data word to accumulator

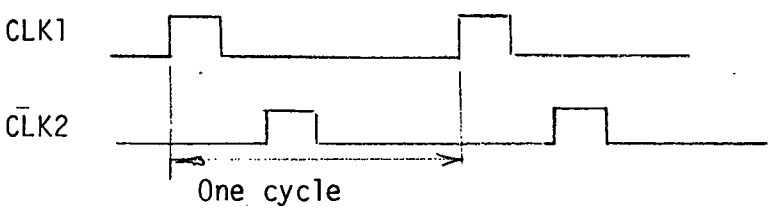
Addressing - The rightmost 2-bits of the instruction word contains the address (when appropriate) of the data register which contains the desired operand. In the SC-1, the programmer can select from four data registers which must be manually loaded (say from thumb-wheel switches):

Operand Address	Data Register
00	W
01	X
10	Y
11	Z

Control Section - That portion of the hardware which sequences the retrieval of the instruction, its decoding, and ultimately its execution.

Arithmetic Section - That portion of the hardware which performs the data manipulation.

Computer Cycle - The time period and sequence of events over which an instruction is retrieved, decoded, executed, and the results strobed into the accumulator. In the SC-1, two-phase clocking, as represented by CLK1 and CLK2, is used.



The CLK1 pulse causes a new instruction word to be shifted into the OP code decoder and the operation to be executed, whereas CLK2 strobes the results into the accumulator.

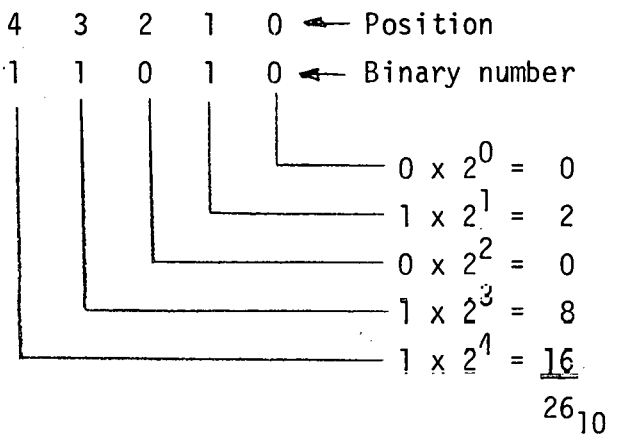
7.2 Data Representation

7.2.1 Number System Conversions

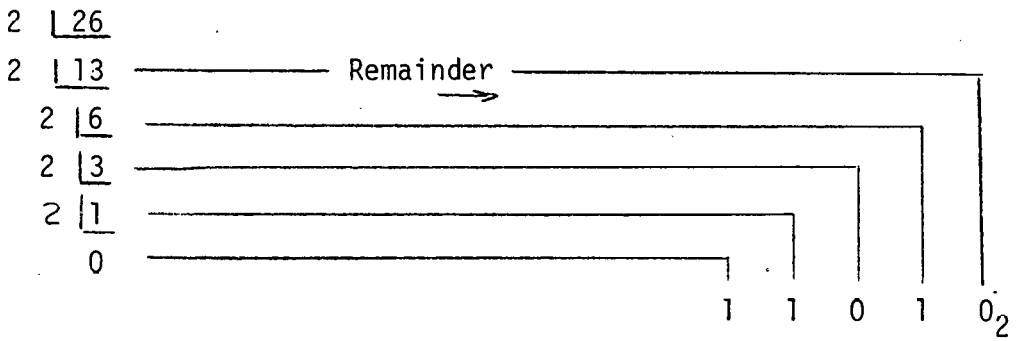
In our study of digital logic we have already introduced the binary, octal, and hexadecimal number systems and compared them to the decimal

system. Previously we were concerned with converting the binary or octal number to its base 10 equivalent. In computer programming, we must also convert in the opposite direction:

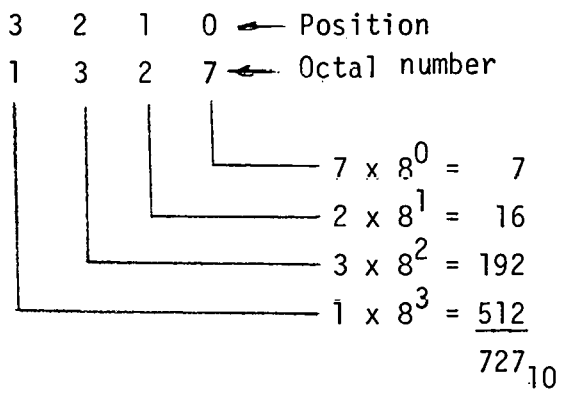
Binary to decimal



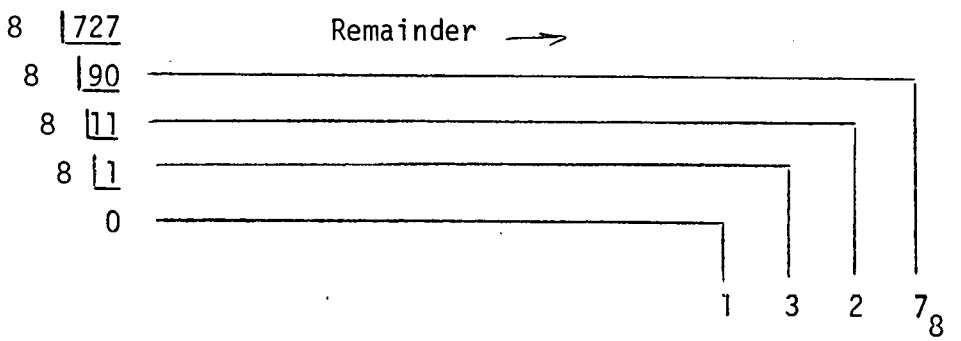
Decimal to binary (division method)



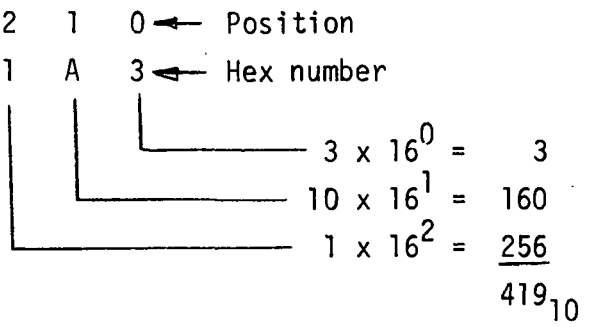
Octal to decimal



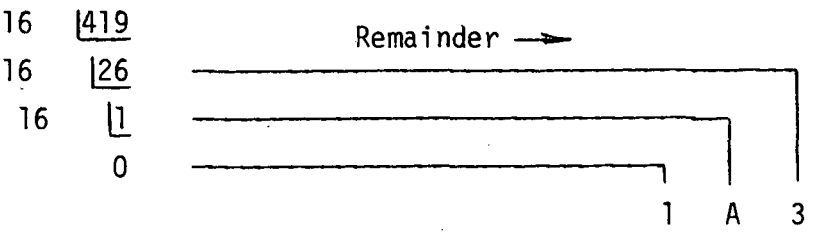
Decimal to octal



Hex to Decimal



Decimal to Hex



7.2.2 Binary Arithmetic

Binary addition follows the same rules that we are accustomed to in decimal arithmetic:

$$\begin{array}{r} 1\ 0\ 1\ \ 0\ 0\ 1 \\ +\ 1\ 1\ 0\ \ 0\ 1\ 1 \\ \hline 1\ 0\ 1\ 1\ \ 1\ 0\ 0_2 \end{array}$$

Binary subtraction is accomplished by adding the first number to the negative of the second. The negative form is the two's complement of the positive binary number. By definition, the two's complement is that number which when added to the original number will result in a sum of unity. The two's complement of a positive binary number is obtained by adding one to its one's complement. To subtract 3_{10} from 5_{10}

3_{10} = 0 1 1

one's complement of 3_{10} = 1 0 0 (just invert bits)

two's complement of 3_{10} = 1 0 1

$$\begin{array}{rl} \therefore\ 5_{10} = & 1\ 0\ 1 \\ -3_{10} = & 1\ 0\ 1 \\ \hline & 1\ 0\ 1\ 0 \end{array}$$

Ignore → 2_{10}

Binary multiplication again follows the same rules as decimal multiplication:

$$
\begin{array}{rl}
4_{10} = & 1\,0\,0 \\
\times\ 5_{10} & \times\ 1\,0\,1 \\
\hline
 & 1\,0\,0 \\
 & 0\,0\,0 \\
 & 1\,0\,0 \\
\hline
20_{10} = & 1\,0\,1\,0\,0_2
\end{array}
$$

It should be noted that the above operation amounts to nothing more than shifting and adding. Later on we will write a computer program to do this.

Binary division is also similar to decimal division:

$$20_{10} \div 5_{10} =$$

$$
\begin{array}{r}
1\,0\,0. \\
1\,0\,1\ \overline{)\,1\,0\,1\,0\,0.} \\
\underline{1\,0\,1} \\
0\,0\,0
\end{array}
= 4_{10}
$$

7.2.3 Data Formats

The representation of data internal to a computer can be broadly divided into fixed-point numbers, floating-point numbers, or alphanumeric data.

Fixed point

For fixed point numbers, straight binary code is used for positive numbers and the two's complement form is used for negative numbers. Note that the MSB becomes a "sign" bit and is really lost as a means of representing magnitude

$$
\begin{array}{rl}
2_{10} = & 0\,0\,0\quad 0\,0\,0\quad 0\,0\,0\quad 0\,1\,0 \\
1_{10} = & 0\,0\,0\quad 0\,0\,0\quad 0\,0\,0\quad 0\,0\,1 \\
0 = & 0\,0\,0\quad 0\,0\,0\quad 0\,0\,0\quad 0\,0\,0 \\
-1_{10} = & 1\,1\,1\quad 1\,1\,1\quad 1\,1\,1\quad 1\,1\,1 \\
-2_{10} = & 1\,1\,1\quad 1\,1\,1\quad 1\,1\,1\quad 1\,1\,0
\end{array}
$$

Floating point

Fixed point numbers are limited in their dynamic range, i.e., for an N-bit data word, the full range of numbers is

$$-2^{N-1} \longrightarrow 0 \longrightarrow 2^{N-1}-1$$

Therefore very large or small numbers are represented by a binary exponential representation similar to scientific notation. This representation is referred to as "floating-point" and consists of a mantissa (fractional part) and an exponent to which the base is raised:

Scientific: $.1100 \times 10^2$

Binary floating point: $.1011 \times 2^4$

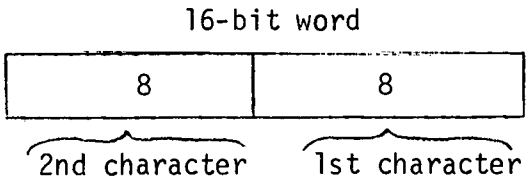
$= 11_{10}$

A floating point data word is therefore divided up into a mantissa and an exponent. It should be noted that both parts can be either positive or negative (two's complement used),

so a sign bit is needed for each.

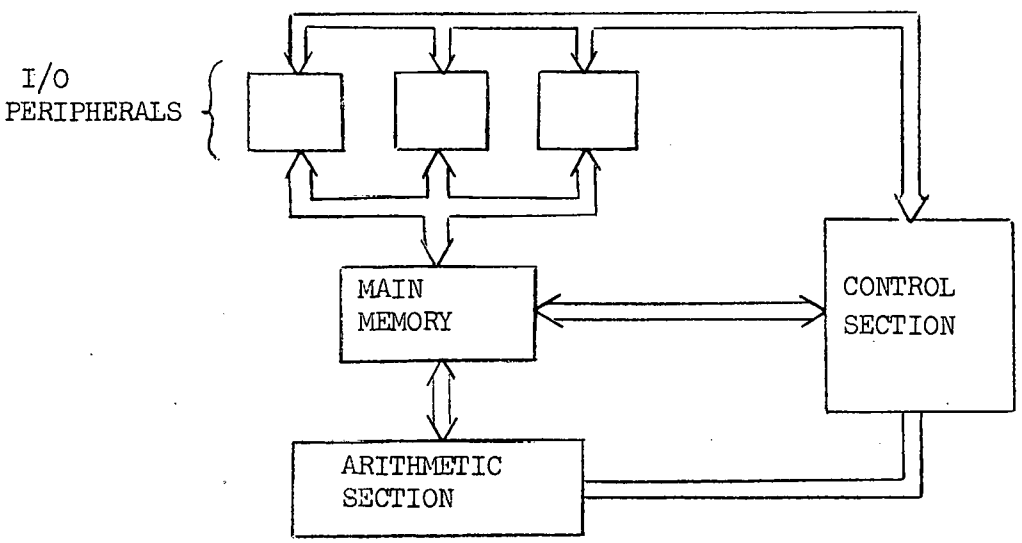
Alphanumeric

Often it is necessary to represent characters such as A-Z, 0-9, or punctuation marks. These must also be represented in the computer by a binary code. Most commonly, the codes used are ASCII (USA Standard Code for Information Interhcange) or BCD. The ASCII codes are shown on the following page. It should be noted that only 6 bits (6-bit code) or 8 bits (8-bit code) are required for each character; therefore more than one character can be packed in a single word.



7.3 Computer System Organization

The following block diagram illustrates the fundamental organization (architecture) of a digital computer. The basic functions that must be performed are data input and output (I/O), data and instruction storage, and data manipulation. The performance of these functions is regulated by the control section.



ASCII* Character Set (5)

Character	8-Bit Octal	6-Bit Octal	Character	8-Bit Octal	6-Bit Octal
A	301	01	!	241	41
B	302	02	"	242	42
C	303	03	#	243	43
D	304	04	$	244	44
E	305	05	%	245	45
F	306	06	&	246	46
G	307	07	'	247	47
H	310	10	(	250	50
I	311	11	)	251	51
J	312	12	*	252	52
K	313	13	+	253	53
L	314	14	,	254	54
M	315	15	-	255	55
N	316	16	.	256	56
O	317	17	/	257	57
P	320	20	:	272	72
Q	321	21	;	273	73
R	322	22	<	274	74
S	323	23	=	275	75
T	324	24	>	276	76
U	325	25	?	277	77
V	326	26	@	300	
W	327	27	[	333	33
X	330	30	\	334	34
Y	331	31	]	335	35
Z	332	32	↑	336	36
0	260	60	←	337	37
1	261	61	Leader/Trailer	200	
2	262	62	LINE FEED	212	
3	263	63	Carriage RETURN	215	
4	264	64	SPACE	240	40
5	265	65	RUBOUT	377	
6	266	66	Blank	000	
7	267	67	BELL	207	
8	270	70	TAB	211	
9	271	71	FORM	214	

* An abbreviation for USA Standard Code for Information Interchange.

Binary Coded Decimal Format (10)

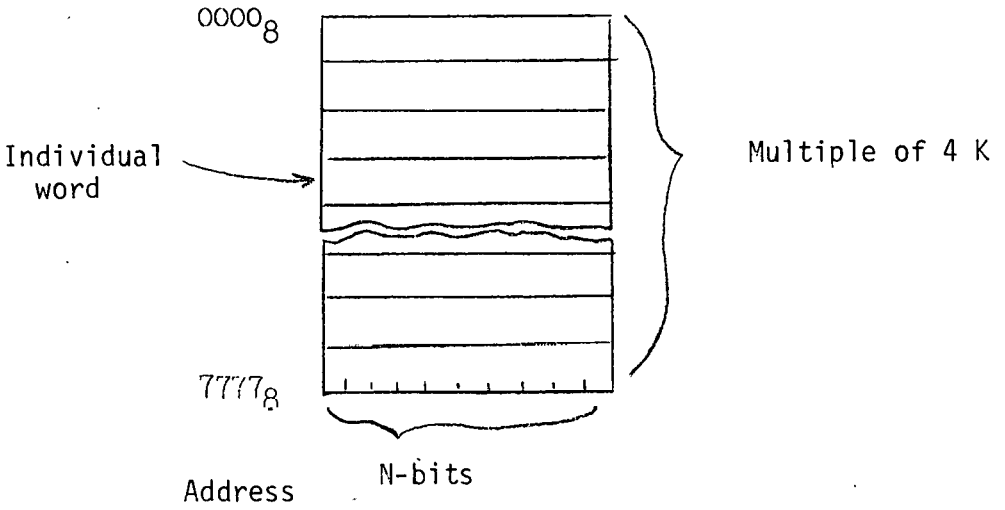
Kennedy 1406 1506 ASCII-BCD Conversion

Symbol[a]	BCD (octal code)	ASCII Equivalent (octal code)	Symbol	BCD (octal code)	ASCII Equivalent (octal code)
(Space)	20	Ø4Ø	A	61	1Ø1
!	52	Ø41	B	62	1Ø2
#	13	Ø43	C	63	1Ø3
$	53	Ø44	D	64	1Ø4
%	34	Ø45	E	65	1Ø5
&	6Ø	Ø46	F	66	1Ø6
'	14	Ø47	G	67	1Ø7
(	34	Ø50	H	7Ø	11Ø
)	74	Ø51	I	71	111
*	54	Ø52	J	41	112
+	6Ø	Ø53	K	42	113
,	33	Ø54	L	43	114
-	4Ø	Ø55	M	44	115
.	73	Ø56	N	45	116
/	21	Ø57	O	46	117
			P	47	12Ø
Ø	12	Ø6Ø	Q	50	121
1	Ø1	Ø61	R	51	122
2	Ø2	Ø62	S	22	123
3	Ø3	Ø63	T	23	124
4	Ø4	Ø64	U	24	125
5	Ø5	Ø65	V	25	126
6	Ø6	Ø66	W	26	127
7	Ø7	Ø67	X	27	13Ø
8	1Ø	Ø7Ø	Y	30	131
9	11	Ø71	Z	31	132
:	15	Ø72	[	75	133
;	56	Ø73	\	36	134
<	76	Ø74	]	55	135
=	13	Ø75			
>	16	Ø76			
?	72	Ø77			
@	14	1ØØ			

From "A Pocket Guide to the Hewlett-Packard Computers," Hewlett-Packard Company, Palo Alto, Cal., 1970.

[a] Other symbols which may be represented in ASCII are converted to spaces in BCD (20)

The main memory of a computer is organized as a sequence of N-bit words, usually some multiple of 4 K (K = 2^{10} = 1024) words in length. Each word has a binary location or address referred to by its octal or hex equivalent.



The actual storage medium is magnetic core or semiconductor arrays. The time required to locate and retrieve a single word is termed the memory access time. Any location can be accessed at random.

I/O Peripherals

The computer makes use of many different peripheral devices for input, output, and storage. In summary, these devices are as follows:

Input	Output	Storage
Teletype	Teletype	Drum memory
Paper tape reader	Paper tape punch	Disc memory
Card reader	Card punch	Magnetic tape
Magnetic tape	Magnetic tape	
	Printer	
	CRT	

Common computer peripherals will be discussed further in section 11.0.

8.0 CENTRAL PROCESSOR

8.1 Hardware Configuration[2]

The figure on the following page represents a fairly close representation of the hardware configuration of a real 12-bit computer (i.e., DEC PDP-8). The items of note at this point are:

Memory - Both instruction and data words are stored in memory. Storage locations are numbered in sequence using octal numbers. The octal number corresponding to a given location (a 12-bit word in this case) is referred to as its address.

Memory Data Register (MD) - Any data word retrieved from memory passes through the MD on its way out.

Memory Address Register (MA) - This register contains the location of the memory word being retrieved.

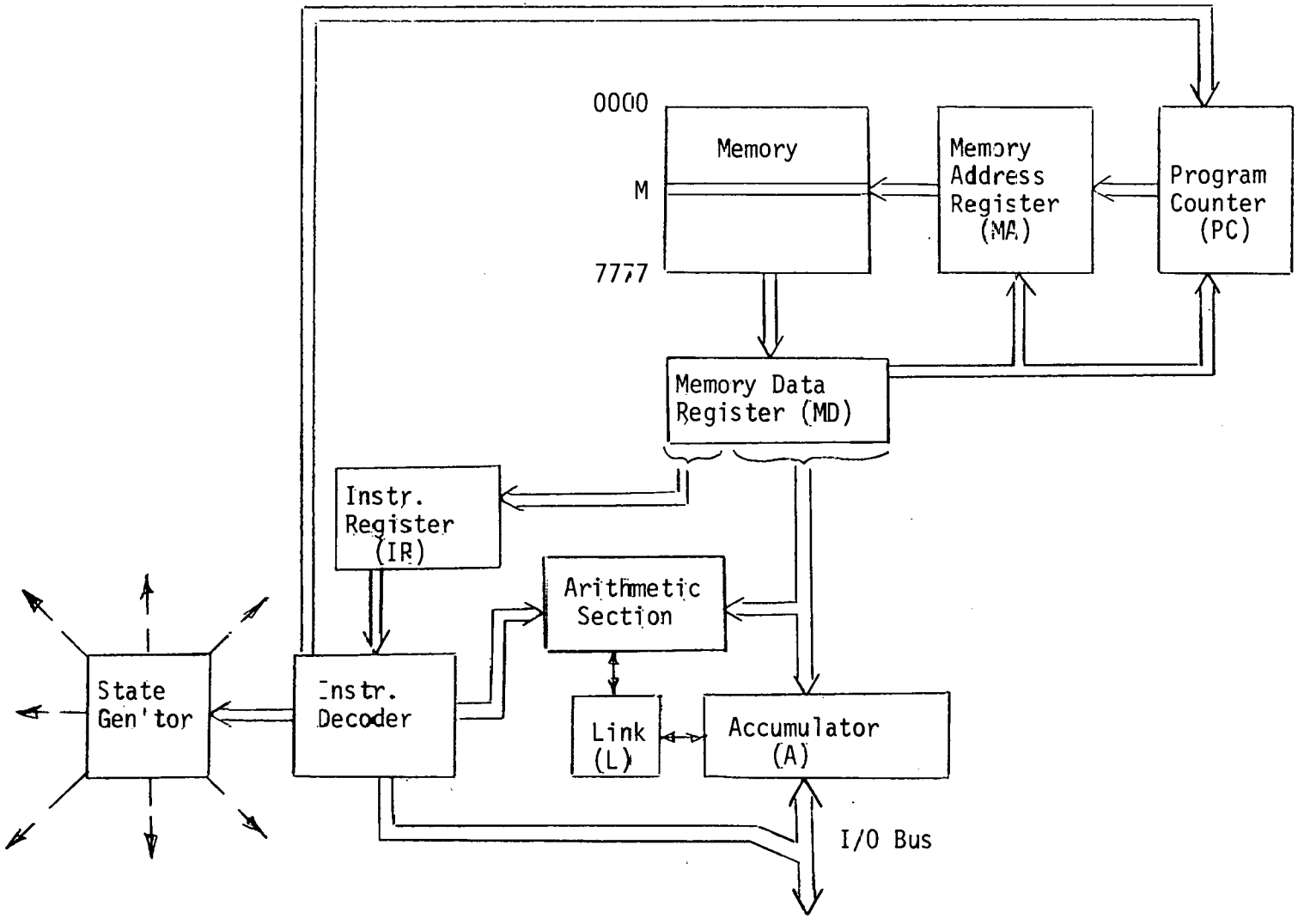
Program Counter (PC) - This register contains the memory address of the next instruction to be executed. Since instructions are generally executed in sequence as they are located in memory, the PC is automatically incremented after each instruction is executed.

Instruction Register (IR) - The 3 MSB's of an instruction word contain the OP CODE. Since the MD may be used again during the execution of the instruction, it is necessary to save the OP CODE in the IR.

Accumulator (A) - The A is a storage register which contains the results of all arithmetic and logical operations performed in the arithmetic section. All programmed I/O to peripheral devices also pass through the A.

Link (L) - The L is a one-bit register which operates with the arithmetic section to act as an extension to the A. An overflow from the A complements the Link.

Instruction Decoder - This device decodes the OP CODE stored in the IR and enables the appropriate gates throughout the system so as to perform the desired function.



TYPICAL CENTRAL
PROCESSOR CONFIGURATION

State Generator - This device generates the proper clock pulses required by a particular instruction for its execution.

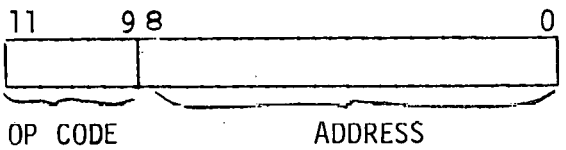
Peripherals - Any device outside the main computer hardware is referred to as a peripheral. The usual means of communication is over the I/O bus.

8.2 Coding Basics

8.2.1 Types of Machine-level Coding

The sequence of computer instructions required to perform a specific task is referred to as a program. The only form of information which the computer can understand is pure binary. Since this is rather cumbersome to the programmer, various coding schemes are in use for coding machine instructions:

Binary Code - In a typical 12 bit instruction word the 3 MSB's contain the OP CODE and the 9 LSB's contain the memory address of the word containing the operand



As an example, the OP CODE for the two's complement add operation (ADD) is 001, and the instruction word contained in memory location 5_{10} required to add the contents of location 9_{10} to the AC is

Address				Contents			
000	000	000	101	001	000	001	001

Octal Coding - Since the 1's and 0's of binary coding are extremely cumbersome, octal coding is often used to represent the same information

Address	Contents
0005	1011

It should be noted, however, that a program coded in octal must still be reduced to the full binary code equivalent before it can be executed.

Mnemonic Coding - Although octal coding is certainly an improvement, it still requires the programmer to memorize the relatively abstract octal equivalent of each OP CODE. In mnemonic coding the OP CODE is represented by some easily remembered symbolic name, while the address remains octal. For instance, the programmer would write the above

instruction as

ADD 11

Again, it must be noted that the mnemonic coding has to be expanded to its binary equivalent before it can be executed by the computer. Virtually every computer has available a special program for doing this called an "Assembler." The assembler treats the mnemonic coding as data and generates as output the executable binary equivalent. The Assembler itself is written in binary code.

8.2.2 Flow Charting

As programs become more complex, it becomes desirable to represent pictorially the sequence of operations and decisions to be performed. The chart on the following page defines the set of flow charting symbol conventions which we will use. As an example, the flow chart shown represents the coding necessary to calculate the absolute difference between five pairs of numbers and type out the results.

8.3 Instruction Set (2)

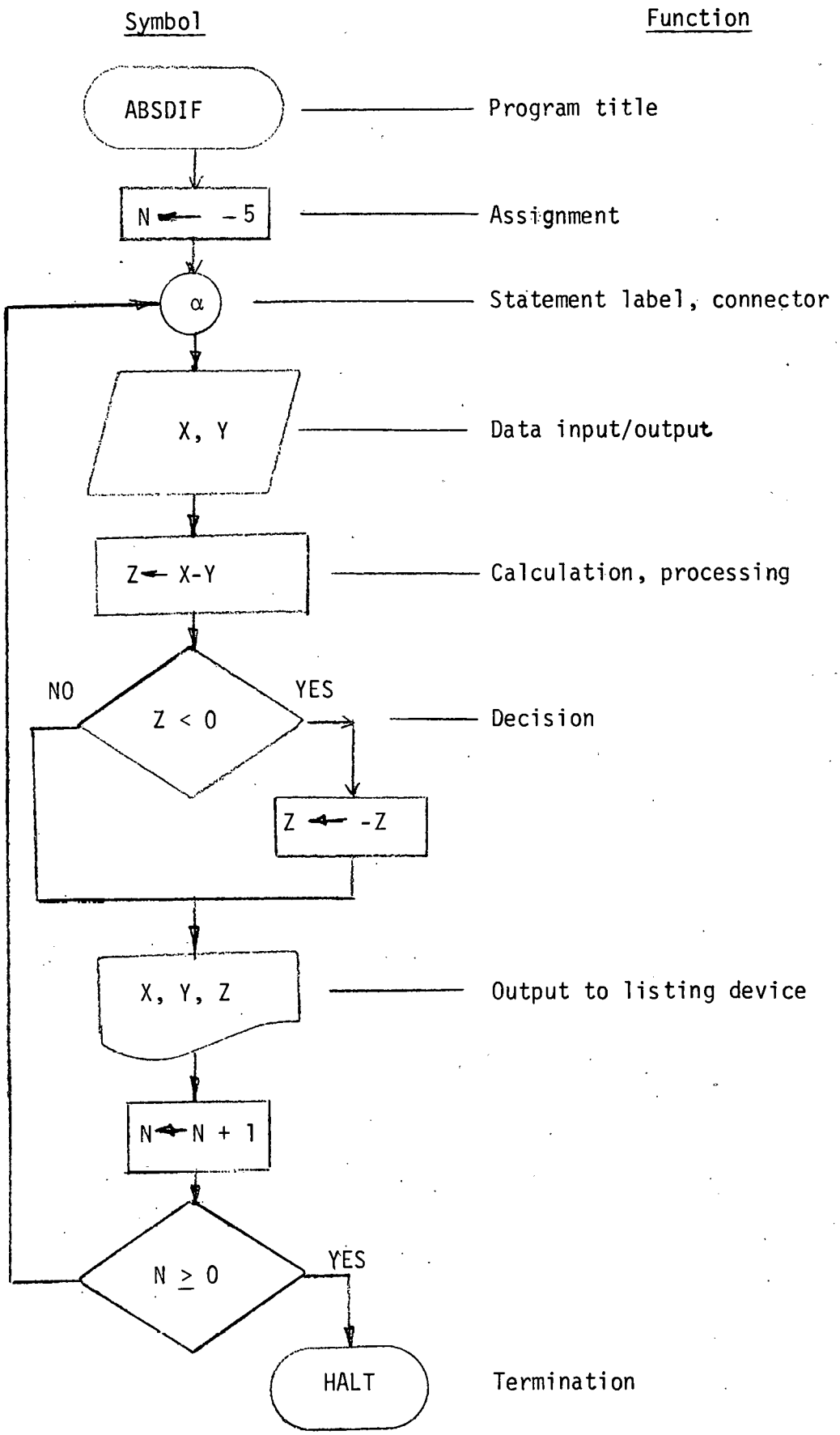
The collection of all possible operations that a computer can perform is called its instruction set.

8.3.1 Notation

In order to conveniently define the instruction set of our hypothetical computer, we rely upon the following notation

Symbol	Meaning
A	Accumulator
L	Link
PC	Program counter
M	Address portion of an instruction
I	Address from which instruction is obtained
(XXX)	Contents of location (XXX)
←	"Becomes" or "is assigned the value"
$\wedge$	Boolean AND
$\overline{\ }$	Boolean NOT (complement)
$\Rightarrow$	Designates the sequence of instructions.
→	"Is copied into"

FLOW CHART SYMBOL CONVENTIONS



8.3.2 Basic Instruction Set[2]

Computer instructions can be classified according to the part of the computer hardware which is addressed, or as to the type of function performed.

Classified by hardware addressed

- memory reference
- register reference
- input/output reference

Classified by function

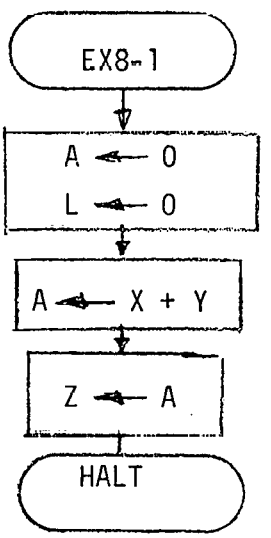
- Move data, arithmetic or logical instructions
- control instructions
- subroutine linkage
- operate instructions
- input-output instructions

From the programmer's viewpoint, the functional classification is the most useful and the following chart defines the instruction set of our hypothetical computer using these groupings.

Example 8-1

Write a program which will add two positive numbers X (= 2234_8) and Y(= 1111_8) to get Z

Flow chart -



A BASIC INSTRUCTION SET(2)

Type	Mnemonic	OP Code	Description	In Brief
Move data	STORE M	3xxx	Store (A) into M; A unchanged	(A) → M
Arithmetic	ADD M	1xxx	Add (M) to A; result in A	(M) + (A) → A
Logical	AND M	0xxx	"And" (M) to A; result in A	(M) ∧ (A) → A
	HALT	7402	Halt program	
	JUMP M	5xxx	Jump, unconditionally to M	⇒ M
	SKIPP	7510	Skip on positive accumulator	(A) = + ⇒ I + 2; (A) ≠ + ⇒ I + 1
	SKIPZ	7440	Skip on zero accumulator	(A) = 0 ⇒ I + 2; (A) ≠ 0 ⇒ I + 1
Control	SKIPN	7550	Skip on negative accumulator	(A) = - ⇒ I + 2; (A) ≠ - ⇒ I + 1
	SKIPL	7420	Skip if link is set to one	(L) = 1 ⇒ I + 2; (L) = 0 ⇒ I + 1
	SKIP	7410	Skip, unconditionally	⇒ I + 2
	ISZ M	2xxx	Increment (M); skip if result zero	(M) + 1 → M (M) = 0 ⇒ I + 2; (M) ≠ 0 ⇒ I + 1
Subroutine linkage	JMS M	4xxx	Jump to subroutine, save PC	(PC) → M; ⇒ M + 1
	CLEAR	7600	Clear A	0 → A
	CLEARL	7100	Clear link	0 → L
	COMPL	7040	Complement A	$(\bar{A})$ → A
	COMPLL	7020	Complement L	$(\bar{L})$ → L
Register	INCR	7001	Increment A by 1	(A) + 1 → A
Reference	ROTL	7004	Rotate A with L left 1 bit	
	ROTR	7010	Rotate A with L right 1 bit	
	CLEAR02	6XX0	Clear flag in the device (XX)	0 → FXX
Input	SKIP02	6XX1	Skip if flag set in the device (XX)	(FXX) = 1 ⇒ I + 2; (FXX) = 0 ⇒ I + 1
Output	READ02	6XX4	Read the data from the device (XX) into A	(XX) → A
	WRITE02	6XX5	Write the data from A into the device (XX)	(A) → XX
	GO02	6XX6	Activates an output of the selector (XX)	
	ION	6001	Enables interrupt flip-flop	

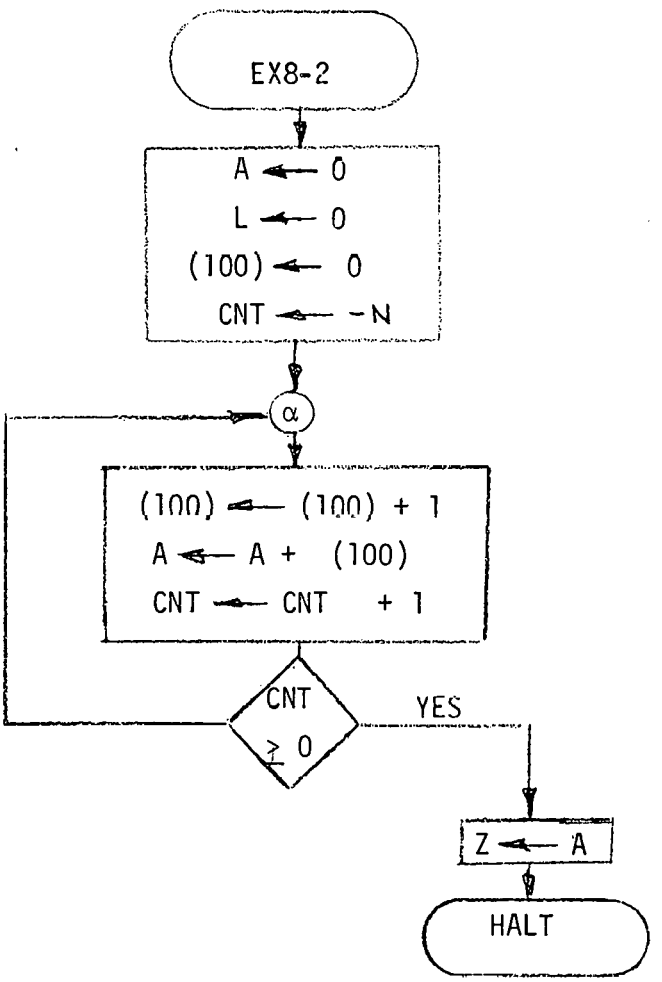
Coding -

Location$_8$	Contents Octal	Mnemonic	Comment
0000	7600	CLEAR	Clear A
0001	7100	CLEARL	and L
0002	1100	ADD 100	
0003	1101	ADD 101	A = X + Y
0004	3200	STORE 200	Z = A
0005	7404	ROTL	
0006	7600	CLEAR	Store MSB of
0007	7404	ROTL	A in LOC 201
0010	3201	STORE 201	
0011	7402	HALT	
0100	2234		X
0101	1111		Y
0200	-		Z
0201	-		ERR

Example 8-2

Write a program to sum up the first N digits of the arithmetic series 1, 2, 3, ..., N

Flow chart -



Coding -

Location$_8$	Contents Octal	Contents Mnemonic	Comment
0000	7600	CLEAR	Clear A, L, and
0001	7100	CLEARL	LOC 100
0002	3100	STORE 100	
0003	1101	ADD 101	Get N from
0004	7040	COMPL	LOC 101, take
0005	7001	INCR	2's complement and
0006	3102	STORE 102	store in LOC 102
0007	2100	ISZ 100	Increment (100) to get next term in series.
0010	1100	ADD 100	Add term to A
0011	2102	ISZ 102	Bump index and skip if zero
0012	5007	JUMP .-3	Jump back 3 locations
0013	3103	STORE 103	Put sum in LOC 103
0014	7004	ROTL	
0015	7600	CLEAR	Put MSB in
0016	7004	ROTL	LOC 104
0017	3104	STORE 104	
0020	7402	HALT	
0100	--		Term
0101	0005		N
0102	--		CNT
0103	--		Z
0104	--		ERR

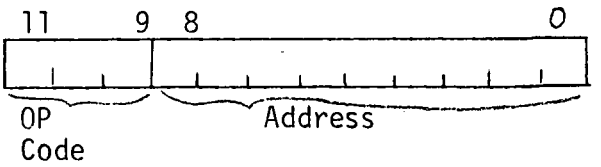
8.4 Addressing

8.4.1 Paging, Direct and Indirect Addressing [2, 5]

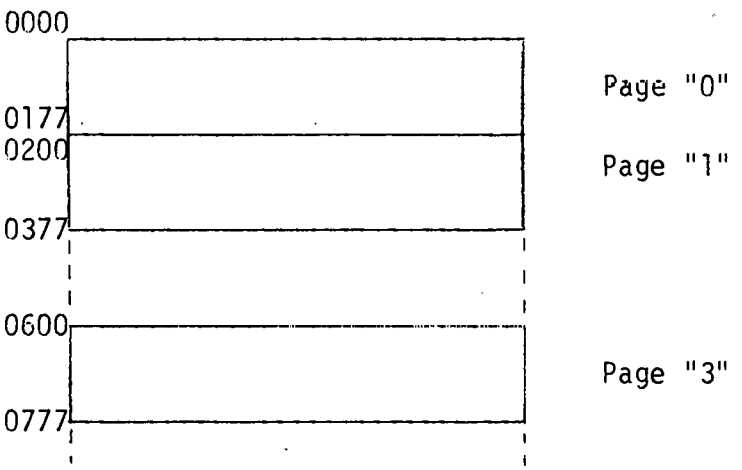
For a memory reference instruction such as

ADD M

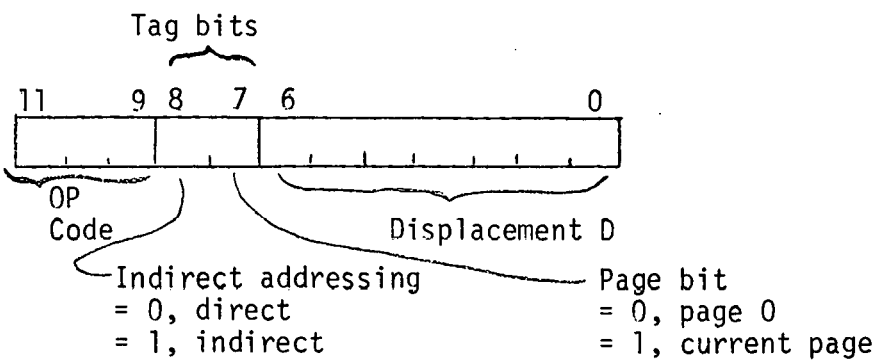
the 9 LSB's contain the address of the operand



But a 9-bit binary address would limit the size of any program to only 512 locations, and many programs are thousands of words long. To get around this limitation, a scheme often used on 12-bit or smaller machines is the "paged" memory. The actual memory is <u>not</u> physically divided up into pages--this is just a technique used in addressing the memory. The first two bits of the



9-bit address are referred to as the tag bits and designate how the paged memory is to be referenced, and the remaining 7 bits are called the displacement D.



Tag bits = 00

The effective address (actual address of the operand) is word D on page 0.

Tag bits = 01

The effective address is word D on the current page (i.e., page on which the instruction is located).

Location	Contents	Mnemonic	Comment
0050	0101	--	--
200	1050	Add 50	A ← 0101 + A
400	1250	Add 450	A ← 0102 + A
450	0102	--	
600	1250	Add 650	A ← 0103 + A
650	0103	--	

Tag bits = 10

The effective address is the contents of word D on page 0. Note that this is now a 12-bit address, so the first 4096 words of memory can be addressed indirectly.

Tag bits = 11

The effective address is the contents of word D on the current page. Again the effective address is 12 bits, so the first 4096 words can be addressed in this manner.

When using the symbolic assembler, the programmer indicates indirect addressing by the symbol I. For instance, the instruction

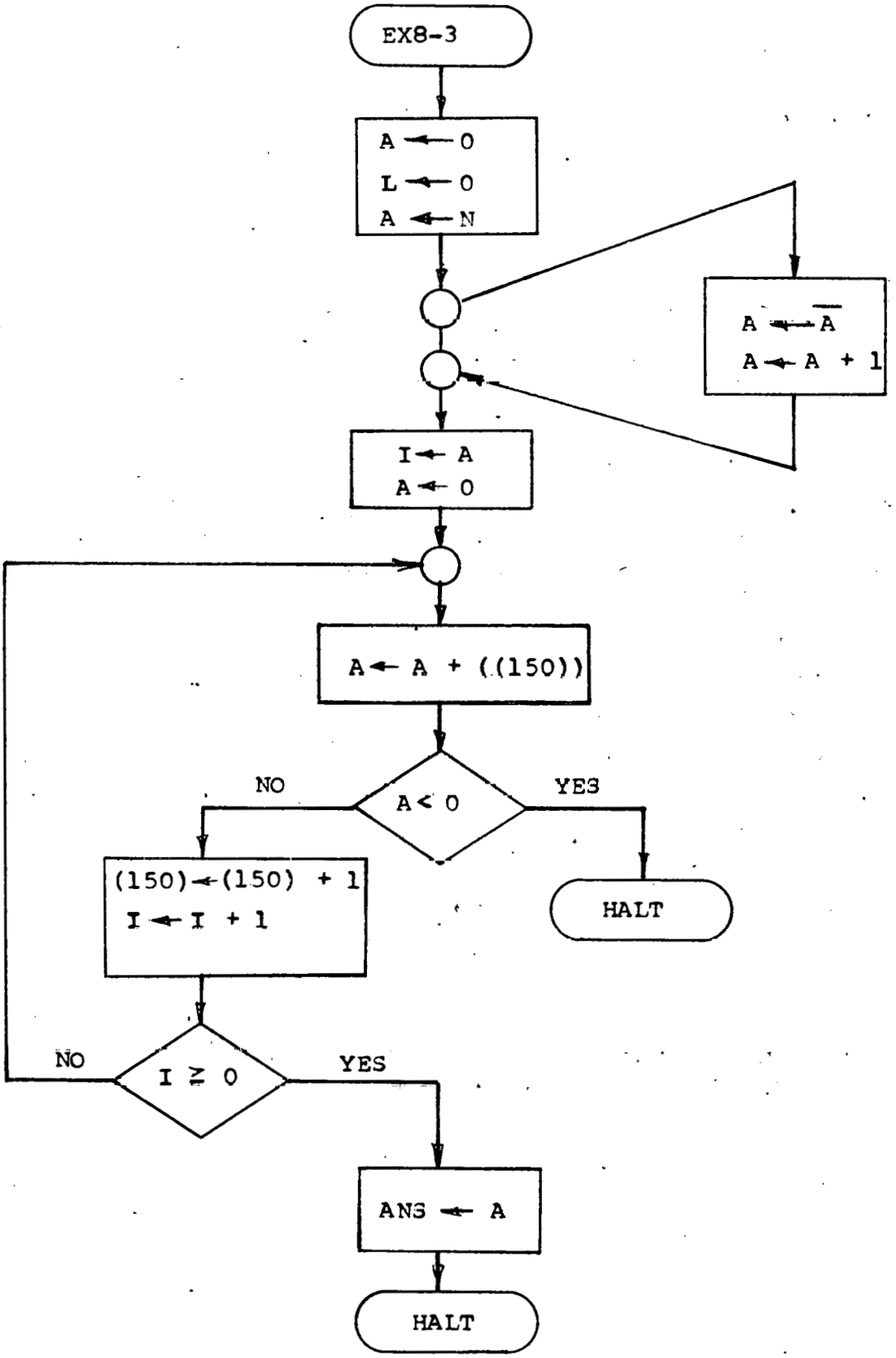
ADD I 100

would add to the A the contents of the address contained in location 100.

Ex 8-3

Write a program to add up a list of N data values which are stored in sequential memory locations. The address of the first data value is contained in location 150, and the number of terms is contained in 151. Store the results in 153.

FLOW CHART


EX8-3
A ← 0
L ← 0
A ← N
A ← $\overline{A}$
A ← A + 1
I ← A
A ← 0
A ← A + ((150))
NO
A < 0
YES
(150) ← (150) + 1
I ← I + 1
HALT
NO
I ≥ 0
YES
ANS ← A
HALT

Coding

	Location	Mnemonic	Comment
MAIN	100	CLEAR	Clear L and A
	101	CLEARL	
	102	ADD 151	Load number to be complemented
	103	JMS 140	and jump to subroutine
	104	STORE 152	Put I into 152
	105	CLEAR	Clear A
	106	ADD I 150	Start of loop--add term
	107	SKIPN	Check for error
	110	JUMP .+2	
	111	HALT	
	112	ISZ 150	Increment pointer
	113	ISZ 152	Increment I and
	114	JUMP .-6	check for 0
	115	STORE 153	
	116	HALT	
SUB	140	- - -	Return address stored here
	141	COMPL	Take two's complement
	142	INCR	
	143	JUMP I 140	Return to main program
MISC.	150	0601	Address of data
	151	0005	N
	152	--	I
	153	--	ANS
DATA	600	--	1st value
	601	--	2nd value
	602	--	etc.

8.4.2 Symbolic Addressing [2, 5]

The memory reference instruction as we have used it thus far has always involved an octal address. More sophisticated assemblers permit symbolic addresses whereby the programmer makes up names for the various locations and

the computer keeps track of these during the assembly process by building up a dictionary of symbolic addresses. The final object code output by the assembler contains the correct address.

Ex 8-4

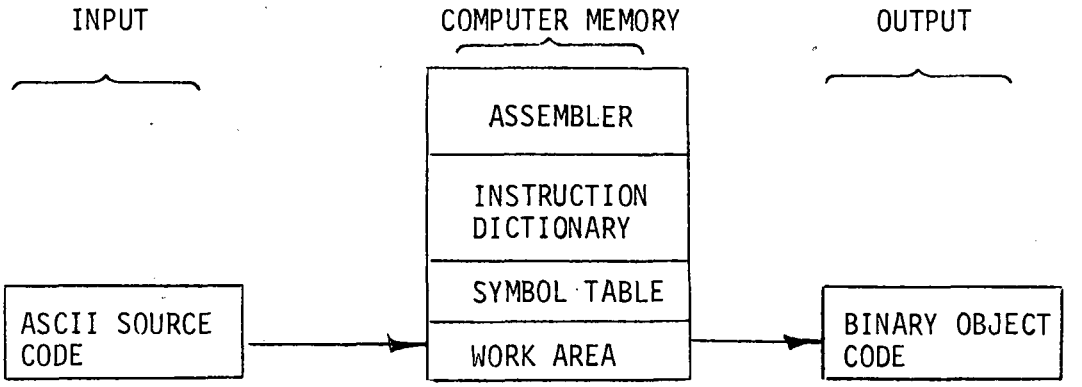
Using symbolic addressing, write a program to add two numbers, X = 10_8 and Y = 17_8 to get Z

		Object Code	
	Source Code	Location	Contents
	CLEAR	0	7600
	CLEARL	1	7100
	ADD X	2	1006
	ADD Y	3	1007
	STORE RES	4	3008
	HALT	5	7402
X	10	6	0010
Y	17	7	0017
RES		8	— —

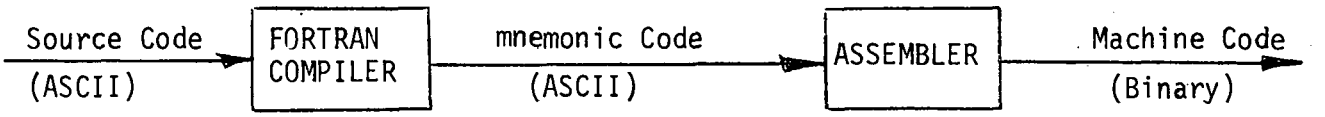
9. PROGRAMMING FUNDAMENTALS

9.1 Assembly Language as FORTRAN[2, 11]

The programming which we have done so far can be described as machine-level coding, i.e., each statement of source code could be translated into a single machine instruction. mnemonic coding is a form of machine level coding, but cannot be executed directly by the computer. A special binary coded program called an assembler does the translation by treating the ASCII source code as data input and generating the proper binary machine instructions as output.



Higher level languages are characterized by very powerful statements that result in several machine instructions for each statement. The program that does the translation is called a compiler. Various languages, such as COBOL, ALGOL, and FORTRAN, are in use throughout the world. In general, compilers generate ASCII mnemonic coding as their output, which must then be processed in a second step by the ASSEMBLER program.



A summary of the available executable statements in FORTRAN IV is shown on the following page, followed by side-by-side comparisons of various statements in assembly language and FORTRAN.

FORTRAN IV Executable Statements (11)

Type	General form	Examples
Input		
Format controlled	READ (𝔇, 𝔉) *ℒist*	READ (5, 11) A(1), A(2), A(3) (input device code; format reference; list)
Simplified	READ (𝔇,*𝒞lass*)	READ (5, DATA) (input class)
Output		
Format controlled	WRITE (𝔇, 𝔉) *ℒist*	WRITE (6, 14) SP, SN, T (output device code; format reference; list)
Simplified	WRITE (𝔇,*𝒞lass*)	WRITE (6, OUT) (output class)
Assignment		
Arithmetic	$\mathfrak{v} = \mathcal{E}$	T0M = C(I)
Logical	$\mathfrak{v} = \mathcal{B}$	G = P .LT. S
Functional	*𝔑ame* = $\mathcal{E}$	SIGNAL = X
Conditionals and conditional transfers		
Logical	IF ($\mathcal{B}$) $\mathcal{Q}$	IF (A .LT. B) G0 T0 10
Logical (special (CDC))	IF ($\mathcal{B}$) $\mathcal{S}_1$, $\mathcal{S}_2$	IF (Z .LT. A(J)) 6, 7
Arithmetic	IF ($\mathcal{E}$) $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$	IF (J - N) 10, 15, 20
Computed G0 T0	G0 T0 ($\mathcal{S}_1$, $\mathcal{S}_2$, . . . , $\mathcal{S}_n$), $\mathcal{I}$	G0 T0 (30, 42), K
Unconditional transfer	G0 T0 $\mathcal{S}$	G0 T0 50
Iteration	D0 $\mathcal{S}$, $\mathcal{J} = \mathcal{I}$, $\mathcal{L}$, $\mathcal{K}$ D0 $\mathcal{S}$, $\mathcal{J} = \mathcal{I}$, $\mathcal{L}$	D0 20 J = 1, N, 2 D0 100 I = 6, 50
(Implied iteration for input-output lists)	($\mathcal{A}(\mathcal{J})$, $\mathcal{J} = \mathcal{I}$, $\mathcal{L}$, $\mathcal{K}$)	(A(I), I = 1, N, 2)
Dummy (loop terminal)	CONTINUE	CONTINUE
Subroutine Call	CALL *𝔑ame*(*ℒist*) CALL *𝔑ame*	CALL SORT(A, 1, N) CALL ERR0R
Subprogram Return (Special)	RETURN RETURN $\mathcal{i}$	RETURN RETURN $20

FORTRAN VS. ASSEMBLY LANGUAGE

	FORTRAN	ASSEMBLY LANGUAGE
Add two numbers and store result		CLEAR
		ADD A
	X = A + B	ADD B
		STORE X
		A 1
		B 2
		X ---
Test for -, 0, + and branch	IF (X) 10, 20, 30	SKIPP
	10	JUMP ZER
		JUMP THI
	20	ZER SKIPZ
		JUMP TEN
	30	JUMP TWE
		TEN
		TWE
		THI
Loops	DO 10 I = 1, INDEX	CLEAR
		ADD CNT
		COMPL
	10 CONTINUE	INCR
		STORE I
		CLEAR
		Loop
		ISZ I
		JUMP .-4
		CNT 0005
		I ---

	FORTRAN	ASSEMBLY LANGUAGE
Subroutine	CALL COMPL (N, I)	CLEAR ADD N JMS COMP STORE I
	SUBROUTINE COMPL (N, I) I = -N RETURN END	COMP — — COMPL INCR JUMP I COMP N 0005 I — —
Tabulation	WRITE (LUN, 10) X, Y, Z 10 FORMAT (3F10.3)	- Very messy -
Special I/O	Not possible without special FORTRAN callable assembly language subroutines	READ 02 WRITE 02 etc.

9.2 Subroutines

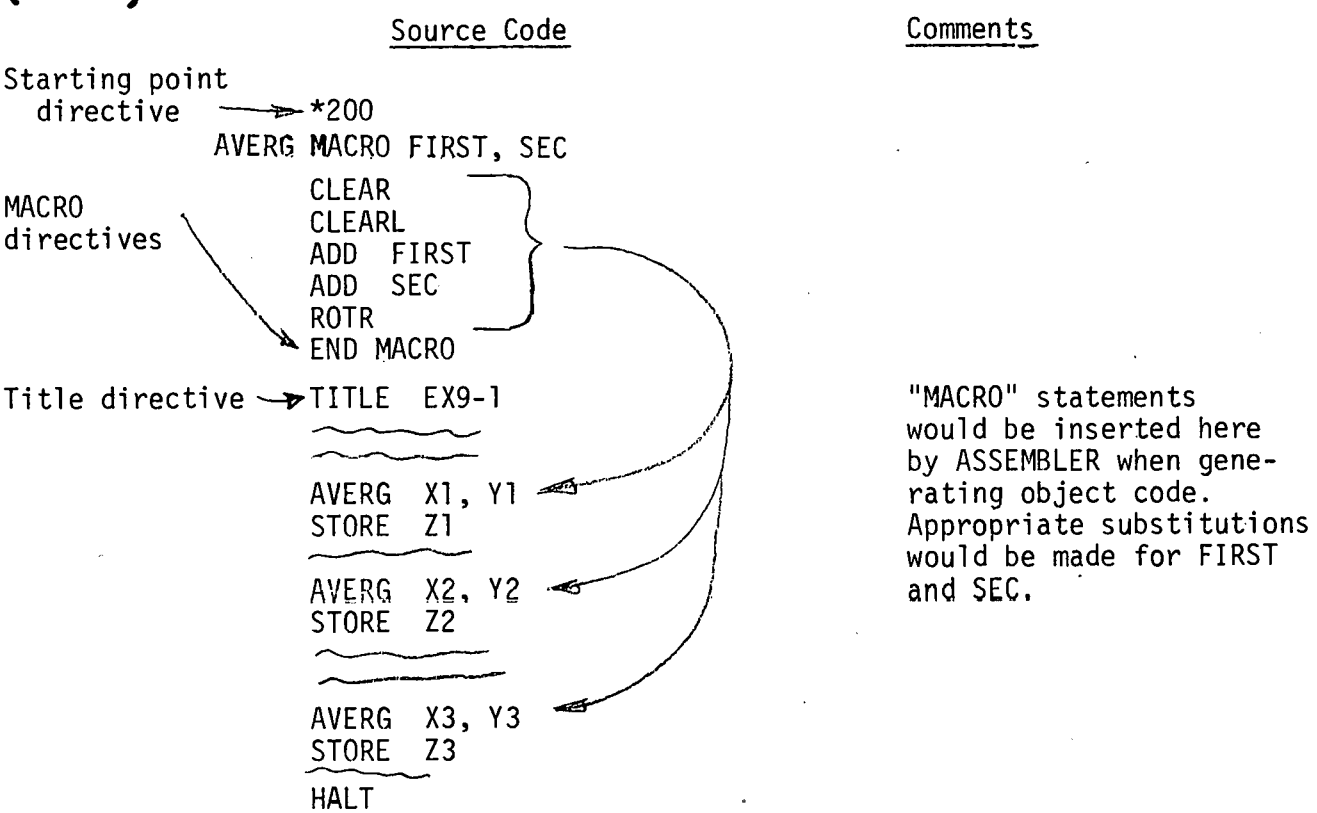
Oftentimes in writing a single program or a group of related programs it becomes necessary to repeat a certain calculation or do some operation over and over. The programmer may then choose to collect the appropriate statements in a distinct grouping called a subprogram. Depending on the programming language, various techniques may be used.

9.2.1 Assembly Language Techniques[2, 5]

Two techniques are commonly used in assembly language programming--Macro's and subroutines. The Macro is strictly a feature of the Assembler (i.e., no hardware is involved). By setting aside a group of statements, defining them as a Macro, and giving them a name, the programmer can effectively insert a copy of these statements whenever he wants. The resulting object code is no shorter than if the programmer had not used the MACRO at all. It's simply a programming convenience.

Ex 9-1

Use a MACRO to average three different pairs of data {X1, Y1}, {X2, Y2}, {X3, Y3}



The second technique for coding subprograms is the subroutine which we have already been exposed to. In all previous examples only one argument was passed. When more than one argument is needed by a subroutine, a technique often used is to insert the data in line right after the JMS statement. In general, subroutines are of somewhat more general use than MACRO's, produce shorter object code, but take longer to execute.

Ex 9-2

Use a subroutine to average three different pairs of data. The address of the first data value is in location 250. The results are to be stored starting in location 260.

```
                Source Code                     Comments
*200
                TITLE EX9-2
                CLEAR
                ADD CNT
                COMPL                           Set up the loop
                INCR
                STORE INDEX

LOOP            CLEAR
                ADD I ADATA
                STORE .+6                       Store data for this
                CLEAR                           loop in two
                ISZ ADATA                       locations following
                ADD I ADATA                     JMS
                STORE .+3
                JMS AVERG
                0000                            Data placed here
                0000
                STORE I ARES                    Store result
                ISZ INDEX                       test for end of loop
                JUMP LOOP
                HALT
CNT             — — —
INDEX           — — —
AVERG           — — —                           Address of first datum
                CLEAR
                CLEARL
                ADD I AVERG                     Calculate average and
                ISZ AVERG                       place data in A
                ADD I AVERG
                ROTR
                ISZ AVERG                       Bump address to next
                JUMP I AVERG                    instruction and return
ADATA           250
ARES            260
```

9.2.2 FORTRAN Techniques [11]

Two techniques are also available for the FORTRAN programmer--FUNCTIONS and SUBROUTINES. A FUNCTION is a special subprogram which returns a value associated with the function name.

Ex 9-3

Do the averaging problem of Ex 9-1 using a FUNCTION statement

```
                    PROGRAM EX9-3
                    DIMENSION X(3), Y(3), RES(3)

                    DO  10 I = 1,3
                    RES(I) = AVERG (X(I), Y(I))
               10   CONTINUE
                    END

                    FUNCTION AVERG (A, B)
                    AVERG = (A + B)/2
                    RETURN
                    END
```

FORTRAN directive tells compiler to save space for X,Y and RES

Calling vector (data)

The general limitation of the FUNCTION subprogram is that only a single numerical result is returned under the associated name.

SUBROUTINES avoid the result limitation of a FUNCTION, but require a CALL statement to get the results.

Ex 9-4

```
          PROGRAM EX9-4
          DIMENSION X(3), Y(3), RES1(3), RES2(3)

          DO 10 I = 1,3
          CALL AVERG (X(I), Y(I), RES1(I), RES2(I))
     10   CONTINUE
          END

          SUBROUTINE AVERG (A, B, C, D)
          C = (A + B)/2
          D = A - B
          RETURN
          END
```

9.3 Mathematical Operations

9.3.1 Single and Double Precision Arithmetic [5]

Two's complement arithmetic yields correct results when the numbers are in the range

$$(-2^{n-1}) \text{ to } (2^{n-1}-1).$$

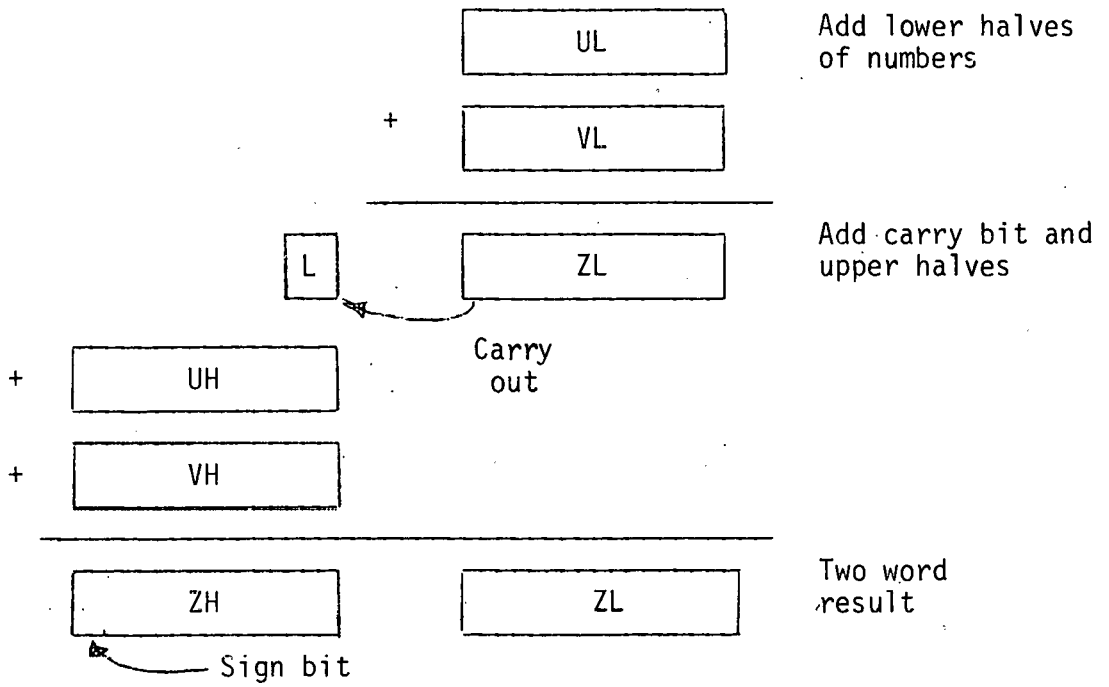
Whenever this range is exceeded, the sign bit (MSB) will be incorrect for use in further calculations. A technique for checking the sign bit was illustrated

in Ex 8-3. The use of a single word for performing such arithmetic is referred to as single-precision.

When integer numbers larger than the basic word size must be handled, double precision (DP) or two-word arithmetic is used. In DP arithmetic the L is used to carry between the two halves of the data words.

Ex 9-5

Write a routine to perform a double precision addition.



Coding

Location	Contents		Comment
DUBADD	CLEARL		Clear link
	CLEAR		Clear accumulator
	ADD	UL	Add *U* low
	ADD	VL	Add *V* low
	STORE	ZL	Store the sum
	CLEAR		Clear accumulator
	ROTL		Get link into accumulator
	ADD	UH	Add *U* high
	ADD	VH	Add *V* high
	STORE	ZH	Store the sum
	HALT		
UL			
UH			
VL			
VH			
ZL			
ZH			

9.3.2 Multiplication and Division[2, 5]

Programs which perform binary multiplication make use of the fact that only shift and add instructions are required. As an example,

Consider the multiplication of 11_{10} by 5_{10}:

11_{10} = 1011

5_{10} = 101

1011

0000

1011

55_{10} = 110111

NOTE: Multiplication of an N-bit number by an M-bit number results in an (N+M)-bit result

It should be noted that in most modern minicomputers, binary multiplication and division are done by hardware.

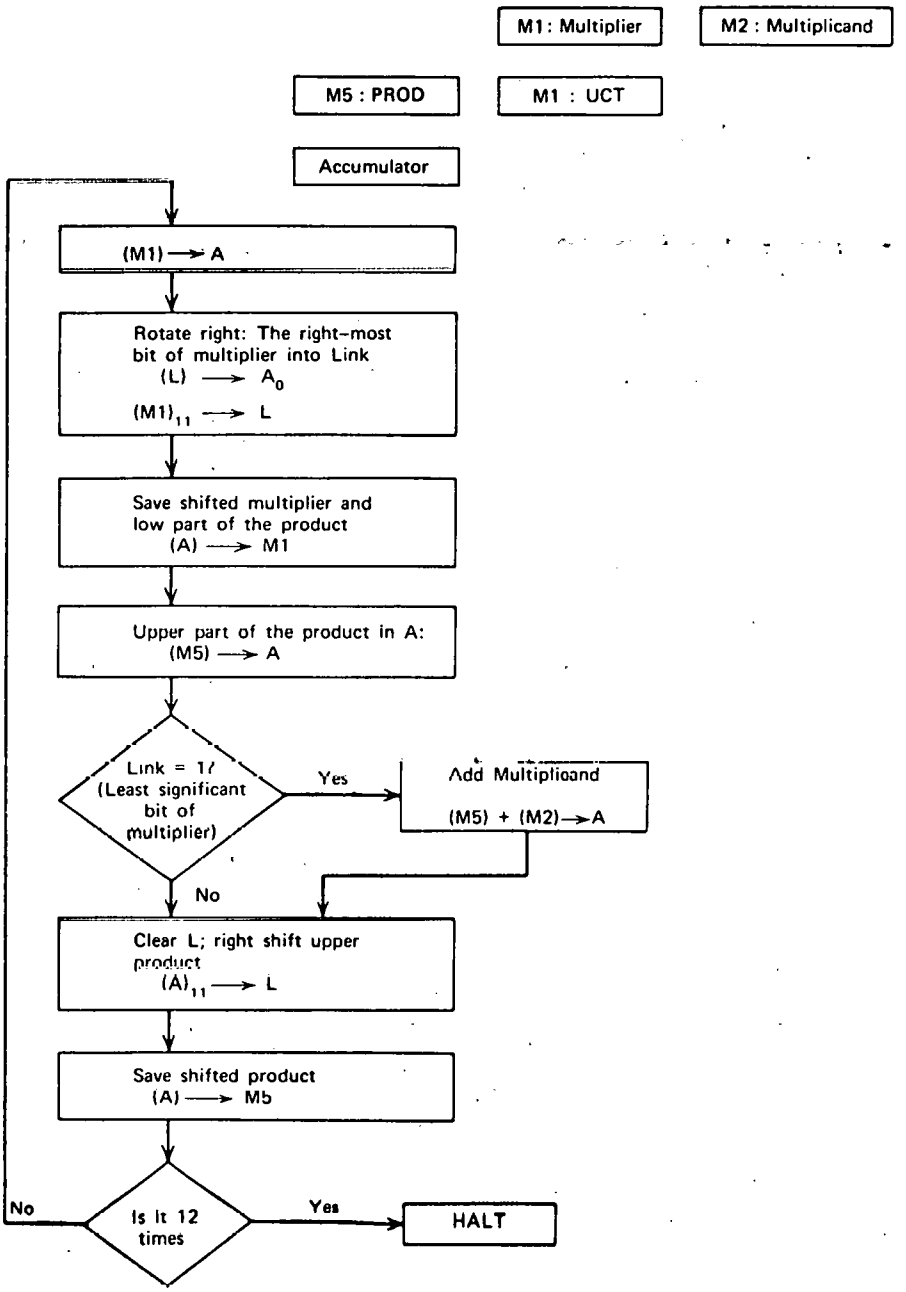
Ex 9-6

Write a program to multiply two binary numbers. Run through the execution of this program using 11_{10} as the multiplicand and 5_{10} as the multiplier. (see next page for flow chart).

Source Code

Location	CONTENTS	COMMENT
MULTPL	CLEAR	Clear accumulator
	ADD MIN12	Take $(-12)_{10}$
	STORE COUNT	Set loop counter (12 bits)
	CLEARL	Clear link
LOOP	CLEAR	Clear accumulator
	ADD M1	Add multiplier
	ROTR	Rotate right
	STORE M1	Store multiplier and part of the product
	ADD M5	Take product
	SKIPL	Is LSB of multiplier = 1
	SKIP	No
	ADD M2	Yes, add multiplicand
	CLEARL	Clear link
	ROTR	Rotate right the product
	STORE M5	Save the product
	ISZ COUNT	Enough times?
	JUMP LOOP	No, do again
	HALT	
M1	0011	
M2	0020	
M5	0000	
COUNT	0000	
MIN12	7764	$(-12)_{10}$

FLOW CHART (2)


M1 : Multiplier
M2 : Multiplicand
M5 : PROD
M1 : UCT
Accumulator
(M1) → A
Rotate right: The right-most bit of multiplier into Link
(L) → A_0
$(M1)_{11}$ → L
Save shifted multiplier and low part of the product
(A) → M1
Upper part of the product in A:
(M5) → A
Link = 1? (Least significant bit of multiplier)
Yes
Add Multiplicand
(M5) + (M2) → A
No
Clear L; right shift upper product
$(A)_{11}$ → L
Save shifted product
(A) → M5
No
Is It 12 times
Yes
HALT

DETAILED EXECUTION OF BINARY MULTIPLICATION

Example

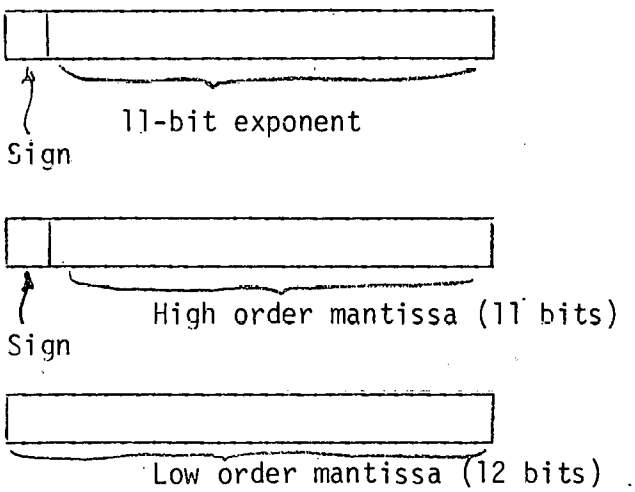
NOTE: M2 (Multiplicand) = 000 000 001 011

	Instr.	L	A	M5 Most Significant Half of Product	M1 Multiplier and Least Significant Half of Product
LOOP 1					
	CLEAR	0	000 000 000 000	000 000 000 000	000 000 000 101
	ADD M1	"	000 000 000 101	"	"
	ROTR	1	000 000 000 010	"	"
	STORE M1	"	"	"	000 000 000 010
	CLEAR	"	000 000 000 000	"	"
	ADD M5	"	000 000 000 000	"	"
	SKIPL	"	"	"	"
	ADD M2	"	000 000 001 011	"	"
	CLEARL	0	"	"	"
	ROTR	1	000 000 000 101	"	"
	STORE M5	"	"	000 000 000 101	"
LOOP 2					
	CLEAR	"	000 000 000 000	"	"
	ADD M1	"	000 000 000 010	"	"
	ROTR	0	100 000 000 001	"	"
	STORE M1	"	"	"	100 000 000 001
	CLEAR	"	000 000 000 000	"	"
	ADD M5	"	000 000 000 101	"	"
	SKIPL	"	"	"	"
	SKIP	"	"	"	"
	CLEARL	0	"	"	"
	ROTR	1	000 000 000 010	"	"
	STORE M5	"	"	000 000 000 010	"
LOOP 3					
	CLEAR	"	000 000 000 000	"	"
	ADD M1	"	100 000 000 001	"	"
	ROTR	1	110 000 000 000	"	"
	STORE M1	"	"	"	110 000 000 000
	CLEAR	"	000 000 000 000	"	"
	ADD M5	"	000 000 000 010	"	"
	SKIPL	"	"	"	"
	ADD M2	"	000 000 001 101	"	"
	CLEARL	0	"	"	"
	ROTR	1	000 000 000 110	"	"
AFTER 12 LOOPS →		0	000 000 000 000	000 000 000 000	000 000 110 111

9.3.3 Floating Point Arithmetic (5)

So far, all mathematical operations have used integer (whole number) arithmetic. To handle fractional numbers or extremely large or small numbers, floating-point (F-P) arithmetic is used. On a small machine such as our hypothetical computer, special floating point subroutines would be supplied by the manufacturer.

In a typical 12-bit computer, three words are required to represent a F-P value:



The FP numbers are represented in a normalized form, i.e., the exponent is adjusted to produce a mantissa which is the largest possible fraction less than one:

	Normalized Form	
Decimal Number	Mantissa	Exponent
12.625	.12625	2
-.0899	-.8990	-1

In programming F-P operations, the programmer sets up calls to the appropriate subroutines. The results are "accumulated" in a group of three dedicated memory locations in page 0.

Ex 9-7

Read a number in from the teletype, multiply by ten, and store the result using F-P arithmetic:

```
                Source Code             Comment
                JMS I FPIN              Call input routine
                JMS I FPMUL             Call multiply routine
                TEN                     Address of multiplicand
                JMS I FPPUT             Call storage routine
                ANS                     Storage address
                HALT                    Halt
FPIN            6200                    Pointer to input routine
FPMUL           6600                    Pointer to multiply routine
FPPUT           7322                    Pointer to storage routine
TEN             4                       Exponent
                2400                    High order mantissa
                0000                    Low order mantissa
ANS             0                   )   Three
                0                   }       word
                0                   )           result
```

In machines equipped with F-P hardware, OP codes for manipulating F-P numbers are part of the basic instruction set. A special F-P accumulator is also provided and the programmer codes the operation just like integer arithmetic except for allowing multi-word results. In machines equipped with F-P hardware, execution times for the F-P arithmetic operations are tremendously decreased. For instance,

	Execution Time	
	Software	Hardware
Multiply	1100 microsec	28 microsec

9.3.4 FORTRAN Arithmetic [2, 11]

FORTRAN was designed to permit mathematical operations to be coded very much like the way they are formulated. Since the character set of the computer is limited, the following symbols are used to represent common operation:

Addition	+
Subtraction	-
Multiplication	*
Division	/
Exponentiation	**

For example, the hypotenuse of a right triangle having sides A and B is

```
C = (A**2 + B**2) **.5
```

Parenthesis should be used freely to make the statements unambiguous. The FORTRAN compiler sets up the order of execution of terms within a given set of parenthesis according to the following heirarchy:

```
**
*, /
+, -
```

For instance, if the equation to be coded in FORTRAN was

$$X = \frac{A + B}{C},$$

then the following statement would give the wrong result

```
X = A + B/C
```

whereas

```
X = (A + B)/C
```

would give correct results. The "bug" was due to the fact that the division operation would precede the addition, thus resulting in

$$X = A + \frac{B}{C}$$

The ability to handle F-P as well as integer numbers is built into FORTRAN. If the programmer uses a name for a variable that begins with the letters I, J, K, L, M, or N, the compiler assumes it is integer unless told otherwise. Variable names that begin with any other character are assumed to be F-P unless otherwise stated

Integer - INDEX, IDATA, NUMBER

F-P - XVAL, DATA, ZNUMBER

It should be noted that operations between integer and F-P variables may lead to peculiar results, and the resulting code may not be machine independent.

FORTRAN compilers are generally blessed with a group of mathematical routines referred to as the "math library." These routines include FUNCTIONS to perform such common operations as trigonometry functions, absolute value, square root, etc. For instance, the hypotenous problem could have been written

```
C = SQRT (A**2 + B**2)
```

and one of the angles of this triangle would be

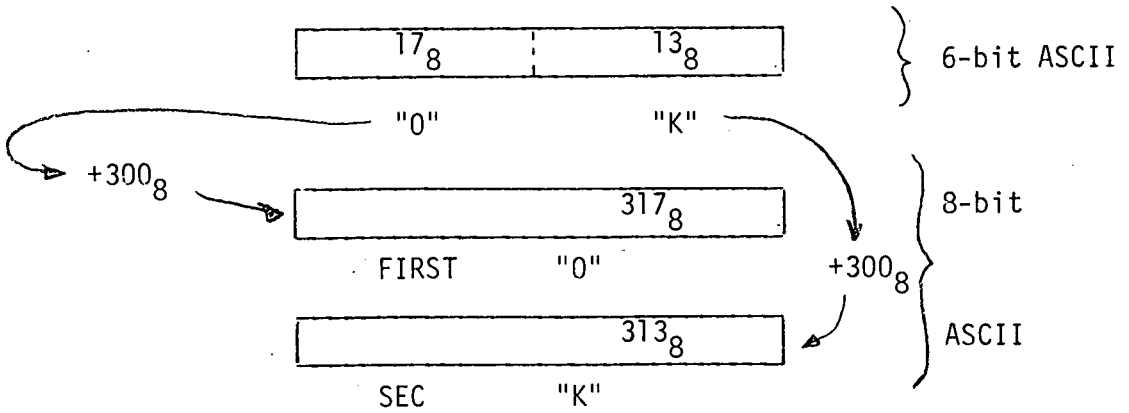
```
PHI = ATAN (B/A)
```

9.4 Alphanumeric Data[2]

Alphanumeric (AN) data must be represented internally in the computer in some numeric code such as 6, 7 or 8 bit ASCII or BCD. Since these codes only require a fraction of a word, more than one AN character can be packed to a word.

Ex 9-8

Unpack the 6-bit ASCII characters "OK" into separate words, convert to 8-bit ASCII, and store in FIRST and SEC.



Flow chart:

```
        CLEARL
        CLEAR
        ADD     SIX
        ROTR
        ROTR
        ROTR
        ROTR
        ROTR
        ROTR
        AND     MASK
        ADD     POS300
        STORE   FIRST
        CLEAR
        ADD     SIX
        AND     MASK
        ADD     POS300
        STORE   SEC
        HALT
SIX     1713
MASK    0077
POS300  0300
FIRST   —
SEC     —
```

9.5 Input-Output (I/O Programming

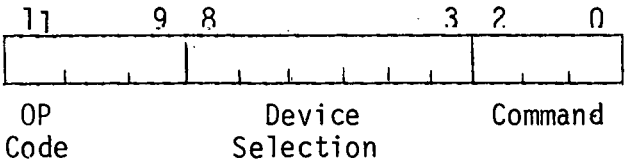
The programming of I/O operations is probably the most difficult part of problem coding. There are two general ways in which data can be transferred between peripheral devices and the computer:

1. Under hardware control directly between the device and memory, referred to as direct memory addressing (DMA), or cycle stealing,
2. under software (programmed control through the accumulator.

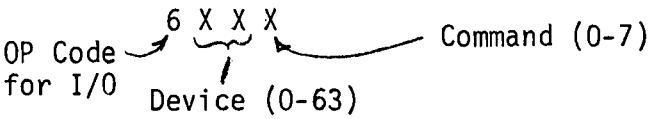
In this section we are concerned with some of the programming aspects of I/O operation. In the next section we will look at the hardware required to execute programmed and cycle-stealing I/O.

9.5.1 Assembly Language Techniques[2, 5]

The I/O instruction for our hypothetical computer is formatted as follows:



The OP code assigned for I/O operations in this machine is 006 and distinguishes this type of operation from others. In octal, the instruction looks like



A computer has only one I/O bus and all the peripheral devices are attached to it. To distinguish one device from another, a device code is issued by the computer with each command. Only the device corresponding to that device code will respond. The command consists of 3 bits, so 8 possible codes can be sent. Exactly how the command codes are interpreted by the device hardware is different for each device.

The types of I/O instructions available to the assembly language user are as follows:

READXX - Reads in a data word from device XX and places it in the accumulator

WRITEXX - Outputs the contents of the accumulator to device XX.

SKIPXX - Because the peripheral devices are usually much slower than the computer, they are provided with a 1-bit register called a flag which the computer can check to see if the peripheral is ready to accept a new command (flag set). The SKIPXX instruction checks the flag of device XX and, if set, the next instruction is skipped.

CLEARXX - Once the ready flag has been set, it can only be cleared by the computer via the CLEARXX instruction.

ION - Enables the interrupt flip-flop

IOF - Disables the interrupt flip-flop

GOXX - Instructs device XX to begin a DMA transfer

Depending on the complexity of the I/O operation involved, the assembly language programmer can include I/O statements in his own applications software, or he can make use of a general-purpose subroutine written for that device called a handler or device driver. Handlers fall into the realm of system software and are generally supplied by the computer system manufacturer. The handlers for user-added devices must, of course, be written by the user. The advantage to the assembly-language programmer is that he doesn't have to be concerned with the peculiarities of a particular device and can eliminate the effort of programming the detailed I/O operation every time he wants to use it.

The following example illustrates one technique for carrying out I/O transfers. A more general approach will be developed in Sec. 10.3.--after we have studied the interface hardware in more detail.

Ex 9-9

Read a character from the teletype keyboard (Device 03), store it, and then "echo" that character to the teletype printer (Device 04).

```
        SOURCE CODE         COMMENT
        CLEAR               Clear A
        CLEAR03             Clear keyboard flag
        SKIP03              } Tight loop to check for
        JUMP .-1            } character ready
        READ03              Read character
        STORE CHAR
        CLEAR               } Clear out the printer
        WRITE04             } buffer
        ADD CHAR            Recall the character
        SKIP04              } Tight loop to check for
        JUMP .-1            } completion of previous WRITE
        WRITE04             Output character
        HALT
CHAR    —
```

9.5.2 FORTRAN Techniques[2, 11]

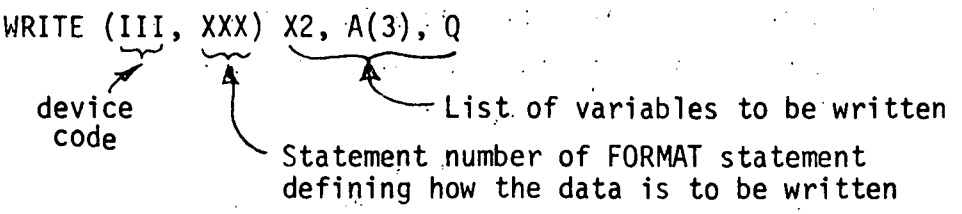
FORTRAN users have three general purpose I/O statements available:

```
READ (III, XXX) X1, A(I), Z
```

III — device code

XXX — Statement number of FORMAT statement defining how the data is arranged.

X1, A(I), Z — List of variables to be read



XXX FORMAT( )

Field definitions and control symbols

Various field definition and control symbols are used in FORMAT statements. The most common are as follows:

/	Skip a line
NX	Skip N spaces on current line
Fd.w	Write out a floating-point variable using up to d digits with w to the right of the decimal point.
Ed.w	Write out a floating-point variable using scientific notation with up to d characters and with w to the right of the decimal point.
Id	Write out an integer variable using up to d digits
dHXXXX..X	Reproduce the d ASCII characters following the H (referred to as Hollerith field).

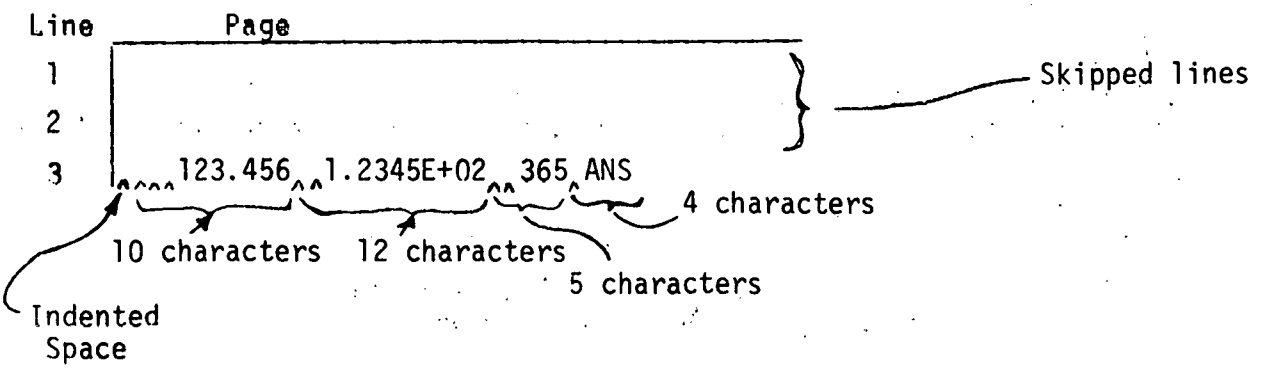
Ex 9-10

Output the variables X, Y and IZ on the third line of a printer page (device 2), indented 1 space. Label the results "ANS".

Source Code

```
      WRITE (2,10) X, Y, IZ
10    FORMAT ( ,//, 1X, F10.3, E12.4, I5, 4H ANS)
```

Results

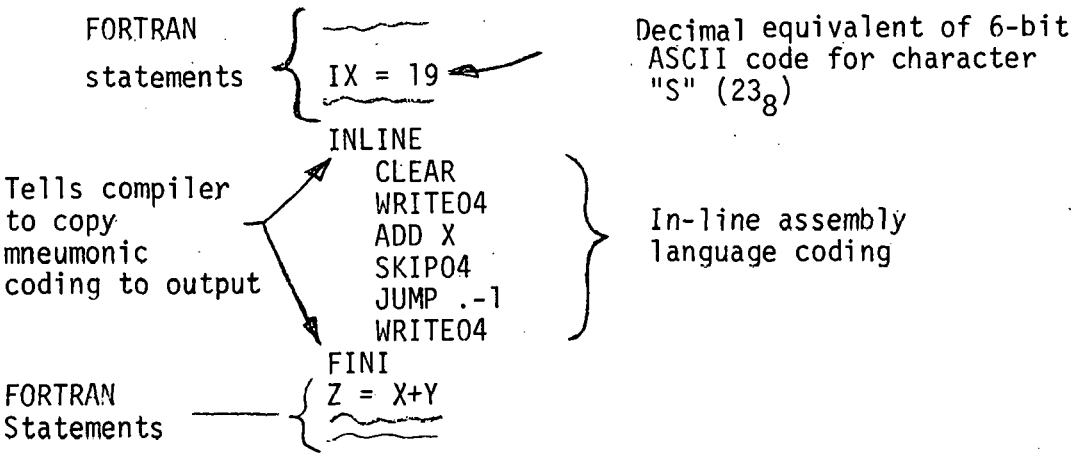


For each I/O statement the FORTRAN compiler generates calls to the appropriate handler subroutine. Because the compiler and handlers are provided by the computer system manufacturer, I/O to devices other than what the manufacturer has provided is greatly impeded in FORTRAN. This becomes a real problem when a user adds some unique device to a system and wishes to do his applications programming in FORTRAN. Two solutions are possible:

1. Write a special subroutine in assembly language which serves as a handler and can be called by the FORTRAN program.
2. Resort to "in-line" assembly language coding in the middle of the FORTRAN program.

Ex 9-11

Use in-line coding to output the ASCII character "S" to the teletype printer.



It should be noted that not all compilers permit in-line coding.

10. CPU AND INTERFACE HARDWARE

10.1 CPU Architecture and Timing[2]

The organizational scheme of the central processing unit (CPU) is referred to as its architecture. The architecture of our hypothetical computer system was introduced in Section 8.1 and is repeated on the following page for convenience. In the following paragraphs we will be concerned with the details of how the hardware functions during instruction execution.

The execution of an instruction takes from 1 to 3 machine cycles, depending on how many times memory must be accessed. The various cycles, or major states are as follows:

FETCH - Get the instruction from memory, increment the PC, decode the instruction, and execute it if a single-cycle instruction.

DEFER - Used only by instructions having indirect-addressed operands. The effective address of the operand is retrieved from memory.

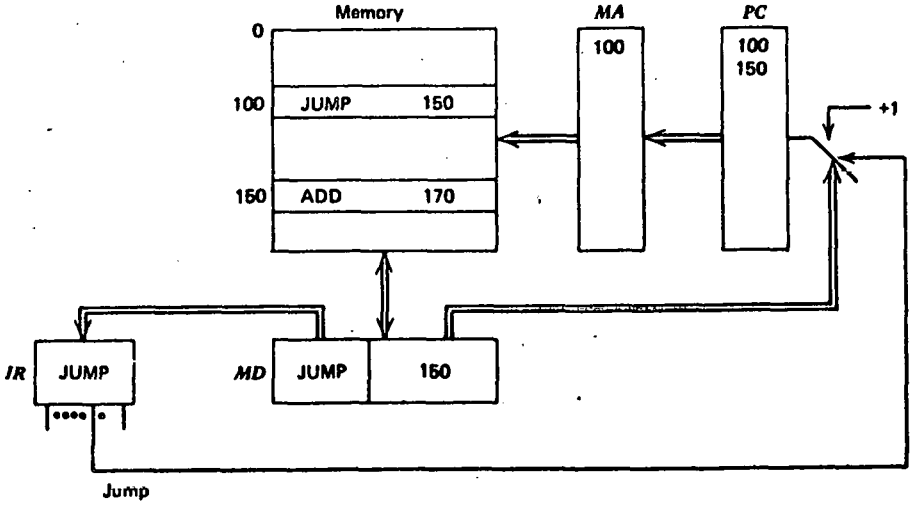
EXECUTE - Implement the decoded instruction. Required by all memory reference instructions.

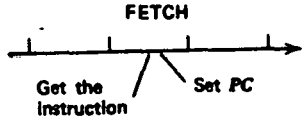
The detailed execution of several types of instructions will be considered below.

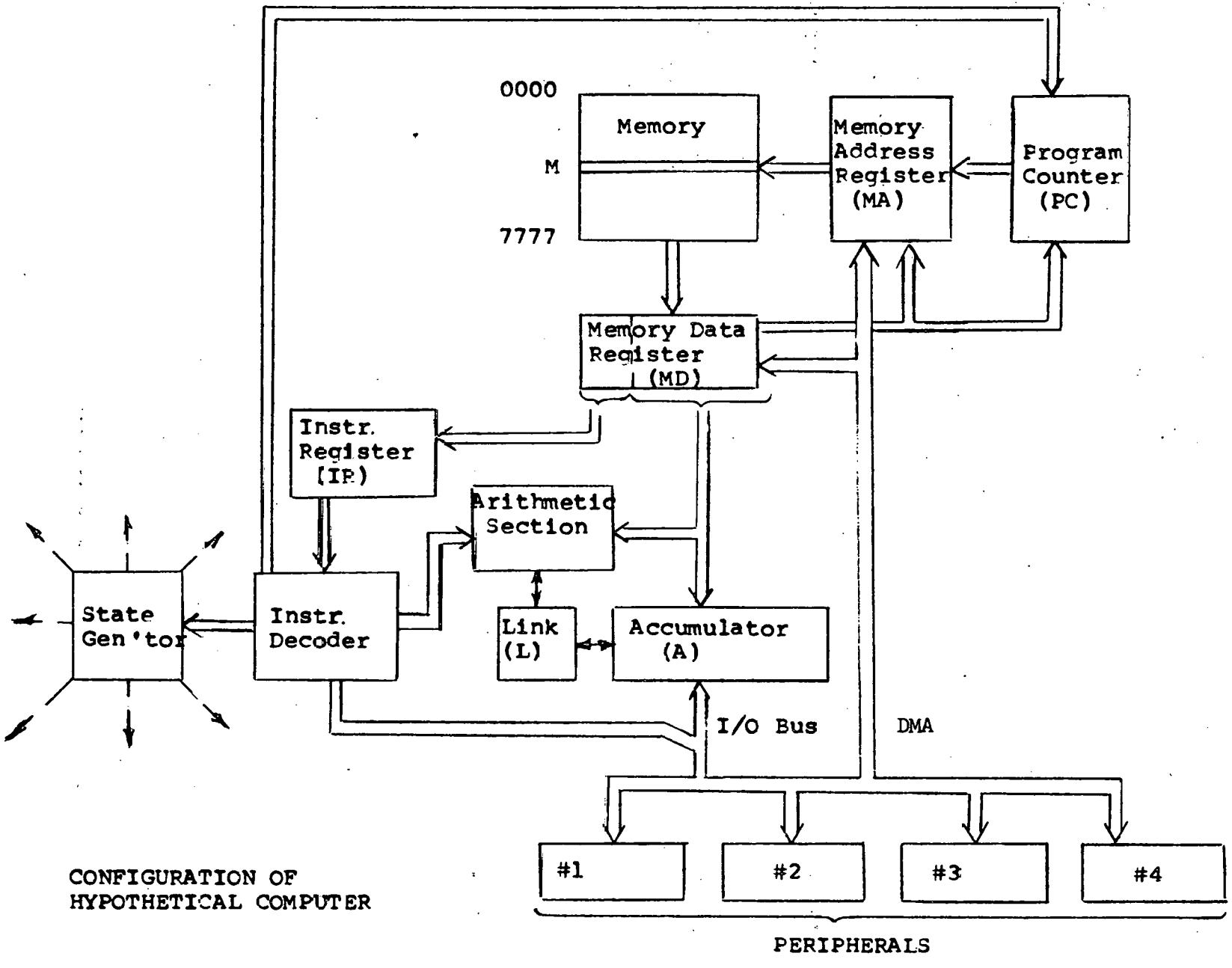
10.1.1 Single-cycle Instructions[2]

All register reference instructions require only a FETCH cycle to retrieve, decode, and execute the instruction. As an example, the CLEAR instruction can be executed immediately after it is decoded, since no further information is needed. Other instructions in this category are SKIPP, COMPL, and ROTR, etc.

The direct-addressed JUMP M instruction is also a single-cycle instruction since it sets the PC counter to the address contained in the instruction and does not involve memory directly:



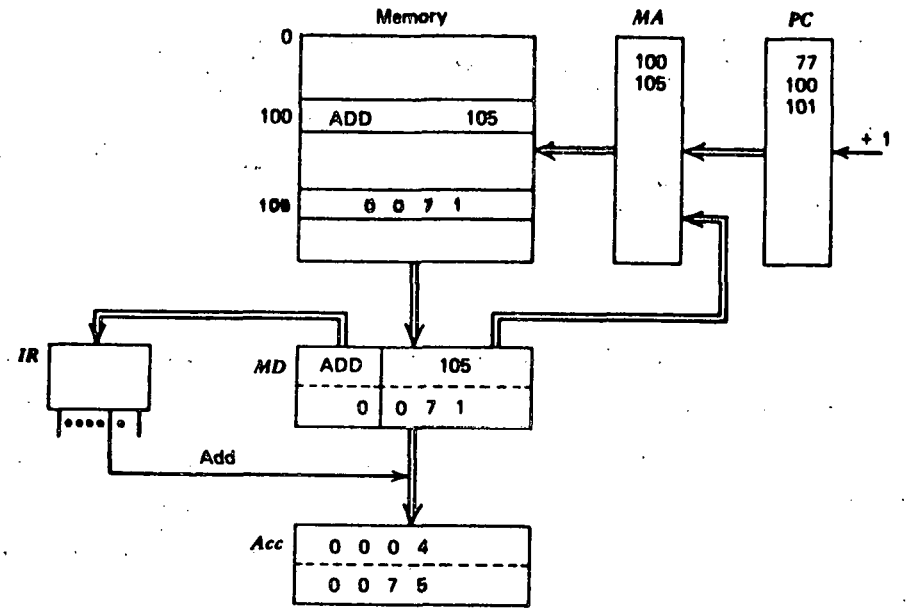


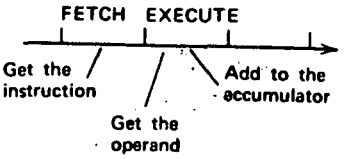


CONFIGURATION OF
HYPOTHETICAL COMPUTER

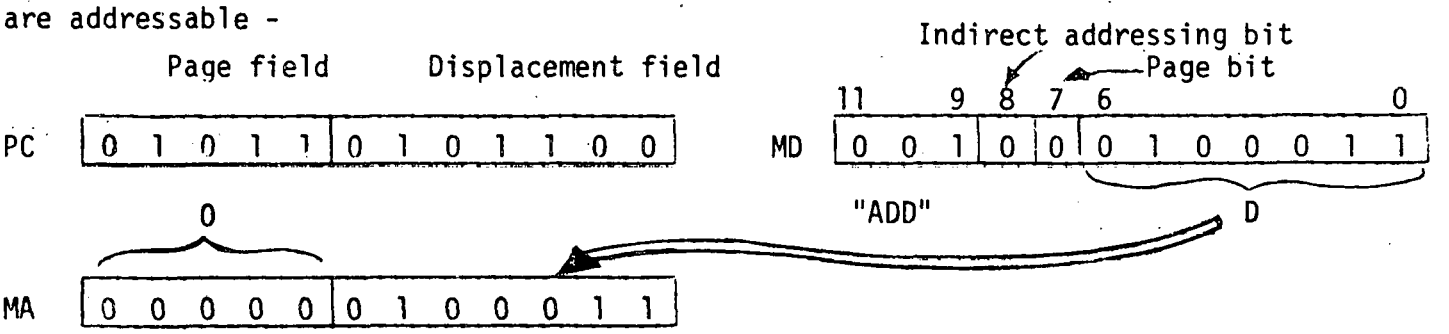
10.1.2 Two-Cycle Instruction (2)

All memory reference instructions using direct addressing require a FETCH cycle to retrieve the instruction, and an EXECUTE cycle to retrieve the operand and perform the operation. Instructions in this category are STORE M, ADD M, AND M, etc.

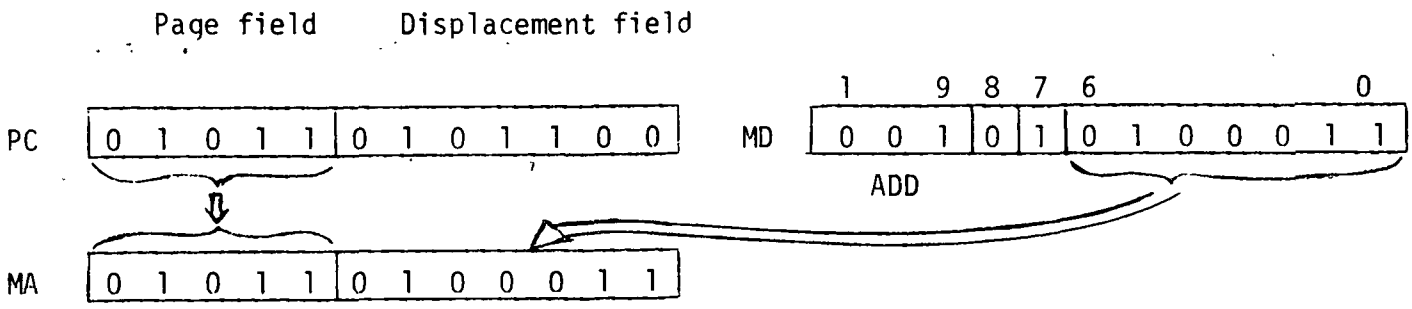




Page relative addressing is achieved by formatting both the PC and MA registers into 7-bit displacement fields and 5-bit "page" fields. If the page bit of the instruction is 0, the page field of the MA is set to 0 and the displacement contained in the instruction copied into the MA. Thus locations 0-177 are addressable -

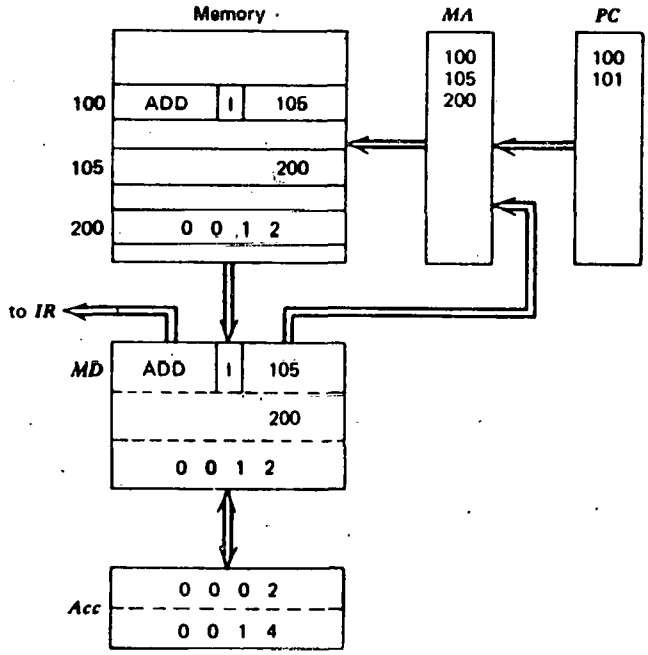


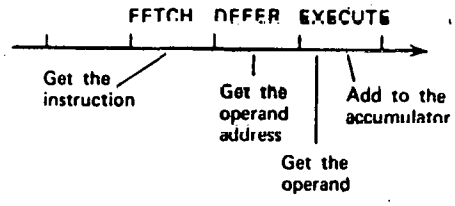
If the page bit of the instruction is 1, the contents of the PC page field are copied into the MA and the displacement from the instruction is written into the displacement field. Thus 000-177, 200-277, 400-477, etc., are addressable



10.1.3 Three-cycle Instruction (2)

Any of the two-cycle memory reference instructions can require three cycles (FETCH, DEFER and EXECUTE) if indirect addressing is used. During the DEFER cycle the address built up in the MA is used to retrieve the address of the operand and bring it out into the MD. The entire MD is then copied into the MA where it is used during the EXECUTE cycle to retrieve the operand and perform the operation.

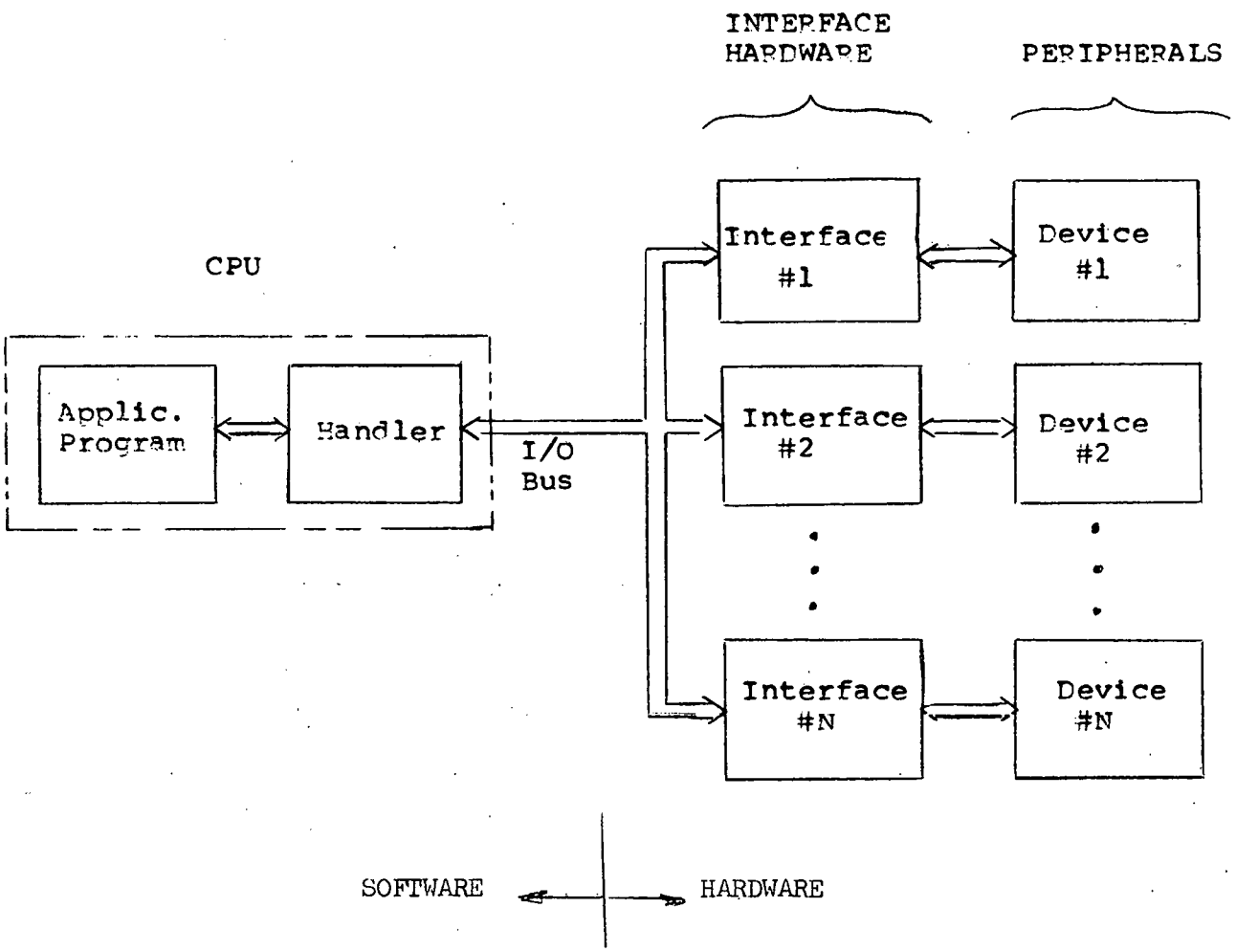




10.2 Program Controlled I/O

10.2.1 Interface Hardware (2)

All communication to peripheral devices takes place over the I/O bus. Every device receives the same device code and command from the CPU in a "party-line" fashion. Each peripheral device, e.g., magnetic tape, paper tape reader, etc., has associated with it a logic circuit called an interface which translates the information sent by the CPU and actuates the peripheral in a manner appropriate for the command received.



A typical CPU/INTERFACE/PERIPHERAL configuration is illustrated on the following page. The cabling between the CPU and the interface constitutes the I/O bus. The interface for only one device is shown. It must be remembered that the same CPU output signals (command code, device code, timing, DATAOUT) go to every interface, whereas each interface develops its own SKIP, INTERRUPT, and DATAIN signals which are DOT/AND-OR'd onto a single set of lines going into the CPU.

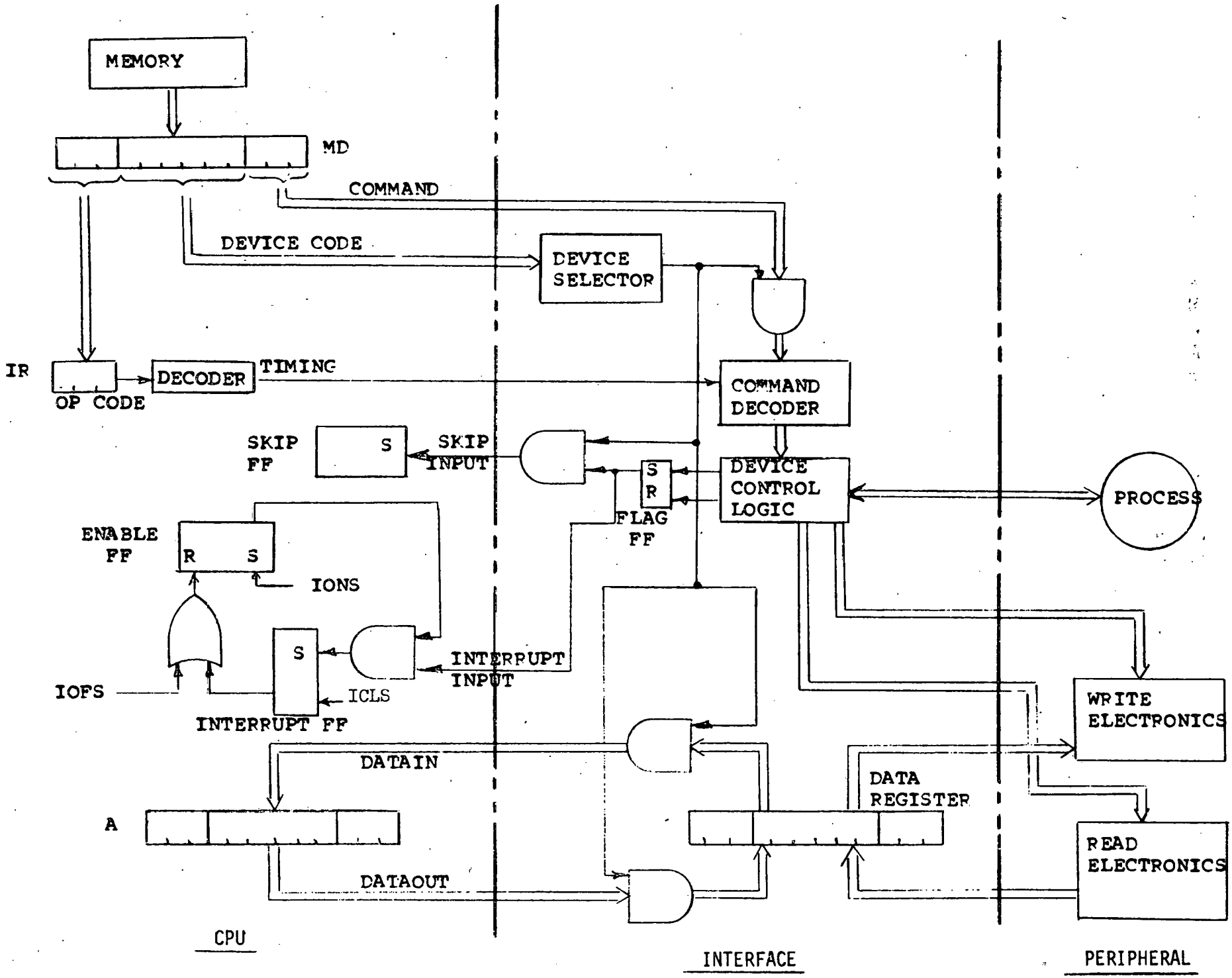
Since every interface receives the I/O command, a device selector and device code combination must be used to identify the device addressed. The device address is decoded and used to gate the command lines into the command decoder. The device control logic activates the device hardware in accordance with the command. If either READ or WRITE operations are involved, the DATAIN or DATAOUT lines connected to the data register are gated by the output of the device selector. When the commanded operation is complete, the device control logic sets the Flag F-F. This Flag can be used in three ways: (1) It can be brought into the computer and tested by executing a SKIPXX instruction which gates the Flag onto the skip line via the device code, (2) it can be used to set the interrupt F-F directly, or most commonly, (3) it does both. Once the Flag F-F has been set, a CLEARXX I/O instruction must be issued to clear it.

The interrupt input line is gated by the interrupt enable F-F to set the interrupt F-F. Before the interrupt F-F can actually be set, it must be enabled by the IONS signal which results from the execution of an ION instruction. When the interrupt F-F sets, it causes a sequence of steps to occur in the CPU. Upon completion of the current instruction, the contents of the PC is stored in location 0, and the PC is set to location 1. Generally speaking, location 1 contains a JUMP to the location of an interrupt handling routine.

Programming examples illustrating the use of the SKIP instruction and interrupts are presented in the next section.

10.2.2 Programmed Data Transfers (2)

Using the interface hardware discussed in the previous section, programmed data transfers can be accomplished in three different ways:


MEMORY
MD
COMMAND
DEVICE CODE
DEVICE SELECTOR
IR
OP CODE
DECODER
TIMING
COMMAND DECODER
SKIP FF
S
SKIP INPUT
S
R
DEVICE CONTROL LOGIC
PROCESS
FLAG FF
ENABLE FF
R
S
IONS
S
INTERRUPT INPUT
IOFS
ICLS
INTERRUPT FF
WRITE ELECTRONICS
DATAIN
DATA REGISTER
A
READ ELECTRONICS
DATAOUT
CPU
INTERFACE
PERIPHERAL

Unconditional: The READ and WRITE commands are issued without regard to whether or not the device is ready for them.

Ex 10-1

Mnemonic	Comment
CLEAR	Clear device data register
WRITE03	
ADD DATA	Output data to device
WRITE03	
DATA 1234	

Conditional: Before each instruction is executed the skip F-F is tested to make sure the addressed device is ready.

Ex 10-2

Mnemonic	Comment
SKIP 23	Test for ready
JUMP .-1	
CLEAR	Input data
READ 23	
STORE DATA	
CLEAR23	Clear flag
DATA 0	

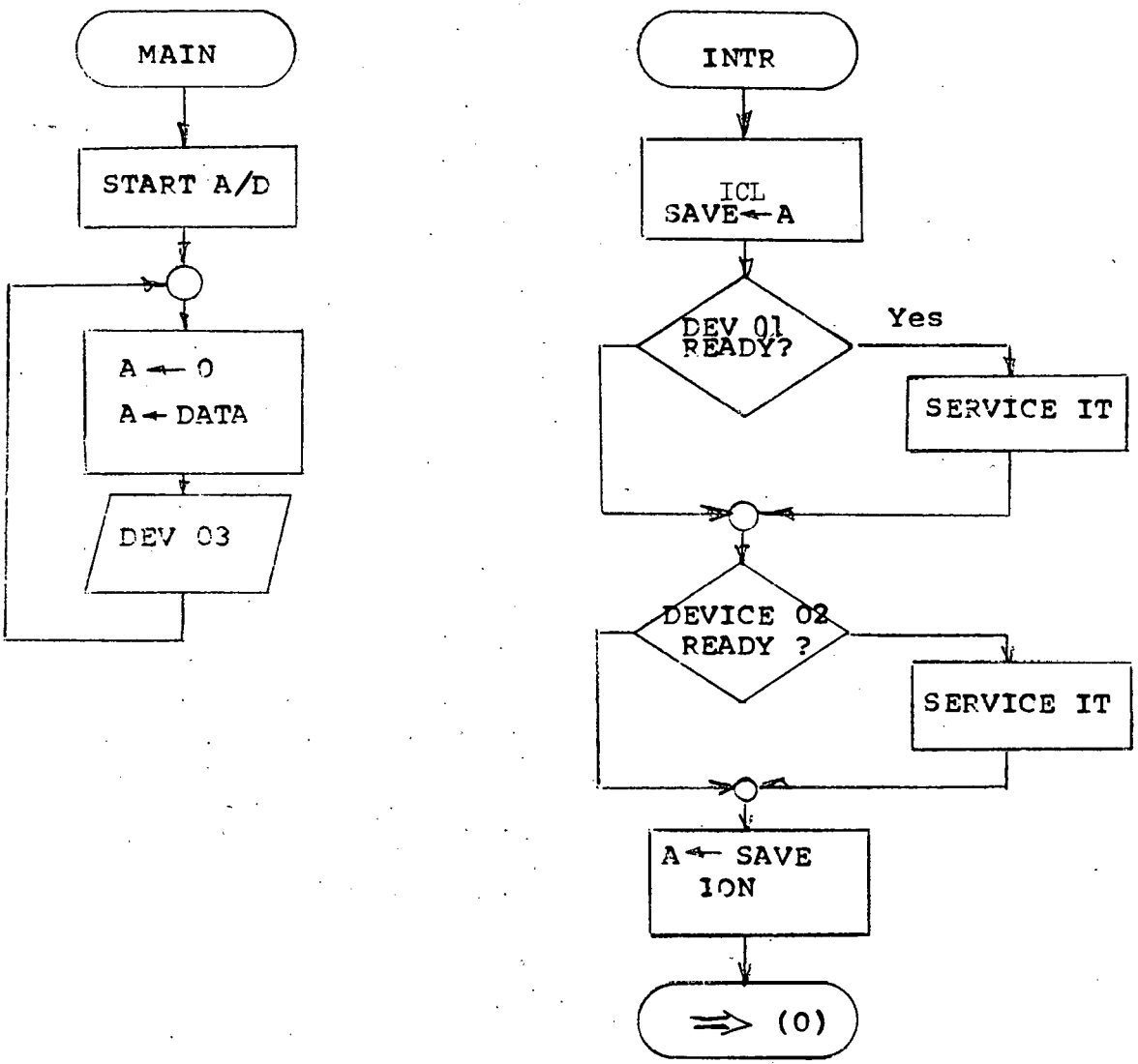
Interrupt Driven: Interrupts can occur at any time during the execution of a program because the interrupt is a hardware function. Since any of the peripherals could have caused the interrupt, the programmer must check the Flag F-F of each by using the SKIPXX instruction. This procedure is called device polling. The routine that takes care of servicing the interrupts is called the interrupt handler.

Ex 10-3

Suppose we have a system in which the only interrupt driven peripherals are a card reader (Device 01) and an A/D system (Device 02). Write a program which reads in a data value from the A/D system everytime it interrupts, and continuously displays the most recent value on a hardware display (Device 03) between interrupts.

It should be noted that in the example all three types of data transfers are used, i.e., unconditional, conditional, and interrupt.

FLOW CHART



Coding

	Location		Mnemonic	Comment
Special page 0 words	0		Ø	Filed in with PC after interrupt
	1		JUMP INTR	Jump to interrupt handler
	2	DATA	Ø	Data goes here
	~		~	
Interrupt Handler	200	INTR	STORE SAVE	Save A
	201		ICL	Clear interrupt
	202		SKIPØ1	Test for card reader
	203		JUMP ATOD	No, try A/D
	204	CARD	~	Service card reader
	~		~	
	207		CLEAR Ø1	Clear flag
	210	ATOD	SKIPØ2	Test for A/D
	211		JUMP OUT	No, go back
	212		CLEAR	Read in data value
	213		READØ2	
	214		STORE DATA	Store it
	215		CLEARØ2	Clear flag
	216	OUT	CLEAR	Restore A
	217		ADD SAVE	
	220		ION	Turn interrupt back on
	221		JUMP I Ø	Go back to program
	222	SAVE	Ø	
Display Program	300		CLEAR	Start up A/D
	301		ADD START	
	302		WRITEØ2	
	303	LOOP	CLEAR	Pick up current value
	304		ADD DATA	
	305		WRITEØ3	Display it
	306		JUMP LOOP	Do it again
	307	START	ØØØ7	

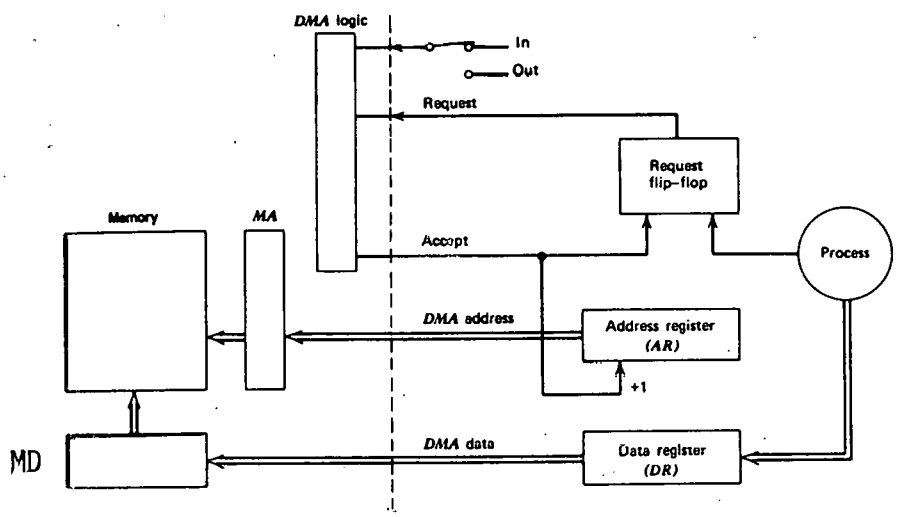
10.2.3 Advanced Interrupt Structures[2]

The type of interrupt scheme which we have just studied is really a very primative one since it has only one level of interrupt priority, i.e., the interrupt hardware is disabled during interrupt processing to prevent interrupts by another device. Also, the devices must be polled to see who caused the interrupt. More sophisticated hardware systems offer multiple levels of interrupt priority--if, say, device B has a lower priority than A, simultaneous interrupt by A and B would be arbitrated by hardware in favor of B. Also the handlers for each device are directly connected, that is, if device X interrupts, program control is transferred directly to a memory location dedicated to that device. In this way device polling is eliminated.

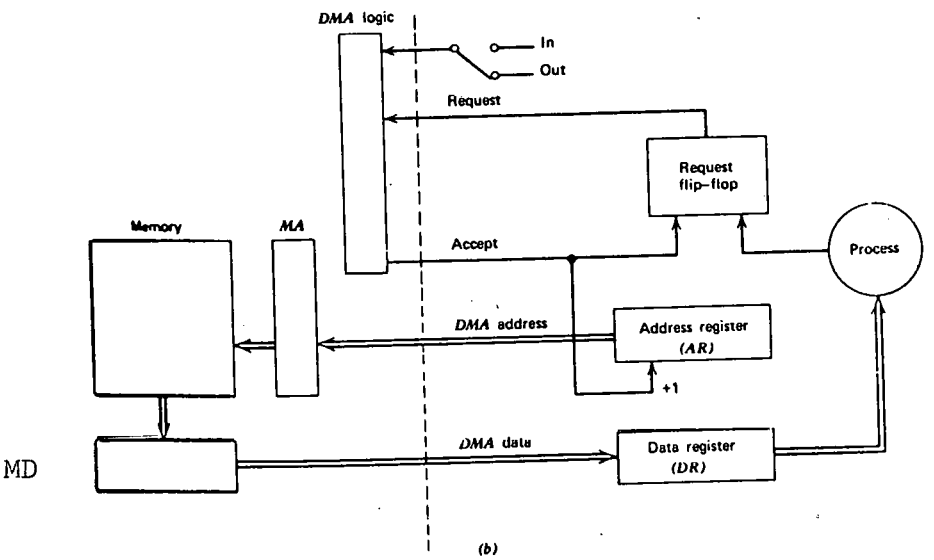
10.3 Direct Memory Access (DMA) I/O

10.3.1 The DMA Interface[2]

The DMA data transfer is a means for transferring data into or out of memory without direct program intervention. The interface hardware causes the transfer to occur independently of the executing program using a technique called cycle-stealing. As compared to programmed I/O, DMA is used in applications requiring large blocks (contiguous locations in memory) of data to be transferred at very high rates (500 + kc), and does not utilize the A. All of the counting and addressing functions performed by software in programmed I/O are carried out by hardware in the DMA interface. The general configuration of a DMA interface is illustrated in the following figures.



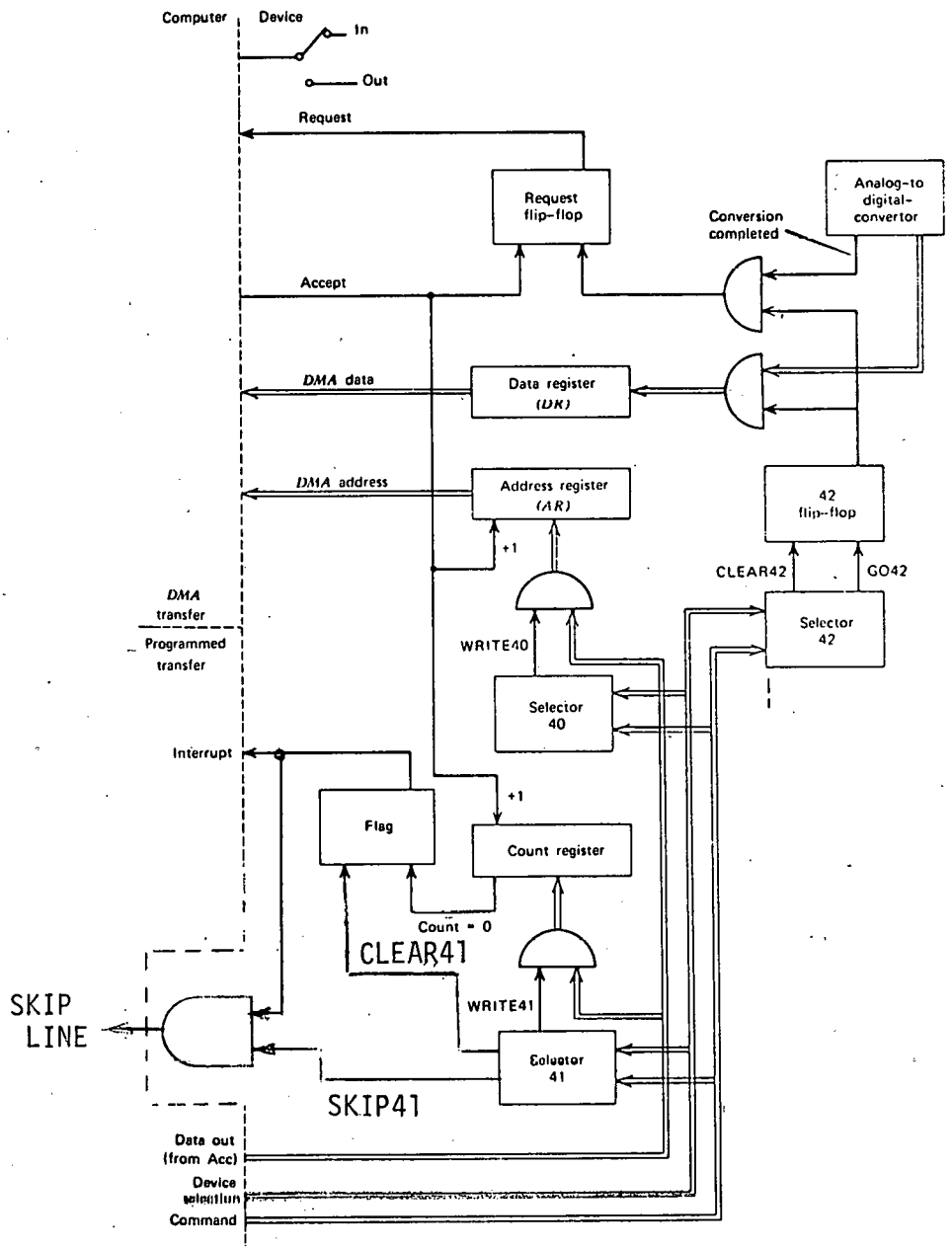
DMA INPUT TRANSFER[2]



DMA OUTPUT TRANSFER(2)

To initiate a DMA transfer the peripheral device sets the request F-F in the interface. During the very next cycle the CPU stops what it was doing, acknowledges the request, transfers the address in the interface address register (AR) to the MA of the CPU, and then carries out the transfer between the data register (DR) and memory. The AR is then incremented and the CPU resumes the program where it left off.

A more detailed interface drawing is shown on the following page. A counter has been added to keep track of the number of words transferred, and a Flag F-F similar to the one used in programmed data transfers is used to interrupt at the completion of the block transfer. The counter register (41), address register (40), and Request F-F (42) are all assigned addresses so that they can be initialized by programmed I/O. So we see that programmed I/O is required to set up and reset the interface, whereas the actual data transfer is by DMA.



DMA INTERFACE HARDWARE (2)

10.3.2 Programming the DMA Transfer (2)

Once set up, the transfer of data using DMA is completely hardware controlled. Programming, however, is required to set up the DMA transfer, tell it when to begin, and terminate the transfer when it is finished.

To initialize the DMA transfer, the following tasks must be performed:

- Initialize the interface address register (AR) to the first memory address
- Initialize the interface count register to the two's complement of the number of data words to be transferred
- If more than one operation is possible (**read**, write, etc.), output a command to indicate the desired function
- Enable the interrupts if the DMA transfer is to be terminated by processing an interrupt (testing the skip line is an alternate method of detecting completion).
- Start the DMA transfer (GOXX instruction).

When the block of data has been transferred, the interface signals by setting the Flag F-F. As in programmed data transfers, the Flag F-F can be used to set either the Skip F-F or the Interrupt F-F. The transfer is terminated by clearing the Flag F-F and turning off the device.

Ex 10-4

Using the DMA interface previously illustrated, set up and transfer 100 words from this A/D peripheral. Since only one device is used in this example, terminate by testing the skip line.

```
            Mnemonic            Comment
START       CLEAR41         }   Clear interface flag and and counter
            CLEAR42         }
            CLEAR           }   Initialize interface address
            ADD ADDRESS     }   register
            WRITE4Ø
            CLEAR           }   Initialize interface count
            ADD COUNT       }   register
            WRITE41         }
```

```
                GO42        Start transfer
                SKIP41      } Test skip F-F until done
                JUMP .-1    }
                CLEAR 42    } Terminate by clearing flag
                CLEAR41     } and interface counter
                HALT
ADDRESS         2ØØØ
COUNT           7634        -100(10)
```

10.3.3 Control and Status Words[2]

Occasionally a peripheral device is so complex that the three command bits of the I/O instruction are not sufficient. In this case a special register called a "Control Register" is added to the interface. Control words output to this register using the WRITEXX are decoded by the interface and used to command the peripheral.

In a similar vein, it is sometimes desirable to know more about what a device is doing than is indicated by the Flag F-F. For this purpose, a Status Register is added to the interface. The contents of this register are called the status word, and each bit is an indicator for a particular situation (e.g., re-winding tape, end-of-tape, end-of-file, etc.) The Status word can be read in at any time using the READXX instruction and examined by software.

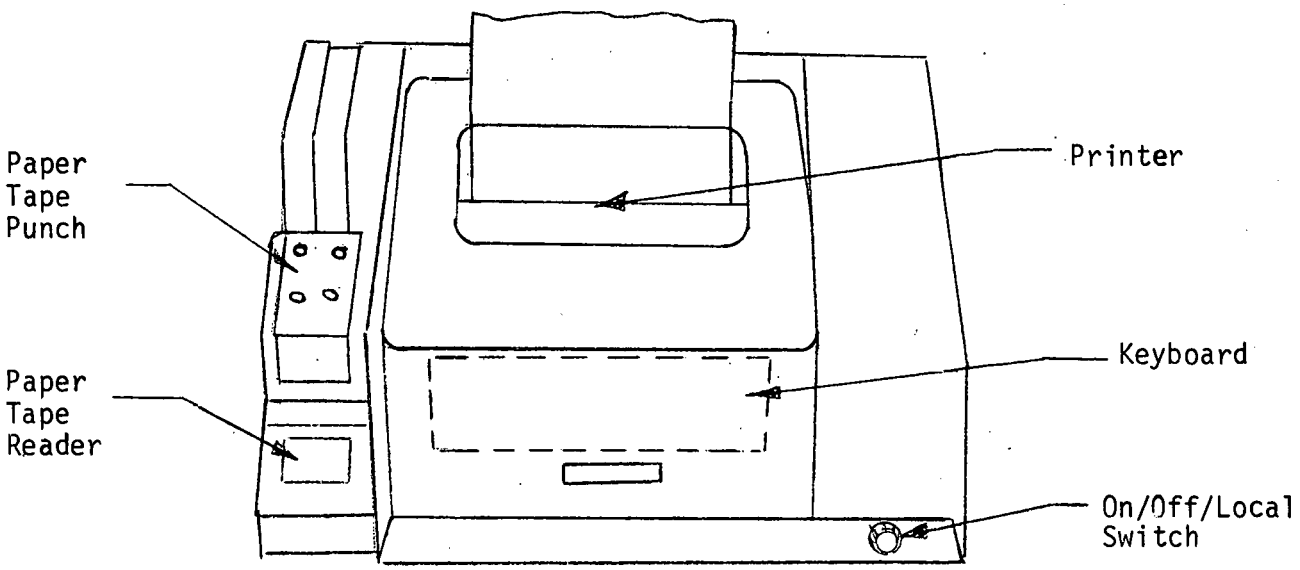
11. PERIPHERALS

11.1 Terminal Devices

The term "terminal" device designates the type of computer peripheral that permits two-way operator/machine communication and generally consists of a keyboard and a display.

11.1.1 Console Typewriter[4, 5]

Most minicomputer systems are available with an ASR-33, ASR-35, KSR-33, or KSR-35 printer/keyboard device manufactured by Teletype® Corporation. All of these models contain a typewriter-like keyboard and 10 char/sec printer, and the AXX models also contain a paper tape reader/punch. The -35 models function similarly to the -33 models but are intended for heavier usage.

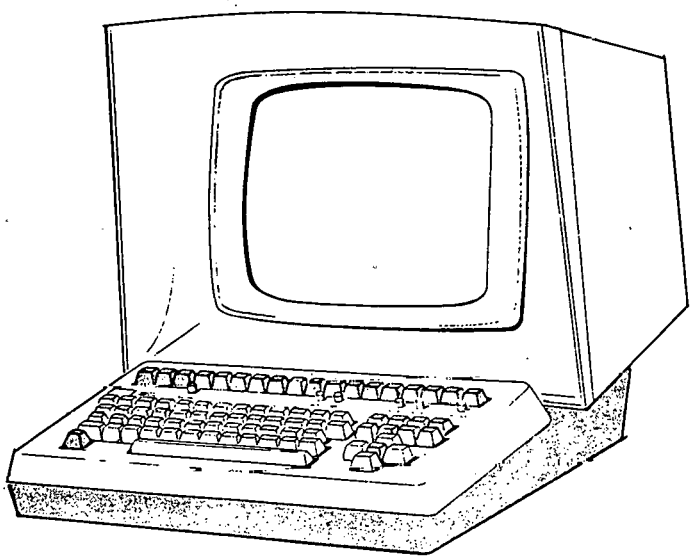


ASR-33 Teletype

These devices contain an 8-bit character buffer (register) which is either read or written into by the computer using ASCII code. Each operation (i.e., read a character, write a character, etc.) must be completed before the next is initiated. Programmed I/O using interrupts is the general type of interface provided for teletype devices.

11.1.2 CRT Consoles[5]

Because the teletype terminal is electromechanical in nature, it is relatively slow, noisy, and unreliable. Terminal devices in which a CRT screen is substituted for the printer can greatly enhance the operator interface. Both storage- and refresh-type screens are used, the latter having the greatest capability but also the greatest cost.

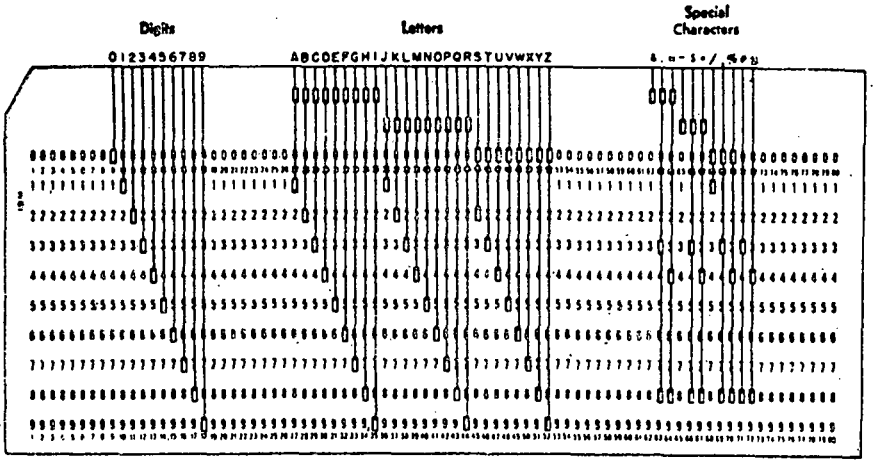


All CRT's have alphanumeric display capabilities, but only a few have graphic (point-plotting or vector) output. Storage-type CRT's can be used to generate very complex drawings, but the display must be completely redrawn to make any changes. Refresh-type screens have the opposite characteristic, in that they are limited in their vector output but permit modification of an existing image. Hard-copy generators are available for both types of CRT's.

CRT devices may be teletype-compatible (i.e., using the same interface as a teletype) or, as in the case of some refresh-type CRT's they can have very high-speed DMA-type interfaces permitting them to use CPU memory as their refresh storage buffer.

11.2 Card Reader/Punch (1)

The use of punched cards is very common in large computer systems for source program storage, and is becoming increasingly popular in mini-computer systems. The standard IBM card contains 80 columns of 12 rows each. One Hollerith character is represented in each column using the code shown on the following page (IBM 029). Binary object code is also occasionally stored on cards in a binary format.

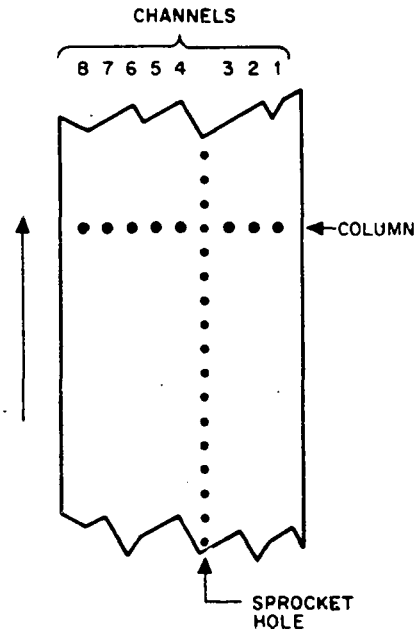


As cards are read, either electro-optical or contact-type sensors detect the punched holes in each column and transmit the data through the device controller, one character at a time, to the CPU. If Hollerith coding is being used, the controller also converts the data to an eight-bit binary code before transmitting it to the CPU. Because card readers are electro-mechanical in nature and relatively slow, programmed I/O using interrupts is the usual means of communication.

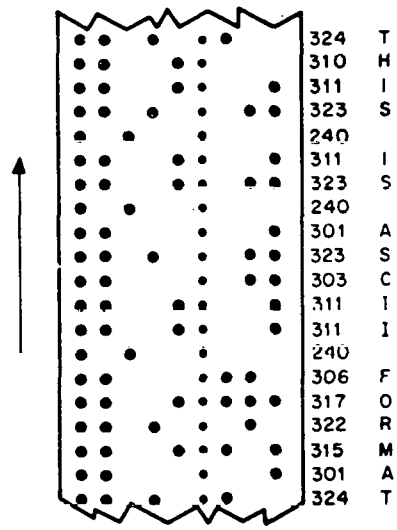
Card punches are a relatively uncommon peripheral because of their slow speed and tremendous cost and will usually only be found in large computer facilities.

11.3 Paper Tape Reader/Punch[4, 5]

Punched paper tape can be read and generated via an ASR-XX teletype device, or by separate high-speed peripherals. Data is punched on paper tape in 8 channels which run along the length of the tape with a set of sprocket holes.



The columns across the tape contain the coded information. If ASCII format is being used, one Hollerith symbol is represented per column as shown here:



ASCII FORMAT(5)
The USA Standard Code for Information Interchange (ASCII) format uses all eight channels of the paper tape to represent a single character (letter, number, or symbol) as shown in the diagram at left.

Other codes, using two or more adjacent columns per datum, are used when binary data must be represented.

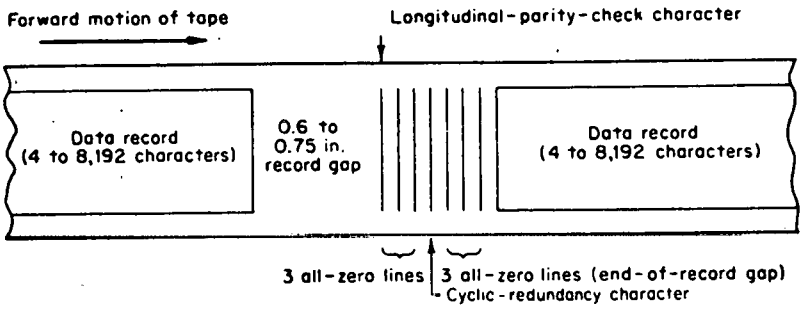
In terms of the computer, paper tape readers and punches are relatively slow (625 char/s and 100 char/s, respectively), so programmed I/O is used with format conversions taking place in the device interface as necessary.

11.4 Magnetic Tape[4]

Data is stored on 1/2-in. IBM compatible magnetic tape in columns of 7 or 9 tracks much the same way as punched paper tape. The data is read from or written to magnetic tape by a DMA transfer from/to a CPU memory buffer area. On tape this buffer is referred to as a record and is represented as a sequence of 6- or 8-bit characters (bytes). The characters are packed at a fixed density, typically 556 or 800 bytes per inch (BPI). The product of the tape's speed and density give the total data transfer rate, e.g.,

800 BPI x 45 IPS = 36000 char/sec

Each 6- or 8-bit character is augmented by another bit called the transverse parity bit. If the basic character contains an odd number of logic ones, the parity bit is set to 1, otherwise it's 0. This is referred to as even parity, and it provides a quick way of checking for 1, 3, 5, etc., incorrect bits.



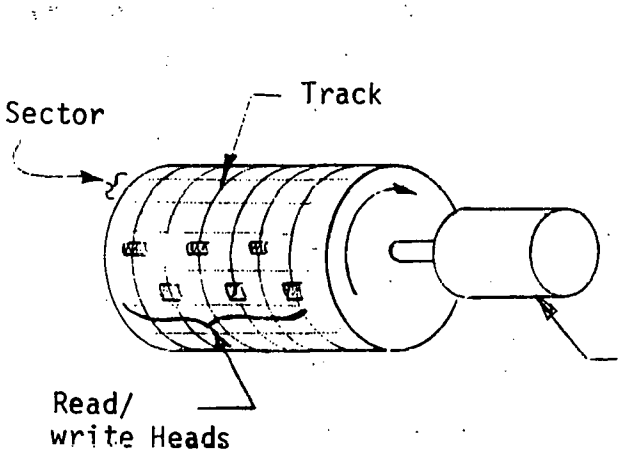
The end of each record is delineated by the format shown above. The longitudinal-parity-check character is just another means of detecting errors, this time checking for errors along each channel over the entire record. A file is a grouping of many records defined by a programmer for use in his application. To denote the end of a file (EOF), the inter-record gap is followed by a 3-in. gap, an EOF character, and a longitudinal-parity-check character.

Magnetic tape devices other than IBM compatible tape are often used on small computer systems when communication to a large data processing center is not required. Common systems are 3/4-in. DEC tape, LINC tape, and various types of cassettes.

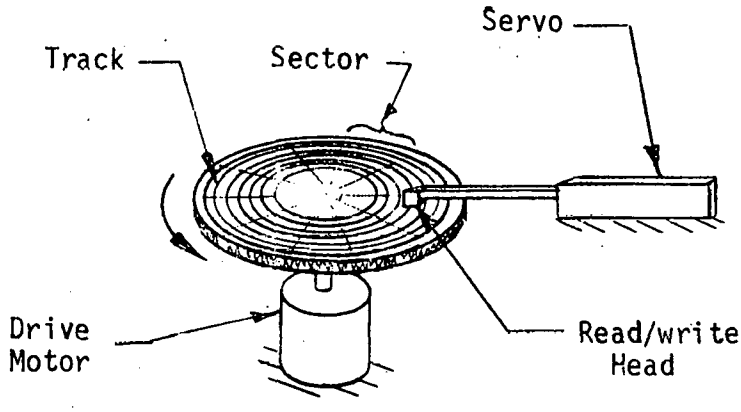
11.5 Magnetic Disc, Drum[4]

A magnetic disc or drum is a common type of mass storage device, the disc being more commonly found on small computers. It can be used to store source code, object code, and numerical data, as can magnetic tape, but has the advantage of being a random-access type device. That is, any particular place in storage can be accessed directly, without having to sort through the recording medium.

In discs or drums the storage medium is the same--data is recorded magnetically on the coated surface of a revolving platter (disc) or drum. The surface is considered to be divided up into tracks and sectors. In a fixed-head design, there is one read/write head per track whereas in the moving head design



Drum with Fixed Heads



Disc with Moving Head

a single read/write head is positioned from one track to another by a servo system. The latter is much cheaper (factor of .5X) but also much slower (5X) in accessing a given track. Once positioned, however, the actual data transfer rate is a function of only the storage density and surface speed. Data transfer rates of 10^5 16-bit words/sec are typical. Many moving-head designs incorporate a removable disc cartridge or disc pack so that the effective storage size becomes infinite.

A new device just appearing on the market is the "floppy" disc. The system combines the advantages of a disc (random access) and magnetic tape (low cost per word of stored data, infinite medium). The disc is a platter of mylar recording tape, about the size of a 45-rpm record, protected by a paper jacket with access openings for the read/write head and drive mechanism. The read/write head in the disc drive is movable and is pressed into contact with the disc by a pressure pad. Although the floppy disc is slower and has less storage/disc than its big brothers, it's inexpensive and convenient, big factors for the minicomputer user.

We have discussed a number of mass-storage type devices. Each of these has its own advantages and disadvantages and are compared in the table on the following page.

11.6 Printers

In this section we will discuss printers--relatively high-speed devices which produce alphanumeric hardcopy on paper a line at a time. Line printers are of two general types--impact and nonimpact.

In impact printers the paper moves past a type slug on a chain or drum, and the type strikes the paper. The image is transferred by means of an inked ribbon or ink-impregnated paper. Other impact printers use a solenoid-driven matrix of pins to create the characters in the form of a dot matrix. A wide range of speeds is possible, from 100 to 2500 lines per minute (lpm)

Many different types of nonimpact printers are available--ink sprayers, electrographic, thermal, electro-optical, and electrostatic. All have relatively silent operating characteristics as compared to impact printers, but can only generate one copy per run. Also, most of these devices are medium-speed printers (300-1000 lpm); however, LLL's Radiation, Inc. electrical discharge printer can operate at 20,000 lpm. Oftentimes nonimpact printers require treated paper, thereby increasing operating costs over impact printers. For many small computer systems, however, the disadvantages of the nonimpact printers are minor compared to their features of quietness, compactness, convenience, and reliability. In the case of electrostatic printers, high-speed graphic output is also possible, a considerable asset in real-time data-acquisition and control systems.

COMPARISON OF MINICOMPUTER INPUT/OUTPUT STORAGE MEDIA AND PERIPHERALS[4]

	PAPER TAPE		MAGNETIC TAPE Synchronous (continuous) operation			DISC/DRUM		
	ASR-33 tele-typewriter	Medium-speed reader/punch	Cassette/ Cartridge	DECtape	Standard (IBM-compatible) tape	Fixed Head	Moving Head	Floppy Disc
16-bit words/sec	5	150 READ 25 PUNCH	250 - 2,500	7,500 (12-bit words) 5,000 (18-bit words)	3,000 - 30,000	250,000	100,000	15,0000
Access time	0	0	12 - 150 sec	up to 100 sec	up to 240 sec	8 - 17 msec avg	30 - 70 ms avg	100 ms avg
Total storage (16-bit words in one unit)	-	50,000	50K - 250K	100K 18-bit words or 150K 12-bit words	200K - 2M	30K - 2M	.5M - 24M	64K - 130K
Typical price (combined input/ output unit and interface)	$300 more than KSR-33	$3,000	$300 - $3,000	$3,400 - $9,700	$5,000 - $13,000	$10,000- $40,000	$10,000- $35,000	$2000 - $4000

11.7 Plotters

Digital plotters serve a wide variety of functions--from simply displaying graphically the results of a calculation, to generating the precision masks required for the fabrication of LSI circuitry.

Hardware-wise, plotters fall into three general categories: drum, flat-bed, or electrostatic. In the drum-type plotter the paper is held on a drum by sprockets and moved incrementally past a pen cross-slide to generate the X-motion. The pen is positioned incrementally on the cross-slide to give the Y-motion. On a flat-bed plotter the paper is fixed to the bed while the pen is positioned incrementally in both the X and Y directions. Resolution for both the drum and flat-bed plotters is on the order of 0.005 in. in smaller sizes. Drum-type plotters are generally available up to 36-in. diameter, while flat-bed plotters range up to 8 ft x 12 ft. The "pen" can be ballpoint, ink pen, cutting stylus, or even a laser.

Electrostatic plotters operate by passing the paper incrementally past a row of closely spaced ($\approx$ 0.010") nibs extending across the full width of the paper. As the paper moves, the appropriate nibs are pulsed, causing the coated surface of the paper to become charged. Passing the paper through a toner medium then results in a black "dot" at each charged point. The complete plot is generated in one pass as the paper moves past the nibs. This makes for very fast plotting ($\approx$ 1.25 in./sec), but the software becomes much more complex because the complete "picture" must be generated and stored digitally in the computer before plotting begins. Because the mechanism for plotting and line-printing is the same for electrostatic devices, the two functions are often combined in a single peripheral called a printer/plotter. Paper widths up to 22 in. are currently available.

11.8 Hybrid and Other Peripherals

Practically every instrument or device imaginable has been interfaced to a computer at one time or another. General purpose A/D and D/A subsytems have already been discussed. Variations along this same idea include transient digitizers, programmable current sources, digital waveform synthesizers, synchro-to-digital converters, plus many others.

Some radically different hybrid devices include graphic digitizers, TTL logic level I/O, contact closure sensors and analog computers. In some large computer systems a minicomputer will serve as a "front-end" communications processor where its entire function is receiving, formatting, and transmitting data between remote terminals and the larger computers. In these circumstances

the small computer is a "peripheral" to the larger, and the remote terminals are "peripherals" to the mini.

12. SYSTEM SOFTWARE

12.1 Operating Systems

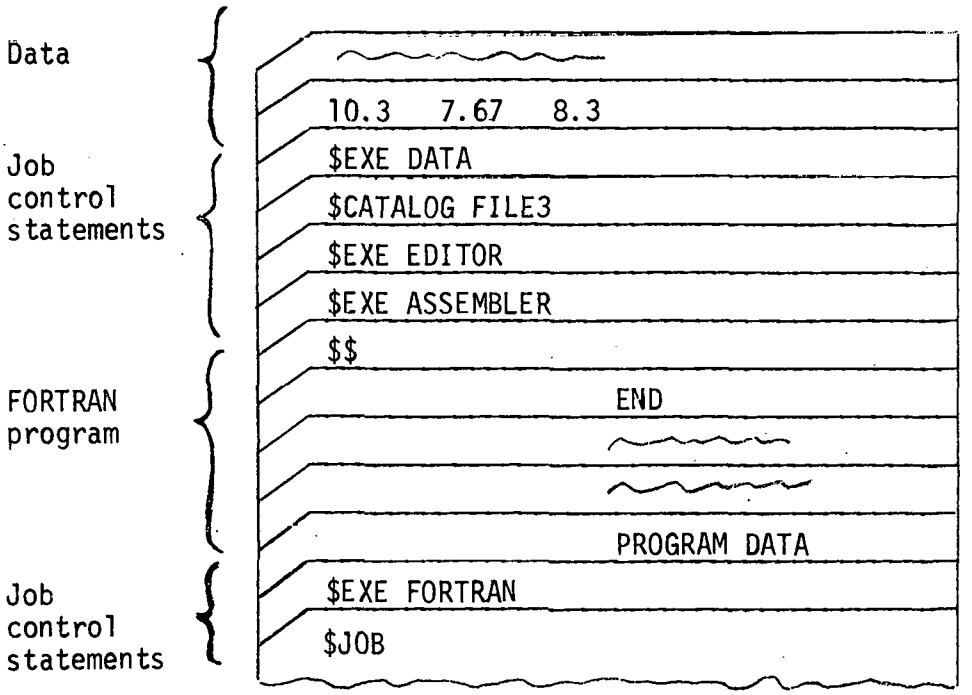
An Operating system (OS) is an assembly language program written to control the execution of other programs. It can be a very powerful program in that it greatly speeds up the routine compiling, assembling, altering, and storing of user programs. Some of the tasks it can perform are as follows:

- Call into execution other system processors such as the FORTRAN compiler, ASSEMBLER, EDITOR, etc., by simple typed commands
- Keep track of where and how programs or data are stored on a mass storage device
- Schedule applications program for execution at various times of day.
- Assign priorities to programs and arbitrate priorities among programs scheduled for execution at the same time.
- Provide simple calls usable by application programs for the I/O to various devices (i.e., handlers are generally part of the OS).

As small computers have developed hardware-wise in recent years, so have the operating systems available for them. The following paragraphs describe the characteristics of various types of operating systems.

12.1.1 Batch OS

A batch operating system carries out its assigned tasks one job at a time. Once the operator gives the command to perform some function, the computer is wholly occupied doing that job. When completed, the OS comes back and asks the operator for the next command. The following represents a typical card deck for compiling, assembling, editing, and executing a FORTRAN program.



12.1.2 Real-Time OS

As compared to batch OS's, the real time OS can activate more than one program at a time, can accommodate priorities, and is able to schedule jobs according to the time of day. The computer operator instructs the OS through commands input through the teletype. Suppose a program called "TEST1" is to be started at 2:30 p.m. and requires 20 minutes to run. Also, "TEST2" is to be started at 2:45 p.m., and although it requires only 30 sec to run, it must not be delayed from starting. The operator commands to do this are

```
/START    TEST1    14:30:00    P2
/START    TEST2    14:45:00    P1
```

Start-up command — Program names — Start time — Assigned priorities

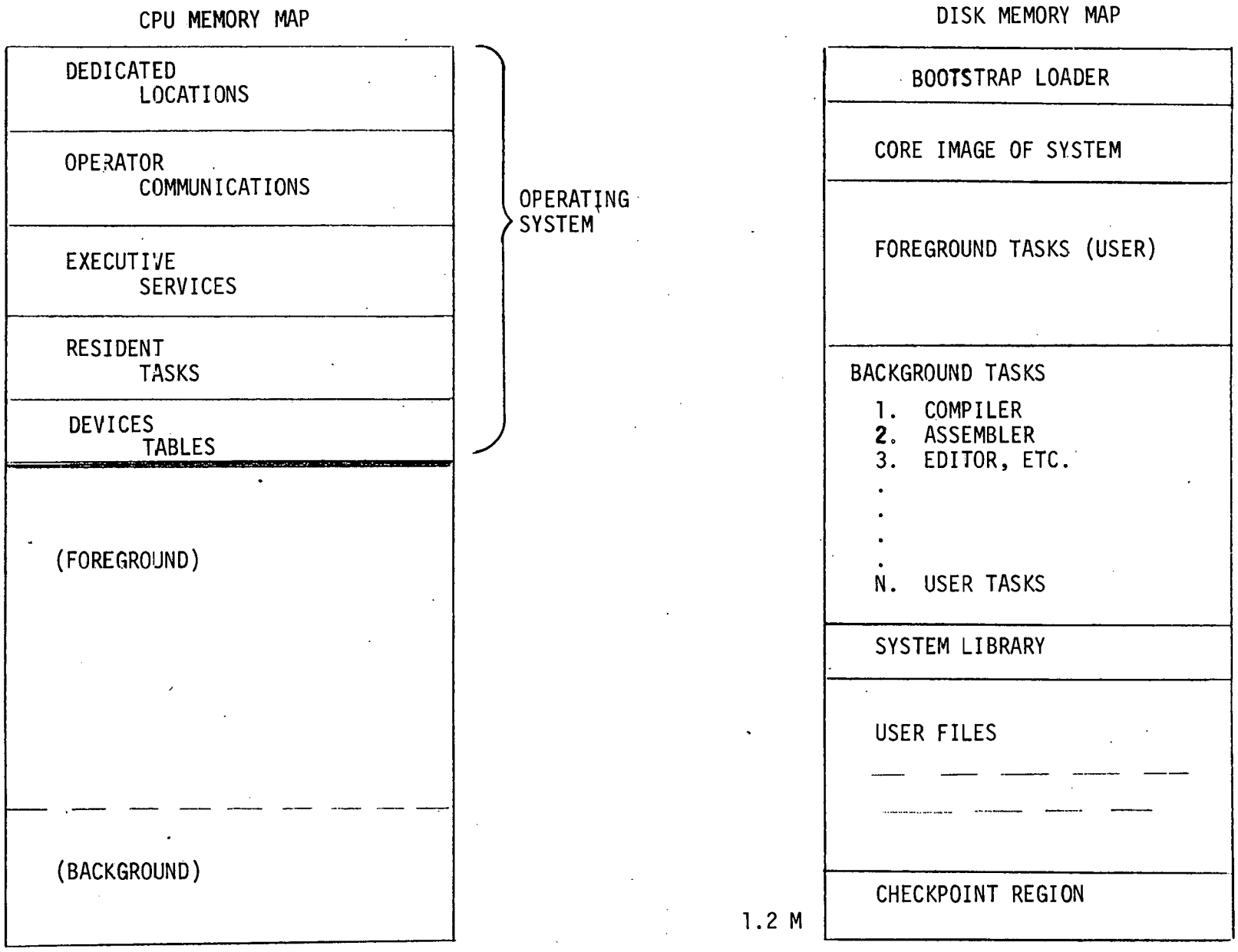
The fact that TEST2 is assigned a higher priority than TEST1 assures that it will have control of the machine any time it needs it.

12.1.3 Time Sharing OS

In a time sharing OS there are many "simultaneous" users of the computer. Each user feels that he has full control of the machine and issues job control commands similar to single-user batch systems. In actuality, the computer can execute the statements of only one program at a time. Therefore execution must jump from one task to another for brief periods of time so that to each user his job appears to be executing, albeit somewhat slowly. In a sophistication of these systems, priority and weighting techniques are used to control the length of time that any one user gets control of the machine. The Livermore Time Sharing System is an example of the latter.

12.1.4 Foreground-Background OS

Several mini-computer system vendors offer foreground-background OS's. The "foreground" consists of one or more programs which can be active simultaneously and executed on a priority basis as in a real-time OS. The "background" supports only one user who operates in a batch fashion, but at an effective priority less than any foreground task. Foreground tasks must be debugged real-time programs, whereas compilations, assembling, editing, etc., can be performed only in the background. The utility of such a system is in a process-control or data acquisition computer system where application software development must continue after the computer goes on-line. Debugged applications programs are run as foreground tasks while continuing software development is done in the background as time is available. In this way software development does not interfere with the critical timing of on-line systems. Core- and disk-memory "maps" for a typical foreground-background system are shown on the following page.



CPU AND DISK "MAPS" OF A TYPICAL FOREGROUND/BACKGROUND REAL-TIME OPERATING SYSTEM.

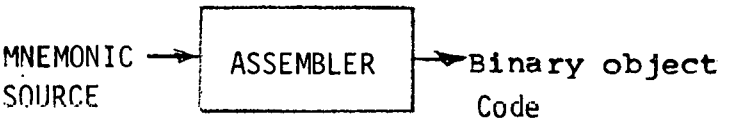
12.1.5 Disk Operating System (DOS)

Any operating system which is capable of making use of a magnetic disk or drum storage device.

12.2 Language Processors

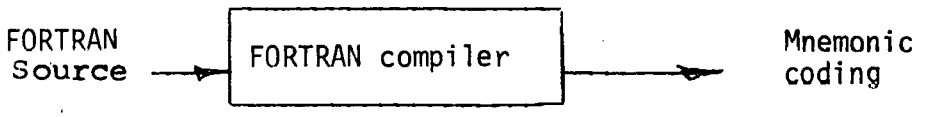
12.2.1 Assembler

The Assembler is a program written directly in binary object code which translates mnemonic source code into equivalent binary object. Assembly language varies from one computer model to another, therefore programs written in this language are compatible with only a limited number of machines.



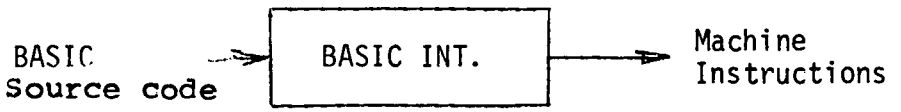
12.2.2 FORTRAN Compiler

The FORTRAN compiler is a program written in assembly language which is used to translate FORTRAN source codes into equivalent mnemonic coding. The mnemonic coding must be assembled before executable binary code is obtained. FORTRAN is a fairly well standardized language and therefore programs written in FORTRAN are considered "machine transportable," that is, not written for a particular computer. The compiler, however, is not machine independent.



12.2.3 BASIC Interpreter

BASIC is a simple high-level language first developed at Dartmouth College. It is easy to learn by inexperienced personnel. As compared to the FORTRAN compiler which digests an entire source code and then outputs mnemonic code which must be assembled before it can be executed, the Basic interpreter takes one line of source code at a time, translates it to machine instructions, and executes it immediately. This greatly speeds up the coding process since it becomes more interactive, but at the expense of the running time of the finished code.

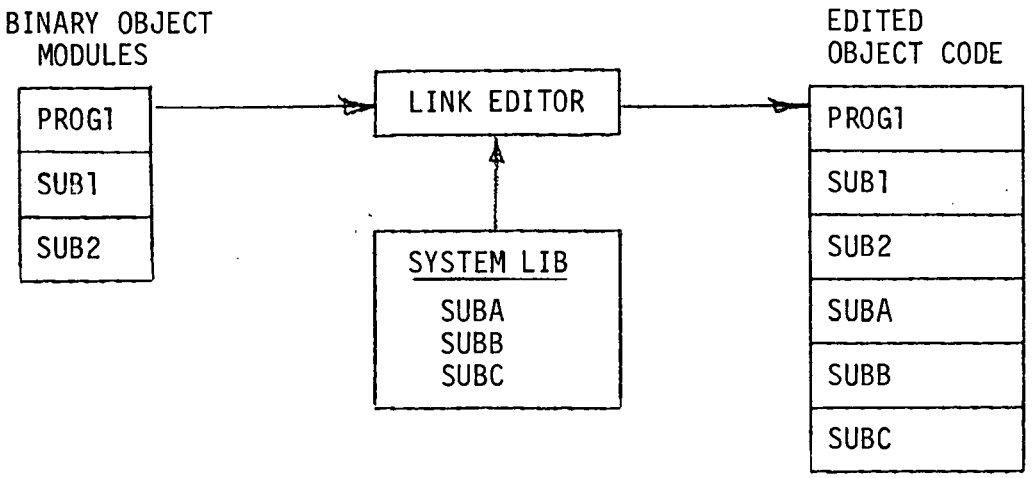


12.3 Utility Routines

12.3.1 LINK EDITOR

The LINK EDITOR is a program which links object modules which have been assembled separately. References to system subroutines (e.g., math library), or a user's library are satisfied at this point. The EDITOR outputs

a ready-to-run object code with all addresses defined and references to external subroutines resolved. Generally the EDITOR also provides a "memory map," a detailed listing of program names and where they would be loaded in memory in the final object code. The object code itself is stored on some mass memory device such as disk, paper tape, etc.



11.3.2 TEXT EDITOR

This program provides a means of manipulating ASCII symbols. It can therefore be used for altering FORTRAN, BASIC, or ASSEMBLER source code, as well as editing text in the journalistic sense.

11.3.3 CATALOGER

The CATALOGER program provides an organized means of storing edited object codes on some mass storage medium. The program maintains a directory of all cataloged modules and keeps track of their size and location for easy access and/or alteration.

11.3.4 DEBUG

The DEBUG program provides a set of commands that are useful in debugging binary object code. The DEBUG processor can generate dumps (a tabulation of the contents of specific memory locations), modify memory or register contents, and various other functions needed in developing software.

REFERENCES

1. Malmstadt, H. V., and Enke, C. C., "Digital Electronics for Scientists," W. A. Benjamin, Inc., New York, N. Y., 1969, 545 p

2. Soucek, B., "Minicomputers in Data Processing and Simulation," John Wiley and Sons, Inc., New York, N. Y., 1972, 467 p

3. Peatman, J. B., "The Design of Digital Systems," McGraw Hill, New York, N. Y., 1972, 457 p

4. Korn, G. A., "Minicomputers for Engineers and Scientists," McGraw Hill, New York, N. Y., 1973, 303 p

5. "Introduction to Programming," Digital Equipment Corporation, Maynard, Mass., 1973

6. "Transistor Manual," General Electric Corporation, Syracuse, N. Y., 1974, 660 p

7. Burlingame, B. G., "The Control Engineers Guide to Integrated Circuit Applications, Part I, Digital Integrated Circuits," Control Engineering, June, 1971, pp 67-79

8. Davis, S., "Linear IC's for Digital Interface Applications, Part I: Monolithic Amplifiers," Computer Design, July 1973, Vol 12, No. 7, pp 59-69

9. "Analog-Digital Conversion Handbook," Analog Devices, Norwood Mass., 1972

10. "A Pocket Guide to the Hewlett-Packard Computers," Hewlett-Packard Company, Palo Alto, CA, 1970

11. "American Standard FORTRAN," American Standards Association, X3.9-1966, New York, N. Y., 1966, 38 pp

DISTRIBUTION

LLL Internal Distribution

Arnold, W. F.	L-123	
Fisher, D. K.	L-115	(10)
Dennis, G. C.	L-123	(115)
Spies, R. J.	L-40	
Strange, F. M.	L-424	
Westbrook, R. W.	L-123	
TID	L-9	(5)

☆ GPO 790-809