1-1

A B C	A·B·C	(A·B·C)'	A'	B'	C'	A'+B'+C'
0 0 0	0	1	1	1	1	1
0 0 1	0	1	1	1	0	1
0 1 0	0	1	1	0	1	1
0 1 1	0	1	1	0	0	1
1 0 0	0	1	0	1	1	1
1 0 1	0	1	0	1	0	1
1 1 0	0	1	0	0	1	1
1 1 1	1	0	0	0	0	0

1-2

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

1-3

(a) $A + AB = A(1+B) = A$

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

(c) $A'BC + AC = C(A'B+A) = C(A'+A)(B+A) = (A+B)C$

(d) $A'B + ABC' + ABC = A'B + AB(C'+C) = A'B + AB =$
$$= B(A'+A) = B$$

1-4

(a) $AB + A(CD + CD') = AB + AC(D+D') = A(B+C)$

(b) $(BC' + A'D)(AB' + CD') =$
$$= \underset{0}{\underline{ABB'C'}} + \underset{0}{\underline{A'AB'D}} + \underset{0}{\underline{BCC'D'}} + \underset{0}{\underline{A'CD'D}} = 0$$

$\underline{1\text{-}5}$ (a) $(A+B)'(A'+B')' = (A'B')(AB) = 0$

(b) $A + A'B + A'B' = A + A'(B+B') = A + A' = 1$

$\underline{1\text{-}6}$ $F = x'y + xyz'$

(a) $F' = (x+y')(x'+y'+z) = x'y' + xy' + y' + xz + y'z$
$= y'(1 + x' + x + z) + xz = y' + xz$

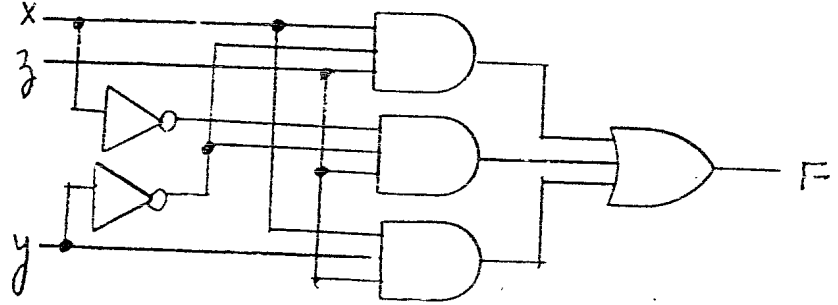(b) $F \cdot F' = (x'y + xyz')(y' + xz) = 0 + 0 + 0 + 0 = 0$

(c) $F + F' = x'y + xyz' + y' + xz(y+y')$
$= x'y + xy(z'+z) + y'(1+xz) = x'y + xy + y'$
$= y(x'+x) + y' = y + y' = 1$

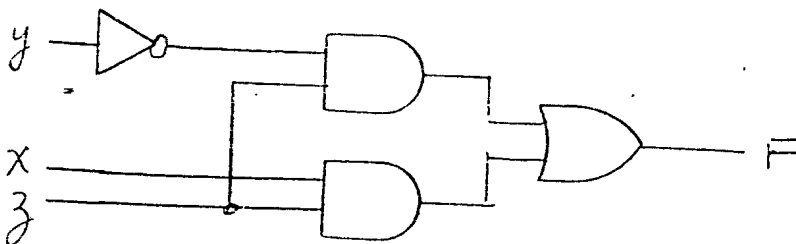$\underline{1\text{-}7}$

(a)

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

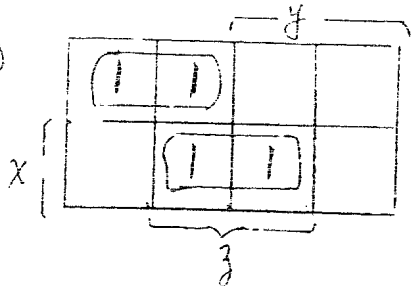(b) $F = xy'z + x'y'z + xyz$

(c) $F = xy'z + x'y'z + xyz$
$= y'z(x+x') + xz(y+y')$
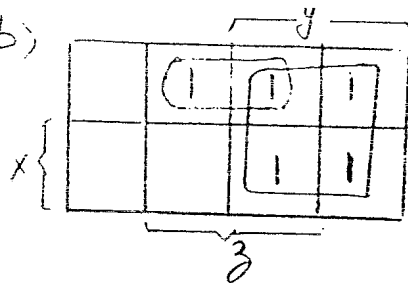$= y'z + xz$

(d) Same as (a)
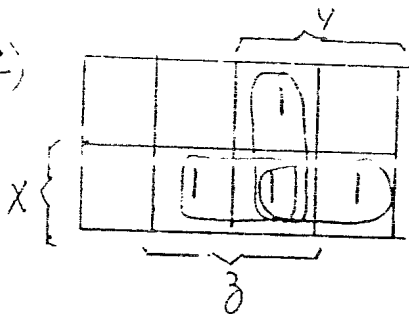

2

$\underline{1-8}$

(a)

$$F = x'y' + xz$$

(b)
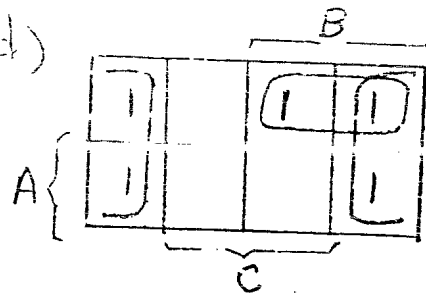

$$F = y + x'z$$

(c)

$$F = xy + xz + yz$$

(d)

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

$\underline{1-9}$

(a)
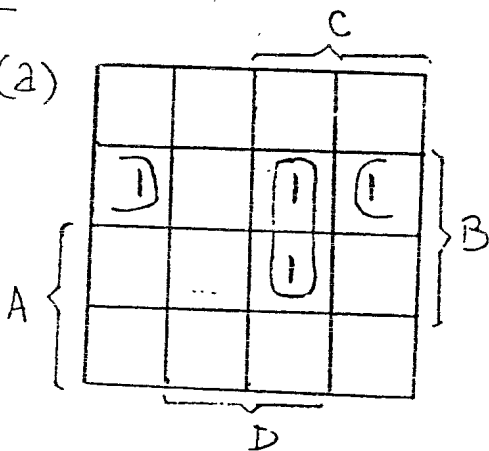
$$F = BCD + A'BD'$$

(b)
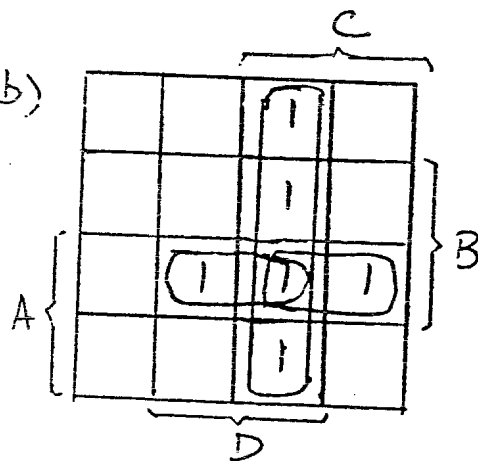
$$F = CD + ABC + ABD$$

(c)
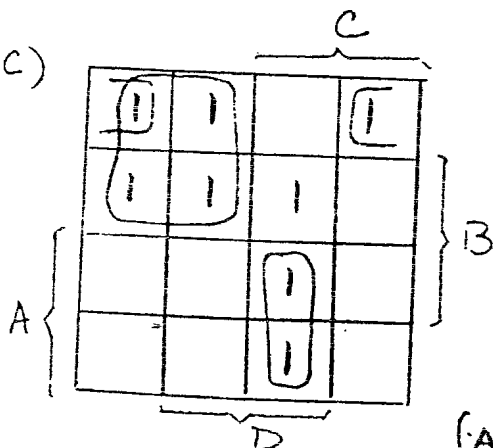
$$F = A'C' + A'B'D' + ACD + \begin{cases} A'BD \\ or \\ BCD \end{cases}$$

(d)

$$F = BD + B'D' + \begin{cases} A'B \\ or \\ A'D' \end{cases}$$

3

1-10

(a)
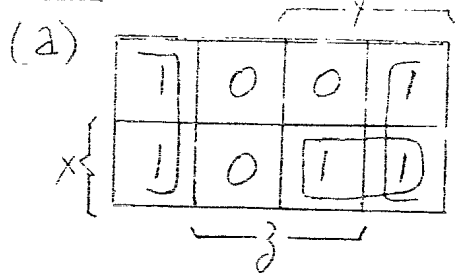
(1) $F = xy + z'$

$F' = x'z + y'z$

(2) $F = (x + z')(y + z')$

(b)
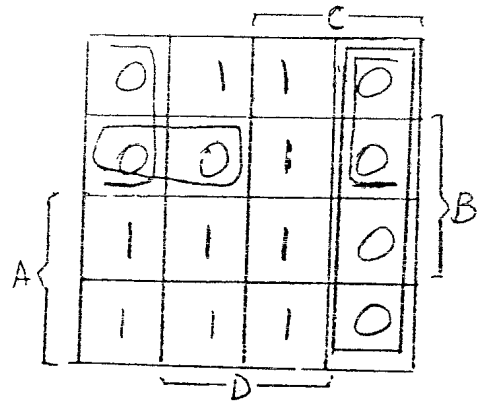
(1) $F = AC' + CD + B'D$

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

1-11
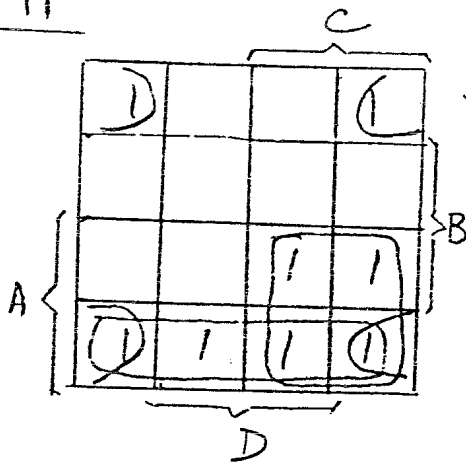
(a)
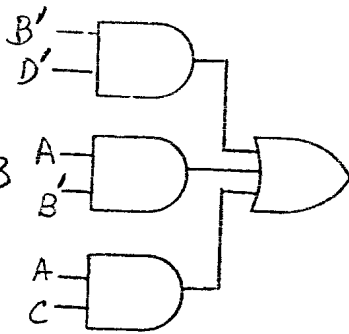
(b)
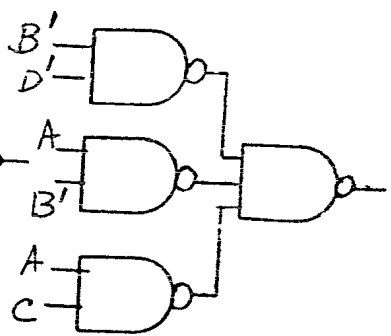
$F = B'D' + AB' + AC$

1-12
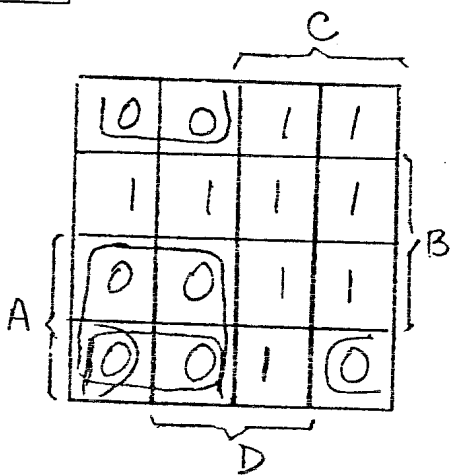
$F' = AC' + BC' + AB'D'$

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

4
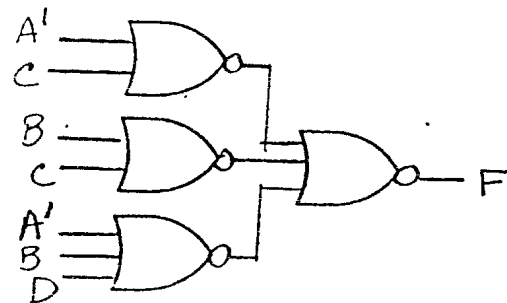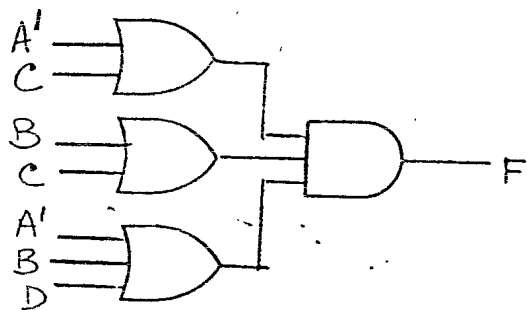
1-13
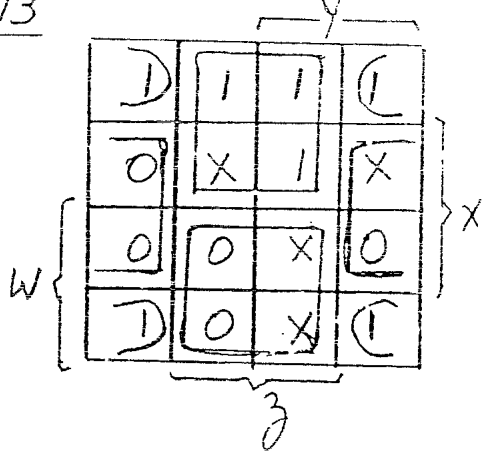
(a) $F = x'z' + w'z$

(b) $= (x'+z)(w'+z')$

1-14

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

$= x'(y'z + yz') + x(y'z' + yz)$ ⟵ See Fig 1-2 (Exclusive-NOR)

$= x'(y \oplus z) + x(y \oplus z)'$

$= x \oplus y \oplus z$

1-15

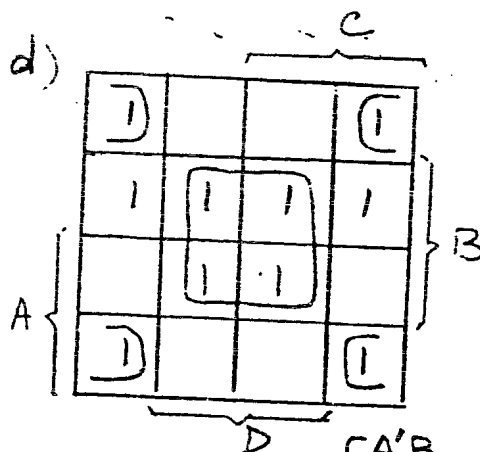| X	Y	Z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1
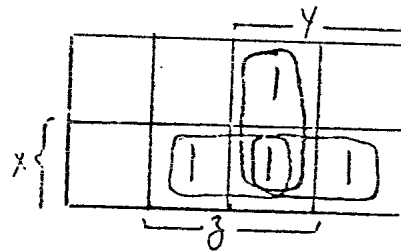
$F = xy + xz + yz$

1-16

X	Y	Z	A	B	C
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	1
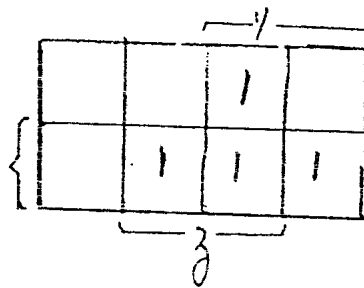
$A = xy + xz + yz$
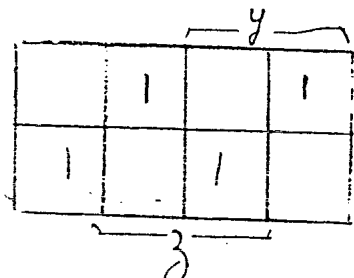
$F = x \oplus y \oplus z$

$C = z'$

By inspection
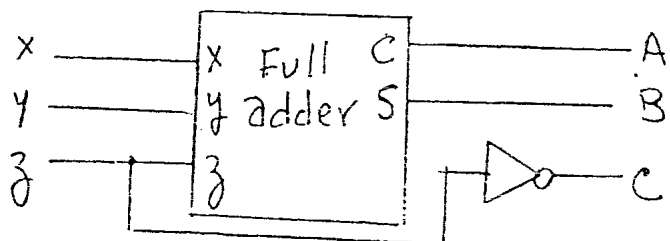
1-17
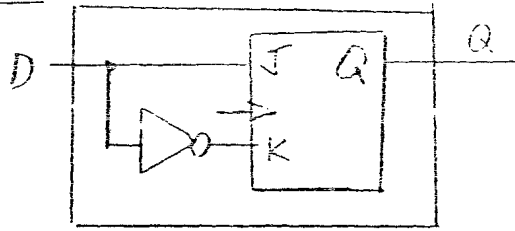
When $D=0$; $J=0$, $K=1$, $Q \rightarrow 0$

When $D=1$; $J=1$, $K=0$, $Q \rightarrow 1$

1-18

See text; Section 1-6 for derivation.

1-19 $\quad D_A = x'y + xA$; $\quad D_B = x'B + xA$; $\quad z = B$

(a)
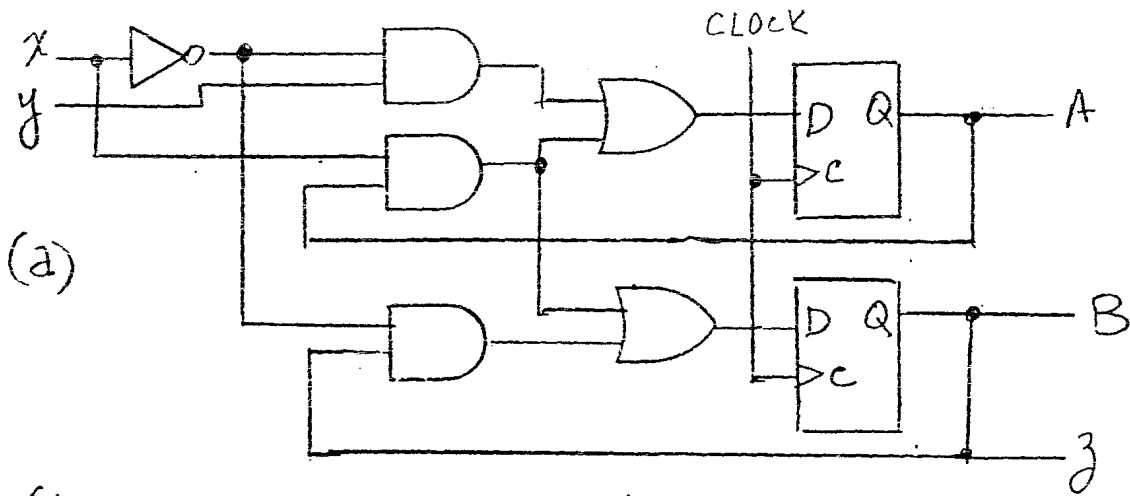
(b)

Present State		Inputs		Next state		Output
A	B	x	y	A	B	z
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	0	0	0	0
0	0	1	1	0	0	0
0	1	0	0	0	1	1
0	1	0	1	1	1	1
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	0	0	0
1	0	0	1	1	0	0
1	0	1	0	1	1	0
1	0	1	1	1	1	0
1	1	0	0	0	1	1
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

1-20

$$J_A = K_A = x$$
$$J_B = K_B = x'y$$
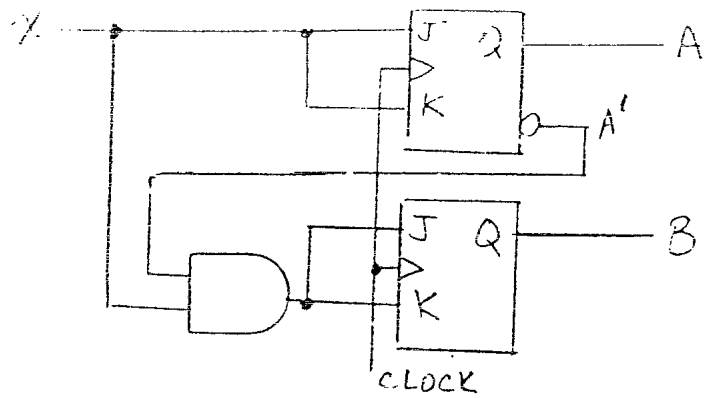
1-21   Count up-down binary counter with enable E

Present state		Inputs		Next state		Flip-flop inputs			
A	B	E	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	X	0	X
0	0	0	1	0	0	0	X	0	X
0	0	1	0	1	1	1	X	1	X
0	0	1	1	0	1	0	X	1	X
0	1	0	0	0	1	0	X	X	0
0	1	0	1	0	1	0	X	X	0
0	1	1	0	0	0	0	X	X	1
0	1	1	1	1	0	1	X	X	1
1	0	0	0	1	0	X	0	0	X
1	0	0	1	1	0	X	0	0	X
1	0	1	0	0	1	X	1	1	X
1	0	1	1	1	1	X	0	1	X
1	1	0	0	1	1	X	0	X	0
1	1	0	1	1	1	X	0	X	0
1	1	1	0	1	0	X	0	X	1
1	1	1	1	0	0	X	1	X	1

$$J_A = (Bx + B'x')E$$

$$K_A = (Bx + B'x')E$$

$$J_B = E$$

$$K_B = E$$

7
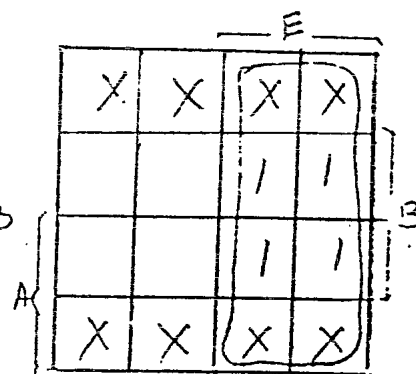
2-1   TTL IC

   (a) Inverters – 2 pins each $12/2 = 6$ gates   7404
   (b) 2-input XOR – 3 pins each $12/3 = 4$ gates   7486
   (c) 3-input OR – 4 pins each $12/4 = 3$ gates
   (d) 4-input AND – 5 pins each $12/5$ 2 gates   7421
   (e) 5-input NOR – 6 pins each $12/6 = 2$ gates   74260
   (f) 8-input NAND – 9 pins   1 gate   7430
   (g) JK flip-flop – 6 pins each $12/6 = 2$ FFs   74107

2-2

   (a) 74155 – Similar to two decoders as in Fig 2-2.
   (b) 74157 – Similar to multiplexers of Fig. 2-5.
   (c) 74194 – Similar to register of Fig. 2-9.
   (d) 74163 – Similar to counter of Fig. 2-11.

2-3

2-4

$$D_0 = (A_1 + A_0 + E')' = A_1'A_0'E$$

$$D_1 = (A_1 + A_0' + E')' = A_1'A_0 E$$

$$D_2 = (A_1' + A_0 + E')' = A_1 A_0' E$$

$$D_3 = (A_1' + A_0' + E')' = A_1 A_0 E$$
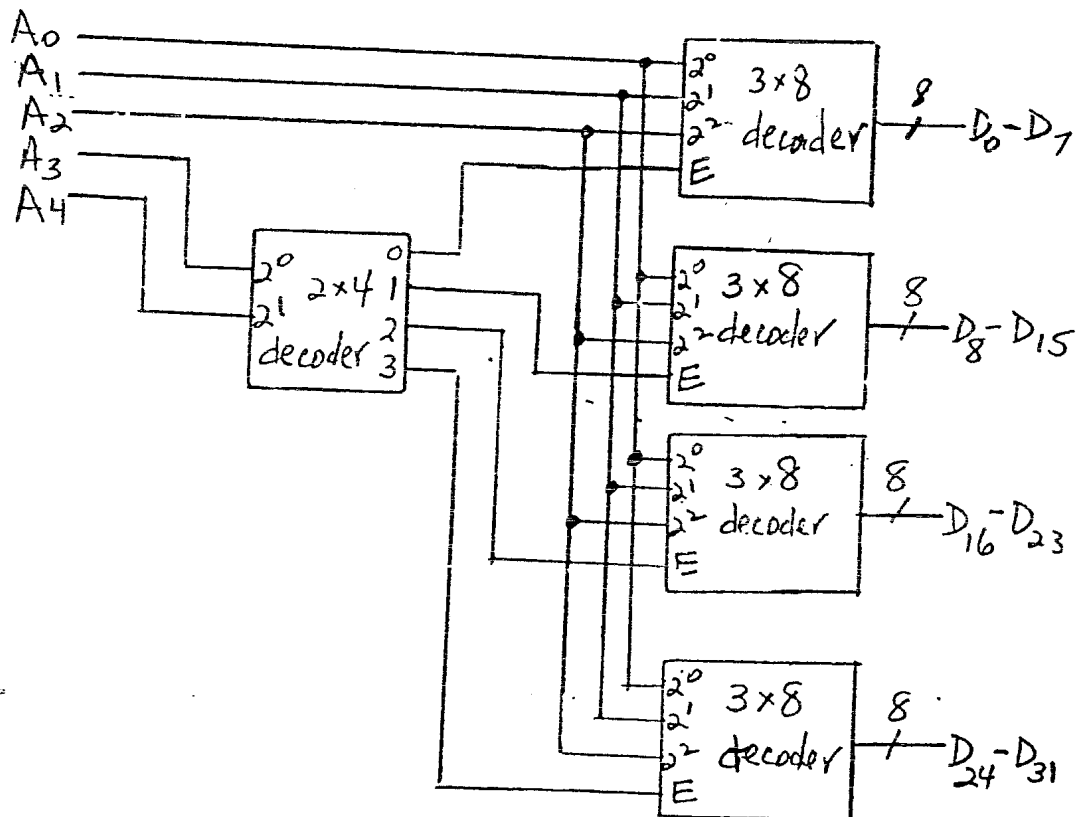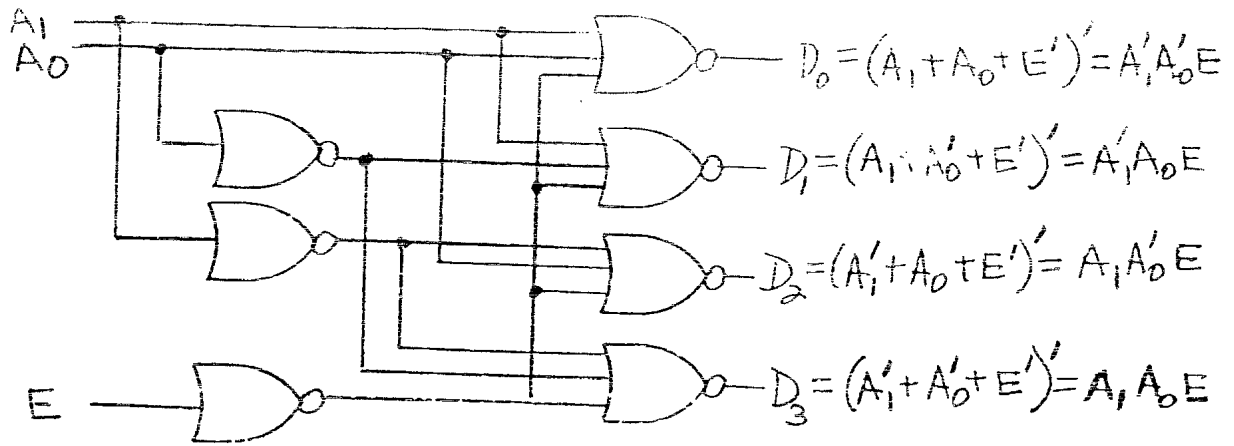
2-5 Remove the inverter from the E input in Fig. 2-2(a).

2-6
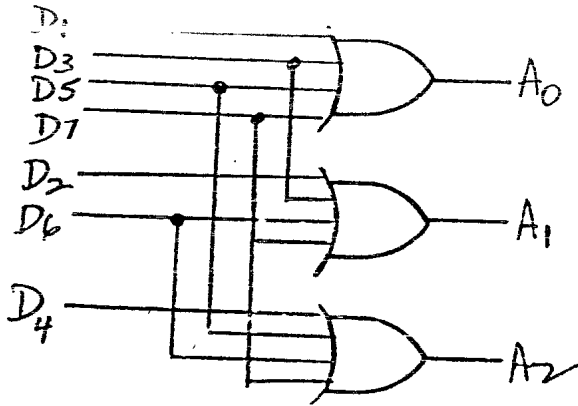
If all inputs equal 0
or if only $D_0 = 1$:
the outputs $A_2 A_1 A_0 = 000$

Needs one more output
to recognize the all
zeros input condition.

2-7

2-8

Select
A_0
A_1
A_2
A_3

B_0
B_1
B_2
B_3

S_1 S_0 Dual
A 4×1
MUX

Y_A

Y_B
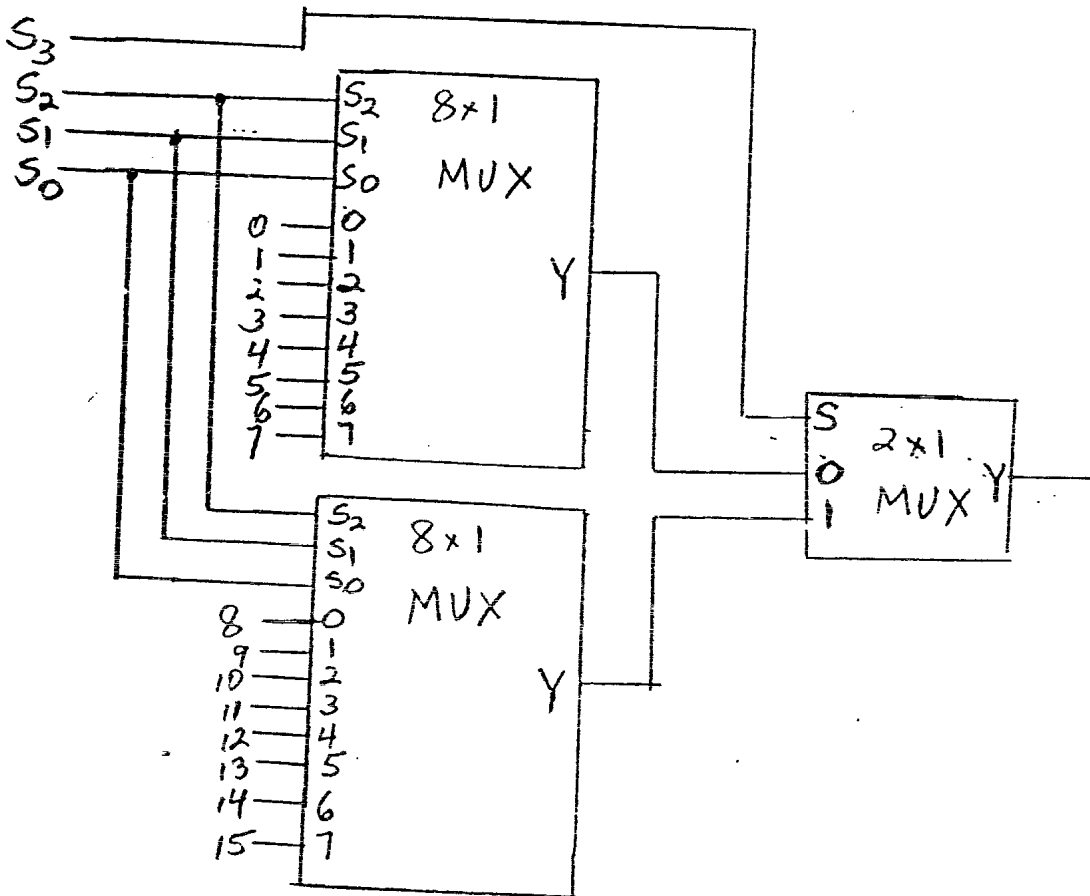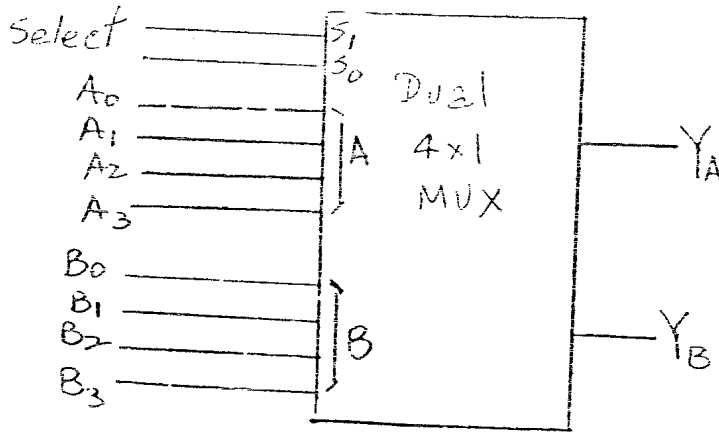
$S_1 S_0$	$Y_A Y_B$
0 0	$A_0 B_0$
0 1	$A_1 B_1$
1 0	$A_2 B_2$
1 1	$A_3 B_3$

Function table

2-9

"Parallel load"
Clock pulses

Clock Register
Fig. 2-6
I_0
I_1
I_2
I_3

A_0
A_1
A_2
A_3
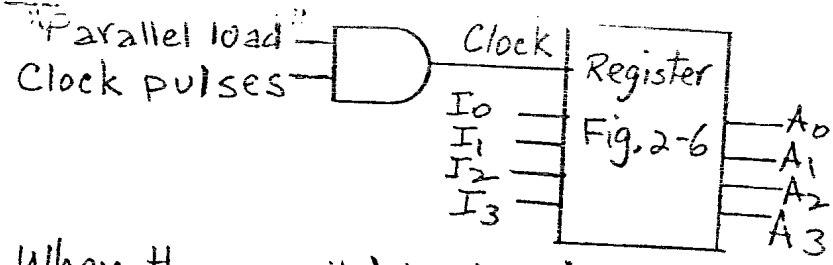
When the parallel load input=1, the clock pulses go through the AND gate and the data inputs are loaded into the register. When the parallel load input =0, the output of the AND gate remains at 0.

2-10

The buffer gate does not perform logic. It is used for signal amplification of the clock input.

2-11

Load

I_0

clear

D Q

>C

A_0
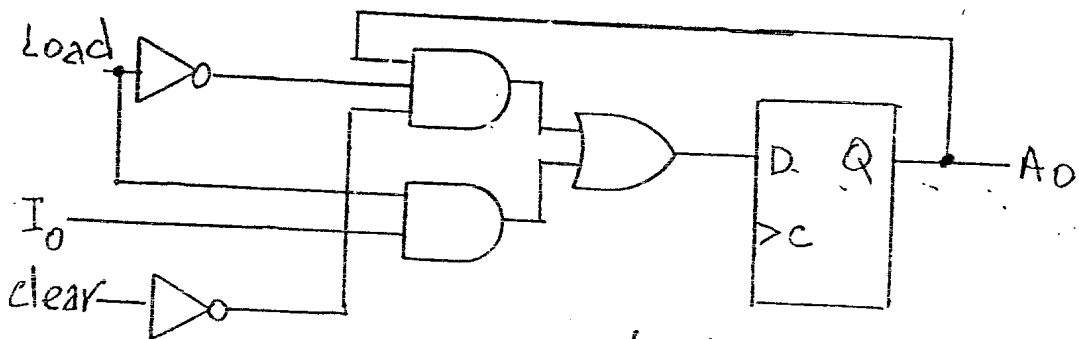
One stage of
Register Fig. 2-7

Load	Clear	D	Operation
0	0	Q(t)	No change
0	1	0	Clear to 0
1	X	I_0	load I_0

Function table

2-12

Input bits	4-bit register	
	1 1 0 1	← Initial value
1	1 1 0 1	
0	1 1 1 0	
1	0 1 1 1	
1	1 0 1 1	
0	1 1 0 1	
1	0 1 1 0	
	1 0 1 1	
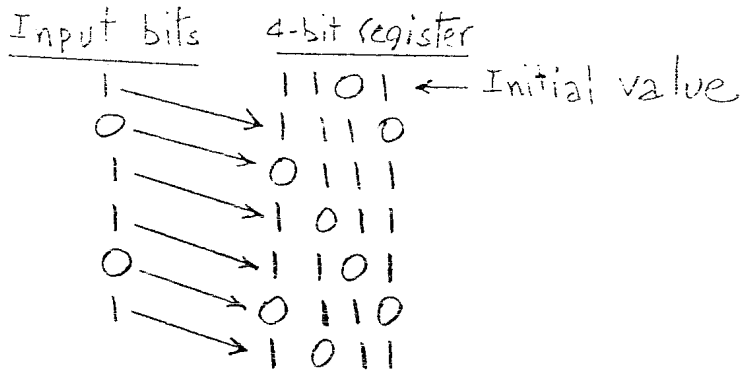
2-13

Serial transfer: One bit at a time by shifting.

Parallel transfer: All bits at the same time.

Input serial data by shifting – output data in parallel.

Input data with parallel load – Output data by shifting.

2-14

$$\rightarrow 1000 \rightarrow 0100 \rightarrow 0010 \rightarrow 0001 \rightarrow$$
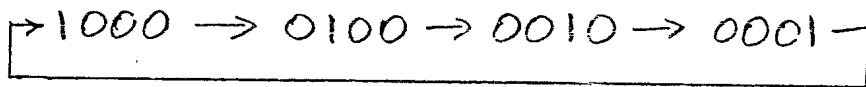
2-15
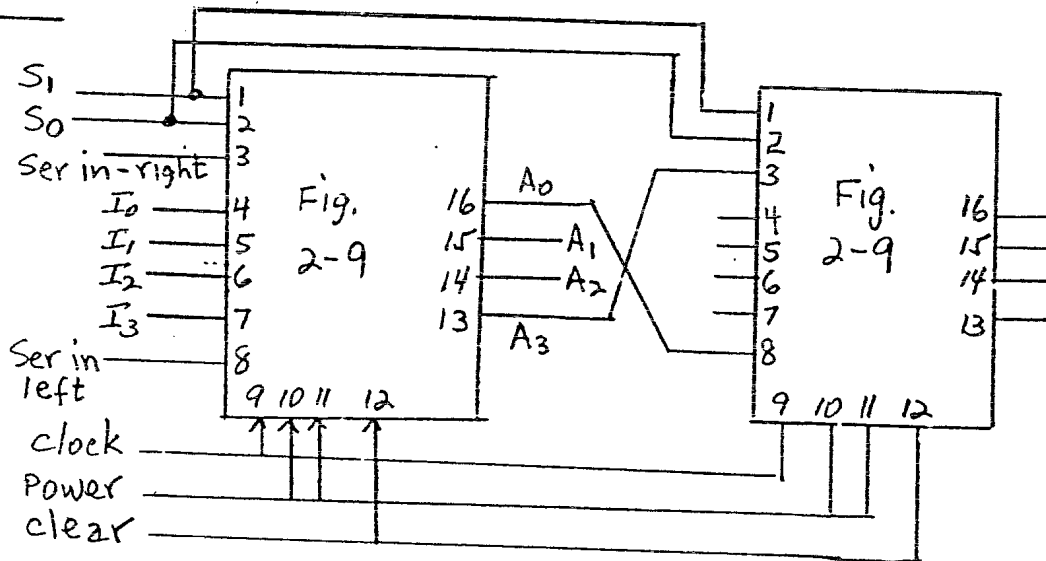
2-16 (a) 4 ; (b) 9

2-17
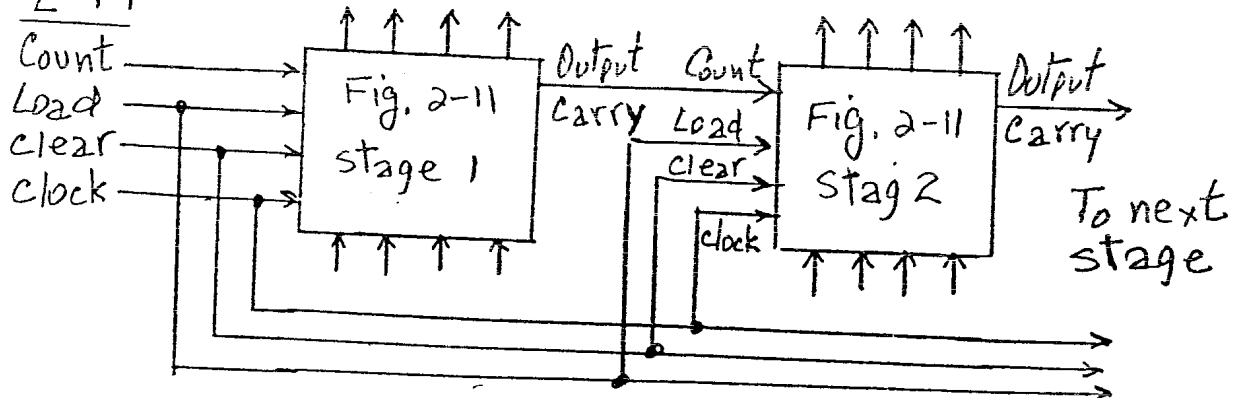
2-18

After the count reaches $N-1 = 1001$, the register loads 0000 from inputs.

Fig. 2-11
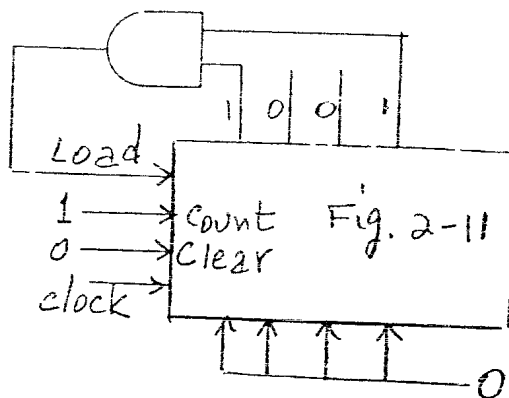
2-19

	Address lines	Data lines
(a) $2K \times 16 = 2^{11} \times 16$	11	16
(b) $64K \times 8 = 2^{16} \times 16$	16	8
(c) $16M \times 32 = 2^{24} \times 32$	24	32
(d) $4G \times 64 = 2^{32} \times 64$	32	64

2-20

(a) $2K \times 2 = 4K = 4096$ bytes

(b) $64K \times 1 = 64K = 2^{16}$ bytes

(c) $2^{24} \times 4 = 2^{26}$ bytes

(d) $2^{32} \times 8 = 2^{35}$ bytes

2-21

$$\frac{4096 \times 16}{128 \times 8} = \frac{2^{12} \times 2^4}{2^7 \times 2^3} = 2^6 = 64 \text{ chips}$$

2-22
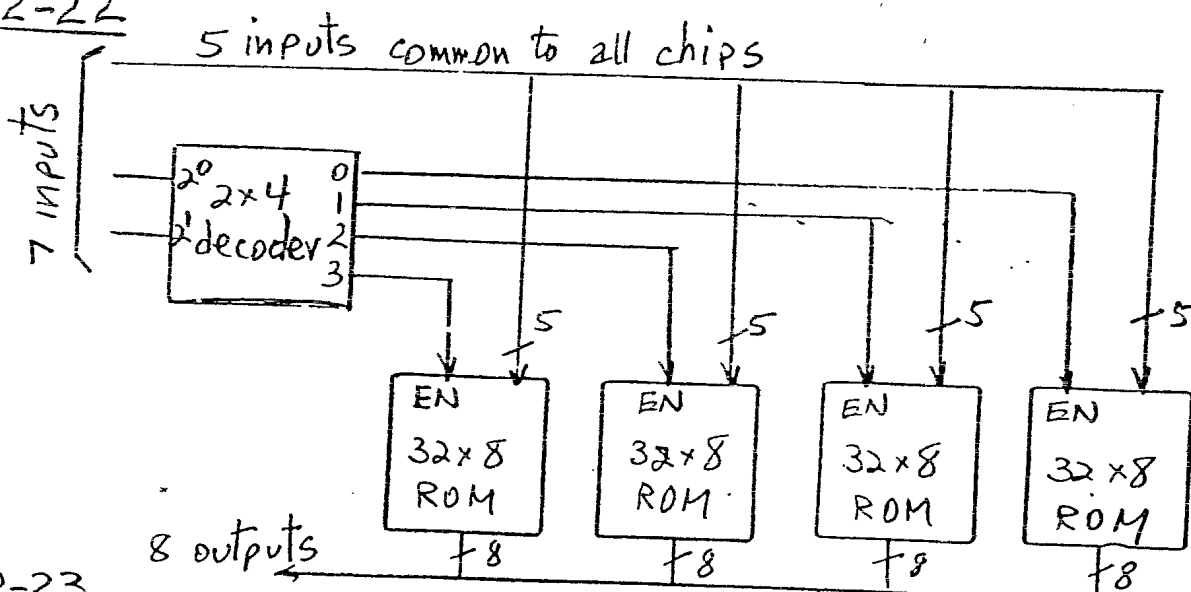
2-23

12 data inputs + 2 enable inputs + 8 data outputs + 2 for power = 24 pins

12

3-1

$(101110)_2 = 32+8+4+2 = 46$

$(1110101)_2 = 64+32+16+4+1 = 117$

$(110110100)_2 = 256+128+32+16+4 = 436$

3-2

$(12121)_3 = 3^4 + 2\times 3^3 + 3^2 + 2\times 3 + 1 = 81+54+9+6+1 = 151$

$(4310)_5 = 4\times 5^3 + 3\times 5^2 + 5 = 500+75+5 = 580$

$(50)_7 = 5\times 7 = 35$

$(198)_{12} = 12^2 + 9\times 12 + 8 = 114+108+8 = 260$

3-3

$(1231)_{10} = 1024+128+64+15 = 2^{10}+2^7+2^6+2^3+2^2+2+1 = (10011001111)_2$

$(673)_{10} = 512+128+32+1 = 2^9+2^7+2^5+1 = (1010100001)_2$

$(1998)_{10} = 1024+512+256+128+64+8+4+2$
$= 2^{10} + 2^9 + 2^8 + 2^7 + 2^6 + 2^3 + 2^2 + 2^1 = (11111001110)_2$

3-4

$(7562)_{10} = (16612)_8$

$(1938)_{10} = (792)_{16}$

$(175)_{10} = (10101111)_2$

3-5

$(F3A7C2)_{16} = (1111\ 0011\ 1010\ 0111\ 1100\ 0010)_2$
$= (74723702)_8$

3-6

$(x^2 - 10x + 31)_r = [(x-5)(x-8)]_{10}$

$= x^2 - (5+8)_{10}\, x + (40)_{10}\, x$

Therefore : $(10)_r = (13)_{10}$ $\quad \underline{r=13}$

Also $(31)_r = 3\times 13 + 1 = (40)_{10}$
$(r=13)$

3-7 $(215)_{10} = 128+64+16+7 = (11010111)_2$

   (a)   0000 11010111   Binary

   (b)   000 011 010 111   Binary coded octal
          0   3   2   7

   (c)   0000 1101 0111   Binary coded hexadecimal
         0   D   7

   (d)   0010 0001 0101   Binary coded decimal
         2   1   5

3-8 $(295)_{10} = 256+32+7 = (100100111)_2$

   (a) 0000 0000 0000 0001 0010 0111

   (b) 0000 0000 0000 0010 1001 0101

   (c) 10110010 00111001 00110101

3-10   JOHN DOE

3-11 87650123; 99019899; 09990048; 999999.

3-12 876100; 909343; 900000; 000000

3-13

01010001; 01111110; 01111111; 11111110; 11111111
01010010; 01111111; 10000000; 11111111; 00000000

3-14

  (a) 5250   (b) 1753   (c) 020   (d) 1200
    +8679   +1360   +900   +9750
  1)3929   0)3113   0)920   1)0950
              ↓ 10's complement ↓
               −6887   −080

3-15

   (a)   (b)   (c)   (d)

  11010   11010   000100   1010100
 +10000   10011   010000   0101100
 1)01010   1)01101   0)010100   1)0000000
 (26−16=10) (26−13=13)   ↓   (84−84=0)
                  −101100
               (4−48=−44)

3-16

$+42 = 0101010$
$-42 = 1010110$

$+13 = 0001101$
$-13 = 1110011$

$(+42)$ 0101010
(-13) 1110011
$(+29)$ 0011101

(-42) 1010110
$(+13)$ 0001101
(-29) 1100011

3-17

$01 \leftarrow$ last two carries $\rightarrow 10$

$+70$ 01000110
$+80$ 01010000
$+150$ 10010110

greater than $+127$ — negative

-70 10111010
-80 10110000
-150 01101010

less than -128 — positive

3-18

(a)
(-638) 9362
$(+785)$ $+0785$
$(+147)$ 0147

(b)
(-638) 9362
(-185) $+9815$
(-823) 9177

3-19

Mantissa

| ± | 26 bits |

Exponent

| S | 8 bits | 36 bits
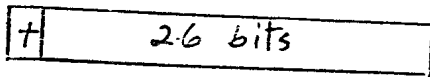
Largest: $+0.1111\ldots.1$
$1 - 2^{-26}$

$+11111111$
$+255$ $(1 - 2^{-26}) \times 2^{+255}$

Smallest: $+0.1000\ldots.0$
(normalized) 2^{-1}

-11111111
-255 2^{-256}

3-20

$46.5 = 32 + 8 + 4 + 2 + 0.5 = (101110.1)_2$

Sign
$\underbrace{0 \ 10111010000000000}_{\text{24-bit mantissa}}$ $\underbrace{00000110}_{\text{8-bit exponent } (+6)}$

15

3-21 (a)

Decimal	Gray code
16	11000
17	11001
18	11011
19	11010
20	11110
21	11111
22	11101
23	11100
24	10100
25	10101
26	10111
27	10110
28	10010
29	10011
30	10001
31	10000

(b)

Decimal	Exess-3 Gray	
9	0010	1010
10	0110	1010
11	0110	1110
12	0110	1111
13	0110	1101
14	0110	1100
15	0110	0100
16	0110	0101
17	0110	0111
18	0110	0110
19	0110	0010
20	0111	0010

3-22 8620

(a) BCD   1000 0110 0010 0000

(b) XS-3   1011 1001 0101 0011

(c) 2421   1110 1100 0010 0000

(d) Binary 10000110101100 (8192+256+128+32+8+4)

3-23

Decimal	BCD with even parity	BCD with odd parity
0	0 0000	1 0000
1	1 0001	0 0001
2	1 0010	0 0010
3	0 0011	1 0011
4	1 0100	0 0100
5	0 0101	1 0101
6	0 0110	1 0110
7	1 0111	0 0111
8	1 1000	0 1000
9	0 1001	1 1001

3-24

$3984 = 0011 \ \ 1111 \ \ 1110 \ \ 0100$

$1100 \ \ 0000 \ \ 0001 \ \ 1011 = 6015$

3-25

A	B	$y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

C	D	$z = C \oplus D$
0	0	0
0	1	1
1	0	1
1	1	0

y	z	$x = y \oplus z$
0	0	0
0	1	1
1	0	1
1	1	0

$\begin{cases} AB = 00 \text{ or } 11 \\ CD = 01 \text{ or } 10 \end{cases}$

$\begin{cases} AB = 01 \text{ or } 10 \\ CD = 00 \text{ or } 11 \end{cases}$

ABCD

0001, 0010, 1101, 1110

0100, 0111, 1000, 1011

Always odd number of 1's

3-26

Same as in Fig. 3-3 but without the complemented circles in the outputs of the gates.

$$P = x \oplus y \oplus z$$

$$Error = x \oplus y \oplus z \oplus P$$

4.1

4-2

$T_0 T_1 T_2 T_3$				$S_1 S_0$		load
0	0	0	0	X	X	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
0	0	1	0	1	0	1
0	0	0	1	1	1	1

$$S_1 = T_2 + T_3$$
$$S_0 = T_1 + T_3$$
$$load = T_0 + T_1 + T_2 + T_3$$
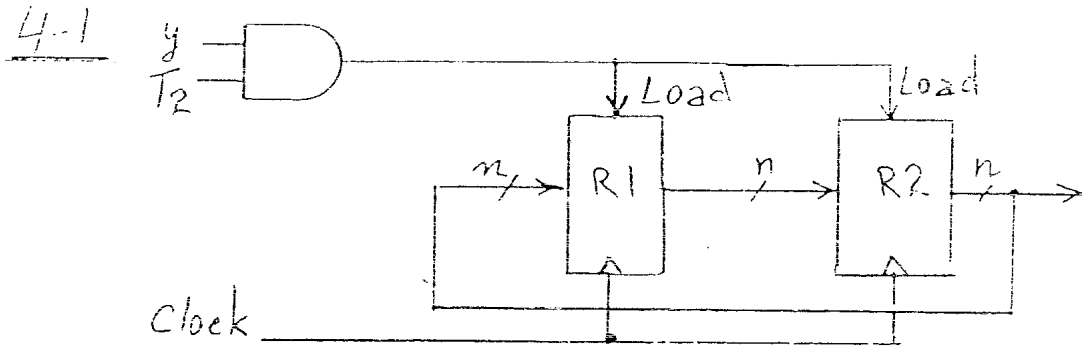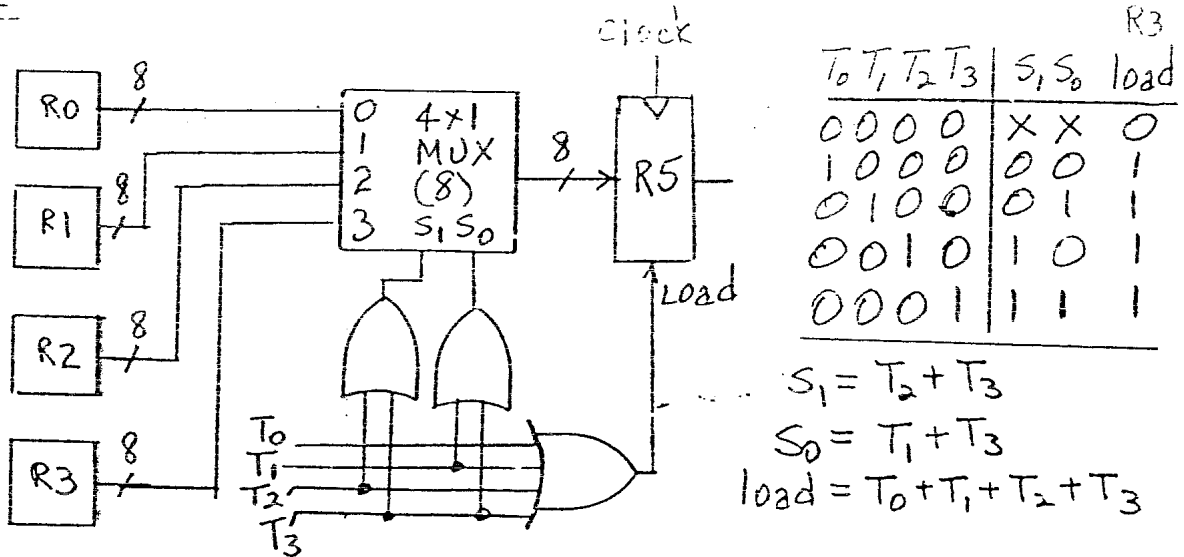
4-3

$$P: \quad R1 \leftarrow R2$$
$$P'Q: \quad R1 \leftarrow R3$$

4-4

Connect the 4-line common bus to the four inputs of each register.

Provide a "load" control input in each register.

Provide a clock input for each register.

To transfer from register C to register A:

Apply $S_1 S_0 = 10$ (to select C for the bus.)

Enable the load input of A

Apply a clock pulse.

Registers C, D
Same as A, B.

4-line
common bus
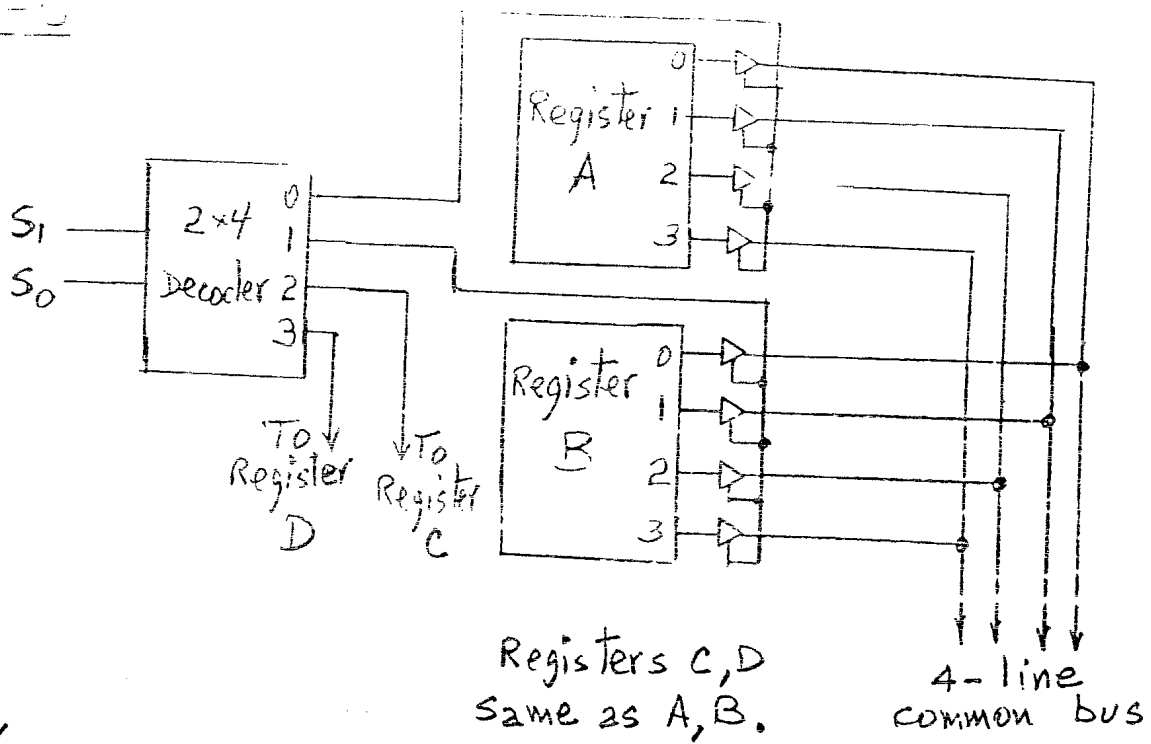
4-6

(a) 4 selection lines to select one of 16 registers.

(b) 16 × 1 multiplexers

(c) 32 multiplexers, one for each bit of the registers.

4-7

(a) Read memory word specified by the address in AR into register R2.

(b) Write content of register R3 into the memory word specified by the address in AR.

(c) Read memory word specified by the address in R5 and transfer content to R5 (destroys previous value)

4-8
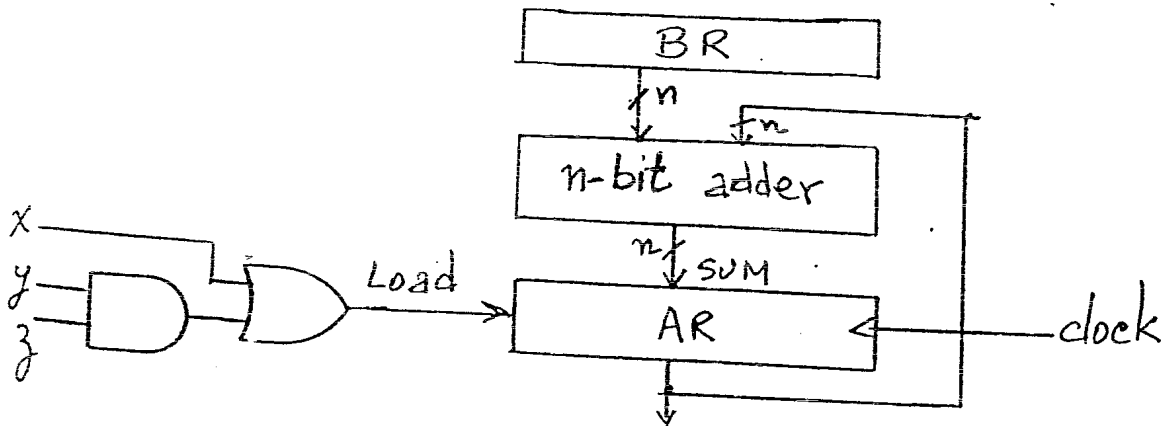
4-9
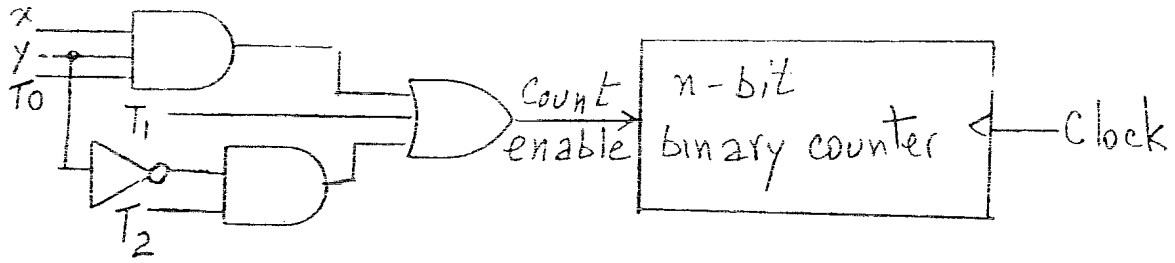
4-10

4-11

4-12
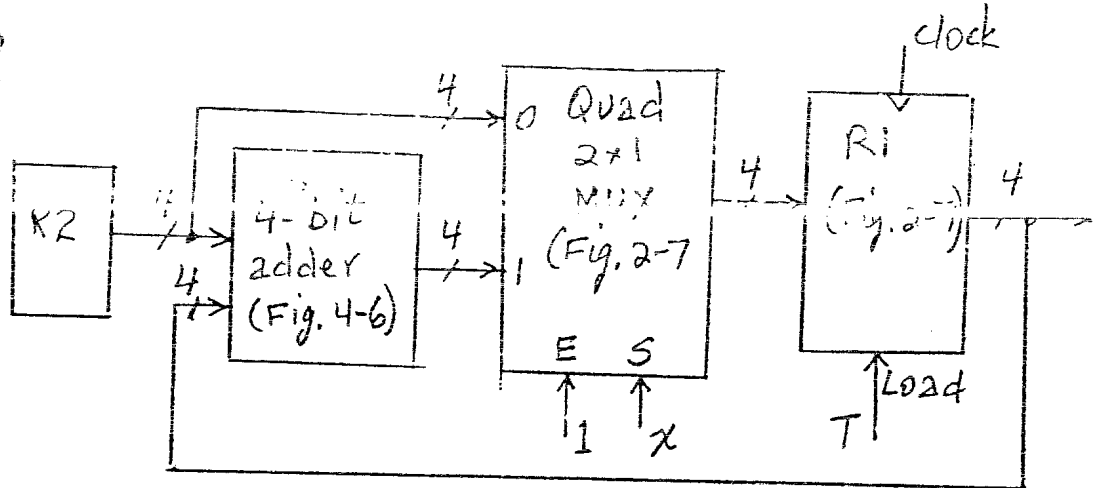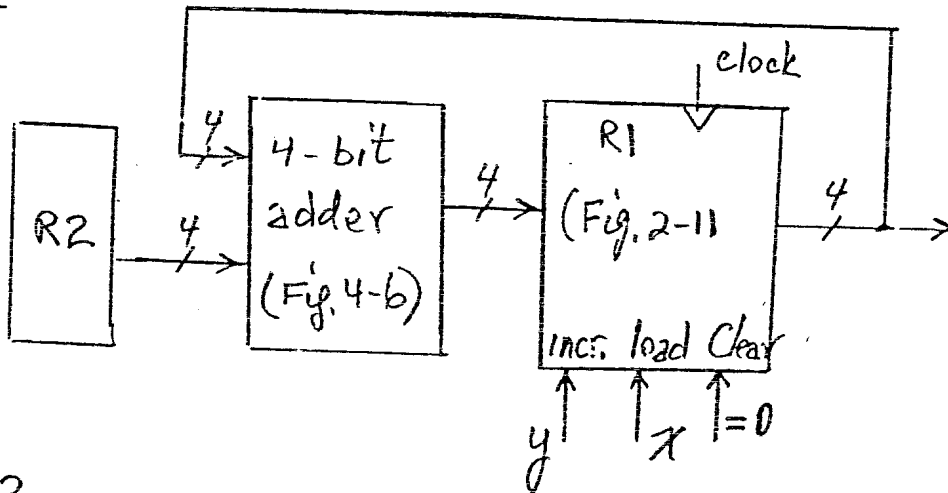
M	A		B	SUM	C₄	
O	0111	+	0110	1101	0	$7+6=13$
O	1000	+	1001	0001	1	$8+9=16+1$
1	1100	−	1000	0100	1	$12-8=4$
1	0101	−	1010	1011	0	$5-10=-5$ (in 2's comp.)
1	0000	−	0001	1111	0	$0-1=-1$ (in 2's comp.)

$\quad A-1 = A + 2\text{'s complement of } 1 = A + 1111$
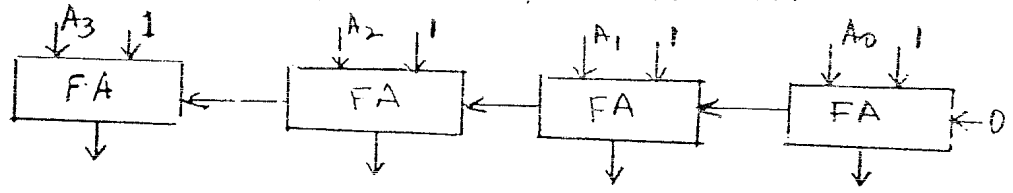
4-14
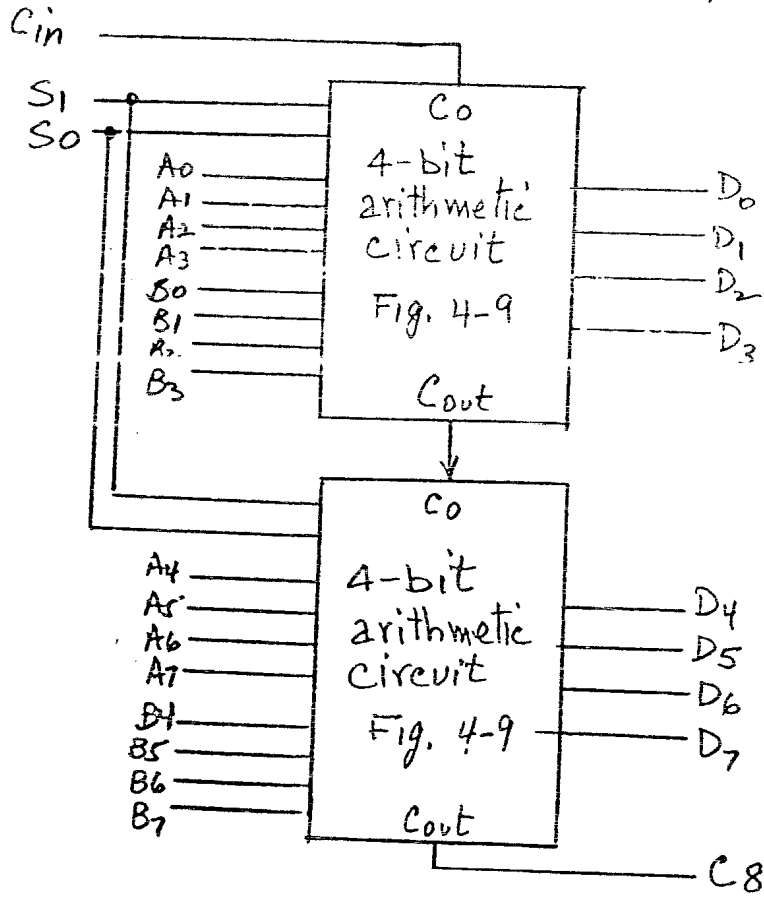
4-15

S	Cin	X	Y	
0	0	A	B	(A+B)
0	1	A	0	(A+1)
1	0	A	1	(A-1)
1	1	A	$\bar{B}$	(A-B)

4-16

X → S₁

Y → S₀   4×1 MUX

Binary value of Fᵢ { 0, 1, 2, 3 } → Fᵢ

$i = 0, 1, 2, \ldots, 15$

4-17

S_1, S_0 → 4×1 MUX → E_i

inputs A_i, B_i to gates feeding 0, 1, 2, 3
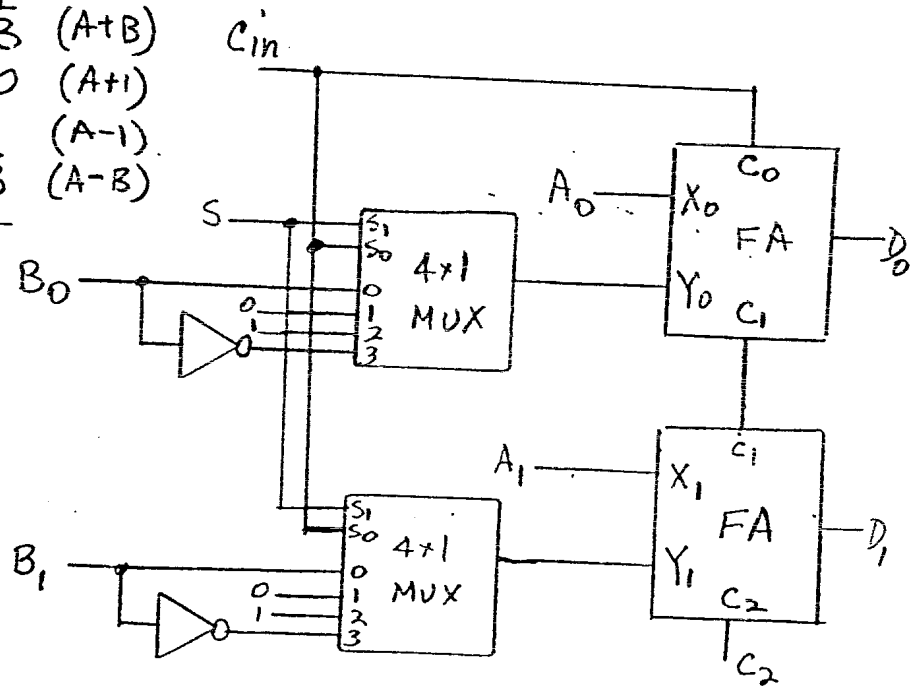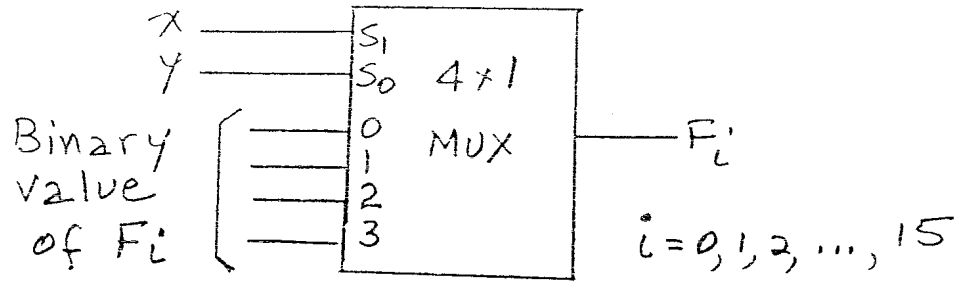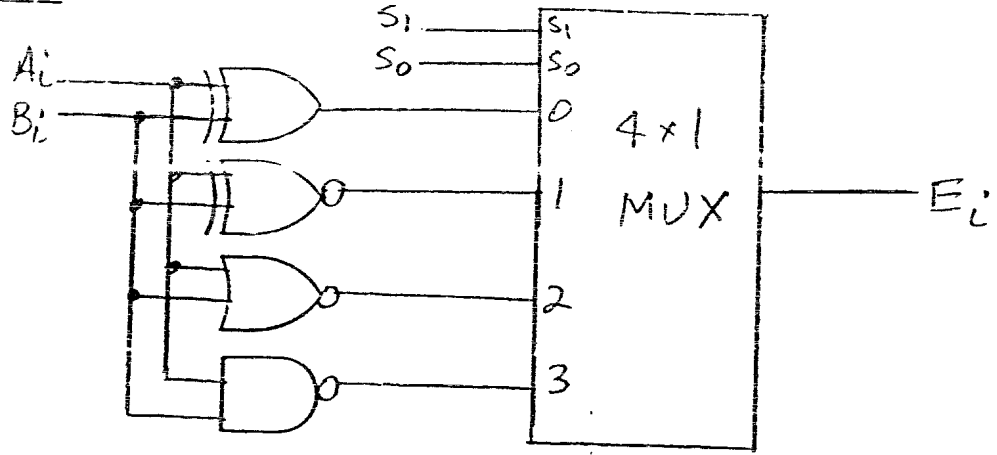
4-18

(a) $A = 1101\,1001$
      $B = 1011\,0100$ ⊕
$A \leftarrow A \oplus B$ $\overline{0110\,1101}$

$A = 1101\,1001$
$B = 1111\,1101$ (OR)
$\overline{1111\,1101}$ $A \leftarrow A \vee B$

4-19

(a) $AR = 1111\,0010$
    $BR = 1111\,1111$ (+)
    $\overline{}$
    $AR = 1111\,0001$ $BR = 1111\,1111$ $CR = 1011\,1001$ $DR = 1110\,1010$

(b) $CR = 1011\,1001$
    $DR = 1110\,1010$ (AND)
    $\overline{CR = 1010\,1000}$

    $BR = 1111\,1111$
    $+\,1$
    $\overline{BR = 0000\,0000}$ $AR = 1111\,0001$ $DR = 1110\,1010$

(c) $AR = 1111\,0001$ (−)
    $CR = 1010\,1000$
    $\overline{AR = 0100\,1001}$; $BR = 0000\,0000$; $CR = 1010\,1000$; $DR = 1110\,1010$

22

4-20

$R = 10011100$

Arithmetic shift right : 11001110
Arithmetic shift left : 00111000 overflow because a negative number changed to positive

4-21

$R = 1101101$

logical shift left : 1011010
Circular shift right : 0101101
logical shift right : 0010110
Circular shift left : 0101100

4-22

$S = 1$ Shift left

A_0 A_1 A_2 A_3 I_L

$H = 0\ 0\ 1\ 0\ 0$ shift left

(from $1\ 0\ 0\ 1\ 0$)
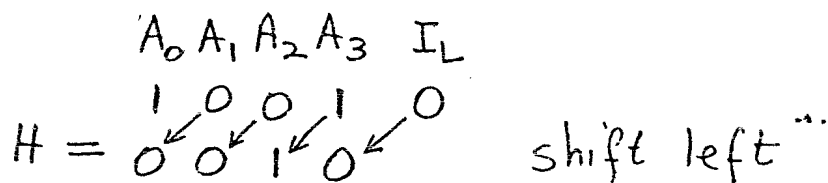
4-23

(a) Cannot complement and increment the same register at the same time.

(b) Cannot transfer two different values (R_2 and R_3) to the same register (R1) at the same time.

(c) Cannot transfer a new value into a register (PC) and increment the original value by one at the same time.

5-1

$$256K = 2^8 \times 2^{10} = 2^{18}$$
$$64 = 2^6$$

(a) Address : 18 bits
Register code : 6 bits
Indirect bit : 1 bit
$\overline{25}$ $32 - 25 = 7$ bits for opcode.

(b)

1	7	6	18	= 32 bits
I	opcode	Register	Address	
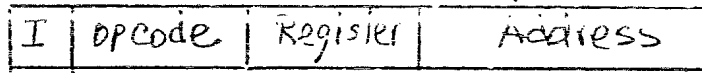
(c) Data : 32 bits ; address : 18 bits.

5-2

A direct address instruction needs two references to memory : (1) Read instruction ; (2) Read operand.

An indirect address instruction needs three references to memory : (1) Read instruction ; (2) Read effective address ; (3) Read operand.

5-3

(a) Memory read to bus and load to IR : $IR \leftarrow M[AR]$

(b) TR to bus and load to PC : $PC \leftarrow TR$

(c) AC to bus, write to memory, and load to DR : $DR \leftarrow AC$, $M[AR] \leftarrow AC$.

(d) Add DR (or INPR) to AC : $AC \leftarrow AC + DR$

5-4

		(1) $S_2 S_1 S_0$	(2) Load (LD)	(3) Memory	(4) Adder
(a)	$AR \leftarrow PC$	010 (PC)	AR	—	—
(b)	$IR \leftarrow M[AR]$	111 (M)	IR	Read	—
(c)	$M[AR] \leftarrow TR$	110 (TR)	—	Write	—
(d)	$DR \leftarrow AC$ $AC \leftarrow DR$	100 (AC)	DR and AC	—	Transfer DR to AC

5-5

(a) $IR \leftarrow M[PC]$ PC cannot provide address to memory.
Address must be transferred to AR first

$AR \leftarrow PC$
$IR \leftarrow M[AR]$

(b) $AC \leftarrow AC + TR$ Add operation must be done with DR. Transfer TR to DR first.

$DR \leftarrow TR$
$AC \leftarrow AC + DR$

(c) $DR \leftarrow DR + AC$ Result of addition is transferred to AC (not DR). To save value of AC, its content must be stored temporary in DR (or TR).

$AC \leftarrow DR, \ DR \leftarrow AC$ (See answer to Problem 5-4(b))
$AC \leftarrow AC + DR$
$AC \leftarrow DR, \ DR \leftarrow AC$

5-6

(a) $\underline{0001} \ \underline{0000 \ 0010 \ 0100} = (1024)_{16}$
    ADD $(024)_{16}$
    ADD content of M[024] to AC   ADD 024

(b) $\underline{1} \underline{011} \ \underline{0001 \ 0010 \ 0100} = (B124)_{16}$
    I STA $(124)_6$
    store AC in M[M[124]]   STA I 124

(c) $\underline{0111} \ \underline{0000 \ 0010 \ 0000} = (7020)_{16}$
    Register   Increment Ac   INC

5-7

CLE   Clear E
CME   Complement E

5-8

Timing diagram showing clock, T_C, T_1, T_2, T_3, C_7, and $C_7 T_3$ waveforms.

Clock pulses labeled T_C, T_1, T_2, T_3, T_0.

$C_7 T_3$

↑ SC goes to 0 causing $T_0 = 1$
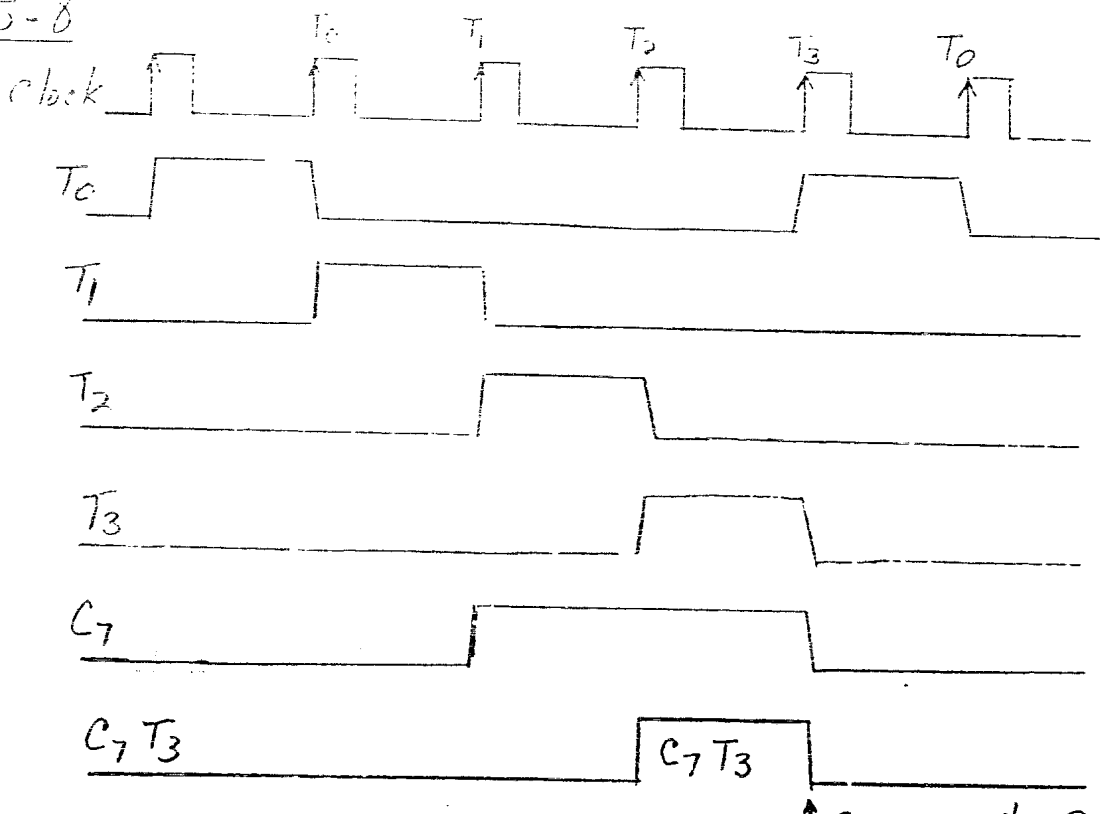
5-9

	E	AC	PC	AR	IR
Initial	1	A937	021	—	—
CLA	1	0000	022	800	7800
CLE	0	A937	022	400	7400
CMA	1	56C8	022	200	7200
CME	0	A937	022	100	7100
CIR	1	D49B	022	080	7080
CIL	1	526F	022	040	7040
INC	1	A938	022	020	7020
SPA	1	A937	022	010	7010
SNA	1	A937	023	008	7008
SZA	1	A937	022	004	7004
SZE	1	A937	022	002	7002
HLT	1	A937	022	001	7001

5-10

	PC	AR	DR	AC	IR
Initial	021	—	—	A937	—
AND	022	083	B8F2	A832	0083
ADD	022	083	B8F2	6229	1083
LDA	022	083	B8F2	B8F2	2083
STA	022	083	—	A937	3083
BUN	083	083	—	A937	4083
BSA	084	084	—	A937	5083
ISZ	022	083	B8F3	A937	6083

5-11

	PC	AR	DR	IR	SC
Initial	7FF	—	—	—	0
T_0	7FF	7FF	—	—	1
T_1	800	7FF	—	EA9F	2
T_2	800	A9F	—	EA9F	3
T_3	800	C35	—	EA9F	4
T_4	800	C35	FFFF	EA9F	5
T_5	800	C35	0000	EA9F	6
T_6	801	C35	0000	EA9F	0

5-12

Memory

3AF	932E
32E	09AC
9AC	8B9F

(a) $9 = (1001)_2$

$I=1$ $\dfrac{1\,001}{\text{ADD}}$ ADD I 32E

(b)

$$AC = 7EC3$$
$$DR = 8B9F \quad (ADD)$$
$$\overline{0A62}$$

(E = 1)

AC = 7EC3

(c) PC = 3AF+1 = 3B0   IR = 932E
AR = 9AC   E = 1
DR = 889F   I = 1
AC = 0A62   SC = 0000

5-13

XOR	$D_0 T_4$:	$DR \leftarrow M[AR]$
	$D_0 T_5$:	$AC \leftarrow AC \oplus DR, \quad SC \leftarrow 0$

ADM	$D_1 T_4$:	$DR \leftarrow M[AR]$
	$D_1 T_5$:	$DR \leftarrow AC, \quad AC \leftarrow AC + DR$
	$D_1 T_6$:	$M[AR] \leftarrow AC, \quad AC \leftarrow DR, \quad SC \leftarrow 0$

SUB	$D_2 T_4$:	$DR \leftarrow M[AR]$
	$D_2 T_5$:	$DR \leftarrow AC, \quad AC \leftarrow DR$
	$D_2 T_6$:	$AC \leftarrow \overline{AC}$
	$D_2 T_7$:	$AC \leftarrow AC + 1$
	$D_2 T_8$:	$AC \leftarrow AC + DR, \quad SC \leftarrow 0$

XCH	$D_3 T_4$:	$DR \leftarrow M[AR]$
	$D_3 T_5$:	$M[AR] \leftarrow AC, \quad AC \leftarrow DR, \quad SC \leftarrow 0$

SEQ	$D_4 T_4$:	$DR \leftarrow M[AR]$
	$D_4 T_5$:	$TR \leftarrow AC, \quad AC \leftarrow AC \oplus DR$
	$D_4 T_6$:	If $(AC = 0)$ then $(PC \leftarrow PC+1)$, $AC \leftarrow TR$, $SC \leftarrow 0$

BPA	$D_5 T_4$:	If $(AC = 0 \wedge AC(15) = 0)$ then $(PC \leftarrow AR), \quad SC \leftarrow 0$

5-14

Converts the ISZ instruction from a memory-reference
instruction to a register-reference instruction.
The new instruction ICSZ can be executed
at time T_3 instead of time T_6, a saving
of 3 clock cycles.

5-15 Modify Fig. 5-4:

Execute memory-reference instruction

5-16

(a)

(b)

Memory

op code
½ address
½ address
operand

←— 8-bits —→
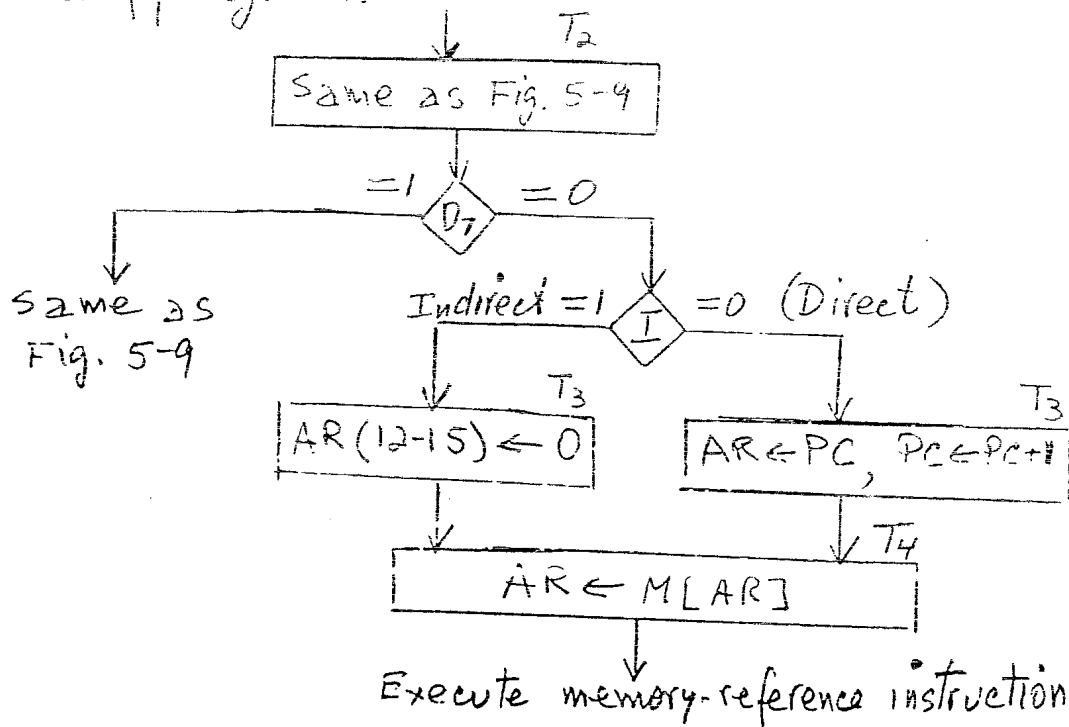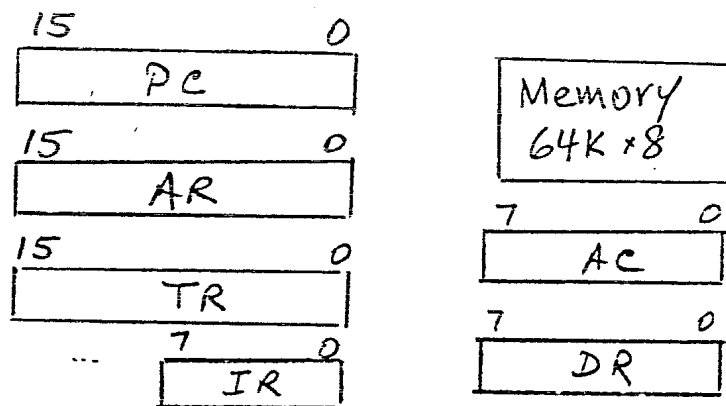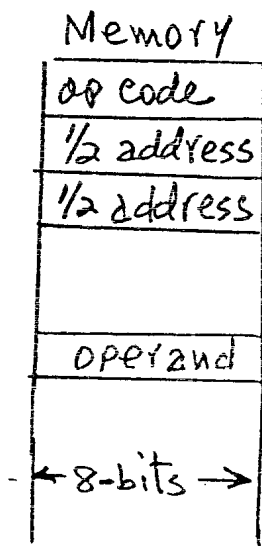
(c) T_0: $IR \leftarrow M[PC]$, $PC \leftarrow PC+1$

T_1: $AR(0\text{-}7) \leftarrow M[PC]$, $PC \leftarrow PC+1$

T_2: $AR(8\text{-}15) \leftarrow M[PC]$, $PC \leftarrow PC+1$

T_3: $DR \leftarrow M[AR]$

5-11

$$IR = \boxed{\text{opcode 1} \mid \text{Address1} \mid \text{opcode 2} \mid \text{Address 2}}$$

6 bits   .4 bits   6 bits   14 bits   =40 bits

Decoder 1   Decoder 2
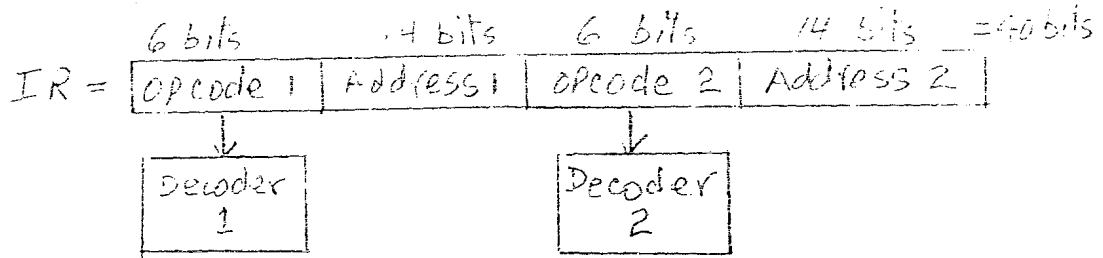
1. Read 40-bit double instruction from memory to IR and then increment PC.
2. Decode opcode 1.
3. Execute instruction 1 using address 1.
4. Decode opcode 2.
5. Execute instruction 2 using address 2.
6. Go back to step 1.

5-18

(a) BUN 2300

(b) ION
    BUN 0 I   (Branch indirect with address 0)

5-19

$X_3'X_1$: READ

$X_1'X_3$: WRITE

$X_1'X_2$: $R \leftarrow AC$

5-20
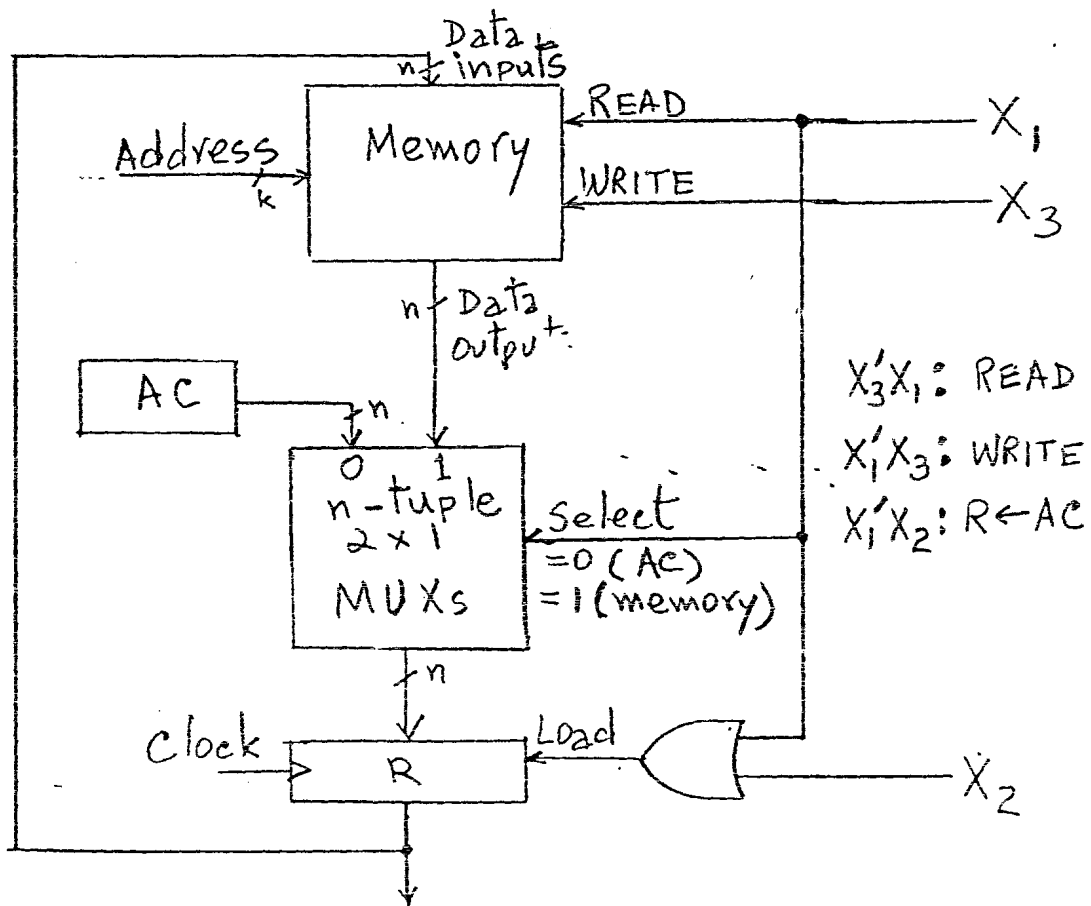
$$J_F = x T_3 + 3 T_2 + w T_5 G \qquad K_F = y T_1 + 3 T_2 + w T_5 G'$$
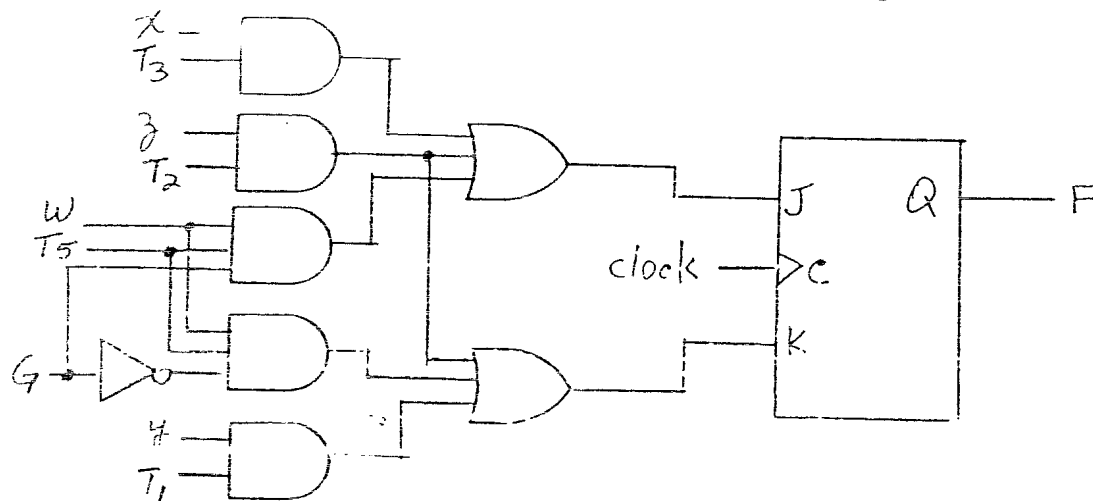
5-21

From Table 5-6: ($z_{DR}=1$ if $DR=0$; $Z_{AC}=1$ if $AC=0$)

$$INR(PC) = R'T_1 + RT_2 + D_6 T_6 Z_{DR} + pB_9 (FGI) + pB_8 (FGO)$$
$$+ rB_4 (AC_{15})' + rB_3 (AC_{15}) + rB_2 Z_{AC} + rB_1 E'$$

$$LD(PC) = D_4 T_4 + D_5 T_5$$

$$CLR(PC) = RT_1$$

The logic diagram is similar to the one in Fig. 5-16.

5-22

$$Write = D_3 T_4 + D_5 T_4 + D_6 T_6 + RT_1 \qquad (M[AR] \leftarrow xx)$$

5-23

$$(T_0 + T_1 + T_2)'(IEN)(FGI + FGO): \quad R \leftarrow 1$$
$$RT_2: \quad R \leftarrow 0.$$

5-24
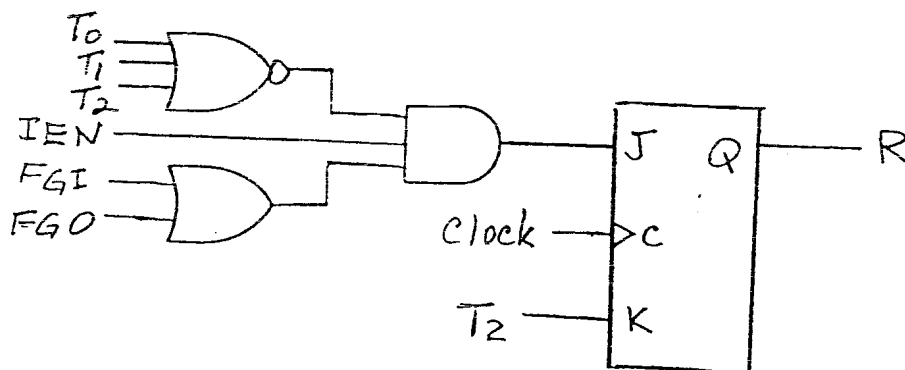
x_2 places PC onto the bus. From Table 5-6:

$R'T_0$: $AR \leftarrow PC$
$R T_0$: $TR \leftarrow PC$
$D_5 T_4$: $M[AR] \leftarrow PC$

$$x_2 = R'T_0 + R T_0 + D_5 T_4 = (R'+R)T_0 + D_5 T_4 = T_0 + D_5 T_4$$

5-25
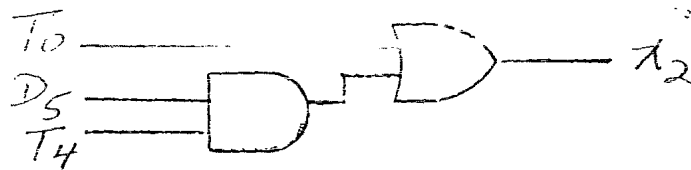
From Table 5-6:

$$CLR(SC) = RT_2 + D_7 T_3 (I'+I) + (D_0 + D_1 + D_2 + D_5) T_5$$
$$+ (D_3 + D_4) T_4 + D_6 T_6$$
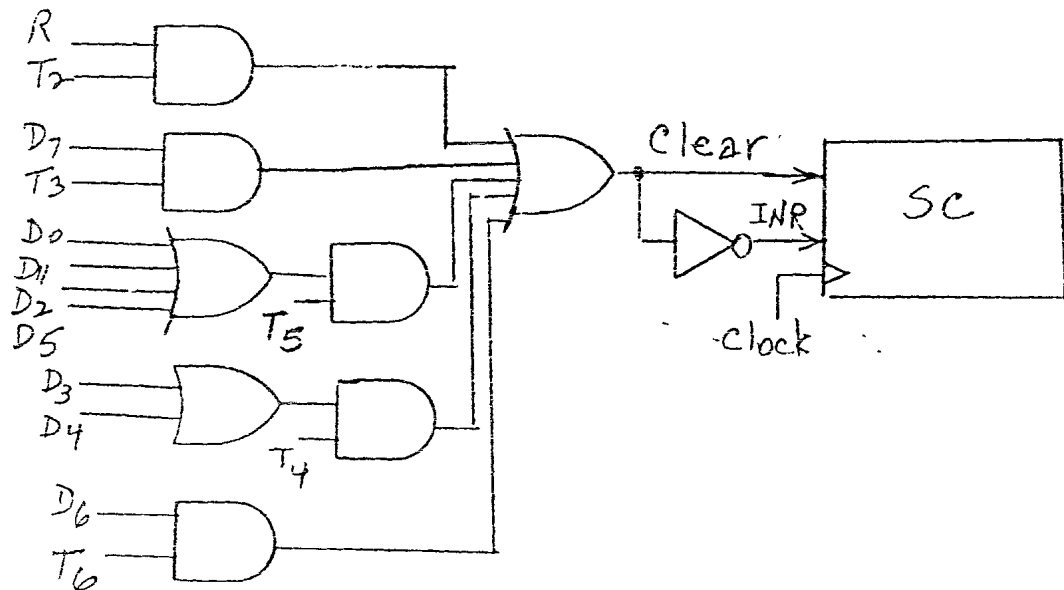
6-1

		AC	PC	IR
010	CLA	0000	011	7800
011	ADD 016	C1A5	012	1016
012	BUN 014	C1A5	014	4014
013	HLT	8184	014	7001
014	AND 017	8184	015	0017
015	BUN 013	8184	013	4013
016	C1A5			
017	93C6			

$$(C1A5)_{16} = 1100\ 0001\ 1010\ 0101$$
$$(93C6)_{16} = 1001\ 0011\ 1100\ 0110 \quad \text{AND}$$
$$\overline{1000\ 0001\ 1000\ 0100} = (8184)_{16}$$

6-2

				AC
100	5103	BSA 103		
101	7200	CMA		FFFE ← Answer
102	7001	HLT		
103	0000	5101 ← Answer		
104	7800	CLA		0000
105	7020	INC		0001
106	C103	BUN 103 I		

6-3

```
CLA
STA  SUM    } SUM = 0

LDA  SUM
ADD  A
ADD  B      } SUM = SUM + A + B
STA  SUM
LDA  C
CMA
INC
ADD  DIF    } DIF = DIF - C
STA  DIF
LDA  SUM
ADD  DIF    } SUM = SUM + DIF
STA  SUM
```

A more efficient compiler will optimize the machine code as follows:

```
LDA  A
ADD  B
STA  SUM
LDA  C
CMA
INC
ADD  DIF
STA  DIF
ADD  SUM
STA  SUM
```

6-4   A line of code such as: LDA I is interpreted by the assembler (Fig. 6-2) as a two symbol field with I as the symbolic address. A line of code such as: LDA I I is interpreted as a three symbol field. The first I is an address symbol and the second I as the Indirect bit.
Answer: Yes, it can be used for this assembler.

6-5

The assembler will not detect an ORG or END if the line has a label; according to the flow chart of Fig. 6-1. Such a label has no meaning and constitutes an error. To detect the error, modify the flow chart of Fig. 6-1:
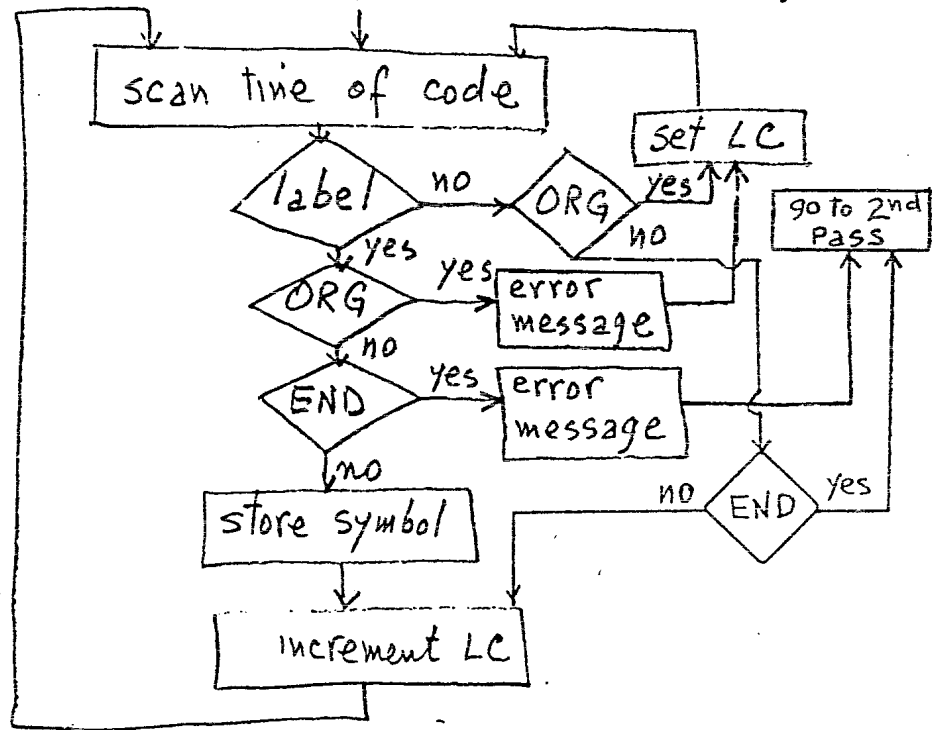
6-6 (a)

memory word	characters	Hex	binary
1	D E	44 45	0100 0100 0100 0101
2	C space	43 20	0100 0011 0010 0000
3	— 3	2D 33	0010 1101 0011 0011
4	5 CR	35 0D	0011 0101 0000 1101

(b) $(35)_{10} = (0000\ 0000\ 0010\ 0011)_2$

$-35 \rightarrow 1111\ 1111\ 1101\ 1101 = (FFDD)_{16}$

6-7 (a)

LOP	105	$(100)_{10} = (0000\ 0000\ 0110\ 0100)_2$
ADS	10B	$(-100)_{10} = (1111\ 1111\ 1001\ 1100)_2 = (FF9C)_{16}$
PTR	10C	
NBR	10D	$(75)_{10} = (0000\ 0000\ 0100\ 1011)_2 = (004B)_{16}$
CTR	10E	
SUM	10F	$(23)_{10} = (0000\ 0000\ 0001\ 0111)_2 = (0017)_{17}$

(b)

Loc	Hex				Loc	Hex		
		ORG	100					
100	210B	LDA	ADS		10B	0150	ADS,	HEX 150
101	310C	STA	PTR		10C	0000	PTR,	HEX 0
102	210D	LDA	NBR		10D	FF9C	NBR,	DEC -100
103	310E	STA	CTR		10E	0000	CTR,	HEX 0
104	7800	CLA			10F	0000	SUM,	HEX 0
105	910C	LOP, ADD	PTR I					ORG 150
106	610C	ISZ	PTR		150	004B		DEC 150
107	610E	ISZ	CTR		⋮	⋮		⋮
108	4105	BUN	LOP		1B3	0017		DEC 23
109	310F	STA	SUM					END
10A	7001	HLT						

6-8 Modify flow chart of Fig. 6-1
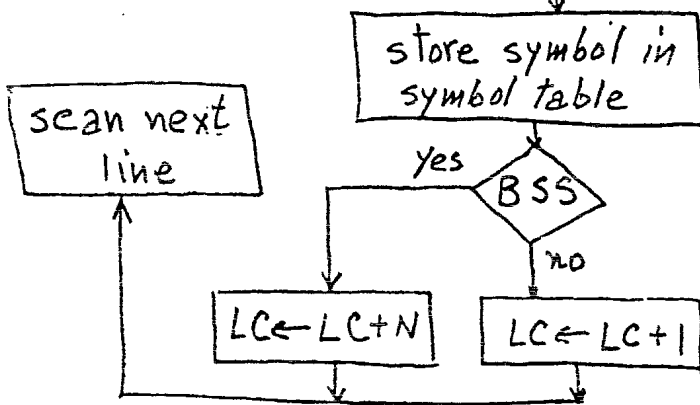
LC=1   A, BSS 10
  2   A+1
  3   A+2
  4   A+3
  5   A+4
  6   A+5
  7   A+6
  8   A+7
  9   A+8
 10   A+9
 11   LC set to 11

example

6-9
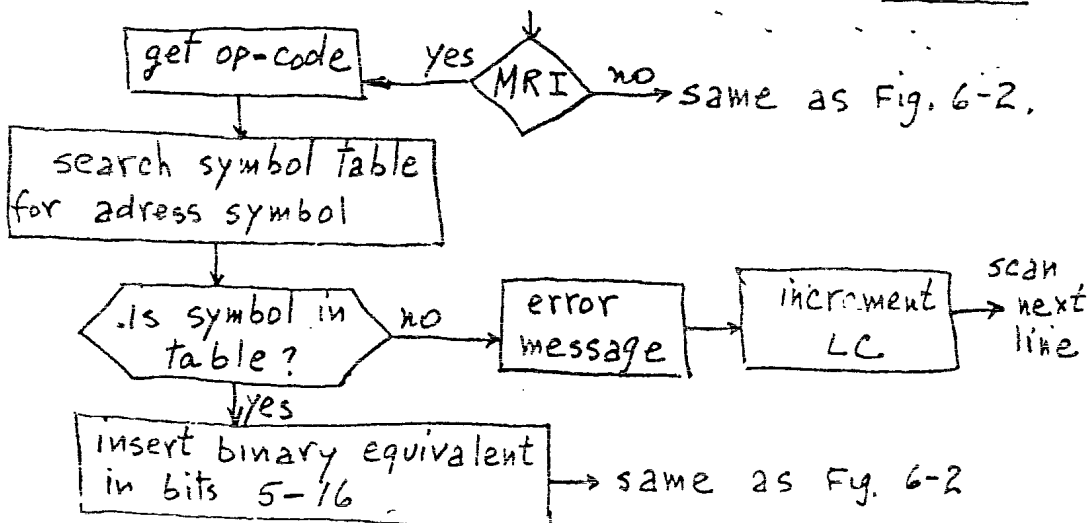
6-10 (a) MRI table →

memory word	symbol	HEX
AND { 1	A N	41 4D
2	D space	44 20
3	Value	00 00
ADD { 4	A D	41 44
5	D space	44 20
6	value	10 00

etc.

(b) non-MRI table →

word	symbol	HEX
CLA { 1	C L	43 4C
2	A space	41 20
3	value	78 00
CLE { 4	C L	43 4C
5	E space	45 20
6	value	74 00

etc.

6-11

```
LDA   B
CMA
INC
ADD   A      / Form  A-B
SPA          / skip if AC positive.
BUN   N10    /(A-B)<0, go to N10
SZA          / skip if AC=0
BUN   N30    /(A-B)>0, go to N30
BUN   N20    / (A-B)=0, go to N20
```

6-12

(a) The program counts the number of 1's in the number stored in location WRD. Since WRD = $(62C1)_{16}$ =

$$(0110\ 0010\ 1100\ 0001)_2$$

number of 1's is 6; so CTR will have $(0006)_{16}$

(b)

```
                   ORG 100
100   7400         CLE
101   7800         CLA
102   3110         STA CTR      /Initialize counter to zero
103   2111         LDA WRD
104   7004         SZA
105   4107         BUN ROT
106   410F         BUN STP      / Word is zero; stop with CTR=0
107   7040  ROT,   CIL          / Bring bit to E
108   7002         SZE
109   410B         BUN AGN      / bit =1, go to count it
10A   4107         BUN ROT      / bit =0, repeat
10B   7400  AGN,   CLE
10C   6110         ISZ CTR      /Increment counter
```

```
    IOD    7004    SZA        /check if remaning bits = 0
    IOE    4107    BUN ROT  /No; rotate again
    IOF    7001  STP, HLT      /yes; stop
    110    0000  CTR, HEX  O
    111    62C1  WRD, HEX 62C1
                       END
```

6-13 $(100)_{16} = (256)_{10}$ 500 to 5FF $\rightarrow (256)_{10}$ locations

```
         ORG    100
         LDA    ADS
         STA    PTR      /Initialize pointer
         LDA    NBR
         STA    CTR      /Initialize counter to -256
         CLA
  LOP,   STA    PTR I    /store zero
         ISZ    PTR
         ISZ    CTR
         BUN    LOP
         HLT
  ADS,   HEX    500
  PTR,   HEX    O
  NBR,   DEC   -256
  CTR,   HEX    O
         END
```

6-14

```
         LDA    A        /Load multiplier
         SZA             /Is it zero?
         BUN    NZR
         HLT             /A=0, product=0 in AC
  NZR,   CMA
         INC
         STA    CTR      /Store -A in counter
         CLA             /Start with AC=0
  LOP,   ADD    B        /Add multiplicand
         ISZ    CTR
         BUN    LOP      /Repeat Loop A times
         HLT
  A,     DEC    —        /multiplier
  B,     DEC    —        /multiplicand
  CTR,   HEX    O        /counter
         END
```

6-15 The first time the program is executed, location CTR will go to 0. If the program is executed again starting from location $(100)_{16}$, location CTR will be incremented and will not reach 0 until it is incremented $2^{16} = 65,536$ times, at which time it will reach 0 again.

We need to initialize CTR and P as follows:

```
        LDA   NBR
        STA   CTR
        CLA
        STA   P
              ↓
           Program
              ↓
NBR,  DEC  -8
CTR,  HEX   0
  P,  HEX   0
```

6-16 Multiplicand is initially in location XL. Will be shifted left into XH (which has zero initially). The partial product will contain two locations PL and PH (initially zero). Multiplier is in location Y. CTR = -16

```
LOP,    CLE
        LDA   Y
        CIR
        STA   Y
        SZE
        BUN   ONE
        BUN   ZRO
ONE,    LDA   XL
        ADD   PL
        STA   PL
        CLA
        CIL
        ADD   XH
        ADD   PH
        STA   PH
        CLE
```

same as beginning of Program in Table 6-14

Double-precision add

$$P \leftarrow X + P$$

Same as program in Table 6-15

Continued next page

38

```
ZRO,    LDA  XL  ⎤  Double-precision
        CIL       |
        STA  XL   |  left-shift  XH + XL
        LDA  XH   |
        CIL       |
        STA  XH  ⎦

        ISZ  CTR ⎤  Repeat 16 times
        BUN  LOP  |
        HLT      ⎦
```

6-17

If multiplier is negative, take the 2's complement of multiplier and multiplicand and then proceed as in Table 6-14 (with CTR=-7).

Flow-Chart :

CTR ← 7
P ← 0

check sign of multiplier

Y ← Ȳ+1   2's complement multiplier

X ← X̄+1   2's complement multiplicand

Proceed as in Table 6-14
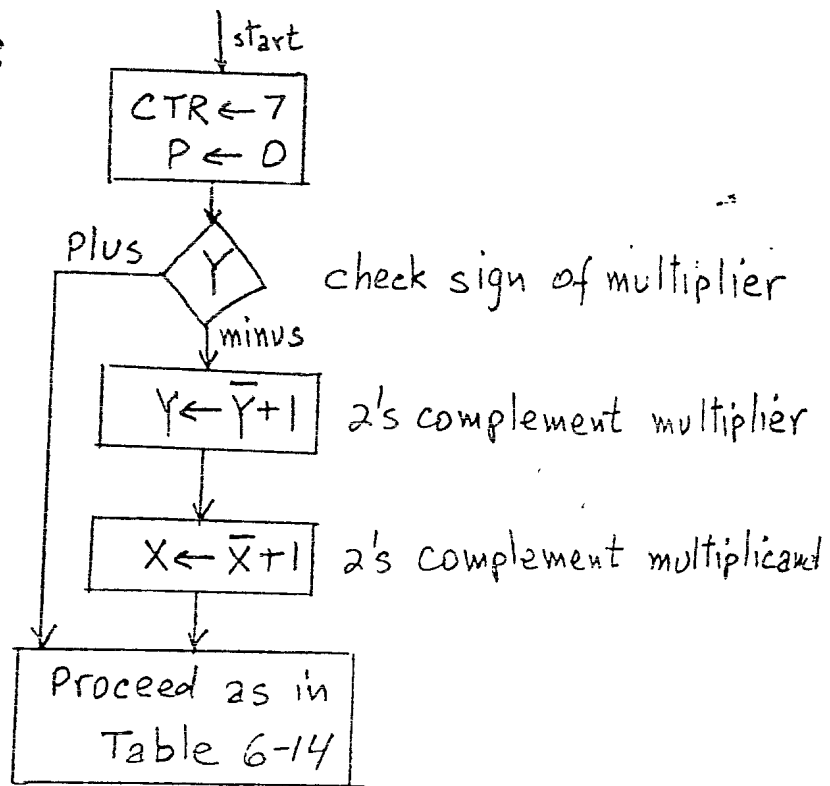
6-18 $C \leftarrow A - B$

```
        CLE
        LDA   BL
        CMA
        INC
        ADD   AL
        STA   CL
save  ⎡ CLA
carry ⎢ CIL
      ⎣ STA   TMP
        LDA   BH
        CMA
        ADD   AH
add carry → ADD   TMP
        STA   CH
        HLT
    TMP, HEX  0
```

To form a double-precision 2's complement of subtrahend BH + BL, a 1's complement is formed and 1 added <u>once</u>.

Thus, BL is complemented and incremented while BH is only complemented.

Location TMP saves the carry from E while BH is complemented.

6-19 $z = x \oplus y = xy' + x'y = [(xy')' \cdot (x'y)']'$

```
        LDA   Y              AND   TMP
        CMA                  CMA
        AND   X              STA   Z
        CMA                  HLT
        STA   TMP
        LDA   X          X,  —
        CMA              Y,  —
        AND   Y          Z,  —
        CMA            TMP,  —
```

6-20

```
        LDA   X
        CLE
        CIL
        SZE
        BUN   ONE
        SPA
        BUN   OVF
        BUN   EXT
  ONE,  SNA
        BUN   OVF
  EXT,  HLT
```

/ zero to low order bit; sign bit in E

40
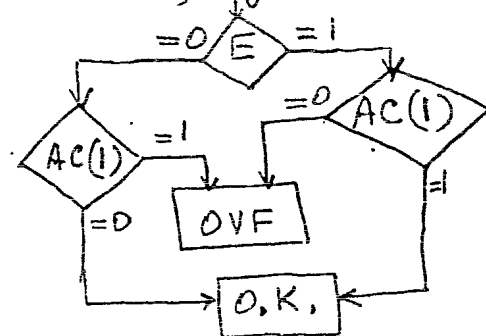
6-21 Calling program   subroutine

```
        BSA    SUB                SUB,   HEX 0
        HEX    1234  /subtrahend         LDA  SUB I
        HEX    4321  /minuend            CMA
        HEX    0     /difference          INC
                                         ISZ  SUB
                                         ADD  SUB I
                                         ISZ  SUB
                                         STA  SUB I
                                         ISZ  SUB
                                         BUN  SUB I
```

6-22

Calling Program

```
        BSA    CMP                       CMA
        HEX    100  /starting address    INC
        DEC    32   /number of words     STA   CTR
                                   LOP,  LDA   PTR I
Subroutine                               CMA
                                         STA   PTR I
        CMP,   HEX 0                     ISZ   PTR
               LDA  CMP I                ISZ   CTR
               STA  PTR                  BUN   LOP
               ISZ  CMP                  ISZ   CMP
               LDA  CMP I                BUN   CMP I
                                   PTR,  ——
                                   CTR,  ——
```

6-23   CR4, HEX 0

```
              CIR
              CIR
              CIR
              CIR
              BUN CR4 I
```

E	AC	AC HEX
I	0000 0111 1001 1100	079C
I	1001 0000 0111 1001	9079

6-24

```
        LDA    ADS                       BUN  LOP
        STA    PTR                       HLT
        LDA    NBR                 ADS,  HEX  400
        STA    CTR                 PTR,  HEX  0
LOP,    BSA    IN2  /subroutine Table 6-20   NBR,  DEC  -512
        STA    PTR I               CTR,  HEX  0
        ISZ    PTR
        ISZ    CTR
```

6-25

```
LDA    WRD                          STA    CH2
AND    MS1                          HLT
STA    CH1                   WRD,   HEX —
LDA    WRD                   CH1,   HEX —
AND    MS2                   CH2,   HEX —
CLE                          MS1,   HEX 00FF
BSA    SR8  /subroutine to   MS2,   HEX FF00
            shift right
            eight times
```

6-26
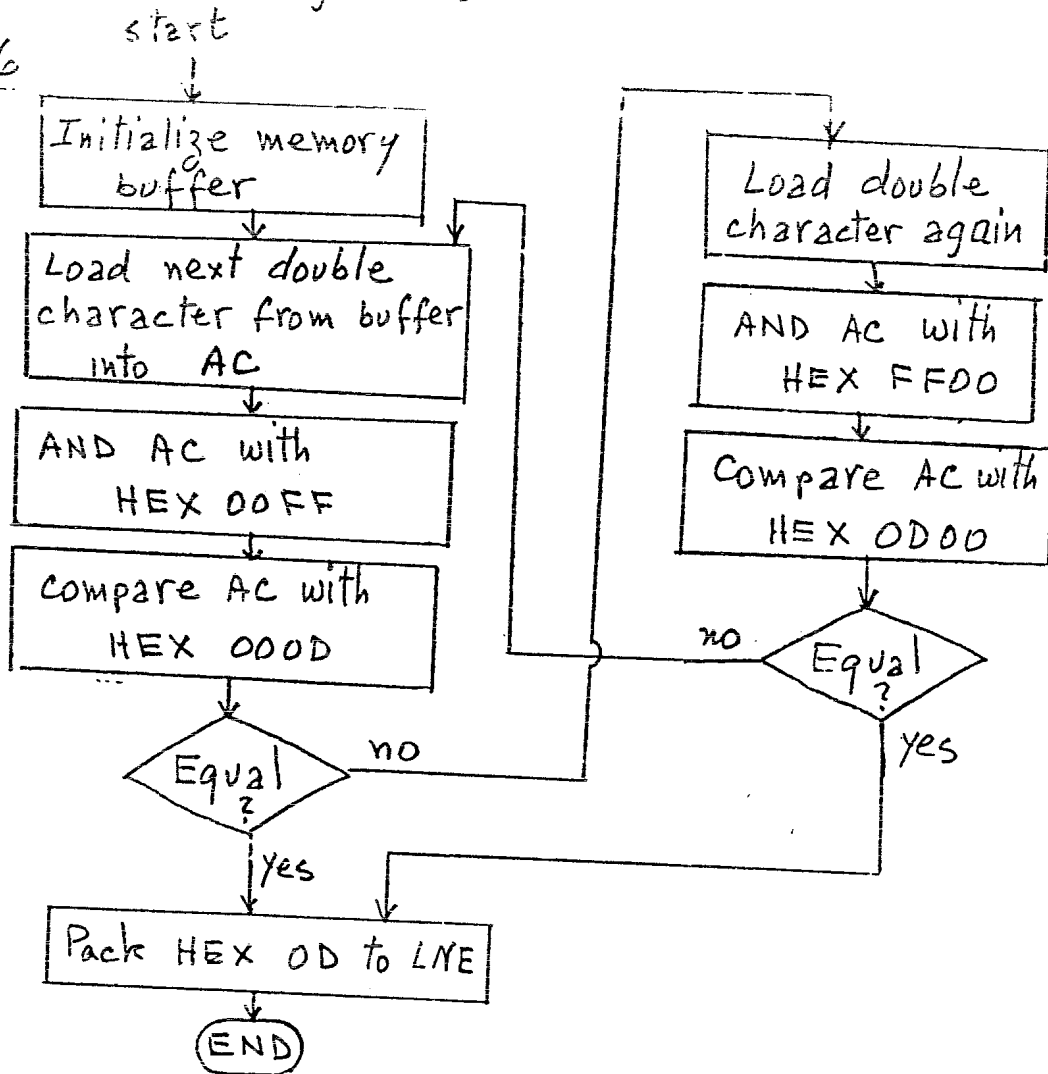
6-27

Location	Hex code			
200	3213	SRV,	STA	SAC
201	7080		CIR	
202	3214		STA	SE
203	F200		SKI	
204	4209		BUN	NXT
205	F800		INP	
206	F400		OUT	
207	B215		STA	PTI I
208	6215		ISZ	PTI
209	F100	NXT,	SKO	
20A	420E		BUN	EXT
20B	A216		LDA	PT2 I
20C	F400		OUT	
20D	6216		ISZ	PT2
20E	2214	EXT,	LDA	SE
20F	7040		CIL	
210	2213		LDA	SAC
211	F080		ION	
212	C000		BUN	ZRO I
213	0000	SAC,	—	
214	0000	SE,	—	
215	0000	PTI,	—	
216	0000	PT2,	—	

6-28

```
SRV,   STA  SAC              NXT,   LDA  MOD
       CIR                          SZA
       STA  SE                      BUN  EXT
       LDA  MOD   /check MOD  service  SKO
       CMA                    output   BUN  EXT
       SZA                    device   LDA  PT2 I
       BUN  NXT  / MOD≠ all 1's        OUT
       SKI ⎤                          ISZ  PT2
       BUN  NXT ⎥ Service
       INP ⎥ input         EXT,  continue as
       OUT ⎥ Device               in Table 6-23
       STA  PTI I ⎥
       ISZ  PTI ⎦
       BUN  EXT  / MOD≠0
```

CHAPTER 7

7-1

A microprocessor is a small size CPU (computer on a chip)
Microprogram is a program for a sequence of microoperations.
The control unit of a microprocessor can be hardwired or
microprogrammed, depending on the specific design.
A microprogrammed computer does not have to be a
microprocessor.

7-2

Hardwired control, by definition, does not contain a
control memory.

7-3

Microoperation — an elementary digital computer operation.
Microinstruction — an instruction stored in control memory.
Microprogram — a sequence of microinstructions.
Microcode — same an microprogram.

7-4
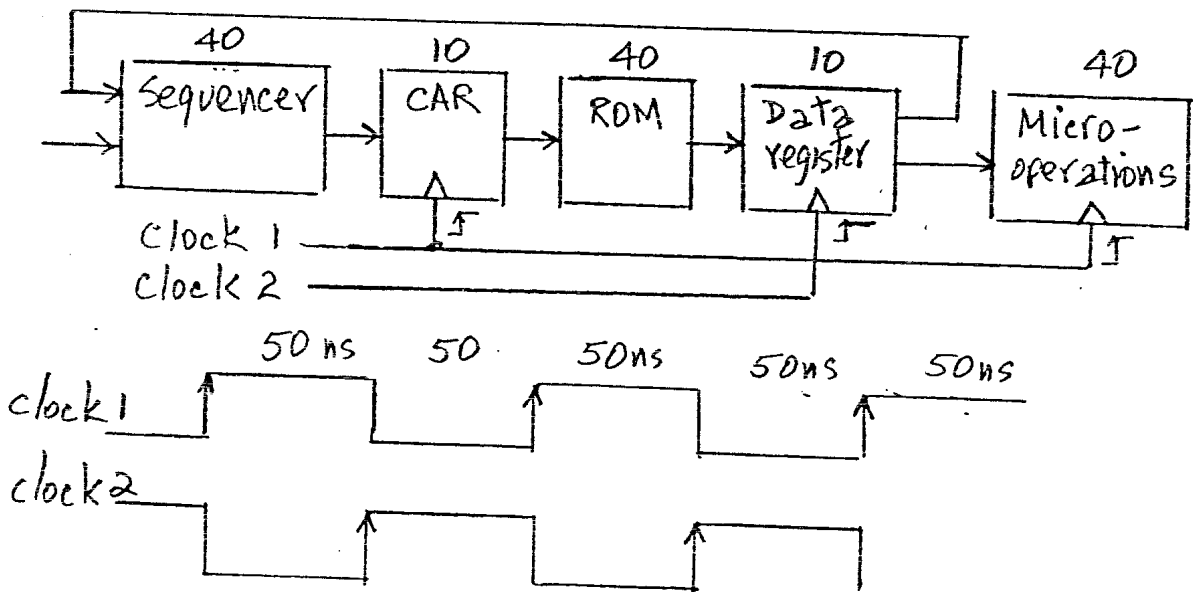
$$\text{frequency of each clock} = \frac{1}{100 \times 10^{-9}} = \frac{1000}{100} \times 10^6 = 10 \text{ MHz}$$

If the data register is removed, we can use a
single phase clock with a frequency of $\frac{1}{90 \times 10^{-9}} = 11.1 \text{ MHz}$

7-5

Control memory $= 2^{10} \times 32$

(a)

6	10	16	$= 32$ bits
Select	Address	Microoperations	
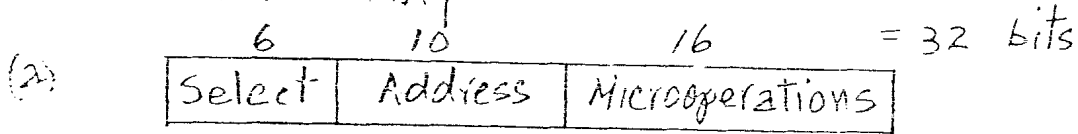
(b) 4 bits

(c) 2 bits

7-6

Control memory $= 2^{12} \times 24$

(a) 12 bits

(b) 12 bits

(c) 12 multiplexers, each of size 4-to-1 line.

7-7

(a) 0001000 $= 8$

(b) 0101100 $= 44$

(c) 0111100 $= 60$

7-8

op code $= 6$ bits

control memory
address $= 11$ bits

00|XXXXXX|000

7-9

n inputs → $2^n \times m$ ROM → m outputs
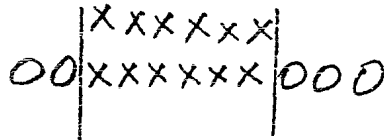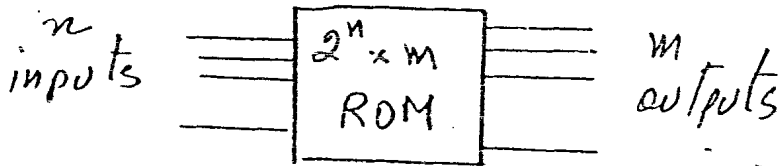
The ROM can be programmed to provide any desired address for a given inputs from the instruction.

7-10

Either multiplexers, three-state gates, or gate logic (equivalent to a mux) are needed to transfer information from many sources to a common destination.

45

7-11

	F1	F2	F3
(a)	011	110	000
(b)	000	100	101
(c)	100	101	000

INCAC INCDR NOP

NOP READ INCPC

DRTAC ACTDR NOP

7-12

				Binary
(a) READ	DR ← M[AR]	F2 = 100		000 100 101
DRTAC	AC ← DR	F3 = 101		

(b) ACTDR	DR ← AC	F2 = 101	000 100 101
DRTAC	AC ← DR	F1 = 100	

(c) ARTPC   PC ← AR   F3 = 110

DRTAC   AC ← DR   F1 = 100 ⎤ Impossible.

WRITE   M[AR] ← DR   F1 = 111 ⎦ Both use F1

7-13

If I = 0, the operand is read in the first microinstruction and added to AC in the second.

If I = 1, the effective address is read into DR and control goes to INDR2. The subroutine must read the operand into DR.

INDR2:   DRTAR   U   JMP   NEXT

READ   U   RET   ---

7-14

(a) Branch if S = 0 and Z = 0 (positive and non-zero AC) — See last instruction in Problem 7-16.

(b)

40: 000 000 000   10 00   1000000

41: 000 000 000   11 00   1000000

42: 000 000 000   01 01   1000011

43: 000 000 110   00 00   1000000

7-15

(a)
60:	CLRAC, COM	U	JMP	INDRCT
61:	WRITE, READ	I	CALL	FETCH
62:	ADD, SUB	S	RET	63 (VEXT)
63:	DRTAC, INCDR	Z	MAP	60

(b)

60: Cannot increment and complement Ac at the same time. With a JMP to INDRCT, control does not return to 61.

61: Cannot read and write at the same time. The CALL behaves as a JMP since there is no return from FETCH.

62: Cannot add and subtract at the same time. The RET will be executed independent of S.

63: The MAP is executed irrespective of Z or 60.

7-16

ORG 16

AND:	... NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
ANDOP:	AND	U	JMP	FETCH

ORG 20

SUB:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	SUB	U	JMP	FETCH

ORG 24

ADM:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	DRTAC, ACTDR	U	JMP	NEXT
	ADD	U	JMP	EXCHANGE+2

(Table 7-2)

```
            ORG 28
BTCL :  NOP              I     CALL   INDRCT
        READ             U     JMP    NEXT
        DRTAC, ACTDR     U     JMP    NEXT
        COM              U     JMP    ANDOP
```

```
            ORG 32
BZ :    NOP              Z     JMP    ZERO
        NOP              U     JMP    FETCH
ZERO:   NOP              I     CALL   INDRCT
        ARTPC            U     JMP    FETCH
```

```
            ORG 36
SEQ :   NOP              I     CALL   INDRCT
        READ             U     JMP    NEXT
        DRTAC, ACTDR     U     JMP    NEXT
        XOR (or SUB)     U     JMP    BEQ1
            ORG 69
BEQ1:   DRTAC, ACTDR     Z     JMP    EQUAL
        NOP              U     JMP    FETCH
EQUAL:  INCPC            U     JMP    FETCH
```

```
            ORG 40
BPNZ :  NOP              S     JMP    FETCH
        NOP              Z     JMP    FETCH
        NOP              I     CALL   INDRCT
        ARTPC            U     JMP    FETCH
```

7-17

ISZ:
NOP		I	CALL	INDRCT
READ		U	JMP	NEXT
INCDR		U	JMP	NEXT
DRTAC, ACTDR		U	JMP	NEXT (or past INDRCT)
DRTAC, ACTDR		Z	JMP	ZERO
WRITE		U	JMP	FETCH
ZERO: WRITE, INCPC		U	JMP	FETCH

7-18

BSA:
NOP		I	CALL	INDRCT
PCTDR, ARTPC		U	JMP	NEXT
WRITE, INCPC		U	JMP	FETCH

7-19

From Table 7-1:

$F3 = 101 \ (5) \quad PC \leftarrow PC+1$

$F3 = 110 \ (6) \quad PC \leftarrow AR$

FROM:
F3 output 6 — Load
F3 output 5 — Increment
AR → 10 → PC → clock
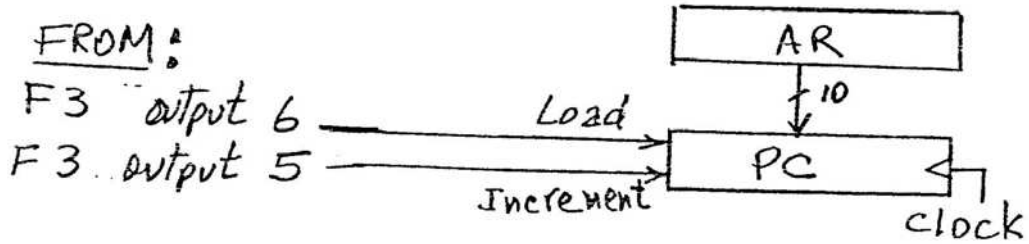
7-20

A field of 5 bits can specify $2^5 - 1 = 31$ microoperations

A field of 4 bits can specify $2^4 - 1 = 15$ microoperations

9 bits   46 microoperations

7-21

See Fig. 8-2(b) for control word example.

(a) 16 registers need 4 bits; ALU need 5 bits, and the shifter need 3 bits, to encode all operations.

4	4	4	5	3	= 20 bits total
SRC1	SRC2	DEST	ALU	SHIFT	

(c)
R5	R6	R4	ADD	SHIFT	
0101	0110	0100	00100	000	R4 ← R5+R6

49

7-22

I_2	I_1	I_0	T	S_1	S_0	L	
0	0	0	0	0	1	0	AD (1)
0	0	0	1	0	0	0	INC(0)
0	0	1	0	0	1	0	AD(1)
0	0	1	1	0	1	0	AD(1)
0	1	0	0	0	0	0	INC(0)
0	1	0	1	0	1	0	AD(1)
0	1	1	0	1	0	0	RET(2)
0	1	1	1	1	0	0	RET(2)
1	0	0	0	0	0	0	INC(0)
1	0	0	1	0	0	0	INC(0)
1	0	1	0	0	1	0	AD(1)
1	0	1	1	0	1	0	AD(1)
1	1	0	0	0	0	0	INC(0)
1	1	0	1	0	1	1	CALL(1)
1	1	1	0	1	1	0	MAP(3)
1	1	1	1	1	1	0	MAP(3)
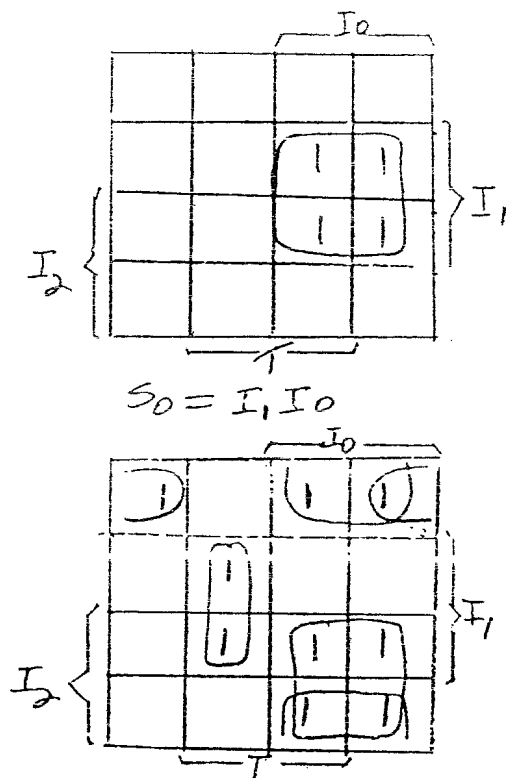
$$S_0 = I_1 I_0$$

$$S_0 = I_2 I_0 + I_1' I_0 + I_1 I_0' T + I_2' I_1' T'$$

$$L = I_2 I_1 I_0' T$$

7-23

(a) See Fig. 4-8 (chapter 4)

(b)
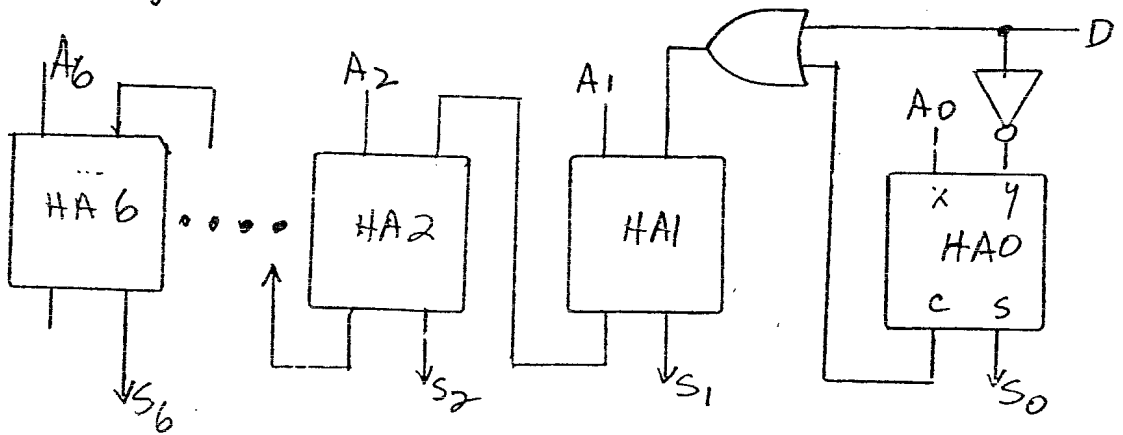
7-24

P is used to determine the polarity of the selected status bit.
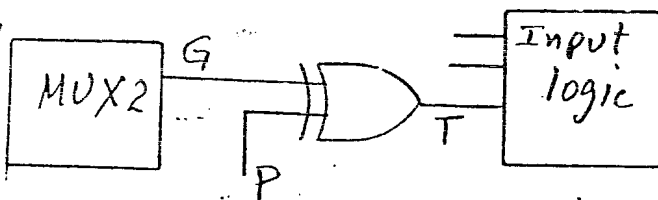
when $P=0$, $T=G$ because $G \oplus 0 = G$,

When $P=1$, $T=G'$ because $G \oplus 1 = G'$

when G is the value of the selected bit in MUX2

8-1

(a) 32 multiplexers, each of size 16×1.

(b) 4 inputs each, to select one of 16 registers.

(c) 4-to-16-line decoder

(d) $32 + 32 + 1 = 65$ data input lines

$32 + 1 = 33$ data output lines.

(e)

4	4	4	6	= 18 bits
SELA	SELB	SELD	OPR	
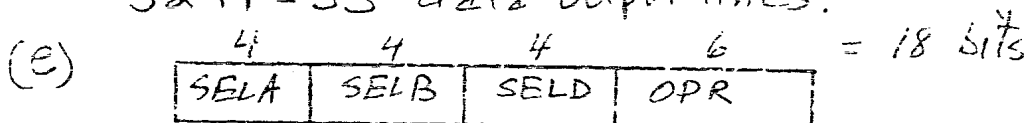
8-2

$30 + 80 + 10 = 120$ nSec.

(The decoder signals propagate at the same as the muxs)

8-3

		SELA	SELB	SELD	OPR	Control word
(a)	R1 ← R2 + R3	R2	R3	R1	ADD	010 011 001 00010
(b)	R4 ← $\overline{R4}$	R4	—	R4	COMA	100 xxx 100 01110
(c)	R5 ← R5 − 1	R5	—	R5	DECA	101 xxx 101 00110
(d)	R6 ← shl R1	R1	—	R6	SHLA	001 xxx 110 11000
(e)	R7 ← Input	Input	—	R7	TSFA	000 xxx 111 00000

8-4

	Control word	SELA	SELB	SELD	OPR	Microoperation
(a)	001 010 011 00101	R1	R2	R3	SUB	R3 ← R1 − R2
(b)	000 000 000 00000	Input	Input	None	TSFA	Output ← Input
(c)	010 010 010 01100	R2	R2	R2	XOR	R2 ← R2 ⊕ R2
(d)	000 001 000 00010	Input	R1	None	ADD	Output ← Input + R1
(e)	111 100 011 10000	R7	R4	R3	SHRA	R3 ← shr R7

8-5

(a) Stack full with 64 items.

(b) Stack empty.

8-6

PUSH : $M[SP] \leftarrow DR$

$SP \leftarrow SP-1$

POP : $SP \leftarrow SP+1$

$DR \leftarrow M[SP]$

	3998
	3999
SP →	4000
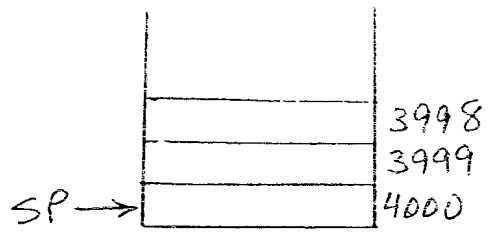
8-7

(a) $AB * CD * EF * + +$

(b) $AB * ABD * CE * + * +$

(c) $FG + E * CD * + B * A +$

(d) $ABCDE + * + * FGH + * /$

8-8

(a) $\dfrac{A}{B-(D+E)*C}$

(b) $A+B-\dfrac{C}{D*E}$

(c) $\dfrac{A}{B*C} - D + \dfrac{E}{F}$

(d) $(((F+G)*E+D)*C+B)*A$

8-9

$(3+4)[10(2+6)+8] = 616$

RPN : $3\ 4 + 2\ 6 + 10 * 8 + *$

				6		10		8		
	4		2	2	8	8	80	80	88	
3	3	7	7	7	7	7	7	7	7	616

$3\ \ 4\ \ +\ \ 2\ \ 6\ \ +\ \ 10\ \ *\ \ 8\ \ +\ \ *$

8-10

WRITE (if not full):

$M[WC] \leftarrow DR$

$WC \leftarrow WC + 1$

$ASC \leftarrow ASC + 1$

READ: (if not empty)

$DR \leftarrow M[RC]$

$RC \leftarrow RC + 1$

$ASC \leftarrow ASC - 1$

Memory may wrap-around

Read from 7 to 0

RC

write

WC

3 bits

ASC

Empty ← all 0's   all 1's → Full

FIFO

	0
	1
	2
	3
	4
	5
	6
	7

8-11

8	12	12	= 32 bits
Op code	Address 1	Address 2	

Two address instructions
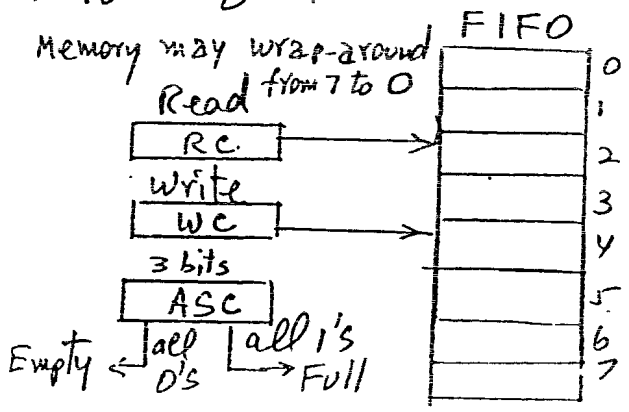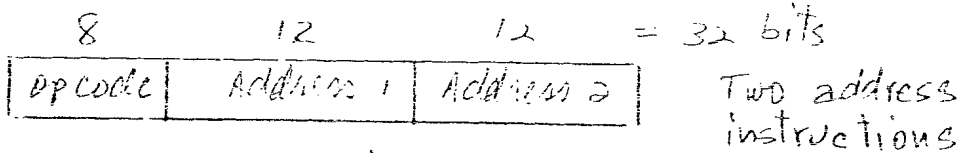
$2^8 = 256$ combinations.

$256 - 250 = 6$ combinations can be used for one address

Op code	Address
6 × 2^{12}	

one address instructions
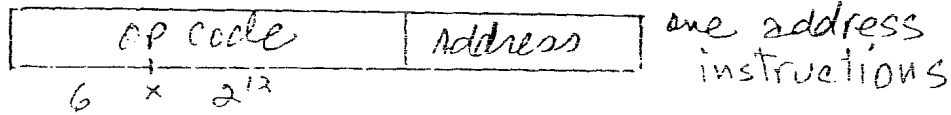
Maximum number of one address instruction:

$$= 6 \times 2^{12} = 24,576$$

8-12

(d) RPN: $X A B - C + D E * F - * G H K * + / =$

8-13

$$256 K = 2^8 \times 2^{10} = 2^{18}$$

Opcode	Mode	Register	Address
5	3	6	18

$= 32$

address = 18 bits
Mode = 3 "
Register = 6 "
                    27 bits
Op code 5
                    32 bits
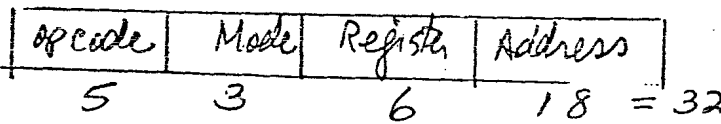
8-14

Z = Effective address

(a) Direct: $Z = Y$

(b) Indirect: $Z = M[Y]$

(c) Relative: $Z = Y + W + 2$

(d) Indexed: $Z = Y + X$

		opcode Mode
PC	W	
	W+1	Y
XR = X	W+2	Next instruction
	Z	operand
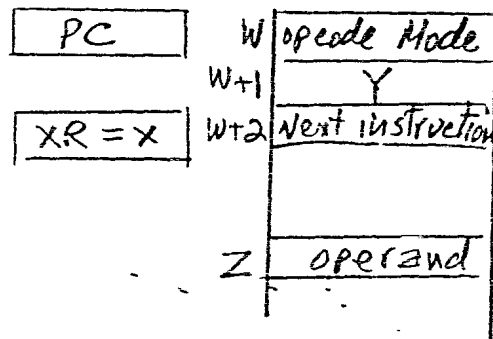
8-15

(a) Relative address = $500 - 751 = -251$

(b) $251 = 00011111011$; $-251 = 11100000101$

(c) $PC = 751 = 00101110111$; $500 = 00011111 0100$

$PC = 751 = 00101110111$
$RA = -251 = {}^+11100000101$
$EA = 500 = 00011110100$

53

8-16 Assuming one word per instruction or operand.

Computational type	Branch type
Fetch instruction	Fetch instruction
Fetch effective address	Fetch effective address
Fetch operand	and transfer to PC
3 memory references	2 memory references.

8-17

The address part of the indexed mode instruction must be set to zero.

8-18

Effective address

(a) Direct : 400

(b) Immediate : 301

(c) Relative : 302+400 = 702

(d) Reg. Indirect : 200

(e) Indexed : 200+400 = 600

Memory

PC → 300	op code Mode
301	400
302	Next instruction

R1 = 200

8-19

$1=C$ $0=C$ $1=C$ 0 = Reset initial carry

```
  6E   C3   56   7A
  13   55   6B   8F
 ────  ───  ───  ───
  82   18   C2   09    Add with carry
```

8-20

```
10011100  AND    10011100  OR    100.11100  XOR
10101010         10101010        10101010
────────         ────────        ─────────
10001000         11111110        00110110
```

8-21

(a) AND with : 0000000011111111

(b) OR with : 0000000011111111

(c) XOR with : 0000111111110000

54

8-22

Initial: 01111011 C=1
SHR: 00111101
SHL: 11110110
SHRA: 00111101
SHLA: 11110110 (overflow)
ROR: 10111101
ROL: 11110110
RORC: 10111101
ROLC: 11110111

8-23

+83 = 01010011   -83 = 10101101
+68 = 01000100   -68 = 10111100

(a) -83 $\quad$ 10101101
$\quad$ +68 $\quad$ + 01000100
$\quad$ $\overline{-15}$ $\quad$ $\overline{11110001}$
$\qquad$ (in 2's complement)

(b) $\qquad$ 1 0 ← carries
$\quad$ −68 $\quad$ 10111100
$\quad$ −83 $\quad$ + 10101101
$\quad$ $\overline{-151}$ $\quad$ $\overline{01101001}$
$\quad$ -128 (overflow)

(c) $-68 = 10111100$
$\quad$ $-34 = 11011110$

(d) $-83 = \overset{\oplus = 1}{10101101}$
$\quad$ $-166 \neq 01011010$
$\qquad$ overflow

8-24

$$Z = F_0' F_1' F_2' F_3' F_4' F_5' F_6' F_7' = (F_0 + F_1 + F_2 + F_3 + F_4 + F_5 + F_6 + F_7)'$$

8-25

(a) $\quad$ 11
$\quad$ 72 $\quad$ 0111 0010
$\quad$ C6 $\quad$ 1100 0110
$\quad$ $\overline{138}$ $\quad$ $\overline{0011 1000}$
$\quad$ C=1 S=0 Z=0 V=0

(b) $\quad$ 01
$\quad$ 72 $\quad$ 0111 0010
$\quad$ 1E $\quad$ 0001 1110
$\quad$ $\overline{90}$ $\quad$ $\overline{1001 0000}$
$\quad$ C=0 S=1 Z=0 V=1

(c) $9A = 10011010 \rbrace$ 2's comp.
$\quad$ 01100110 ←
$\quad$ 72 $\quad$ 0111 0010
$\quad$ $\overline{D8}$ $\quad$ $\overline{1101 1000}$
$\quad$ C=0 S=1 Z=0 V=1
$\quad$ (Borrow = 1

(d) $\quad$ 72 $\quad$ 0111 0010
$\quad$ 8D $\quad$ 1000 1100
$\quad$ $\overline{00}$ $\quad$ $\overline{0000 0000}$
$\quad$ C=0 S=0 Z=1 V=0

(e) C=0 S=0 Z=1 V=0

8-26

$C=1$ if $A<B$, therefore $C=0$ if $A \geq B$

$Z=1$ if $A=B$, therefore $Z=1$ if $A \neq B$

For $A>B$ we must have $A \geq B$ provided $A \neq B$

$\qquad$ or $C=0$ and $Z=0$ $(c'z')=1$

For $A \leq B$ we must have $A<B$ or $A=B$

$\qquad$ or $C=1$ or $Z=1$ $(c+z)=1$

8-27

$A \geq B$ implies that $A-B \geq 0$ (positive or zero)

$\quad$ sign $s=0$ if no overflow (positive)

$\quad$ or $s=1$ if overflow (sign reversal)

$\quad$ Boolean expression: $s'v'+sv=1$ or $(s \oplus v)=0$

$A<B$ is the complement of $A \geq B$ ($A-B$ negative)

$\quad$ then $s=1$ if $V=0$

$\quad$ or $s=0$ if $V=1$ $\qquad (s \oplus v)=1$

$A>B$ Implies $A \geq B$ but not $A=B$

$\qquad (s \oplus v)=0$ and $z=0$

$A \leq B$ Implies $A<B$ or $A=B$

$\qquad s \oplus v=1$ or $z=1$

8-28
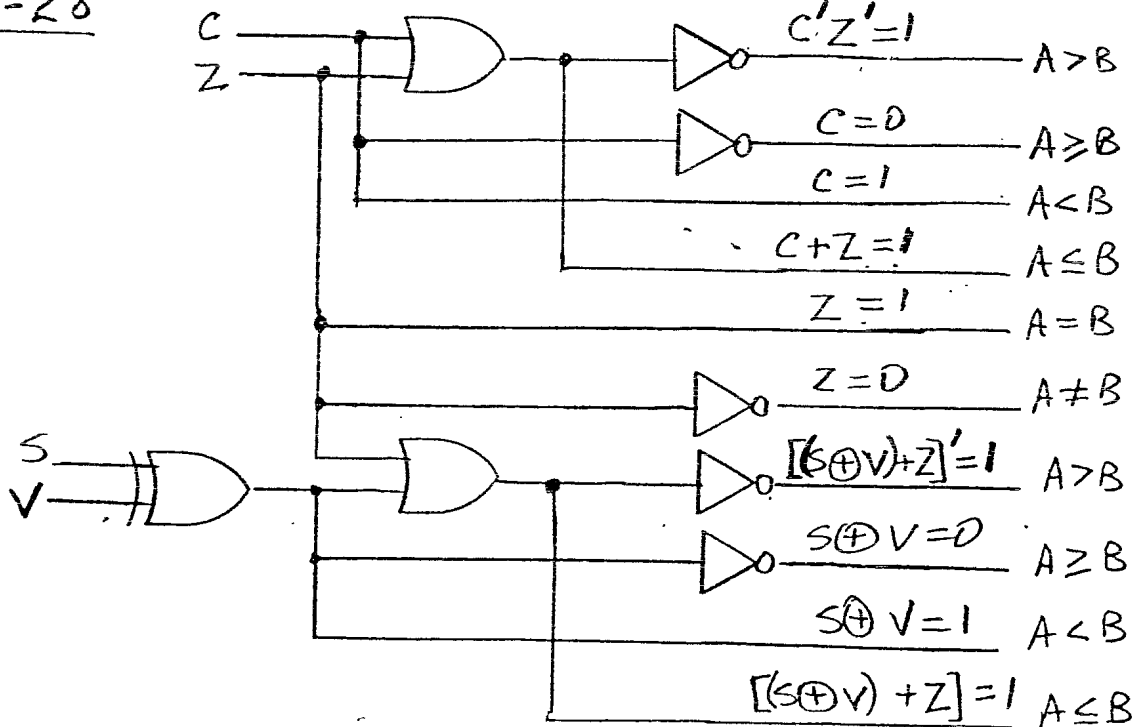
8-29

		Unsigned	Signed

$$A = 01000001 \quad\quad 65 \quad\quad\quad +65$$
$$B = 10000100 \quad\quad 132 \quad\quad\quad -124$$
$$A+B = 11000101 \quad\quad 197 \quad\quad\quad -59$$

(c) $C = 0 \quad\quad Z = 0 \quad\quad S = 1 \quad\quad V = 0$

(d) BNC $\quad\quad$ BNZ $\quad\quad$ BM $\quad\quad$ BNV

8-30

(a) $A = 01000001 = 65$
$B = 10000100 = 132$
$A-B = 10111101 = -67$ (2's comp. of 01000011)

(b) C (borrow) $= 1$; $Z = 0 \quad\quad 65 < 132$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad A < B$

(c) BL, BLE, BNE

8-31

(a) $A = 01000001 = +65$
$B = 10000100 = -124$
$A-B = 10111101 \quad +189 = \overset{+}{010111101}$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad 9 \text{ bits}$

(b) $S = 1$ (sign reversal) $\quad\quad +189 > 127$
$\quad Z = 0$
$\quad V = 1$ (overflow) $\quad\quad 65 > -124$
$\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad A > B$
(c) BGT, BGE, BNE

8-32

	PC	SP	Top of Stack
Initial	1120	3560	5320
After CALL	6720	3559	1122
After RETURN	1122	3560	5320

8-33

Branch instruction- Branch without being able to return,

Subroutine call — Branch to subroutine and then return to calling program.

Program interrupt — Hardware initiated branch with possibility to return,

8-34

See Sec. 8-7 under "Types of Interrupts".

8-35

(a) $SP \leftarrow SP-1$
$M[SP] \leftarrow PSW$
$SP \leftarrow SP-1$
$M[SP] \leftarrow PC$
$TR \leftarrow IAD$ (TR is a temporary register)
$PSW \leftarrow M[TR]$

$TR \leftarrow TR+1$
$PC \leftarrow M[TR]$
Go to fetch phase.

(b) $PC \leftarrow M[SP]$
$SP \leftarrow SP+1$
$PSW \leftarrow M[SP]$
$SP \leftarrow SP+1$

8-37

Window size $= L + 2C + G$
  Computer 1: $10 + 12 + 10 = 32$
  Computer 2: $8 + 16 + 8 = 32$
  Computer 3: $16 + 32 + 16 = 64$

Register file $= (L+C) W + G$
  Computer 1: $(10+6) 8 + 10 = 16 \times 8 + 10 = 138$
  Computer 2: $(8+8) 4 + 8 = 16 \times 4 + 8 = 72$
  Computer 3: $(16+16) 16 + 16 = 32 \times 16 + 16 = 528$

8-38

(a) SUB R22, #1, R22 $R22 \leftarrow R22 - 1$ (Subtract 1)

(b) XOR R22, #-1, R22 $R22 \leftarrow R22 \oplus$ all 1's $(x \oplus 1 = x')$

(c) SUB R0, R22, R22 $R22 \leftarrow 0 - R22$

(d) ADD R0, R0, R22 $R22 \leftarrow 0 + 0$

(e) SRA R22, #2, R22   Arithmetic shift right twice

(f) OR R1, R1, R1 $R1 \leftarrow R1 \lor R1$

   or ADD R1, R0, R1 $R1 \leftarrow R1 + 0$

   or SLL R1, #0, R1   shift left 0 times

8-39

(a) JMP Z, #3200, (R0) $PC \leftarrow 0 + 3200$

(b) JMPR Z, -200 $PC \leftarrow 3400 + (-200)$

9-1
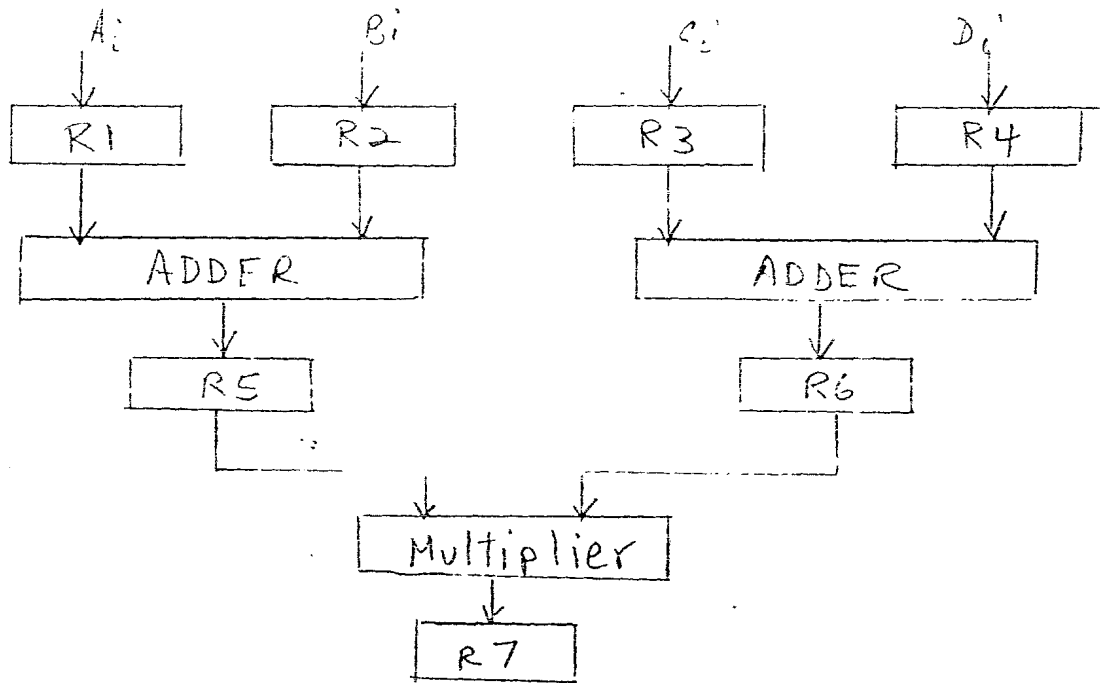
9-2

Segment	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8					
2		T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8				
3			T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8			
4				T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8		
5					T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	
6						T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8

$$(k - n - 1)t_p = 6 + 8 - 1 = 13 \text{ cycles}$$

9-3

$k = 6$ segment
$n = 200$ tasks $\quad (k + n - 1) = 6 + 200 - 1 = 205$ cycles

9-4

$t_n = 50$ nS
$k = 6$
$t_p = 10$ nS
$n = 100$

$$S = \frac{n t_n}{(k + n - 1)t_r} = \frac{100 \times 50}{(6 - 99) \times 10} = 4.76$$

$$S_{max} = \frac{t_n}{t_p} = \frac{50}{10} = 5$$

9-5

(a) $t_p = 45 + 5 = 50$ ns $k = 3$

(b) $t_n = 40 + 45 + 15 = 100$ ns

(c) $S = \dfrac{n t_n}{(k+n-1) t_p} = \dfrac{10 \times 100}{(3+9)50} = 1.67$ for $n = 10$

$$= \dfrac{100 \times 100}{(3+99)50} = 1.96 \quad \text{for } n = 100$$

(d) $S_{max} = \dfrac{t_n}{t_p} = \dfrac{100}{50} = 2$

9-6

(a) See discussion in Sec. 10-3 on array multipliers. There are $8 \times 8 = 64$ AND gates in each segment and an 8-bit binary adder (in each segment).

(b) There are 7 segments in the pipeline

(c) Average time $= \dfrac{k+n-1}{n} t_p = \dfrac{(n+6)\,30}{n}$

For $n = 10$ $t_{AV} = 48$ ns

For $n = 100$ $t_{AV} = 31.8$ ns

For $n \to \infty$ $t_{AV} = 30$ ns

To increase the speed of multiplication, a carry-save (Wallace tree) adder is used to reduce the propagation time of the carries.

9-7

(a) Clock cycle $= 95 + 5 = 100$ ns (time for segment 3)

for $n = 100$; $k = 4$, $t_p = 100$ ns.

Time to add 100 numbers $= (k+n-1) t_p = (4+99)\,100$

$$= 10,300 \text{ ns} = 10.3 \mu s$$

(b) Divide segment 3 into two segments of $50+5=55$ and $45+5=50$ ns. This makes $t_p = 55$ ns; $k = 5$

$(k+n-1) t_p = (5+99)\,55 = 5,720 \text{ ns} = 5.72 \mu s$

9-8 Connect output of adder to input $E \times 2^b$ in a feedback path and use input $A \times 2^a$ for the data X_1 through X_{100}. Then use a scheme similar to the one described in conjunction with the adder pipeline in Fig. 9-12.

9-9

One possibility is to use the six operations listed in the beginning of Sec. 9-4.

9-10

See Sec. 9-4: (1) prefetch target instruction; (b) use a branch target buffer; (c) use a loop buffer; (d) use branch prediction. (Delayed branch is a software procedure)

9-11

1. Load $R1 \leftarrow M[312]$
2. Add $R2 \leftarrow R2 + M[313]$
3. Increment $R3$
4. Store $M[314] \leftarrow R3$

1	2	3	4th step
FI	DA	FO	EX
	FI	DA	FO
		FI	DA
			FI

Segment EX : Transfer memory word to R1.
Segment FO : Read $M[313]$.
Segment DA : Decode (increment) instruction.
Segment FI : Fetch (the store) instruction from memory.

9-12

Load: $R1 \leftarrow$ Memory
Increment: $R1 \leftarrow R1 + 1$

1	2	3	4
I	A	E	
	I	A	E

R1 is loaded in E _____
It's too early to increment it in A

9-13

Insert a No-op instruction between the two instructions in the example of Problem 9-12 (above).

9-14

		1	2	3	4	5	6	7
101	Add R2 to R3	I	A	E				
102	Branch to 104		I	A	E			
103	Increment R1			-	-			
104	Store R1					I	A	E
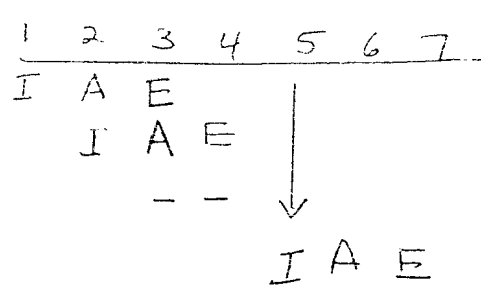
9-15 Use example of Problem 9-14.

		1	2	3	4	5	6
101	Branch to 105	I	A	E	↓		
102	Add R2 to R3		I	A	E		
103	No-operation			I	A	E	
104	Increment R1				↓		
105	Store R1					I	A	E
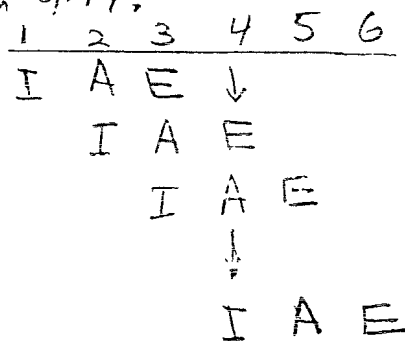
9-16

(a) There are 40 product terms in each inner product, $40^2 = 1,600$ inner products must be evaluated, one for each element of the product matrix.

(b) $40^3 = 64,000$

9-17

$8 + 60 + 4 = 72$ clock cycles for each inner product. There are $60^2 = 3600$ inner products. Product matrix takes $3600 \times 72 = 259,200$ clock cycles to evaluate.

9-18

memory array 1 use addresses: 0, 4, 8, 12, ..., 1020. Array 2: 1, 5, 9, 13, ..., 1021; Array 3: 2, 6, 10, ..., 1022. Array 4: 3, 7, 11, ..., 1023.

9-19

$$\frac{250 \times 10^9}{100 \times 10^6} = 2,500 \text{ sec} = 41.67 \text{ minutes}$$

9-20

Divide the 400 operations into each of the four processors. Processing time is: $\frac{400}{4} \times 40 = 4,000$ nsec

Using a single pipeline, processing time is $= 400 \times 10 = 4,000$ nsec

10-1

10-2
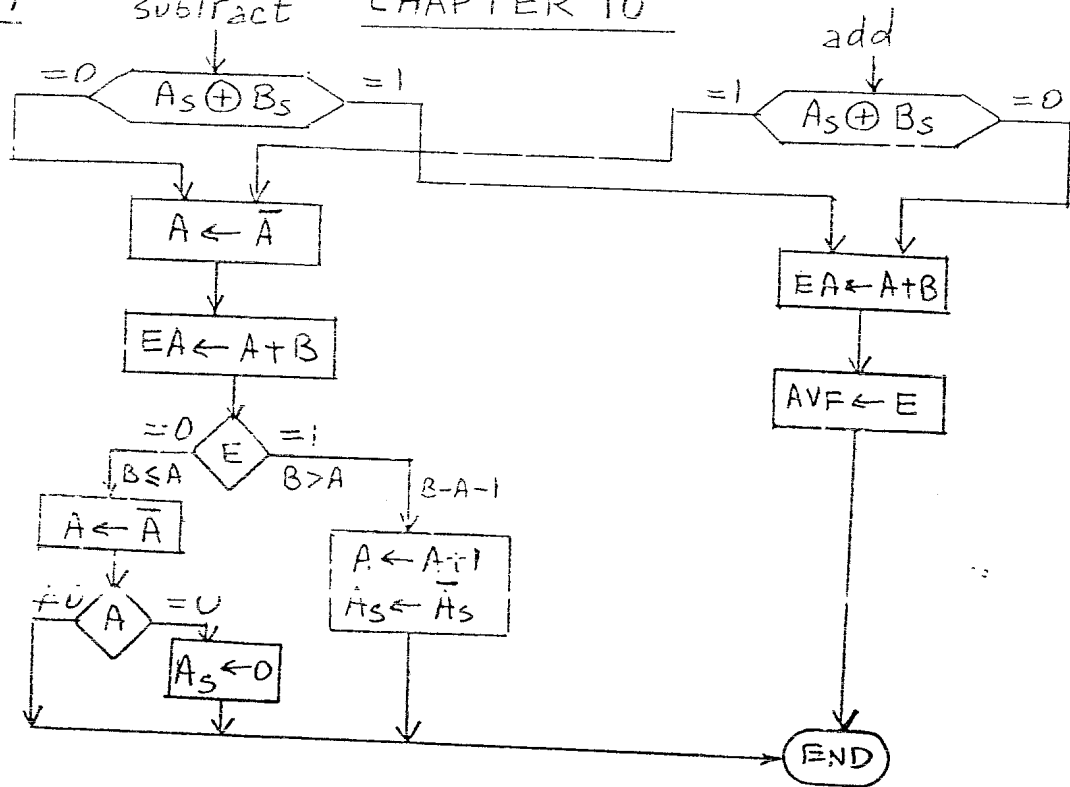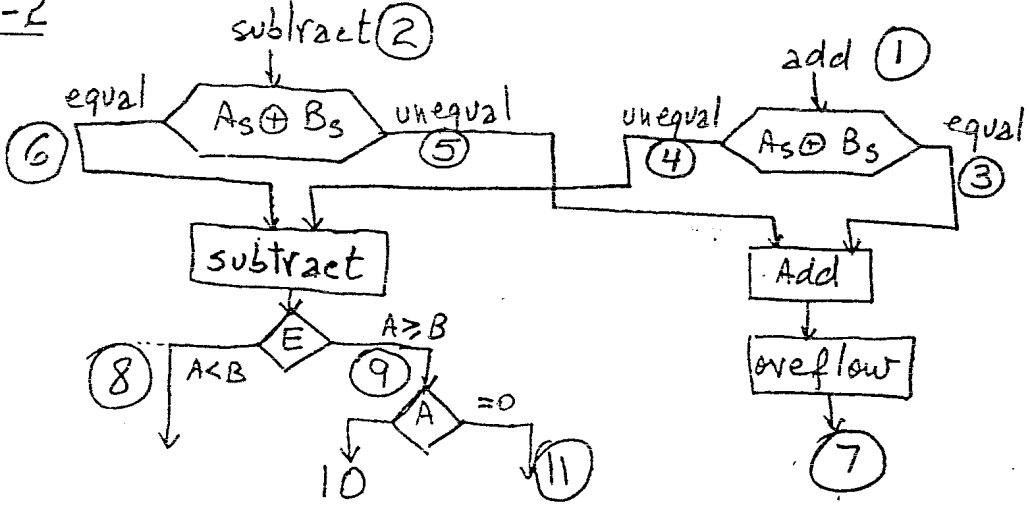
$2^6 - 1 = 63$, Oveflow if sum greater than $|63|$

(a) $(+45) + (+31) = 76$ ① ③ ⑦ ← Path.   AVF = 1

(b) $(-31) + (-45) = -76$ ① ③ ⑦   AVF = 1

(c) $(+45) - (+31) = 14.$ ② ⑥ ⑨ ⑩   AVF = 0

(d) $(+45) - (+45) = 0$ ② ⑥ ⑨ ⑪   AVF = 0

(e) $(-31) - (+45) = -76$ ② ⑤ ⑦ .   AVF = 1

10-3

(a) +35   0 100011   (b) −35   1 011101
    +40   0 101000   −40   1 011000
    +75   1/001011   −70   0/110101

F=0 E=1 ←———— Carries ———→ F=1 E=0

F⊕E=1 ; overflow   F⊕E=1 ; overflow

10-4   (a)   (b)   (c)

case	operation in sign-magnitude	operation in sign-2's complement	required result in sign-2's complement
1.	$(+X) + (+Y)$	$(0+X)+(0+Y)$	$0 + (X+Y)$
2.	$(+X) + (-Y)$	$(0+X)+2^k+(2^k-Y)$	$0+(X-Y)$ if $X \geq Y$ $2^k+2^k-(Y-X)$ if $X<Y$
3.	$(-X) + (+Y)$	$2^k+(2^k-X)+(0+Y)$	$0+(Y-X)$ if $Y \geq X$ $2^k+2^k-(X-Y)$ if $Y<X$
4.	$(-X)+(-Y)$	$(2^k+2^k-X)+(2^k+2^k-Y)$	$2^k+2^k-(X+Y)$

It is necessary to show that the operations in column (b) produce the results listed in column (c).

case 1. column (b) = colum (c)

case 2. If $X \geq Y$ then $(X-Y) \geq 0$ and consists of k bits. operation in column (b) gives: $2^{2k}+(X-Y)$. Discard carry $2^{2k}=2^n$ to get $0+(X-Y)$ as in column (c) If $X<Y$ then $(Y-X)>0$. Operation gives $2^k + 2^k-(Y-X)$ as in colum (c).

case 3. is the same as case 2 with X and Y reversed

case 4. Operation in column (b) gives: $2^{2k}+2^k+2^k-(X-Y)$. Discard carry $2^{2k}=2^n$ to obtain result of (c): $2^k + (2^k-X-Y)$

10-5

B_s	BR		
A_s	AC		

combinational circuit → V overflow
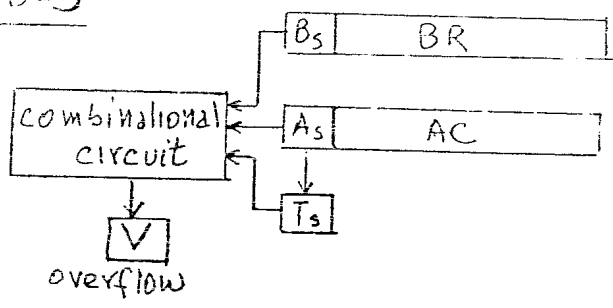
Transfer Augend sign into T_s. Then add : $AC \leftarrow AC + BR$
A_s will have sign of sum.

Boolean function for circuit:

$$V = T_s' B_s' A_s + T_s B_s A_s'$$

Truth Table for combin. circuit

T_s	B_s	A_s	V	
0	0	0	0	
0	0	1	1	change of sign
0	1	0	0	quantities
0	1	1	0	subtracted
1	0	0	0	
1	0	1	0	
1	1	0	1	change of sign
1	1	1	0	

10-6 (a)

```
 -9    1 0110
 -6    1 1001
-15   0/1111
```
F=1 E=0 ← Carries

$E \oplus F = 1$ but there should be no overflow since result is -15

Add end around carry F as needed in signed-1's complement addition :

```
 0 1111
    + 1
 1 0000  = -15
```

(b) The procedure $V \leftarrow E \oplus F$ is valid for 1's complement numbers provided we check the result $0 \underbrace{1111...11}_{A}$ when A_s

$V = 1$.   Algorithm → Add


```
Add
┌──────────────┐
│ AC ← AC + BR │
│ V ← E ⊕ F    │
└──────────────┘
```
=1 ◇ V =0 → ◇ F =0
is $A_s=0$ yes   =1
and A = 11111...11.
no → (overflow)
AC ← AC+1
(no overflow V←0)
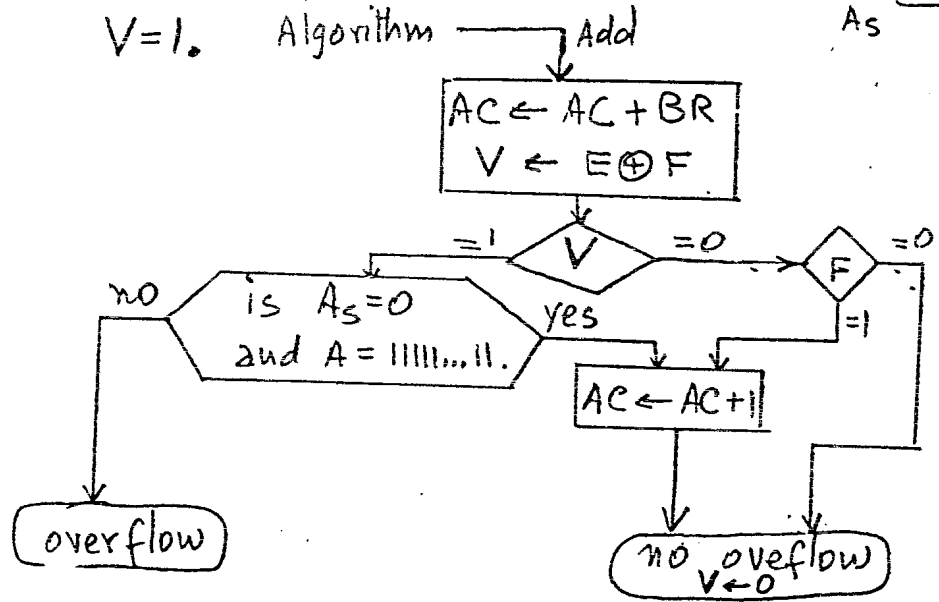
10-7

Add algorithm flowchart is shown above (Prob. 10-6b)

10-8 Maximum value of numbers is $r^n - 1$. It is necessary to show that maximuumum product is less than or equal to $r^{2n} - 1$. Maximum product is:

$$(r^n - 1)(r^n - 1) = r^{2n} - 2r^n + 1 \leq r^{2n} - 1$$

which gives: $2 \leq 2r^n$ or $1 \leq r^n$

This is always true since $r \geq 2$ and $n \geq 1$

10-9 Multiplicand $B = 11111 = (31)_{10}$ $\qquad$ $31 \times 21 = 651$

	E	A	Q	SC	
Multiplier in Q	0	00000	10101	101	$Q = (21)_{10}$
$Q_n = 1$, add B		11111			
	0	11111			
shr EAQ	0	01111	11010	100	
$Q_n = 0$, shr EAQ	0	00111	11101	011	
$Q_n = 1$, add B		11111			
	1	00110			
shr EAQ	0	10011	01110	010	
$Q_n = 0$, shr EAQ	0	01001	10111	001	
$Q_n = 1$, add B		11111			
	1	01000			
shr EAQ		10100	01011,	000	

$$\underbrace{1010001011}_{(651)_{10}}$$

10-10(a) $\dfrac{10100011}{1011} = 1110 + \dfrac{1001}{1011}$ $\qquad\qquad$ $\dfrac{163}{11} = 14 + \dfrac{9}{11}$

$B = 1011 \qquad \overline{B} + 1 = 0101 \qquad DVF = 0$

	E	A	Q	SC
Dividend in AQ	0	1010	0011	100
shl EAQ	1	0100	0110	
add $\overline{B}+1$, suppress carry		0101		
		0101		
E=1, set Q_n to 1	1	1001	0111	011
shl EAQ	1	0010	1110	
add $\overline{B}+1$, suppress carry		0101		
		0101		
E=1, set Q_n to 1	1	0111	1111	010
shl EAQ	0	1111	1110	
add $\overline{B}+1$, carry to E		0101		
E=1, set Q_n to 1	1	0100	1111	001
shl EAQ	0	1001	1110	
add $\overline{B}+1$, carry to E		0101		
E=0, leave $Q_n = 0$	0	1110	1110	
add B		1011		
restore remainder	1	$\underbrace{1001}_{\text{remainder}}$	$\underbrace{1110}_{\text{quotient}}$	000

67

10-10(b)

$\dfrac{1111}{0011} = 0101 \qquad B = 0011 \qquad \bar{B}+1 = 1101$

	E	A	Q	SC
Dividend in Q, $A=0$		0000	1111	100
shl EAQ	0	0001	1110	
add $\bar{B}+1$		1101		
$E=0$, leave $Q_n=0$	0	1110	1110	
add B		0011		
restore partial remainder	1	0001		011
shl EAQ	0	0011	1100	
add $\bar{B}+1$		1101		
$E=1$, set Q_n to 1	1	0000	1101	010
shl EAQ	0	0001	1010	
add $\bar{B}+1$		1101		
$E=0$, leave $Q_n=0$	0	1110	1010	
add B		0011		
restore partial remainder	1	0001		001
shl EAQ	0	0011	0100	
add $\bar{B}+1$		1101		
$E=1$, set Q_n to 1	1	0000	0101	000
		remainder	quotient	

10-11

$A+\bar{B}+1$ performs: $A + 2^n - B = 2^n + A - B$

adding B: $(2^n + A - B) + B = 2^n + A$

remove end-carry 2^n to obtain A.

10-12

To correspond with correct result. In general:

$$\frac{A}{B} = Q + \frac{R}{B}$$

Where A is dividend, Q the quotient and R the remainder. Four possible signs for A and B:

$\dfrac{+52}{+5} = +10 + \dfrac{+2}{+5} = +10.4 \qquad \dfrac{-52}{+5} = -10 + \dfrac{-2}{+5} = -10.4$

$\dfrac{+52}{-5} = -10 + \dfrac{+2}{-5} = -10.4 \qquad \dfrac{-52}{-5} = +10 + \dfrac{-2}{-5} = +10.4$

The sign of the remainder (2) must be same as sign of dividend (52).

10-13

Add one more stage to Fig. 10-10 with 4 AND gates and a 4-bit adder.

(a) $(+15) \times (+13) = +195 = (0\ 0110\ 00011)_2$

$BR = 0\ 1111\ (+15)$; $\overline{BR}+1 = 1\ 0001\ (-15)$; $QR = 0\ 1101\ (+13)$

Q_n Q_{n+1}		AC	QR	Q_{n+1}	SC
	Initial	00000	01101	0	101
1 0	Subtract BR	10001			
		10001			
	ashr ——	11000	10110	1	100
0 1	Add BR	01111			
		00111			
	ashr ——	00011	11011	0	011
1 0	Subtract BR	10001			
		10100			
	ashr ——	11010	01101	1	010
1 1	ashr ——	11101	00110	1	001
0 1	Add BR	01111			
		01100			
	ashr ——	00110	00011	0	000

$+\ 195$

(b) $(+15) \times (-13) = -195 = (1100\ 11101)_{\text{2's comp.}}$

$BR = 0\ 1111\ (+15)$; $\overline{BR}+1 = 1\ 0001\ (-15)$; $QR = 1\ 0011\ (-13)$

Q_n Q_{n+1}		AC	QR	Q_{n+1}	SC
	Initial	00000	10011	0	101
1 0	Subtract BR	10001			
		10001			
	ashr ——	11000	11001	1	100
1 1	ashr ——	11100	01100	1	011
0 1	Add BR	01111			
		01011			
	ashr ——	00101	10110	0	010
0 0	ashr ——	00010	11011	0	001
1 0	Subtract BR	10001			
		10011			
	ashr ——	11001	11101	1	000

$-\ 195$

Dividend in AQ
Divisor in B

$Q_s \leftarrow A_s \oplus B_s$
$SC \leftarrow n-1$

$EA \leftarrow A + \bar{B} + 1$

E

$=1$

$=0$

$DVF \leftarrow 0$

$EA \leftarrow A+B$
$DVF \leftarrow 1$

END
(Divide overflow)

shl EAQ
$Q_n \leftarrow E$

Q_n

$=0$

$=1$

$EA \leftarrow A+B$

$EA \leftarrow A+\bar{B}+1$

$SC \leftarrow SC-1$

SC

$\neq 0$

$=0$

shl Q, $Q_n \leftarrow E$

Q_n

$=0$

$=1$

$A \leftarrow A+B$

END
Quotient is in Q
Remainder is in A
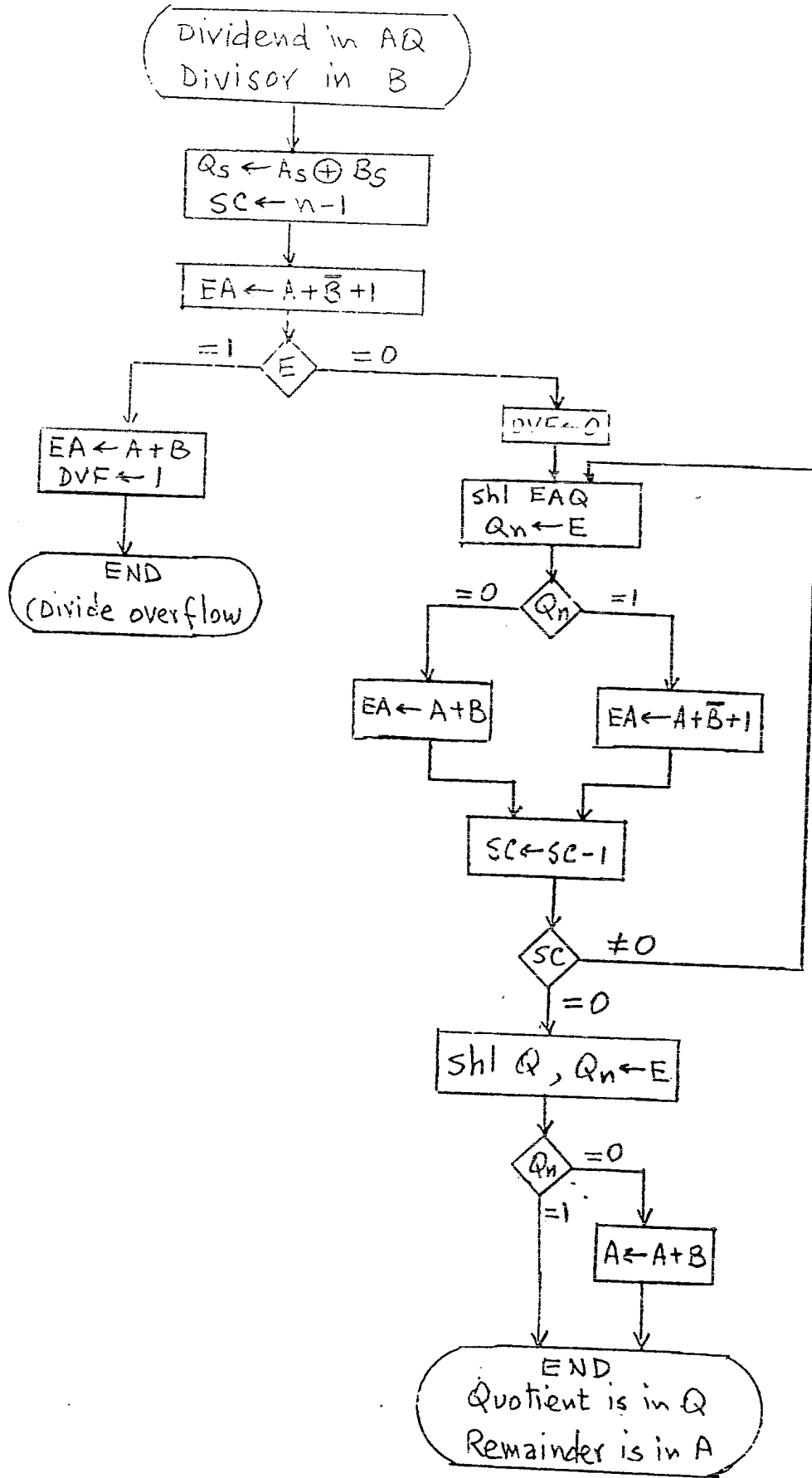
10-16 The algorithm for square-root is similar to division with the radicand being equivalent to the dividend and a "test value" being equivalent to the divisor.

Let A be the radicand, Q the square-root, and R the remainder such that $Q^2 + R = A$ or:

$$\sqrt{A} = Q \text{ and a remainder}$$

General coments:

1. For k bits in A (k even), Q will have $k/2$ bits:

$$Q = q_1 q_2 q_3 \cdots q_{k/2}$$

2. The first test value is 01
   The second test value is $0 q_1 01$
   The third test value is $00 q_1 q_2 01$
   The fourth test value is $000 q_1 q_2 q_3 01$ etc.

3. Mark the bits of A in groups of two starting from left.

4. The procedure is similar to the division restoring method as shown in the following example:

```
         q₁  q₂  q₃  q₄
         1   1   0   1      = Q = 13
       √10  10  10  01      = A = 169
         01                 subtract first test value 01
         ‾‾
         01                 Answer positive; let q₁ = 1
    ...  01 10              bring down next pair
         01 01              subtract second test value 0q₁01
         ‾‾‾‾‾
         00 01             answer positive; let q₂ = 1
         00 01 10          bring down next pair
         00 1 10 1         subtract third test value 00q₁q₂01
         ‾‾‾‾‾‾‾
          negative         answer negative; let q₃ = 0
         00 0 1 10         restore partial remainder
         00 0 1 10 01      bring down next pair
         00 0 1 10 01      subtract fourth test value 000q₁q₂q₃01
         ‾‾‾‾‾‾‾‾‾‾
Remainder = 00000           answer positive (zero); let q₄ = 1
```

71

10-17 (a) e = exponent $e+64$ = biased exponent

e	$e+64$	biased exponent
-64	$-64+64=0$	0 000 000
-63	$-63+64=1$	0 000 001
-62	$-62+64=2$	0 000 010
-1	$-1+64=63$	0 111 111
0	$0+64=64$	1 000 000
$+1$	$1+64=65$	1 000 001
$+62$	$62+64=126$	1 111 110
$+63$	$63+64=127$	1 111 111

(b) The biased exponent follows the same algorithm as a magnitude comparator. See Sec. 9-2

(c) $(e_1+64)+(e_2+64) = (e_1+e_2+64)+64$

  subtract 64 to obtain biased exponent sum

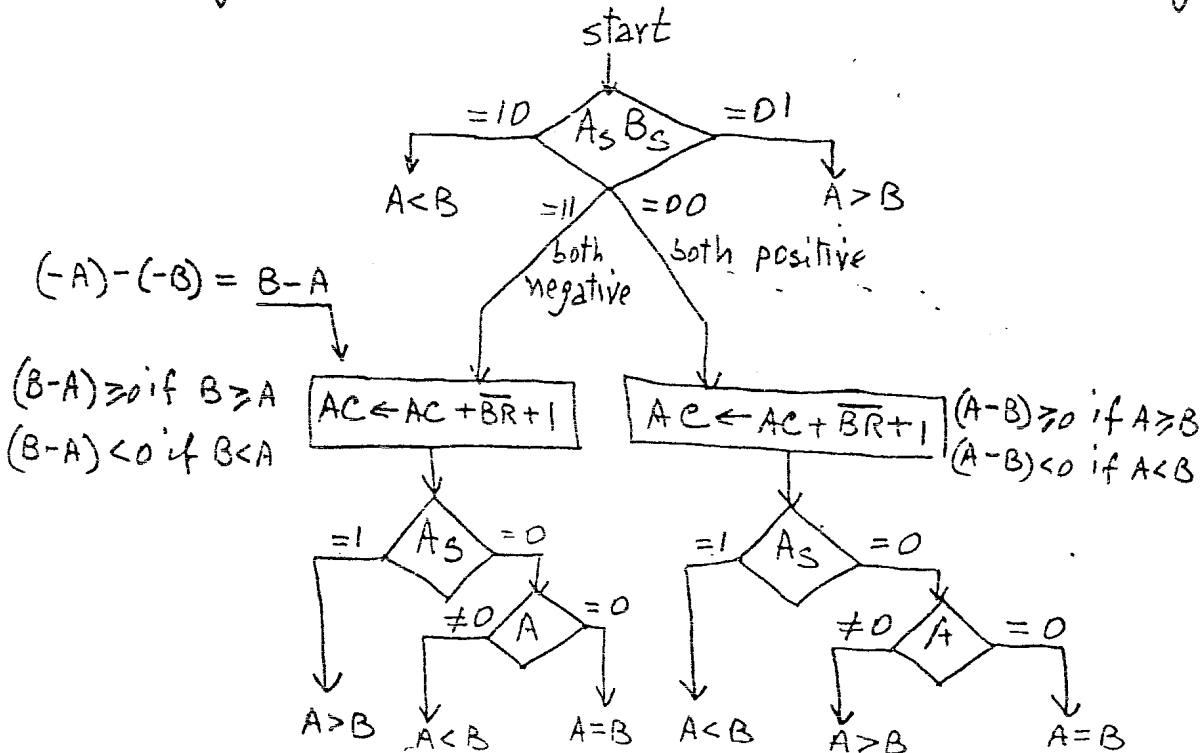(d) $(e_1+64)-(e_2-64) = e_1+e_2$

  add 64 to obtain biased exponend difference,

10-18

(a) $AC = A_s\ A_1\ A_2\ A_3\ \dots\ A_n$
   $BS = B_s\ B_1\ B_2\ B_3\ \dots\ B_n$

If signs are unlike — the one with a 0 (plus) is larger.
If signs are alike — both numbers are either positive or negative

start

$A_s B_s$

$=10 \rightarrow A<B$

$=01 \rightarrow A>B$

$=11$ both negative

$=00$ both positive

$(-A)-(-B) = B-A$

$(B-A)\geqslant 0$ if $B\geqslant A$
$(B-A)<0$ if $B<A$

$AC \leftarrow AC + \overline{BR} + 1$

$AC \leftarrow AC + \overline{BR} + 1$

$(A-B)\geqslant 0$ if $A\geqslant B$
$(A-B)<0$ if $A<B$

A_s
$=1$ $=0$

$\neq 0$ A $=0$

$A>B$ $A<B$ $A=B$

A_s
$=1$ $=0$

$\neq 0$ A $=0$

$A<B$ $A>B$ $A=B$

72

10-18 (b)

$$
\begin{array}{c|ccccccc}
 & A_S & A_1 & A_2 & \ldots & & & A_n \\
\hline
+2 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
+1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
-2 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
-3 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
\end{array}
$$

10-19

$$
\begin{array}{c}
\overbrace{A_S \ A_1 \ A_2 \ A_3 \ \ldots \ A_n} \\
\underbrace{B_S \ B_1 \ B_2 \ B_3 \ \ldots \ B_n}
\end{array}
\quad A \quad B
$$

(a)

(b)
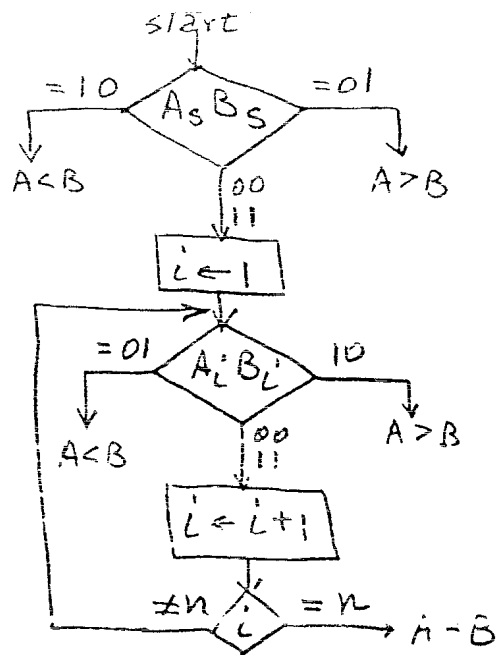
10-20

Fig 10-8
mantissa allignment

same as Fig. 10-8

10-21 Let "e" be a flip-flop that holds end-carry after exponent addition.

start

check for zeros

$B_S \leftarrow \bar{B_S}$ (sub) — OP — (add)

$ea \leftarrow a + \bar{b} + 1$

e: $=0$ ($a < b$) / $=1$ ($a \geq b$)

a: $\neq 0$ / $= 0$

restore: $a \leftarrow a + b$

swap:
$BR \leftarrow AC$
$AC \leftarrow BR$

$a \leftarrow a + \bar{b} + 1$ ($a < b$)

a contains 2's complement of difference
b contains larger exponent

shr A
$a \leftarrow a + 1$

a: $\neq 0$ / $= 0$

$a \leftarrow b$

add mantissas and normalize
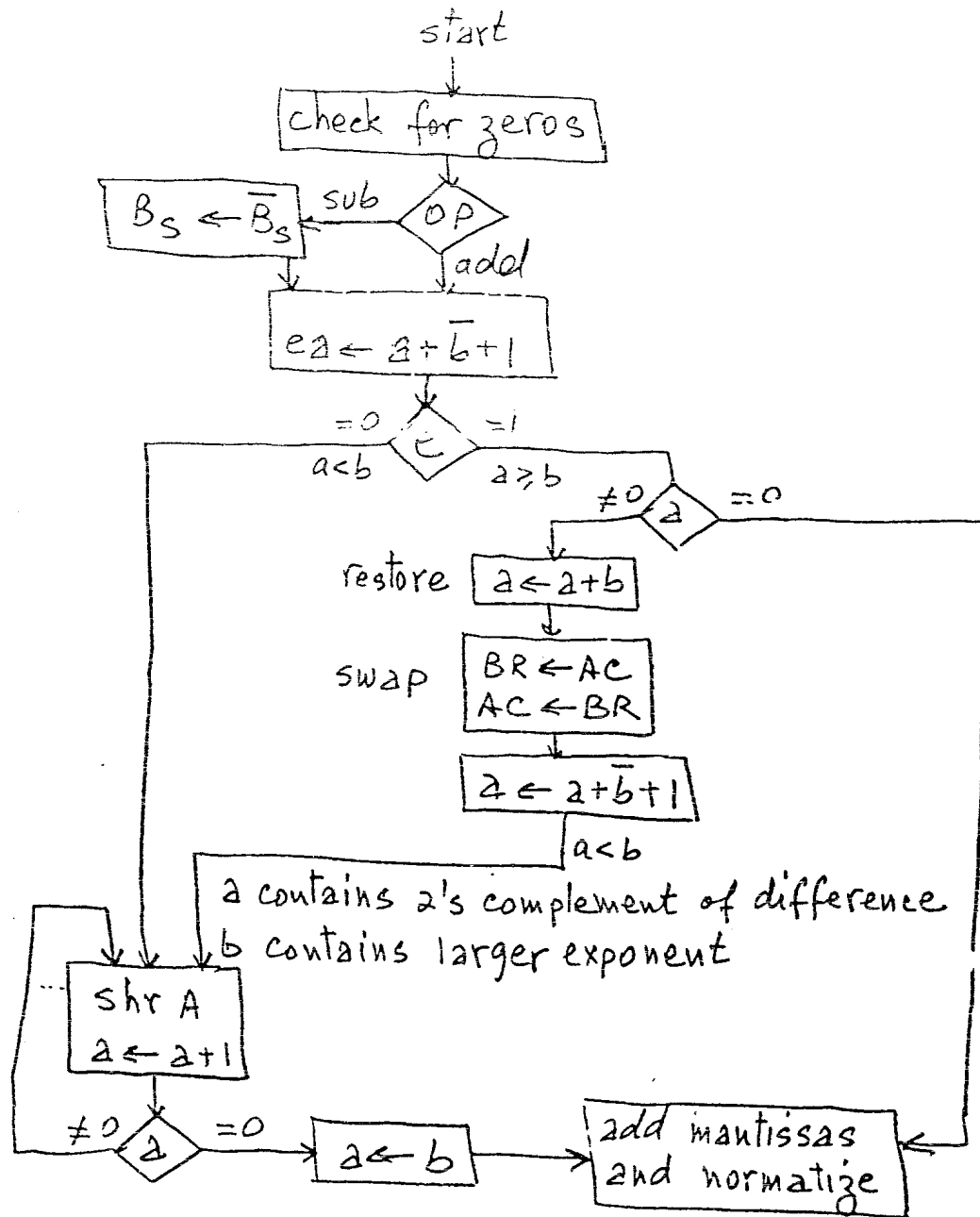
10-22

When 2 numbers of n bits each are multiplied, the product is no more than $2n$ bits long — see Prob. 9-7.

10-23 $\dfrac{\text{dividend}}{\text{divisor}}$ $A = 0.1xxxx$ $B = 0.1xxxx$ where $x = 0, 1$

(a) If $A < B$ then after shift we have $A = 1.xxxx$ and 1st quotient bit is a 1.

(b) if $A \geq B$, dividend alignment results in $A = 0.01xxxx$ then after the left shift $A \geq B$ and first quotient bit $= 1$.

10-24

$$\frac{\text{dividend}}{\text{divisor}} = \frac{.1\,\overbrace{xxxx}^{n-1\ bits} * 2^{e_1}}{.1yyyy * 2^{e_2}} = .1\partial\partial\partial\partial * 2^{e_1-e_2} + \frac{.00000\,\overbrace{rrrrr}^{remainder} * 2^{e_1}}{.1yyyy * 2^{e_2}}$$
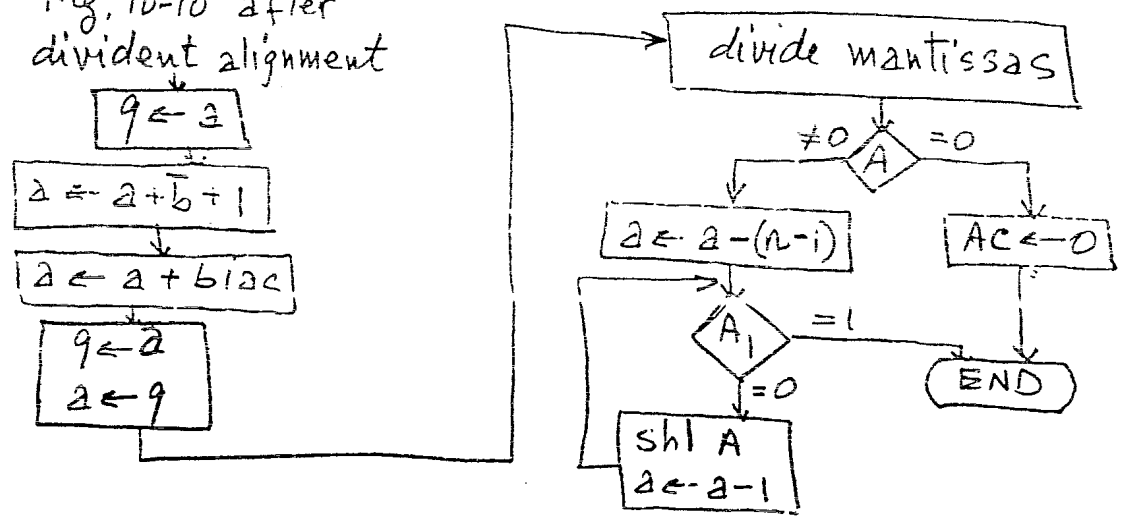
Remainder bits $rrrrr$ have a binary-point $(n-1)$ bits to the left.

Fig. 10-10 after dividend alignment

10-25

(a) When the exponents are added or incremented

(b) When the exponents are subtracted or decremented

(c) Check end-carry after addition and carry after increment or decrement.

10-26

Assume integer mantissa of $n-1=5$ bits (excluding sign)

(a) Product:

$$\begin{array}{cc} A & Q \\ xxxxx & xxxxx. \end{array} * 2^{\partial}$$

binary-point for integer

Product in AC : $xxxxx. * 2^{\partial+5}$

(b) Single precision normalized dividend : $xxxxx. * 2^{\partial}$

Dividend in AQ :

$$\begin{array}{cc} A & Q \\ xxxxx & 00000. \end{array} * 2^{\partial-5}$$

10-27

Neglect Be and Ae from Fig. 10-14. Apply carry directly to E.

Bs		10^3	10^2	10^1	10^0		B

| E | ← | BCD arith. Unit | | | | | |

| As | | 10^3 | 10^2 | 10^1 | 10^0 | | A |

75

10-28

$$\begin{array}{r} 673 \\ -\ 356 \\ \hline 317 \end{array}$$

10's comp. of $356 = 644$

$$\begin{array}{r} 673 \\ +\ 644 \\ \hline \text{carry} \to 1)\ 317 \end{array}$$
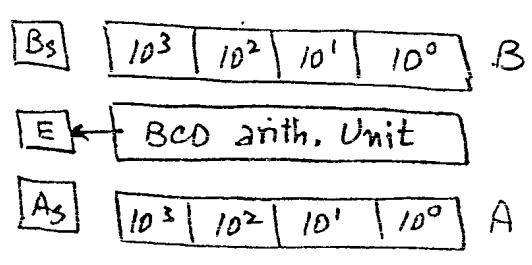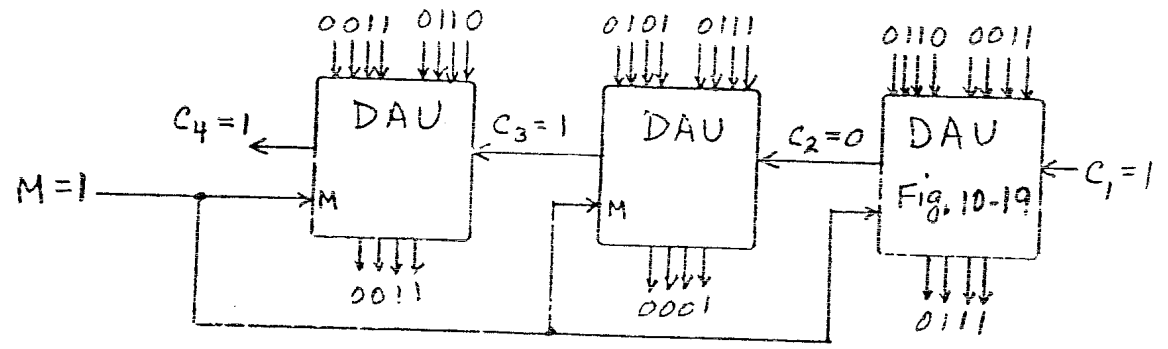
10-29
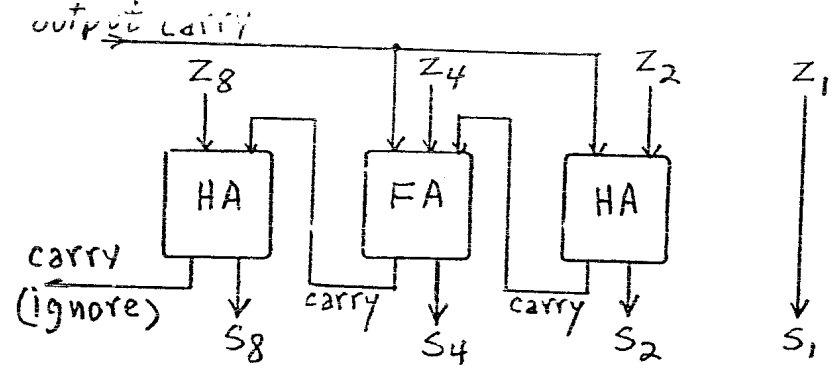
10-30

	inputs	outputs	
	$B_8\ B_4\ B_2\ B_1$	$X_8\ X_4\ X_2\ X_1$	
0	0 0 0 0	1 0 0 1	9
1	0 0 0 1	1 0 0 0	8
2	0 0 1 0	0 1 1 1	7
3	0 0 1 1	0 1 1 0	6
4	0 1 0 0	0 1 0 1	5
5	0 1 0 1	0 1 0 0	4
6	0 1 1 0	0 0 1 1	3
7	0 1 1 1	0 0 1 0	2
8	1 0 0 0	0 0 0 1	1
9	1 0 0 1	0 0 0 0	0

$d \cdot (B_8\ B_4\ B_2\ B_1) = \Sigma(10, 11, 12, 13, 14, 15)$ are don't-care conditions

$X_8 = B_8'\ B_4'\ B_2'$

$X_4 = B_4\ B_2' + B_4'\ B_2$

$X_2 = B_2$

$X_1 = B_1'$

dec	Z uncorrected	Y corrected		dec	Z uncorrected	Y corrected
0	0110	0011		10	1 0000	1 0011
1	0111	0100		11	1 0001	1 0100
2	1000	0101		12	1 0010	1 0101
3	1001	0110		13	1 0011	1 0110
4	1010	0111		14	1 0100	1 0111
5	1011	1000		15	1 0101	1 1000
6	1100	1001		16	1 0110	1 1001
7	1101	1010		17	1 0111	1 1010
8	1110	1011		18	1 1000	1 1011
9	1111	1100		19	1 1001	1 1100

No output carry

$$Y = Z - 3 = Z + 13 - 16$$

ignore carry ↑

Uncorrected = output carry
carry

$$Y = Z + 3$$
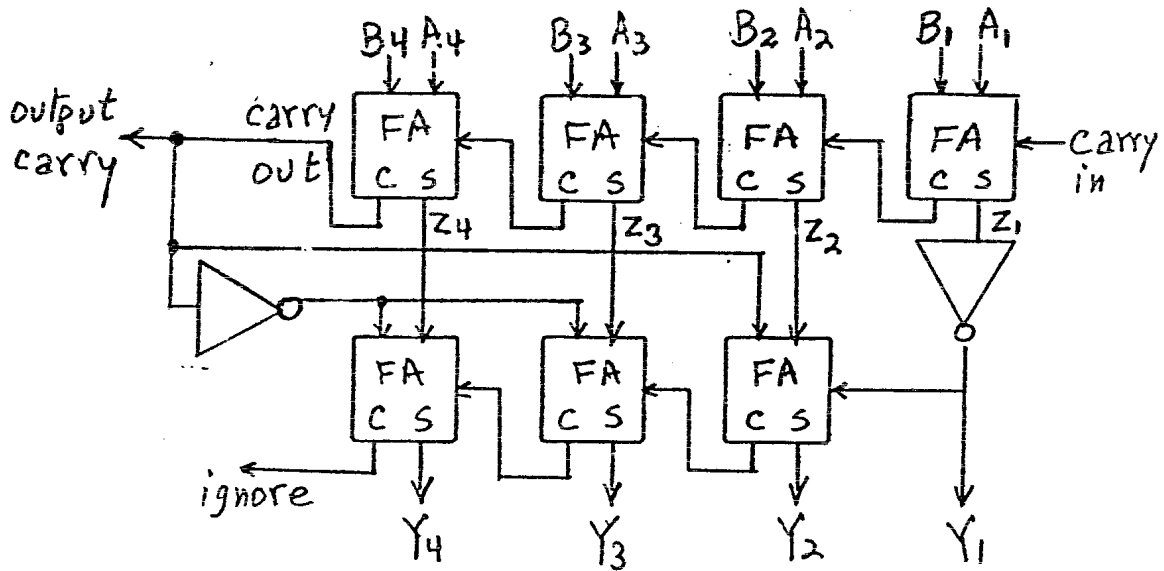
10-32 The excess-3 code is self-complementing code. Therefore, to get 9's complement we need to complement each bit.

$M = 0$ for $x = B$

$M = 1$ for $x = 9$'s comp. of B

M	B_i	$x_i = B_i \oplus M$
0	0	0 $\big\}$ $x_i = B_i$
0	1	1
1	0	1 $\big\}$ $x_i = B_i'$
1	1	0

	10^3	10^2	10^1	10^0	BR
B_s					

FA ← BCD Arithm. Unit

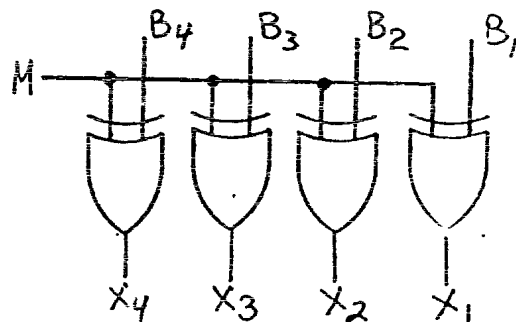|	10^3	10^2	10^1	10^0	AC
A_s					

V
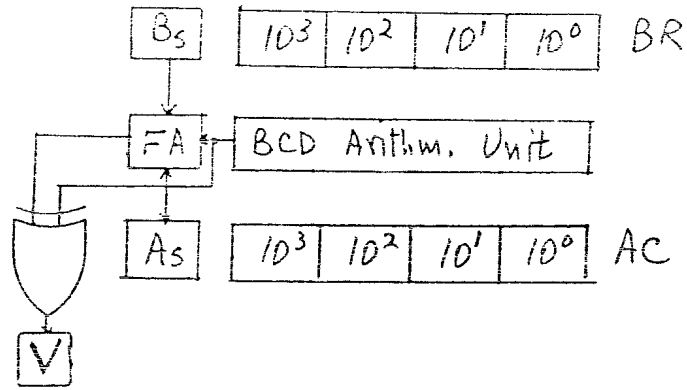
Algorithm is similar to flow chart of Fig. 10-2

10-34 (a) $B = 410$

	A_e	A	Q	SC
initial — — — — —	0	000	152	3
$Q_L \neq 0$ — — — — —	0	470	151	
$Q_L \neq 0$ — — — — —	0	940	150	
$Q_L = 0$, dshr — — —	0	094	015	2
	0	564	014	
	1	034	013	
$Q_L \neq 0$ — — —	1	504	012	
	1	974	011	
	2	444	010	
$Q_L = 0$, dshr — —	0	244	401	1
$Q_L \neq 0$ — — — —	0	714	400	
$Q_L = 0$, dshr — —	0	071	440	0

Product

(b)
```
      999
    × 199
    ───────
     8991   — first partial product   $A_e = 8$
  + 89910
  ─────────
    98901   — second partial product  $A_e = 9$
  +999900
  ─────────
   198801   — final product            $A_e = 1$
```

10-35

$$\frac{1680}{32} = 52 + \frac{16}{32} \qquad B = 032$$

$$\bar{B}+1 = 968 \text{ (10's comp.)}$$

	E	Ae	A	Q	SC
initial	0	0	16	80	2
dshl		1	68	00	
add $\bar{B}+1$		9	68		
E=1	1	1	36	01	
add $\bar{B}+1$		9	68		
	1	1	04	02	
		9	68		
	1	0	72	03	
		9	68		
	1	0	40	04	
		9	68		
	1	0	08	05	
		9	68		
E=0	0	9	76		
add B		0	32		
restore remainder	1	0	08	05	1

(continued here)

	E	Ae	A	Q	SC
dshl	—	0	80	50	1
add $\bar{B}+1$	—	9	68		
E=1	1	0	48	51	
		9	68		
	1	0	16	52	
		9	68		
E=0	0	9	84		
add B	—	0	32		
	1	0	16	52	0

remainder   quotient

10-36

(a) At the termination of multiplication we shift right the content of A to get zero in Ae.

(b) At the termination of division, B is added to the negative difference. The negative difference is in 10's complement so Ae =9. Adding Be =0 to Ae =9 produces a carry and makes Ae =0.

10-37

Change the symbols as defined in Table 10-1 and use same algorithms as in Sec. 10-4 but with multiplication and division of mantissas as in Sec. 10-5.

11-1

$$\underset{12 = \underbrace{000011}_{\text{To } CS}\,\underset{\underset{RSI}{\uparrow}\,\underset{RSO}{\uparrow}}{00}}{\overset{\overbrace{A_7 - A_2}\,A_1 A_0}{}}$$

12 = 000011 00

13 = 000011 01

14 = 000011 10

15 = 000011 11

$CS = A_2 A_3 A_4' A_5' A_6' A_7'$

$RSI = A_1$

$RSO = A_0$

11-2

Interface	Port A	Port B	Control Reg	Status Reg
#1	1000 0000	1000 0001	1000 0010	1000 0011
2	0100 0000	0100 0001	0100 0010	0100 0011
3	0010 0000	0010 0001	0010 0010	0010 0011
4	0001 0000	0001 0001	0001 0010	0001 0011
5	0000 1000	0000 1001	0000 1010	0000 1011
6	0000 0100	0000 0101	0000 0110	0000 0111

11-3

Character printer ; Line printer ; Laser printer ;
Digital plotter ; Graphic display ; voice output ;
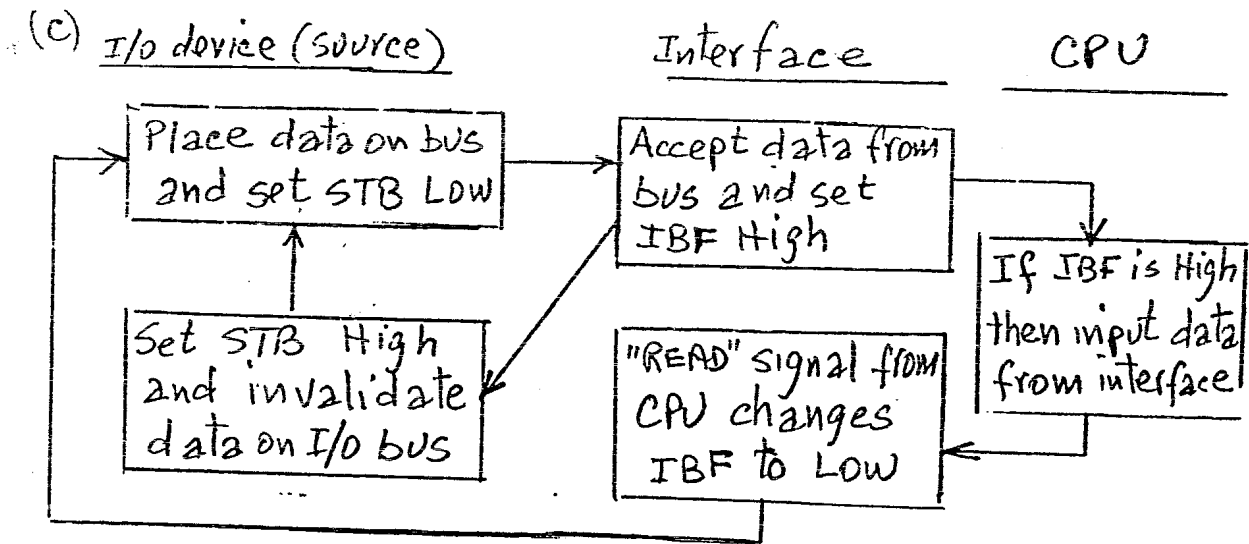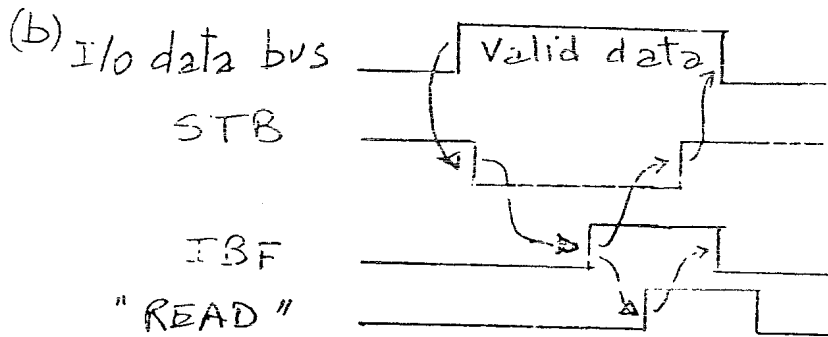Digital to analog converter ; Instrument indicator.

11-5

See text discussion in Sec. 11-2.

11-6

(a) Status command - Checks status of flag bit.

(b) Control command - Moves magnetic head in disk.

(c) Status command - cheks if device power is on,

(d) Control command - Moves paper position,

(e) Data input command - Reads value of a register

11-7

(a)

CPU data bus / "READ" I/O — Interface Unit — I/O data bus / STB / IBF — I/O device

(b)

I/o data bus — Valid data

STB

IBF

"READ"

(c) I/o device (source)   Interface   CPU

- Place data on bus and set STB LOW
- Accept data from bus and set IBF High
- If IBF is High then input data from interface
- Set STB High and invalidate data on I/o bus
- "READ" signal from CPU changes IBF to LOW
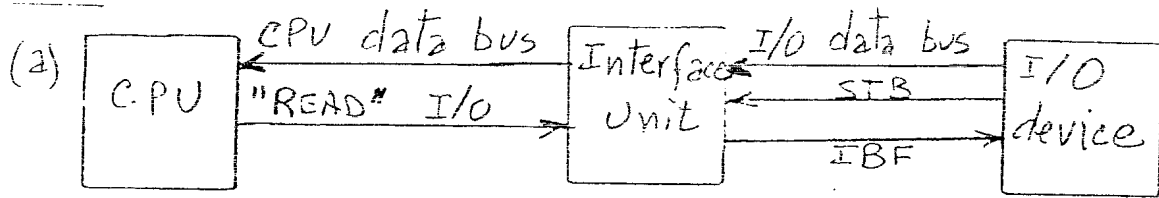
11-8

$$20 \, MHz = 20 \times 10^6 \, Hz \qquad T = \frac{10^{-6}}{20} = 50 \quad n\,sec.$$

← 50 nsec →

clock

Address

READ STROBE ← 40 nsec → Data from memory

WRITE STROBE

DATA for memory

81
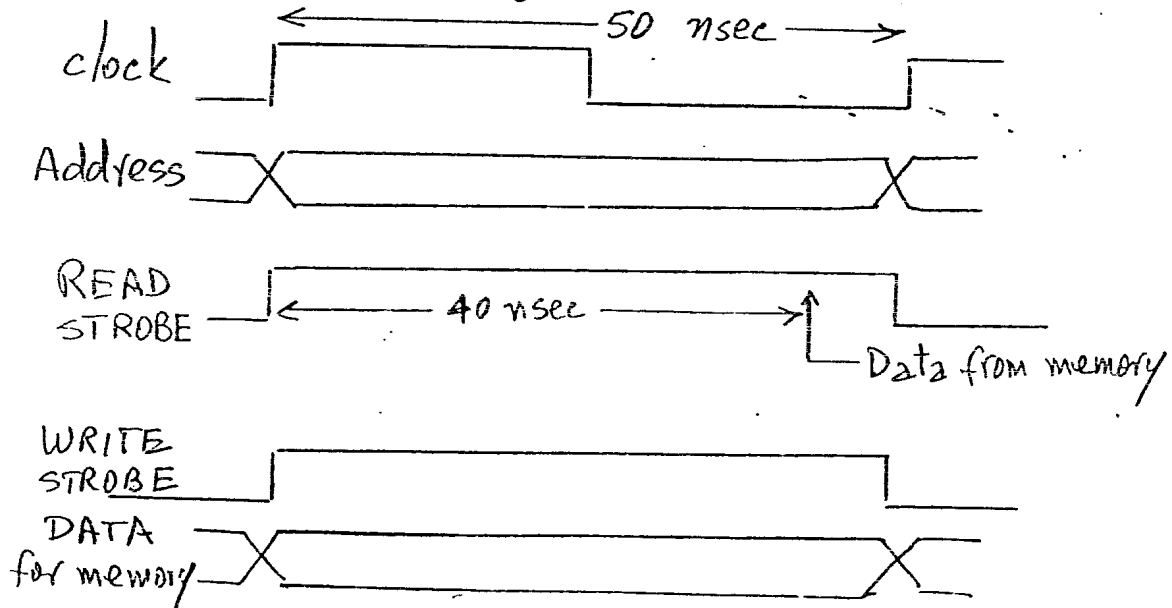
11-9
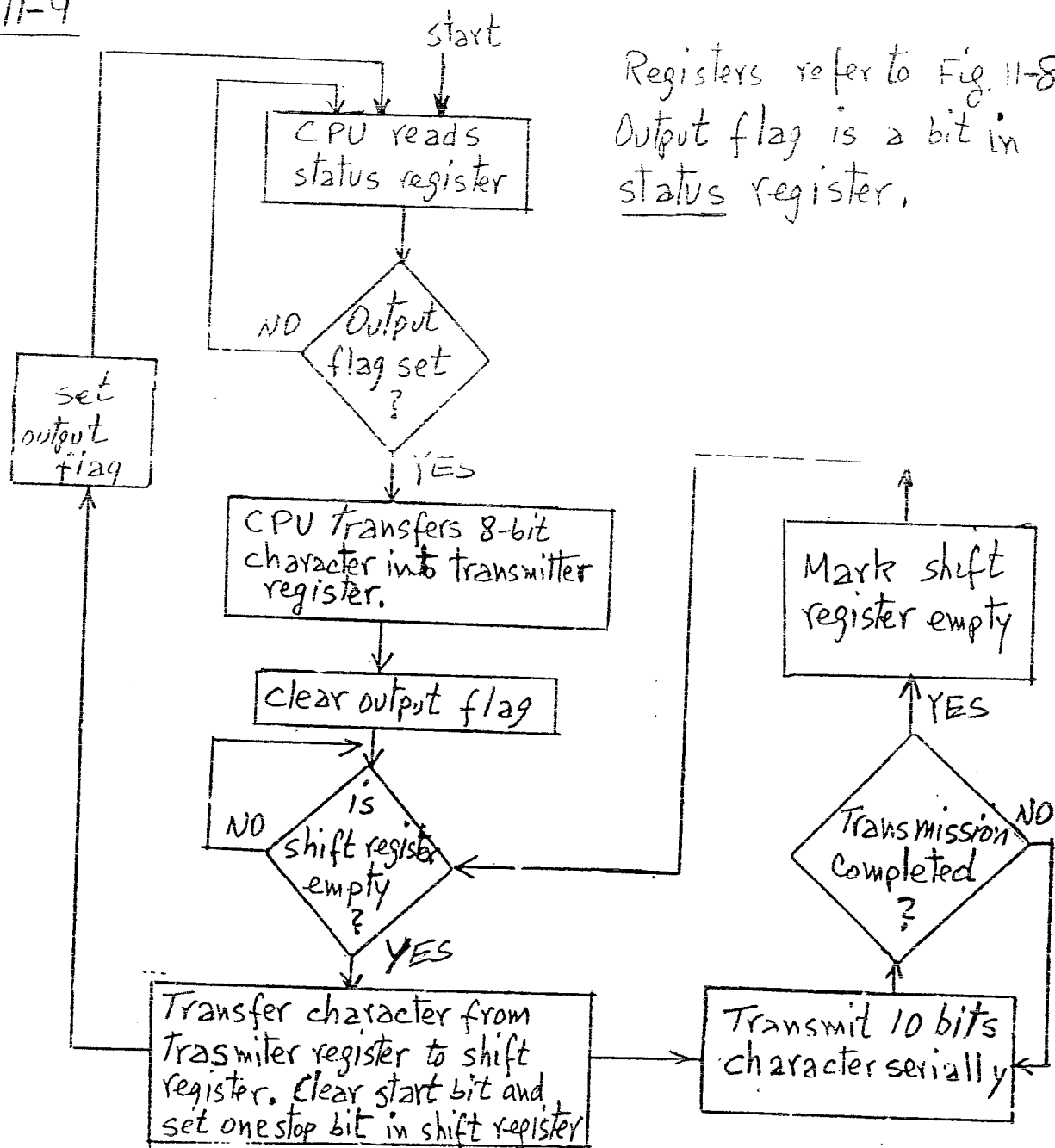
Registers refer to Fig. 11-8.
Output flag is a bit in status register.

Flowchart boxes:
- start
- CPU reads status register
- Output flag set ? — NO → (loop back) / YES ↓
- set output flag
- CPU transfers 8-bit character into transmitter register.
- clear output flag
- is shift register empty ? — NO (loop back) / YES ↓
- Transfer character from Trasmiter register to shift register. Clear start bit and set one stop bit in shift register
- Transmit 10 bits character serially
- Transmission Completed ? — NO / YES →
- Mark shift register empty

11-10

1. Output flag to indicate when transmitter register is empty.
2. Input flag to indicate when receiver register is full.
3. Enable interrupt if any flag is set.
4. Parity error ; (5) Framing error ; (6) Overrun error.

11-11

10 bits : Start bit + 7 ASCII + parity + stop bit.
From Table 11-1 ASCII W = 1010111
with even parity = 11010111
with start and stop bits = 1110101110

82

11-12

(a) $\dfrac{1200}{8} = 150$ characters per second (cps)

(b) $\dfrac{1200}{11} = 109$ cps

(c) $\dfrac{1200}{10} = 120$ cps

11-13

(a) $\dfrac{k \text{ bytes}}{(m-n) \text{ bytes/sec}} = \dfrac{k}{m-n}$ sec.
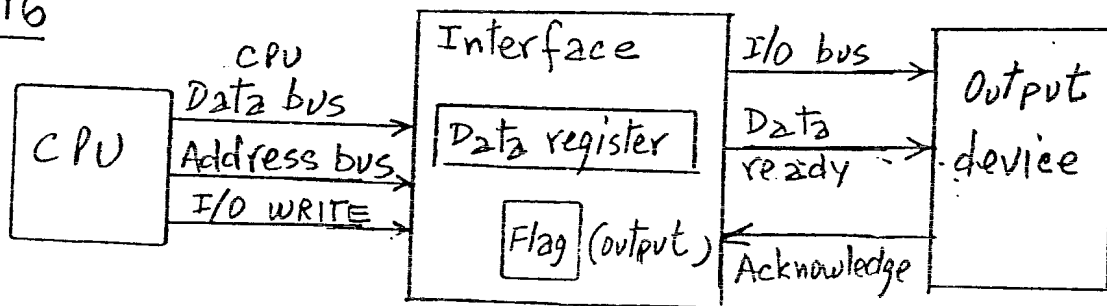
(b) $\dfrac{k}{n-m}$ sec.   (c) No need for FIFO

11-14

Initial	F = 0011	Output ← R4
After delete =1	F = 0010	
After delete =0	F = 0001	R4 ← R3
After insert =1	F = 1001	R1 ← Input
(Insert goes to 0)	F = 0101	R2 ← R1
	F = 0011	R3 ← R2

11-15

	Input ready	output ready	$F_1 - F_4$
(a) Empty buffer	1	0	0000
(b) Full buffer	0	1	1111
(c) Two items	1	1	0011

11-16

Flag = 0 if data register full (After CPU writes data)

Flag = 1 if data register empty (After the transfer to device)

When flag goes to 0, enable "Data ready" and place data on I/O bus. When "Acknowledge" is enabled, set the flag to 1 and disable "ready" handshake line.

11-17 CPU Program flow chart:

```
                    start
                      ↓              ↓
        ┌─────────────────────────┐
        │ Read data from          │
        │ memory into a CPU       │
        │ Register                │
        └─────────────────────────┘
                      ↓
        ┌─────────────────────────┐
        │ Read status register    │
        │ from interface and      │
        │ check output flag bit   │
        └─────────────────────────┘
                      ↓
              =0  ◇ flag
                      │ =1
                      ↓
        ┌─────────────────────────┐
        │ Write data into interface│
        │ data register           │
        └─────────────────────────┘
                      │
     yes ⟨ Transferred all data? ⟩ NO
  (Continue)
```
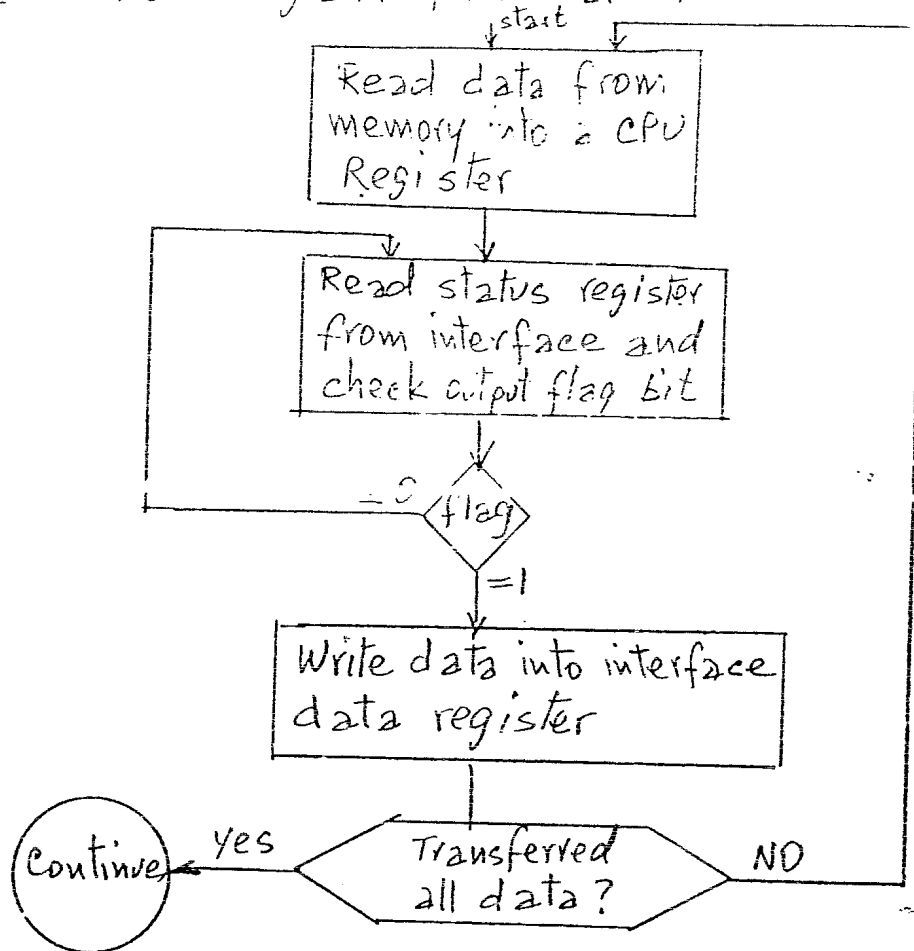
11-18

See text Section 11-4.

11-19

If an interrupt is recognized in the middle of an instruction execution, it is necessary to save all the information from control registers in addition to processor registers. The state of the CPU to be saved is more complex.

11-20

	Device 1	Device 2
(1) Initially, device 2 sends an interrupt request:	$PI=0; PO=0; RF=0$	$PI=0; PO=0; RF=1$
(2) Before CPU responds with acknowledge, device 1 sends interrupt request:	$PI=0; PO=0; RF=1$	$PI=0; PO=0; RF=1$
(3) After CPU sends an acknowledge, device 1 has priority:	$PI=1; PO=0; RF=1$ VAD enable = 1	$PI=0; PO=0; RF=1$ VAD enable = 0

11-22   Table 11-2.

I_0	I_1	I_2	I_3	x y	IST
1	X	X	X	0 0	1
0	1	X	X	0 1	1
0	0	1	X	1 0	1
0	0	0	1	1 1	1
0	0	0	0	X X	0

Map simplification
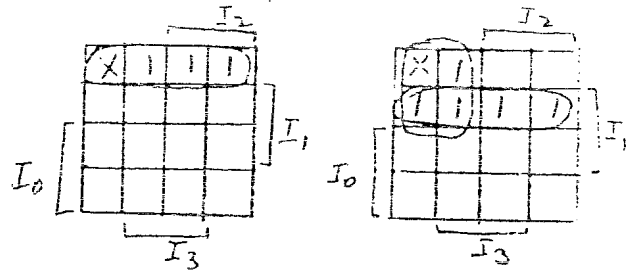
$$x = I_0' I_1'$$

$$y = I_0' I_1 + I_0' I_2'$$

11-23

Same as Fig. 11-14. Needs 8 AND gates and an 8×3 decoder.

11-24

I_0	I_1	I_2	I_3	I_4	I_5	I_6	I_7	$x y z$	IST
1	X	X	X	X	X	X	X	0 0 0	1
0	1	X	X	X	X	X	X	0 0 1	1
0	0	1	X	X	X	X	X	0 1 0	1
0	0	0	1	X	X	X	X	0 1 1	1
0	0	0	0	1	X	X	X	1 0 0	1
0	0	0	0	0	1	X	X	1 0 1	1
0	0	0	0	0	0	1	X	1 1 0	1
0	0	0	0	0	0	0	1	1 1 1	1
0	0	0	0	0	0	0	0	X X X	0

(b)

Binary	hexadeciwal
1010 0000	A0
1010 0100	A4
1010 1000	A8
1010 1100	AC
1011 0000	B0
1011 0100	B4
1011 1000	B8
1011 1100	BC

11-25

$$76 = (0100 1100)_2$$

Replace the six 0's by 010011, xy

11-26

Set the mask bit belonging to the interrupt source so it can interrupt again.

At the beginning of the service routine, check the value of the return address in the stack. If it is an address within the source service program, then the same source has interrupted again while being serviced.

11-27

The service routine checks the flags in sequence to determine which one is set. The first flag that is checked has the highest priority level. The priority level of the other sources corresponds to the order in which the flags are checked.

11-27

When the CPU communicates with the DMA controller, the read and write lines are used as inputs from the CPU to the DMA controller.

When the DMA controller communicates with memory, the read and write lines are used as outputs from the DMA to memory.

11-28

(a) CPU initiates DMA by transferring:

256 to the word count register.

1230 to the DMA address register.

Bits to the control register to specify a write operation.

(b)

1. I/o device sends a "DMA request."
2. DMA sends BR (bus request) to CPU.
3. CPU responds with a BG (bus grant).
4. Contents of DMA address register are placed in address bus.
5. DMA sends "DMA acknowledge" to I/o device and enables the write control line to memory.
6. Data word is placed on data bus by I/o device.
7. Increment DMA address register by 1 and Decrement DMA word count register by 1
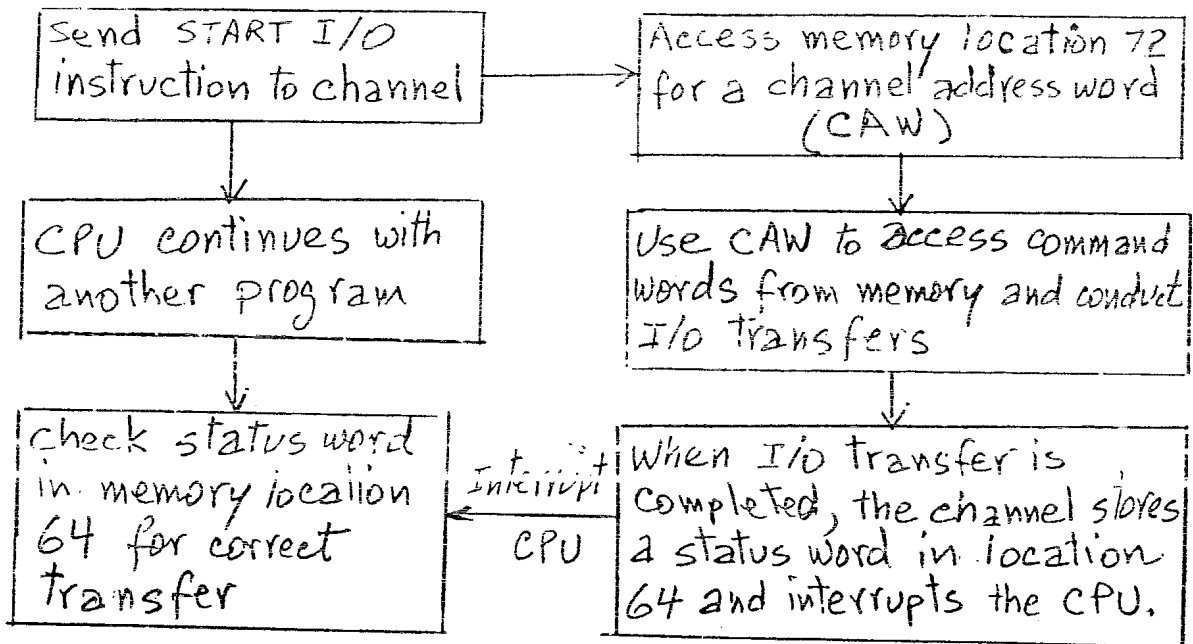8. Repeat steps 4-7 for each data word transferred

11-29

CPU refers to memory on the average once (or more) every 1 μsec. ($1/10^6$). Characters arrive one every $1/2400 = 416.6$ μsec. Two characters of 8 bits each are packed into a 16-bit word every $2 \times 416.6 = 833.3$ μsec. The CPU is slowed down by no more than $(1/833.3) \times 100 = 0.12\%$.

11-30

The CPU can wait to fetch instructions and data from memory without any damage occuring except loss of time. DMA usually transfers data from a device that cannot be stopped since information continues to flow so loss of data may occur.

```
┌─────────────────────┐        ┌──────────────────────────┐
│ Send START I/O      │        │ Access memory location 72│
│ instruction to      │───────▶│ for a channel address    │
│ channel             │        │ word (CAW)               │
└─────────────────────┘        └──────────────────────────┘
           │                              │
           ▼                              ▼
┌─────────────────────┐        ┌──────────────────────────┐
│ CPU continues with  │        │ Use CAW to access command│
│ another program     │        │ words from memory and    │
│                     │        │ conduct I/O transfers    │
└─────────────────────┘        └──────────────────────────┘
           │                              │
           ▼                              ▼
┌─────────────────────┐ Interrupt ┌──────────────────────────┐
│ Check status word   │◀────────│ When I/O transfer is     │
│ in memory location  │   CPU    │ completed, the channel   │
│ 64 for correct      │         │ stores a status word in  │
│ transfer            │         │ location 64 and interrupts│
└─────────────────────┘         │ the CPU.                 │
                                 └──────────────────────────┘
```

11-32   There are 26 letters and 10 numerals.

$$26 \times 26 + 26 \times 10 = 936 \text{ possible addresses.}$$

11-33

The processor transmits the address of the terminal followed by ENQ (enquiry) code 0000 0101. The terminal responds with either ACK (acknowledge) or NAK (negative acknowledge) or the terminal does not respond during a timeout period. If the processor receives an ACK, it sends a block of text.

11-34

```
DLE   STX   DLE   DLE   ETX   DLE   DLE   ETX   DLE   ETX
 ↑           ↑                 ↑                 ↑
delete     delete            delete            delete

      STX          DLE   ETX         DLE   ETX         ETX
```

32-bit text = 0001 0000 1000 0011 0001 0000 1000 0011

11-35

32 bits between two flags; 48 bits including the flags.

11-36

Information to be sent (1023):   0 11111111 111
After zero insertion, information transmitted: 0 111110 111110
                                                     delete
Information received after 0's deletion: 0 11111 11111

12-1 (a) $\frac{2048}{128} = 16$ chips

(b) $2048 = 2^{11}$ 11 lines to address 2048 bytes
$128 = 2^7$ 7 lines to address each chip
4 lines to decoder for selecting 16 chips

(C) 4×16 decoder

12-2

(a) 8 chips are needed with address lines connected in parallel.

(b) $16 \times 8 = 128$ chips. Use 14 address lines $(16K = 2^{14})$
10 lines specify the chip address
4 lines are decoded into 16 chip-select inputs.

12-3

10 pins for inputs, 4 for chip-select, 8 for outputs, 2 for power.
Total of 24 pins.

12-4

$4096/128 = 32$ RAM chips ; $4096/512 = 8$ ROM chips.
$4096 = 2^{12}$ — There 12 common address lines +1 line to
select between RAM and ROM.

Component	Address	16 15 14 13	12 11 10 9	8 7 6 5	4 3 2 1
RAM	0000-0FFF	0 0 0 0	←5×32 decoder→	X X X	X X X X
ROM	1000-1FFF	0 0 0 1	← 3×8 → decoder	X X X X X	X X X X

to $\overline{CS2}$

12-5

RAM $2048/256 = 8$ chips ; $2048 = 2^{11}$; $256 = 2^8$
ROM $4096/1024 = 4$ chips ; $4096 = 2^{12}$; $1024 = 2^{10}$
Interface $4 \times 4 = 16$ registers ; $16 = 2^4$

Component	Address	16 15 14 13	12 11 10 9	8 7 6 5	4 3 2 1
RAM	0000-07FF	0 0 0 0	0 ← 3×8 → decoder	X X X X	X X X X
ROM	4000-4FFF	0 1 0 0	← 2×4 → decoder X X	X X X X	X X X X
Interface	8000-800F	1 0 0 0	0 0 0 0	0 0 0 0	X X X X

12-6

The processor selects the external register with an address 8000 hexadecimal. Each bank of 32K bytes are selected by addresses 0000 – 7FFF. The processor loads an 8-bit number into the register with a single 1 and 7 0's. Each output of the register selects one of the 8 banks of 32K bytes through a chip-select input. A memory bank can be changed by changing the number in the register.

<u>12-7</u> Average time = T_s + time for half revolution +
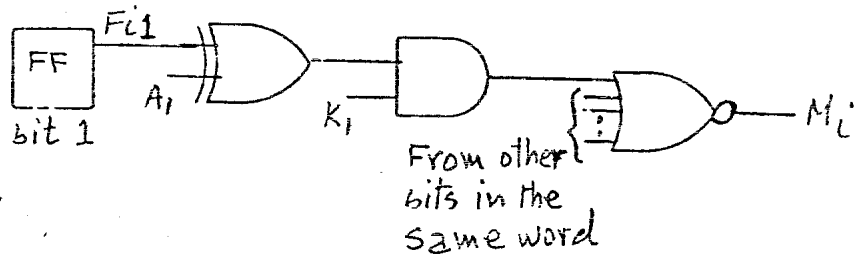
$\qquad$ + time to read a sector.

$$T_a = T_s + \frac{1}{2R} + \frac{1/s}{N_t} \times \frac{1}{R}$$

<u>12-8</u> An eight-track tape reads 8 bits (one character) at the sam time. Transfer rate $= 1600 \times 120 = 192,000$

$\qquad\qquad\qquad\qquad\qquad\qquad$ charaters/s

<u>12-9</u>

$\qquad$ From Sec. 12-4: $M_i = \prod_{j=1}^{n} \left[(A_j \oplus F_{ij})' + K_j' \right]$

$$M_i' = \sum_{j=1}^{n} (A_j \odot F_{ij}) K_j$$

$\qquad$ From other bits in the same word

<u>12-10</u>

$\qquad$ A match occurs if $T_i = 1$

$\qquad\qquad$ match $= M_i T_i$

match if 1
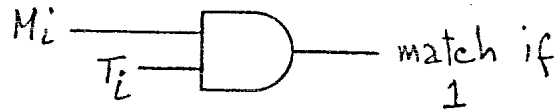
<u>12-11</u>

$$M_i = \left(\prod_{j=1}^{n} A_j F_{ij} + A_j' F_{ij}' + K_j' \right) \cdot (K_1 + K_2 + K_3 + \cdots + K_n)$$

$\qquad$ At least one key bit K_i must be equal to 1

<u>12-12(c)</u>

outputs

89
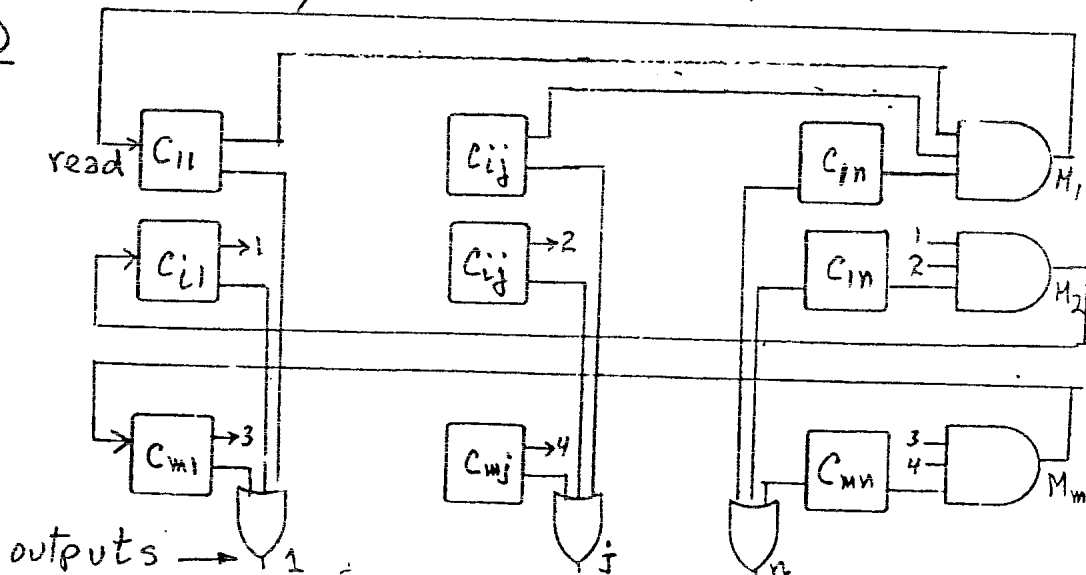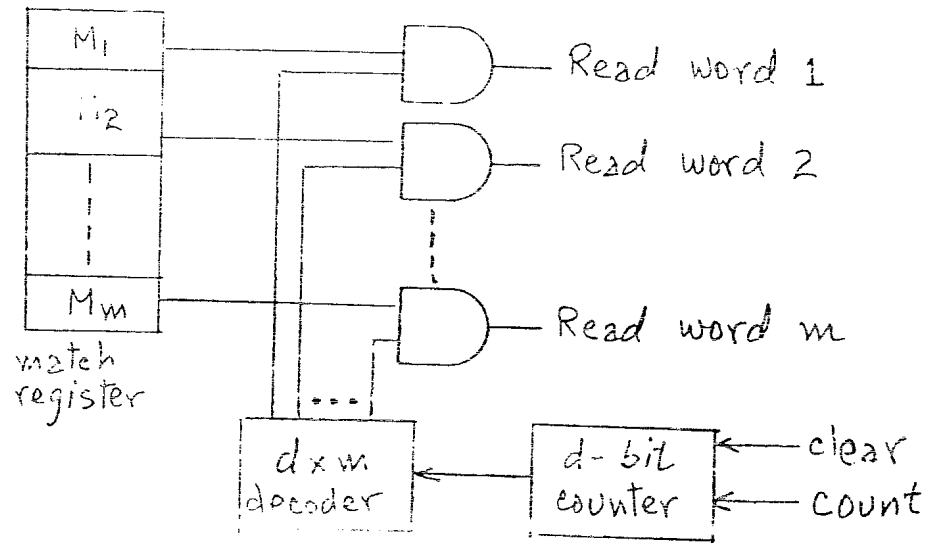
A d-bit counter drives a d-to-m line decoder where $2^d = m$ (m — No. of words in memory). For each count, the M_i bit is checked and if 1, the corresponding read signal for word i is activated.
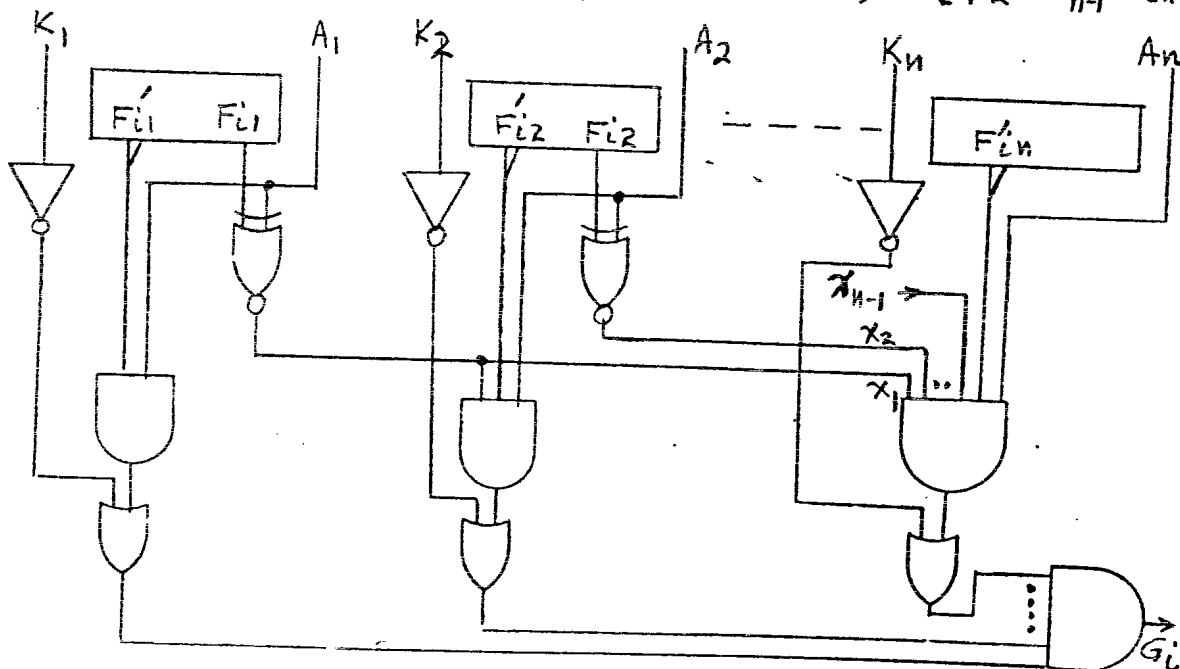
12-14

Let $x_j = A_j F_{ij} + A'_i F'_{ij}$ (argument bit = memory word bit)
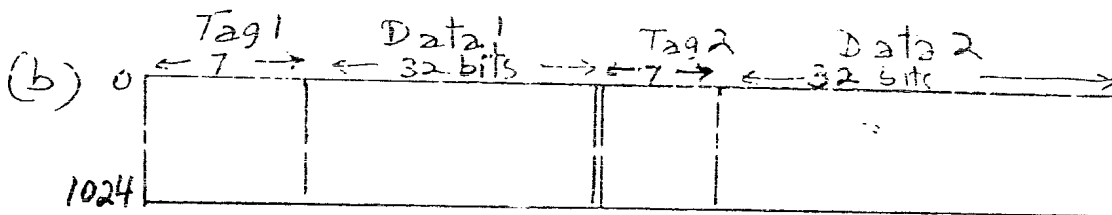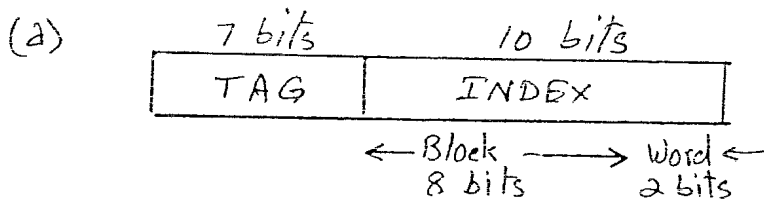
Output indicator $G_i = 1$ if:

$A_1 F'_{i1} = 1$ and $K_1 = 1$ (First bit in $A = 1$ while $F_{i1} = 0$)

or if $x_1 A_2 F'_{i2} = 1$ and $K_2 = 1$ (First pair of bits are equal and second bit in $A = 1$ while $F_{i2} = 0$)

— etc.

$$G_i = (A_1 F'_{i1} + K'_1)(x_1 A_2 F'_{i2} + K'_2)(x_1 x_2 A_3 F'_{i3} + K'_3) \cdots (x_1 x_2 \cdots x_{n-1} A_n F'_{in} + K'_n)$$

12-15

$128K = 2^{17}$; For a set size of 2, the index address has 10 bits to accomodate $\frac{2048}{2} = 1024$ words of cache.

(a)

7 bits	10 bits
TAG	INDEX

←— Block 8 bits —→ →Word← 2 bits

(b)

	Tag1 ←7→	Data1 ←32 bits→	Tag2 ←7→	Data2 ←32 bits→
0				
1024				

Size of cache memory is $1024 \times 2(7+32)$
$$= 1024 \times 78$$

12-16

(a) $0.9 \times \underbrace{100}_{\text{cache access}} + 0.1 \times \underbrace{11000}_{\substack{\text{cache+memory} \\ \text{access}}} = 90 + 110 = 200 \text{ nsec.}$

(b) $0.2 \times \underbrace{1000}_{\substack{\text{write access}}} + 0.8 \times \underbrace{200}_{\substack{\text{read access} \\ \text{from (a)}}} = 200 + 160 = 360 \text{ nsec.}$
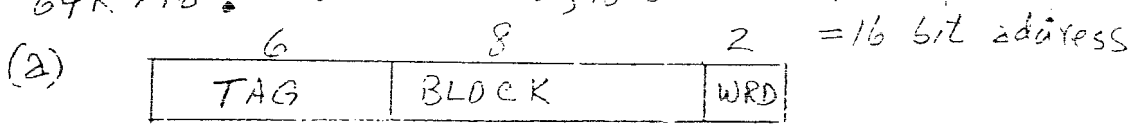
(c) Hit ratio $= 0.8 \times 0.9 = 0.72$
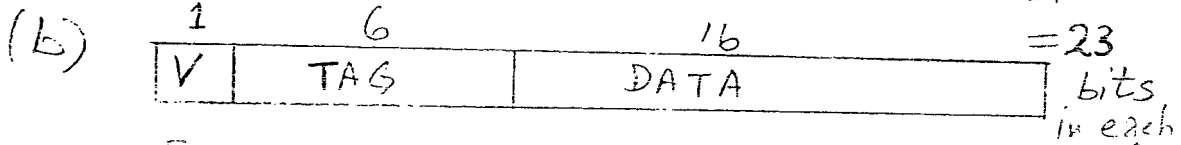
12-17

Sequence: A B C D B E D A C E C E

	LRU —→			
Count value =	3	2	1	0
Initial words =	A	B	C	D
B is a hit	A	C	D	B
E is a miss	C	D	B	E
D is a hit	C	B	E	D
A is a miss	B	E	D	A
C is a miss	E	D	A	C
E is a hit	D	A	C	E
C is a hit	D	A	E	C
E is a hit	D	A	C	E

12-18

64K × 16 : 16 bit address; 16-bit data.

(a)

6	8	2	= 16 bit address
TAG	BLOCK	WRD	

INDEX = 10 bit cache address.

(b)

1	6	16	= 23 bits in each word of cache
V	TAG	DATA	

(c) $2^8 = 256$ blocks of 4 words each

12-19

(a) Address space = 24 bits $2^{24} = 16M$ words

(b) Memory space = 16 bits $2^{16} = 64K$ words

(c) $\dfrac{16M}{2K} = 8K$ pages $\dfrac{64K}{2K} = 32$ blocks

12-20

The pages that are not in main memory are:

Page	Address	address that will cause fault
2	2K	2048 – 3071
3	3K	3072 – 4095
5	5K	5120 – 6143
7	7K	7168 – 8191

12-21

4 2 0 1 2 6 1 4 0 1 0 2 3 5 7

Page reference	(a) Pages in main memory	Contents of FIFO (First in →)	(b) Pages in memory	LRU	Most recently used
Initial	0 1 2 4	4 2 0 1	0 1 2 4	4 2 0 1	
2	0 1 2 4	4 2 0 1	0 1 2 4	4 0 1 2	
6	0 1 2 6	2 0 1 6	0 1 2 6	0 1 2 6	
1	0 1 2 6	2 0 1 6	0 1 2 6	0 2 6 1	
4	0 1 4 6	0 1 6 4	1 2 4 6	2 6 1 4	
0	0 1 4 6	0 1 6 4	0 1 4 6	6 1 4 0	
1	0 1 4 6	0 1 6 4	0 1 4 6	6 4 0 1	
0	0 1 4 6	0 1 6 4	0 1 4 6	6 4 1 0	
2	1 2 4 6	1 6 4 2	0 1 2 4	4 1 0 2	
3	2 3 4 6	6 4 2 3	0 1 2 3	1 0 2 3	
5	2 3 4 5	4 2 3 5	0 2 3 5	0 2 3 5	
7	2 3 5 7	2 3 5 7	2 3 5 7	2 3 5 7	

12-22

    600AF   and   F00AF

12-23

Logical address:

7 bits	5 bits	12 bits	= 24 bits
Segment	Page	Word	

Physical address:

12 bits	12 bits
Block	Word

12-24

Segment $36 = (0100100)_2$ (7-bit binary)

Page $15 = (01111)_2$ (5-bit binary)

Word $2000 = (011111010000)_2$ (12-bit binary)

Logical address = 0100100 01111 011111010000

                (24-bit binary)

13-1

Tightly coupled multiprocessors require that all processors in the system have access to a common global memory. In loosely coupled multiprocessors, the memory is distributed and a mechanism is required to provide message-passing between the processors. Tightly coupled systems are easier to program since no special steps are required to make shared data available to two or more processors. A loosely coupled system required that sharing of data be implemented by the messages.

13-2

The address assigned to common memory is never assigned to any of the local memories. The common memory is recognized by its distinct address.

13-3

P×m switches

13-4

$\log_2 n$ stages with $\frac{n}{2}$ switches in each stage.

13-5

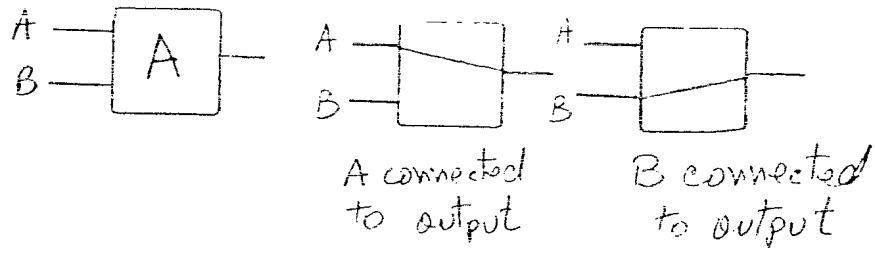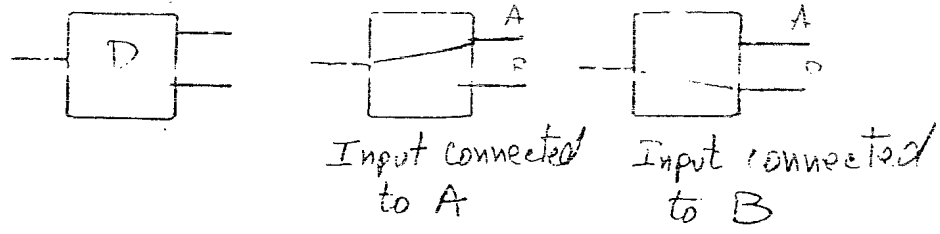Inputs 0, 2, 4, and 6 will be disconnected from outputs 2 and 3.
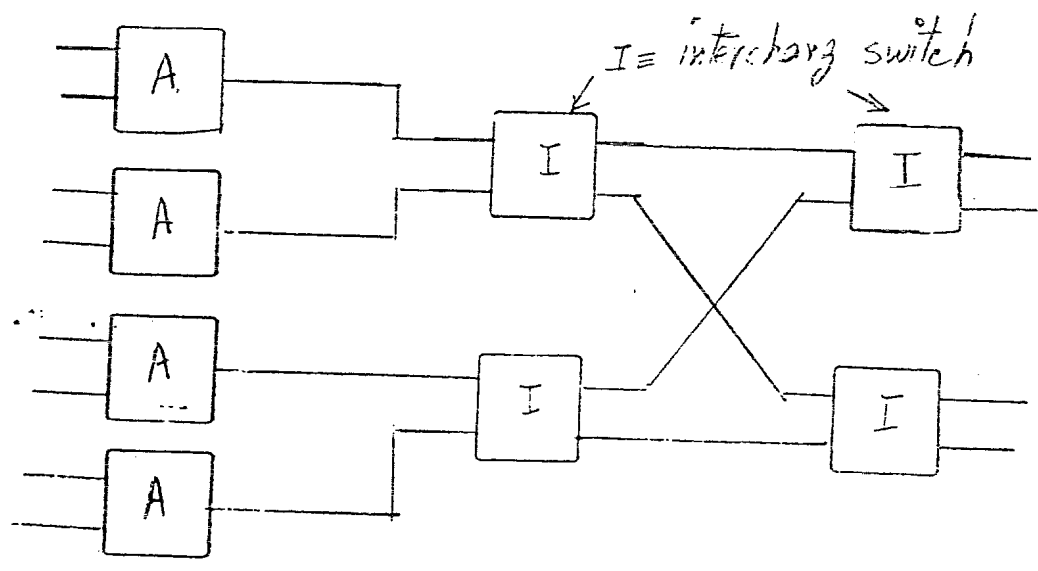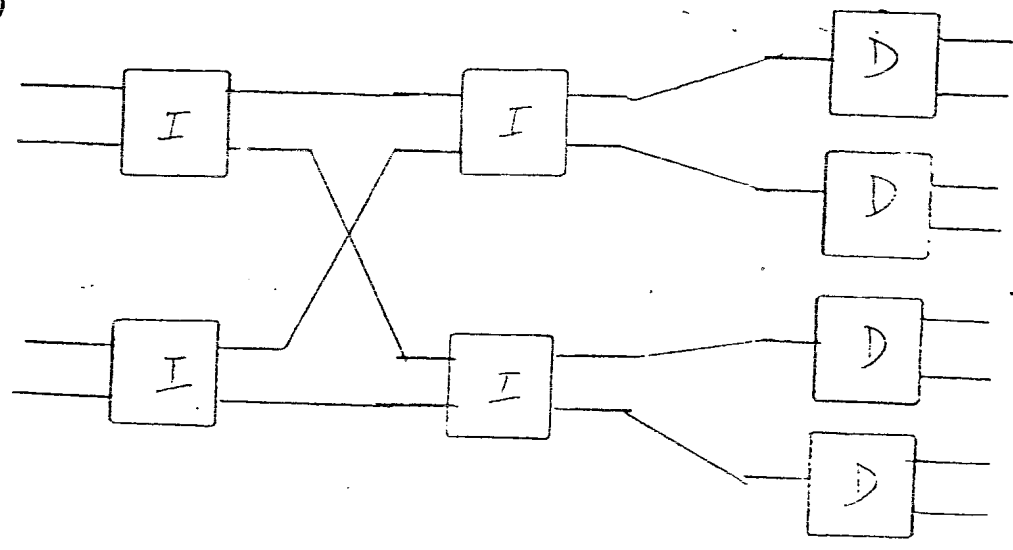
13-6

Arbitration switch:

(a)

A connected
to output

B connected
to output

Distribution switch:

Input connected
to A

Input connected
to B

(b)

I ≡ intercharg switch

(c)
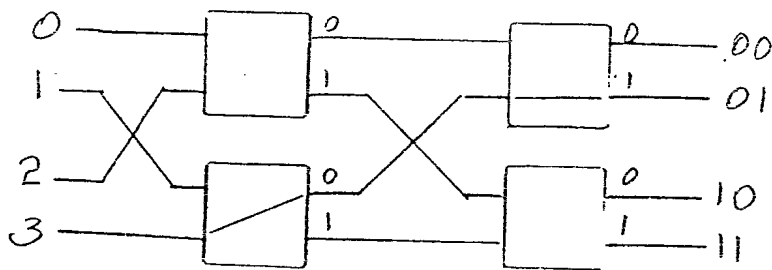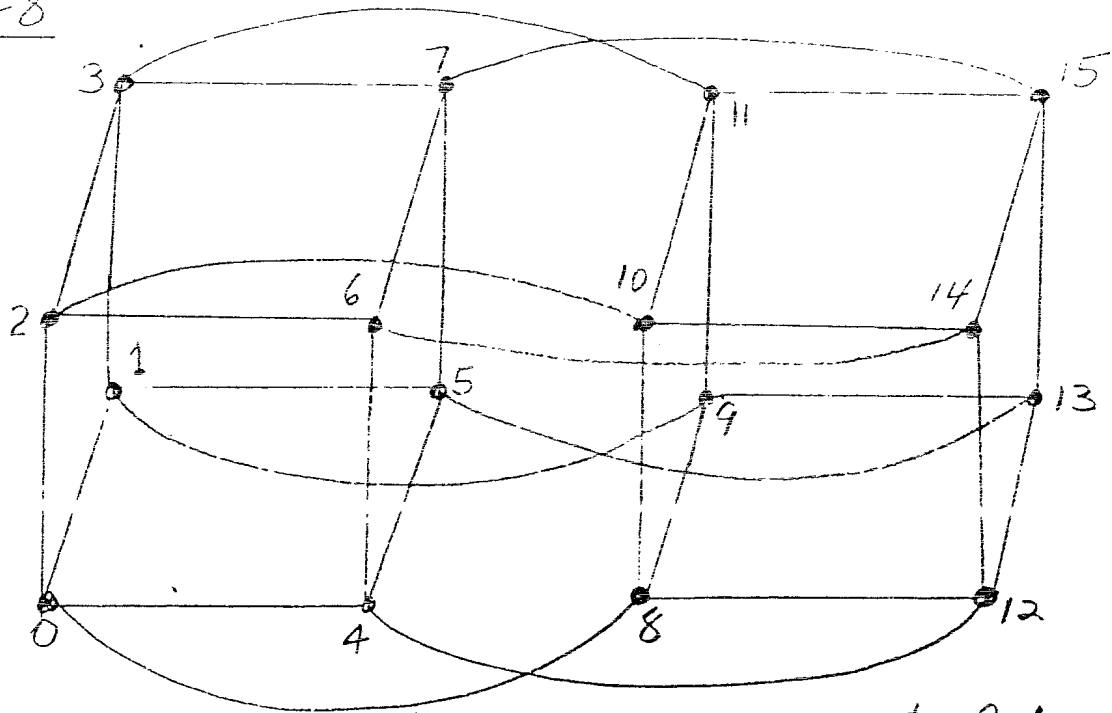
$$7 = 0111 \oplus$$
$$9 = \underline{1001}$$
$$\overline{1110} = three\ axes$$

Paths from 7 to 9:

7 - 15 - 13 - 9
7 - 15 - 11 - 9
7 - 3 - 11 - 9
7 - 3 - 1 - 9
7 - 5 - 13 - 9
7 - 5 - 1 - 9

13-9

Request for bus transfer

Transfer completed

or PI = 0 *
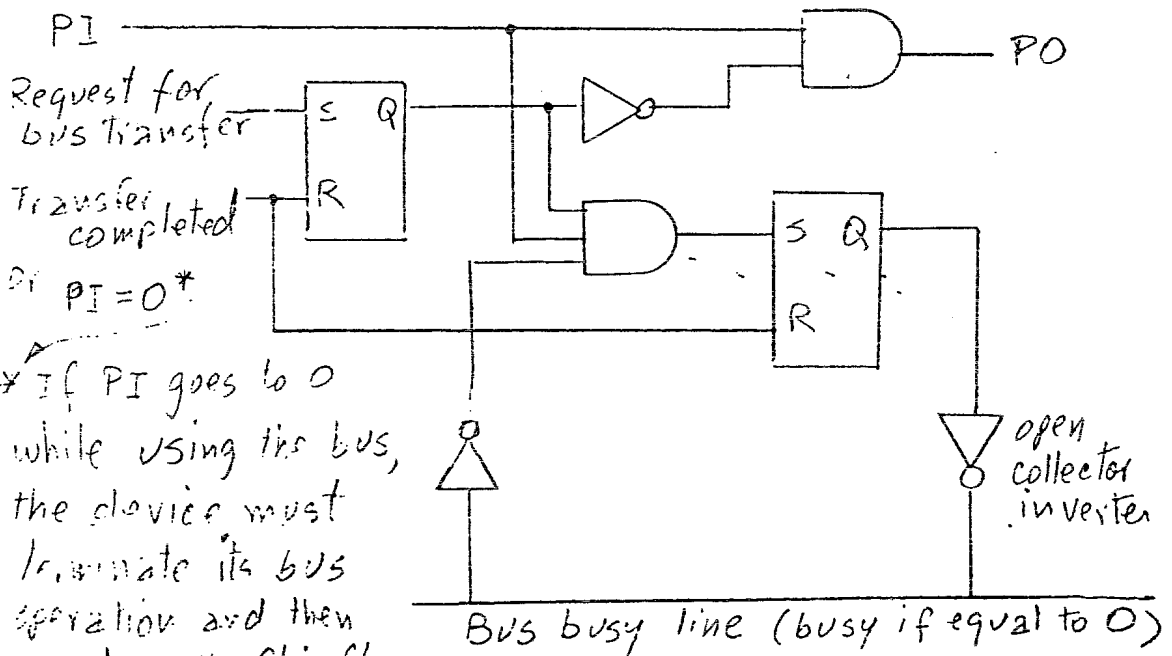
* If PI goes to 0 while using the bus, the device must terminate its bus operation and then reset both flip-flops.

Bus busy line (busy if equal to 0)

open collector inverter

13-10

Encoder input	0110	
Encoder output	01	(I_1 has highest priority)
Decoder input	01	
Decoder output	0100	Arbiter 2 (I_1) is acknowledged

13-11

As explained in the text, connect output PO from arbiter 4 into input PI of arbiter 1. Once the line is disabled, the arbiter that releases the bus has the lowest priority.

13-12

Memory access needed to send data from one processor to another must be synchronized with test-and-set instructions. Most of the time would be taken up by unsuccessful test by the receiver. One way to speed the transfer would be to send an interrupt request to the receiving processor.

13-13

(a) <u>Mutual exclusion</u> implies that each processor claims exclusive control of the resources allocated to it.

(b) <u>Critical section</u> is a program sequence that must be completly executed without interruptions by other processors.

(Continued in next page)

13-13 (Continued)

(c) <u>Hardware lock</u> is a hardware signal to assure that a memory read is followed by a memory write without interruption from another processor.

(d) <u>Semaphore</u> is a variable that indicates the number of processes attempting to use the critical section..

(e) <u>Test and set instruction</u> causes a read-modify-write memory operation so that the memory location cannot be accessed and modified by another processor.

11-14

Cache coherency is defined as the situation in which all cache copies of shared variables in a multiprocessor system have the same value at all times. A snoopy cache controller is a monitoring action that detects a write operation into any cache. The cache coherence problem can be resolved by either updating or invalidating all other cache values of the written information.