CHAPTER 1

1-1

A	B	C	A·B·C	$(A\cdot B\cdot C)'$	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) $(BC' + A'D)(AB' + CD') =$

$= \underbrace{ABB'C'}_{0} + \underbrace{A'AB'D}_{0} + \underbrace{BCC'D'}_{0} + \underbrace{A'CD'D}_{0} = 0$

1-5 (a) $(A+B)'(A'+B')' = (A'B')(AB) = 0$

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

1-6 $F = x'y + xyz'$

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

$= y'(1 + x' + x + z) + xz = y' + xz$

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

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

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

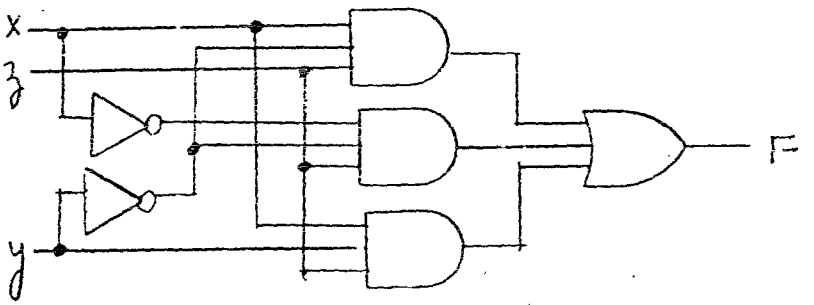
$= y(x'+x) + y' = y + y' = 1$

1-7

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

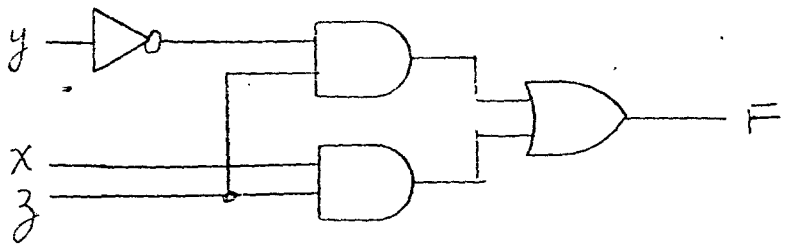


(d) Same as (a)

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

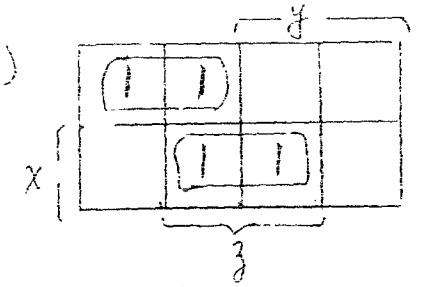
$= y'z(x+x') + xz(y+y')$

$= y'z + xz$



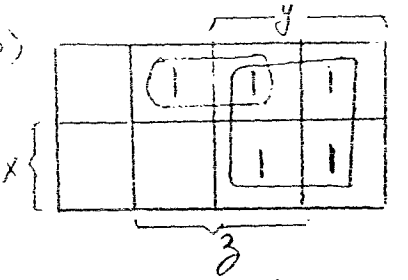
1-8

(a)



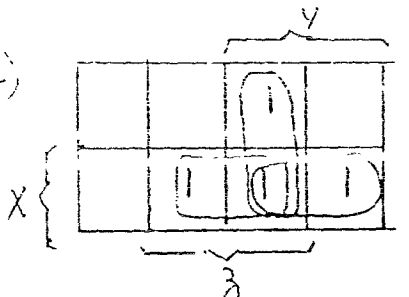
$F = x'y' + xz$

(b)



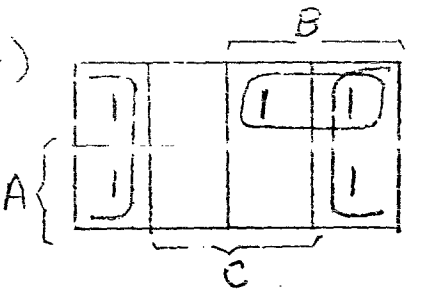
$F = y + x'z$

(c)



$F = xy + xz + yz$

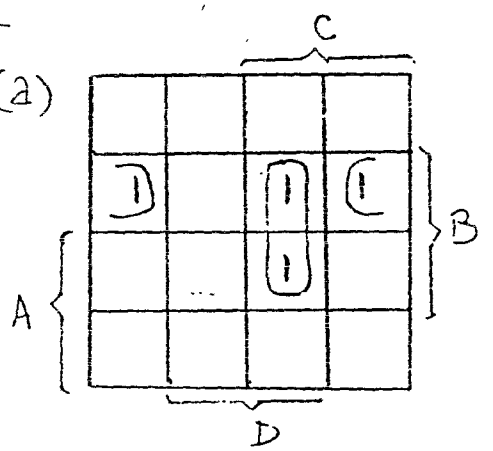
(d)



$F = C' + A'B$

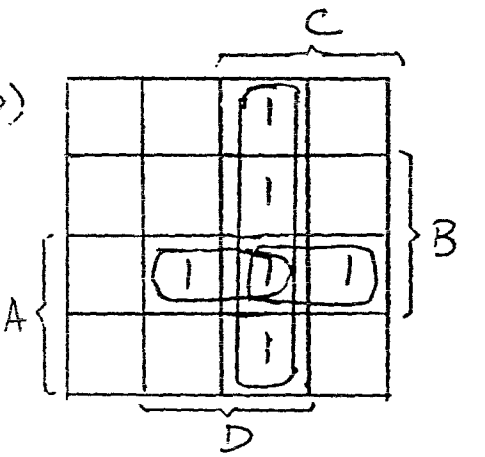
1-9

(a)



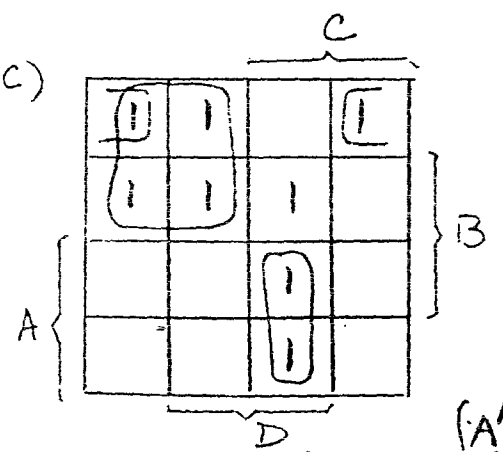
$F = BCD + A'BD'$

(b)



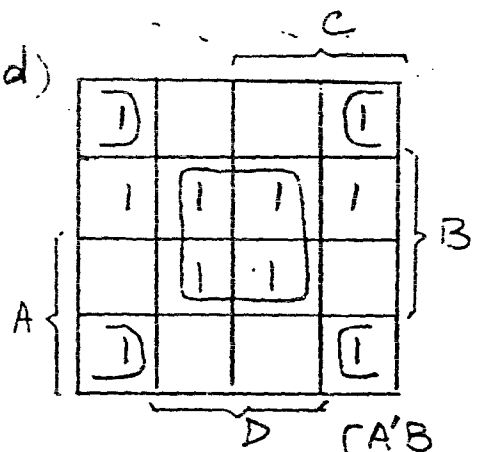
$F = CD + ABC + ABD$

(c)



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

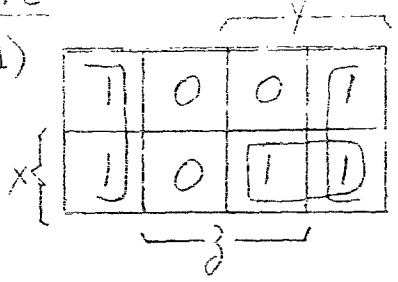
(d)



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

1-10

(a)

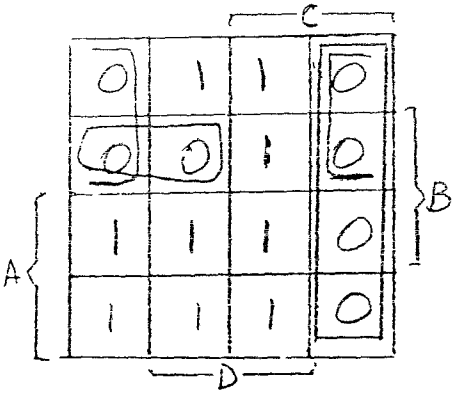


(1) $F = xy + z'$

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

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

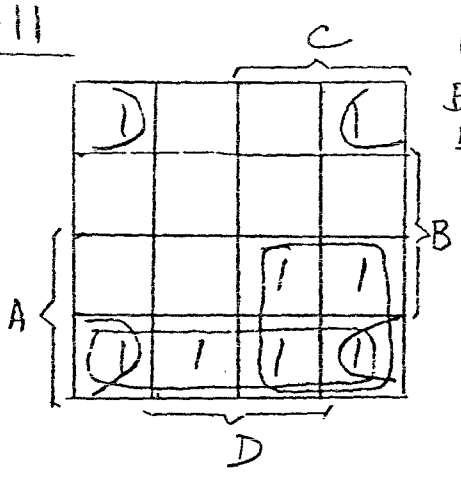
(b)



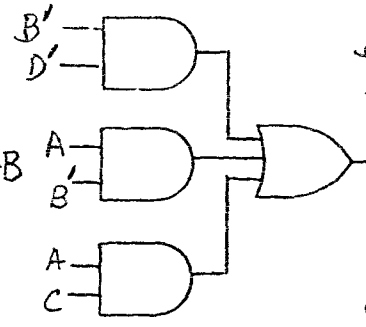
(1) $F = AC' + CD + B'D$

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

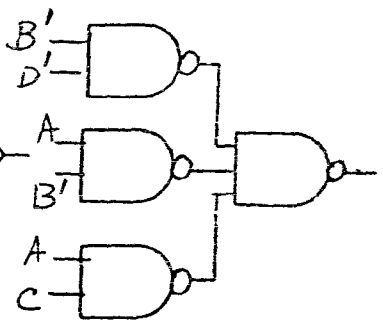
1-11



(a)

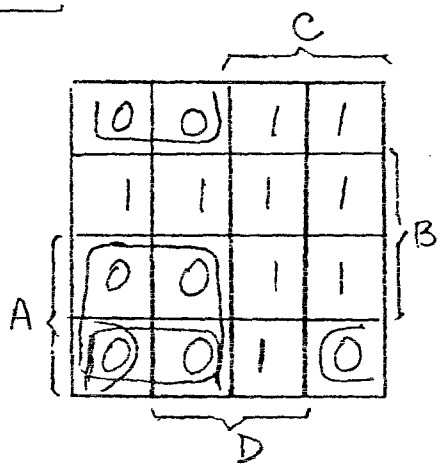


(b)



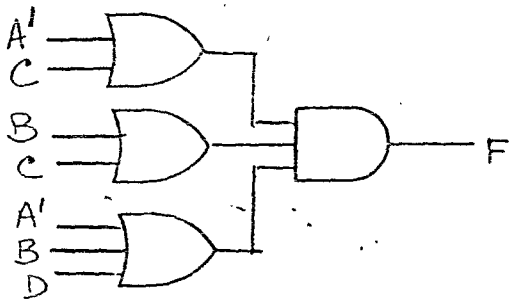
$F = B'D' + AB' + AC$

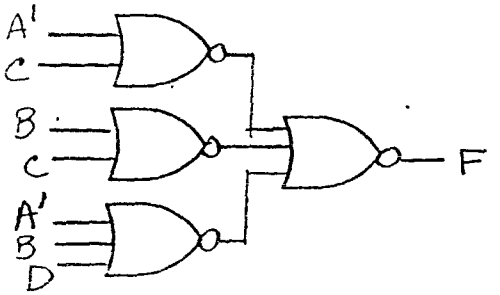
1-12



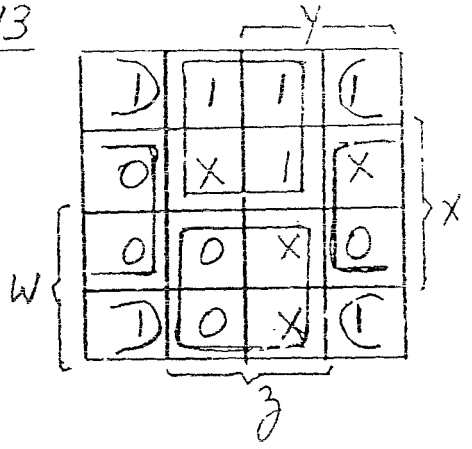
$F' = AC' + B'C' + AB'D'$

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





1-13



(a) $F = x'z' + w'z$

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

1-14

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

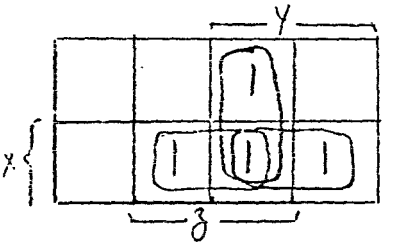
$= x'(y'z + yz') + x(y'z' + yz)$ ← See Fig 1-2 (Exclusive-NOR)

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

$= x \oplus y \oplus z$

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



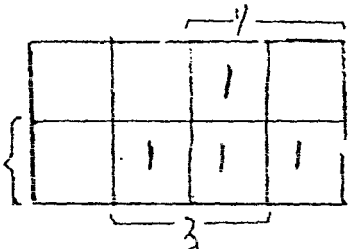
$F = xy + xz + yz$

1-16

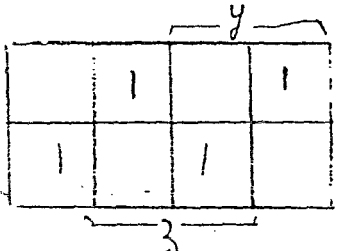
x	y	z	A	B	C
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	1	0	1	0	1
1	1	1	1	1	1

$C = z'$

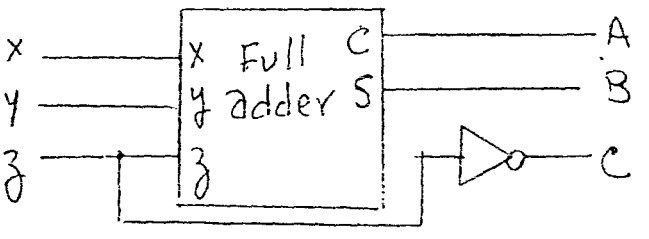
By inspection



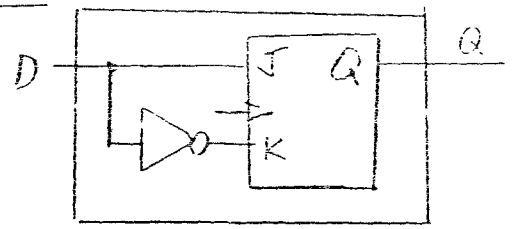
$A = xy + xz + yz$



$F = x \oplus y \oplus z$



1-17



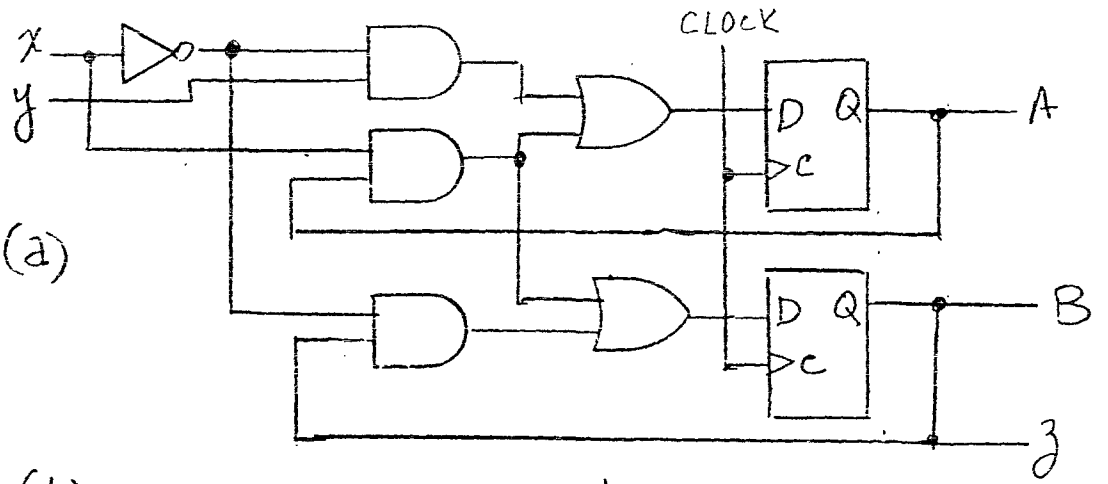
When $D=0$; $J=0, K=1, Q \to 0$

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

1-18

See text; Section 1-6 for derivation.

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



(a)

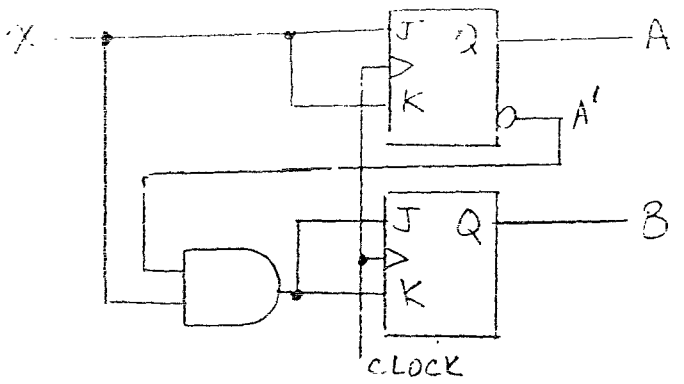
(b)

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

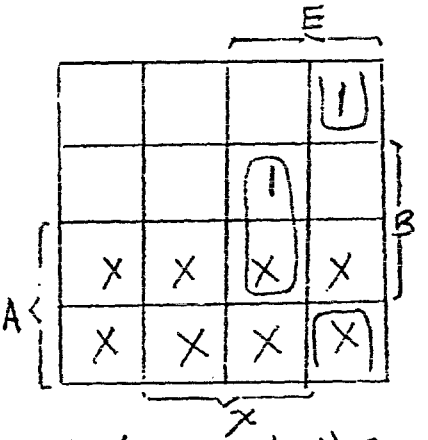
$J_A = K_A = x$

$J_B = K_B = A'x$

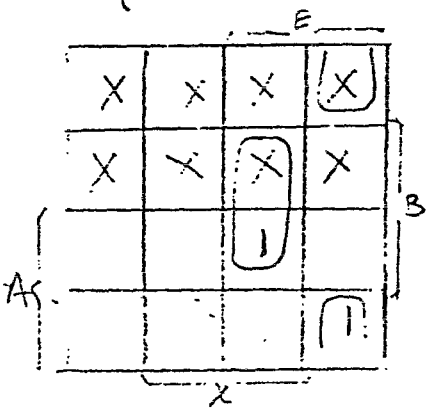


1-21 Count up-down binary counter with enable E

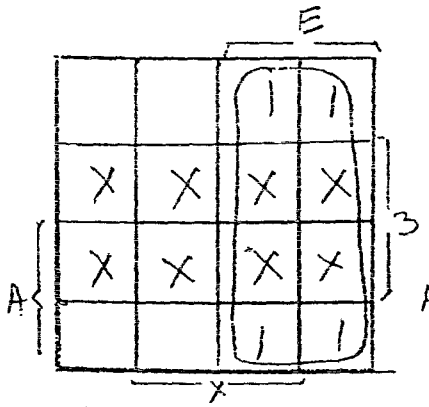
Present state		Inputs		Next state		Flip-flop inputs			
A	B	E	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	0	0	0	X	0	X
0	0	0	1	0	0	0	X	0	X
0	0	1	0	1	1	1	X	1	X
0	0	1	1	0	1	0	X	1	X
0	1	0	0	0	1	0	X	X	0
0	1	0	1	0	1	0	X	X	0
0	1	1	0	0	0	0	X	X	1
0	1	1	1	1	0	1	X	X	1
1	0	0	0	1	0	X	0	0	X
1	0	0	1	1	0	X	0	0	X
1	0	1	0	0	1	X	1	1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



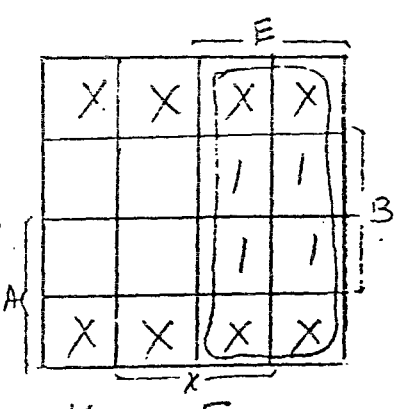
$J_A = (Bx + B'x')E$



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



$J_B = E$



$K_B = E$

CHAPTER 2

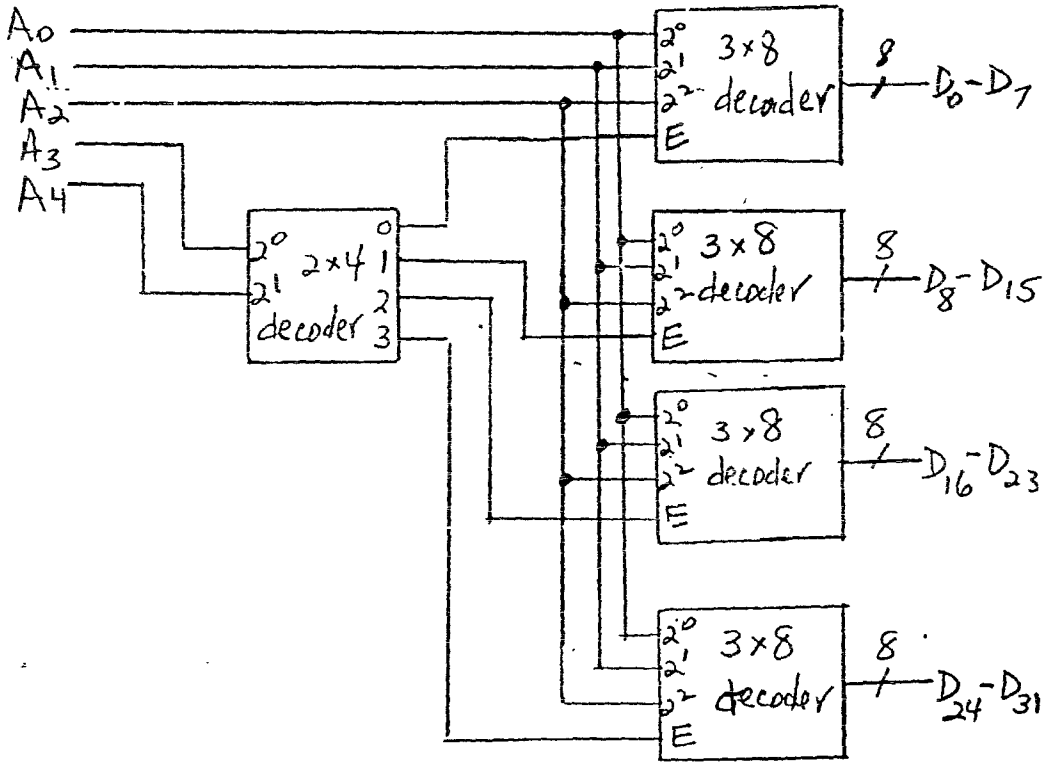
2-1

		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

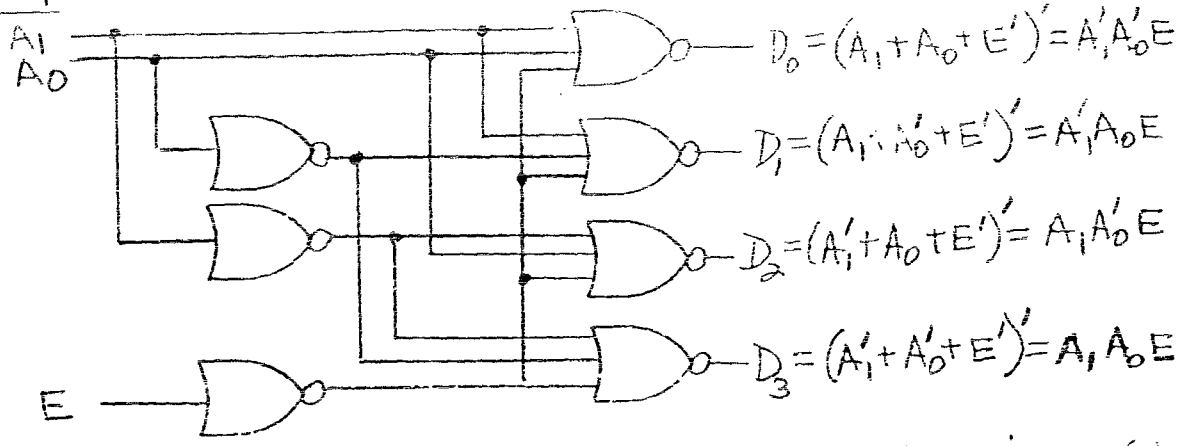
2-2

(a) 74155 – Similar to two decoders as in Fig 2-2.
(b) 7415 7 – 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

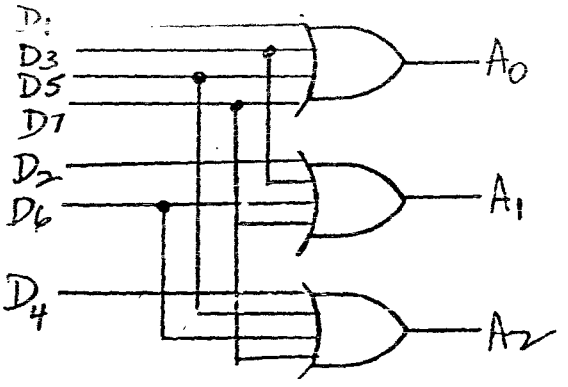


2-4



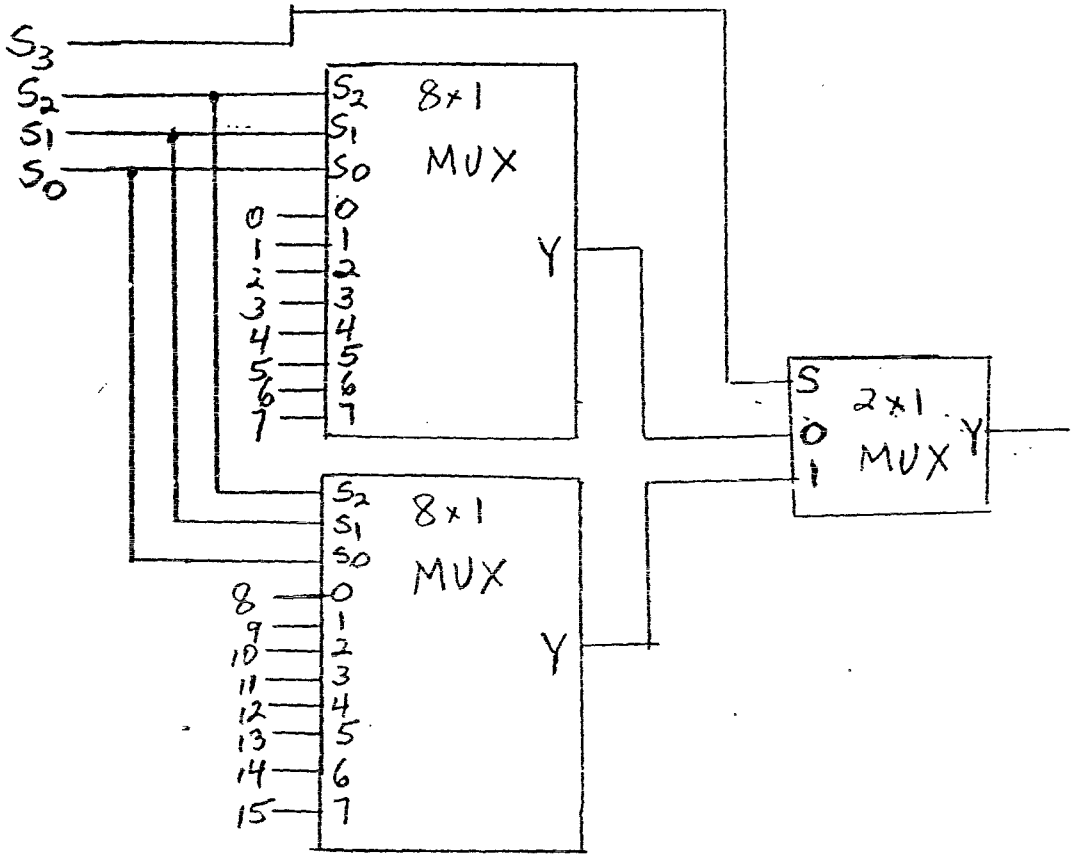
2-5 Remove the inverter from the E input in Fig. 2-2(a).

2-6

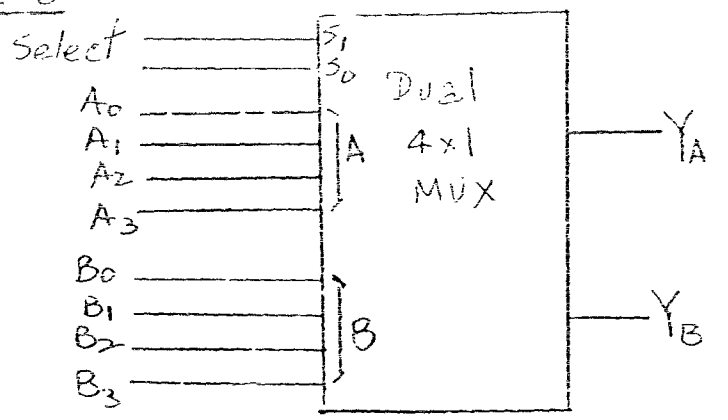


If all inputs equal 0 or if only $D_0=1$: the outputs $A_2A_1A_0=000$. Needs one more output to recognize the all zeros input condition.

2-7



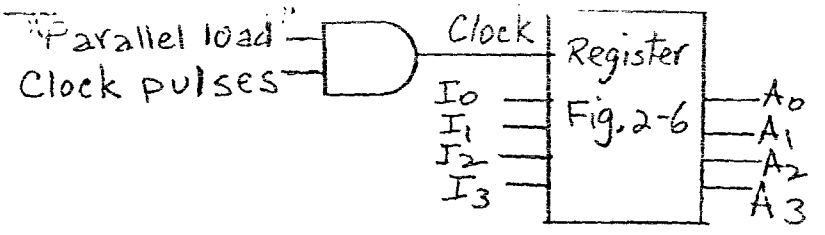
2-8



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

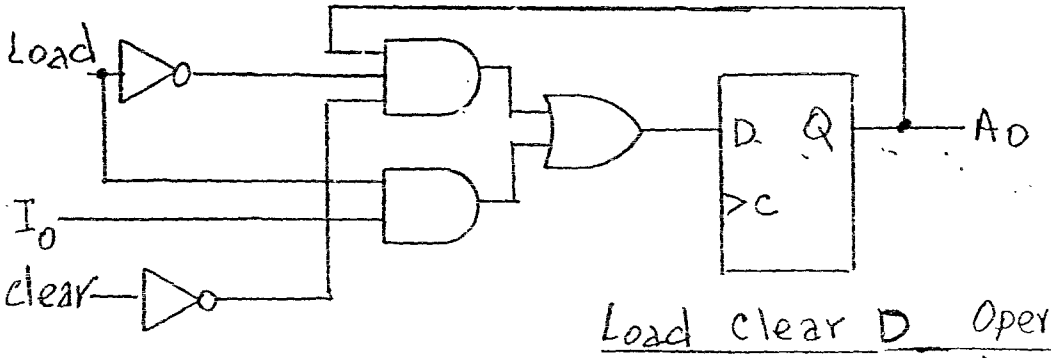


When the parallel load input = 1, the clock pulses go through the AND gate and the data inputs are loaded into the register. When the parallel load input = 0, the output of the AND gate remains at 0.

2-10

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

2-11



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

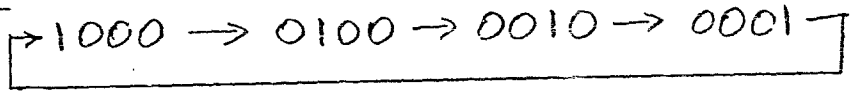
2-12

Input bits	4-bit register
	1101 ← Initial value
1	1110
0	0111
1	1011
1	1101
0	0110
1	1011

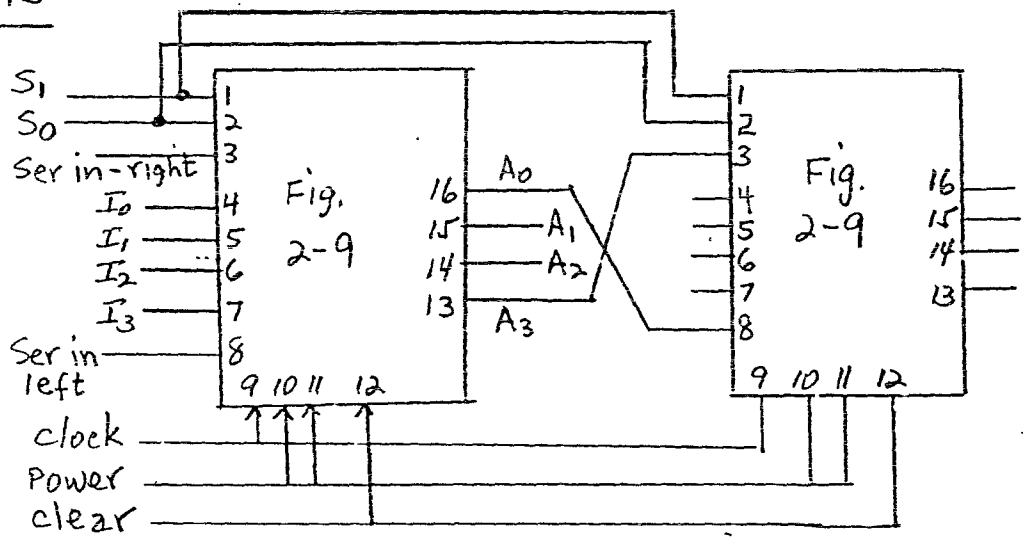
2-13

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

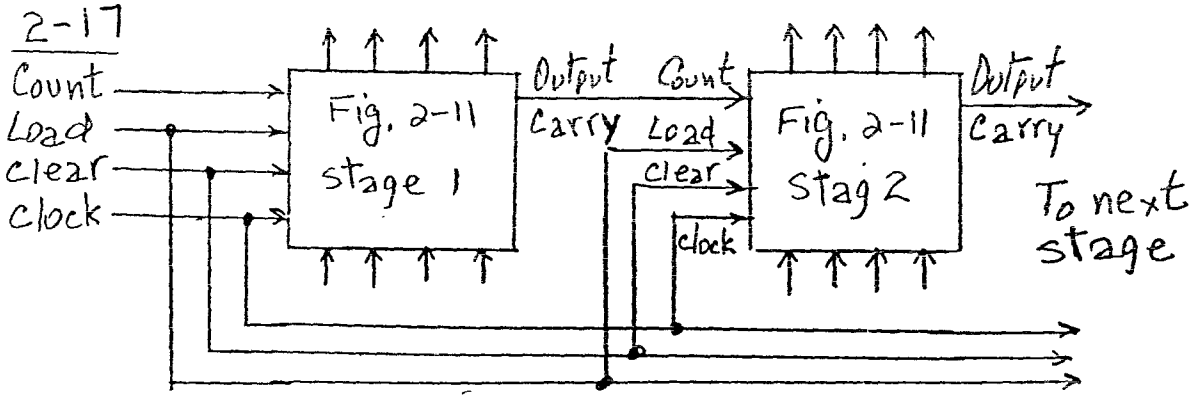


2-15



2-16 (a) 4 ; (b) 9

2-17



2-18

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

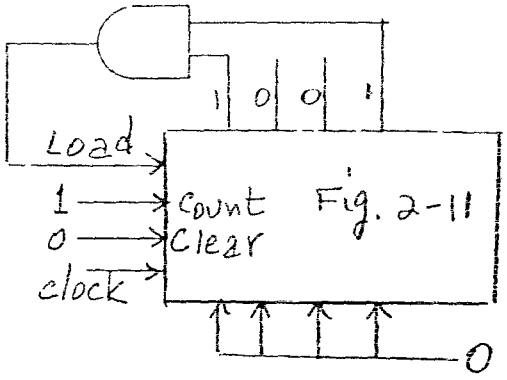


Fig. 2-11

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

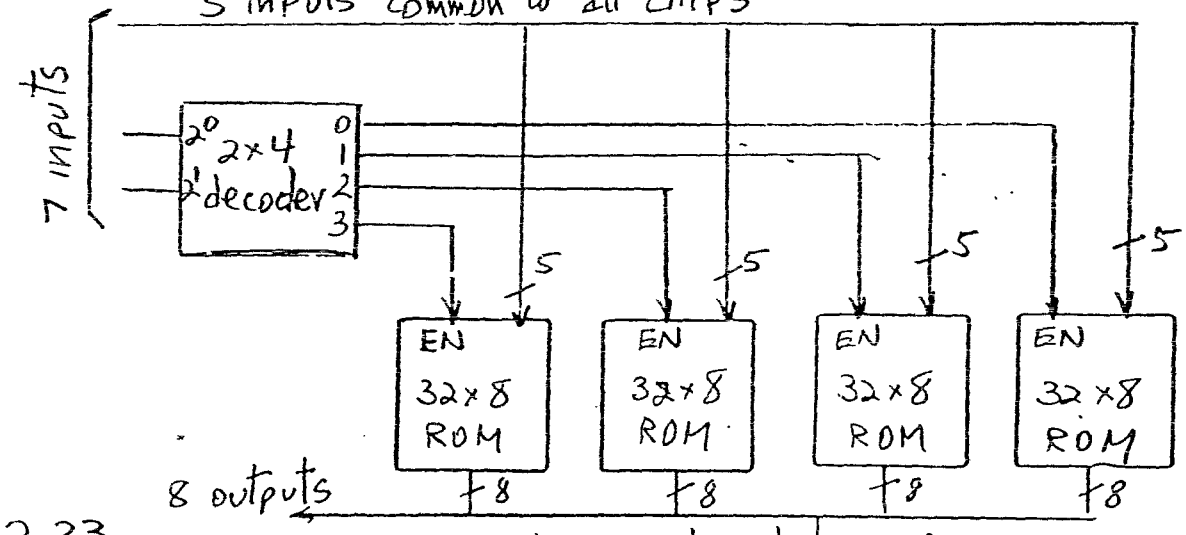
(b) $64K \times 1 = 64K = 2^{16}$ bytes

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

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

2-21 $\frac{4096 \times 16}{128 \times 8} = \frac{2^{12} \times 2^4}{2^7 \times 2^3} = 2^6 = 64$ chips

2-22



2-23

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

CHAPTER 3

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

$(12121)_3 = 3^4 + 2\times3^3 + 3^2 + 2\times3 + 1 = 81+54+9+6+1 = 151$

$(4310)_5 = 4\times5^3 + 3\times5^2 + 5 = 500 + 75 + 5 = 580$

$(50)_7 = 5\times7 = 35$

$(198)_{12} = 12^2 + 9\times12 + 8 = 144+108+8 = 260$

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

Also $(31)_r = 3\times13 + 1 = (40)_{10}$

$(r=13)$

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

(a) 0000 1101 0111 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 + 8679 = 13929

(b) 1753 + 1360 = 03113
↓ 10's complement ↓
−6887

(c) 020 + 900 = 0920
−080

(d) 1200 + 9750 = 10950

3-15

(a)	(b)	(c)	(d)
11010	11010	000100	1010100
+10000	+10011	010000	0101100
101010	101101	010100	00000000
(26−16=10)	(26−13=13)	↓ −101100	(84−84=0)
		(4−48=−44)	

3-16

+42 = 0101010
−42 = 1010110

(+42) 0101010
(−13) 1110011
(+29) 0011101

+13 = 0001101
−13 = 1110011

(−42) 1010110
(+13) 0001101
(−29) 1100011

3-17 01 ← last two carries → 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....1 = $1-2^{-26}$
+11111111 = +255
$(1-2^{-26}) \times 2^{+255}$

Smallest: +0.1000....0 (normalized) = 2^{-1}
−11111111 = −255
2^{-256}

3-20

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

Sign
0 10111010000000000 — 24-bit mantissa
00000110 — 8-bit exponent (+6)

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 0010
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	← AB = 00 or 11, CD = 01 or 10
1	0	1	← AB = 01 or 10, CD = 00 or 11
1	1	0	

ABCD

0001, 0010, 1101, 1110

0100, 0111, 1000, 1011

Always odd number of 1's

3-26

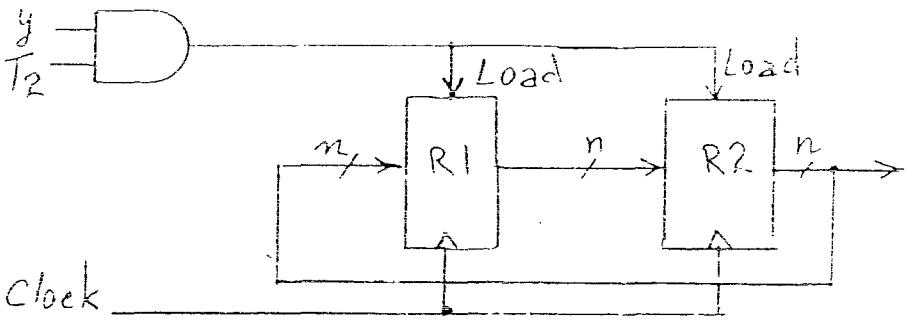
Same as in Fig. 3-3 but without the complemented circles in the outputs of the gates.

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

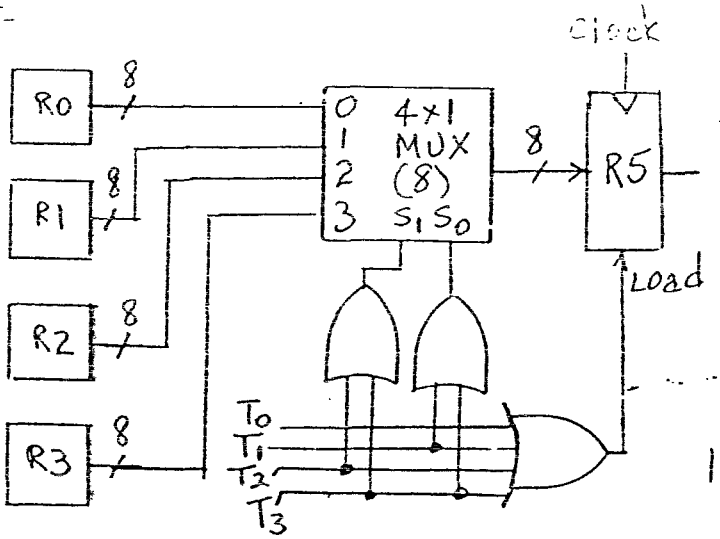
$$\text{Error} = x \oplus y \oplus z \oplus P$$

CHAPTER 4

4-1



4-2



T_0 T_1 T_2 T_3	S_1 S_0	R3 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: R1 ← R2

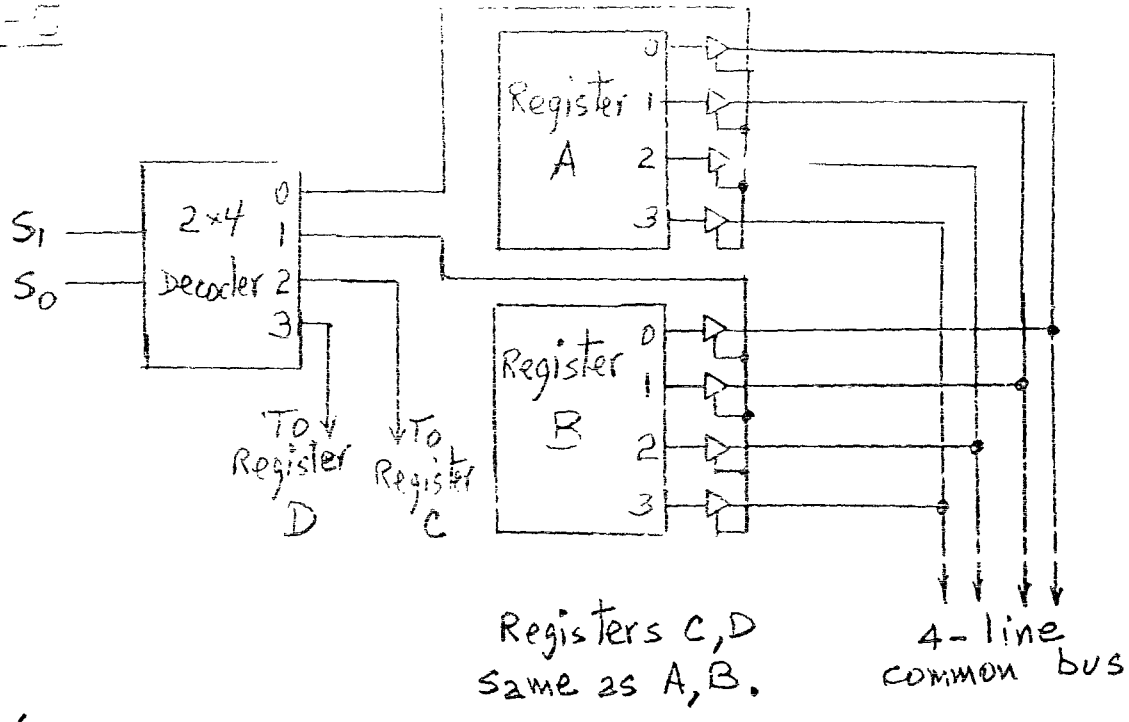
P'Q: R1 ← R3

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 $S_1S_0 = 10$ (to select C for the bus.)
Enable the load input of A
Apply a clock pulse.

4-5



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.

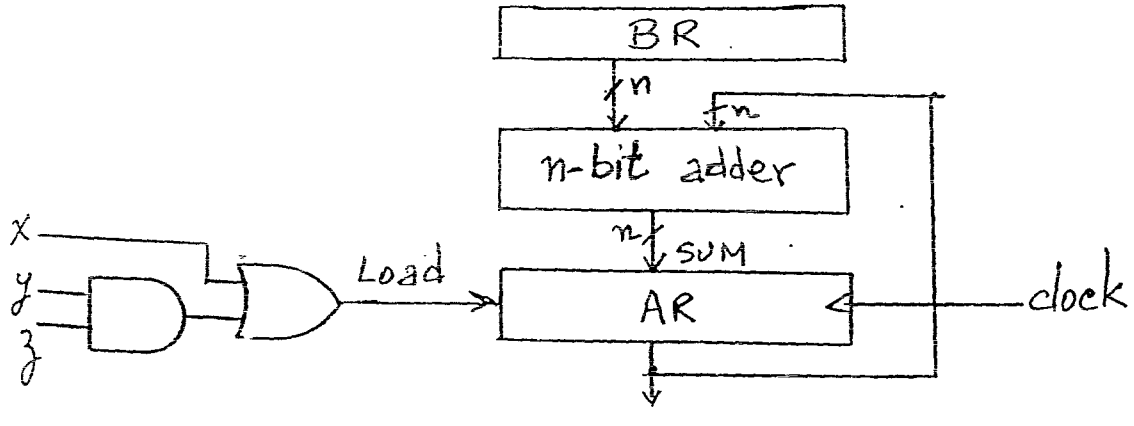
4-7

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

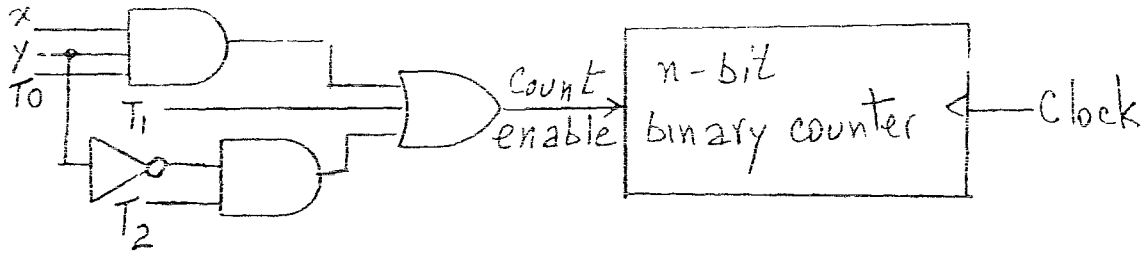
(b) Write content of register R3 into the memory word specified by the address in AR.

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

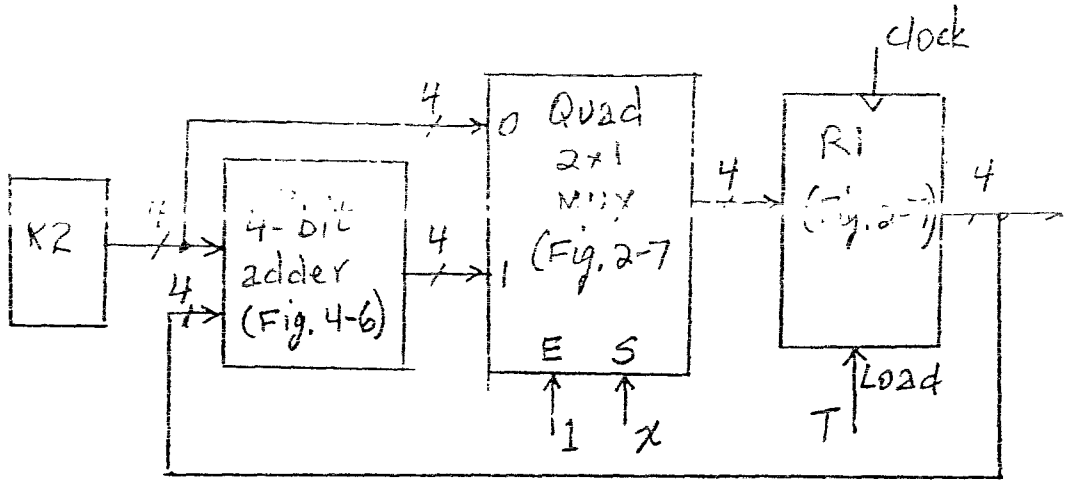
4-8



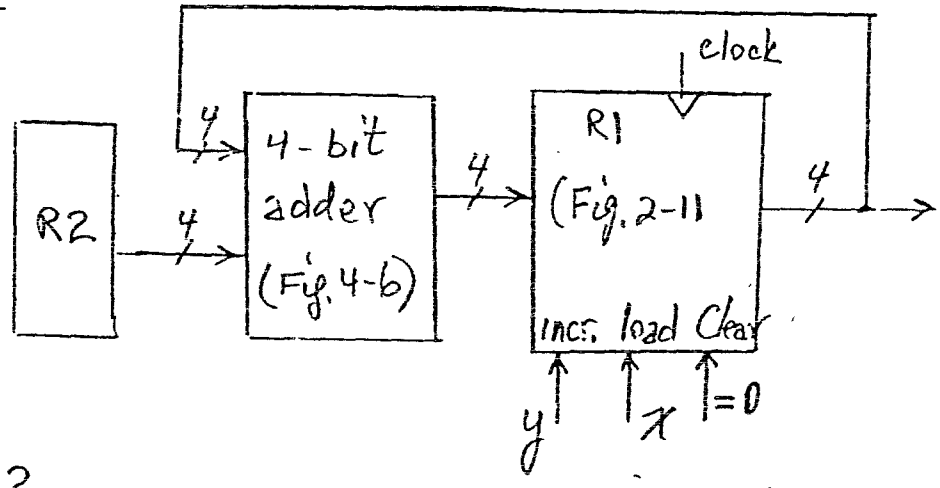
4-9



4-10



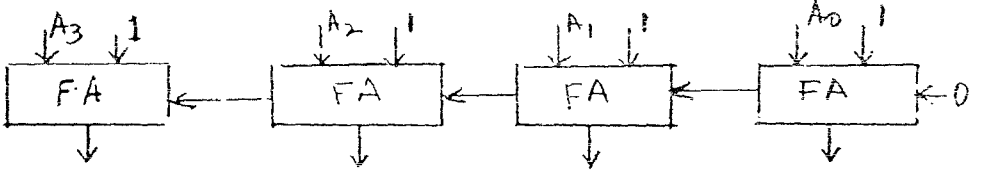
4-11



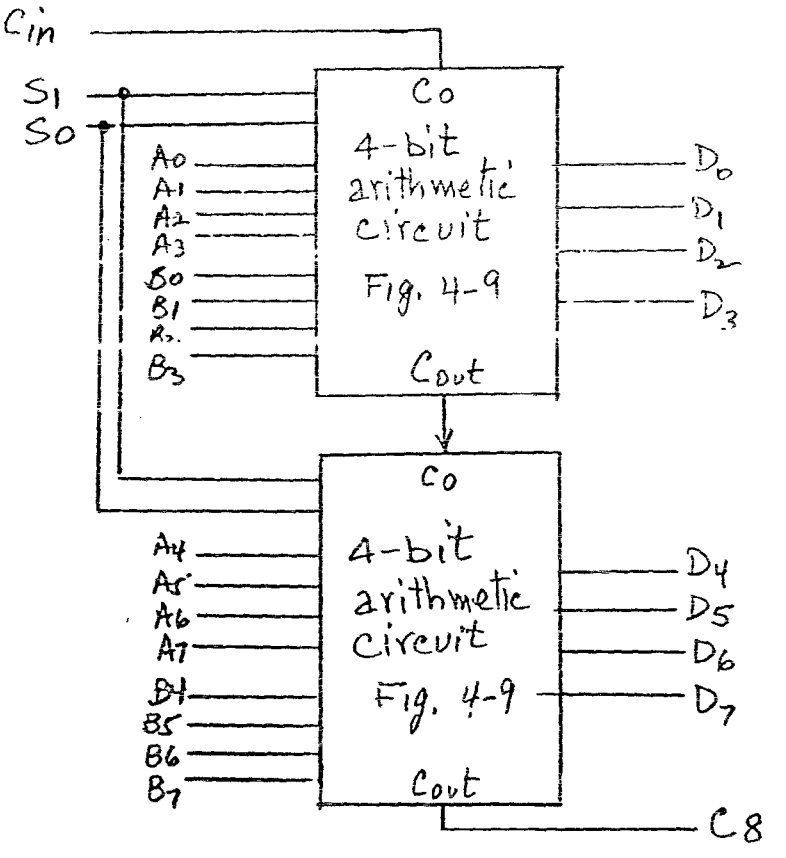
4-12

M	A	B	Sum	C_4	
0	0111	+ 0110	1101	0	7+6 = 13
0	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.)

4-13 A − 1 = A + 2's complement of 1 = A + 1111

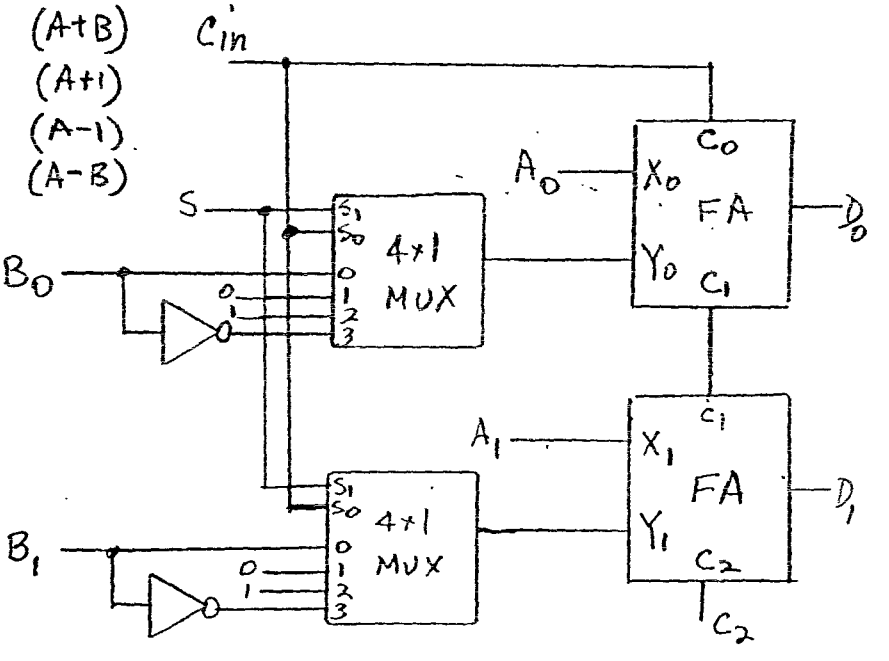


4-14

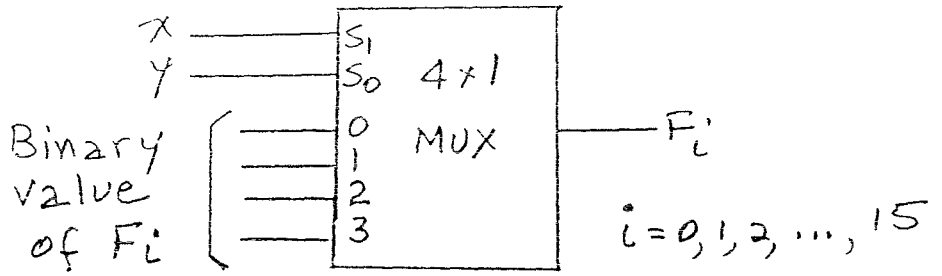


4-15

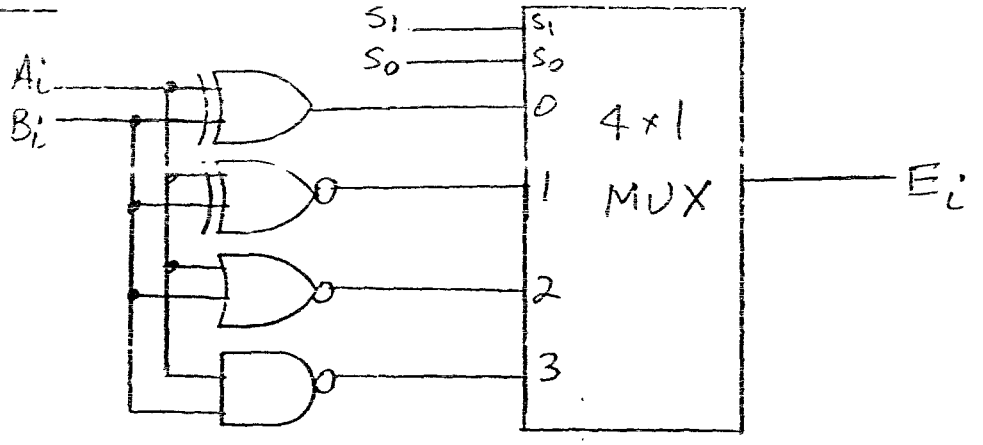
S	Cin	X	Y	
0	0	A	B	(A+B)
0	1	A	0	(A+1)
1	0	A	1	(A−1)
1	1	A	$\overline{B}$	(A−B)



4-16



4-17



4-18

(a) $A = 11011001$
$B = 10110100$ (⊕)
$A \leftarrow A \oplus B \quad 01101101$

$A = 11011001$
$B = 11111101$ (OR)
$11111101 \quad A \leftarrow A \vee B$

4-19

(a) $AR = 11110010$
$BR = 11111111$ (+)
$AR = 11110001 \quad BR = 11111111 \quad CR = 10111001 \quad DR = 11101010$

(b) $CR = 10111001$
$DR = 11101010$ (AND)
$CR = 10101000$

$BR = 11111111$
$+ \quad 1$
$BR = 00000000 \quad AR = 11110001 \quad DR = 11101010$

(c) $AR = 11110001$
$CR = 10101000$ (−)
$AR = 01001001$; $BR = 00000000$; $CR = 10101000$; $DR = 11101010$

4-20

R = 10011100

Arithmetic shift right : 11001110

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

4-21

R = 11011101

logical shift left : 10111010

Circular shift right : 01011101

logical shift right : 00101110

Circular shift left : 01011100

4-22

S=1 Shift left

$A_0\ A_1\ A_2\ A_3\ I_L$

1 0 0 1 0

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

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.

CHAPTER 5

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
25

32 − 25 = 7 bits for opcode.

(b)

1	7	6	18
I	Opcode	Register	Address

= 32 bits

(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 ← M[AR]

(b) TR to bus and load to PC: PC ← TR

(c) AC to bus, write to memory, and load to DR:
DR ← AC, M[AR] ← AC

(d) Add DR (or INPR) to AC: AC ← AC + DR

5-4

		(1) $S_2S_1S_0$	(2) Load (LD)	(3) Memory	(4) Adder
(a)	AR ← PC	010 (PC)	AR	—	—
(b)	IR ← M[AR]	111 (M)	IR	Read	—
(c)	M[AR] ← TR	110 (TR)	—	Write	—
(d)	DR ← AC AC ← DR	100 (AC)	DR and AC	—	Transfer DR to AC

5-5

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

AR ← PC
IR ← M[AR]

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

DR ← TR
AC ← AC + DR

(c) DR ← 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 ← DR, DR ← AC (See answer to Problem 5-4(d))
AC ← AC + DR
AC ← DR, DR ← AC

5-6

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

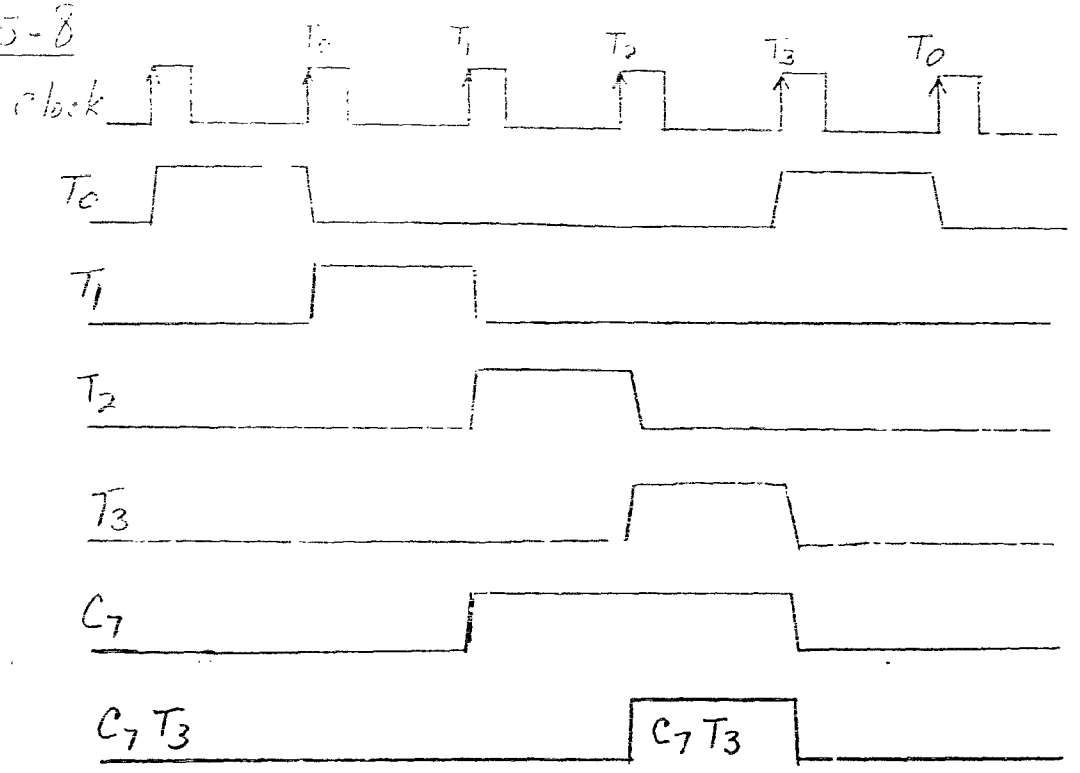
(b) 1011 0001 0010 0100 $= (B124)_{16}$
I STA (124)6
store AC in M[M[124]] STA I 124

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

5-7

CLE clear E
CME Complement E

5-8



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

(a) $9 = (1001)_2$

I=1 1 001 ADD ADD I 32E

Memory

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

AC = 7EC3

(b)

AC = 7EC3 (ADD)
DR = 8B9F
0A62

E = 1

(c) PC = 3AF+1 = 3B0
AR = 9AC
DR = 8B9F
AC = 0A62

IR = 932E
E = 1
I = 1
SC = 0000

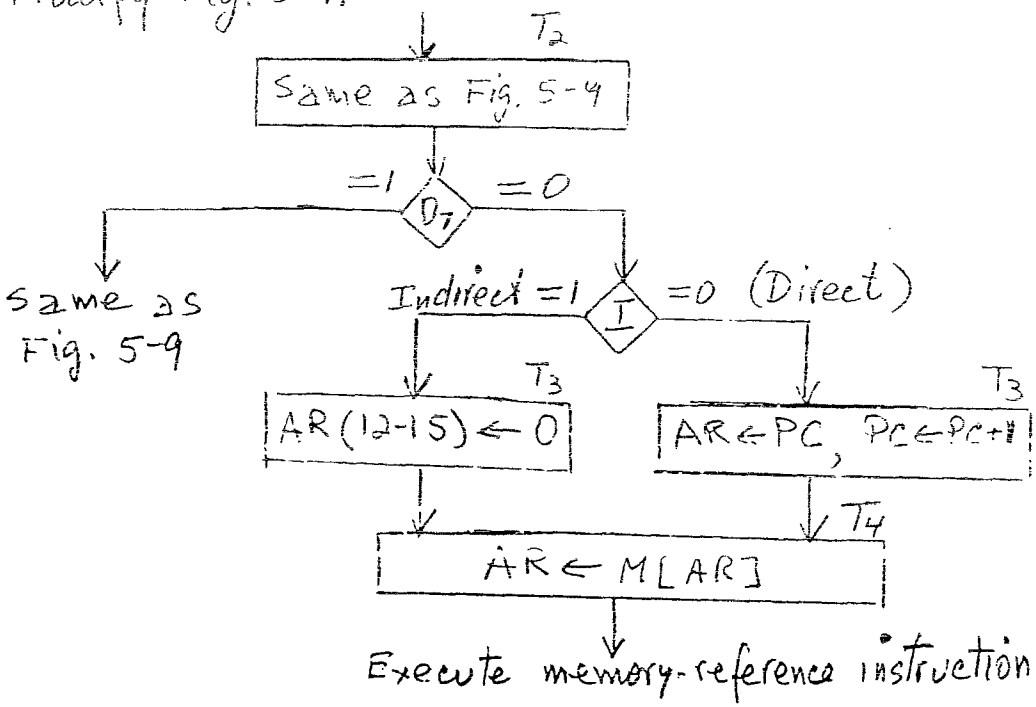
5-13

XOR	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \oplus DR$, $SC \leftarrow 0$
ADM	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$DR \leftarrow AC$, $AC \leftarrow AC + DR$
	D_1T_6:	$M[AR] \leftarrow AC$, $AC \leftarrow DR$, $SC \leftarrow 0$
SUB	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$DR \leftarrow AC$, $AC \leftarrow DR$
	D_2T_6:	$AC \leftarrow \overline{AC}$
	D_2T_7:	$AC \leftarrow AC + 1$
	D_2T_8:	$AC \leftarrow AC + DR$, $SC \leftarrow 0$
XCH	D_3T_4:	$DR \leftarrow M[AR]$
	D_3T_5:	$M[AR] \leftarrow AC$, $AC \leftarrow DR$, $SC \leftarrow 0$
SEQ	D_4T_4:	$DR \leftarrow M[AR]$
	D_4T_5:	$TR \leftarrow AC$, $AC \leftarrow AC \oplus DR$
	D_4T_6:	If $(AC = 0)$ then $(PC \leftarrow PC + 1)$, $AC \leftarrow TR$, $SC \leftarrow 0$
BPA	D_5T_4:	If $(AC = 0 \wedge AC(15) = 0)$ then $(PC \leftarrow AR)$, $SC \leftarrow 0$

5-14

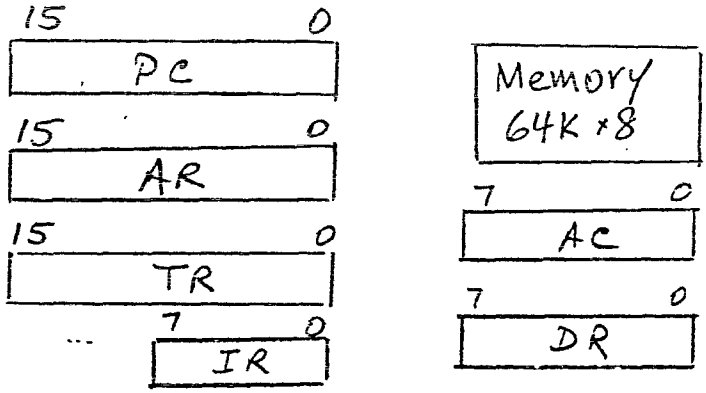
Converts the ISZ instruction from a memory-reference instruction to a register-reference instruction. The new instruction ICSZ can be executed at time T_3 instead of time T_6, a saving of 3 clock cycles.

5-15 Modify Fig. 5-4:

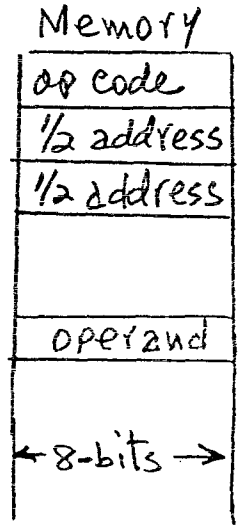


5-16

(a)



(b)



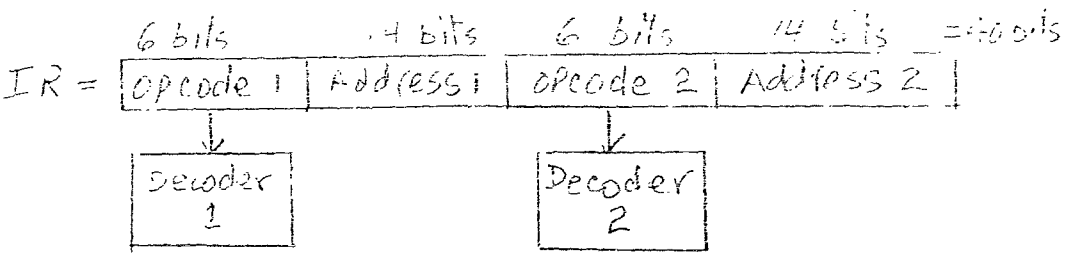
(c) T_0: IR ← M[PC], PC ← PC+1

T_1: AR(0-7) ← M[PC], PC ← PC+1

T_2: AR(8-15) ← M[PC], PC ← PC+1

T_3: DR ← M[AR]

5-11



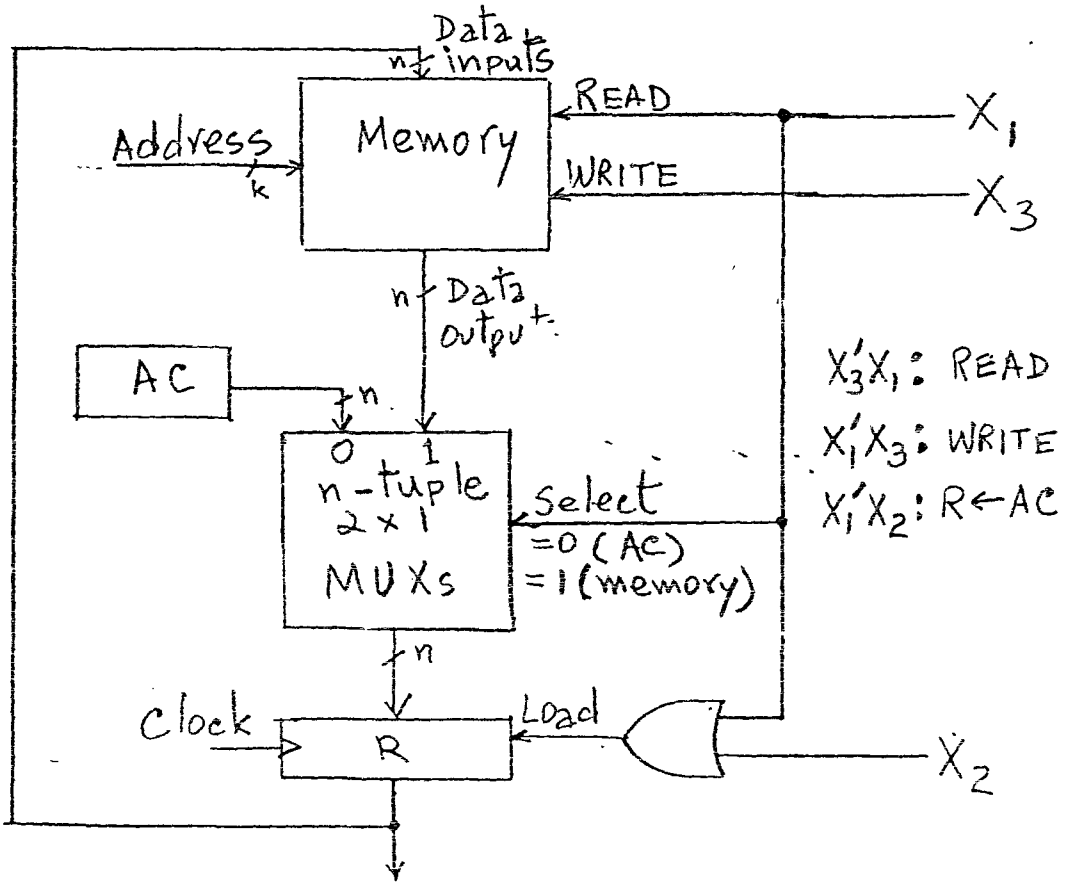
1. Read 40-bit double instruction from memory to IR and then increment PC,
2. Decode opcode 1.
3. Execute instruction 1 using address 1.
4. Decode opcode 2.
5. Execute instruction 2 using address 2.
6. Go back to step 1.

5-18

(a) BUN 2300

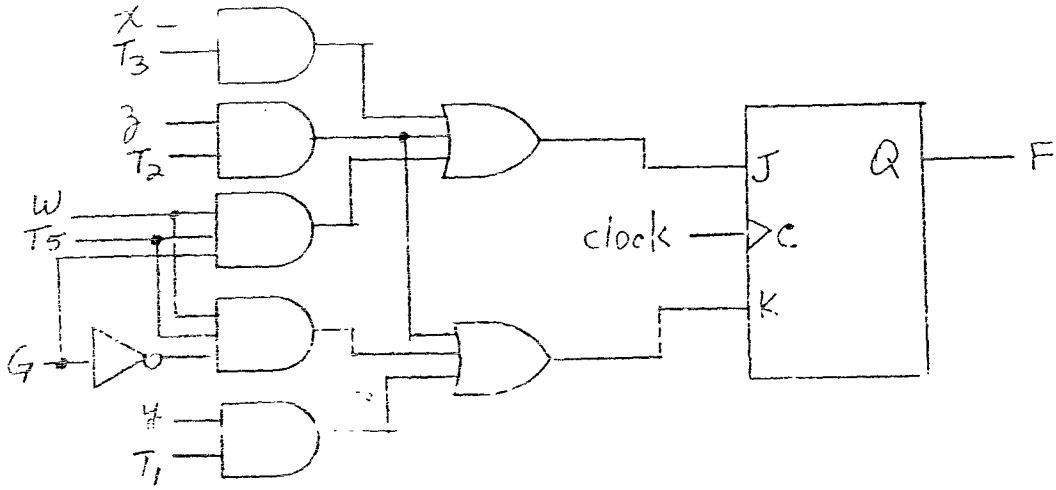
(b) ION
BUN 0 I (Branch indirect with address 0)

5-19



5-20

$J_F = xT_3 + zT_2 + wT_5G \qquad K_F = yT_1 + zT_2 + wT_5G'$



5-21 From Table 5-6: ($Z_{DR}=1$ if DR=0 ; $Z_{AC}=1$ if AC=0)

$$INR(PC) = R'T_1 + RT_2 + D_6T_6Z_{DR} + pB_9(FGI) + pB_8(FGO) + rB_4(AC_{15})' + rB_3(AC_{15}) + rB_2Z_{AC} + rB_1E'$$

$$LD(PC) = D_4T_4 + D_5T_5$$

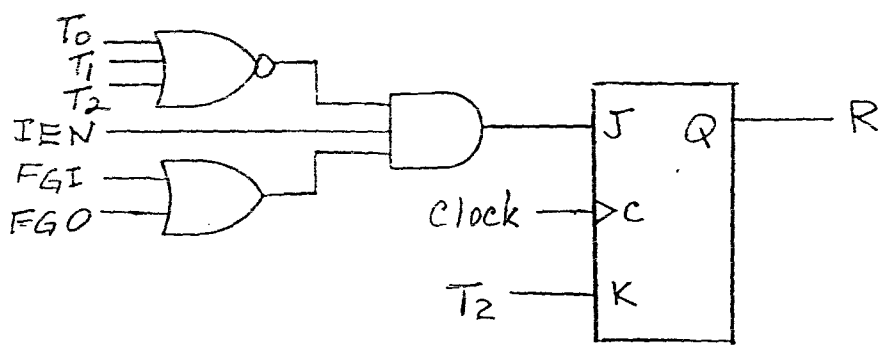
$$CLR(PC) = RT_1$$

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

5-22 $Write = D_3T_4 + D_5T_4 + D_6T_6 + RT_1$ (M[AR] ← xx)

5-23 $(T_0+T_1+T_2)'(IEN)(FGI+FGO)$: R ← 1

RT_2 : R ← 0



5-24

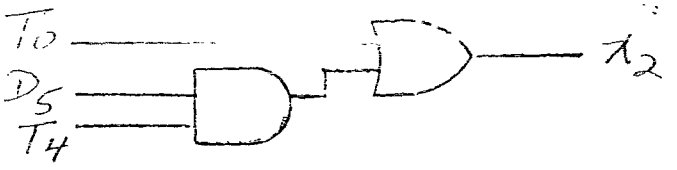
x_2 places PC onto the bus. From Table 5-6:

$R'T_0$: AR ← PC

RT_0 : TR ← PC

D_5T_4 : M[AR] ← PC

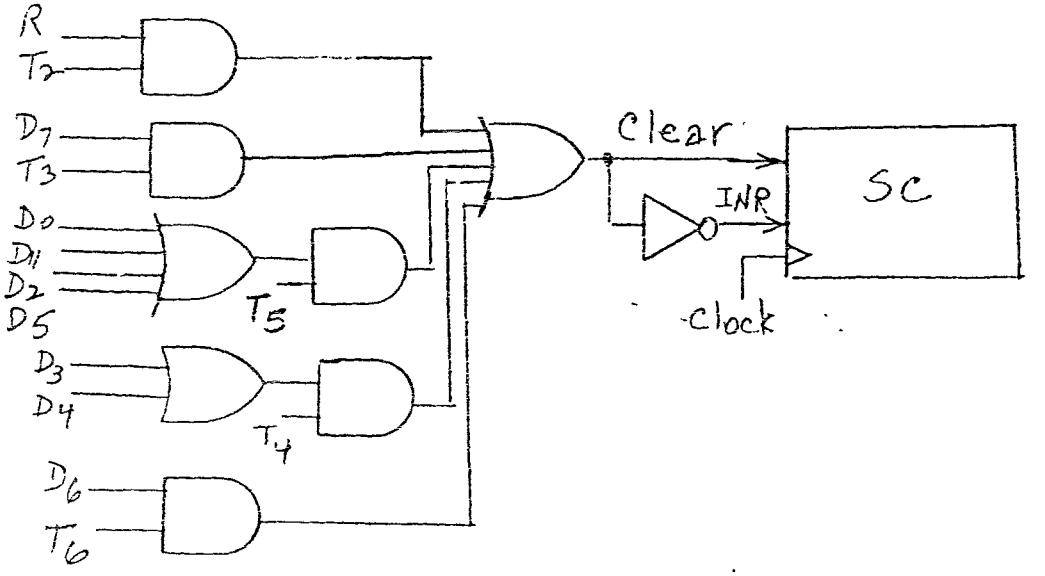
$$x_2 = R'T_0 + RT_0 + D_5T_4 = (R'+R)T_0 + D_5T_4 = T_0 + D_5T_4$$



5-25

From Table 5-6:

$$CLR(SC) = RT_2 + D_7T_3(I'+I) + (D_0+D_1+D_2+D_5)T_5 + (D_3+D_4)T_4 + D_6T_6$$



CHAPTER 6

6-1

```
010   CLA
011   ADD 016
012   BUN 014
013   HLT
014   AND 017
015   BUN 013
016   C1A5
017   93C6
```

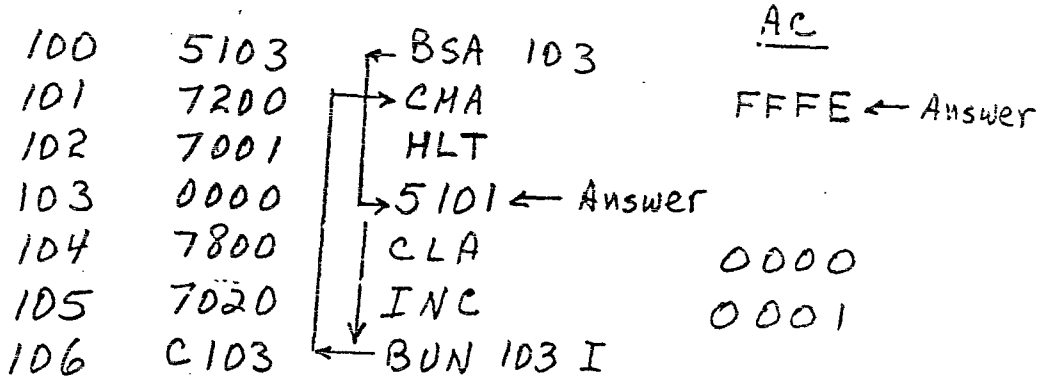
AC	PC	IR
0000	011	7800
C1A5	012	1016
C1A5	014	4014
8184	014	7001
8184	015	0017
8184	013	4013

$(C1A5)_{16} = 1100\ 0001\ 1010\ 0101$

$(93C6)_{16} = 1001\ 0011\ 1100\ 0110$ AND

$1000\ 0001\ 1000\ 0100 = (8184)_{16}$

6-2



6-3

```
CLA
STA SUM      } SUM = 0

LDA SUM
ADD A
ADD B        } SUM = SUM + A + B
STA SUM

LDA C
CMA
INC          } DIF = DIF − C
ADD DIF
STA DIF

LDA SUM
ADD DIF      } SUM = SUM + DIF
STA SUM
```

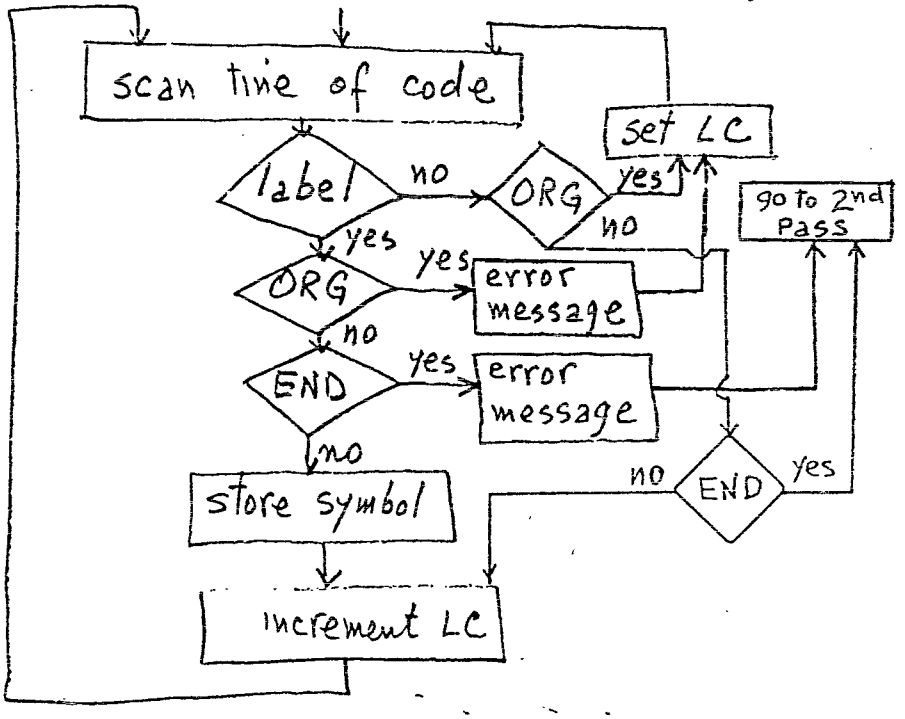
A more efficient compiler will optimize the machine code as follows:

```
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
ADS	10B
PTR	10C
NBR	10D
CTR	10E
SUM	10F

$(100)_{10} = (0000\ 0000\ 0110\ 0100)_2$

$(-100)_{10} = (1111\ 1111\ 1001\ 1100)_2 = (FF9C)_{16}$

$(75)_{10} = (0000\ 0000\ 0100\ 1011)_2 = (004B)_{16}$

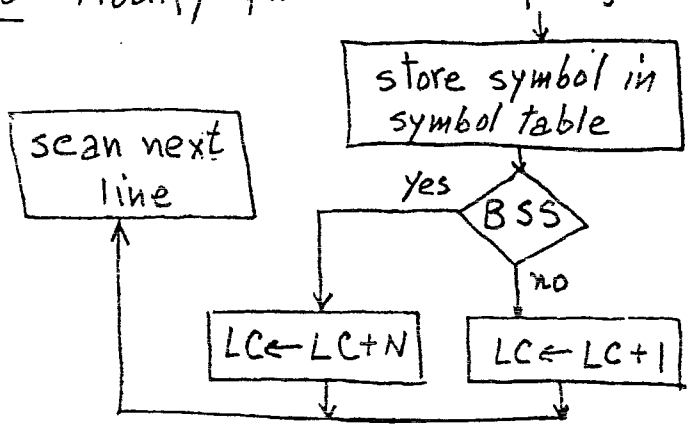
$(23)_{10} = (0000\ 0000\ 0001\ 0111)_2 = (0017)_{16}$

(b)

Loc	Hex			
			ORG	100
100	210B		LDA	ADS
101	310C		STA	PTR
102	210D		LDA	NBR
103	310E		STA	CTR
104	7800		CLA	
105	910C	LOP,	ADD	PTR I
106	610C		ISZ	PTR
107	610E		ISZ	CTR
108	4105		BUN	LOP
109	310F		STA	SUM
10A	7001		HLT	

Loc	Hex		
10B	0150	ADS,	HEX 150
10C	0000	PTR,	HEX 0
10D	FF9C	NBR,	DEC -100
10E	0000	CTR,	HEX 0
10F	0000	SUM,	HEX 0
			ORG 150
150	004B		DEC 150
⋮	⋮		⋮
1B3	0017		DEC 23
			END

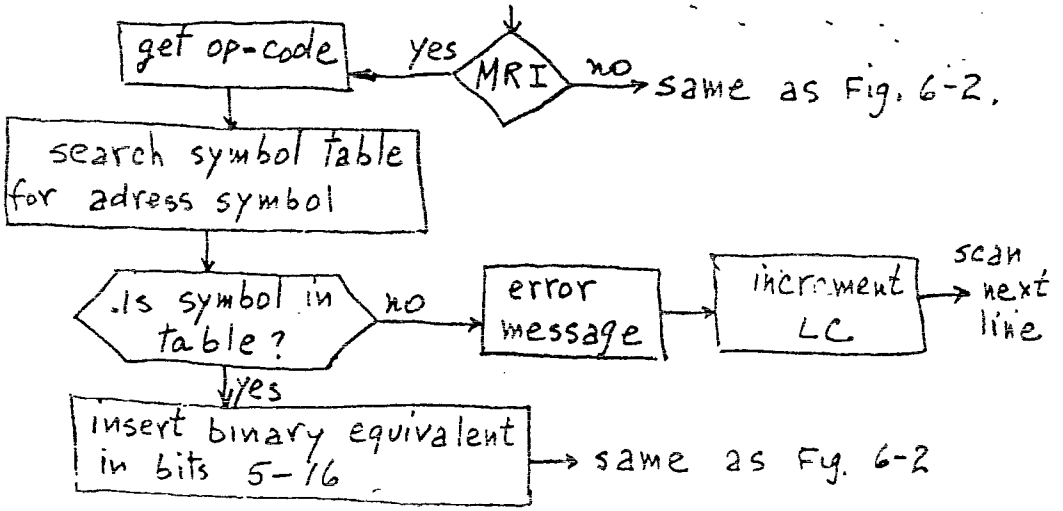
6-8 Modify flow chart of Fig. 6-1




```
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	0000
ADD	4	A D	41 44
	5	D space	44 20
	6	Value	1000

etc.

(b) non-MRI table →

	word	symbol	HEX
CLA	1	C L	43 4C
	2	A space	41 20
	3	Value	7800
CLE	4	C L	43 4C
	5	E space	45 20
	6	Value	7400

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
```

6-12 (b) Continued

```
10D   7004   SZA       /check if remaining bits = 0
10E   4107   BUN ROT   / No; rotate again
10F   7001   STP, HLT  / Yes; stop
110   0000   CTR, HEX 0
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  0
NBR,  DEC  -256
CTR,  HEX  0
      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  0      / 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
```

same as beginning of Program in Table 6-14

```
ONE,  LDA   XL
      ADD   PL
      STA   PL
      CLA
      CIL
      ADD   XH
      ADD   PH
      STA   PH
      CLE
```

Double-precision add

$P \leftarrow X + P$

Same as program in Table 6-15

Continued next page

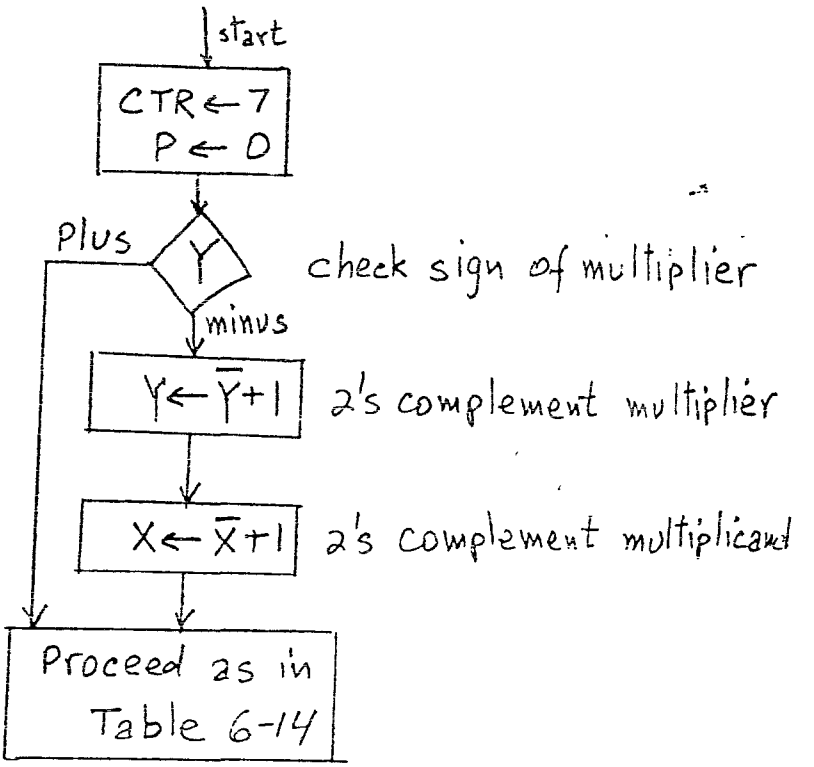
6-16 continued

```
ZRO,    LDA   XL      ┐
        CIL           │  Double-precision
        STA   XL      │  left-shift XH+XL
        LDA   XH      │
        CIL           │
        STA   XH      ┘
        ISZ   CTR     ┐
        BUN   LOP     │  Repeat 16 times
        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:



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 once.

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

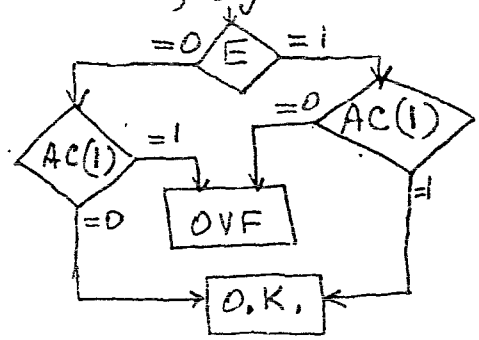
Location TMP saves the carry from E while BH is complemented.

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

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

6-20

```
        LDA  X
        CLE
        CIL     / zero to low order bit; sign bit in E
        SZE
        BUN  ONE
        SPA
        BUN  OVF
        BUN  EXT
ONE,    SNA
        BUN  OVF
EXT,    HLT
```



6-21 Calling program

```
BSA   SUB
HEX   1234   /subtrahend
HEX   4321   /minuend
HEX   0      /difference
```

Subroutine

```
SUB,  HEX  0
      LDA  SUB I
      CMA
      INC
      ISZ  SUB
      ADD  SUB I
      ISZ  SUB
      STA  SUB I
      ISZ  SUB
      BUN  SUB I
```

6-22

Calling Program

```
BSA   CMP
HEX   100   /starting address
DEC   32    /number of words
```

Subroutine

```
CMP,  HEX  0
      LDA  CMP I
      STA  PTR
      ISZ  CMP
      LDA  CMP I
      CMA
      INC
      STA  CTR
LOP,  LDA  PTR I
      CMA
      STA  PTR I
      ISZ  PTR
      ISZ  CTR
      BUN  LOP
      ISZ  CMP
      BUN  CMP I
PTR,  —
CTR,  —
```

6-23

```
CR4,  HEX  0
      CIR
      CIR
      CIR
      CIR
      BUN  CR4 I
```

	AC	AC HEX
E		
1	0000 0111 1001 1100	079C
1	1001 0000 0111 1001	9079

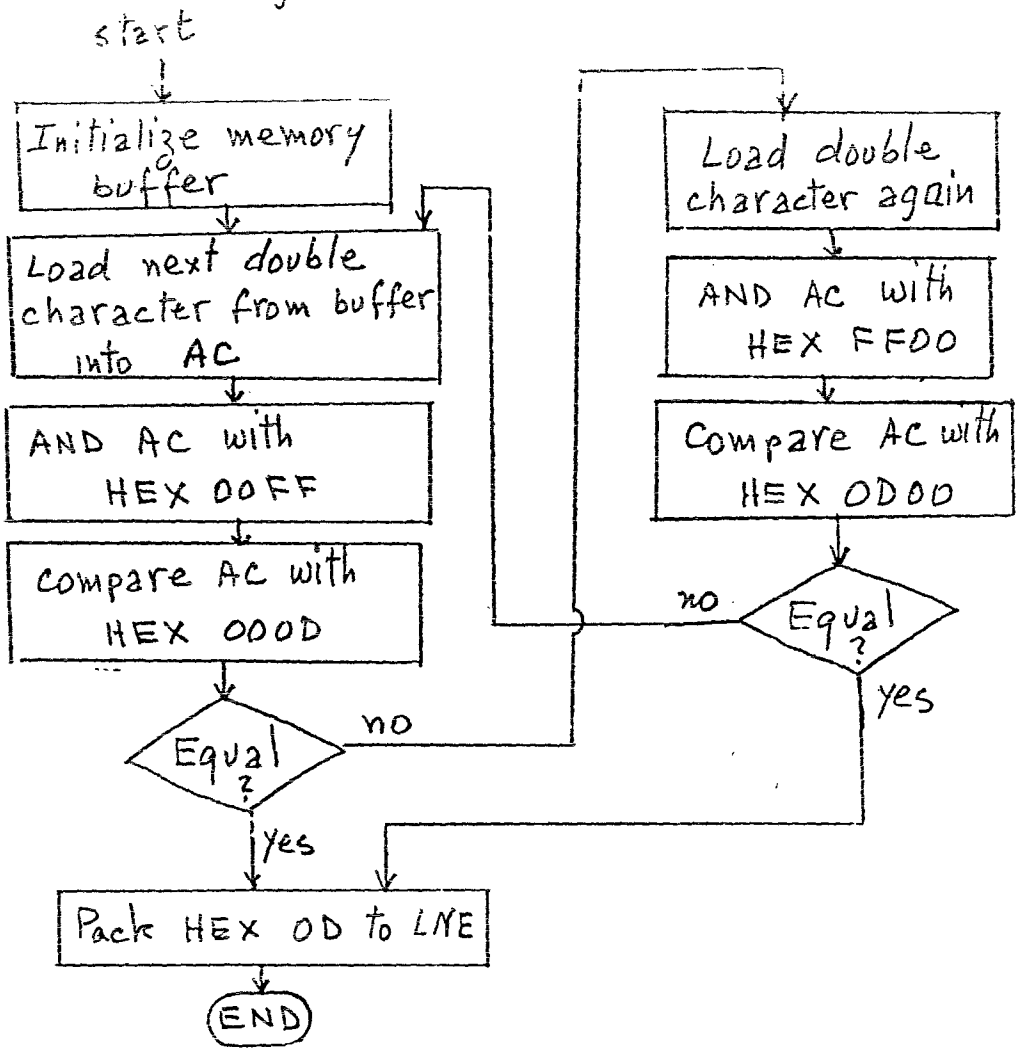
6-24

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

6-25

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

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
      CIR
      STA  SE
      LDA  MOD   /check MOD
      CMA
      SZA
      BUN  NXT   / MOD≠ all 1's
      SKI                  ┐
      BUN  NXT             │ service
      INP                  │ input
      OUT                  │ Device
      STA  PTI I           │
      ISZ  PTI             ┘
      BUN  EXT   / MOD≠0

NXT,  LDA  MOD
      SZA
      BUN  EXT
      SKO                  ┐ service
      BUN  EXT             │ output
      LDA  PT2 I           │ device
      OUT                  │
      ISZ  PT2             ┘
EXT,  continue as
      in Table 6-23
```

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.

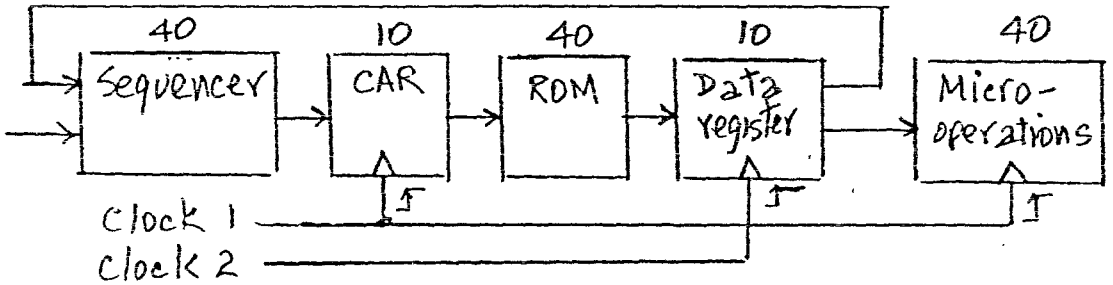
7-2

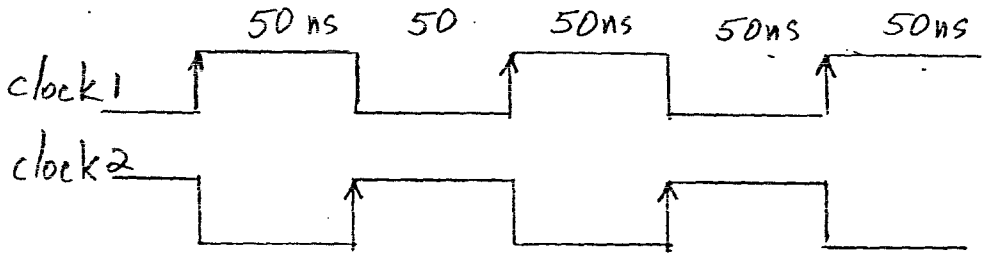
Hardwired control, by definition, does not contain a control memory.

7-3

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

7-4





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

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

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

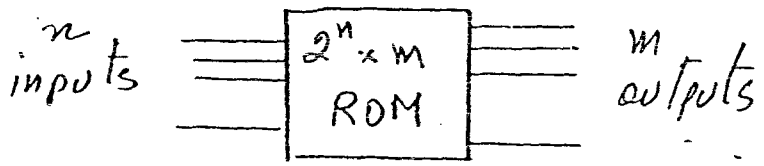
(c) 0111100 = 60

7-8 op code = 6 bits

control memory address = 11 bits

00 | xxxxxx | 000

7-9



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.

7-11

	F1	F2	F3			
(a)	011	110	000	INCAC	INCDR	NOP
(b)	000	100	101	NOP	READ	INCPC
(c)	100	101	000	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. Both use F1
	WRITE	M[AR] ← DR	F1 = 111	

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 (NEXT)
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)

7-16 (CONTINUED)

```
                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 fetch INDRCT)
	DRTAC, ACTDR	Z	JMP	ZERO
	WRITE	U	JMP	FETCH
ZERO:	WRITE, INCPC	U	JMP	FETCH

7-18

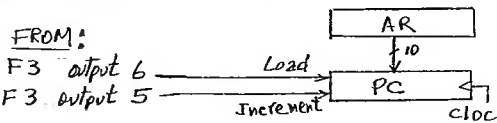
BSA:	NOP	I	CALL	INDRCT
	PCTDR, ARTPC	U	JMP	NEXT
	WRITE, INCPC	U	JMP	FETCH

7-19

From Table 7-1:

F3 = 101 (5) PC ← PC+1

F3 = 110 (6) PC ← AR



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.

(2) 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

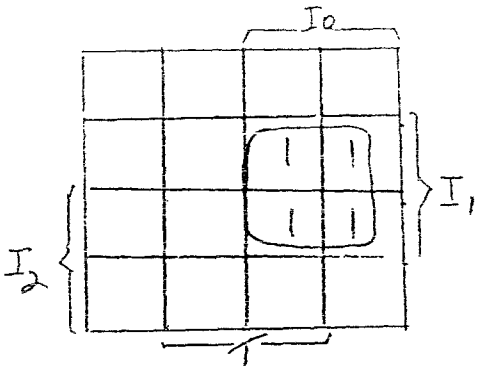
(c)

SRC1 (R5)	SRC2 (R6)	DEST (R4)	ALU (ADD)	SHIFT
0101	0110	0100	00100	000

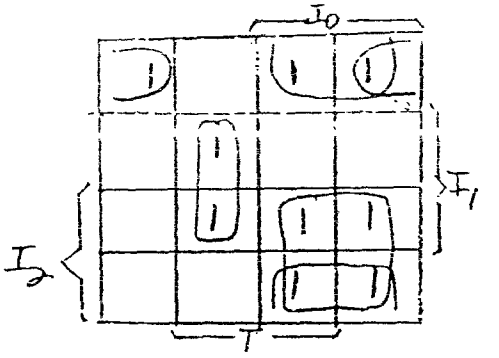
R4 ← R5 + R6

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)



$S_0 = I_1 I_0$



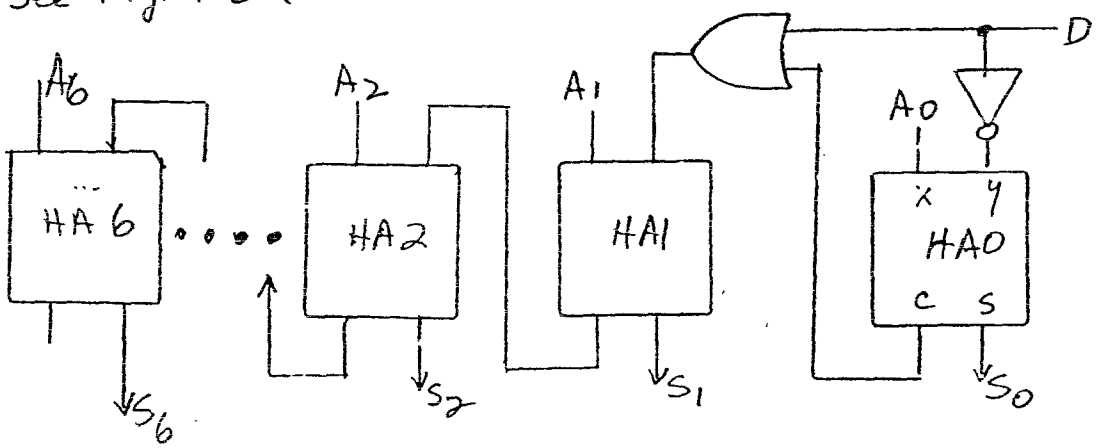
$S_0 = I_2 I_0 + I_2' I_0 + I_1 I_0' T + I_2' I_1' T'$

$L = I_2 I_1 I_0' T$

7-23

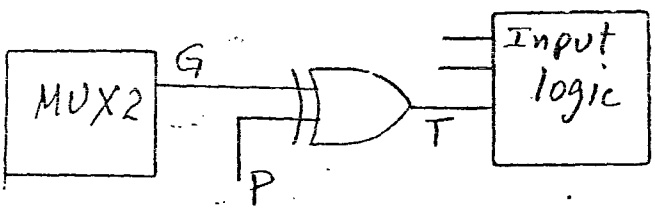
(a) See Fig. 4-8 (chapter 4)

(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 $G \oplus 1 = G'$

Wher G is the value of the setected bit in MUX2

CHAPTER 8

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
SELA	SELB	SELD	OPR

= 18 bits

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

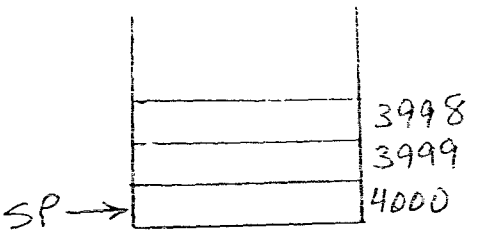
(a) Stack full with 64 items.

(b) Stack empty.

8-6

PUSH : $M[SP] \leftarrow DR$
$SP \leftarrow SP-1$

POP : $SP \leftarrow SP+1$
$DR \leftarrow M[SP]$



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

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

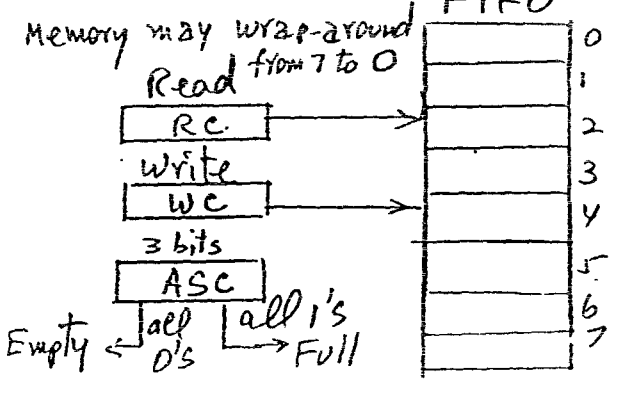
RPN: 3 4 + 2 6 + 10 * 8 + *

				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	+	*

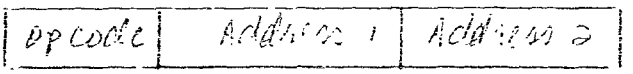
8-10

WRITE (if not full):
$M[WC] \leftarrow DR$
$WC \leftarrow WC+1$
$ASC \leftarrow ASC+1$

READ: (if not empty)
$DR \leftarrow M[RC]$
$RC \leftarrow RC+1$
$ASC \leftarrow ASC-1$



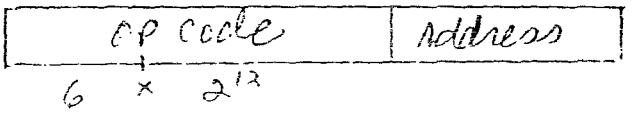
8-11



Two address instructions

$2^8 = 256$ combinations.

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



one address instructions

Maximum number of one address instruction:

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

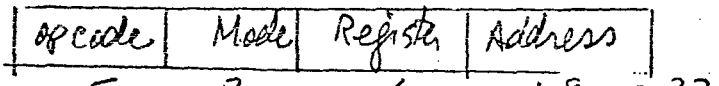
8-12

(d) RPN: XAB−C+DE*F−*GHK*+/=

8-13

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

address = 18 bits
Mode = 3 "
Register = 6 "
27 bits
op code 5
32 bits



8-14

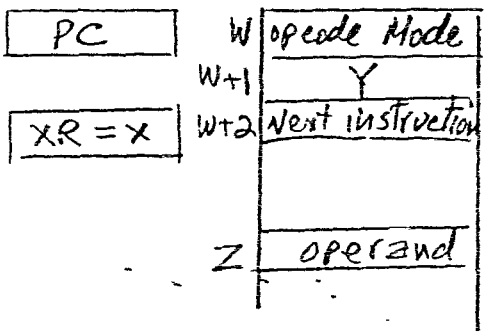
Z = Effective address

(a) Direct: Z = Y

(b) Indirect: Z = M[Y]

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

(d) Indexed: Z = Y + X



8-15

(a) Relative address = 500 − 751 = −251

(b) 251 = 000011111011; −251 = 111100000101

(c) PC = 751 = 001011101111; 500 = 000111110100

PC = 751 = 001011101111
RA = −251 = +111100000101
EA = 500 = 000111110100

8-16 Assuming one word per instruction or operand.

Computational type	Branch type
Fetch instruction	Fetch instruction
Fetch effective address	Fetch effective address and transfer to PC
Fetch operand	
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

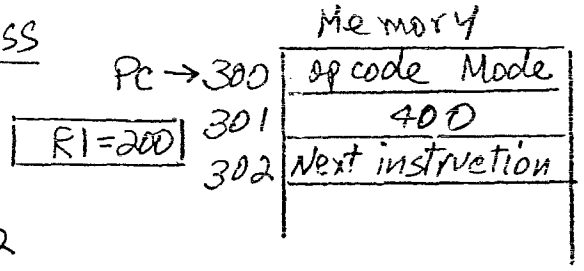
(a) Direct : 400

(b) Immediate : 301

(c) Relative : 302+400 = 702

(d) Reg. Indirect : 200

(e) Indexed : 200+400 = 600



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	10011100	10011100
AND 10101010	OR 10101010	XOR 10101010
10001000	11111110	00110110

8-21

(a) AND with : 0000000011111111

(b) OR with : 0000000011111111

(c) XOR with : 0000111111110000

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 + 68 = −15: 10101101 + 01000100 = 11110001 (in 2's complement)

(b) −68 + (−83) = −151 (overflow): 10111100 + 10101101 = 01101001 (carries: 10); −151 ≠ −128

(c) −68 = 10111100, −34 = 11011110

(d) −83 = 10101101, −166 ≠ 01011010 (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) 72 + C6 = 138: 01110010 + 11000110 = 00111000
C=1 S=0 Z=0 V=0

(b) 72 + 1E = 90: 01110010 + 00011110 = 10010000
C=0 S=1 Z=0 V=1

(c) 9A = 10011010 → 2's comp. 01100110
72 − 9A = D8: 01110010 + 01100110 = 11011000
C=0 S=1 Z=0 V=1
(Borrow = 1)

(d) 72 + 8D = 00: 01110010 + 10001100 = 00000000
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 $A \neq B$

or $C=0$ and $Z=0$ $\quad (C'Z')=1$

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

or $C=1$ or $Z=1$ $\quad (C+Z)=1$

8-27

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

sign $S=0$ if no overflow (positive)

or $S=1$ if overflow (sign reversal)

Boolean expression: $S'V'+SV=1$ or $(S \oplus V)=0$

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

then $S=1$ if $V=0$

or $S=0$ if $V=1$ $\quad (S \oplus V)=1$

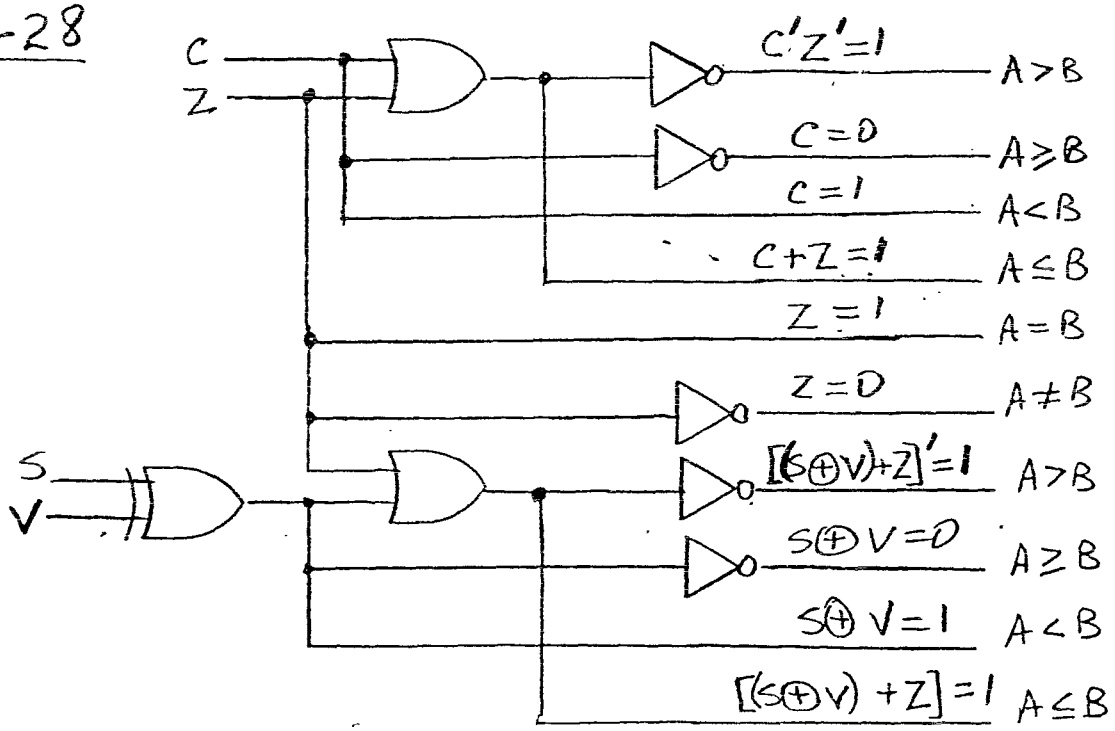
$A>B$ Implies $A \geq B$ but not $A=B$

$(S \oplus V)=0$ and $Z=0$

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

$S \oplus V=1$ or $Z=1$

8-28



8-29

	Unsigned	Signed
A = 01000001	65	+65
B = 10000100	132	-124
A+B = 11000101	197	-59

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

(d) BNC BNZ BM 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 65 < 132, A < B

(c) BL, BLE, BNE

8-31

(a) A = 01000001 = +65

B = 10000100 = -124

A-B = 10111101 +189 = +01011101 (9 bits)

(b) S = 1 (sign reveral) +189 > 127

Z = 0

V = 1 (overflow) 65 > -124, 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 ← SP-1
M[SP] ← PSW
SP ← SP-1
M[SP] ← PC
TR ← IAD (TR is a temporary register)
PSW ← M[TR]
TR ← TR+1
PC ← M[TR]
Go to fetch phase.

(b)
PC ← M[SP]
SP ← SP+1
PSW ← M[SP]
SP ← 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 ← R22 − 1 (Subtract 1)

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

(c) SUB R0, R22, R22 — R22 ← 0 − R22

(d) ADD R0, R0, R22 — R22 ← 0 + 0

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

(f) OR R1, R1, R1 — R1 ← R1 V R1

or ADD R1, R0, R1 — R1 ← R1 + 0

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

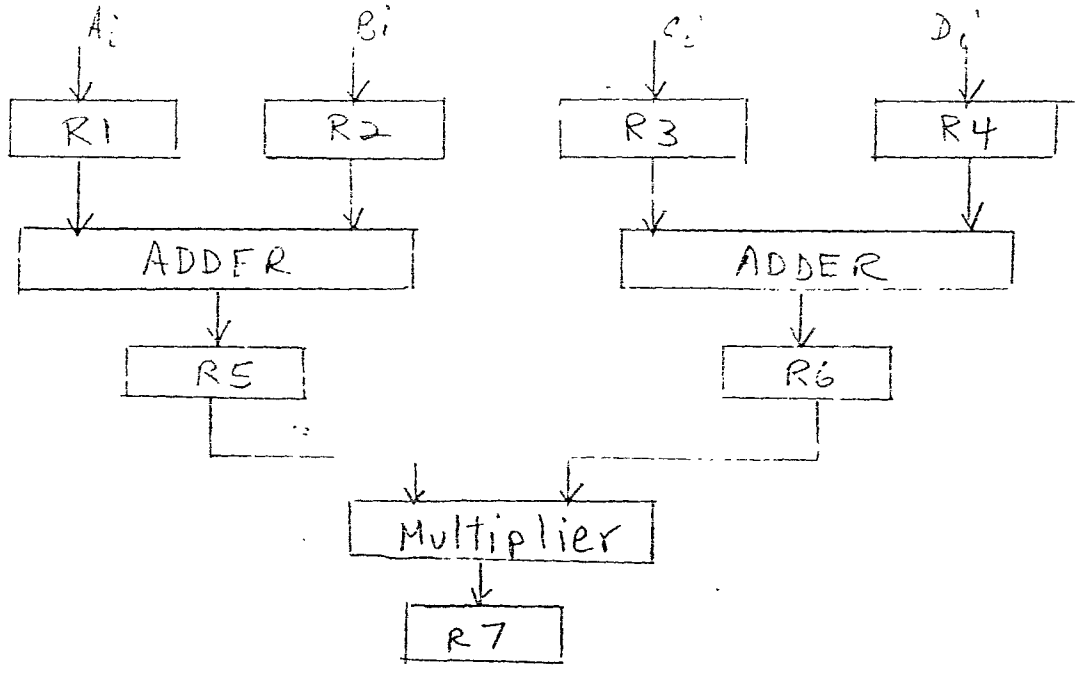
8-39

(a) JMP Z, #3200, (R0) — PC ← 0 + 3200

(b) JMPR Z, -200 — PC ← 3400 + (-200)

CHAPTER 9

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

$(k-n-1)t_p = 6+8-1 = 13$ cycles

9-3

$k = 6$ segment
$n = 200$ tasks

$(k+n-1) = 6+200-1 = 205$ cycles

9-4

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

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

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

9-5

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

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

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

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

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

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 $= \frac{k+n-1}{n} t_p = \frac{(n+6)30}{n}$

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 $= (k+n-1) t_p = (4+99)100$
$= 10,300$ 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; $k = 5$
$(k+n-1) t_p = (5+99)55 = 5,720$ ns $= 5.72\,\mu$s

9-8 Connect output of adder to input $B \times 2^b$ 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: (a) 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 ← M[312]
2. Add R2 ← R2 + M[313]
3. Increment R3
4. Store M[314] ← 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 ← Memory
Increment: R1 ← R1 + 1

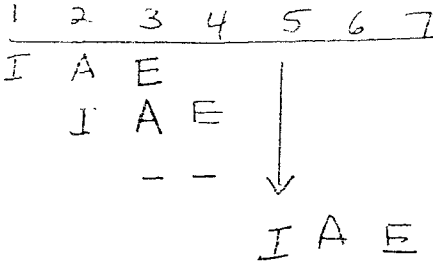
1	2	3	4
I	A	E	
	I	A	E

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).

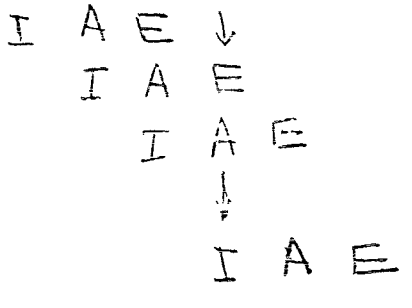
9-14

101 Add R2 to R3
102 Branch to 104
103 Increment R1
104 Store R1



9-15 Use example of Problem 9-14.

101 Branch to 105
102 Add R2 to R3
103 No-operation
104 Increment R1
105 Store R1



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.

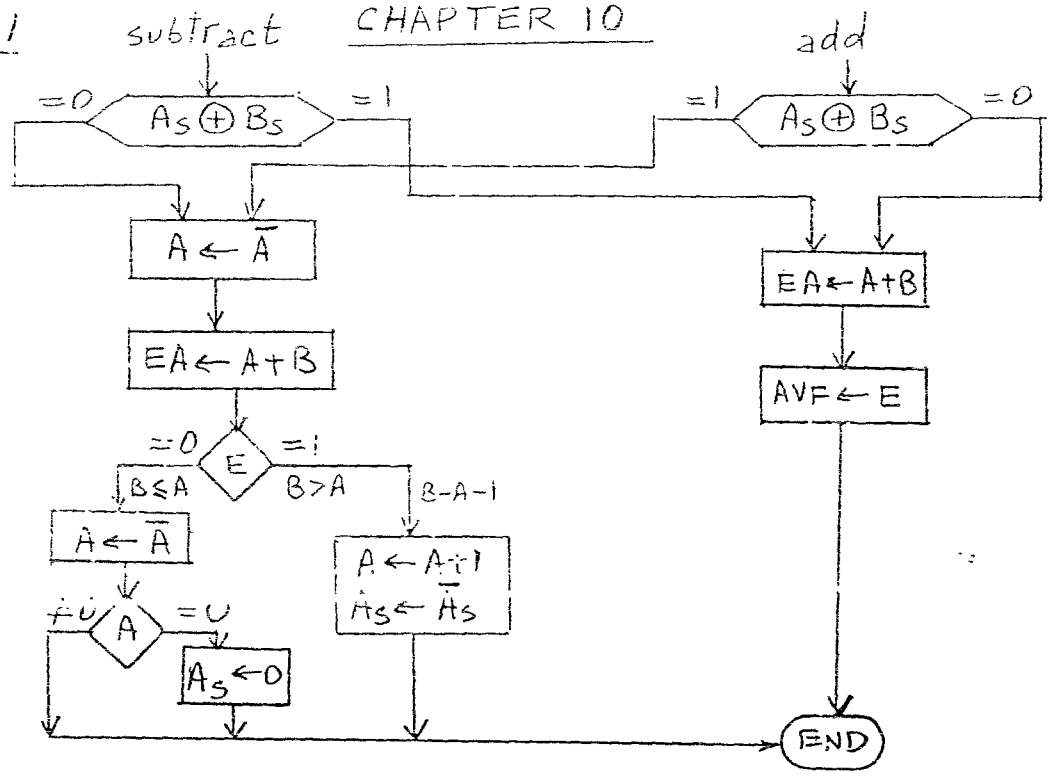
9-18 memory array 1 use addresses: 0, 4, 8, 12, ..., 1020. Array 2: 1, 5, 9, 13, ..., 1021; Array 3: 2, 6, 10, ..., 1022. Array 4: 3, 7, 11, ..., 1023.

9-19 $\frac{250 \times 10^9}{100 \times 10^6} = 2,500$ sec $= 41.67$ minutes

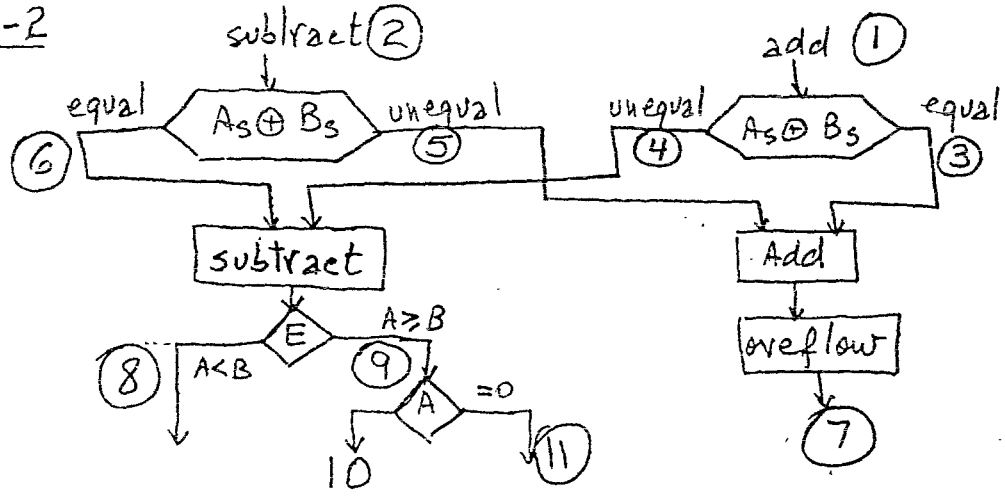
9-20 Divide the 400 operations into each of the four processors. Processing time is: $\frac{400}{4} \times 40 = 4,000$ nsec

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
+40 0 101000
+75 1 001011
F=0 E=1

F⊕E = 1 ; overflow

(b) −35 1 011101
−40 1 011000
−70 0 110101
F=1 E=0

Carries →

F⊕E = 1 ; overflow

10-4

case	(a) operation in sign-magnitude	(b) operation in sign-2's complement	(c) 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)

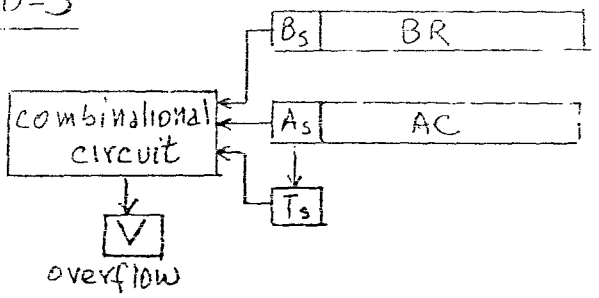
If $X<Y$ then $(Y-X)>0$. Operation gives $2^k+2^k-(Y-X)$ as in colum (c).

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



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

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	

Boolean function for circuit:

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

10-6 (a)

$$\begin{array}{rll} -9 & 1 & 0110 \\ -6 & 1 & 1001 \\ \hline -15 & 0 & 1111 \end{array}$$

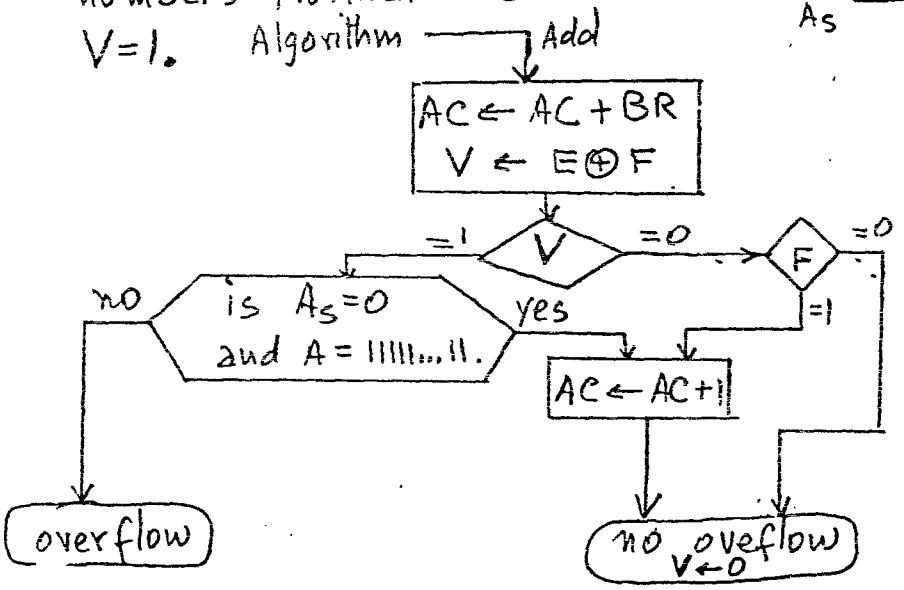
F=1 E=0 ← Carries

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

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

$$\begin{array}{r} 0\ 1111 \\ +\ 1 \\ \hline 1\ 0000 \end{array} = -15$$

(b) The procedure $V \leftarrow E \oplus F$ is valid for 1's complement numbers provided we check the result $0\ \underbrace{1111\ldots11}_{A}$ when A_s $V=1$. Algorithm → Add



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 maximumum 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 $B = 11111 = (31)_{10}$ $31 \times 21 = 651$

Multiplier in Q — $Q = (21)_{10}$

	E	A	Q	SC
	0	00000	10101	101
Q_n=1, add B		11111		
	0	11111		
shr EAQ		01111	11010	100
Q_n=0, shr EAQ		00111	11101	011
Q_n=1, add B		11111		
	1	00110		
shr EAQ	0	10011	01110	010
Q_n=0, shr EAQ		01001	10111	001
Q_n=1, add B		11111		
	1	01000		
shr EAQ		10100	01011	000

$(651)_{10}$

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

$B = 1011$ $\overline{B}+1 = 0101$ 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		
E=1, set Q_n to 1	1	1001	0111	011
shl EAQ	1	0010	1110	
add $\overline{B}+1$, suppress carry		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	1001	1110	000

remainder quotient

10-10(b) $\frac{1111}{0011} = 0101$ $\quad B = 0011$ $\quad \overline{B}+1 = 1101$

	E	A	Q	SC
Dividend in Q, A=0		0000	1111	100
shl EAQ	0	0001	1110	
add $\overline{B}+1$		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 $\overline{B}+1$		1101		
E=1, set Q_n to 1	1	0000	1101	010
shl EAQ	0	0001	1010	
add $\overline{B}+1$		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 $\overline{B}+1$		1101		
E=1, set Q_n to 1	1	0000	0101	000
		remainder	quotient	

10-11

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

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

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

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.

10-14 (a) $(+15) \times (+13) = +195 = (0\ 0111\ 0000\ 11)_2$

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			
			11010			
		ashr	11010	01101	1	010
1	1	ashr	11101	00110	1	001
0	1	Add BR	01111			
			01100			
		ashr	00110	00011	0	000

$+195$

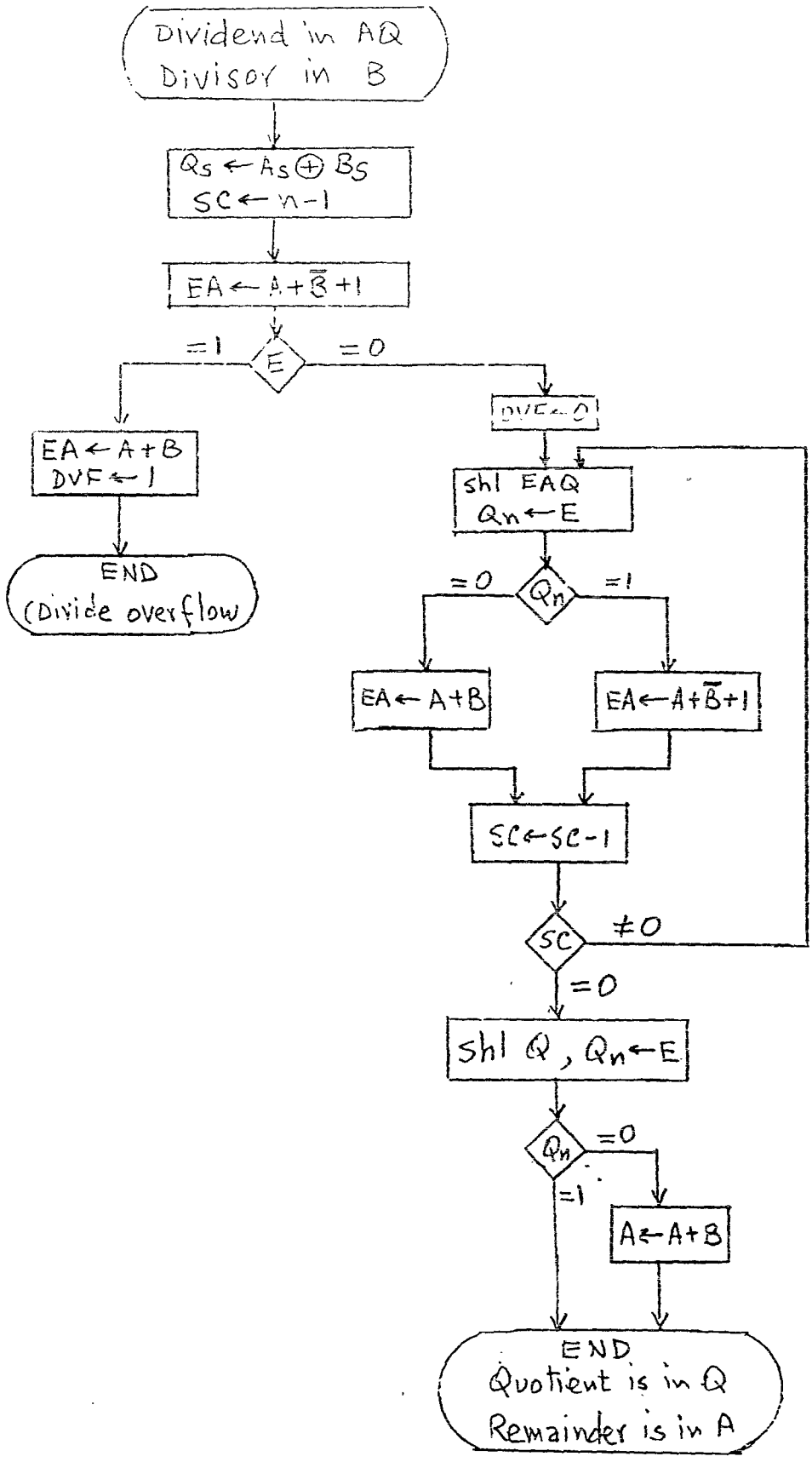
(b) $(+15) \times (-13) = -195 = (1100\ 1111\ 01)_{2\text{'s comp.}}$

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

10-15


Dividend in AQ
Divisor in B
$Q_S \leftarrow A_S \oplus B_S$
$SC \leftarrow n-1$
$EA \leftarrow A + \overline{B} + 1$
E
=1
=0
EA ← A+B
DVF ← 1
END
(Divide overflow
DVF ← 0
shl EAQ
$Q_n \leftarrow E$
Q_n
=0
=1
EA ← A+B
EA ← A+B̄+1
SC ← SC-1
SC
≠0
=0
shl Q, $Q_n \leftarrow E$
Q_n
=0
=1
A ← 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 comments:

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 $0q_101$
The third test value is $00q_1q_201$
The fourth test value is $000q_1q_2q_301$ 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 11 01           subtract third test value 00q1q201
  ---------
    negative          answer negative; let q3 = 0
   00 01 10           restore partial remainder
   00 01 10 01        bring down next pair
   00 01 10 01        subtract fourth test value 000q1q2q301
  ------------
Remainder = 00000     answer positive (zero); let q4 = 1
```

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.

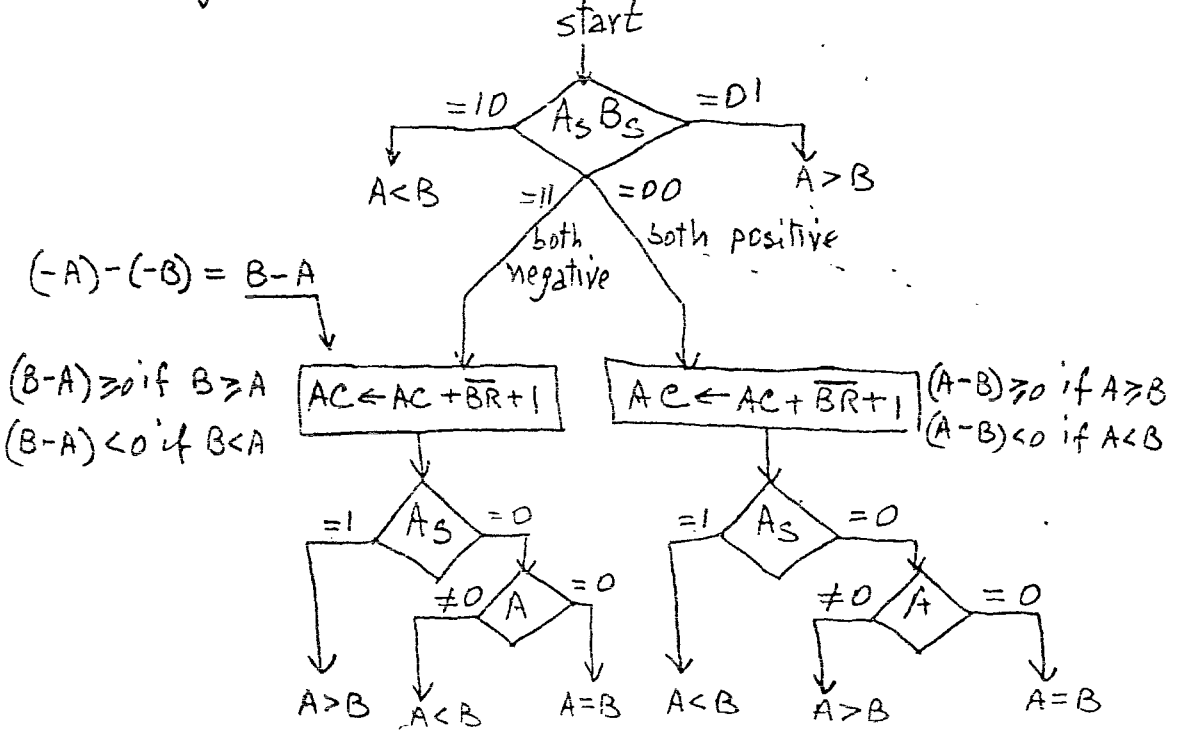
10-18

(a) $AC = A_S\ A_1\ A_2\ A_3\ \ldots\ A_n$

$BS = B_S\ B_1\ B_2\ B_3\ \ldots\ B_n$

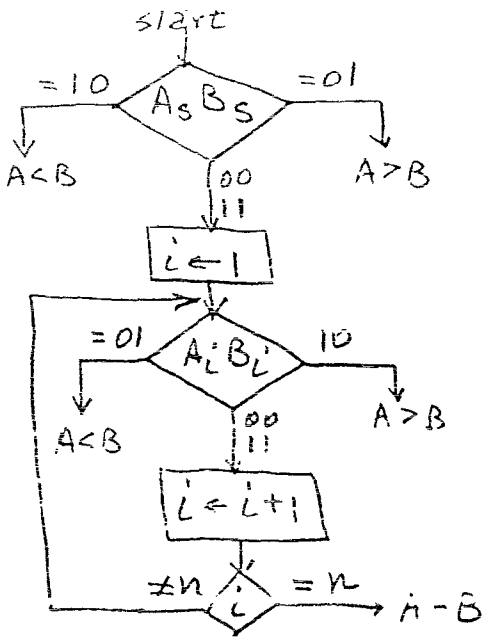
If signs are unlike – the one with a 0 (plus) is larger.

If signs are alike – both numbers are either positive or negative



10-18 (6)

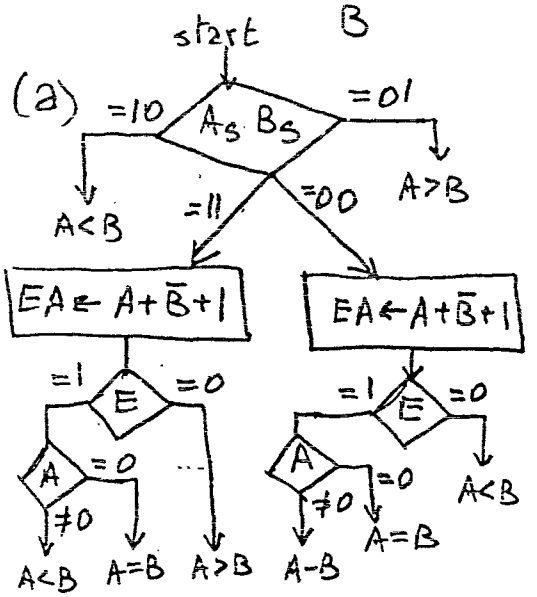
	A_S	A_1	A_2	...	A_n
+2	0	0	0	0	10
+1	0	0	0	0	01
0	0	0	0	0	00
−1	1	1	1	1	11
−2	1	1	1	1	10
−3	1	1	1	1	01

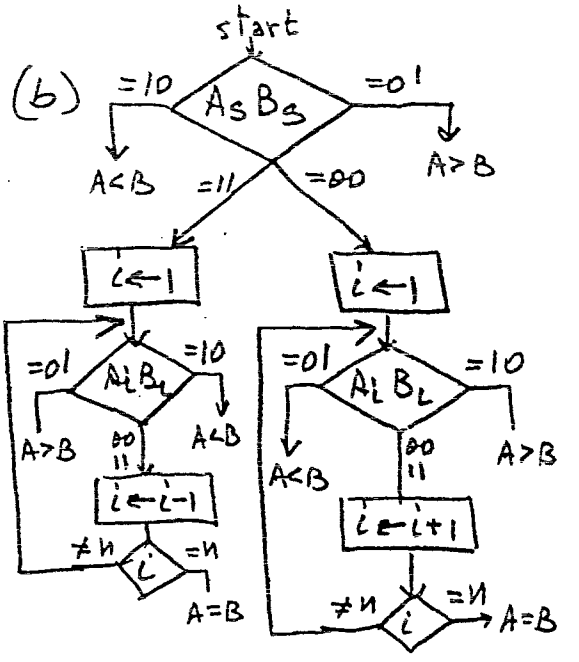


10-19

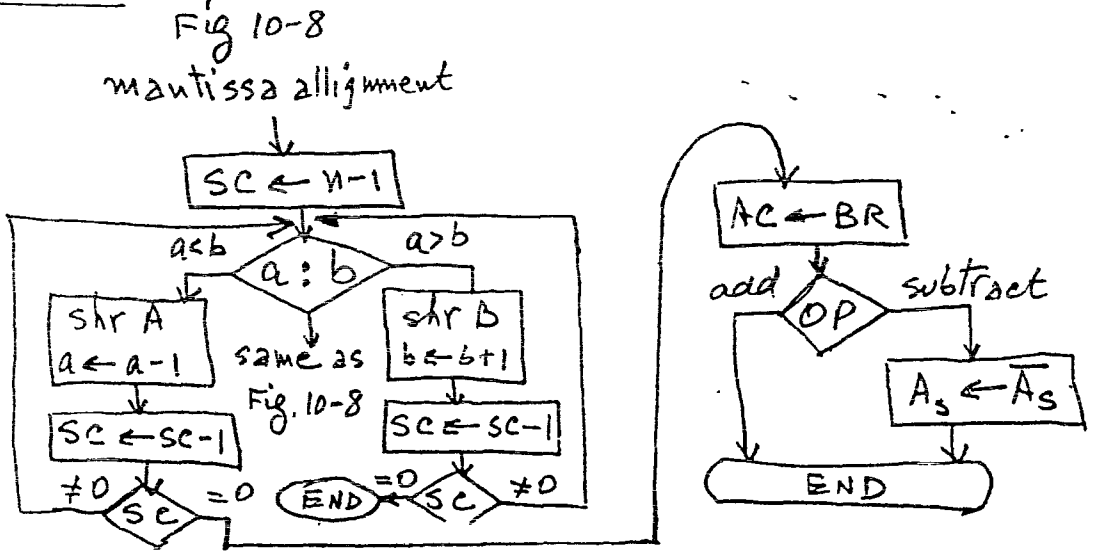
A_S $A_1 A_2 A_3 \ldots A_n$ (A)

B_S $B_1 B_2 B_3 \ldots B_n$ (B)

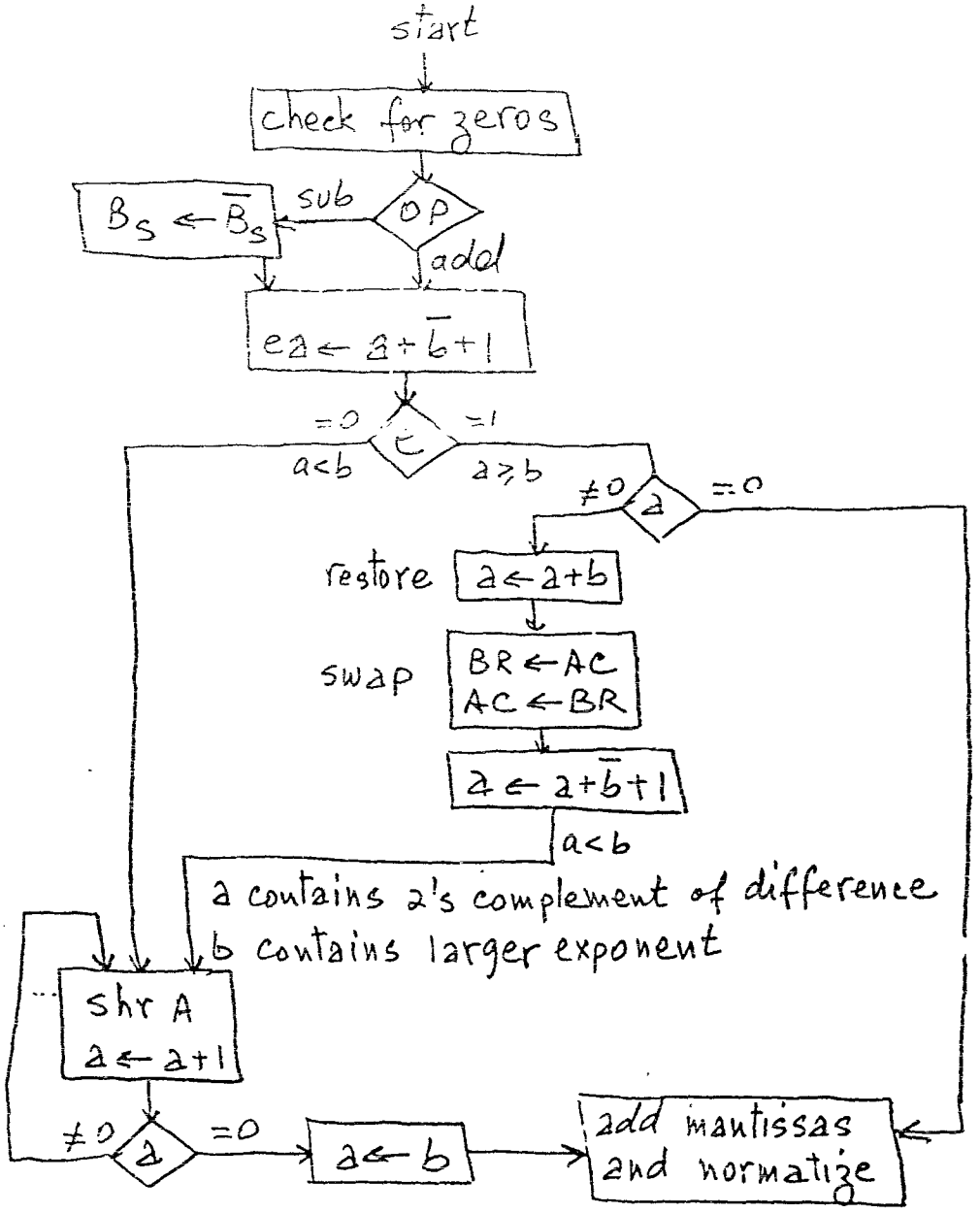




10-20



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



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 $\frac{\text{dividend}}{\text{divisor}}$ $A = 0.1xxxx$ $B = 0.1xxxx$ where $x = 0, 1$

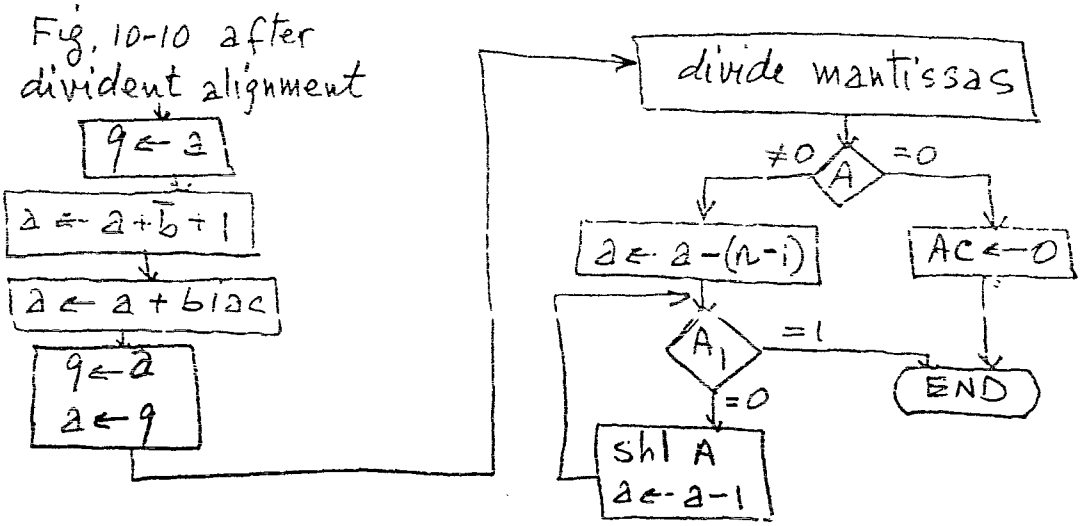
(a) If A<B then after shift we have $A = 1.xxxx$ and 1st quotient bit is a 1.

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

10-24

$$\frac{\text{dividend}}{\text{divisor}} = \frac{\overbrace{.1xxxx}^{n-1\text{ bits}} * 2^{e_1}}{.1yyyy * 2^{e_2}} = .1aaaa * 2^{e_1-e_2} + \frac{\overbrace{.0000rrrrr}^{\text{remainder}} * 2^{e_1}}{.1yyyy * 2^{e_2}}$$

Remainder bits rrrrr have a binary-point (n-1) bits to the left.



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: A Q
xxxxx xxxxx. $*2^3$
↖ binary-point for integer

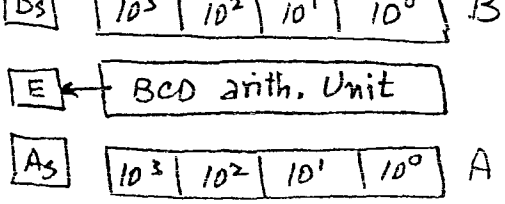
Product in AC: xxxxx. $*2^{3+5}$

(b) Single precision normalized dividend: xxxxx. $*2^3$

Dividend in AQ: A Q
xxxxx 00000. $*2^{3-5}$

10-27

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

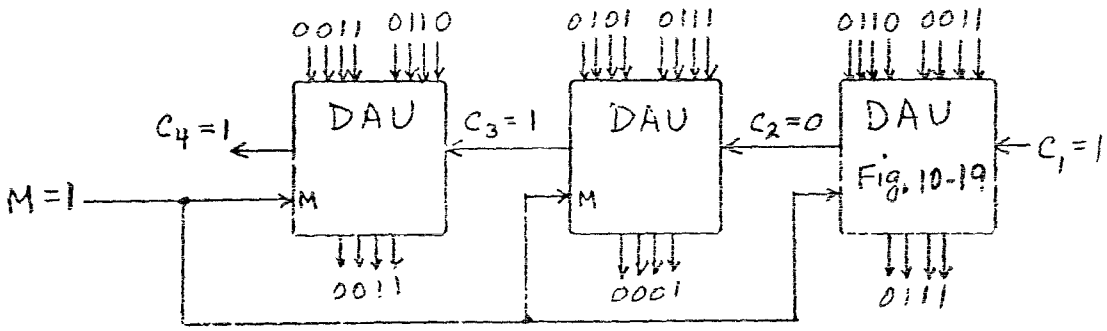


10-28

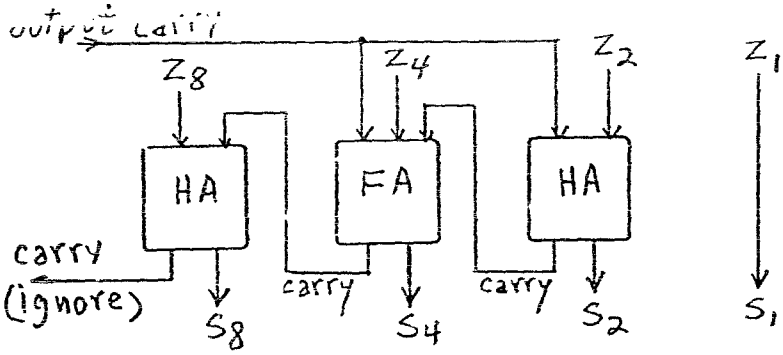
$$\begin{array}{r} 673 \\ -\ 356 \\ \hline 317 \end{array}$$

10's comp. of 356 = 644

$$\begin{array}{r} 673 \\ +\ 644 \\ \hline \text{carry} \rightarrow 1)\ 317 \end{array}$$

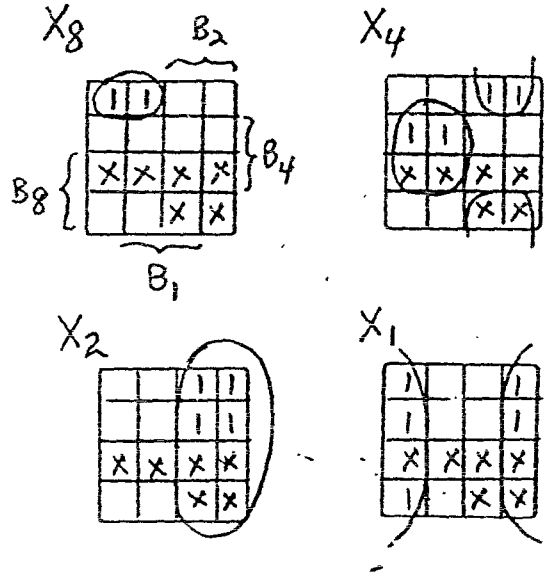


10-29



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



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

$X_8 = B_8' B_4' B_2'$

$X_4 = B_4 B_2' + B_4' B_2$

$X_2 = B_2$

$X_1 = B_1'$

10-31

dec	Z uncorrected	Y corrected
0	0110	0011
1	0111	0100
2	1000	0101
3	1001	0110
4	1010	0111
5	1011	1000
6	1100	1001
7	1101	1010
8	1110	1011
9	1111	1100

No output carry

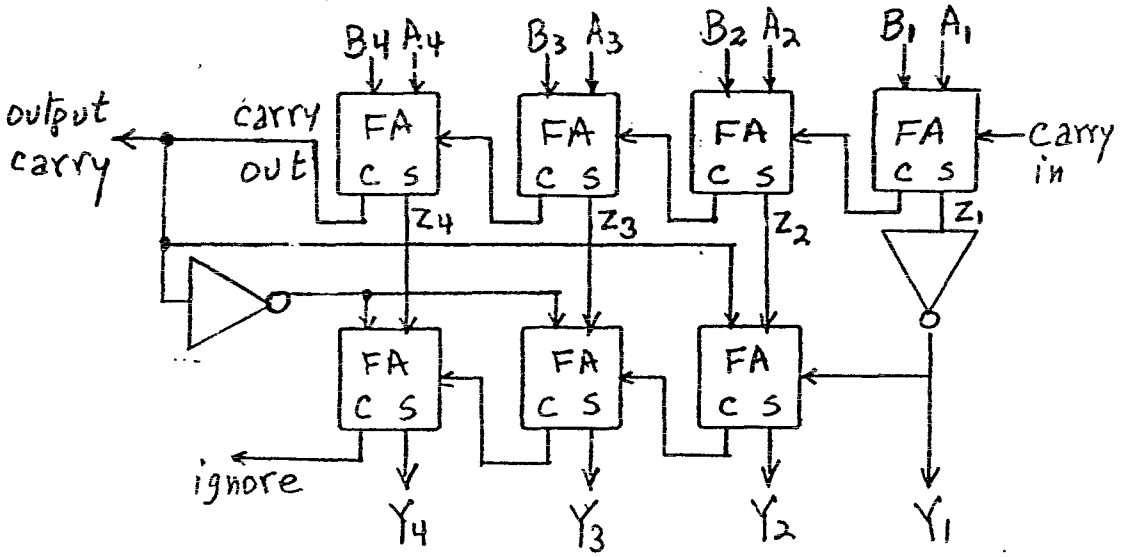
$Y = Z - 3 = Z + 13 - 16$

ignore carry

dec	Z uncorrected	Y corrected
10	10000	10011
11	10001	10100
12	10010	10101
13	10011	10110
14	10100	10111
15	10101	11000
16	10110	11001
17	10111	11010
18	11000	11011
19	11001	11100

Uncorrected carry = output carry

$Y = Z + 3$

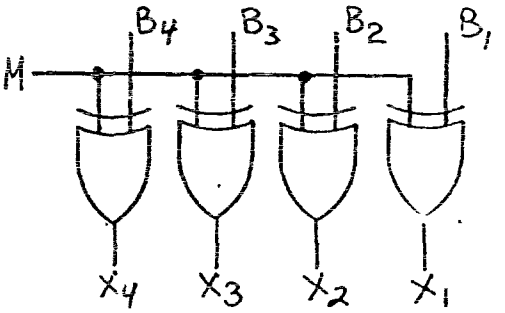


10-32. The excess-3 code is self-complementing code. Therefore, to get 9's complement we need to complement each bit.

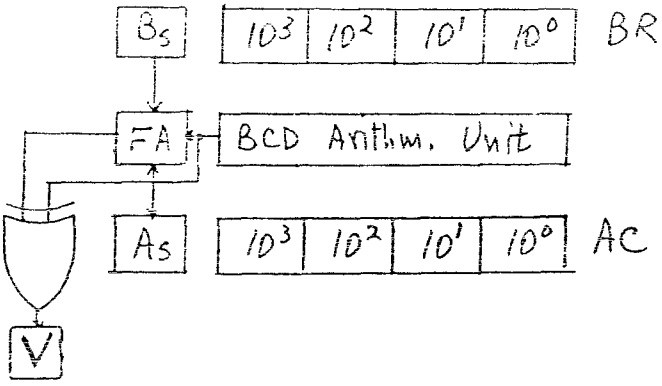
M=0 for x=B

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

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



10-33



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

10-34 (a) B = 410

	Ae	A	Q ↓ Q_L	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
$Q_L \neq 0$	0	564	014	
	1	034	013	
	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)

$$\begin{array}{r} 999 \\ \times\ 199 \\ \hline 8991 \\ +\ 89910 \\ \hline 98901 \\ +\ 99900 \\ \hline 198801 \end{array}$$

first partial product Ae = 8

second partial product Ae = 9

final product Ae = 1

10-35 $\frac{1680}{32} = 52 + \frac{16}{32}$

B = 032
$\bar{B}+1 = 968$ (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 $A_e = 9$. Adding $B_e = 0$ to $A_e = 9$ produces a carry and makes $A_e = 0$.

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.

CHAPTER 11

11-1

$A_7 - A_2$ A_1 A_0

12 = 000011 00

13 = 000011 01

14 = 000011 10

15 = 000011 11

To CS RS1 RS0

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

$RS1 = A_1$

$RS0 = A_0$

11-2

Interface	Port A	Port B	Control Reg	Status Reg
# 1	10000000	10000001	10000010	10000011
2	01000000	01000001	01000010	01000011
3	00100000	00100001	00100010	00100011
4	00010000	00010001	00010010	00010011
5	00001000	00001001	00001010	00001011
6	00000100	00000101	00000110	00000111

11-3

Character printer; Line printer; Laser printer; Digital ploller; 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.

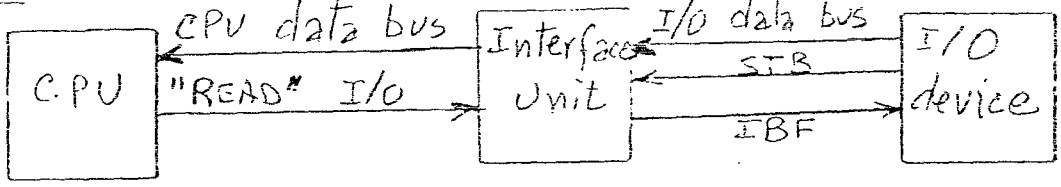
(c) Status command - cheks if device power is on.

(d) Control command - Moves paper position.

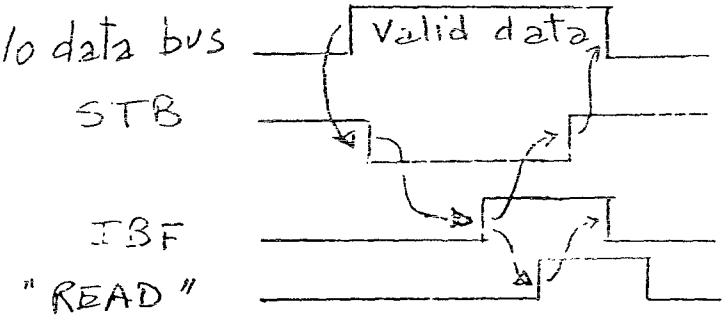
(e) Data input command - Reads value of a register

11-7

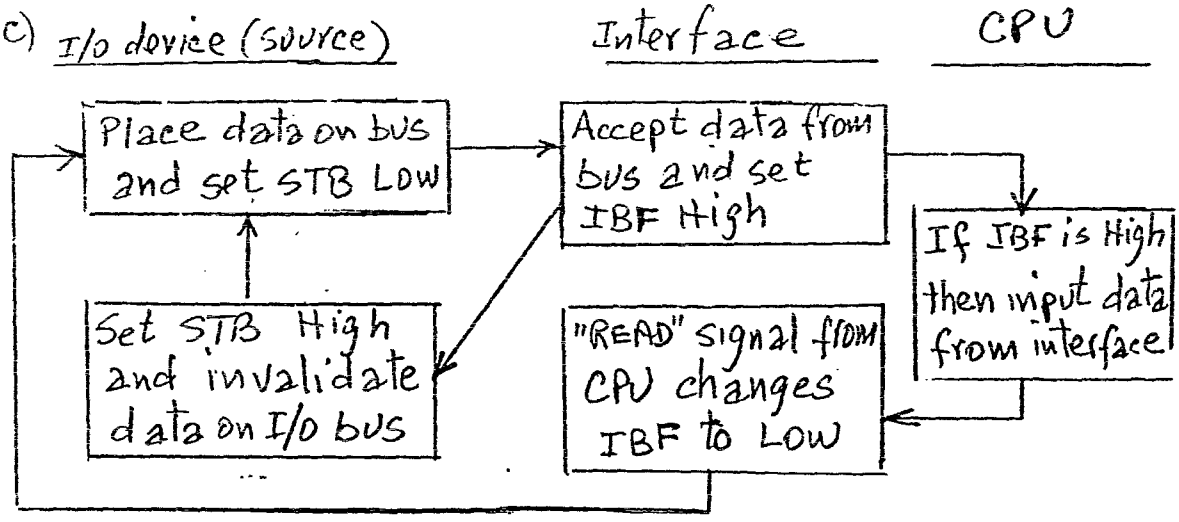
(a)



(b)

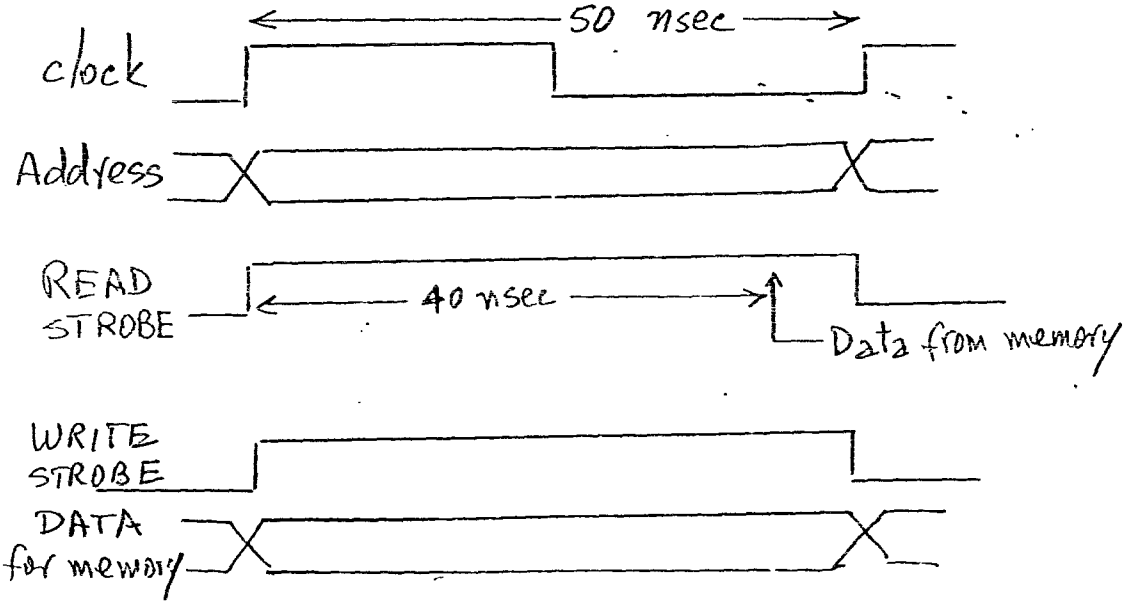


(c)

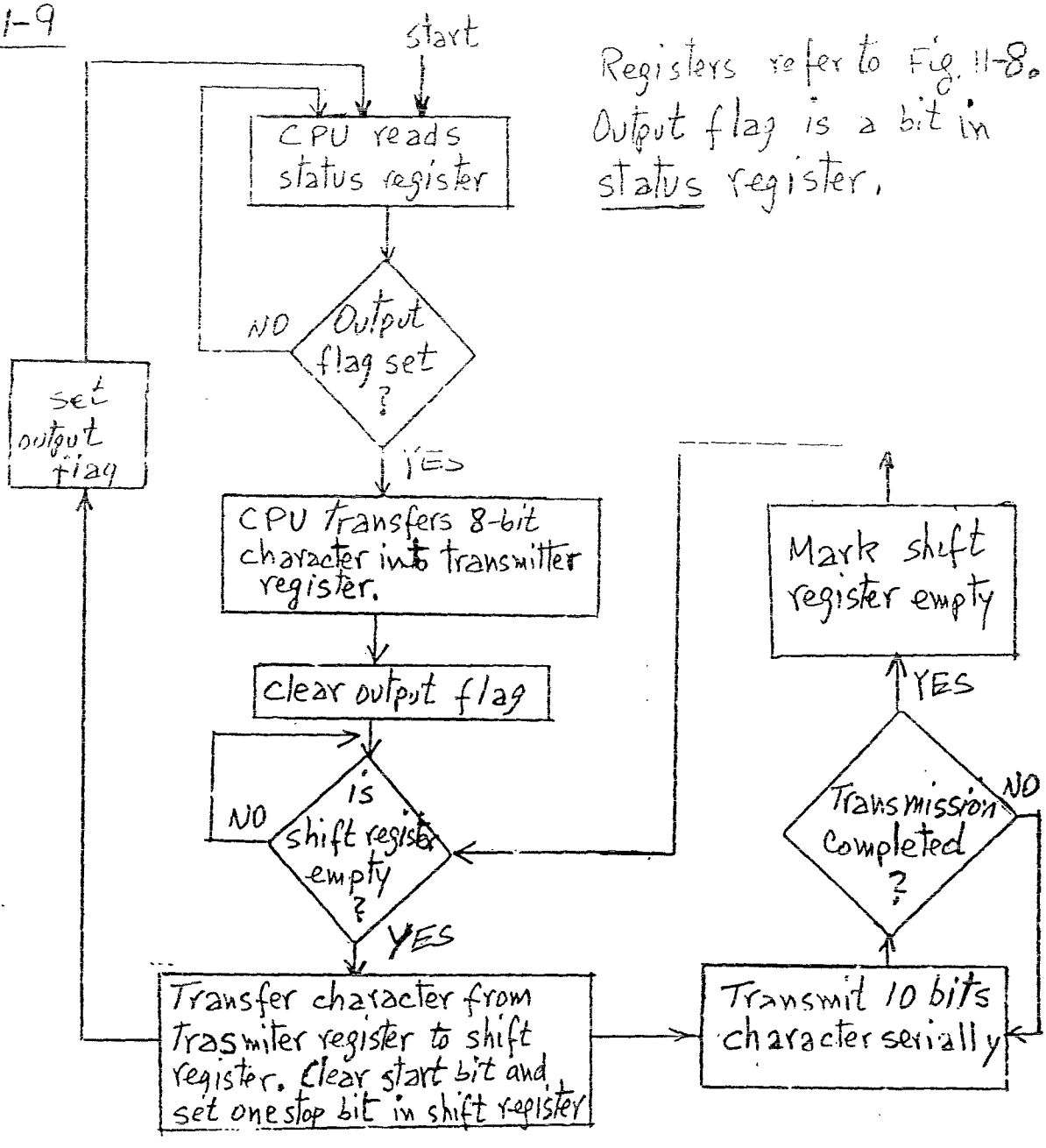


11-8

$20\,MHz = 20 \times 10^6\,Hz \qquad T = \frac{10^{-6}}{20} = 50 \text{ nsec.}$



11-9



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

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

11-12

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

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

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

11-13

(a) $\frac{k \text{ bytes}}{(m-n) \text{ bytes/sec}} = \frac{k}{m-n}$ sec.

(b) $\frac{k}{n-m}$ sec. (c) No need for FIFO

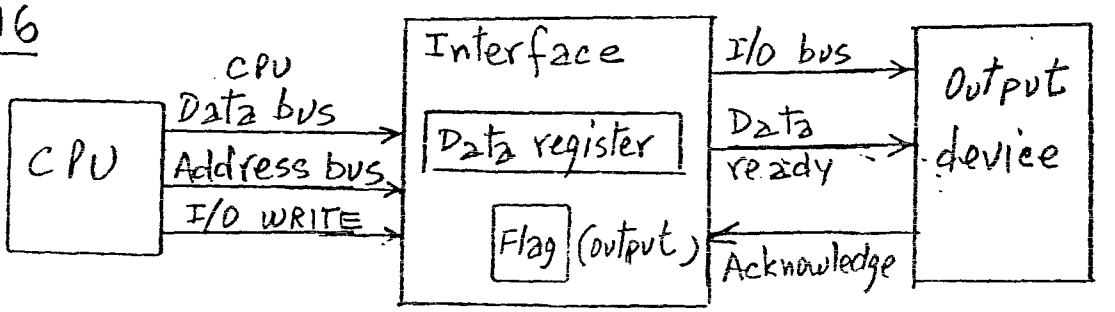
11-14

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

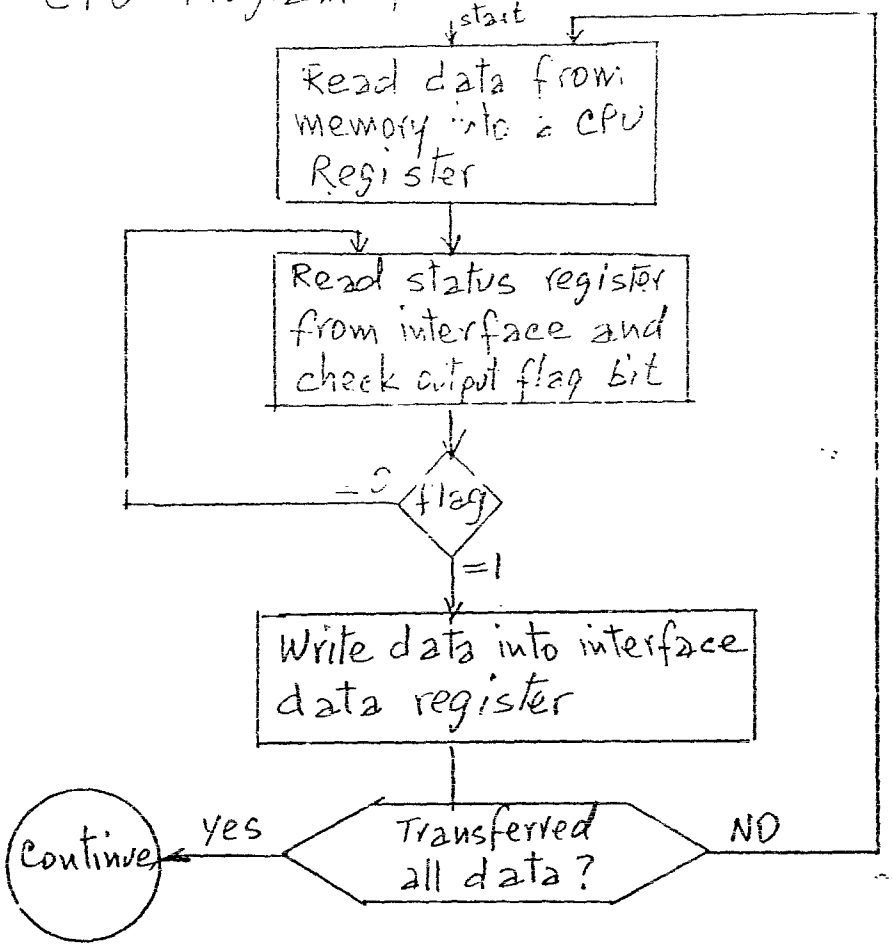


Flag = 0 if data register full (After CPU writes data)

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:



11-18

See text Section 11-4.

11-19

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

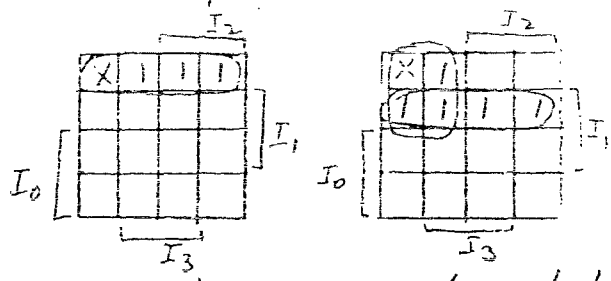
11-20

	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



$x = I_0' I_1'$ $\quad y = I_0' I_1 + I_0' I_2'$

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	xy3	IST
1	x	x	x	x	x	x	x	000	1
0	1	x	x	x	x	x	x	001	1
0	0	1	x	x	x	x	x	010	1
0	0	0	1	x	x	x	x	011	1
0	0	0	0	1	x	x	x	100	1
0	0	0	0	0	1	x	x	101	1
0	0	0	0	0	0	1	x	110	1
0	0	0	0	0	0	0	1	111	1
0	0	0	0	0	0	0	0	xxx	0

(b)

Binary	hexadecimal
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

$76 = (01001100)_2$

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.

11-21

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

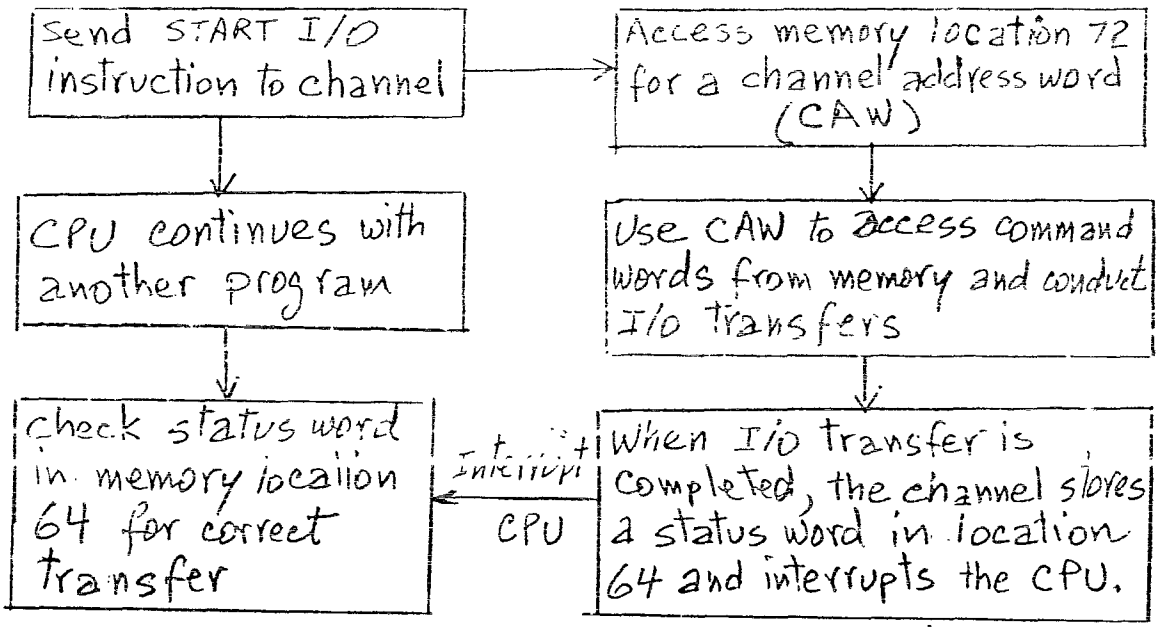
11-29

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.

11-31



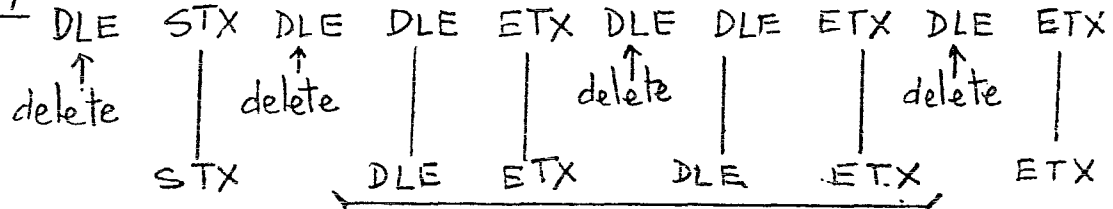
11-32 There are 26 letters and 10 numerals.

$26 \times 26 + 26 \times 10 = 936$ possible addresses.

11-33

The processor transmits the address of the terminal followed by ENQ (enquiry) code 0000 0101. The terminal responds with eithe 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



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 1111 1111 111

After zero insertion, information transmitted: 0 1111 10 11111 0 (delete)

Information received after 0's deletion: 0 1111 1 11111

CHAPTER 12

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) 4x16 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	← 5x32 decoder →	x x x	x x x x
ROM	1000-1FFF	0 0 0 1	← 3x8 decoder →	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 ← 3x8 decoder →	x x x	x x x x
ROM	4000-4FFF	0 1 0 0	← 2x4 decoder →	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.

12-7 Average time = T_s + time for half revolution + time to read a sector.

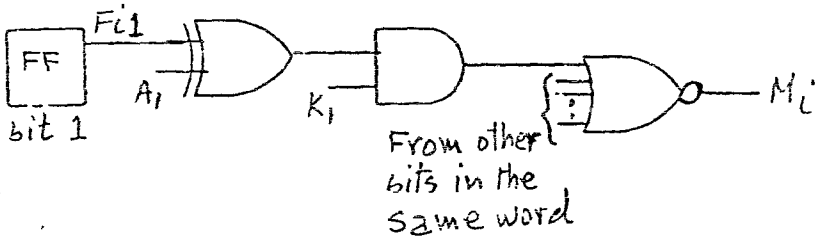
$$T_a = T_s + \frac{1}{2R} + \frac{b_j}{N_t} \times \frac{1}{R}$$

12-8 An eight-track tape reads 8 bits (one character) at the same time. Transfer rate = 1600 × 120 = 192,000 charaters/s

12-9

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

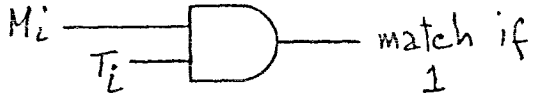
$$M_i' = \sum_{j=1}^{n} (A_j \oplus F_{ij}) K_j$$



12-10

A match occurs if $T_i = 1$

match = $M_i T_i$

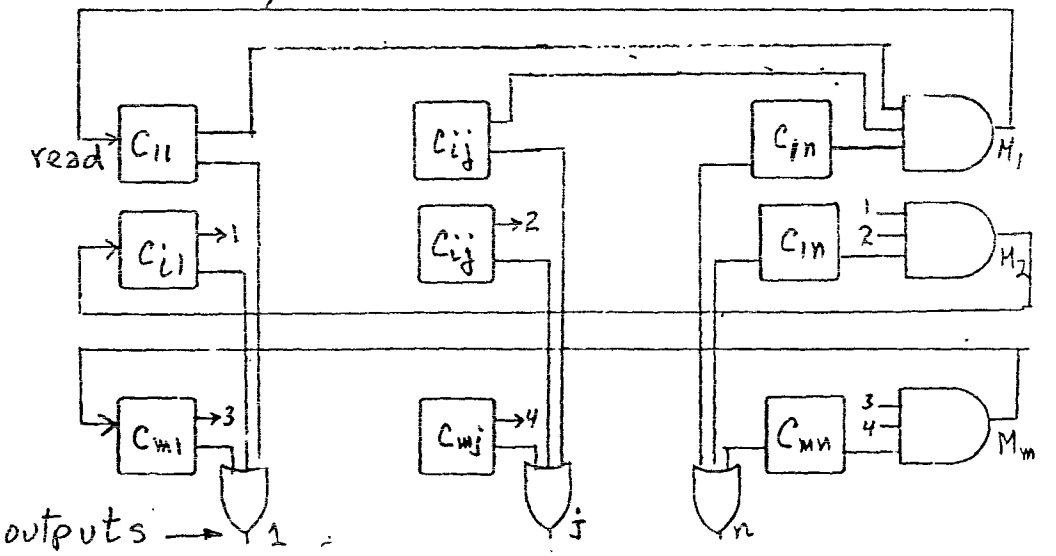


12-11

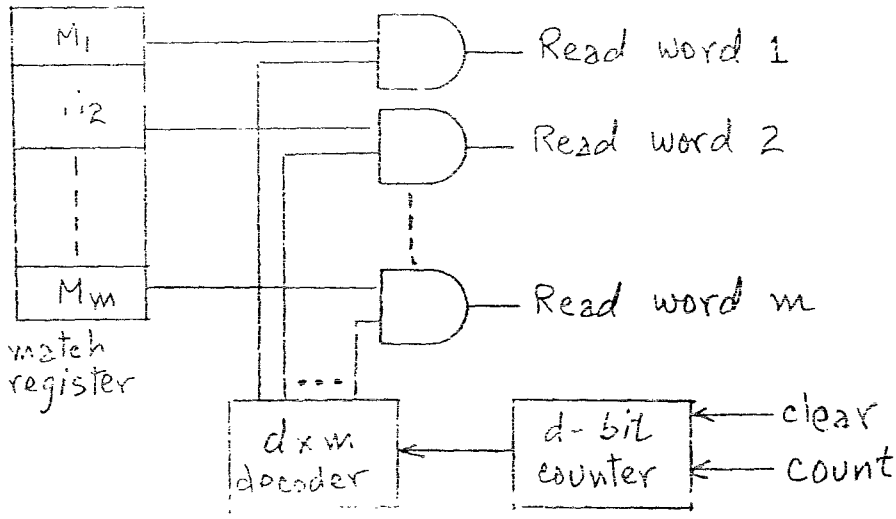
$$M_i = \left(\prod_{j=1}^{n} A_j F_{ij} + A_j' F_{ij}' + K_j' \right) \cdot (K_1 + K_2 + K_3 + \cdots + K_n)$$

At least one key bit K_i must be equal to 1

12-12(c)



12-13



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 $x_j = A_j F_{ij} + A_j' F_{ij}'$ (argument bit = memory word bit)

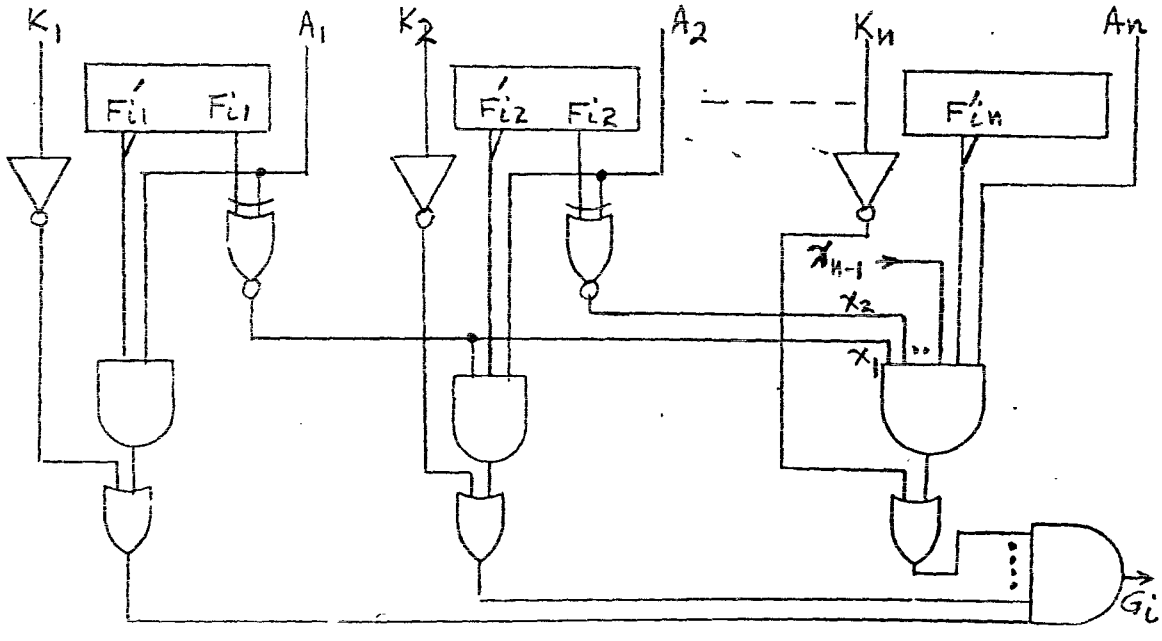
Output indicator $G_i = 1$ if:

$A_1 F_{i1}' = 1$ 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.

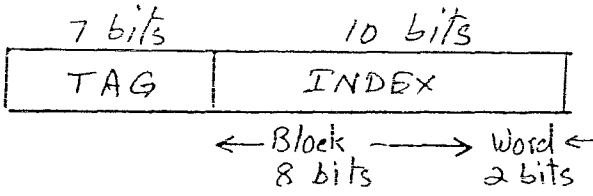
$$G_i = (A_1 F_{i1}' + K_1')(x_1 A_2 F_{i2}' + K_2')(x_1 x_2 A_3 F_{i3}' + K_3') \ldots (x_1 x_2 \ldots x_{n-1} A_n F_{in}' + K_n')$$



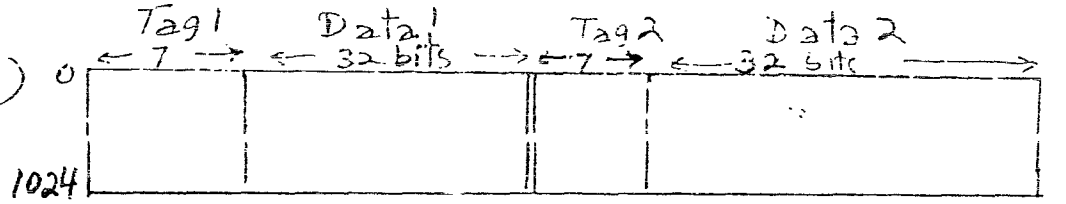
12-15

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

(a)



(b)



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

12-16

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

(b) $0.2 \times \underbrace{1000}_{\text{write access}} + 0.8 \times \underbrace{200}_{\text{read access from (a)}} = 200 + 160 = 360$ nsec.

(c) Hit ratio $= 0.8 \times 0.9 = 0.72$

12-17

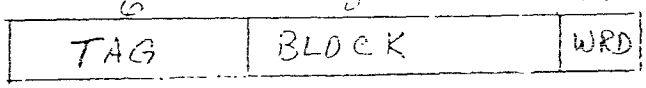
Sequence: A B C D B E D A C E C E

	LRU → 3	2	1	0
Count value =				
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.

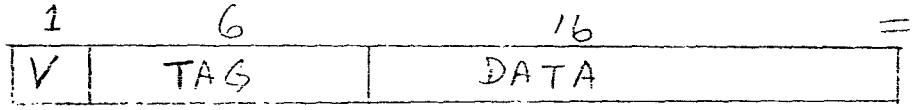
(a)



= 16 bit address

INDEX = 10 bit cache address.

(b)



= 23 bits in each word of cache

(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) $\frac{16M}{2K}$ = 8K pages $\frac{64K}{2K}$ = 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 14 0 10 2 3 5 7

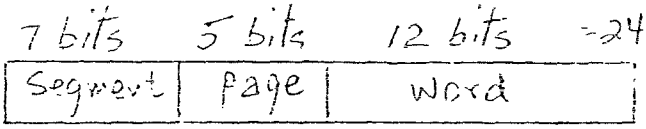
Page reference	(a) Pages in main memory (First-in)	Contents of FIFO	(b) Pages in memory	LRU ↓ Most recently used ↓
Initial	0124	4201	0124	4201
2	0124	4201	0124	4012
6	0126	2016	0126	0126
1	0126	2016	0126	0261
4	0146	0164	1246	2614
0	0146	0164	0146	6140
1	0146	0164	0146	6401
0	0146	0164	0146	6410
2	1246	1642	0124	4102
3	2346	6423	0123	1023
5	2345	4235	0235	0235
7	2357	2357	2357	2357

12-22

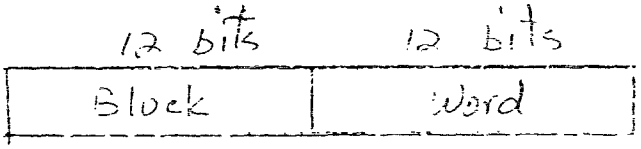
600AF and F00AF

12-23

Logical address:



Physical address:



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 = 010010001111011111010000

(24-bit binary)

CHAPTER 13

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

$p \times m$ switches

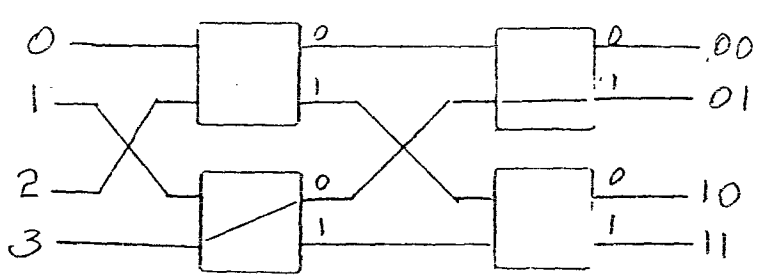
13-4

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

13-5

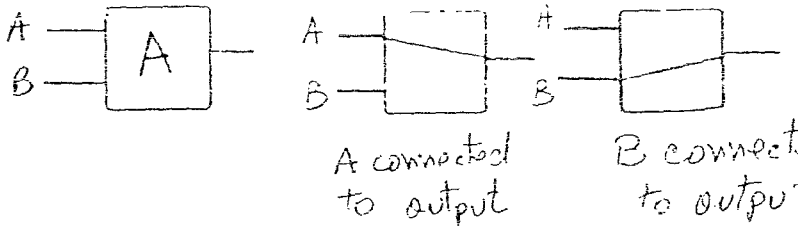
Inputs 0, 2, 4, and 6 will be disconnected from outputs 2 and 3.

13-6

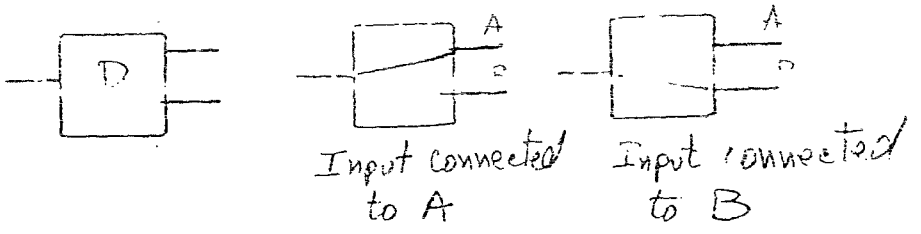


13-7 Arbitration switch:

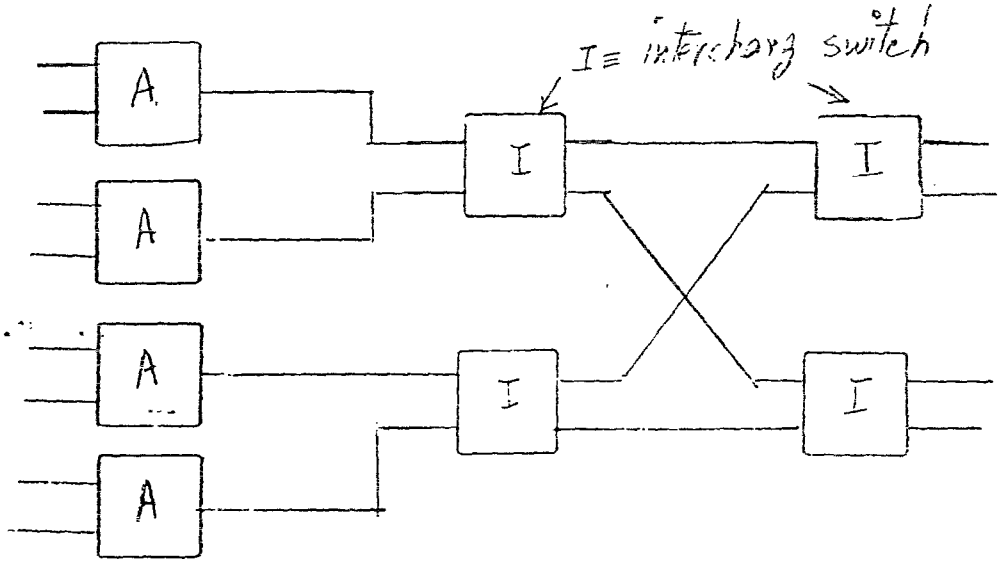
(a)



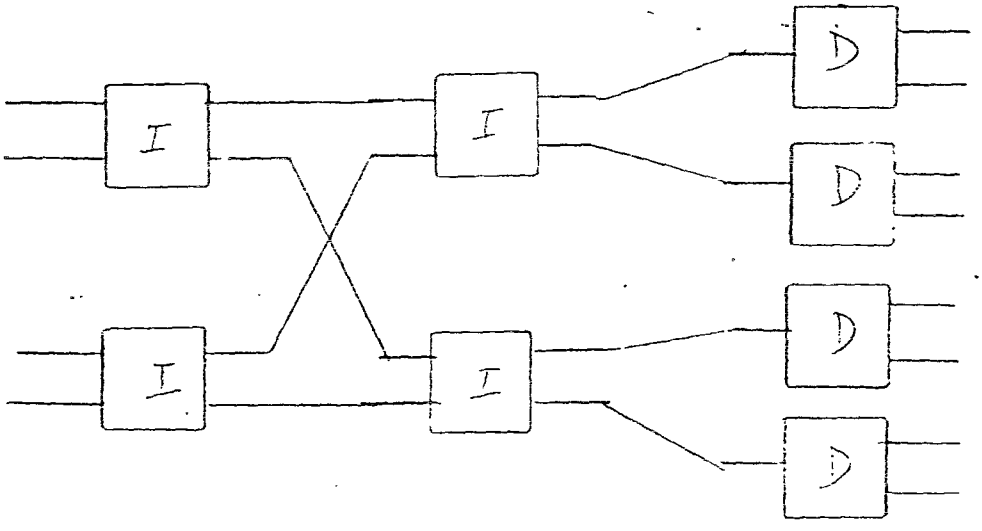
Distribution switch:



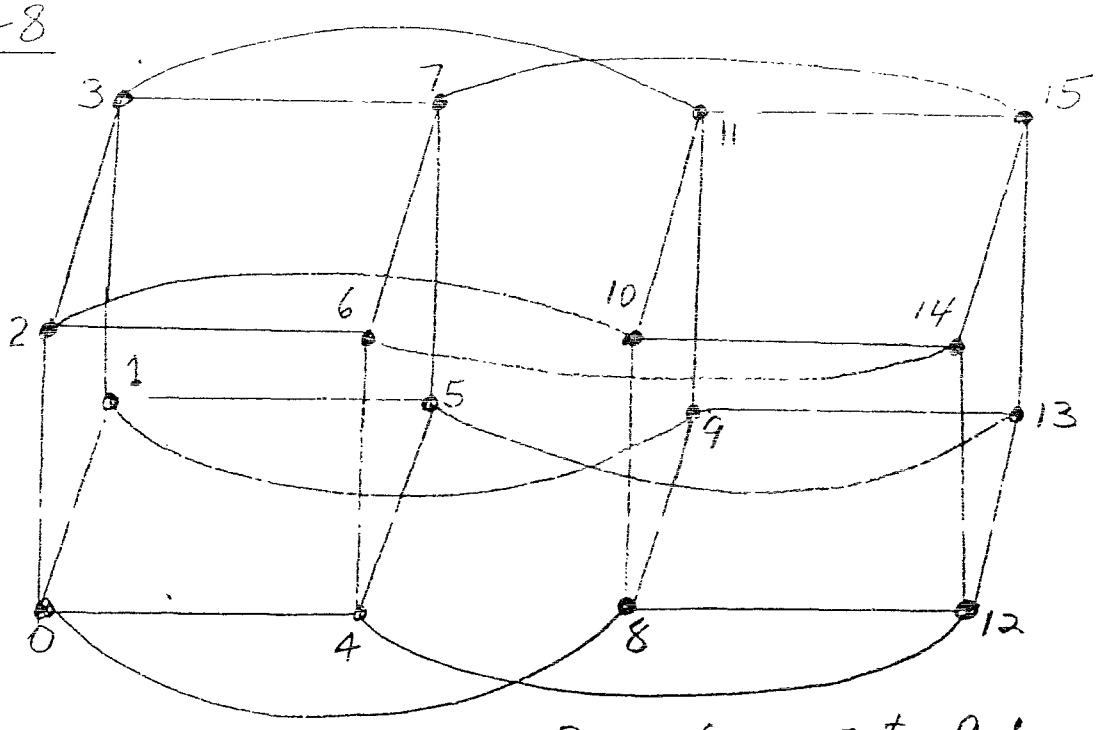
(b)



(c)



13-8

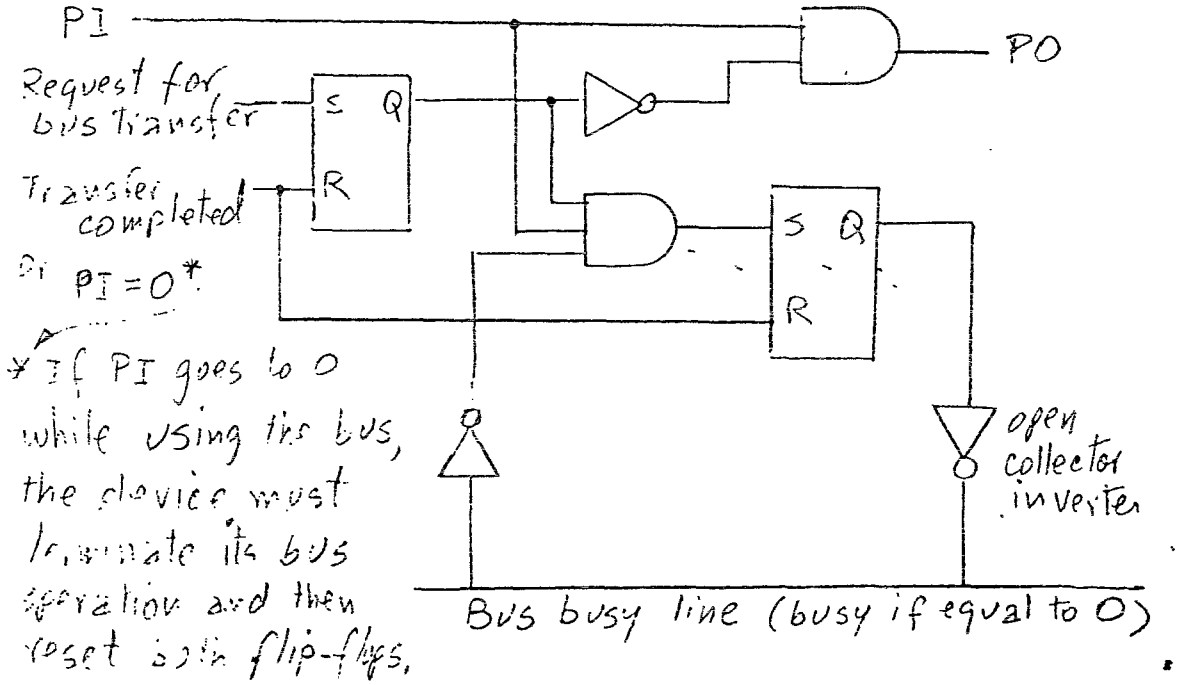


7 = 0111
9 = 1001 ⊕
1110 = three axes

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



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

13-10

Encoder input	0 1 1 0
Encoder output	01 (I1 has highest priority)
Decoder input	01
Decoder output	0 1 0 0 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 unsuccesful test by the receiver. One way to speed the transfer would be to send an interrupt request to the receiving processor.

13-13

(a) Mutual exclusion implies that each processor claims exclusive control of the resources alocated to it.

(b) Critical section is a program sequence that must be comptely executed without interruptions by othe processors.

(Contined in next page)

13-13 (Continued)

(c) Hardware lock is a hardware signal to ensure that a memory read is followed by a memory write without interruption from another processor.

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

(e) Test and set instruction 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.