

SOLUTIONS MANUAL  
M. MORRIS MANO

---

---

# COMPUTER SYSTEM ARCHITECTURE

---

---

Third Edition

Solutions Manual

# Computer System Architecture

## TABLE OF CONTENTS

Chapter 1 .....	4
Chapter 2 .....	11
Chapter 3 .....	16
Chapter 4 .....	20
Chapter 5 .....	26
Chapter 6 .....	34
Chapter 7 .....	45
Chapter 8 .....	51
Chapter 9 .....	59
Chapter 10 .....	63
Chapter 11 .....	80
Chapter 12 .....	89
Chapter 13 .....	95

## CHAPTER 1

$$\nabla \text{-----} = \text{-----} \nabla$$

### 1.1

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

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

### 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

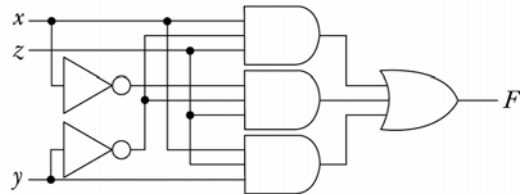
- (a)  $F = x'y + xyz'$   
 $F' = (x + y')(x' + y' + z) = x'y' + xy' + y' + xz + y'z$   
 $= y'(1 + x' + x + z) + xz = y' + xz$   
 (b)  $F \cdot F' = (x'y + xyz')(y' + xz) = 0 + 0 + 0 + 0 = 0$   
 (c)  $F + F' = x'y + xyz' + y' + xz (y + y')$   
 $= x'y + xy(z' + z) + y'(1 + xz) = x'y + xy + y'$   
 $= y(x' + x) + y' = y + y' = 1$

### 1.7

(a)

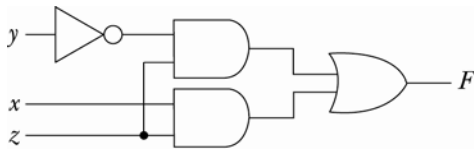
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

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



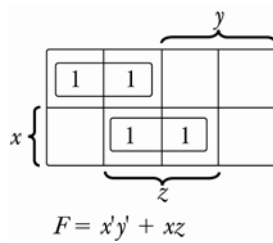
(c)  $F = xy'z + x'y'z + xyz$   
 $= y'z(x + x') + xz(y + y')$   
 $= y'z + xz$

(d) Same as (a)

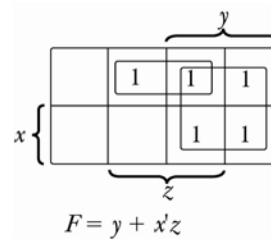


### 1.8

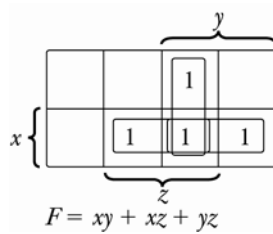
(a)



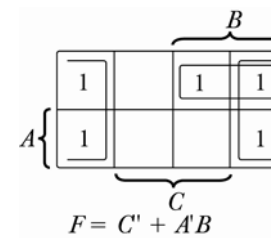
(b)



(c)

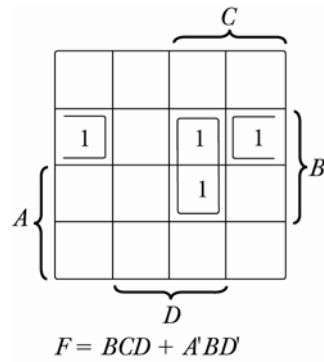


(d)

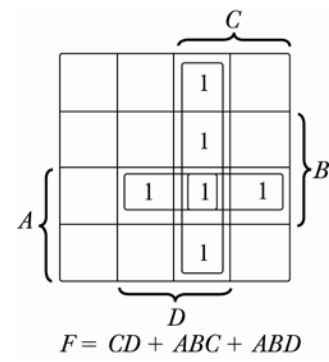


### 1.9

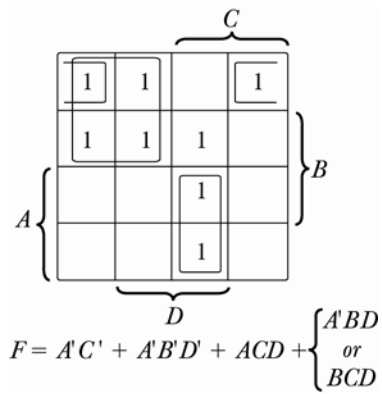
(a)



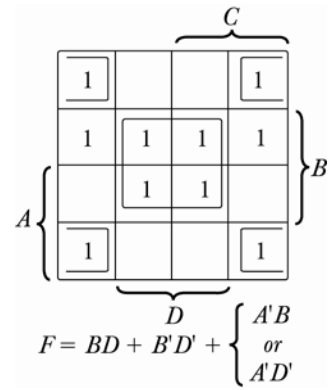
(b)



(c)

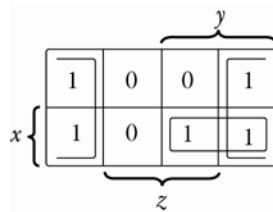


(d)



### 1.10

(a)



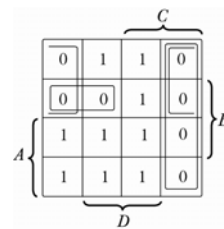
(1)

$$F = xy + z'$$

(2)

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

(b)



(1)

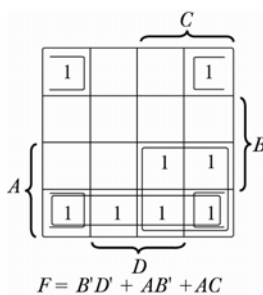
$$F = AC' + CD + B'D$$

(2)

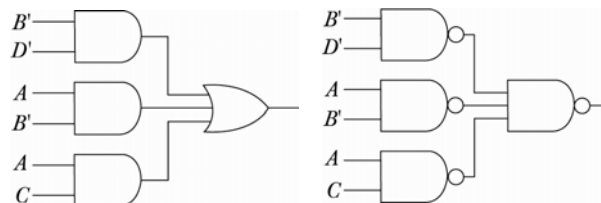
$$F = (A + D)(C' + D)(A + B' + C)$$

### 1.11

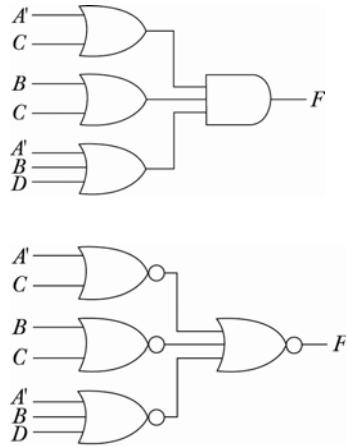
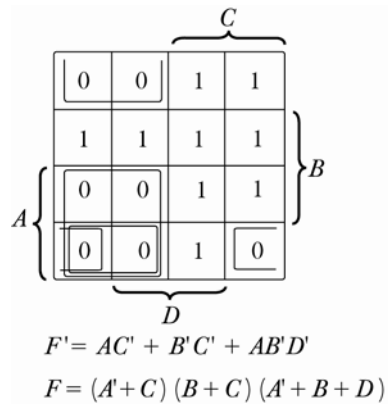
(a)



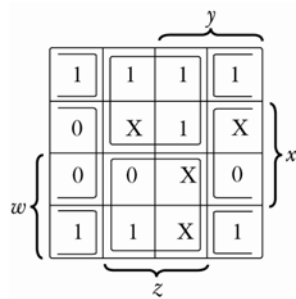
(b)



### 1.12



### 1.13



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

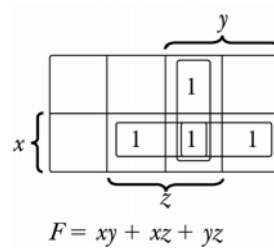
### 1.14

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

See Fig. 1.2  
(Exclusive - NDR)

### 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

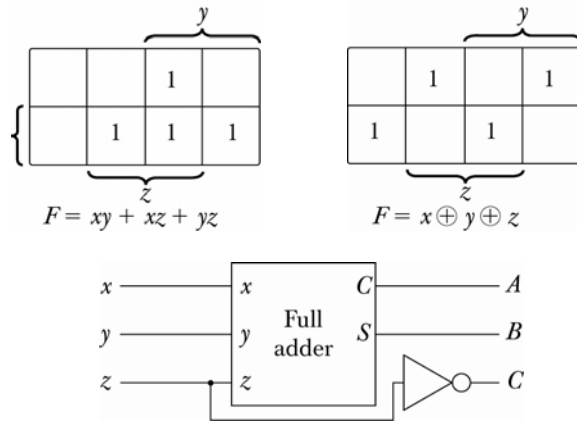


### 1.16

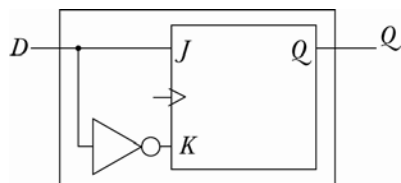
x y z	A B C
0 0 0	0 0 1
0 0 1	0 1 0
0 1 0	0 1 1
0 1 1	1 0 0
1 0 0	0 1 1
1 0 1	1 0 0
1 1 0	1 0 1
1 1 1	1 1 1

$c = z'$

By inspection



### 1.17



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

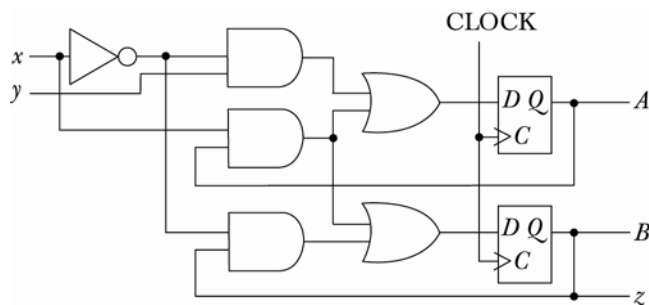
When  $D = 1$ ;  $J = 1$ ,  $K = 0$ ,  $Q \rightarrow 1$

### 1.18

See text, Section 1.6 for derivation.

### 1.19

(a)  $D_A = x'y + xA$ ;  $D_B = x'B + xA$ ;  $z = B$





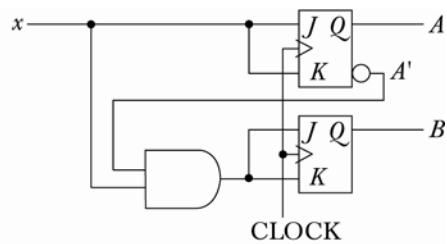
(b)

Present state AB	Inputs x y	Next state A B	Output 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**

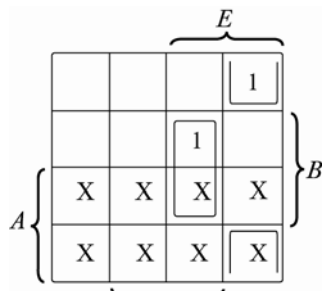
$$J_A = K_A = x$$

$$J_B = K_B = A'x$$

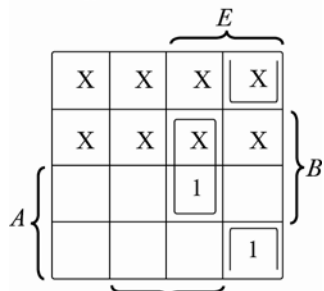
**1.21**

Count up-down binary counter with table E

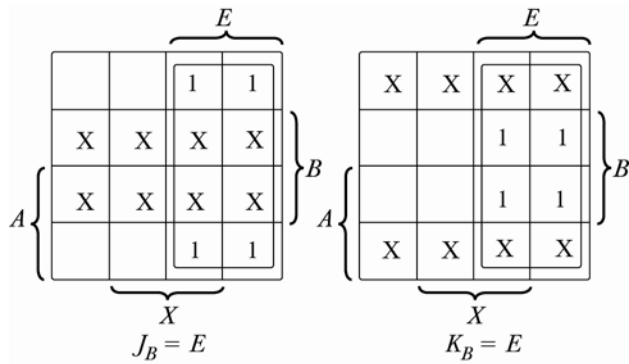
Present state A B	Inputs EX	Next state A B	Flip-flop inputs	
			$J_A K_A$	$J_B K_B$
0 0	0 0	0 0	0 X	0 X
0 0	0 1	0 0	0 X	0 X
0 0	1 0	1 1	1 X	1 X
0 0	1 1	0 1	0 X	1 X
0 1	0 0	0 1	0 X	X 0
0 1	0 1	0 1	0 X	X 0
0 1	1 0	0 0	0 X	X 1
0 1	1 1	1 0	1 X	X 1
1 0	0 0	1 0	X 0	0 X
1 0	0 1	1 0	X 0	0 X
1 0	1 0	0 1	X 1	1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

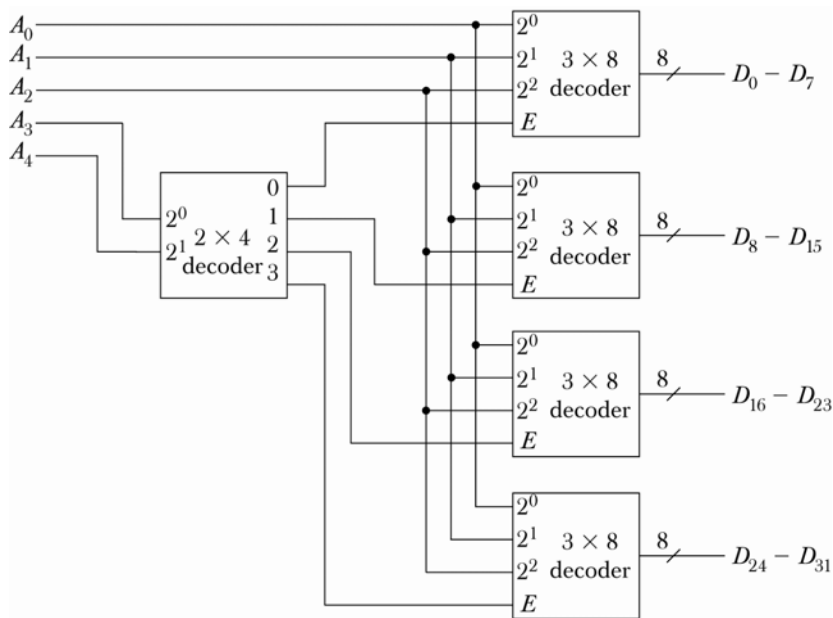
### 2.1

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

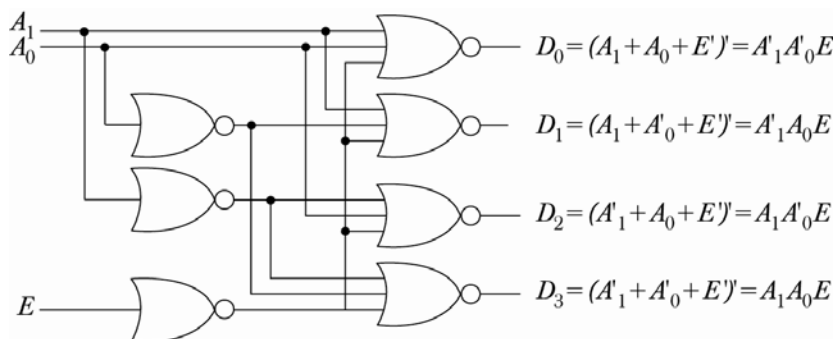
### 2.2

- (a) 74155 – Similar to two decoders as in Fig. 2.2.
- (b) 74157 – Similar to multiplexers of Fig. 2.5.
- (c) 74194 – Similar to register of Fig. 2.9.
- (d) 74163 – Similar to counter of Fig. 2.11

### 2.3



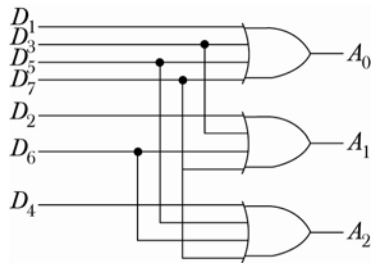
### 2.4



## 2.5

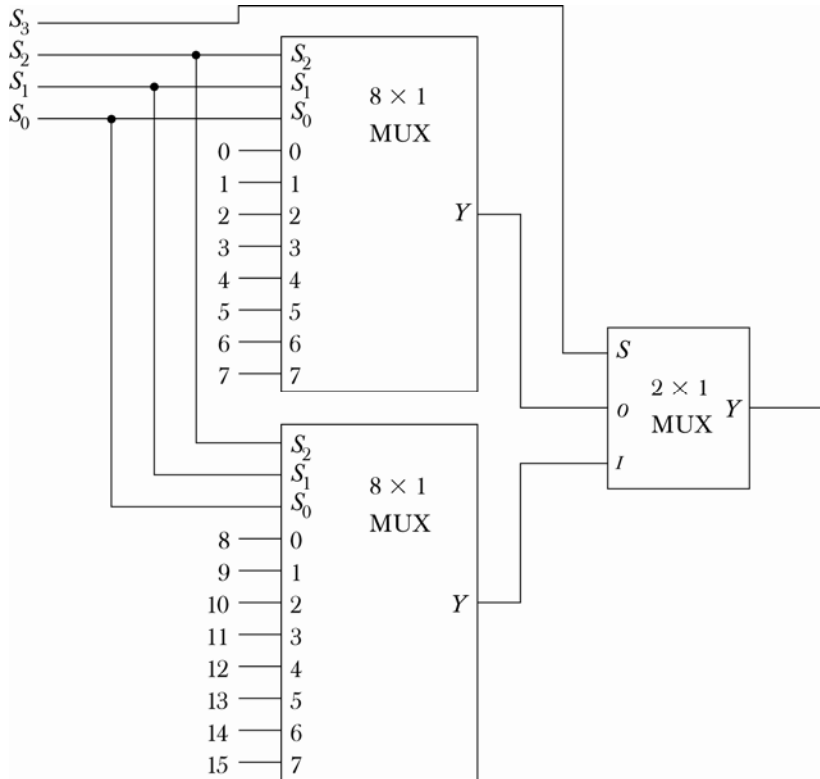
Remove the inverter from the E input in Fig. 2.2(a).

## 2.6

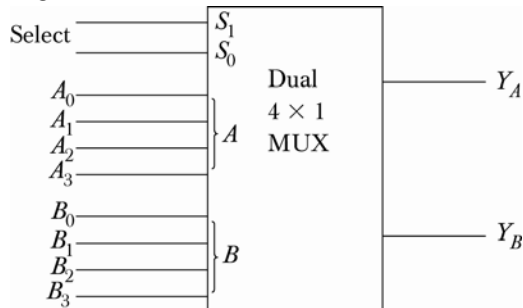


If all inputs equal 0 or if only  $D_0 = 1$ :  
the outputs  $A_2A_1A_0 = 000$ .  
Needs one more output to recognize  
the all zeros input condition.

## 2.7



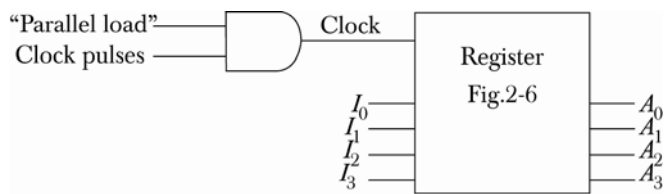
## 2.8



$S_1S_0$	$Y_A Y_B$
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

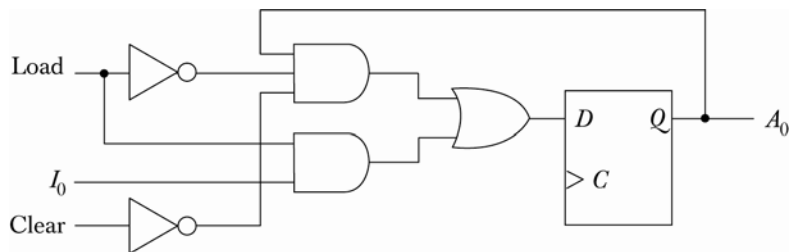


When the parallel load input = 1, the clock pulses go through the AND gate and the data inputs are loaded into the register when the parallel load input = 0, the output of the AND gate remains at 0.

## 2.10

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

## 2.11

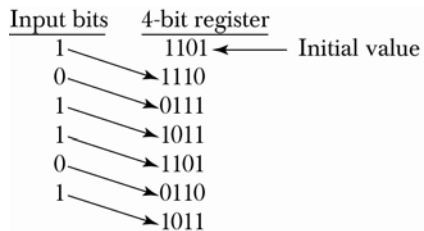


One stage of Register Fig. 2.7

Load	Clear	D	Operation
0	0	Q(t)	no change
0	1	0	Clear to 0
1	x	I <sub>0</sub>	load I <sub>0</sub>

Function table

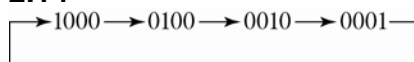
## 2.12



## 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



**2.17**

Count  
Load  
Clear  
Clock

Fig. 2-11 stage 1

Output Carry

Count  
Load  
Clear  
Clock

Fig. 2-11 stage 2

Output Carry

To next Stage

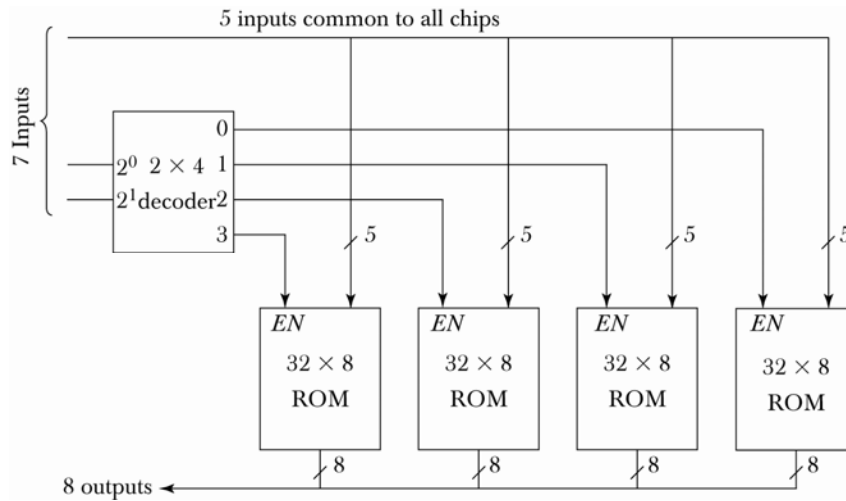
		Address lines	Data lines
(a)	$2K \times 16 = 2^{11} \times 16$	11	16
(b)	$64K \times 8 = 2^{16} \times 16$	16	8
(c)	$16M \times 32 = 2^{24} \times 32$	24	32
(d)	$4G \times 64 = 2^{32} \times 64$	32	64

**2.20**

- (a)  $2K \times 2 = 4K = 4096$  bytes  
 (b)  $64K \times 1 = 64K = 2^{16}$  bytes  
 (c)  $2^{24} \times 4 = 2^{26}$  bytes  
 (d)  $2^{32} \times 8 = 2^{35}$  bytes

**2.21**

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

**2.22****2.23**

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

## 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

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

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

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

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

### 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

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

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

$$\text{Therefore: } (10)_r = (13)_{10}$$

$$r = 13$$

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

$$(r = 13)$$

### 3.7

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

$$(a) \quad 000011010111 \quad \text{Binary}$$

$$(b) \quad \begin{array}{cccc} 000 & 011 & 010 & 111 \\ 0 & 3 & 2 & 7 \end{array} \quad \text{Binary coded octal}$$

$$(c) \quad \begin{array}{ccc} 0000 & 1101 & 0111 \\ 0 & D & 7 \end{array} \quad \text{Binary coded hexadecimal}$$

$$(d) \quad \begin{array}{ccc} 0010 & 0001 & 0101 \\ 2 & 1 & 5 \end{array} \quad \text{Binary coded decimal}$$

### 3.8

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

$$(a) \quad 0000 \ 0000 \ 0000 \ 0001 \ 0010 \ 0111$$

$$(b) \quad 0000 \ 0000 \ 0000 \ 0010 \ 1001 \ 0101$$

$$(c) \quad 10110010 \quad 00111001 \quad 00110101$$

### 3.10

JOHN DOE



**3.11**

87650123; 99019899; 09990048; 999999.

**3.12**

876100; 909343; 900000; 000000

**3.13**

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

**3.14**

(a)	5250	(b)	1753	(c)	020	(d)	1200
	+ 8679		+ 1360		+ 900		+ 9750
	<u>1)3929</u>		<u>0)3113</u>		<u>0)920</u>		<u>1)0950</u>
			↓ = 10's complement □				
			- 6887		- 080		

**3.15**

(a)	(b)	(c)	(d)
11010	11010	000100	1010100
+10000	+10011	+ 010000	+ 0101100
<u>1)01010</u>	<u>1)01101</u>	<u>0)010100</u>	<u>1)0000000</u>
		↓	
(26 - 16 = 10)	(26 - 13 = 13)	-101100	(84 - 84 = 0)
		(4 - 48 = -44)	

**3.16**

+ 42 = 0101010	+13 = 0001101
- 42 = 1010110	-13 = 1110011
(+42) 0101010	(- 42) 1010110
<u>(-13) 1110011</u>	<u>(+ 13) 0001101</u>
(+29) 0011101	(- 29) 1100011

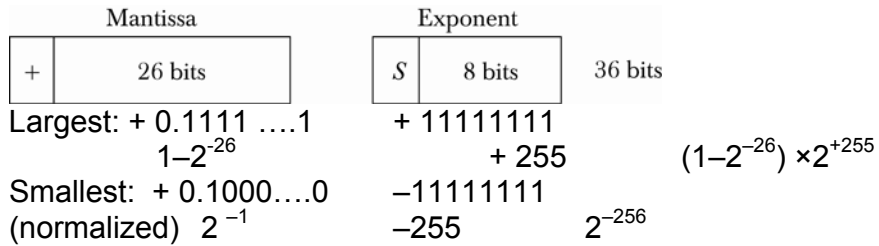
**3.17** 01 ← last two carries → 1 0

+ 70	01000110	- 70	10111010
+ 80	<u>01010000</u>	- 80	<u>10110000</u>
+150	10010110	- 150	01101010
↑	↑	↑	↑
greater	negative	less than	positive
than		- 128	
+127			

**3.18**

(a)	(-638)	9362	(b)	(-638)	9362
	<u>(+785)</u>	+ <u>0785</u>		<u>(-185)</u>	+ <u>9815</u>
	(+147)	0147		(-823)	9177

### 3.19



### 3.20

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

Sign

0101110100000000

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	Excess-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

<u>Decimal</u>	<u>BCD with even parity</u>	<u>BCD with odd parity</u>
0	00000	10000
1	10001	00001
2	10010	00010
3	00011	10011
4	10100	00100
5	00101	10101
6	00110	10110
7	10111	00111
8	11000	01000
9	01001	11001

### 3.24

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

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

### 3.25

<u>A B</u>	<u>Y = A ⊕ B</u>
0 0	0
0 1	1
1 0	1
1 1	0

<u>C D</u>	<u>Z = C ⊕ D</u>
0 0	0
0 1	1
1 0	1
1 1	0

<u>y</u>	<u>z</u>	<u>x = y ⊕ z</u>
0	0	0
0	1	1 ←
1	0	1 ←
1	1	0

$$1 \leftarrow \begin{cases} AB=00 \text{ or } 11 \\ CD=01 \text{ or } 10 \end{cases}$$

$$1 \leftarrow \begin{cases} AB=01 \text{ or } 10 \\ CD=00 \text{ or } 11 \end{cases}$$

### ABCD

0001, 0010, 1101, 1110

0100, 0111, 1000, 1011

Always odd number of 1's

### 3.26

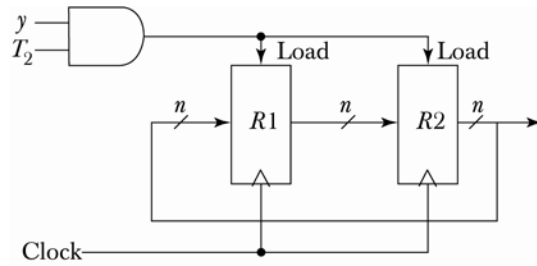
Same as in Fig. 3.3 but without the complemented circles in the outputs of the gates.

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

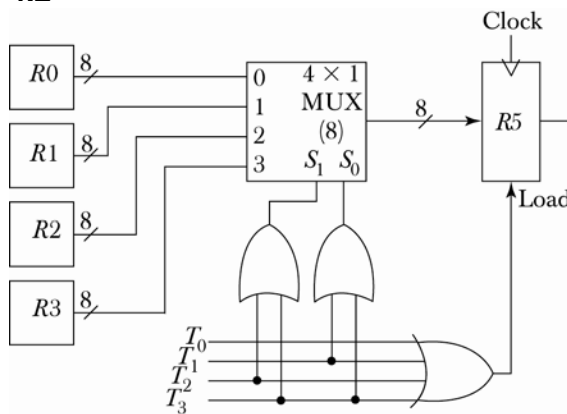
$$\text{Error} = x \oplus y \oplus z \oplus P$$

## CHAPTER 4

### 4.1



### 4.2



$T_0$	$T_1$	$T_2$	$T_3$	$S_1$	$S_0$	$R_3$	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	

$$S_1 = T_2 + T_3$$

$$S_0 = T_1 + T_3$$

$$\text{load} = T_0 + T_1 + T_2 + T_3$$

### 4.3

P:  $R1 \leftarrow R2$

P'Q:  $R1 \leftarrow R3$

### 4.4

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

Provide a "load" control input in each register.

Provide a clock input for each register.

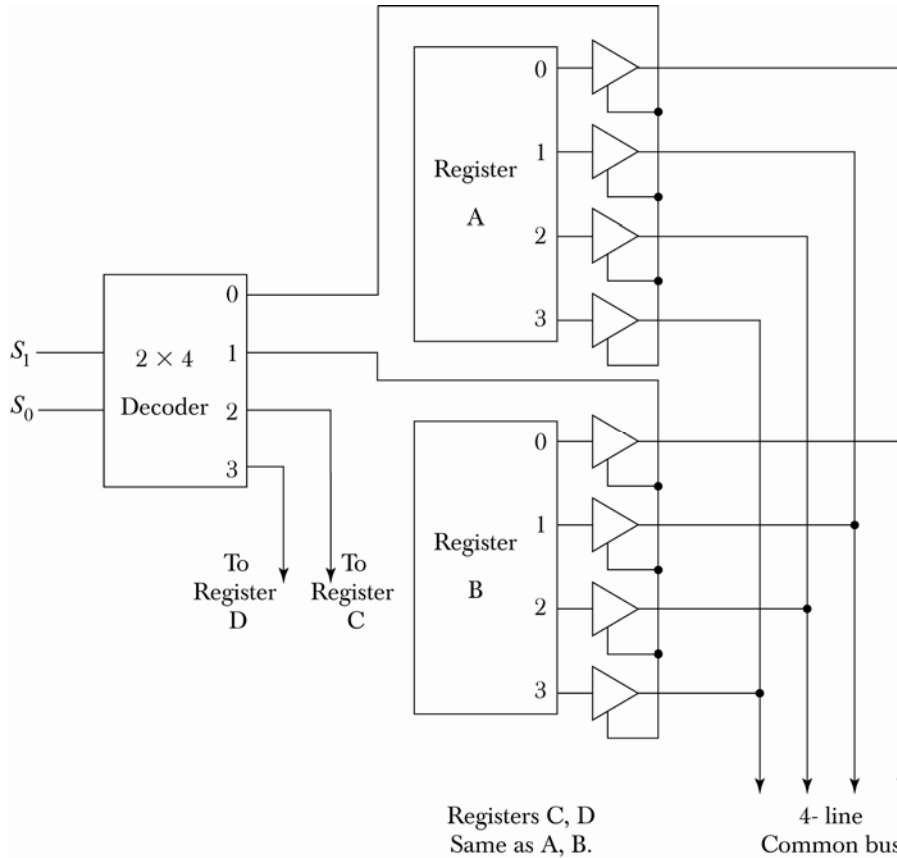
To transfer from register C to register A:

Apply  $S_1S_0 = 10$  (to select C for the bus.)

Enable the load input of A

Apply a clock pulse.

#### 4.5



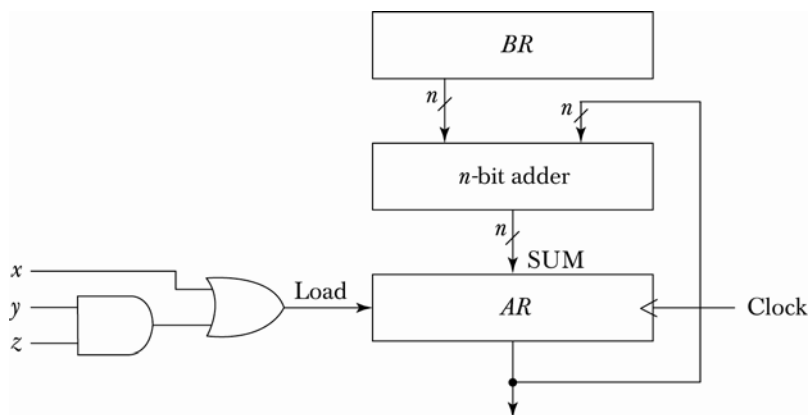
#### 4.6

- 4 selection lines to select one of 16 registers.
- $16 \times 1$  multiplexers.
- 32 multiplexers, one for each bit of the registers.

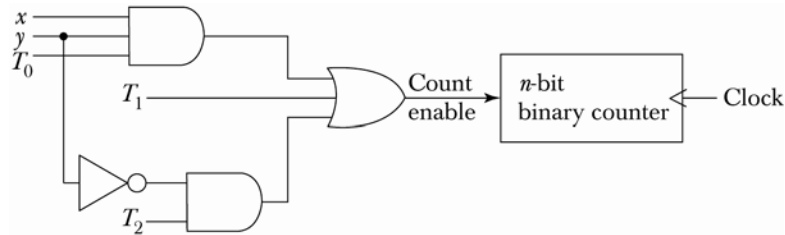
#### 4.7

- Read memory word specified by the address in AR into register R2.
- Write content of register R3 into the memory word specified by the address in AR.
- Read memory word specified by the address in R5 and transfer content to R5 (destroys previous value)

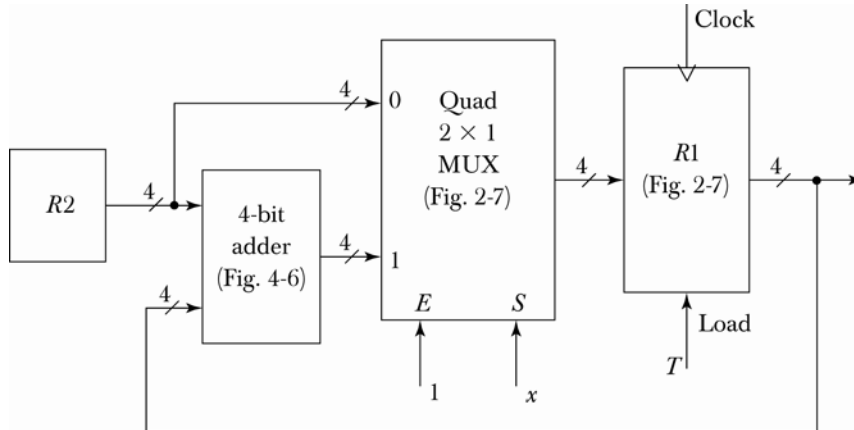
#### 4.8



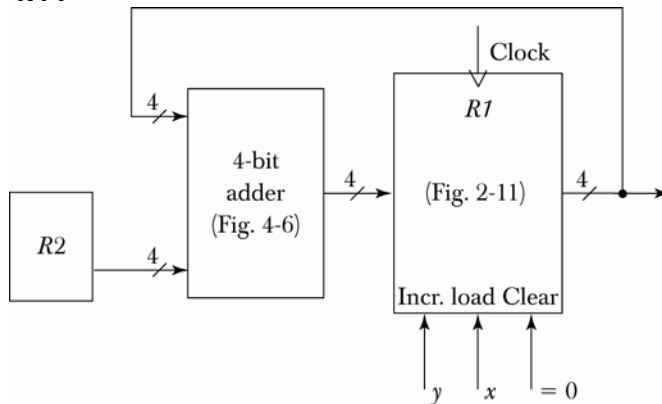
4.9



4.10



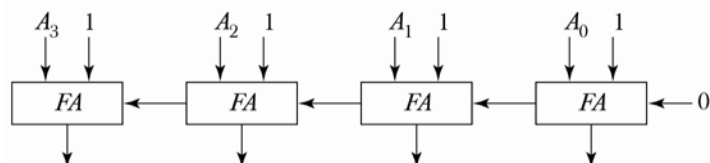
4.11



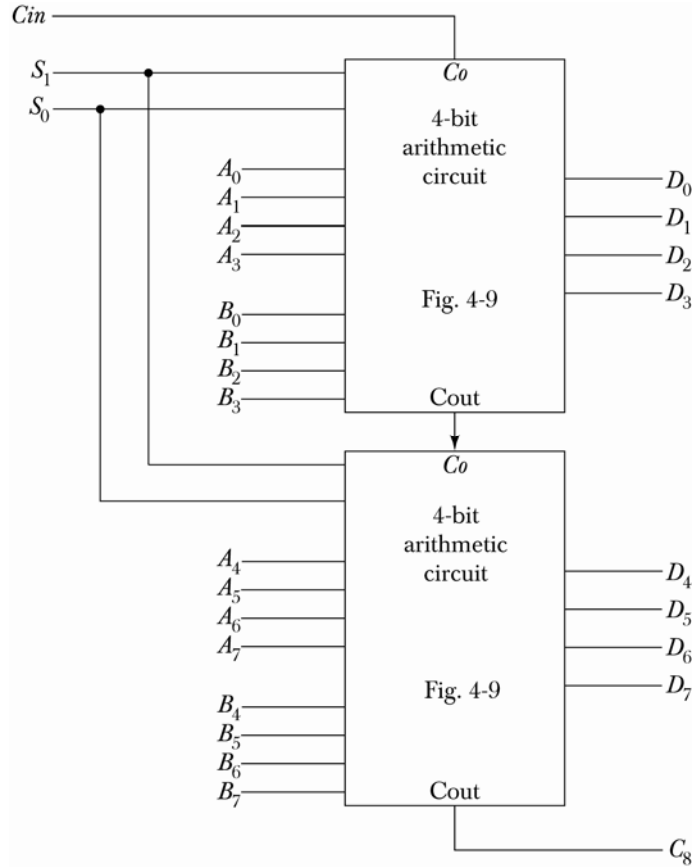
4.12

<u>M</u>	<u>A</u>	<u>B</u>	<u>Sum</u>	<u>Cu</u>	
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  $A - 1 = A + 2\text{'s complement of } 1 = A + 1111$

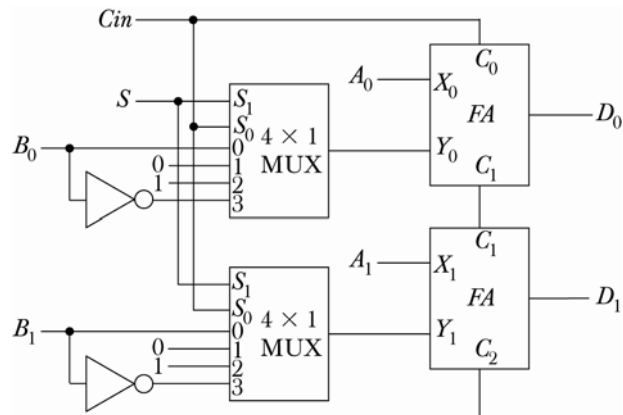


#### 4.14

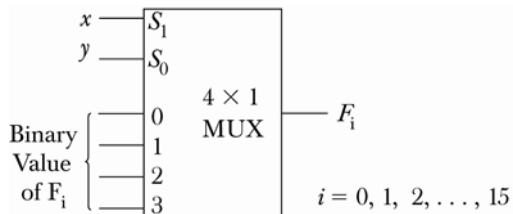


#### 4.15

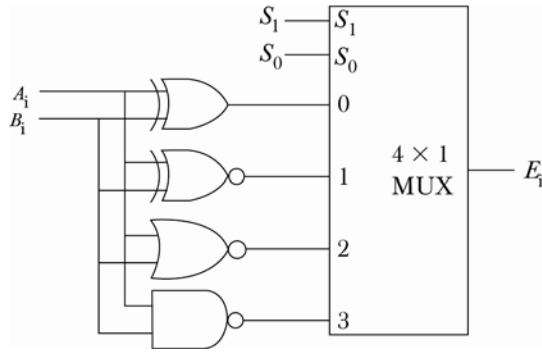
S	Cin	X	Y	
0	0	A	B	(A + B)
0	1	A	0	(A + 1)
1	0	A	1	(A - 1)
1	1	A	$\bar{B}$	(A - B)



#### 4.16



#### 4.17



#### 4.18

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

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

#### 4.19

(a)  $AR = 11110010$   
 $BR = 11111111(+)$   
 $AR = 11110001$   $BR = 11111111$   $CR = 10111001$   $DR = 1110$   
 1010

(b)  $CR = 10111001$   $BR = 1111 1111$   
 $DR = 11101010$  (AND)  $+1$   
 $CR = 10101000$   $BR = 0000 0000$   $AR = 1111 0001$   $DR = 11101010$

(c)  $AR = 11110001$   $(-1)$   
 $CR = 10101000$   
 $AR = 01001001$ ;  $BR = 00000000$ ;  $CR = 10101000$ ;  $DR = 11101010$

#### 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

 $A_0 A_1 A_2 A_3 I_L$ 

$$\begin{array}{ccccccc}
 & & 1 & 0 & 0 & 1 & 0 \\
 & \swarrow & \swarrow & \swarrow & \swarrow & & \\
 H = & 0 & 0 & 1 & 0 & & \text{shift left}
 \end{array}$$

**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 ( $R_1$ ) 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

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

$$64 = 2^6$$

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

- (b) 1 7 6 18 = 32 bits

I	opcode	Register	Address
---	--------	----------	---------

- (c) Data; 32 bits; address: 18 bits.

### 5.2

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

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

### 5.3

- (a) Memory read to bus and load to IR:  $IR \leftarrow M[AR]$   
 (b) TR to bus and load to PC:  $PC \leftarrow TR$   
 (c) AC to bus, write to memory, and load to DR:  
 $DR \leftarrow AC, M[AR] \leftarrow AC$   
 (d) Add DR (or INPR) to AC:  $AC \leftarrow AC + DR$

### 5.4

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

### 5.5

- (a)  $IR \leftarrow M[PC]$  PC cannot provide address to memory. Address must be transferred to AR first  
 $AR \leftarrow PC$   
 $IR \leftarrow M[AR]$   
 (b)  $AC \leftarrow AC + TR$  Add operation must be done with DR. Transfer TR to DR first.  
 $DR \leftarrow TR$   
 $AC \leftarrow AC + DR$

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

AC  $\leftarrow$  DR, DR  $\leftarrow$  AC      (See answer to Problem 5.4(d))  
 AC  $\leftarrow$  AC + DR  
 AC  $\leftarrow$  DR, DR  $\leftarrow$  AC

## 5.6

(a)  $\begin{array}{r} 0001 \\ \text{ADD} \end{array} \quad \begin{array}{r} 0000 \quad 0010 \quad 0010 \\ (024)_{16} \end{array} = (1024)_{16}$   
 ADD content of M[024] to AC      ADD 024

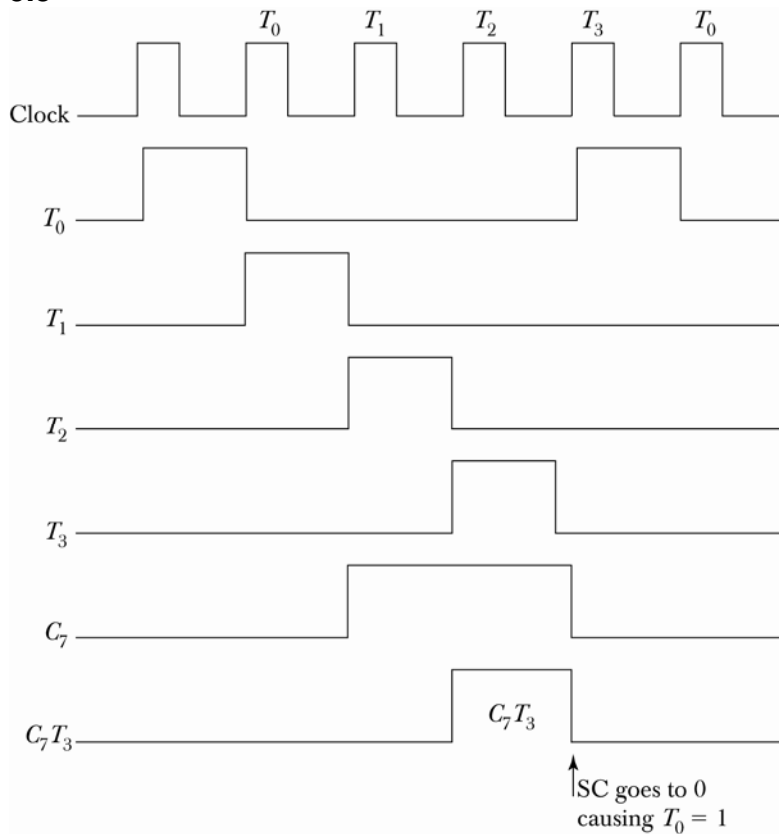
(b)  $\frac{1\ 011\ 0001\ 0010\ 0100}{1\ \text{STA} \quad (124)_6} = (B124)_{16}$   
Store AC in M[M[124]]      STA I 124

(c)  $\frac{0111}{\text{Register}} \quad \frac{0000 \ 0010 \ 0000}{\text{Increment AC}} = (7020)_{16}$   
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 <sub>0</sub>	7FF	7FF	—	—	1
T <sub>1</sub>	800	7FF	—	EA9F	2
T <sub>2</sub>	800	A9F	—	EA9F	3
T <sub>3</sub>	800	C35	—	EA9F	4
T <sub>4</sub>	800	C35	FFFF	EA9F	5
T <sub>5</sub>	800	C35	0000	EA9F	6
T <sub>6</sub>	801	C35	0000	EA9F	0

## 5.12

(a) 9 = (1001)

1|001/  
I=1 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 = 3BO      IR = 932E  
AR = 7AC      E = 1  
DR = 8B9F      I = 1  
AC = 0A62      SC = 0000

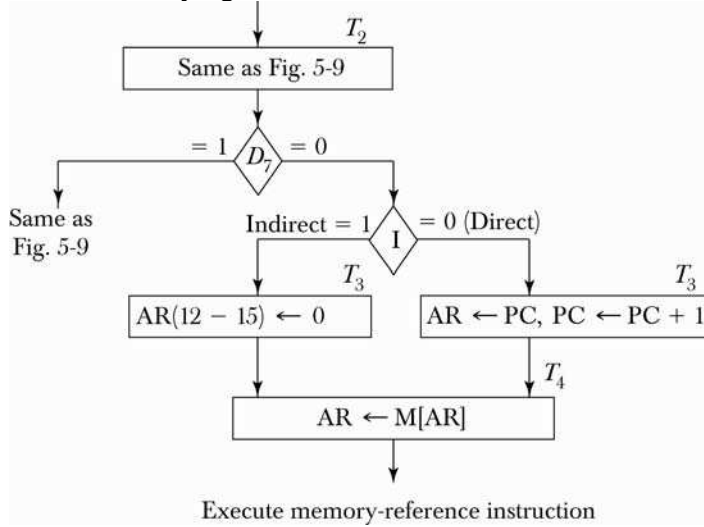
### 5.13

<u>XOR</u>	D <sub>0</sub> T <sub>4</sub> :	DR ← M[AR]
	D <sub>0</sub> T <sub>5</sub> :	AC ← AC ⊕ DR, SC ← 0
<hr/>		
<u>ADM</u>	D <sub>1</sub> T <sub>4</sub> :	DR ← M[AR]
	D <sub>1</sub> T <sub>5</sub> :	DR ← AC, AC ← AC + DR
	D <sub>1</sub> T <sub>6</sub> :	M[AR] ← AC, AC ← DR, SC ← 0
<hr/>		
<u>SUB</u>	D <sub>2</sub> T <sub>4</sub> :	DR ← M[AR]
	D <sub>2</sub> T <sub>5</sub> :	DR ← AC, AC ← DR
	D <sub>2</sub> T <sub>6</sub> :	AC ← $\overline{AC}$
	D <sub>2</sub> T <sub>7</sub> :	AC ← AC + 1
	D <sub>2</sub> T <sub>8</sub> :	AC ← AC + DR, SC ← 0
<hr/>		
<u>XCH</u>	D <sub>3</sub> T <sub>4</sub> :	DR ← M[AR]
	D <sub>3</sub> T <sub>5</sub> :	M[AR] ← AC, AC ← DR, SC ← 0
<hr/>		
<u>SEQ</u>	D <sub>4</sub> T <sub>4</sub> :	DR ← M[AR]
	D <sub>4</sub> T <sub>5</sub> :	TR ← AC, AC ← AC ⊕ DR
	D <sub>4</sub> T <sub>6</sub> :	If (AC = 0) then (PC ← PC + 1), AC ← TR, SC ← 0
<hr/>		
<u>BPA</u>	D <sub>5</sub> T <sub>4</sub> :	If (AC = 0 ∧ AC (15) = 0) then (PC ← AR), SC ← 0
<hr/>		

### 5.14

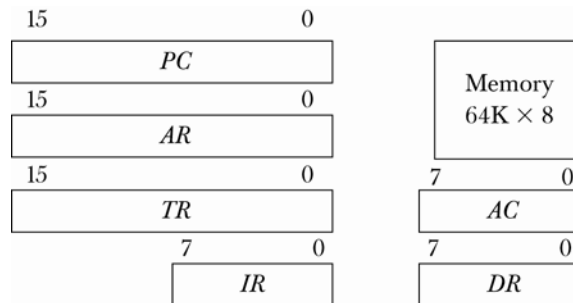
Converts the ISZ instruction from a memory-reference instruction to a register-reference instruction. The new instruction ICSZ can be executed at time T<sub>3</sub> instead of time T<sub>6</sub>, a saving of 3 clock cycles.

**5.15** Modify fig. 5.9.

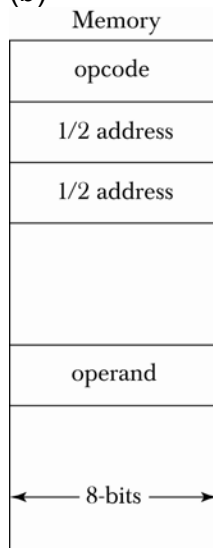


**5.16**

(a)



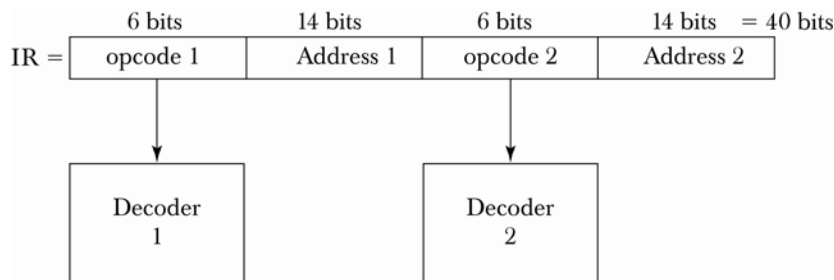
(b)



(c)  $T_0$ :  $IR \leftarrow M(PC), PC \leftarrow PC + 1$

$$T_1: \quad AR(0-7) \leftarrow M[PC], PC \leftarrow PC + 1$$
$$T_2: \quad AR(8-15) \leftarrow M[PC], PC \leftarrow PC + 1$$
$$T_3: \quad DR \leftarrow M[AR]$$

## 5.17



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.

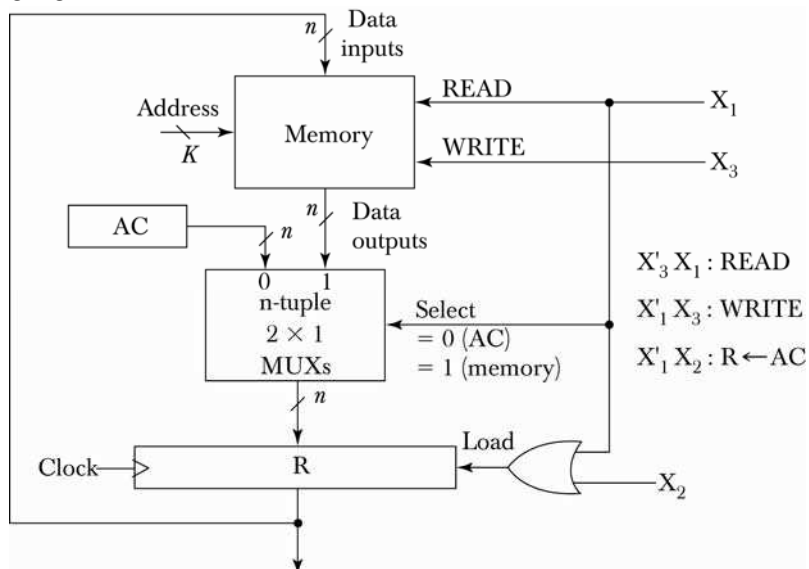
## 5.18

(a) BUN 2300

(b) ION

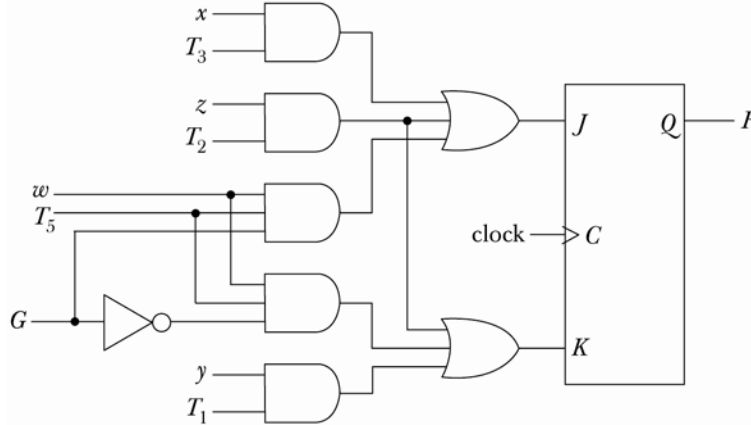
BUN 0 I (Branch indirect with address 0)

## 5.19



### 5.20

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



### 5.21

From Table 5.6: ( $Z_{DR} = 1$  if  $DR = 0$  ;  $Z_{AC} = 1$ , if  $AC = 0$  )

$$\text{INR}(\text{PC}) = R'T_1 + RT_7 + D_6T_6Z_{DR} + PB_9(\text{FGI}) + PB_8(\text{FGO}) \\ + rB_4 + (AC_{15})' + rB_3(AC_{15}) + rB_2Z_{AC} + rB_1E'$$

$$\text{LD}(\text{PC}) = D_4T_4 + D_5T_5$$

$$\text{CLR}(\text{PC}) = RT_1$$

The logic diagram is similar to the one in Fig. 5.16.

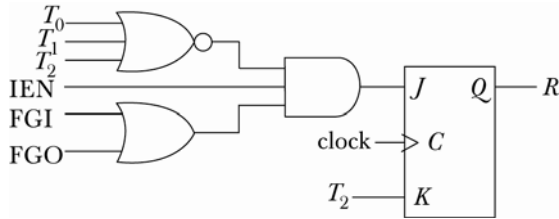
### 5.22

$$\text{Write} = D_3T_4 + D_5T_4 + D_6T_6 + RT_1$$

$$(M[\text{AR}] \leftarrow \text{xx})$$

### 5.23

$$\begin{array}{ll} (T_0 + T_1 + T_2)' (\text{IEN}) (\text{FGI} + \text{FGO}) & : \quad R \leftarrow 1 \\ RT_2 & : \quad R \leftarrow 0 \end{array}$$



### 5.24

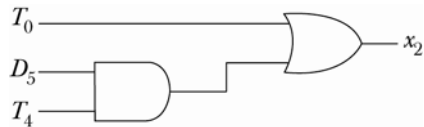
$X_2$  places PC onto the bus. From Table 5.6:

$$R'T_0: \text{AR} \leftarrow \text{PC}$$

$$RT_0: \text{TR} \leftarrow \text{PC}$$

$$D_5T_4: M[\text{AR}] \leftarrow \text{PC}$$

$$X_2 = R'T_0 + RT_0 + D_5T_4 = (R' + R)T_0 + D_5T_4 = T_0 + D_5T_4$$

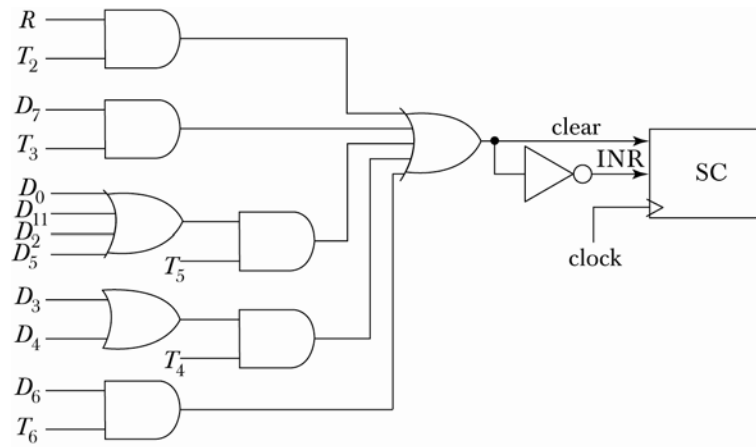




### 5.25

From Table 5.6:

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



## CHAPTER 6

### 6.1

			<u>AC</u>	<u>PC</u>	<u>IR</u>	
010	CLA		0000	011	7800	
011	ADD	016	C1A5	012	1016	
012	BUN	014	C1A5	014	4014	
013	HLT		8184	014	7001	
014	AND	017	8184	015	0017	
015	BUN	013	8184	013	4013	
016	C1A5					
017	93C6					
	$(C1A5)_{16}$	=	1100	0001	1010	0101
	$(93C6)_{16}$	=	1001	0011	1100	0110
			1000	0001	1000	0100
						= $(8184)_{16}$

### 6.2

			<u>Ac</u>	
100	5103	BSA 103		
101	7200	CMA	FFFE A	← Answer
102	7001	HLT		
103	0000	5101		← Answer
104	7800	CLA	0000	
105	7020	INC	0001	
106	C103	BUN 103 I		

### 6.3

CLA  
 STA SUM } SUM=0  
 LDA SUM  
 ADD A } SUM=SUM + A + B  
 ADD 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

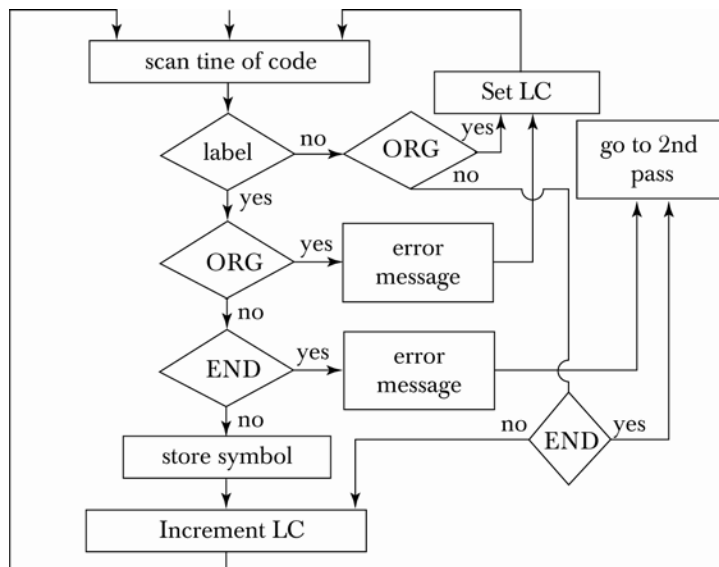
#### 6.4

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 OD	0011 0101 0000 1101

(b)  $(35)_{10} = (0000\ 0000\ 0010\ 0011)_2$   
 $-35 \rightarrow 1111\ 1111\ 1101\ 1101 = (FFDD)_{16}$

#### 6.7

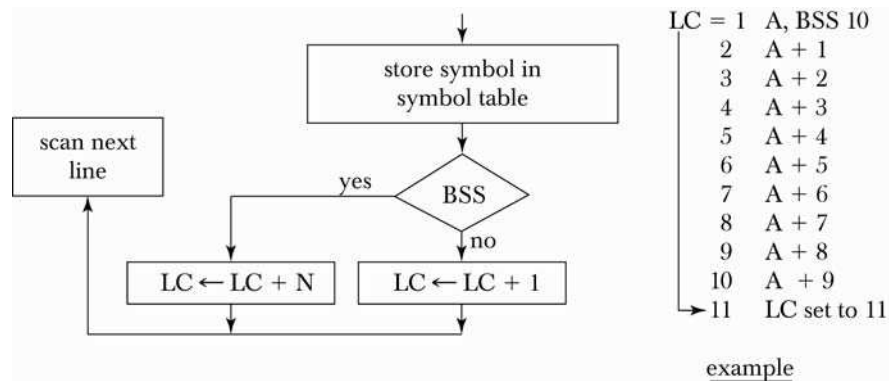
(a)		
LOP	105	$(100)_{10} = (0000\ 0000\ 0110\ 0100)_2$
ADS	10B	
PTR	10C	$(-100)_{10} = (1111\ 1111\ 1001\ 1100)_2 = (FF9C)_{16}$
NBR	10D	
CTR	10E	$(75)_{10} = (0000\ 0000\ 0100\ 1011)_2 = (0048)_{16}$
SUM	10F	$(23)_{10} = (0000\ 0000\ 0001\ 0111)_2 = (0017)_{16}$

(b)

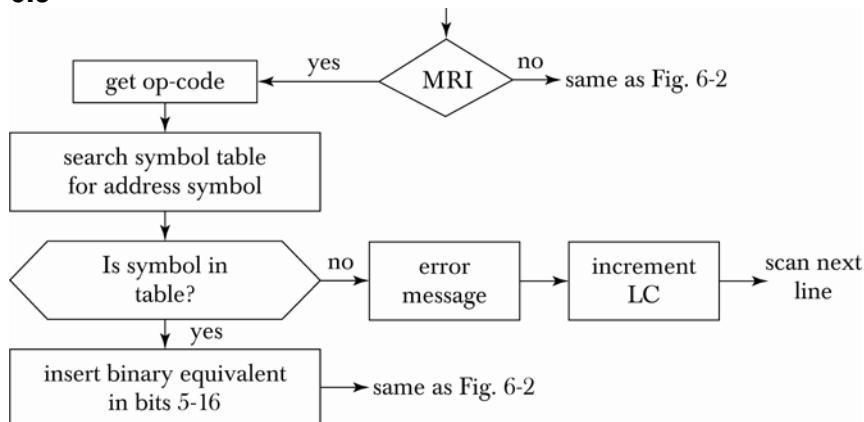
Loc	Hex	ORG	100	Loc	Hex	
100	210B	LDA	ADS	10B	0150	ADS, HEX 150
101	310C	STA	PTR	10C	0000	PTR, HEX 0
102	210D	LDA	NBR	10D	FF9C	NBR, DEC-100
103	310E	STA	CTR	10E	0000	CTR, HEX 0
104	7800	CLA		10F	0000	SJH, HEX 0
105	910C	LOP, ADD	PTR I			ORG 150
106	610C	ISZ	PTR	150	004B	DEC 150
107	610E	ISZ	CTR	:	:	: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

## 6.8

Modify flow chart of Fig. 6.1



## 6.9



## 6.10

### (a) MRI Table

	<u>Memory word</u>	<u>Symbol</u>	<u>HEX</u>
AND	1	A N	41 4D
	2	D Space	44 20
	3	value	00 00
ADD	4	A D	41 44
	5	D space	44 20
	6	value	10 00
		etc.	

### (b) non - MRI Table

	<u>Memory word</u>	<u>Symbol</u>	<u>HEX</u>
CLA	1	C L	43 4C
	2	A Space	41 20
	3	value	78 00
CLE	4	C L	43 4C
	5	E space	45 20
	6	value	74 00
		etc.	

## 6.11

LDA	B	
CMA		
INC		
ADD	A	/Form A-B
SPA		/skip if AC positive
BUN	N10	/(A-B) < 0, go to N 10
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
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	O	
111	62C1	WRD, HEX	62C1	
		END		

### 6.13

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

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

### 6.15

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

We need to initialize CTR and P as follows:

```

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

### 6.16

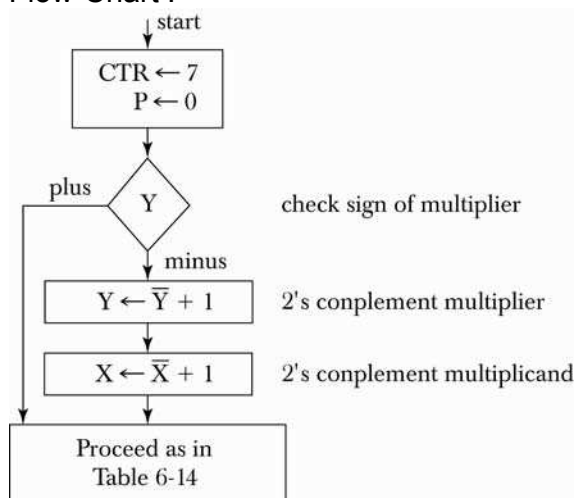
Multiplicand is initially in location XL. Will be shifted left into XH (which has zero initially). The partial product will contain two locations PL and PH (initially zero). Multiplier is in location Y. CTR = -16

LOP,	CLE		}	Same as beginning of program in Table 6.14
	LDA	Y		
	CIR			
	STA	Y		
	SZE		}	Double-precision add $P \leftarrow X + P$ Same as program In Table 6.15
	BUN	ONE		
	BUN	ZRO		
ONE,	LDA	XL		
	ADD	PL	}	Double-precision left-shift $XH + XL$
	STA	PL		
	CLA			
	CIL			
	ADD	XH	}	Repeat 16 times.
	ADD	PH		
	STA	PH		
	CLE			
ZRO,	LDA	XL	}	Repeat 16 times.
	CIL			
	STA	XL		
	LDA	XH		
	CIL		}	Repeat 16 times.
	STA	XH		
	ISZ	CTR		
	BUN	LOP		
	HLT			

### 6.17

If multiplier is negative, take the 2's complement of multiplier and multiplicand and then proceed as in Table 6.14 (with  $CTR = -7$ ).

Flow-Chart :





### 6.18

```

C ← A - B
  CLE
  LDA BL
  CMA
  INC
  ADD AL
  STA AL
  CLA
  CIL
  STA TMP
  LDA BH
  CMA
  ADD AH
  ADD TMP
  STA CH
  HLT
  TMP, HEX 0
  
```

Save Carry

Add carry →

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

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

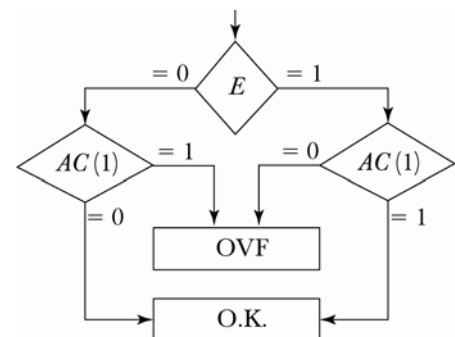
```

LDA      Y
CMA
AND      X      AND TMP
CMA
STA      Z
STA      TMP    HLT
LDA      X      X, ---
CMA
AND      Y      Y, ---
CMA
AND      Y      Z, ---
CMA
            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
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
IN
ISZ SUB
ADD SUB I
ISZ SUB
STA SUB
ISZ SUB
BUN SUB I
```

## 6.22

### Calling Program

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

```
CMA
INC
STA CTR
LOP, LDA PTR I
CMA
```

### Subroutine

```
CMP, HEX 0
LDA CMP I
STA PTR
ISZ CMP
LDA CMP I
STA PIR I
ISZ PTR
ISZ CTR
BUN LOP
ISZ CMP
BUN CMP I
PTR, ---
CTR, ---
```

## 6.23

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

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

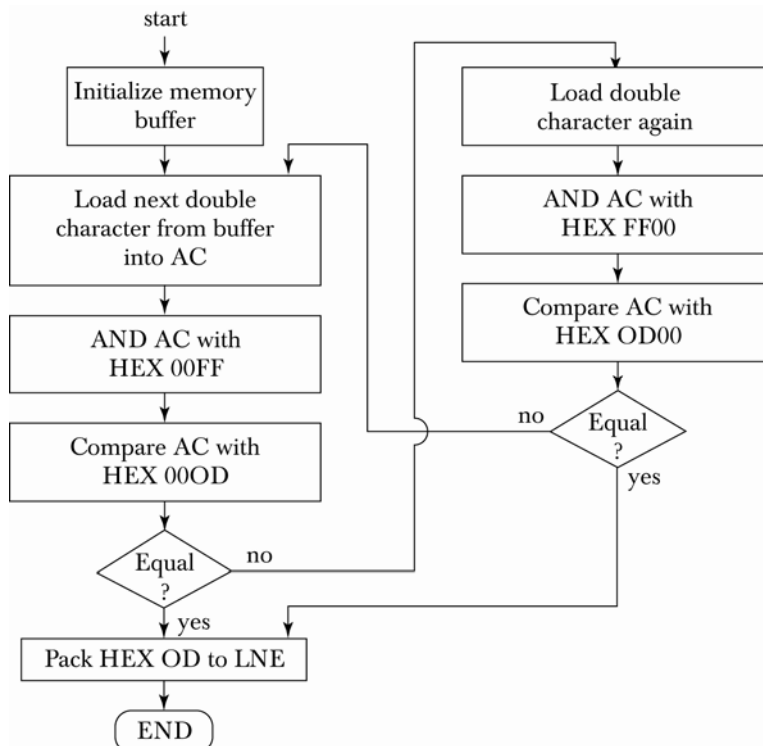
## 6.24

LDA	ADS		BUN	LOP
STA	PTR		HTA	
LDA	NBR		ADS, HEX	400
STA	CTR		PTR, HEX	0
LOP,	BSA	IN2	/subroutine Table 6.20	
	STA	PTR I	NBR, DEC	-512
	ISZ	PTR	CTR, HEX	0
	ISZ	CTR		

## 6.25

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

## 6.26



## 6.27

### Location      Hex code

200	3213
201	7080
202	3214
203	F200
204	4209
205	F800
206	F400
207	B215
208	6215
209	F100
20A	420E
20B	A216
20C	F400
20D	6216
20E	2214
20F	7040
210	2213
211	F080
212	C000
213	0000
214	0000
215	0000
216	0000

SRV,	STA	SAC
	CIR	
	STA	SE
	SKI	
	BUN	NXT
	INP	
	OUT	
	STA	PT1 I
	ISZ	PT1
NXT,	SKO	
	BUN	EXT
	LDA	PT2 I
	OUT	
	ISZ	PT2
EXT,	LDA	SE
	CIL	
	LDA	SAC
	ION	
	BUN	ZR0 I
SAC,	---	
SE,	---	
PT1,	---	
PT2,	---	

## 6.28

SRV,	STA	SAC
	CIR	
	STA	SE
	LDA	MOD /check MOD
	CMA	
	SZA	
	BUN	NXT /MOD ≠ all 1's
	SKI	
	BUN	NXT
	INP	
	OUT	
	STA	PT1 I
	ISZ	PT1
	BUN	EXT /MOD ≠ 0

service  
out put  
device

service  
input  
device

NXT,	LDA	MOD
	SZA	
	BUN	EXT
	SKO	
	BUN	EXT
	LDA	PT2 I
	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

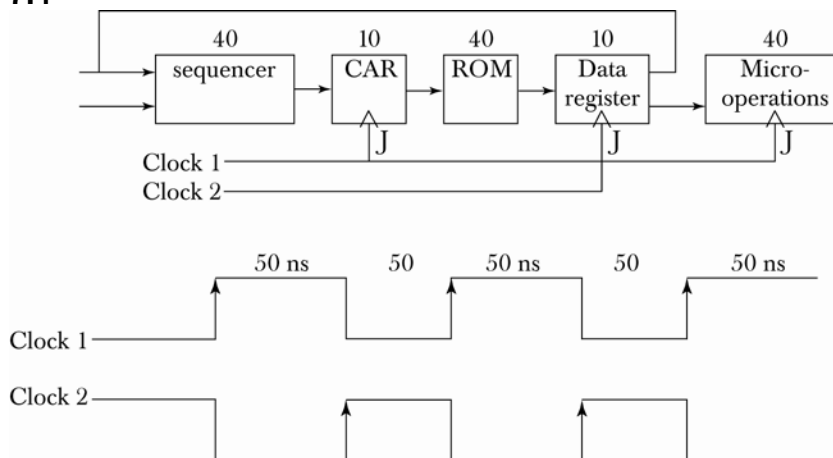
Micro operation - an elementary digital computer operation.

Micro instruction - an instruction stored in control memory.

Micro program - a sequence of microinstructions.

Micro code - same as microprogram.

### 7.4



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

If the data register is removed, we can use a single phase

$$\text{clock with a frequency of } \frac{1}{90 \times 10^{-9}} = 11.1 \text{ MHz}.$$

### 7.5

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

$$(a) \quad \begin{matrix} 6 & 10 & 16 \end{matrix} = 32 \text{ bits}$$

Select	Address	Micro operations
--------	---------	------------------

(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

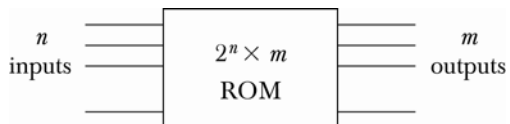
- (a) 0001000 = 8
- (b) 0101100 = 44
- (c) 0111100 = 60

### 7.8

opcode = 6 bits  
control memory  
address = 11 bits

$$00 \left| \begin{array}{c} \times \times \times \times \times \times \\ \times \times \times \times \times \times \end{array} \right| 000$$

### 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

	<u>F1</u>	<u>F2</u>	<u>F3</u>	
(a)	011	110	000	INCAC INCDR NOP
(b)	000	100	101	NOP READ INCPC
(c)	100	101	000	DRTAC ACTDR NOP

### 7.12

			<u>Binary</u>
(a)	READ DRTAC	DR ← M[AR] AC ← DR	F2 = 100 F3 = 101 001 100 101
(b)	ACTDR DRTAC	DR ← AC AC ← DR	F2 = 101 F1 = 100 000 100 101

(c)	ARTPC	PC $\leftarrow$ AR	F3 = 110	Impossible. Both use F1
	DRTAC	AC $\leftarrow$ DR	F1 = 100	
	WRITE	M[AR] $\leftarrow$ 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.

INDR 2 :	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	INDR CTS
	61	:	WRITE, READ	I	CALL	FETCH
	62	:	ADD, SUB	S	RET	63(NEXT)
	63	:	DRTAC, INC DR	Z	MAP	60

(b)

60	:	Cannot increment and complement AC at the same time. With a JMP to INDRCT, control does not return to 61.
61	:	Cannot read and write at the same time. The CALL behaves as a JMP since there is no return from FETCH.
62	:	Cannot add and subtract at the same time. The RET will be executed independent of S.
63	:	The MAP is executed irrespective of Z or 60.

### 7.16

	ORG 16			
AND :	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
ANDOP :	AND	U	JMP	FETCH

	ORG 20			
SUB :	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	SUB	U	JMP	FETCH

	ORG 24			
ADM :	NOP	I	CALL	INDRCT

	READ	U	JMP	NEXT
	DRTAC, ACTDR	U	JMP	NEXT
	ADD	U	JMP	EXCHANGE +2

(Table 7.2)

	ORG 28			
BICL :	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			
BEQ 1 :	DRTAC, ACTDR	Z	JMP	EQUAL
	NOP	U	JMP	FETCH
EQUAL :	INC PC	U	JPM	FETCH

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

## 7.17

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

## 7.18

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

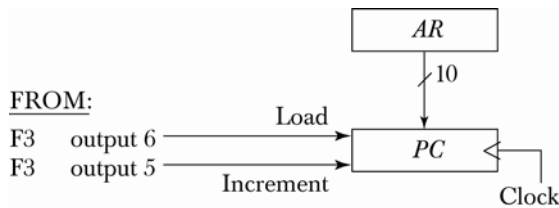


### 7.19

From Table 7.1 :

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

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



### 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.

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

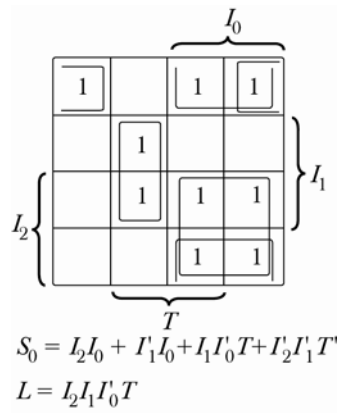
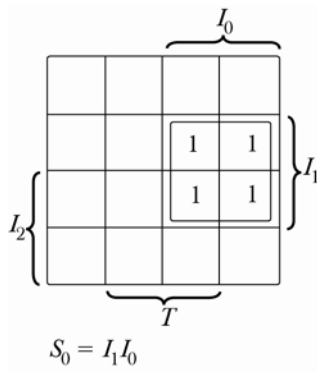
4	4	4	5	3	=	20 bits
SRC 1	SRC 2	DEST	ALU	SHIFT		

- (c)

R5	R6	R4	ADD	SHIFT	
0101	0110	0100	00100	000	$R4 \leftarrow R5 + R6$

### 7.22

$I_2$	$I_1$	$I_0$	T	$S_1$	$S_0$	L	
0	0	0	0	0	1	0	AD1
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(a)
0	1	1	1	1	0	0	RET(a)
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)



## 7.23

## CHAPTER 8

### 8.1

- (a) 32 multiplexers, each of size  $16 \times 1$ .  
 (b) 4 inputs each, to select one of 16 registers.  
 (c) 4-to-16 – line decoder  
 (d)  $32 + 32 + 1 = 65$  data input lines  
 $32 + 1 = 33$  data output lines.  
 (e) 4 4 4 6 = 18 bits

SELA	SELB	SELD	OPR
------	------	------	-----

### 8.2

$30 + 80 + 10 = 120$  n sec.

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

### 8.3

		SELA	SELB	SELD	OPR	Control word
(a)	$R1 \leftarrow R2 + R3$	R2	R3	R1	ADD	010 011 001 00010
(b)	$R4 \leftarrow \overline{R4}$	R4	—	R4	COMA	100 xxx 100 01110
(c)	$R5 \leftarrow R5 - 1$	R5	—	R5	DECA	101 xxx 101 00110
(d)	$R6 \leftarrow SH1 R1$	R1	—	R6	SHLA	001 xxx 110 11000
(e)	$R7 \leftarrow \text{Input}$	Input	—	R7	TSFA	000 xxx 111 00000

### 8.4

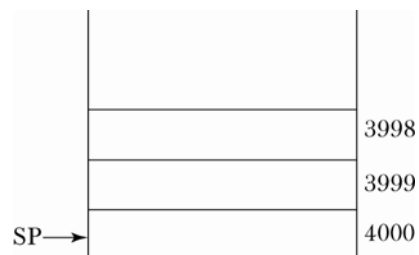
	Control word	SELA	SELB	SELD	OPR	Microoperation
(a)	001 010 011 00101	R1	R2	R3	SUB	$R3 \leftarrow R1 - R2$
(b)	000 000 000 00000	Input	Input	None	TSFA	$\text{Output} \leftarrow \text{Input}$
(c)	010 010 010 01100	R2	R2	R2	XOR	$R2 \leftarrow R2 \oplus R2$
(d)	000 001 000 00010	Input	R1	None	ADD	$\text{Output} \leftarrow \text{Input} + R1$
(e)	111 100 011 10000	R7	R4	R3	SHRA	$R3 \leftarrow \text{shr}R7$

### 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

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

$$(b) \quad A + B \frac{C}{D * E}$$

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

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

### 8.9

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

$$\text{RPN : } 34 + 26 + 10 * 8 + *$$

				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) :

$M[WC] \leftarrow DR$

$WC \leftarrow WC + 1$

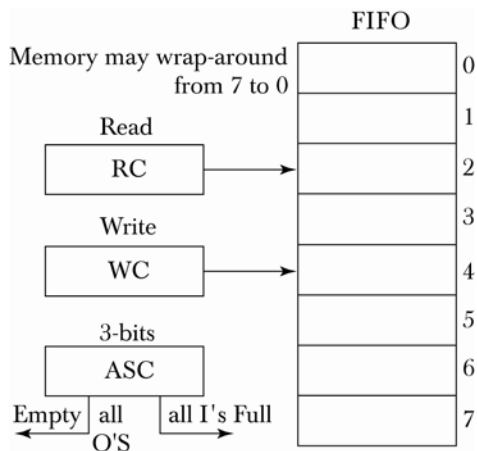
$ASC \leftarrow ASC + 1$

READ : (if not empty)

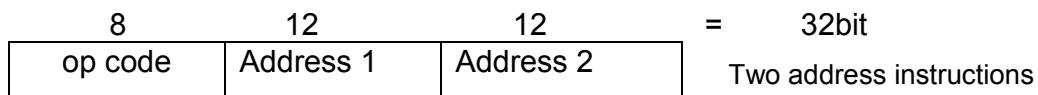
$DR \leftarrow M[RC]$

$RC \leftarrow RC + 1$

$ASC \leftarrow ASC - 1$



### 8.11



$$2^8 = 256 \text{ combinations.}$$

$$256 - 250 = 6 \text{ combinations can be used for one address}$$

op code	Address	One address instructor
$6 \times 2^{12}$		

Maximum number of one address instruction:  
 $= 6 \times 2^{12} = 24,576$

### 8.12

(d) RPN:  $\times AB - C + DE \times F - \times GHK \times + /=$

### 8.13

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

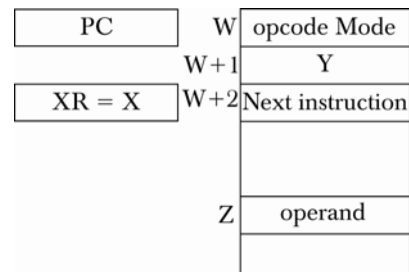
op code	Mode	Register	Address	
5	3	6	18	= 32

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

### 8.14

Z = Effective address

- (a) Direct:  $Z = Y$   
(b) Indirect:  $Z = M[Y]$   
(c) Relative:  $Z = Y + W + 2$   
(d) Indexed:  $Z = Y + X$



### 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.

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

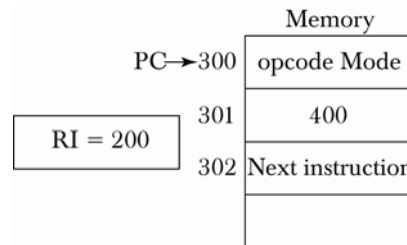
### 8.17

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  
 10101010    AND   10101010    OR   10101010    XOR  
 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 (over flow)

ROR: 10111101

ROL: 11110110

RORC: 10111101

ROLC: 11110111

### 8.23

+ 83 = 01010011 - 83 = 10101101

+ 68 = 01000100 - 68 = 10111100

- (a) - 83    10101101
- + 68    +01000100
- 15    11110001
- (in 2's complement)



### 8.27

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

Sign  $S = 0$  if no over flow (positive)

or  $S = 1$  if over flow (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  $V = 1$   $(S \oplus V) = 1$

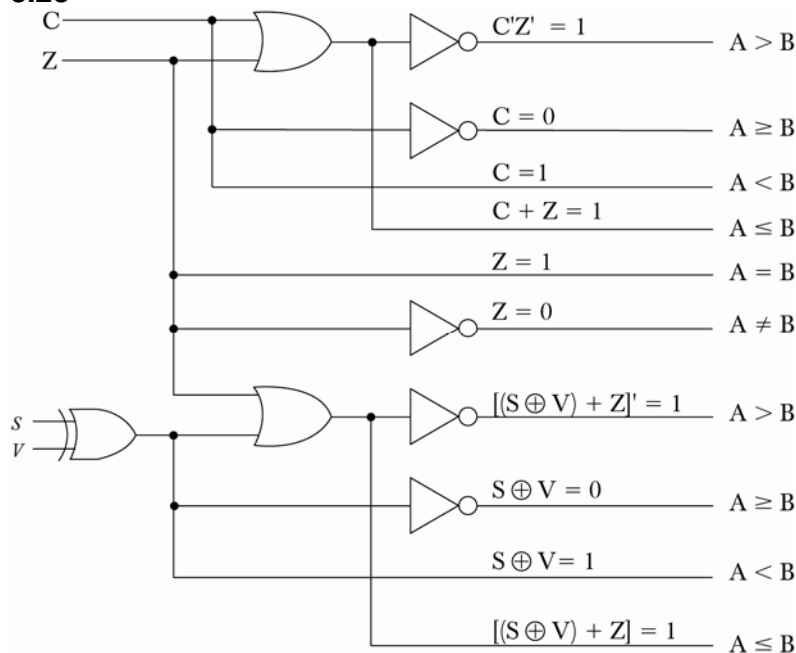
$A > B$  Implies  $A \geq B$  but not  $A = B$

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

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

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

### 8.28



### 8.29

	<u>Unsigned</u>	<u>Signed</u>
$A = 01000001$	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)

$$\begin{array}{rcl}
 A = 01000001 & = & +65 \\
 B = 10000100 & = & -124 \\
 \hline
 A - B = 10111101 & +189 & = \underline{010111101} \\
 & & \text{9 bits}
 \end{array}$$

(b) S = 1 (sign reversal) +189 &gt; 127

Z = 0

V = 1 (overflow) 65 &gt; -124

A &gt; B

(c) BGT, BGE, BNE

**8.32**

	<u>PC</u>	<u>SP</u>	<u>Top of Stack</u>
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.	(a) 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 × 8 + 10 = 138

Computer 2: (8 + 8) 4 + 8 = 16 × 4 + 8 = 72

Computer 3: (16 + 16) 16 + 16 = 32 × 16 + 16 = 528

**8-38**

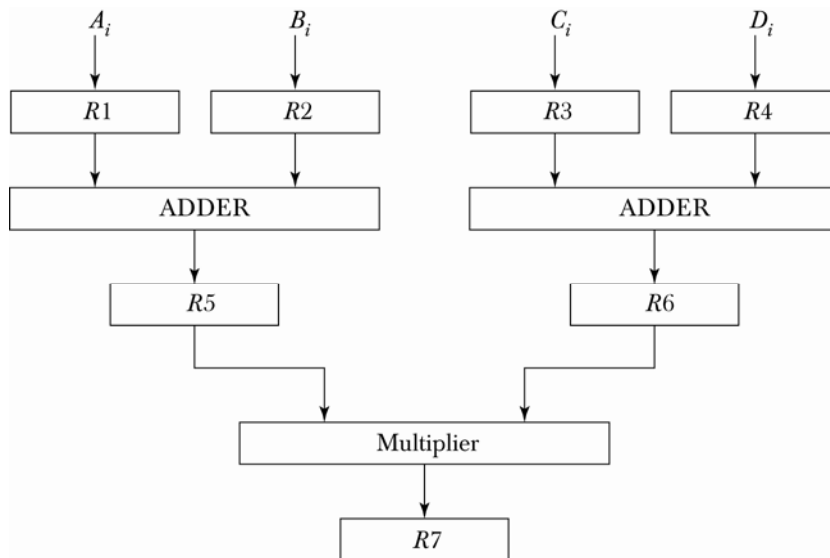
- |                        |   |
|------------------------|---|
| (a) SUB R22, # 1, R22  | $R22 \leftarrow R22 - 1$ (Subtract 1)                         |
| (b) XOR R22, # -1, R22 | $R22 \leftarrow R22 \oplus \text{all 1's } (x \oplus 1 = x')$ |
| (c) SUB R0, R22, R22   | $R22 \leftarrow 0 - R22$                                      |
| (d) ADD R0, R0, R22    | $R22 \leftarrow 0 + 0$  |
| (e) SRA R22, # 2, R22  | Arithmetic shift right twice                                  |
| (f) OR R1, R1, R1      | $R1 \leftarrow R1 \vee R1$                                    |
| or ADD R1, R0, R1      | $R1 \leftarrow R1 + 0$  |
| or SLL R1, # 0, R1     | shift left 0 times  |

**8-39**

- |                         |                                |
|-------------------------|--------------------------------|
| (a) JMP Z, # 3200, (RO) | $PC \leftarrow 0 + 3200$       |
| (b) JMPR Z, - 200       | $PC \leftarrow 3400 + (- 200)$ |

## CHAPTER 9

### 9.1



### 9-2

Segment	1	2	3	4	5	6	7	8	9	10	11	12	13
1	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>					
2		T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>				
3			T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>			
4				T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>		
5					T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>	
6						T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	T <sub>5</sub>	T <sub>6</sub>	T <sub>7</sub>	T <sub>8</sub>

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

### 9.3

k = 6 segments

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

### 9.4

t<sub>n</sub> = 50 ns

k = 6

t<sub>p</sub> = 10 ns

n = 100

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

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

### 9.5

- (a)  $t_p = 45 + 5 = 50 \text{ ns}$   $k = 3$   
 (b)  $t_n = 40 + 45 + 15 = 100 \text{ ns}$

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

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

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

### 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  $n = 10$   $t_{AV} = 48 \text{ ns}$   
 For  $n = 100$   $t_{AV} = 31.8 \text{ ns}$   
 For  $n \rightarrow \infty$   $t_{AV} = 30 \text{ ns}$

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 =  $95 + 5 = 100 \text{ ns}$  (time for segment 3)  
 For  $n = 100$ ,  $k = 4$ ,  $t_p = 100 \text{ ns}$ .  
 Time to add 100 numbers =  $(k+n-1)t_p = (4+99)100$   
 $= 10,300 \text{ ns} = 10.3 \mu\text{s}$   
 (b) Divide segment 3 into two segments of  $50 + 5 = 55$   
 and  $45 + 5 = 50 \text{ ns}$ . This makes  $t_p = 55 \text{ ns}$ ;  $k = 5$   
 $(k+n-1)t_p = (5+99)55 = 5,720 \text{ ns} = 5.72 \mu\text{s}$

**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: (1) prefetch target instruction; (b) use a branch target buffer; (c) use a 100p buffer; (d) use branch prediction. (Delayed branch is a software procedure.)

9.11

	1	2	3	4 <sup>th</sup> step
1. Load $R1 \leftarrow M[312]$	FI	DA	FO	EX
2. Add $R2 \leftarrow R2 + M[313]$	FI	FI	DA	FO
3. Increment R3			FI	DA
4. Store $M[314] \leftarrow R3$				FI

Segment EX: transfer memory word to R1.


Segment FO: Read  $M[313]$ .

Segment DA: Decode (increment) instruction.

Segment FI: Fetch (the store) instruction from memory.

9.12

Load: $R1 \leftarrow \text{Memory}$	1	2	3	4
Increment: $R1 \leftarrow R1 + 1$	I	A	E	
		I	A	E

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

9.13

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

9.14

	1	2	3	4	5	6	7
101 Add R2 to R3	I	A	E				
102 Branch to 104	I	A	E				
103 Increment R1		-	-				
104 Store R1					I	A	E

9.15 Use example of Problem 9-14.

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

- 9.16 (a) There are 40 product terms in each inner product,  $40^2 = 1,600$  inner products must be evaluated, one for each element of the product matrix.  
(b)  $40^3 = 64,000$

- 9.17  $8 + 60 + 4 = 72$  clock cycles for each inner product. There are  $60^2 = 3600$  Inner products. Product matrix takes  $3600 \times 72 = 259,200$  clock cycles to evaluate.

9.18

memory array 1 use addresses: 0, 4, 8, 12, ..., 1020.

Array 2: 1, 5, 9, 13, ..., 1021; Array 3: 2, 6, 10, ..., 1022.

Array 4: 3, 7, 11, ..., 1023.

**9.19**

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

**9.20**

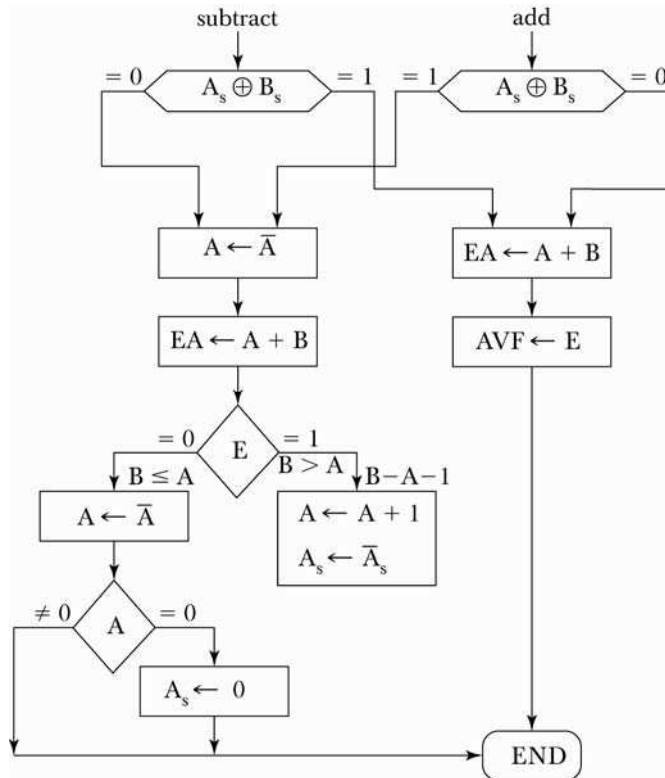
Divide the 400 operations into each of the four

Processors, Processing time is:  $\frac{400}{4} \times 40 = 4,000 \text{ nsec}$ .

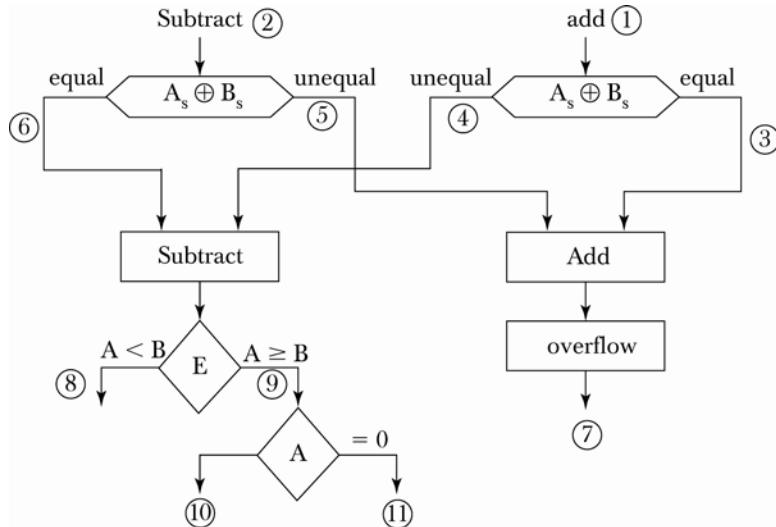
Using a single pipeline, processing time is 400 to 4000 nsec.

## CHAPTER 10

### 10.1



### 10.2



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

(a)  $(+45) + (+31) = 76$

(b)  $(-31) + (-45) = -76$

(c)  $(+45) - (+31) = 14$

(d)  $(+45) - (+45) = 0$

(e)  $(-31) - (+45) = -76$

① ③

① ③

② ⑥

② ⑥

② ⑤

⑦ ← path

⑦

⑨ ⑩

⑨ ⑪

⑦

AVF = 1

AVF = 1

AVF = 0

AVF = 0

AVF = 1

### 10.3

(a) 
$$\begin{array}{r} +35 \\ +40 \\ \hline +75 \end{array}$$
 
$$\begin{array}{r} 0\ 100011 \\ 0\ 101000 \\ \hline 1\ 001011 \end{array}$$

$F = 0$   $E = 1 \leftarrow$  carries  $\rightarrow F = 1$   $E = 0$   
 $F \oplus E = 1$ ; overflow  $F \oplus E = 1$ ; overflow

(b) 
$$\begin{array}{r} -35 \\ -40 \\ \hline -75 \end{array}$$
 
$$\begin{array}{r} 1\ 011101 \\ 1\ 011000 \\ \hline 0\ 110101 \end{array}$$

### 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) = column (c)

Case 2.

If  $X \geq Y$  then  $(X - Y) \geq 0$  and consists of  $k$  bits.

operation in column (b) given:  $2^{2k} + (X - Y)$ . 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 column (c).

Case 3.

is the same as case 2 with  $X$  and  $Y$  reversed

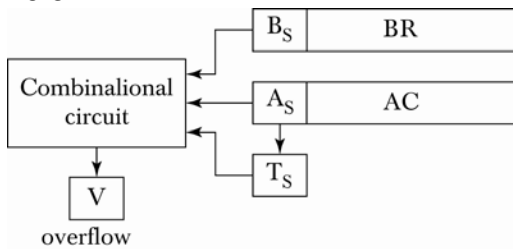
Case 4.

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 Avgend sign into Ts.

Then add:  $AC \leftarrow AC + BR$

As 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	
0	1	0	0	change of sign quantities subtracted
0	1	1	0	
1	0	0	0	
1	0	1	0	
1	1	0	1	change of sign
1	1	1	0	

Boolean functions for circuit:

$$V = T'_s B'_s A_s + T_s b_s A'_s$$



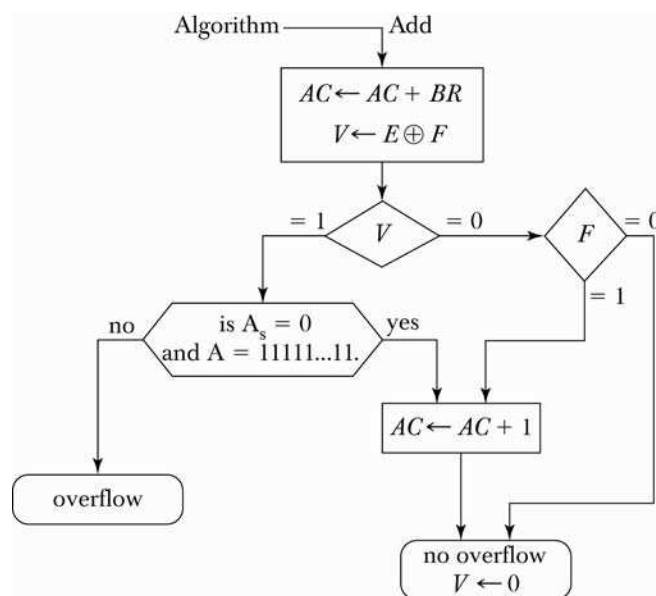
### 10.6 (a)

$$\begin{array}{r} -9 \quad 1\ 0110 \\ -6 \quad 1\ 1001 \\ \hline -15 \quad 0\ 1111 \end{array}$$
 Add end around carry F as needed in signed - 1's complement addition:  

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

$F = 1 \quad E = 0 \leftarrow$  Carries  
 $E \oplus F = 1$  but there should be no overflow since result is -15

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



### 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 maximum 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 = 1 1 1 1 = (31) <sub>10</sub>	31 × 21 = 651
	<u>E</u>	<u>A</u> <u>Q</u> <u>SC</u>
Multiplier in Q - -	0	0000 10101 101   Q = (21) <sub>10</sub>
Q <sub>n</sub> = 1, add B - - -		<u>11111</u>
	0	11111
shr EAQ - - - -		01111 11010 100
Q <sub>n</sub> = 0, shr EAQ - -		00111 11101 011
Q <sub>n</sub> = 1, add B - -		<u>11111</u>
	1	00110
shr EAQ - - - -	0	10011 01110 010
Q <sub>n</sub> = 0, shr EAQ - -		01001 10111 001
Q <sub>n</sub> = 1, add B - -		<u>11111</u>
	1	01000
shr EAQ - - - -		<u>1010001011</u> 000
		(651) <sub>10</sub>

## 10.10 (a)

$$\frac{10100011}{1011} = 1110 + \frac{1001}{1011} \qquad \frac{163}{11} = 14 + \frac{9}{11}$$

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

	<u>E</u>	<u>A</u>	<u>Q</u>	<u>SC</u>
Dividend in AQ - - - -	0	1010	0011	100
shl EAQ - - - - -	1	0100	0110	
add $\bar{B} + 1$ , suppress carry - -		<u>0101</u>		
E = 1, set Q <sub>n</sub> to 1 - - - -	1	1001	0111	011
shl EAQ - - - - -	1	0010	1110	
add $\bar{B} + 1$ , suppress carry -		<u>0101</u>		
E = 1, set Q <sub>n</sub> to 1 - - - -	1	0111	1111	010
shl EAQ - - - - -	0	1111	1110	
add $\bar{B} + 1$ , carry to E - -		<u>0101</u>		
E = 1, set Q <sub>n</sub> to 1 - - - -	1	0100	1111	001
shl EAQ - - - - -	0	1001	1110	
add $\bar{B} + 1$ , carry to E - - -		<u>0101</u>		
E = 0, leave Q <sub>n</sub> = 0 - - - -	0	1110	1110	
add B - - - - -		<u>1011</u>		
restore remainder - -	1	<u>1001</u>	<u>1110</u>	000
		remainder	quotient	

### 10.10 (b)

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

### 10.11

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

$$(+15) \times (+13) = +195 = (0\ 011000011)_2$$

$$BR = 01111 (+15); \overline{BR} + 1 = 10001 (-15); QR = 01101 (+13)$$

$Q_n Q_{n+1}$		<u>AC</u>	<u>QR</u>	<u><math>Q_{n+1}</math></u>	<u>SC</u>
	Initial	00000	01101	0	101
1 0	Subtract BR	<u>10001</u>			
		10001			
	ashr	11000	10110	1	100
0 1	Add BR	<u>01111</u>			
		00111			
	ashr	00011	11011	0	011
1 0	Subtract BR	<u>10001</u>			
		10100			
	ashr	11010	01101	1	010
1 1	ashr	11101	00110	1	001
0 1	Add BR	<u>01111</u>			
		01100			
	ashr	<u>00110</u>	<u>00011</u>	0	000
		+195			

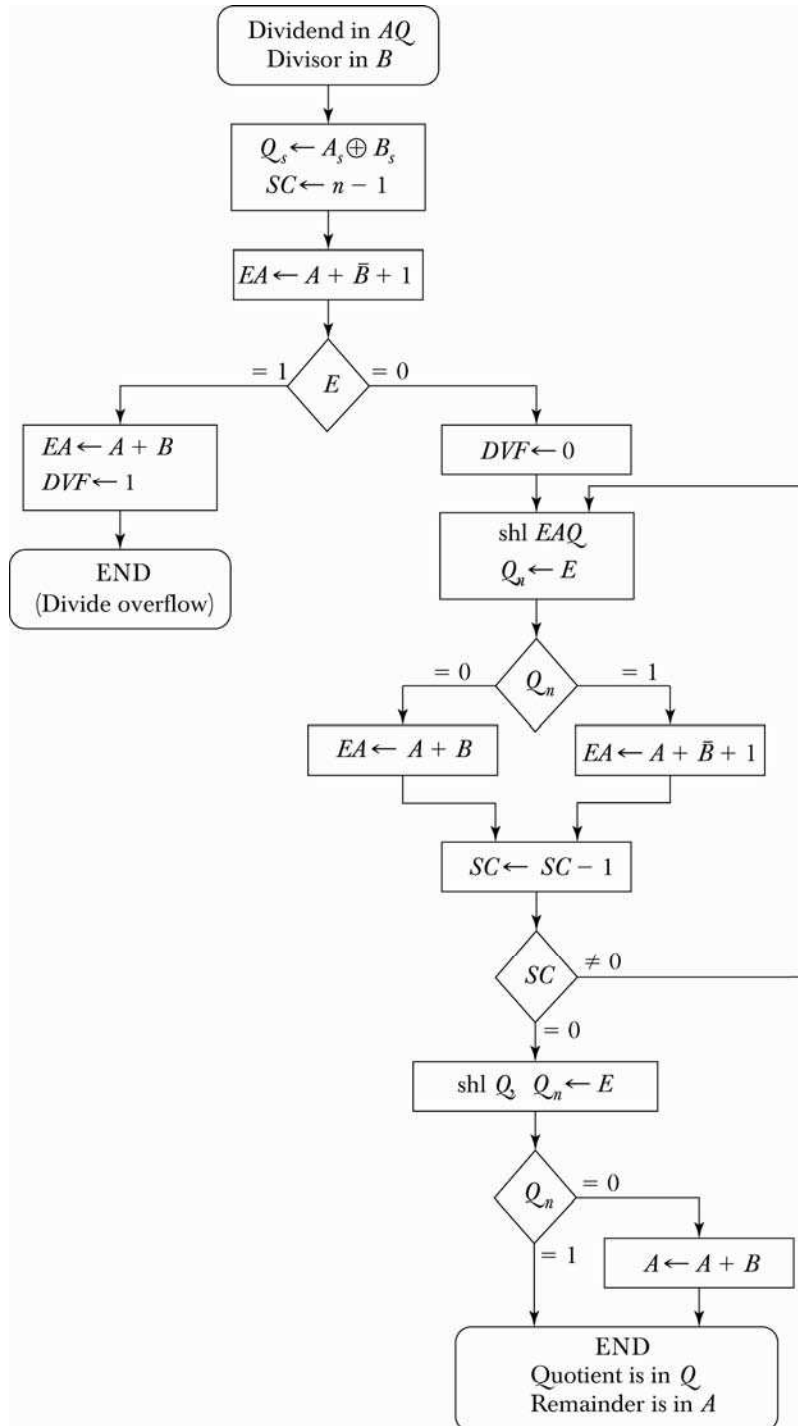
### (b)

$$(+15) \times (-13) = -195 = (1100\ 111101)_{2's\ comp.}$$

$$BR = 0\ 11111 (+15); \overline{BR} + 1 = 10001 (-15); QR = 10011 (-13)$$

$Q_n Q_{n+1}$		<u>AC</u>	<u>QR</u>	<u><math>Q_{n+1}</math></u>	<u>SC</u>
	Initial	00000	10011	0	101
1 0	Subtract BR	<u>10001</u>			
		10001			
	ashr	11000	11001	1	100
1 1	ashr	11100	01100	1	011
0 1	add BR	<u>01111</u>			
		01011			
	ashr	00101	10110	0	010
0 0	ashr	00010	11011	0	001
1 0	Subtract BR	<u>10001</u>			
		10011			
	ashr	<u>11001</u>	<u>11101</u>	1	000
		-195			

## 10.15



## 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 division. Let  $A$  be the radicand,  $Q$  the square-root, and  $R$  the remainder such that  $Q^2 + R = A$  or:  
 $\sqrt{A} = Q$  and a remainder

General coments:

1. For k bits in A (k even), Q will have  $\frac{k}{2}$  bits:  
 $Q = 9_1 9_2 9_3 \dots 9_{k/2}$
2. The first test value is 01  
 The second test value is  $09_1 01$   
 The third test value is  $009_1 9_2 01$   
 The fourth test value is  $0009_1 9_2 9_3 01$  etc.
3. Mark the bits of A in groups of two starting from left.
4. The procedure is similar to the division restoring method as shown in the following example:

$9_1$	$9_2$	$9_3$	$9_4$	
1	1	0	1	$= Q = 13$
$\sqrt{10}$	10	10	01	$= A = 169$
01				subtract first test value 01
01				Answer positive; let $9_1=1$
01	10			bring down next pair
01	01			subtract second test value $09_1 01$
00	01			answer positive; let $9_2 = 1$
00	01	10		bring down next pair
01	11	01		subtract third test value $009_1 9_2 01$
	negative			answer negative; let $9_3 = 0$
00	01	10		restore partial remainder
00	01	10	01	bring down next pair
00	01	10	01	subtract fourth test value $0009_1 9_2 9_3 01$
	Remainder = 00000			answer positive (zero); let $9_4=1$

**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

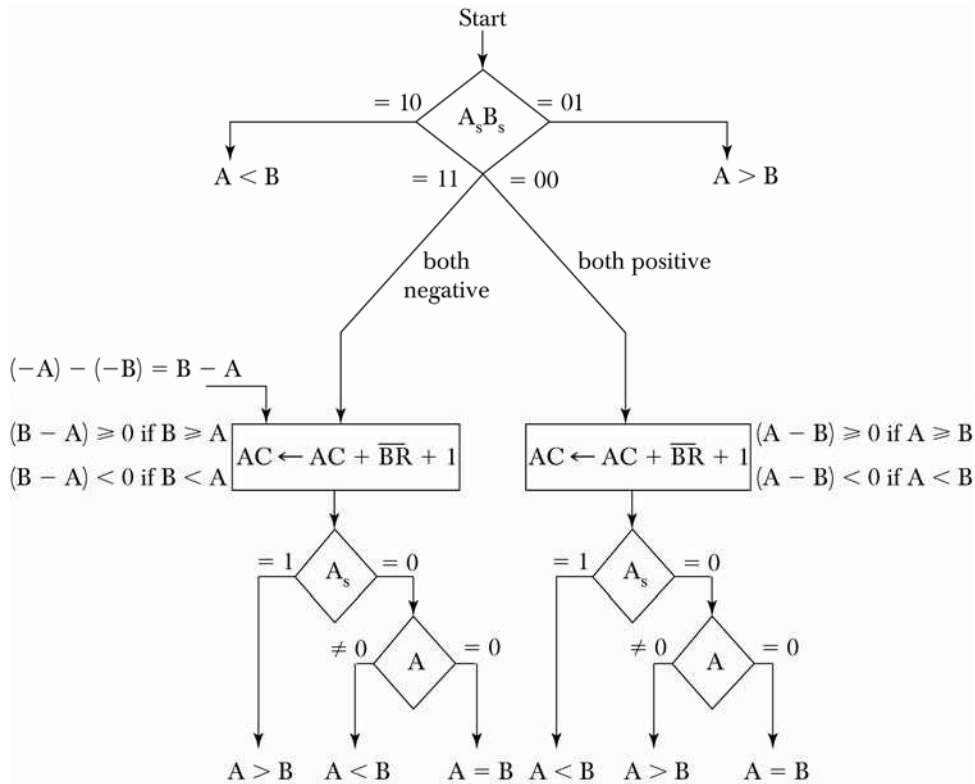
- (b) The biased exponent follows the same algorithm as a magnitude comparator – See Sec. 9–2.
- (c)  $(e_1 + 64) + (e_2 + 64) = (e_1 + e_2 + 64) + 64$   
 subtract 64 to obtain biased exponent sum
- (d)  $(e_1 + 64) - (e_2 - 64) = e_1 + e_2$   
 add 64 to obtain biased exponent difference.

### 10.18

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

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

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



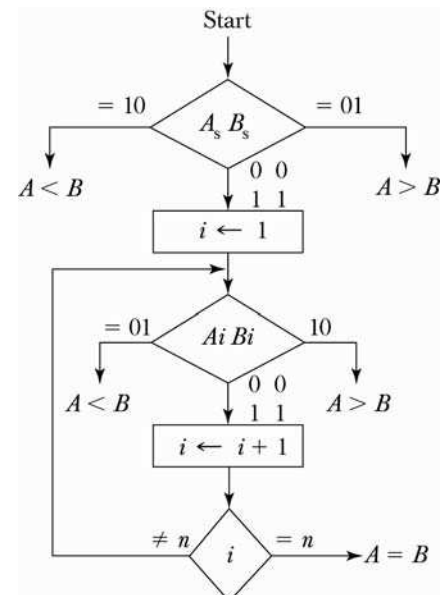
### 10.18 (b)

	$A_s$	$A_1$	$A_2$	$\dots$	$A_n$	
+ 2	0	0	0	0	1	0
+ 1	0	0	0	0	0	1
0	0	0	0	0	0	0
1	1	1	1	1	1	1
- 2	1	1	1	1	1	0
- 3	1	1	1	1	0	1

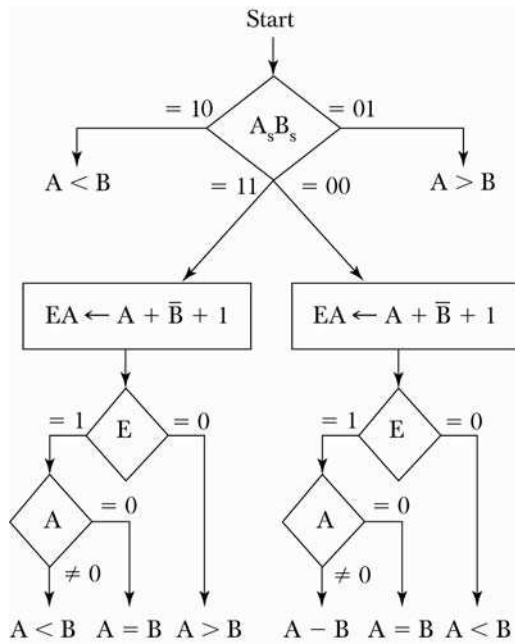
### 10.19

$$A_s \overbrace{A_1 A_2 A_3 \dots A_n}^A$$

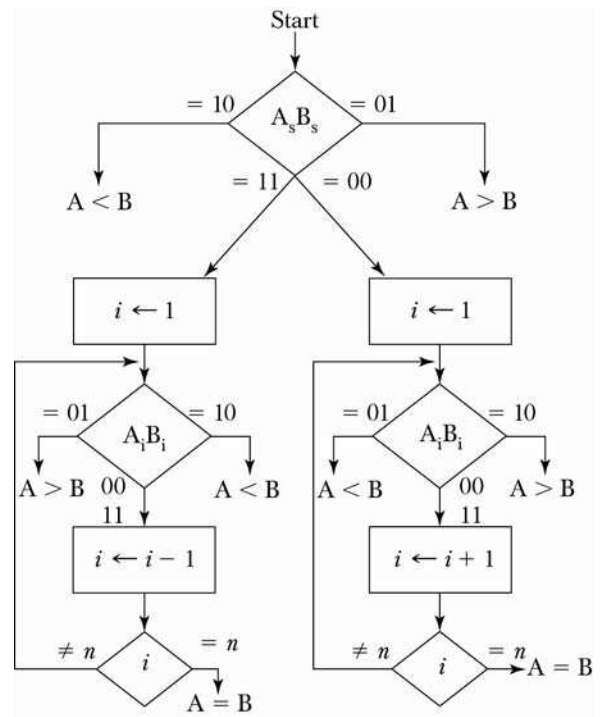
$$B_s \overbrace{B_1 B_2 B_3 \dots B_n}^B$$



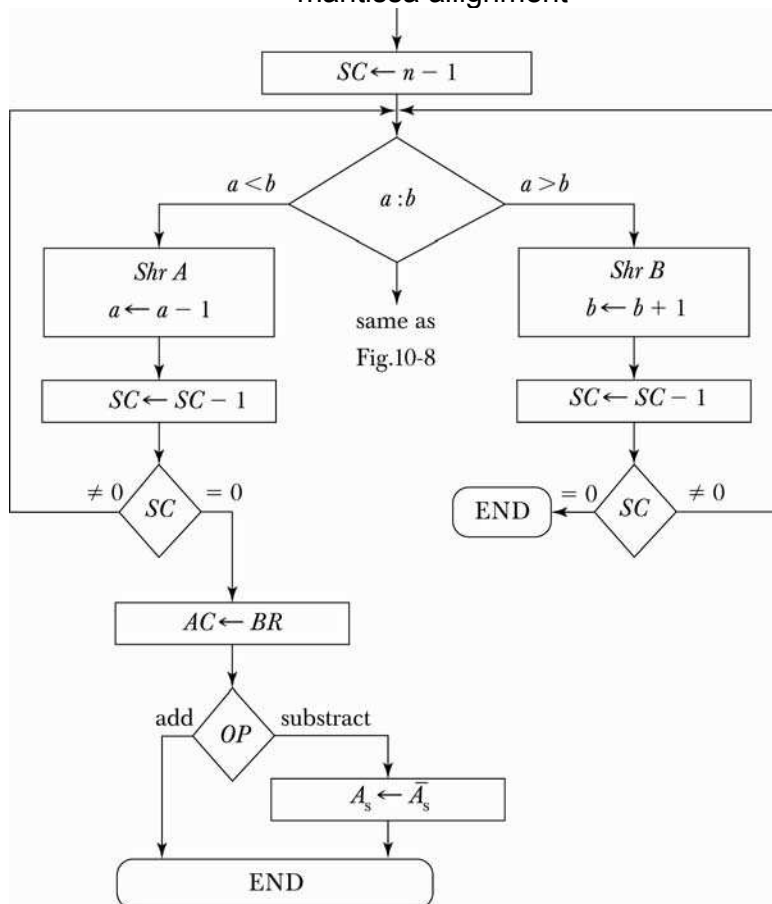
(a)



(b)

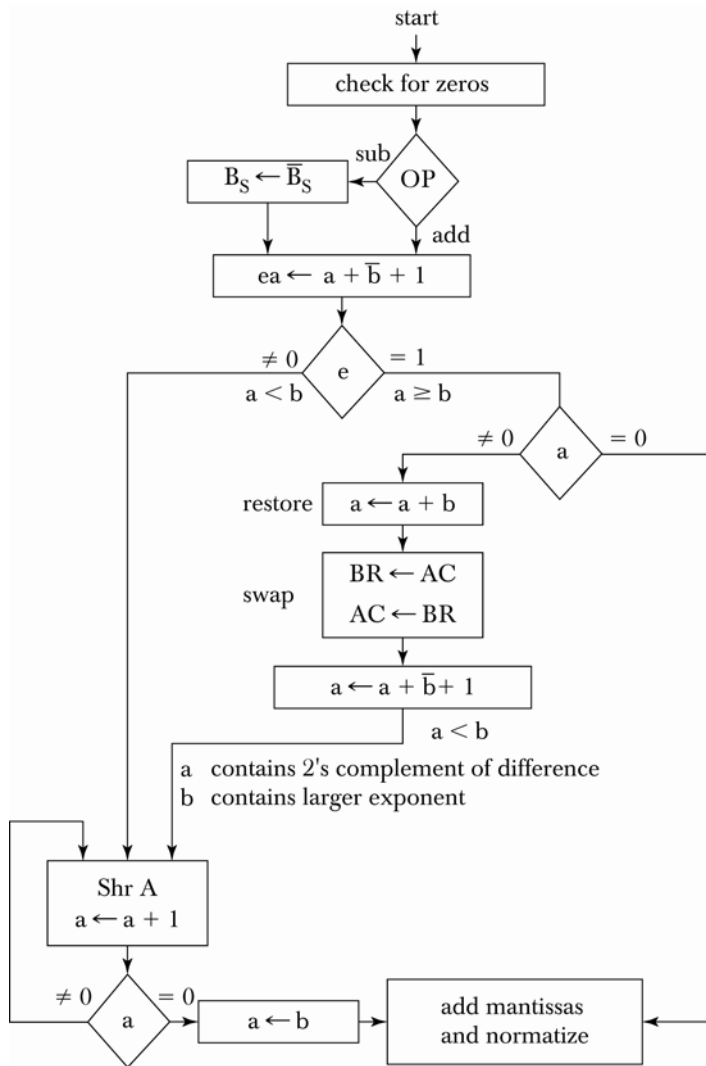


10.20

Fig 10.8  
mantissa alignment



**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.

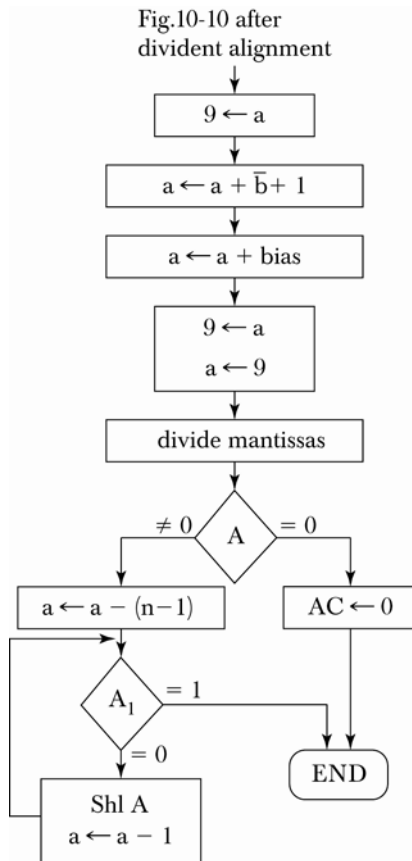
**10.23** 
$$\frac{\text{dividend}}{\text{divisor}} = \frac{A = 0.1 \text{ xxxx}}{B = 0.1 \text{ xxxx}} \text{ where } x = 0, 1$$

- (a) If  $A < B$  then after shift we have  $A = 1. \text{ xxxx}$  and 1<sup>st</sup> quotient bit is A 1.  
 (b) If  $A \geq B$ , dividend alignment results in  $A = 0.01 \text{ xxxx}$  then after the left shift  $A \geq B$  and first quotient bit = 1.

**10.24**

$$\frac{\text{dividend}}{\text{divisor}} = \frac{\overbrace{.1 \text{ x x x x}}^{n-1 \text{ bits}} * 2^{e_1}}{\overbrace{.1 \text{ y y y y}} * 2^{e_2}} = .1 \text{ z z z z} * 2^{e_1 - e_2} + \frac{\overbrace{.00000 \text{ r r r r r}}^{\text{remainder}} * 2^{e_1}}{.1 \text{ y y y y} * 2^{e_2}}$$

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



### 10.25

- When the exponents are added or incremented.
- When the exponents are subtracted or decremented.
- Check end-carry after addition and carry after increment or decrement.

### 10.26

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

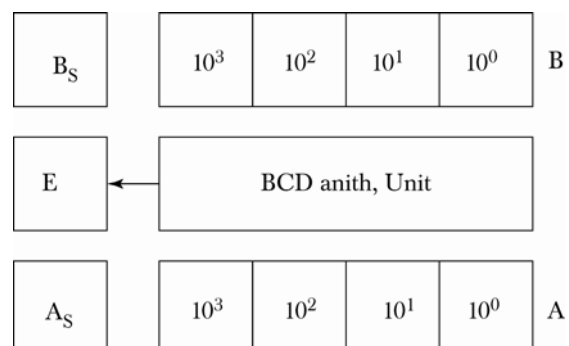
- Product:
 

A	Q
xxxxx	xxxxx. * $2^z$

  
 ↖ binary-point for integer  
 Product in AC: xxxxx. \*  $2^{z+5}$
- Single precision normalized dividend: xxxxx. \*  $2^z$   
 Dividend in AQ:
 

A	Q
xxxxx	00000. * $2^{z-5}$

- 10.27** Neglect  $B_e$  and  $A_e$  from Fig. 10-14. Apply carry directly to E.

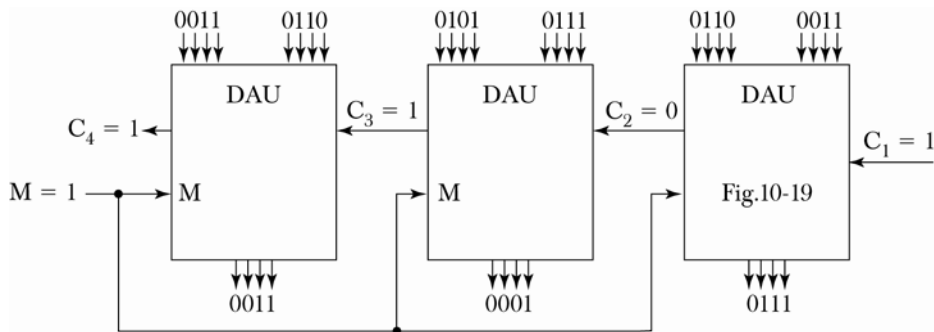


## 10.28

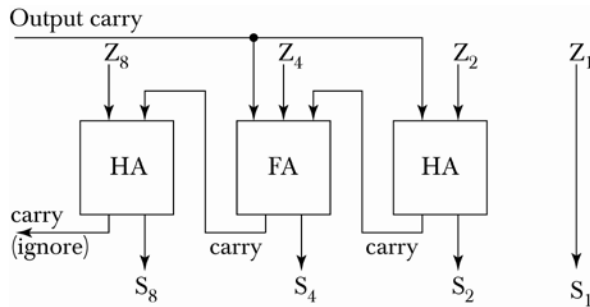
$$\begin{array}{r} 673 \\ - 356 \\ \hline 317 \end{array}$$

$$10\text{'s comp. of } 356 = \begin{array}{r} 673 \\ + \\ \hline 1029 \\ - 356 \\ \hline 673 \end{array}$$

carry  $\rightarrow 1$



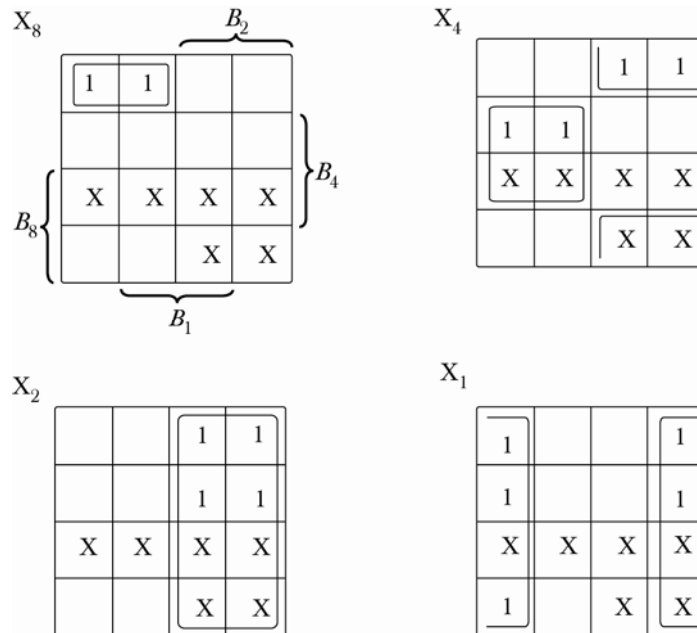
## 10.29



## 10-30

	inputs				outputs				
	B <sub>8</sub>	B <sub>4</sub>	B <sub>2</sub>	B <sub>1</sub>	X <sub>8</sub>	X <sub>4</sub>	X <sub>2</sub>	X <sub>1</sub>	
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<sub>8</sub> B<sub>4</sub> B<sub>2</sub> B<sub>1</sub>) =  $\Sigma$  (10, 11, 12, 13, 14, 15)  
are don't-care conditions



$$\begin{aligned}
 X_8 &= B'_8 B'_4 B'_2 \\
 X_4 &= B_4 B'_2 + B'_4 B_2 \\
 X_2 &= B_2 \\
 X_1 &= B'_1
 \end{aligned}$$

### 10.31

Dec	Z uncorrected	Y corrected
0	0 1 1 0	0 0 1 1
1	0 1 1 1	0 1 0 0
2	1 0 0 0	0 1 0 1
3	1 0 0 1	0 1 1 0
4	1 0 1 0	0 1 1 1
5	1 0 1 1	1 0 0 0
6	1 1 0 0	1 0 0 1
7	1 1 0 1	1 0 1 0
8	1 1 1 0	1 0 1 1
9	1 1 1 1	1 1 0 0

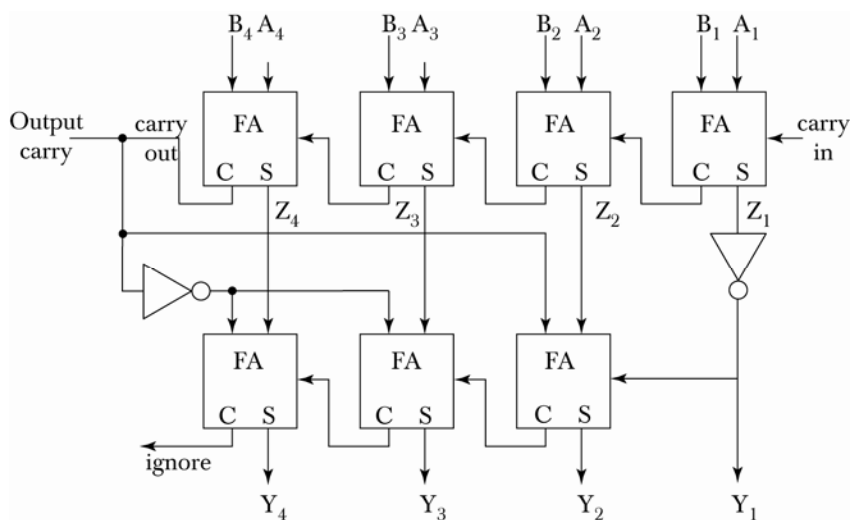
No output carry

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

ignore carry

	Z	Y
dec	uncorrected	corrected
10	1 0 0 0 0	1 0 0 1 1
11	1 0 0 0 1	1 0 1 0 0
12	1 0 0 1 0	1 0 1 0 1
13	1 0 0 1 1	1 0 1 1 0
14	1 0 1 0 0	1 0 1 1 1
15	1 0 1 0 1	1 1 0 0 0
16	1 0 1 1 0	1 1 0 0 1
17	1 0 1 1 1	1 1 0 1 0
18	1 1 0 0 0	1 1 0 1 1
19	1 1 0 0 1	1 1 1 0 0

↑                      ↑  
 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  $X = B$

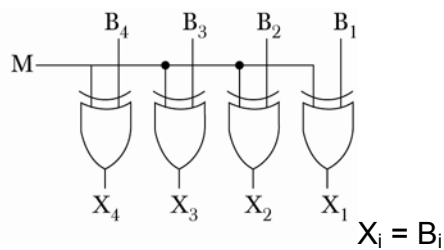
$M = 1$  for  $X = 9$ 's comp. of  $B$

$M \ B_i \mid x_i = B_i \oplus M$

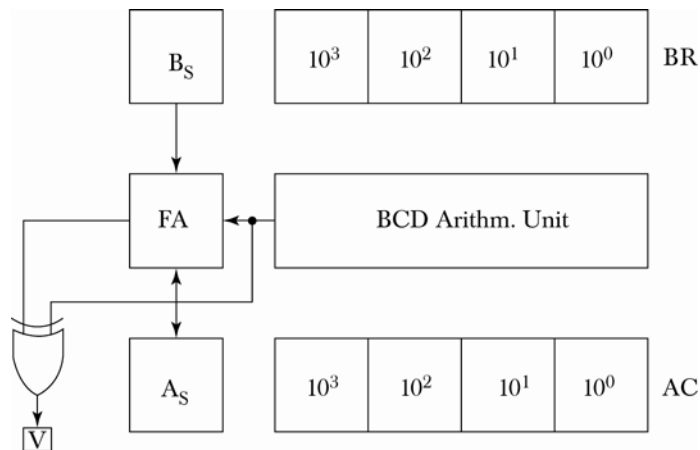
0	0	0
0	1	1
1	0	1
1	1	0

$x_i = B_i$

$x_i = B'_i$



### 10.33



Algorithm is similar to flow chart of Fig. 10.2

### 10.34

(a) B = 470

	Ae	A	Q	sc
Initial	0	000	152	3
$Q_L \neq 0$ -----	0	470	151	
$Q_L \neq 0$ -----	0	940	150	
$Q_L = 0$ , d shr -----	0	094	015	2
	0	564	014	
$Q_L \neq 0$ -----	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)

999	
<u>× 199</u>	
8991	– first partial product
+ 89910	
<u>98901</u>	– second partial product
+ 99900	
<u>198801</u>	– final product
	Ae = 8
	Ae = 9
	Ae = 1

### 10.35

$$\frac{1680}{32} = 52 + \frac{16}{32}$$

$$B = 032,$$

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

	E	AC	A	Q	SC	
initial						(Continued here)
dshl	0	0 16	80	2		
add $\bar{B} + 1$		1 68	00			
E = 1		9 68				
add $\bar{B} + 1$	1	1 36	01			
		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		

	E	AC	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	---	9 84				
add B	--	0 32				
	1	0 16	52	0		
						remainder quotient

### 10.36

- At the termination of multiplication we shift right the content of A to get zero in Ae.
- At the termination of division, B is added to the negative difference. The negative difference is in 10's complement so Ae = 9. Adding Be = 0 to Ae = 9 produces a carry and makes Ae = 0.

### 10.37

Change the symbols as defined in Table 10.1 and use same algorithms as in sec. 10.4 but with multiplication and division of mantissas as in sec. 10.5.

## CHAPTER 11

### 11.1

	<u>A<sub>1</sub>—A<sub>2</sub></u>	<u>A<sub>i</sub> A<sub>0</sub></u>	
12 =	000011	0 0	CS = A <sub>2</sub> A <sub>3</sub> A' <sub>4</sub> A' <sub>5</sub> A' <sub>6</sub> A' <sub>7</sub>
13 =	000011	0 1	RS1 = A <sub>1</sub>
14 =	000011	1 0	RS0 = A <sub>0</sub>
15 =	<u>000011</u>	1 1	
	To CS	RSI RSO	

### 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 plotter; Graphic display; Voice output; Digital to analog converter; Instrument indicator.

### 11.5

See text discussion in See, 11.2.

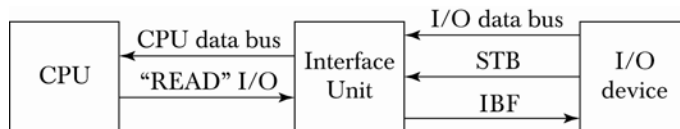
### 11.6

- Status command – Checks status of flag bit.
- Control command – Moves magnetic head in disk.
- Status command – checks if device power is on.
- Control command – Moves paper position.
- Data input command – Reads value of a register.

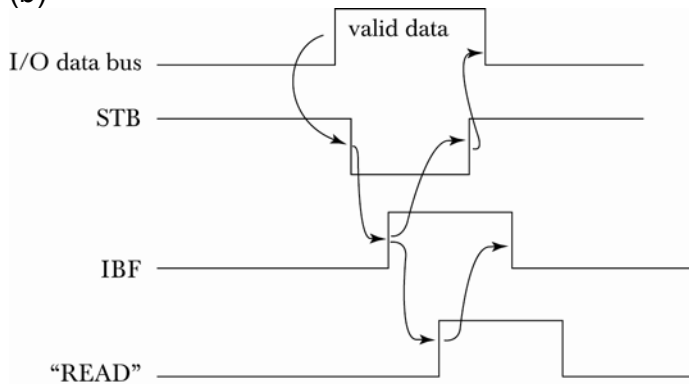


### 11.7

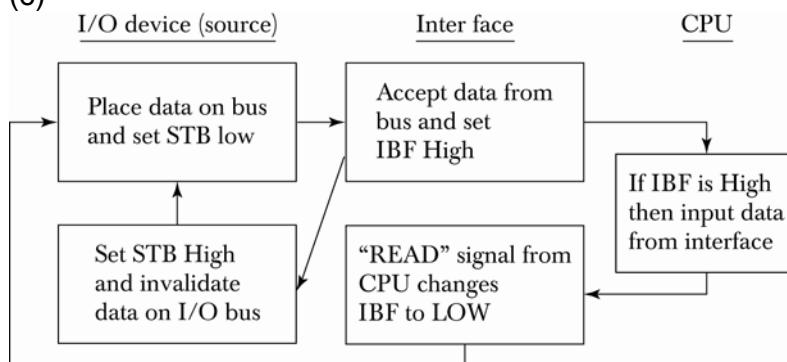
(a)



(b)

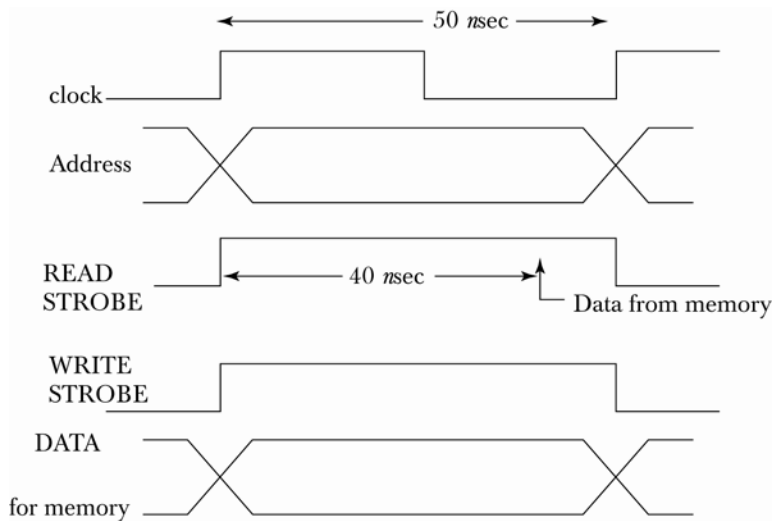


(c)



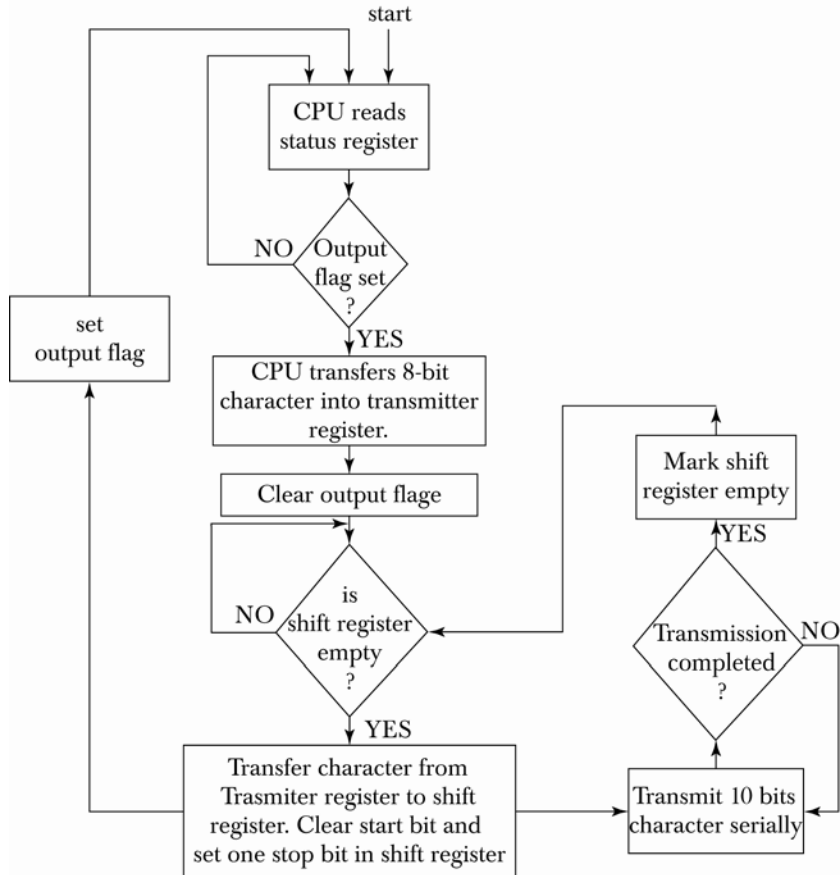
### 11.8

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



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

(b)  $\frac{k}{n - m} \text{ sec.}$

(c) No need for FIFO

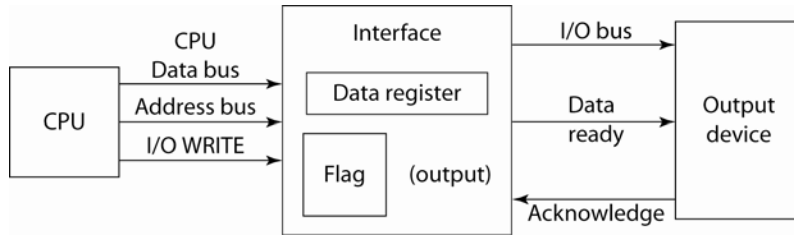
**11.14**

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

**11.15**

		<u>Input ready</u>	<u>output ready</u>	<u>F<sub>1</sub> – F<sub>4</sub></u>
(a)	Empty buffer	1	0	0000
(b)	Full buffer	0	1	1111
(c)	Two items	1	1	0011

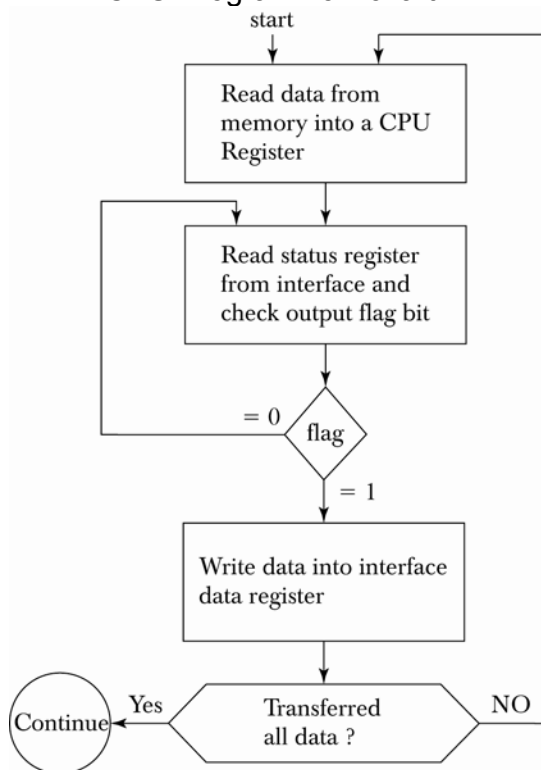
### 11.16



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

### 11.20

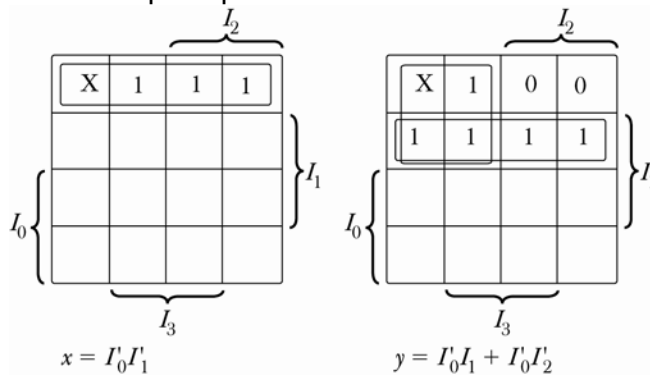
	Device 1	Device 2
(1) Initially, device 2 sends an interrupt request:	PI = 0; PO = 0; RF = 0	PI = 0; PO = 0; RF = 1
(2) Before CPU responds with acknowledge, device 1 sends interrupt request:	PI = 0; PO = 0; RF = 1	PI = 0; PO = 0; RF = 1
(3) After CPU sends an acknowledge, device 1 has priority:	PI =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

### 11.22

Table 11.2

$I_0$	$I_1$	$I_2$	$I_3$	x	y	lst
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



### 11.23

Same as Fig. 11.14. Needs 8 AND gates and an  $8 \times 3$  decoder.

**11.24 (a)**

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

(b)

<u>Binary</u>	<u>hexadecimal</u>
1010 0000	A0
1010 0100	A4
1010 1000	A8
1010 1100	AC
1011 0000	B0
1011 0100	B4
1011 1000	B8
101 11100	BC

**11.25**76 = (01001100)<sub>2</sub>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.

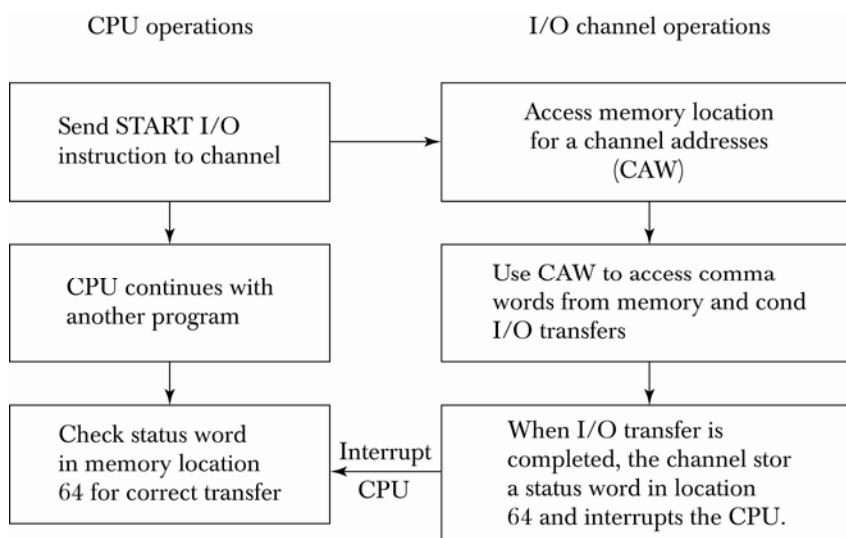
### 11.29

CPU refers to memory on the average once (or more) every  $1 \mu \text{ sec. } (1/10^6)$ . Characters arrive one every  $1/2400 = 416.6 \mu \text{ sec.}$  Two characters of 8 bits each are packed into a 16-bit word every  $2 \times 416.6 = 833.3 \mu \text{ sec.}$  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 occurring 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



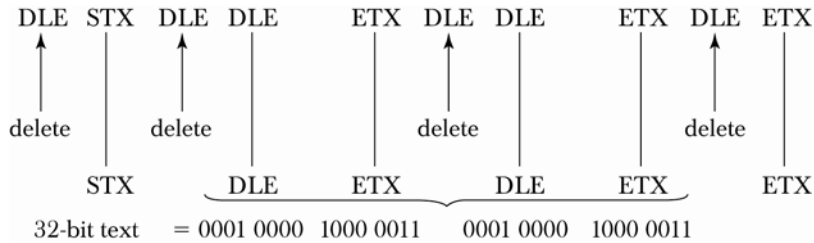
### 11.32

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 either ACK (acknowledge) or NAK (negative acknowledge) or the terminal does not respond during a timeout period. If the processor receives an ACK, it sends a block of text.

### 11.34



### 11.35

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

### 11.36

Information to be sent (1023):

0111111111

After zero insertion, information transmitted:

0111110111110

delete

Information received after O's deletion:

0111111111



## CHAPTER 12

### 12.1

- (a)  $\frac{2048}{128} = 16$  chips
- (b)  $2048 = 2^{11}$  11 lines to address 2078 bytes.  
 $128 = 2^7$  7 lines to address each chip  
 4 lines to decoder for selecting 16 chips
- (c)  $4 \times 16$  decoder

### 12.2

- (a) 8 chips are needed with address lines connected in parallel.
- (b)  $16 \times 8 = 128$  chips. Use 14 address lines ( $16 \text{ k} = 2^{14}$ )  
 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	$\xleftarrow{5 \times 32 \text{ decoder}}$				x	x	x		x	x	x	x
ROM	4000-1FFF	0	0	0	1	$\xleftarrow{3 \times 8 \text{ decoder}}$				x	x	x	x	x	x	x	x
		$\xrightarrow{\text{to } \overline{\text{CS2}}}$															

### 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	$\xleftarrow{3 \times 8 \text{ decoder}}$				x	x	x	x	x	x	x
ROM	4000-4FFF	0	1	0	0		$\xleftarrow{2 \times 4 \text{ decoder}}$				x	x	x	x	x	x	x
Interface	8000-800F	1	0	0	0	0	0	0	0	0000	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-bits number into the register with a single 1 and 7 (0's). Each output of the register selects one of the 8 bank 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.

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

## 12.8

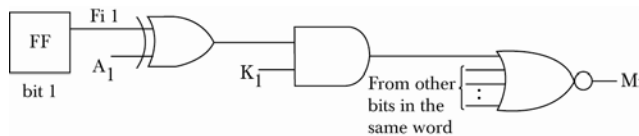
An eight-track tape reads 8 bits (one character) at the same time.

Transfer rate =  $1600 \times 120 = 192,000$  characters/s

## 12.9

From Sec. 12.4:  $M_i = \prod_{g=1}^n [(A_j \oplus F_{ig})' + K'_g]$

$$M'_i = \sum_{g=1}^n (A_j \oplus F_{ig}) K_j$$



## 12.10

A Match occurs if  $T_i = 1$

$$\text{Match} = M_i T_i$$



## 12.11

$$M_i = \left( \prod_{g=1}^n A_j F_{ig} + A'_g F'_{ig} + K'_g \right) \cdot (K_1 + K_2 + K_3 + \dots + K_n)$$

At least one key bit  $k_i$  must be equal to 1

The diagram illustrates a match register system. On the left, a vertical column represents the match registers, labeled  $M_1$ ,  $M_2$ , and  $M_m$ , with vertical ellipsis between  $M_2$  and  $M_m$ . The entire column is labeled "match register". Each  $M_i$  is connected to an AND gate. The output of the AND gate for  $M_1$  is labeled "Read word 1", for  $M_2$  is "Read word 2", and for  $M_m$  is "Read word m". Vertical ellipsis between the AND gates indicates intermediate read words. Below the match registers, a "d x m decoder" block is shown, which receives inputs from the match registers. To the right of the decoder is a "d-bit counter" block, which has two inputs: "clear counter" and "count".

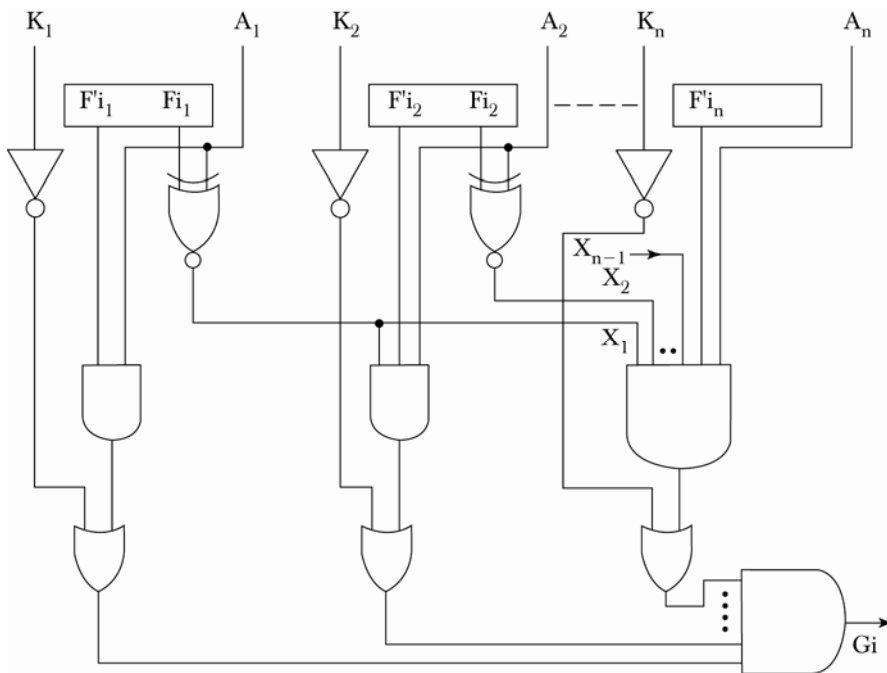
## 12.14

Output indicator  $G_i = 1$  if:

etc.

(First pair of bits are equal and second bit in  $A = 1$  while  $F_{i2} = 0$ )

- 91 -



### 12.15

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

(a)

7 bits	10 bits
TAG	INDEX

← Block → Words ←  
8 bits 2 bits

(b)



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

### 12.16

(a)  $0.9 \times \frac{100}{\text{cache access}} + 0.1 \times \frac{11000}{\text{cache + memory access}} = 90 + 110 = 200 \text{ n sec.}$

(b)  $0.2 \times \frac{1000}{\text{write access}} + 0.8 \times \frac{200}{\text{read access}} = 200 + 160 = 360 \text{ n sec.}$   
 from (a)

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

### 12.17

Sequence: ABCDBEDACECE

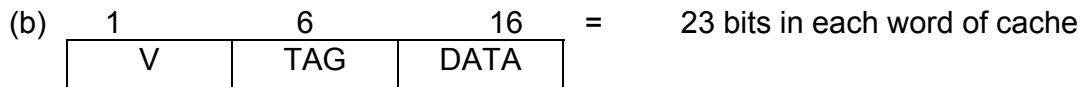
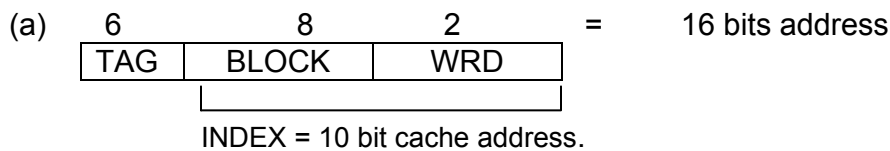
LRU



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

### 12.18

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



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

### 12.19

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

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

(c)  $\frac{16M}{2K} = 8$  K 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 1 4 0 1 0 2 3 5 7

Page reference	(a) First-in		(b) LRU	
	Pages in main memory	Contents of FIFO	Pages in memory	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**

600 AF and F00AF

**12.23**

Logical address:      7 bits      5 bits      12 bits      = 24 bits

Segment	Page	Word
---------	------	------

Physical address:      12 bits      12 bits

Block	Word
-------	------

**12.24**Segment 36 = (0100100)<sub>2</sub> (7-bit binary)Page 15 = (01111)<sub>2</sub> (5-bit binary)Word 2000 = (011111010000)<sub>2</sub> (12-bit binary)

Logical address = 0100100    01111    011111010000  
(24-bit binary)

## CHAPTER 13

### 13.1

Tightly coupled multiprocessors require that all processed 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

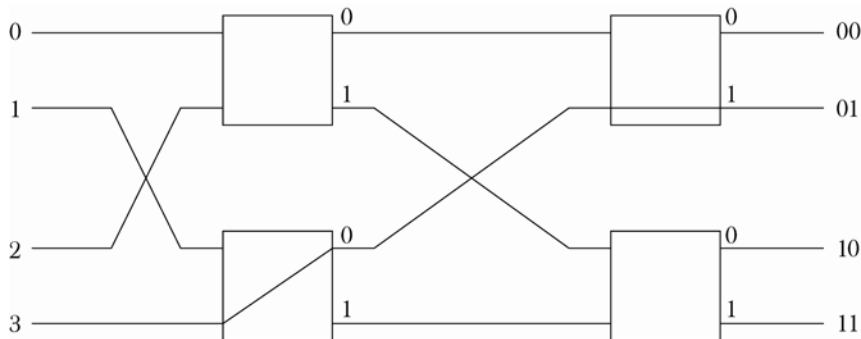
### 13.4

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

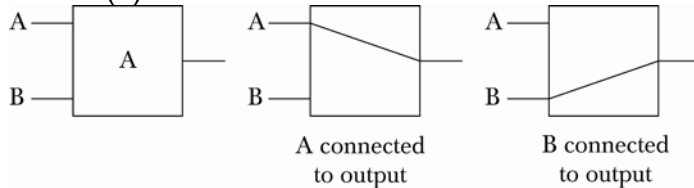
### 13.5

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

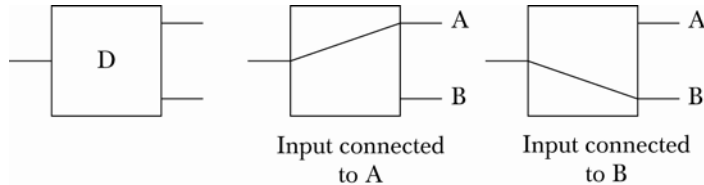
### 13.6



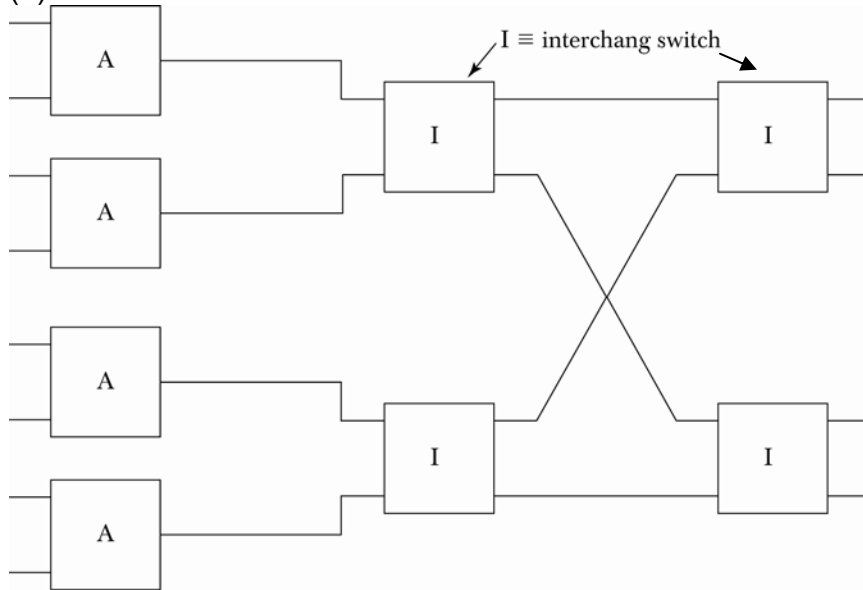
### 13.7 (a) Arbitration switch:



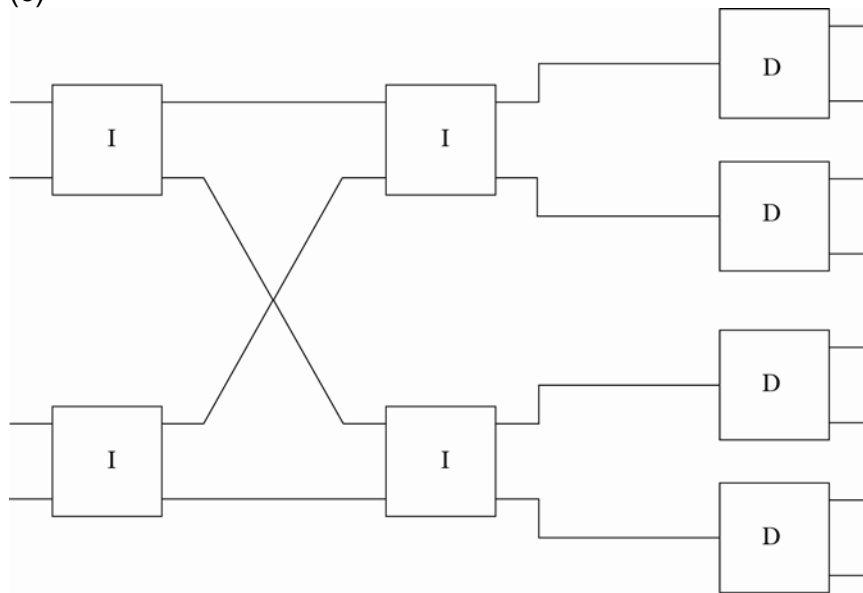
### Distribution switch:



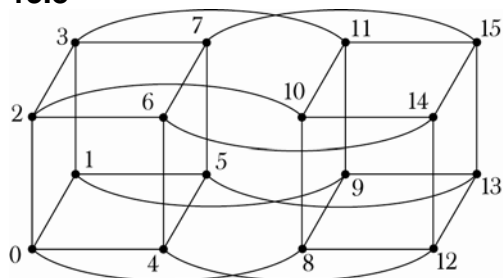
(b)



(c)



### 13.8



$$7 = 0111$$

$$9 = \underline{1001} \oplus$$

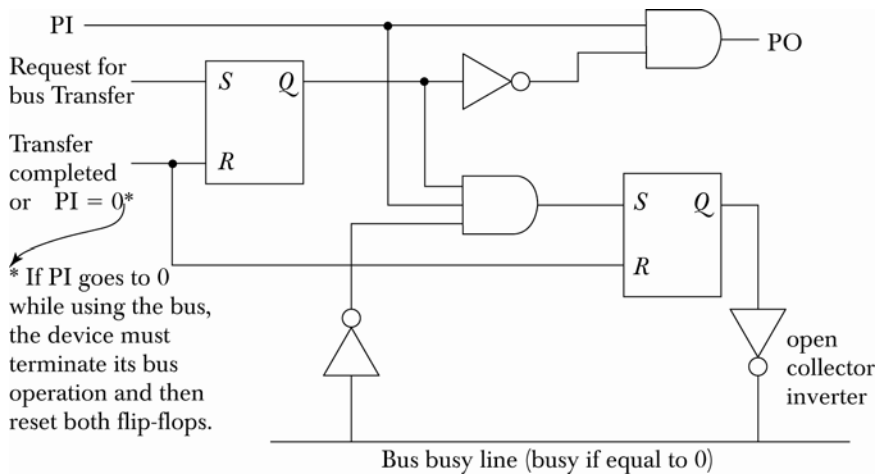
$$1110 = \text{three axes}$$



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



### 13.10

	1 $I_0$	2 $I_1$	3 $I_2$	4 $I_3$	
Encoder input	0	1	1	0	
Encoder output		0	1		( $I_1$ has highest priority)
Decoder input		0	1		
Decoder output	0	1	0	0	Arbiter 2( $J_1$ ) is acknowledged

### 13.11

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

### 13.12

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

**13.13**

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

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

(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 snoop 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.