SOLUTIONS MANUAL

Computer Systems Organization and Architecture

John D. Carpinelli

Table of Contents

Chapter 1

1.

x	y	z	$x + y'$	$y + z$	$(x + y')(y + z)$	xy	xz	$y'z$	$xy + xz + y'z$
0	0	0	1	0	0	0	0	0	0
0	0	1	1	1	1	0	0	1	1
0	1	0	0	1	0	0	0	0	0
0	1	1	0	1	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0
1	0	1	1	1	1	0	1	1	1
1	1	0	1	1	1	1	0	0	1
1	1	1	1	1	1	1	1	0	1

2. a)

w	x	y	z	wx	xz	y'	$wx + xz + y'$
0	0	0	0	0	0	1	1
0	0	0	1	0	0	1	1
0	0	1	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	0	0	0	1	1
0	1	0	1	0	1	1	1
0	1	1	0	0	0	0	0
0	1	1	1	0	1	0	1
1	0	0	0	0	0	1	1
1	0	0	1	0	0	1	1
1	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0
1	1	0	0	1	0	1	1
1	1	0	1	1	1	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	1	0	1

b)

w	x	y	z	$w + x + y + z$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

c)

w	x	y	z	$w'x'yz$	$w'xyz$	$w'x'yz'$	$w'xyz'$	$w'x'yz + w'xyz + w'x'yz' + w'xyz'$
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	0
0	0	1	0	0	0	1	0	1
0	0	1	1	1	0	0	0	1
0	1	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0
0	1	1	0	0	0	0	1	1
0	1	1	1	0	1	0	0	1
1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0
1	1	0	1	0	0	0	0	0
1	1	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0

3.

a	b	ab	$(ab)'$	a'	b'	$a' + b'$
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

a	b	$a + b$	$(a + b)'$	a'	b'	$a'b'$
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

4. a) $w' + x' + y' + z'$

b) $w' + x' + y'z$

c) $(w' + x') + (w' + y') + (w' + z') + (x' + y') + (x' + z') + (y' + z') = w' + x' + y' + z'$

5. a)

$wx \backslash yz$	00	01	11	10
00	1	1	0	0
01	0	1	1	0
11	0	0	1	1
10	1	0	0	1

$w'x'y' + w'xz + wxy + wx'z'$

or

$x'y'z' + w'y'z + xyz + wyz'$

b)

$wx \backslash yz$	00	01	11	10
00	1	1	0	1
01	1	1	1	0
11	1	1	1	0
10	1	0	0	1

$x'z' + w'y' + xz + xy'$

or

$x'z' + w'y' + xz + y'z'$

6. a)

$wx \backslash yz$	00	01	11	10
00	1	0	0	1
01	1	X	1	1
11	0	1	X	0
10	0	X	X	0

$w'z' + xy$

b)

$wx \backslash yz$	00	01	11	10
00	X	1	1	X
01	0	X	X	0
11	0	0	X	0
10	X	X	X	X

x'

7. a)

$wx \backslash yz$	00	01	11	10
00	0	1	0	0
01	0	1	1	0
11	1	1	1	1
10	0	0	0	0

$wx + xz + w'y'z$

b)

$wx \backslash yz$	00	01	11	10
00	1	0	0	0
01	1	1	0	0
11	1	1	0	1
10	1	0	0	1

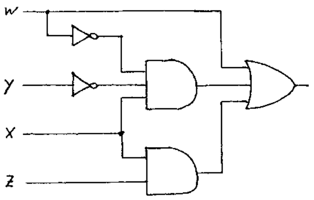
$y'z' + xy' + wz'$

c)

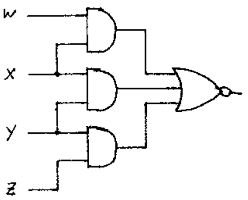
$wx \backslash yz$	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

Already minimal

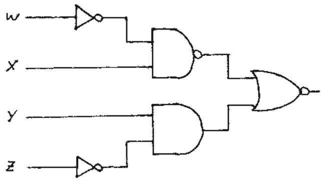
8. a)



b)



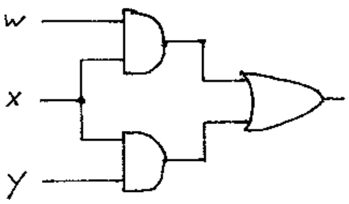
c)



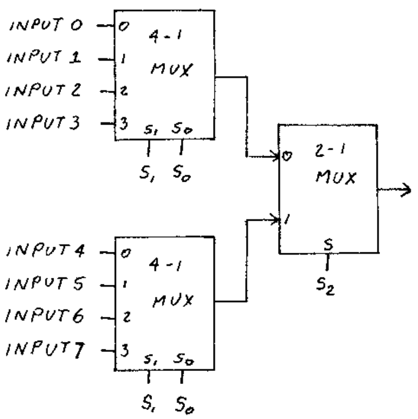
9. $wxy' + wxz + w'xy + xyz'$:

$wx \backslash yz$	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	1	1	1	1
10	0	0	0	0

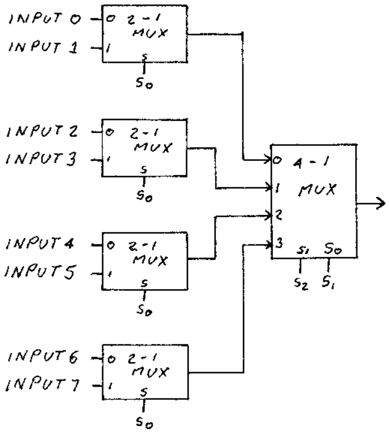
$wx + xy$



10. a)



b)



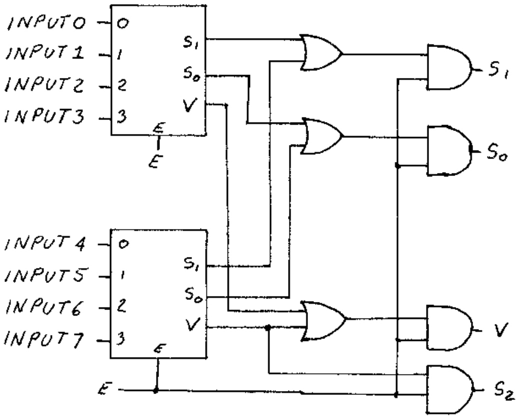
11. Change the AND gates to NAND gates. The rest of the circuit is unchanged.

12. Remove the tri-state buffers and do one of the following:

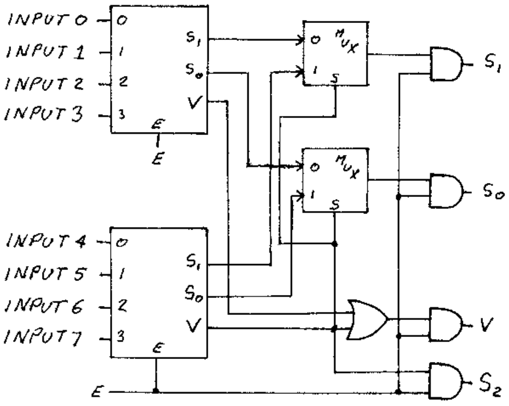
a) Change each 2-input AND gate to a 3-input AND gate. Each gates' inputs should be its two original inputs and *E*, or

b) Have each AND gate's output serve as an input to another 2-input AND gate, one gate for each original AND gate. The second input to the new 2-input AND gates is *E*.

13.



14.



15. Set up Karnaugh maps for each output, then develop minimal logic expressions and design the appropriate logic circuits.

$X > Y$:

$X_1X_0\backslash Y_1Y_0$	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$X = Y$:

$X_1X_0\backslash Y_1Y_0$	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

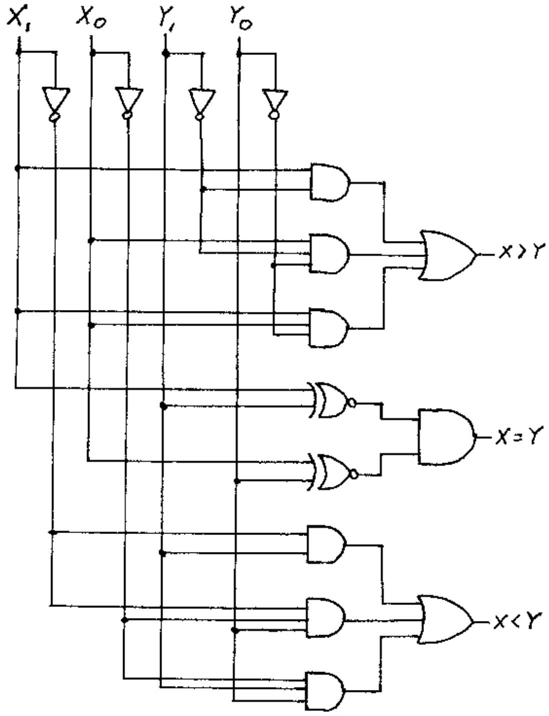
$X < Y$:

$X_1X_0\backslash Y_1Y_0$	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$(X > Y) = X_1Y_1' + X_0Y_1'Y_0' + X_1X_0Y_0'$

$(X = Y) = X_1'X_0'Y_1'Y_0' + X_1'X_0Y_1'Y_0 + X_1X_0'Y_1Y_0' + X_1X_0Y_1Y_0 = (X_1 \oplus Y_1)'(X_0 \oplus Y_0)'$

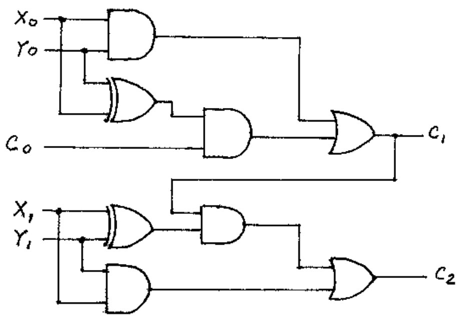
$(X < Y) = X_1'Y_1 + X_1'X_0'Y_0 + X_0'Y_1Y_0$



16. $C_3 = X_2Y_2 + (X_2 \oplus Y_2)(X_1Y_1 + (X_1 \oplus Y_1)(X_0Y_0 + (X_0 \oplus Y_0)C_0))$

$C_4 = X_3Y_3 + (X_3 \oplus Y_3)(X_2Y_2 + (X_2 \oplus Y_2)(X_1Y_1 + (X_1 \oplus Y_1)(X_0Y_0 + (X_0 \oplus Y_0)C_0)))$

17.



18

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	1	0	1	1
01	0	1	0	1
11	X	X	X	X
10	1	0	X	X

$d = X_2'X_0' + X_2'X_1 + X_1X_0' + X_2X_1'X_0$

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	1	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	X	X

$e = X_2'X_0' + X_1X_0'$

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	1	0	0	0
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

$f = X_3 + X_2X_0' + X_2X_1' + X_1'X_0'$

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	0	0	1	1
01	1	1	0	1
11	X	X	X	X
10	1	1	X	X

$g = X_3 + X_2X_0' + X_1X_0' + X_2'X_1$

19.

X_3	X_2	X_1	X_0	a	b	c	d	e	f	g
0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	1	1	1
0	0	1	0	0	0	1	0	0	1	0
0	0	1	1	0	0	0	0	1	1	0
0	1	0	0	1	0	0	1	1	0	0
0	1	0	1	0	1	0	0	1	0	0
0	1	1	0	0	1	0	0	0	0	0
0	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	1	1	0	0

a:

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	0	1	0	0
01	1	0	0	0
11	X	X	X	X
10	0	0	X	X

$a = X_3'X_2'X_1'X_0 + X_2X_1'X_0'$

b:

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	0	0	0	0
01	0	1	0	1
11	X	X	X	X
10	0	0	X	X

$b = X_2X_1'X_0 + X_2X_1X_0'$

c:

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	0	0	0	1
01	0	0	0	0
11	X	X	X	X
10	0	0	X	X

$c = X_2'X_1X_0'$

d:

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	0	1	0	0
01	1	0	1	0
11	X	X	X	X
10	0	1	X	X

$d = X_2X_1'X_0' + X_2'X_1'X_0 + X_2X_1X_0$

e:

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	0	1	1	0
01	1	1	1	0
11	X	X	X	X
10	0	1	X	X

$e = X_2X_1' + X_0$

f:

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	0	1	1	1
01	0	0	1	0
11	X	X	X	X
10	0	0	X	X

$f = X_1X_0 + X_3'X_2'X_0 + X_2'X_1$

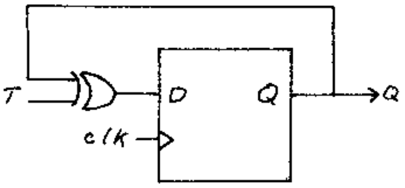
g:

$X_3X_2\backslash X_1X_0$	00	01	11	10
00	1	1	0	0
01	0	0	1	0
11	X	X	X	X
10	0	0	X	X

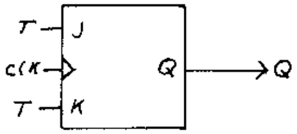
$g = X_3'X_2'X_1' + X_2X_1X_0$

20. The four inputs can be in one of 24 (= 4!) possible orders. Since each sorter has two possible states (MAX = X MIN = Y, or MAX = Y MIN = X), n sorters can have up to 2^n states. Four sorters can have only 2^4 = 16 states, not enough to sort all 24 possible input orders. Five sorters have 2^5 = 32 states, which could be sufficient. (This argument establishes a lower bound; it does not guarantee the existence of a 5-sorter network that can sort four inputs. Since the sorting network of Figure 1.24(b) matches this bound, it is a minimal network.)

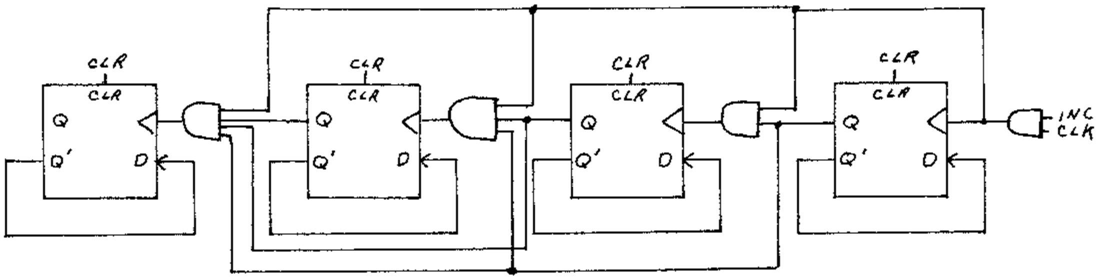
21. a)



b)



22. A flip-flop is clocked if the increment signal and clock are asserted, and all flip-flops to its right are 1.



23. Each clock is driven by Q of the flip-flop to its right instead of Q'. The clock of the rightmost flip-flop is unchanged. All other signals are unchanged.

24.

X_2	X_1	X_0	Q_2	Q_1	Q_0	J_2	K_2	J_1	K_1	J_0	K_0
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	1	0	X	1	X	X	0
0	1	0	1	1	0	1	X	X	0	0	X
0	1	1	0	1	0	0	X	X	0	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	1	0	0	X	0	0	X	X	1
1	1	0	1	1	1	X	0	X	0	1	X
1	1	1	1	0	1	X	0	X	1	X	0

J_2:

$X_2\backslash X_1X_0$	00	01	11	10
0	0	0	0	1
1	X	X	X	X

$J_2 = X_1X_0'$

J_1:

$X_2\backslash X_1X_0$	00	01	11	10
0	0	1	X	X
1	0	0	X	X

$J_1 = X_2'X_0$

J_0:

$X_2\backslash X_1X_0$	00	01	11	10
0	1	X	X	0
1	0	X	X	1

$J_0 = X_2'X_1' + X_2X_1$

K_2:

$X_2\backslash X_1X_0$	00	01	11	10
0	X	X	X	X
1	1	0	0	0

$K_2 = X_1'X_0'$

K_1:

$X_2\backslash X_1X_0$	00	01	11	10
0	X	X	0	0
1	X	X	1	0

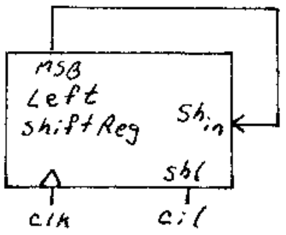
$K_1 = X_2X_0$

K_0:

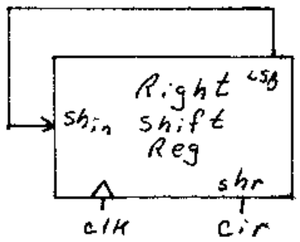
$X_2\backslash X_1X_0$	00	01	11	10
0	X	0	1	X
1	X	1	0	X

$K_0 = X_2'X_1 + X_2X_1'$

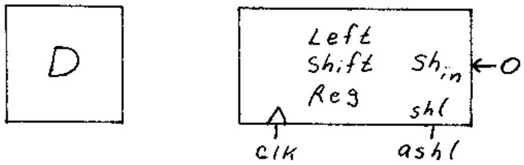
25. a)


MSB
Left
shiftReg
Sh_in
shl
clk
cil

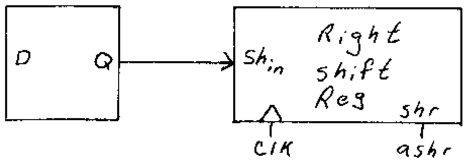
b)


Right
LSB
sh_in
shift
Reg
shr
clk
cir

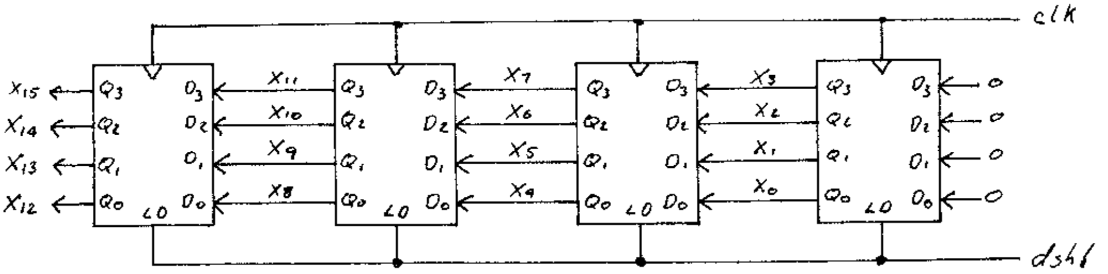
26. a)


D
Left
shift
Reg
Sh_in
0
shl
clk
ashl

b)


D
Q
Sh_in
Right
shift
Reg
shr
clk
ashr

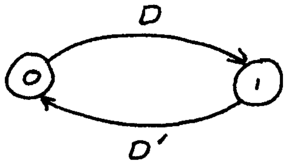
27.


clk
X15
X14
X13
X12
Q3
Q2
Q1
Q0
LD
D3
D2
D1
D0
X11
X10
X9
X8
X7
X6
X5
X4
X3
X2
X1
X0
0
0
0
0
dshl

Chapter 2

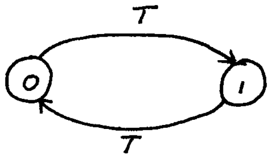
1. a)

Present State	*D*	Next State
0	0	0
0	1	1
1	0	0
1	1	1



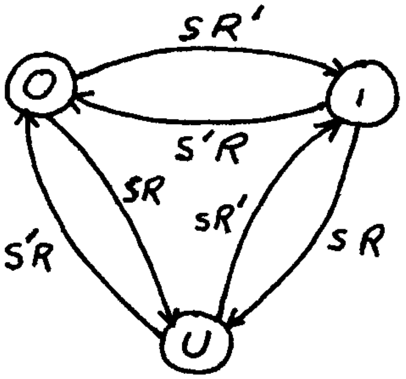
b)

Present State	*T*	Next State
0	0	0
0	1	1
1	0	1
1	1	0



2.

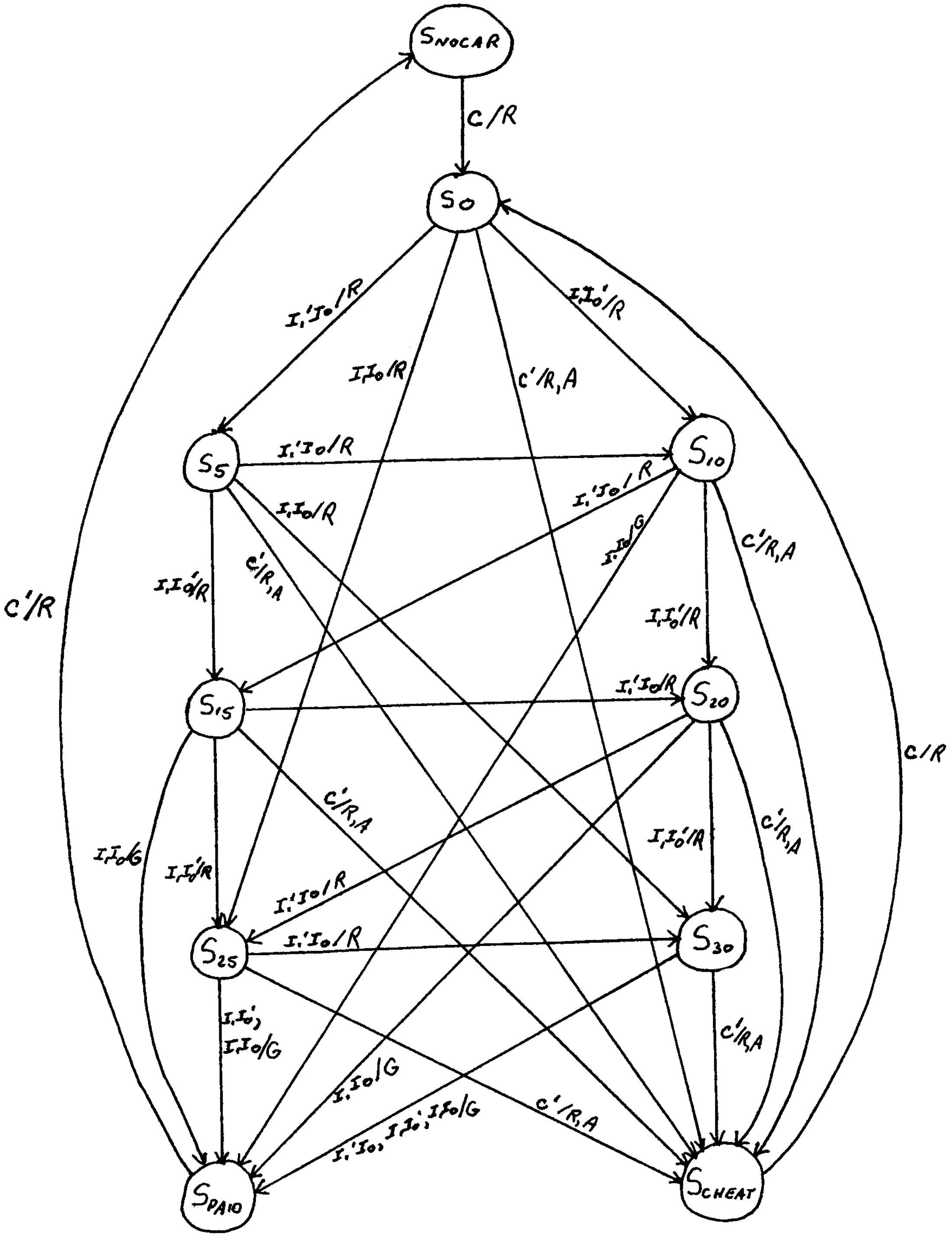
Present State	*S*	*R*	Next State
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	U
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	U
U	0	0	U
U	0	1	0
U	1	0	1
U	1	1	U



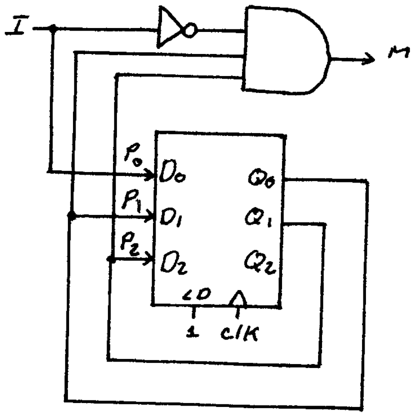
3. Add the following states to the state table. Since all additions are self-loops, it is not necessary to change the state diagram.

Present State	*C*	I_1	I_0	Next State	*R*	*G*	*A*
S_{NOCAR}	0	0	1	S_{NOCAR}	1	0	0
S_{NOCAR}	0	1	0	S_{NOCAR}	1	0	0
S_{NOCAR}	0	1	1	S_{NOCAR}	1	0	0
SP_{AID}	1	0	1	SP_{AID}	0	1	0
SP_{AID}	1	1	0	SP_{AID}	0	1	0
SP_{AID}	1	1	1	SP_{AID}	0	1	0
S_{CHEAT}	0	0	1	S_{CHEAT}	1	0	1
S_{CHEAT}	0	1	0	S_{CHEAT}	1	0	1
S_{CHEAT}	0	1	1	S_{CHEAT}	1	0	1

4.

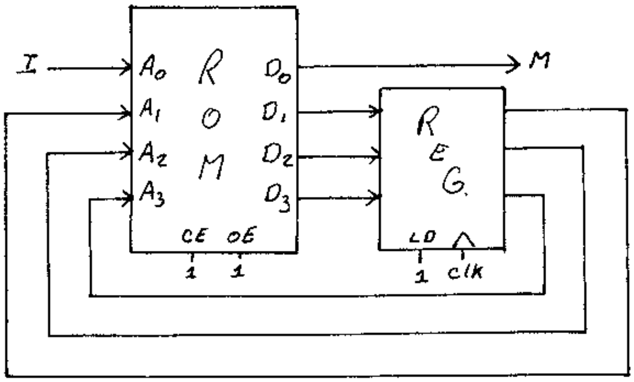

S_{NOCAR}
C/R
S_0
$I_1'I_0/R$
I_1I_0'/R
I_1I_0/R
$C'/R,A$
S_5
$I_1'I_0/R$
S_{10}
I_1I_0/R
$I_1'I_0/R$
I_1I_0/G
$C'/R,A$
$C'/R,A$
I_1I_0'/R
C'/R
I_1I_0'/R
S_{15}
$I_1'I_0/R$
S_{20}
C/R
$C'/R,A$
I_1I_0'/R
$C'/R,A$
I_1I_0/G
I_1I_0'/R
$I_1'I_0/R$
S_{25}
$I_1'I_0/R$
S_{30}
$I_1I_0',$
I_1I_0/G
$C'/R,A$
I_1I_0/G
$I_1'I_0, I_1I_0', I_1I_0/G$
$C'/R,A$
S_{PAID}
S_{CHEAT}

5.



6.

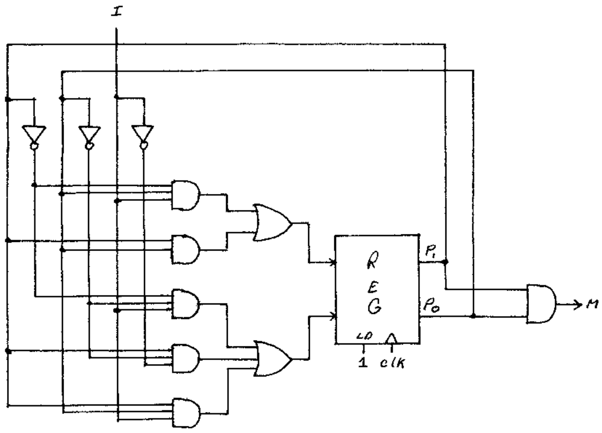
Address	Data (Mealy)	Data (Moore)
0000	0000	0000
0001	0010	0010
0010	0100	0100
0011	0110	0110
0100	1000	1000
0101	1010	1010
0110	1101	1100
0111	1110	1110
1000	0000	0000
1001	0010	0010
1010	0100	0100
1011	0110	0110
1100	1000	1001
1101	1010	1011
1110	1101	1100
1111	1110	1110



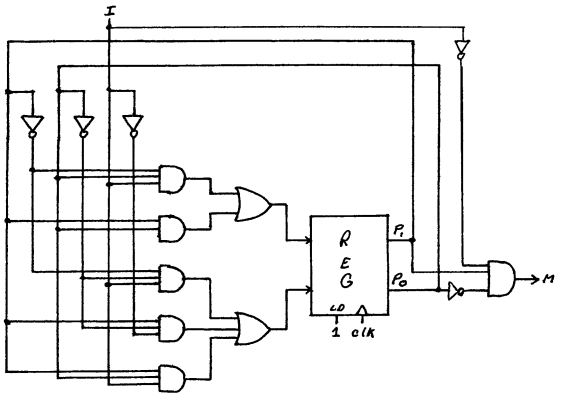
7.

Present State	*I*	Next State	*M*
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	11	0
10	1	10	0
11	0	00	1
11	1	01	1

$N_1 = P_1'P_0I + P_1P_0'$
$N_0 = P_1'P_0'I + P_1P_0'I' + P_1P_0I$
$M = P_1P_0$

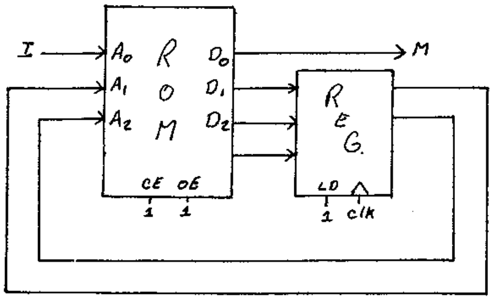


8.



9.

Address	Data (Mealy)	Data (Moore)
000	000	000
001	010	010
010	000	000
011	100	100
100	111	110
101	100	100
110	000	001
111	010	011



10. State value assignments (P_3 - P_0): S_0 = 0000 S_5 = 0001 S_{10} = 0010 S_{15} = 0011 S_{20} = 0100
S_{25} = 0101 S_{30} = 0110 S_{PAID} = 0111 S_{NOCAR} = 1000 S_{CHEAT} = 1001

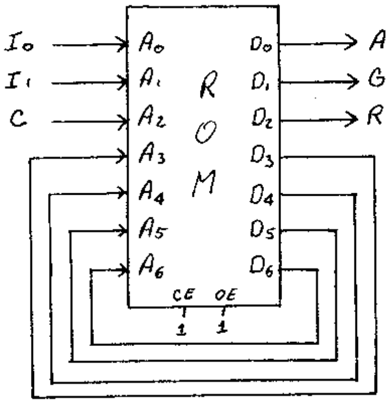
$N_3 = C'$
$N_2 = P_3'CI_1I_0 + P_3'(P_2 + P_1)CI_1I_0' + P_3'(P_2 + P_1P_0)CI_1'I_0 + P_2CI_1'I_0'$
$N_1 = P_3'(P_2 + P_1 + P_0)CI_1I_0 + P_3'(P_2 + P_1')CI_1I_0' + P_3'(P_1'P_0 + P_1P_0' + P_2P_1P_0)CI_1'I_0 + P_1P_0CI_1'I_0'$
$N_0 = P_3'(P_2 + P_1 + P_0')CI_1I_0 + P_3'(P_0 + P_2P_1)CI_1I_0' + P_3'(P_0' + P_2P_1)CI_1'I_0 + P_3'P_0CI_1'I_0' + P_3P_0C$
$\quad + P_3'(P_2' + P_1' + P_0')C'$
$R = S_{PAID}'$
$G = S_{PAID}$
$A = S_{CHEAT}$

11. State value assignments (P_3 - P_0): S_0 = 0000 S_5 = 0001 S_{10} = 0010 S_{15} = 0011 S_{20} = 0100
S_{25} = 0101 S_{30} = 0110 S_{PAID} = 0111 S_{NOCAR} = 1000 S_{CHEAT} = 1001

$N_3 = C'$
$N_2 = P_3'CI_1I_0 + P_3'(P_2 + P_1)CI_1I_0' + P_3'(P_2 + P_1P_0)CI_1'I_0 + P_2CI_1'I_0'$
$N_1 = P_3'(P_2 + P_1 + P_0)CI_1I_0 + P_3'(P_2 + P_1')CI_1I_0' + P_3'(P_1'P_0 + P_1P_0' + P_2P_1P_0)CI_1'I_0 + P_1P_0CI_1'I_0'$
$N_0 = P_3'(P_2 + P_1 + P_0')CI_1I_0 + P_3'(P_0 + P_2P_1)CI_1I_0' + P_3'(P_0' + P_2P_1)CI_1'I_0 + P_3'P_0CI_1'I_0' + P_3P_0C$
$\quad + P_3'(P_2' + P_1' + P_0')C'$
$R = G'$
$G = P_3'(P_2 + P_1)CI_1I_0 + P_3'P_2P_0CI_1 + P_3'P_2P_1C(I_1 + I_0) + P_3'P_2P_1P_0C$
$A = P_3'(P_2 + P_1 + P_0)C'$

12.

Address	Data
0000XXX	1001101 1001101 1001101 1001101 0000100 0001100 0010100 0101100
0001XXX	1001101 1001101 1001101 1001101 0001100 0010100 0011100 0110100
0010XXX	1001101 1001101 1001101 1001101 0010100 0011100 0100100 0111010
0011XXX	1001101 1001101 1001101 1001101 0011100 0100100 0101100 0111010
0100XXX	1001101 1001101 1001101 1001101 0100100 0101100 0110100 0111010
0101XXX	1001101 1001101 1001101 1001101 0101100 0110100 0111010 0111010
0110XXX	1001101 1001101 1001101 1001101 0110100 0111010 0111010 0111010
0111XXX	1000100 1000100 1000100 1000100 0111010 0111010 0111010 0111010
1000XXX	1000100 1000100 1000100 1000100 0000100 0000100 0000100 0000100
1001XXX	1001101 1001101 1001101 1001101 0000100 0000100 0000100 0000100
1010XXX	1000100 1000100 1000100 1000100 1000100 1000100 1000100 1000100
1011XXX	1000100 1000100 1000100 1000100 1000100 1000100 1000100 1000100
1100XXX	1000100 1000100 1000100 1000100 1000100 1000100 1000100 1000100
1101XXX	1000100 1000100 1000100 1000100 1000100 1000100 1000100 1000100
1110XXX	1000100 1000100 1000100 1000100 1000100 1000100 1000100 1000100
1111XXX	1000100 1000100 1000100 1000100 1000100 1000100 1000100 1000100



13. N_2:

$P_2P_1 \backslash P_0U$	00	01	11	10
00	0	0	0	0
01	0	0	1	0
11	0	0	0	0
10	1	1	0	1

N_1:

$P_2P_1 \backslash P_0U$	00	01	11	10
00	0	0	1	0
01	1	1	0	1
11	0	0	0	0
10	0	0	0	0

N_2:

$P_2P_1 \backslash P_0U$	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	0	0	0
10	0	1	0	1

14. The next state logic is the same as for the Moore machine.

$$N_2 = P_2P_0' + P_2U' + P_1P_0U$$
$$N_1 = P_1P_0' + P_1U' + P_2'P_1'P_0U$$
$$N_0 = P_0'U + P_0U'$$
$$C = P_2'P_1'P_0'U' + P_2P_1'P_0U$$
$$V_2 = P_2'P_1P_0U + P_2P_1'P_0' + P_2P_1'P_0U'$$
$$V_1 = P_2'P_1'P_0U + P_2'P_1P_0' + P_2'P_1P_0U'$$
$$V_0 = (P_2' + P_1')P_0'U + (P_2' + P_1)P_0U'$$

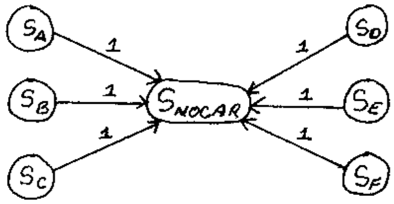
15. All possible next state values are already used.

16. State value assignments (P_3 - P_0): $S_0 = 0000$ $S_5 = 0001$ $S_{10} = 0010$ $S_{15} = 0011$ $S_{20} = 0100$
$S_{25} = 0101$ $S_{30} = 0110$ $S_{PAID} = 0111$ $S_{NOCAR} = 1000$ $S_{CHEAT} = 1001$ $S_A = 1010$
$S_B = 1011$ $S_C = 1100$ $S_D = 1101$ $S_E = 1110$ $S_F = 1111$

Add to state table:

Present State	C	I_1	I_0	Next State	R	G	A
1010	X	X	X	1000	0	0	0
1011	X	X	X	1000	0	0	0
1100	X	X	X	1000	0	0	0
1101	X	X	X	1000	0	0	0
1110	X	X	X	1000	0	0	0
1111	X	X	X	1000	0	0	0

Add to state diagram:



$N_3 = C' + P_3(P_2 + P_1)$

$N_2 = P_3'CI_1I_0 + P_3'(P_2 + P_1)CI_1I_0' + P_3'(P_2 + P_1P_0)CI_1'I_0 + P_2CI_1'I_0'$

$N_1 = P_3'(P_2 + P_1 + P_0)CI_1I_0 + P_3'(P_2 + P_1')CI_1I_0' + P_3'(P_1'P_0 + P_1P_0' + P_2P_1P_0)CI_1'I_0 + P_1P_0CI_1'I_0'$

$N_0 = P_3'(P_2 + P_1 + P_0')CI_1I_0 + P_3'(P_0 + P_2P_1)CI_1I_0' + P_3'(P_0' + P_2P_1)CI_1'I_0 + P_3'P_0CI_1'I_0' + P_3P_0C$
$+ P_3'(P_2' + P_1' + P_0')C'$

$R = S_{PAID}'$

$G = S_{PAID}$

$A = S_{CHEAT}$

17. $N_3 = P_2P_1P_0U' + P_3(P_2' + P_1' + P_0' + U)$

$N_2 = P_3P_2(P_0 + U) + P_2P_1 + P_3'P_1P_0'U'$

$N_1 = P_3' (P_2 + P_1)U' + P_2P_1P_0U' + P_1U$

$N_0 = (P_3' + P_2)P_1'U + P_3'P_2U + P_0U'$

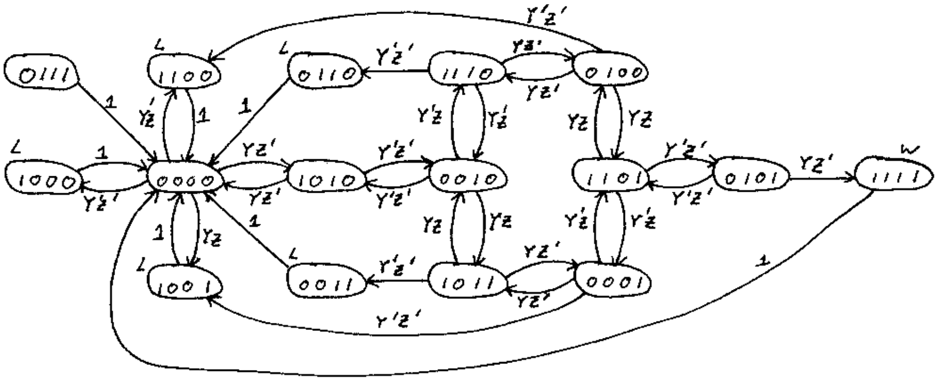
$C = P_2'P_1'P_0'$

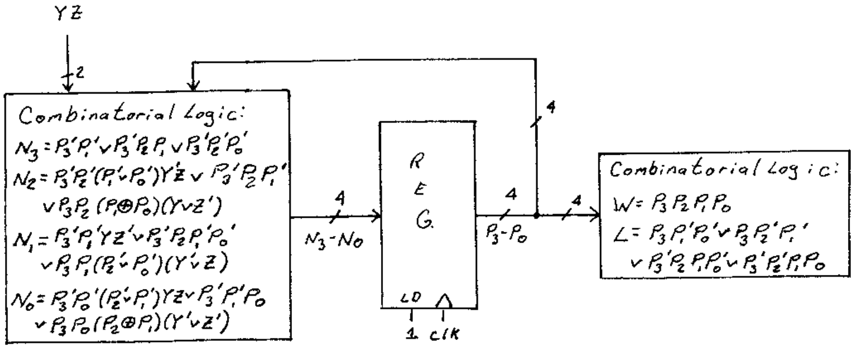
$V_2 = P_3P_1'P_0 + P_3P_2P_0'$

$V_1 = P_3'P_1P_0' + P_2P_1P_0$

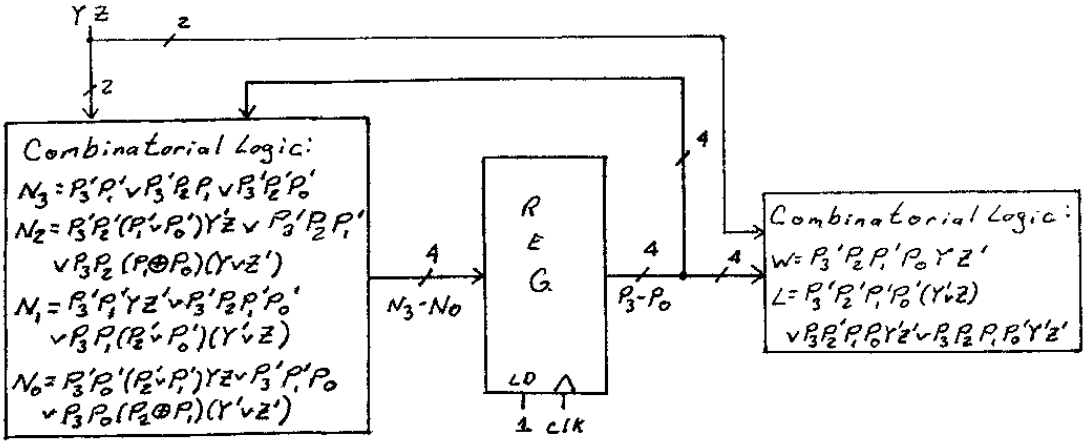
$V_0 = P_3'P_2'P_0 + P_2P_1P_0 + P_3P_1'P_0$

18.

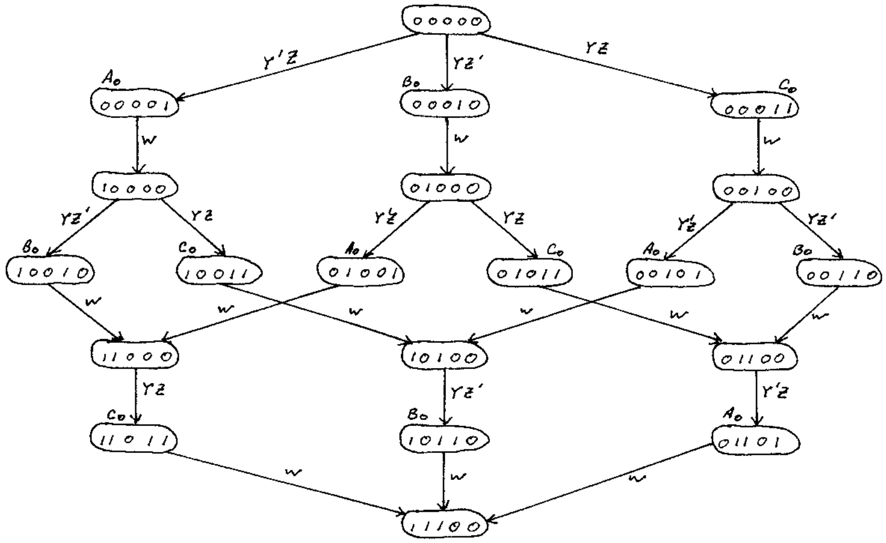

0111
1100
0110
1110
0100
1000
0000
1010
0010
1101
0101
1111
1001
0011
1011
0001
L
W
1
Y'Z'
YZ'
Y'Z
YZ

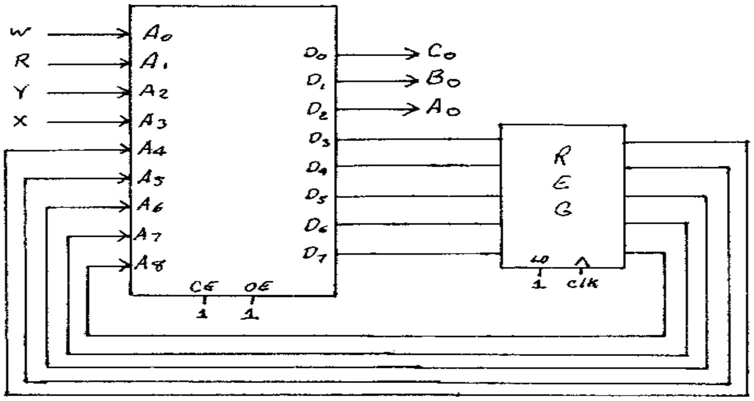

YZ
2
Combinatorial Logic:
$N_3 = P_3'P_1' \vee P_3'P_2P_1 \vee P_3'P_2'P_0'$
$N_2 = P_3'P_2'(P_1' \vee P_0')Y'Z \vee P_3'P_2P_1' \vee P_3P_2(P_1 \oplus P_0)(Y \vee Z')$
$N_1 = P_3'P_1'YZ' \vee P_3'P_2P_1'P_0' \vee P_3P_1(P_2' \vee P_0')(Y' \vee Z)$
$N_0 = P_3'P_0'(P_2' \vee P_1')YZ \vee P_3'P_1'P_0 \vee P_3P_0(P_2 \oplus P_1)(Y' \vee Z')$
4
N_3-N_0
R
E
G
LD
1
clk
4
P_3-P_0
4
4
Combinatorial Logic:
$W = P_3P_2P_1P_0$
$L = P_3P_1'P_0' \vee P_3P_2'P_1' \vee P_3'P_2P_1P_0' \vee P_3'P_2'P_1P_0$

19.


YZ
2
2
Combinatorial Logic:
$N_3 = P_3'P_1' \vee P_3'P_2P_1 \vee P_3'P_2'P_0'$
$N_2 = P_3'P_2'(P_1' \vee P_0')Y'Z \vee P_3'P_2P_1' \vee P_3P_2(P_1 \oplus P_0)(Y \vee Z')$
$N_1 = P_3'P_1'YZ' \vee P_3'P_2P_1'P_0' \vee P_3P_1(P_2' \vee P_0')(Y' \vee Z)$
$N_0 = P_3'P_0'(P_2' \vee P_1')YZ \vee P_3'P_1'P_0 \vee P_3P_0(P_2 \oplus P_1)(Y' \vee Z')$
4
N_3-N_0
R
E
G
LD
1
clk
4
P_3-P_0
4
4
Combinatorial Logic:
$W = P_3'P_2P_1'P_0YZ'$
$L = P_3'P_2'P_1'P_0'(Y' \vee Z) \vee P_3P_2'P_1P_0Y'Z' \vee P_3P_2P_1P_0'Y'Z'$

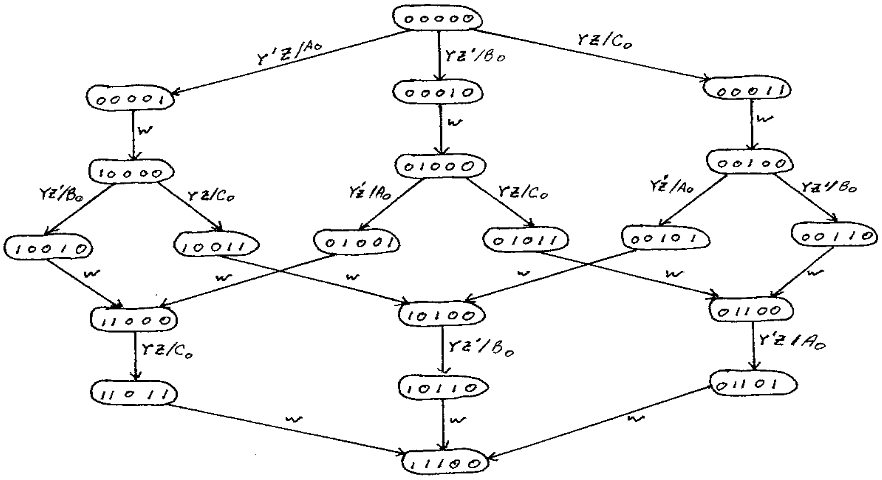
20. States are of the form *ABCYZ*, where *A*|*B*|*C* = 0 if a player may signal, or 1 if the player may not signal. *YZ* represents the player answering the question (01 = player 1, 10 = player 2, 11 = player 3, 00 = no player). Although not shown in the diagram, there is an arc from every state back to state 00000 with condition *R*.

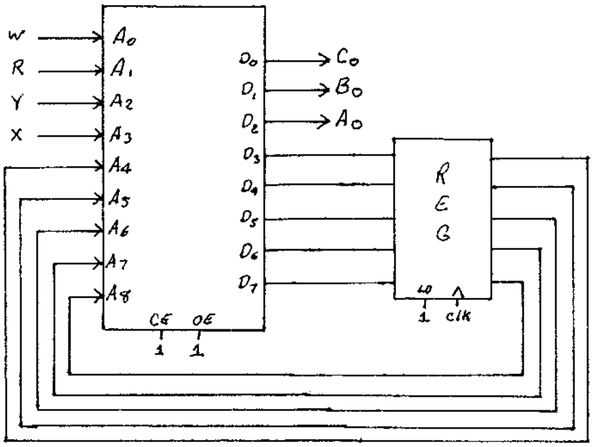




Address	Data	Address	Data	Address	Data
XXXXX XX1X	00000 000	00110 XX00	00110 010	10010 XX00	10010 010
00000 000X	00000 000	00110 XX01	01100 010	10010 XX01	11000 010
00000 010X	00001 000	01000 X00X	01000 000	10011 XX00	10011 001
00000 100X	00010 000	01000 010X	01001 000	10011 XX01	10100 001
00000 110X	00011 000	01000 110X	01011 000	10100 0X0X	10100 000
00001 XX00	00001 100	01001 XX00	01001 100	10100 100X	10110 000
00001 XX01	10000 100	01001 XX01	11000 100	10100 110X	10100 000
00010 XX00	00010 010	01011 XX00	01011 001	10110 XX00	10110 010
00010 XX01	01000 010	01011 XX01	01100 001	10110 XX01	11100 010
00011 XX00	00011 001	01100 000X	01100 000	11000 0X0X	11000 000
00011 XX01	00100 001	01100 010X	01101 000	11000 100X	11000 000
00100 000X	00100 000	01100 1X0X	01100 000	11000 110X	11011 000
00100 010X	00101 000	01101 XX00	01101 100	11011 XX00	11011 001
00100 100X	00110 000	01101 XX01	11100 100	11011 XX01	11100 001
00100 110X	00100 000	10000 0X0X	10000 000	11100 XX0X	11100 000
00101 XX00	00101 100	10000 100X	10010 000	All others	00000 000
00101 XX01	10100 100	10000 110X	10011 000		

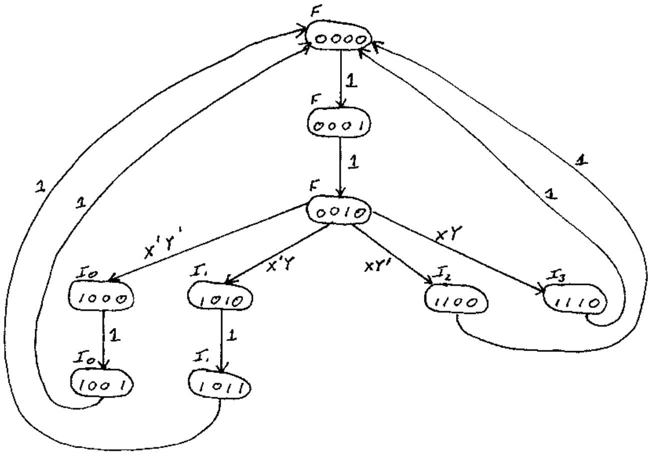
21. States are of the form *ABCYZ*, where *A*|*B*|*C* = 0 if a player may signal, or 1 if the player may not signal. *YZ* represents the player answering the question (01 = player 1, 10 = player 2, 11 = player 3, 00 = no player). Although not shown in the diagram, there is an arc from every state back to state 00000 with condition *R*.



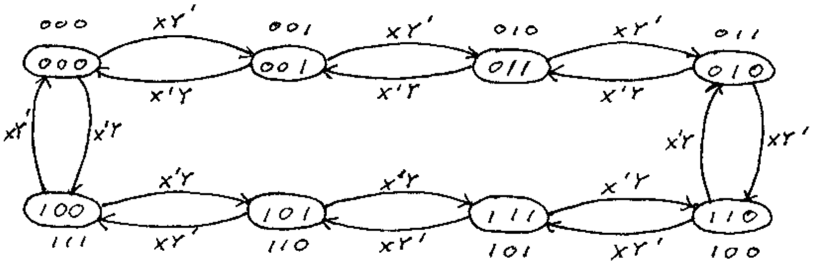


Address	Data	Address	Data	Address	Data
XXXXX XX1X	00000 000	00110 XX00	00110 010	10010 XX00	10010 010
00000 000X	00000 000	00110 XX01	01100 000	10010 XX01	11000 000
00000 010X	00001 100	01000 X00X	01000 000	10011 XX00	10011 001
00000 100X	00010 010	01000 010X	01001 100	10011 XX01	10100 000
00000 110X	00011 001	01000 110X	01011 001	10100 0X0X	10100 000
00001 XX00	00001 100	01001 XX00	01001 100	10100 100X	10110 010
00001 XX01	10000 000	01001 XX01	11000 000	10100 110X	10100 000
00010 XX00	00010 010	01011 XX00	01011 001	10110 XX00	10110 010
00010 XX01	01000 000	01011 XX01	01100 000	10110 XX01	11100 000
00011 XX00	00011 001	01100 000X	01100 000	11000 0X0X	11000 000
00011 XX01	00100 000	01100 010X	01101 100	11000 100X	11000 000
00100 000X	00100 000	01100 1X0X	01100 000	11000 110X	11011 001
00100 010X	00101 100	01101 XX00	01101 100	11011 XX00	11011 001
00100 100X	00110 010	01101 XX01	11100 000	11011 XX01	11100 000
00100 110X	00100 000	10000 0X0X	10000 000	11100 XX0X	11100 000
00101 XX00	00101 100	10000 100X	10010 010	All others	00000 000
00101 XX01	10100 000	10000 110X	10011 001		

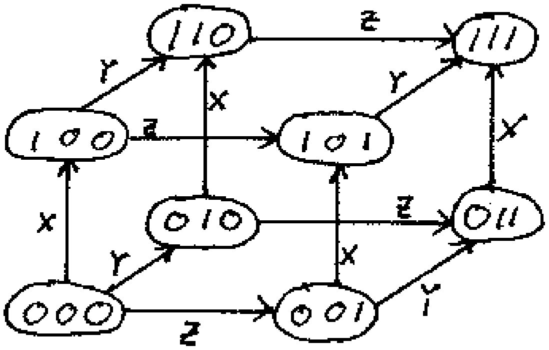
22.



23.



24.



25. P_1: $P_0X'Y$ should be P_0XY
P_0: P_0XY' should be $P_0X'Y'$
B: P_0 should be P_0'

26. *CLR*: 0*XY'* should be 1*XY'*
Counter input D_0: 0*X'Y* should be 0*XY*
A: 1 should be 0

27.

Address	Correct Data
0011	0<u>1</u>110
0100	0101<u>1</u>
1011	<u>0</u>0111

Chapter 3

1. a) Data movement b) Data operation c) Program control d) Data operation e) Data operation

2. a) Data operation b) Program control c) Data movement d) Data movement e) Data operation

3. a) Direct b) Implied c) Implicit

4. a) Implicit b) Direct c) Implicit

5. a) Implicit b) Direct c) Implicit

6. a) Register Direct b) Immediate c) Implicit d) Immediate e) Direct

7. a) Implicit b) Direct c) Indirect d) Register Indirect e) Register Direct

8. a) Register Direct b) Register Indirect c) Implicit d) Implicit e) Immediate

9. a) $AC = 11$ b) $AC = 12$ c) $AC = 10$ d) $AC = 11$ e) $AC = 10$ f) $AC = 33$ g) $AC = 41$

10. a) $AC = 11$ b) $AC = 12$ c) $AC = 30$ d) $AC = 31$ e) $AC = 10$ f) $AC = 23$ g) $AC = 31$

11. a) $AC = 11$ b) $AC = 12$ c) $AC = 20$ d) $AC = 21$ e) $AC = 10$ f) $AC = 43$ g) $AC = 21$

```
12.   a) MUL X,B,C    b) MOV X,B     c) LOAD B      d) PUSH A
         ADD X,X,A       MUL X,C        MUL C          PUSH B
         ADD X,X,D       ADD X,A        ADD A          PUSH C
                         ADD X,D        ADD D          MUL
                                        STORE X        PUSH D
                                                       ADD
                                                       ADD
                                                       POP X
```

```
13.   a) MUL T,A,B    b) MOV T,A     c) LOAD A      d) PUSH A
         MUL T,T,C       MUL T,B        MUL B          PUSH B
         ADD X,E,F       MUL T,C        MUL C          MUL
         MUL X,X,D       MOV X,E        STORE T        PUSH C
         ADD X,X,T       ADD X,F        LOAD E         MUL
                         MUL X,D        ADD F          PUSH D
                         ADD X,T        MULT D         PUSH E
                                        ADD T          PUSH F
                                        STORE X        ADD
                                                       MUL
                                                       ADD
                                                       POP X
```

14.

a)

```
MUL X,B,C
SUB X,A,X
MUL T,E,F
ADD T,T,D
MUL X,X,T
```

b)

```
MOV T,B
MUL T,C
MOV X,A
SUB X,T
MOV T,E
MUL T,F
ADD T,D
MUL X,T
```

c)

```
LOAD B
MUL C
STORE T
LOAD A
SUB T
STORE X
LOAD E
MUL F
ADD D
MUL X
STORE X
```

d)

```
PUSH A
PUSH B
PUSH C
MUL
SUB
PUSH D
PUSH E
PUSH F
MUL
ADD
MUL
POP X
```

15.

Processor	Time per instruction	# Instructions	Total time	
0	35 ns	4	140 ns	
1	50 ns	3	150 ns	
2	70 ns	2	140 ns	
3	100 ns	1	100 ns	fastest

16.

Processor	Time per instruction	# Instructions	Total time	
0	35 ns	8	280 ns	
1	50 ns	5	250 ns	fastest
2	70 ns	4	280 ns	
3	100 ns	3	300 ns	

17.

Processor	Time per instruction	# Instructions	Total time	
0	35 ns	12	420 ns	fastest
1	50 ns	9	450 ns	
2	70 ns	7	490 ns	
3	100 ns	5	500 ns	

18.

Processor	Time per instruction	# Instructions	Total time	
0	35 ns	12	420 ns	fastest
1	50 ns	11	550 ns	
2	70 ns	8	560 ns	
3	100 ns	5	500 ns	

19.

```
LDAC 1001H
MVAC
LDAC 1002H
ADD
MVAC
LDAC 1003H
ADD
MVAC
LDAC 1004H
ADD
MVAC
LDAC 1005H
ADD
MVAC
LDAC 1006H
ADD
MVAC
LDAC 1007H
ADD
MVAC
LDAC 1008H
ADD
MVAC
LDAC 1009H
ADD
MVAC
LDAC 100AH
ADD
STAC 1000H
```

20.

```
        LXI H, 1001H
        MVI B,0AH
        XRA A
Loop:   ADD M
        INX H
        DCR B
        JNZ Loop
        STA 1000H
```

21.

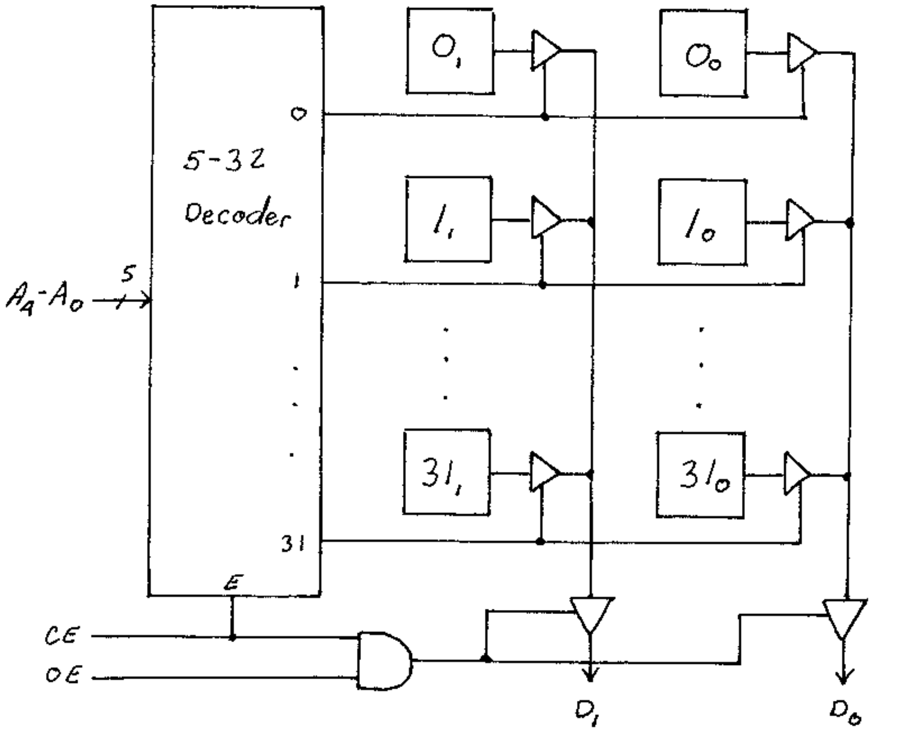
```
        CLAC
        INAC
        STAC FA         FA = 1
        INAC
        STAC FB         FB = 2
        STAC Count      Count = 2
        STAC FN         FN = 2
        MVAC
        LDAC
        SUB
        JMPZ Done       If n = 2 then done
Loop:   LDAC FA
        MVAC
        LDAC FB
        ADD
        STAC FA         FA = FA + FB
        LDAC Count
        INAC
        STAC Count      Count = Count + 1
        MVAC
        LDAC n
        SUB             If Count = n then
        JMPZ DoneA       done, result in FA

        LDAC FB
        MVAC
        LDAC FA
        ADD
        STAC FB         FB = FB + FA
        LDAC Count
        INAC
        STAC Count      Count = Count + 1
        MVAC
        LDAC n
        SUB             If Count = n then
        JMPZ DoneB       done, result in FB
        JUMP Loop       Not done, loop back
DoneA:  LDAC FA
        STAC FN         FN = FA
        JUMP Done
DoneB:  LDAC FB
        STAC FN         FN = FB
Done:   …
```

22.

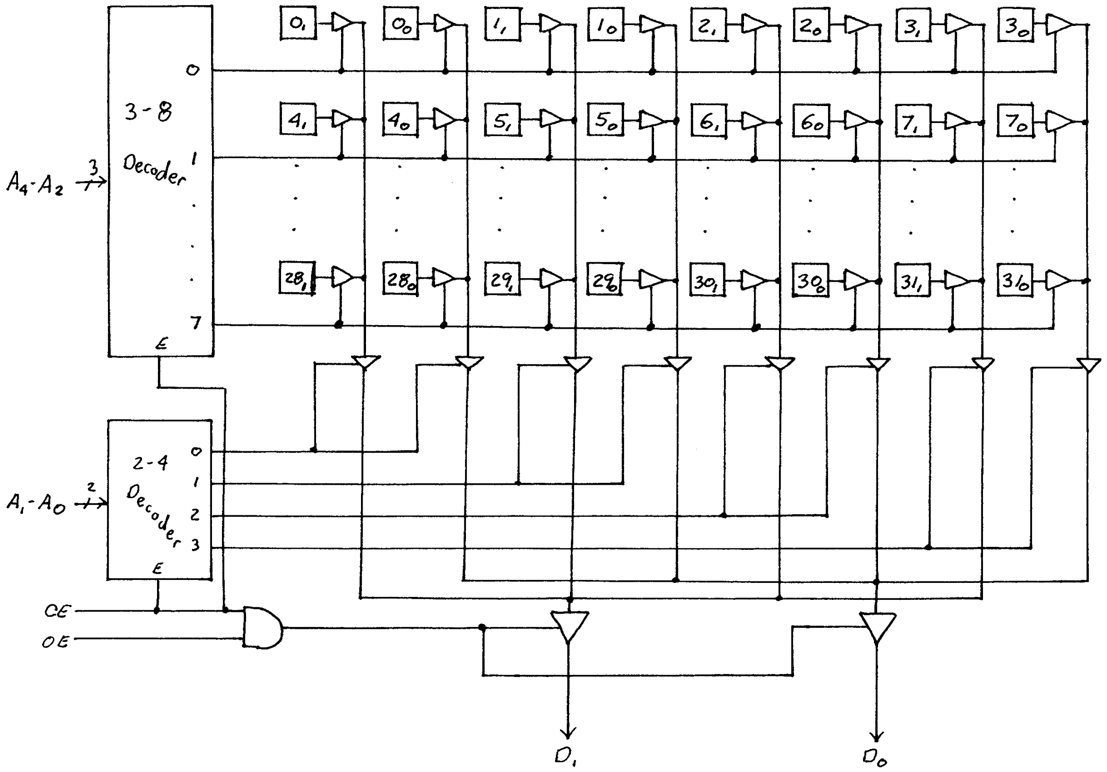
```
        LDA n
        MOV D,A         D = n
        MVI B,1         B = FA
        MVI A,2
        MOV C,A         C = FB
        DCR D
        DCR D
        JZ Done         Initially A = FA
Loop:   ADD B
        MOV B,A         FA = FA + FB
        DCR D           If D = 0 then done
        JZ Done
        ADD C
        MOV C,A         FB = FB + FA
        DCR D           If D = 0 then done
        JNZ Loop        Not done, loop back
Done:   STA FN          Store FN
```

Chapter 4

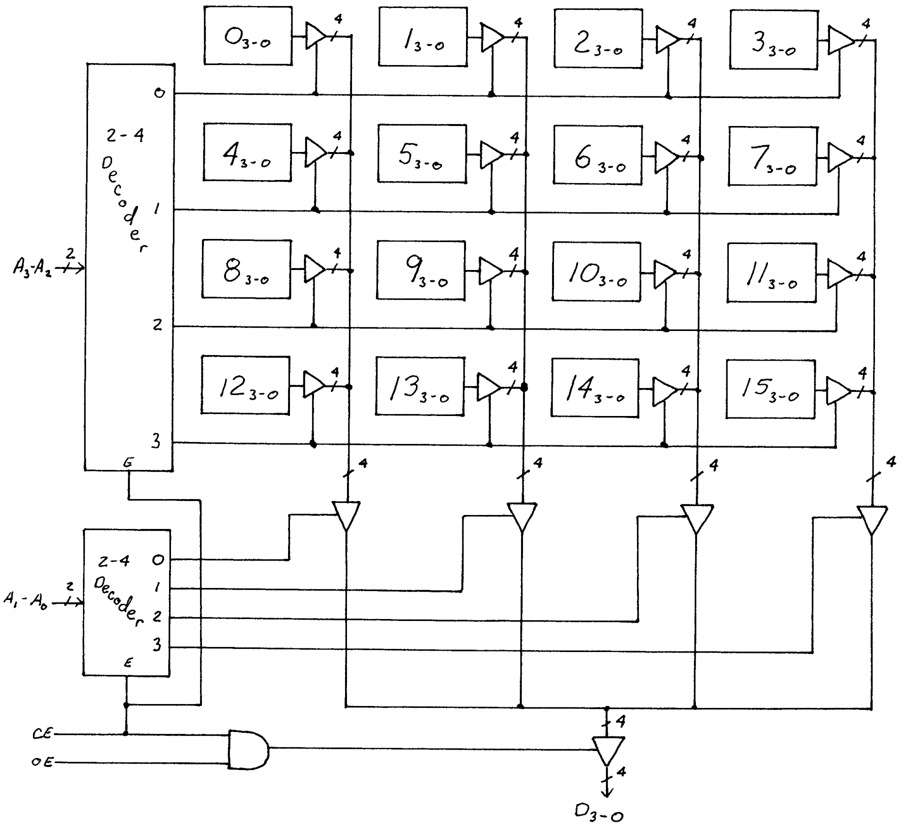
1.


5-32 Decoder
0
1
31
E
A_4-A_0
5
0_1
0_0
1_1
1_0
31_1
31_0
CE
OE
D_1
D_0

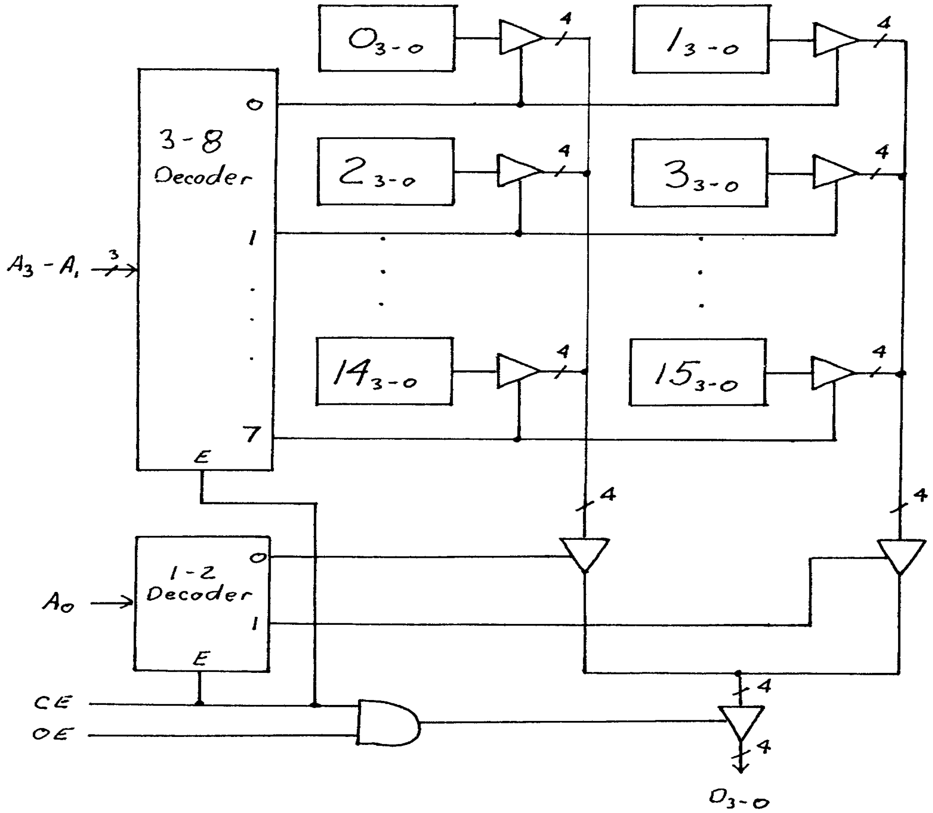
2.


3-8 Decoder
0
1
7
E
A_4-A_2
3
2-4 Decoder
0
1
2
3
E
A_1-A_0
2
0_1
0_0
1_1
1_0
2_1
2_0
3_1
3_0
4_1
4_0
5_1
5_0
6_1
6_0
7_1
7_0
28_1
28_0
29_1
29_0
30_1
30_0
31_1
31_0
CE
OE
D_1
D_0

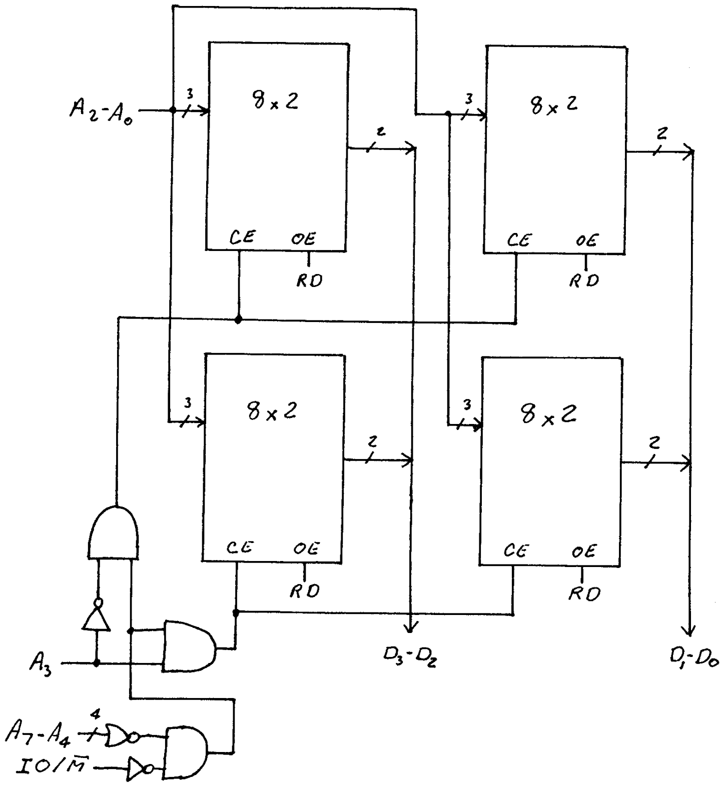
3.


0_{3-0}
1_{3-0}
2_{3-0}
3_{3-0}
4_{3-0}
5_{3-0}
6_{3-0}
7_{3-0}
8_{3-0}
9_{3-0}
10_{3-0}
11_{3-0}
12_{3-0}
13_{3-0}
14_{3-0}
15_{3-0}
4
2-4 Decoder
0
1
2
3
E
A_3-A_2
2
A_1-A_0
CE
OE
O_{3-0}

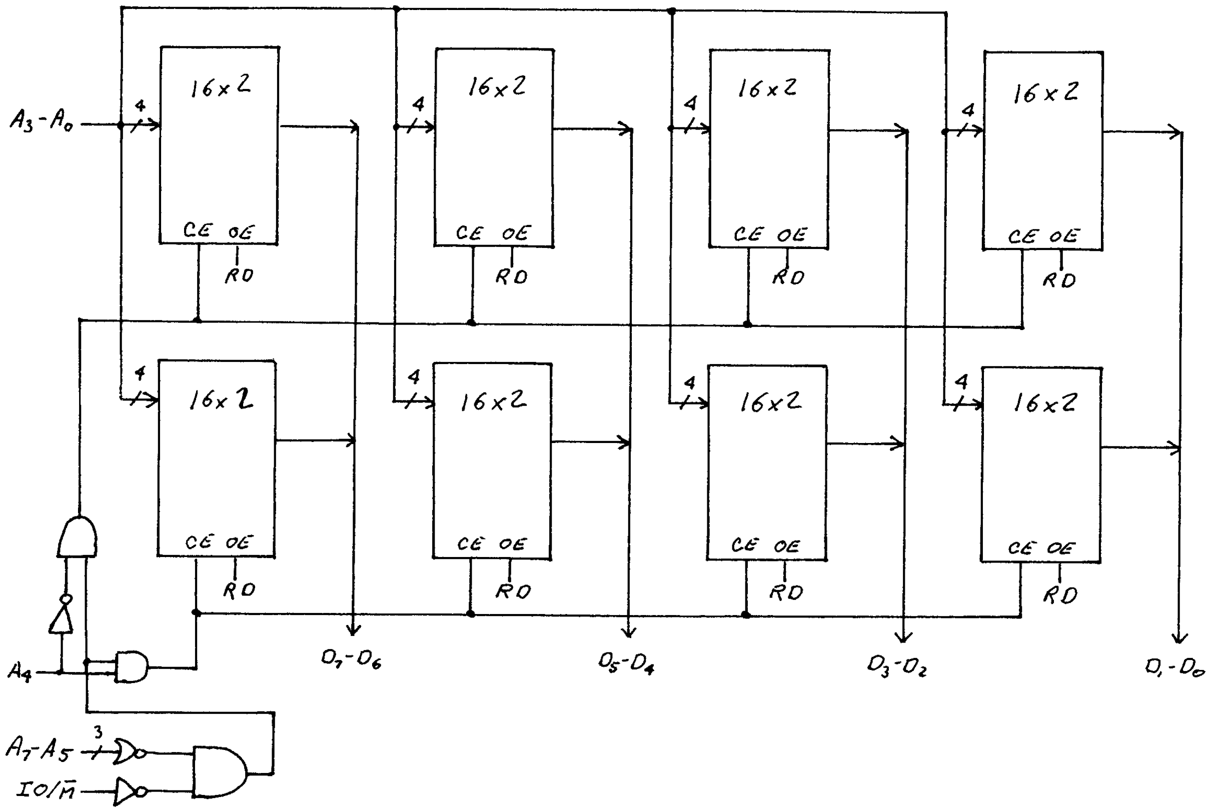
4.


0_{3-0}
1_{3-0}
2_{3-0}
3_{3-0}
14_{3-0}
15_{3-0}
4
3-8 Decoder
0
1
7
E
A_3 - A_1
3
1-2 Decoder
A_0
CE
OE
O_{3-0}

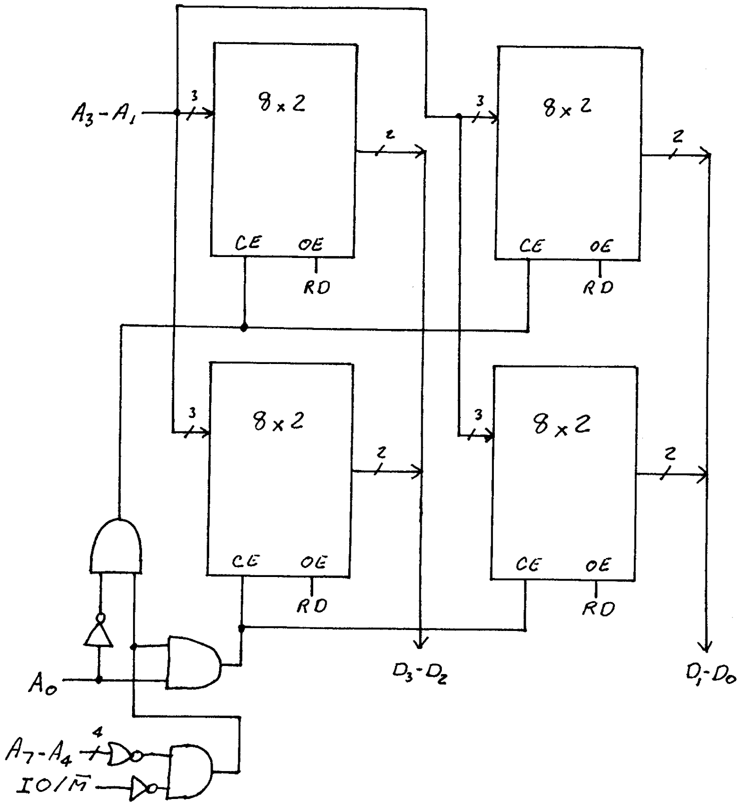
5.


A_2-A_0
3
8 x 2
2
CE
OE
RD
A_3
4
A_7-A_4
IO/$\overline{M}$
D_3-D_2
D_1-D_0

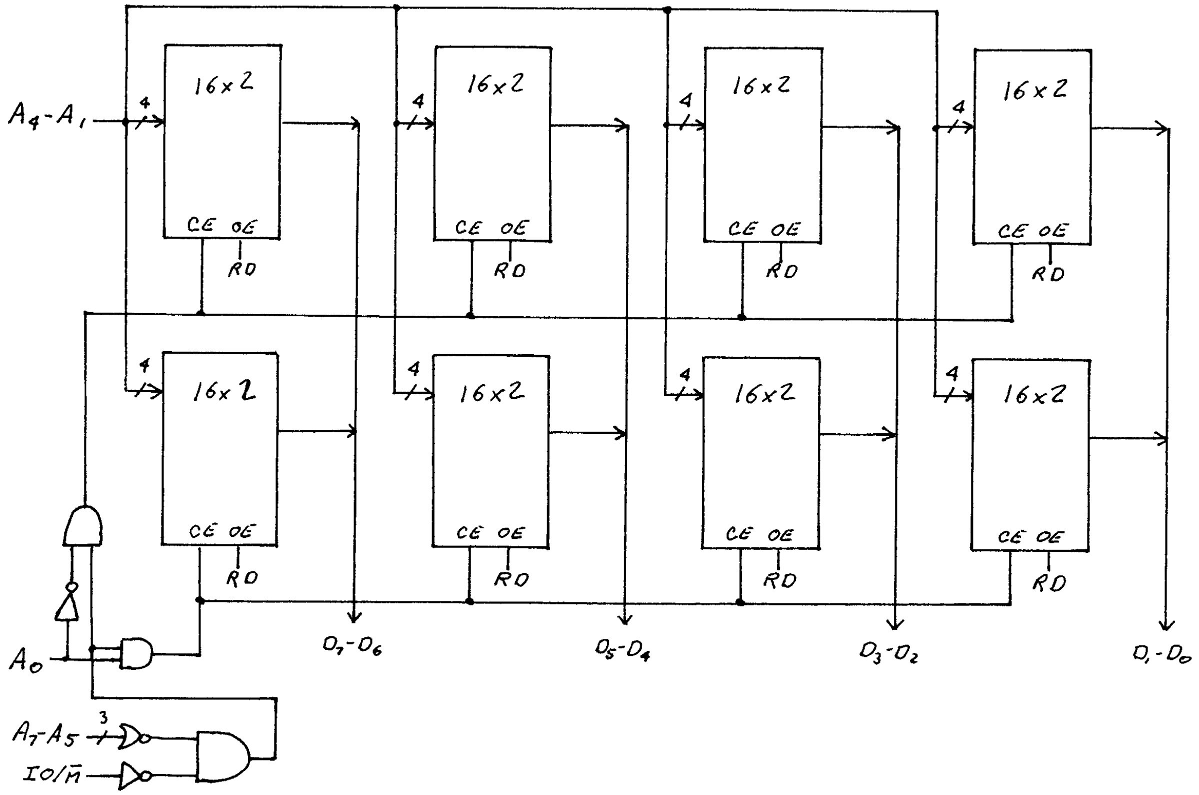
6.


A_3-A_0
4
16x2
CE
OE
RD
A_4
3
A_7-A_5
IO/$\overline{M}$
D_7-D_6
D_5-D_4
D_3-D_2
D_1-D_0

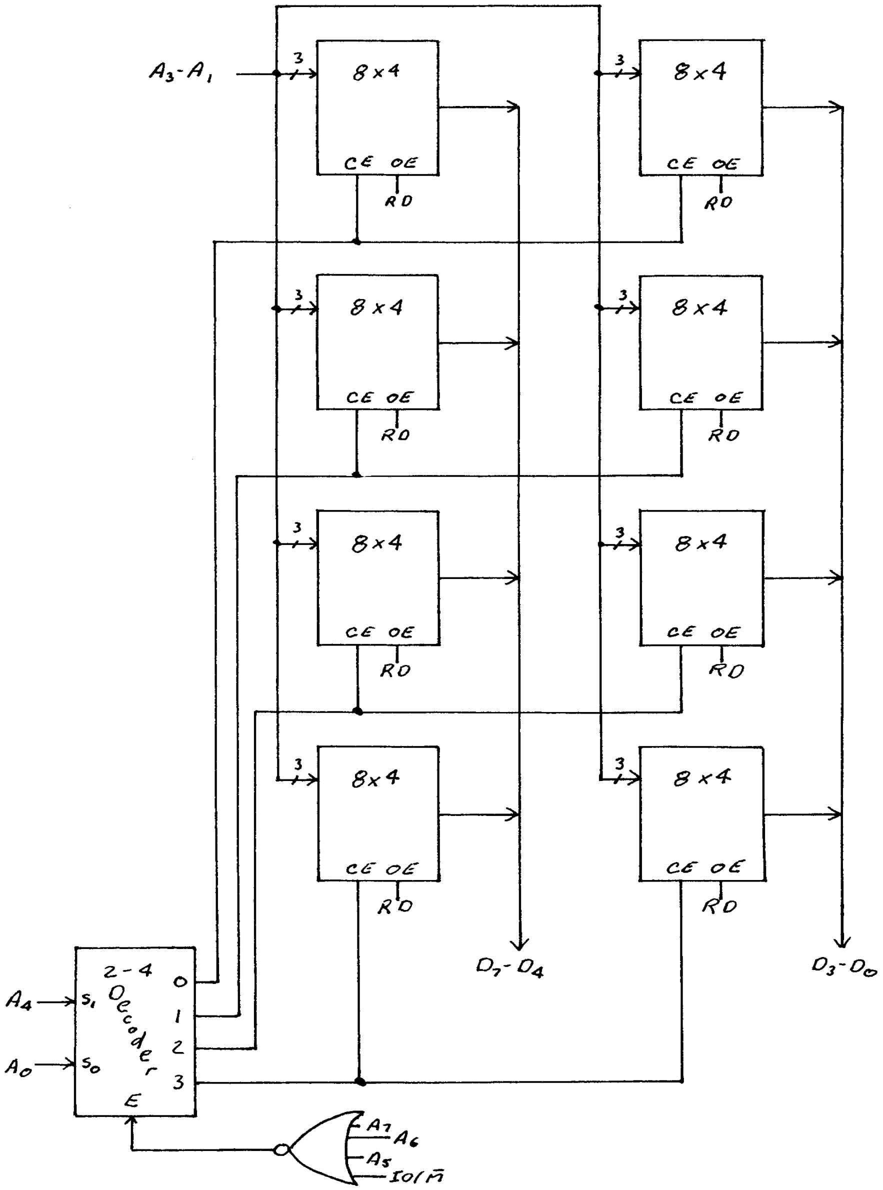
7.


A_3-A_1
3
8 x 2
2
CE
OE
RD
A_0
A_7-A_4
4
$IO/\overline{M}$
D_3-D_2
D_1-D_0

8.


A_4-A_1
4
16 x 2
CE
OE
RD
A_0
A_7-A_5
3
$IO/\overline{M}$
D_7-D_6
D_5-D_4
D_3-D_2
D_1-D_0

9.



10. a.) $CE = A_7'A_6'A_5'A_4'(IO/\overline{M})'$ $OE = RD$

b.) $CE = A_7'A_6'A_5'A_4(IO/\overline{M})'$ $OE = RD$

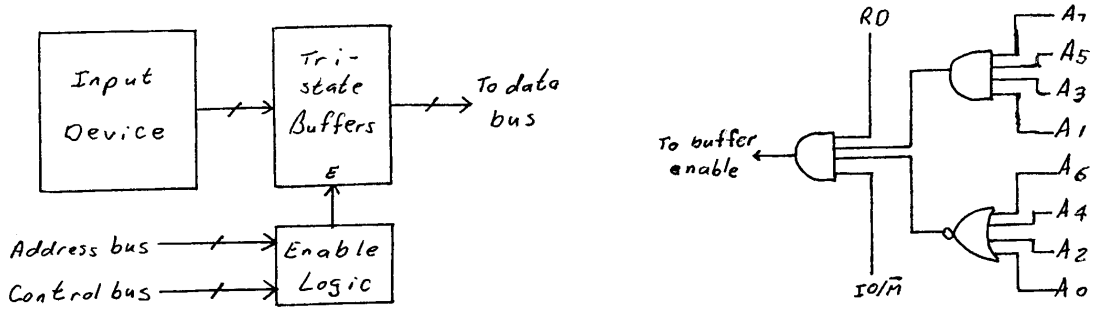
c.) $CE = A_7A_6A_5A_4(IO/\overline{M})'$ $OE = RD$

11.

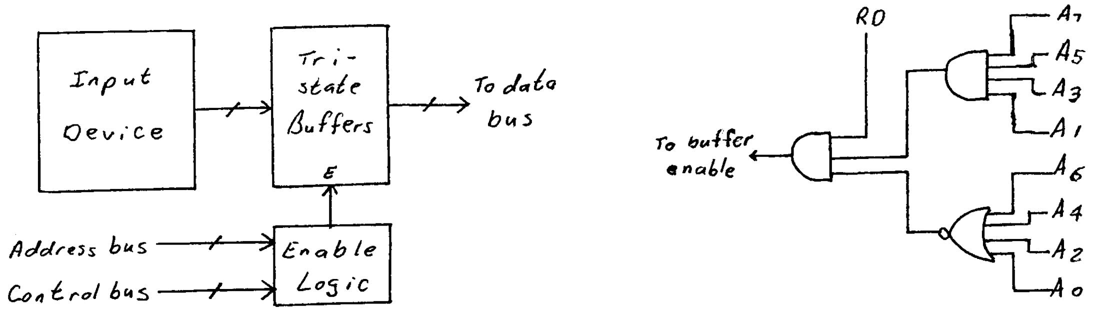
	Big Endian		Little Endian	
a)	22	12H	22	78H
	23	34H	23	56H
	24	56H	24	34H
	25	78H	25	12H
b)	22	09H	22	27H
	23	27H	23	09H
c)	22	05H	22	12H
	23	55H	23	12H
	24	12H	24	55H
	25	12H	25	05H

12. Start each value at location 4*X*, where $X \in I \geq 0$, 20 for example.

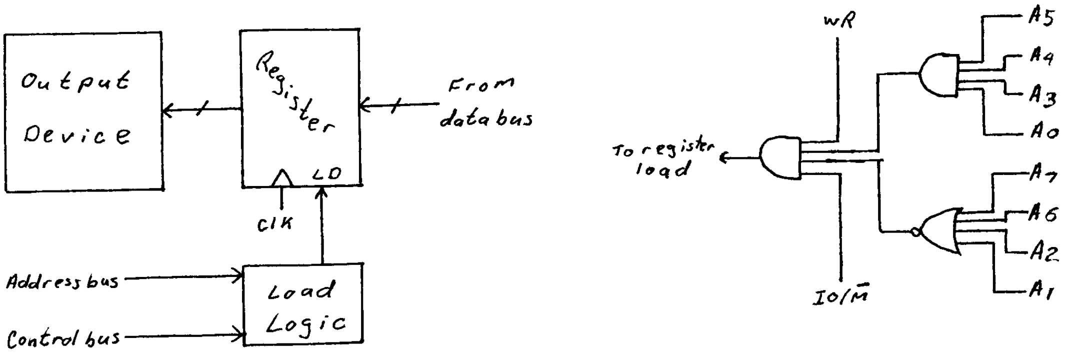
13.



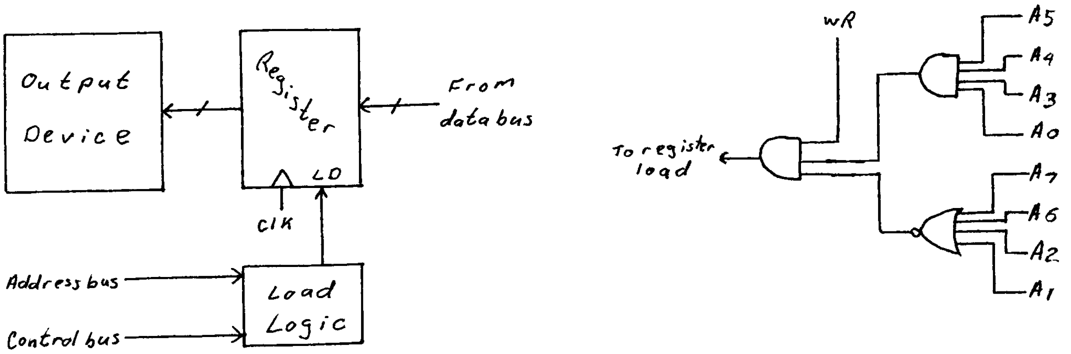
14. This is the same as the previous problem, except $IO/\overline{M}$ is not included.



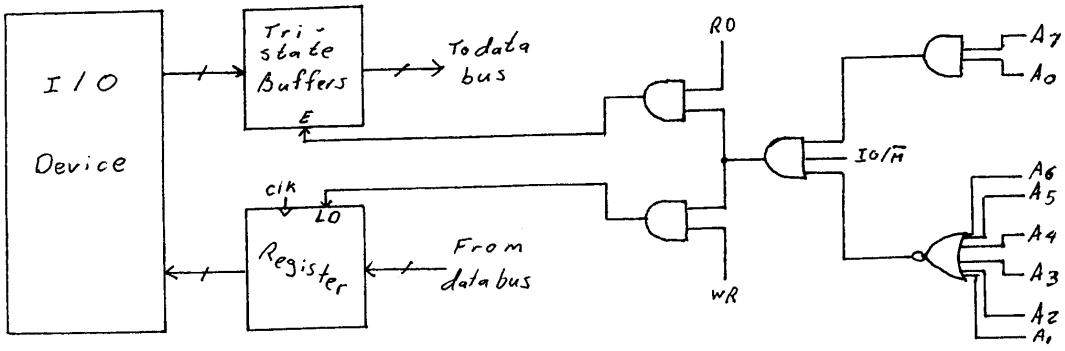
15.



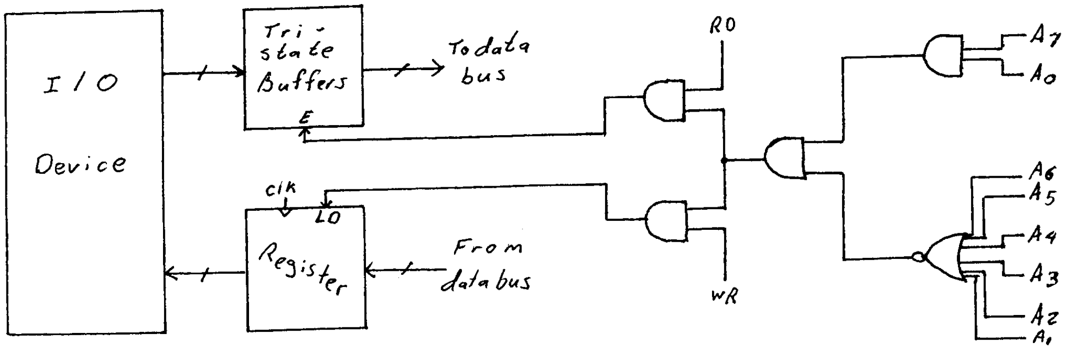
16. This is the same as the previous problem, except $IO/\overline{M}$ is not included.



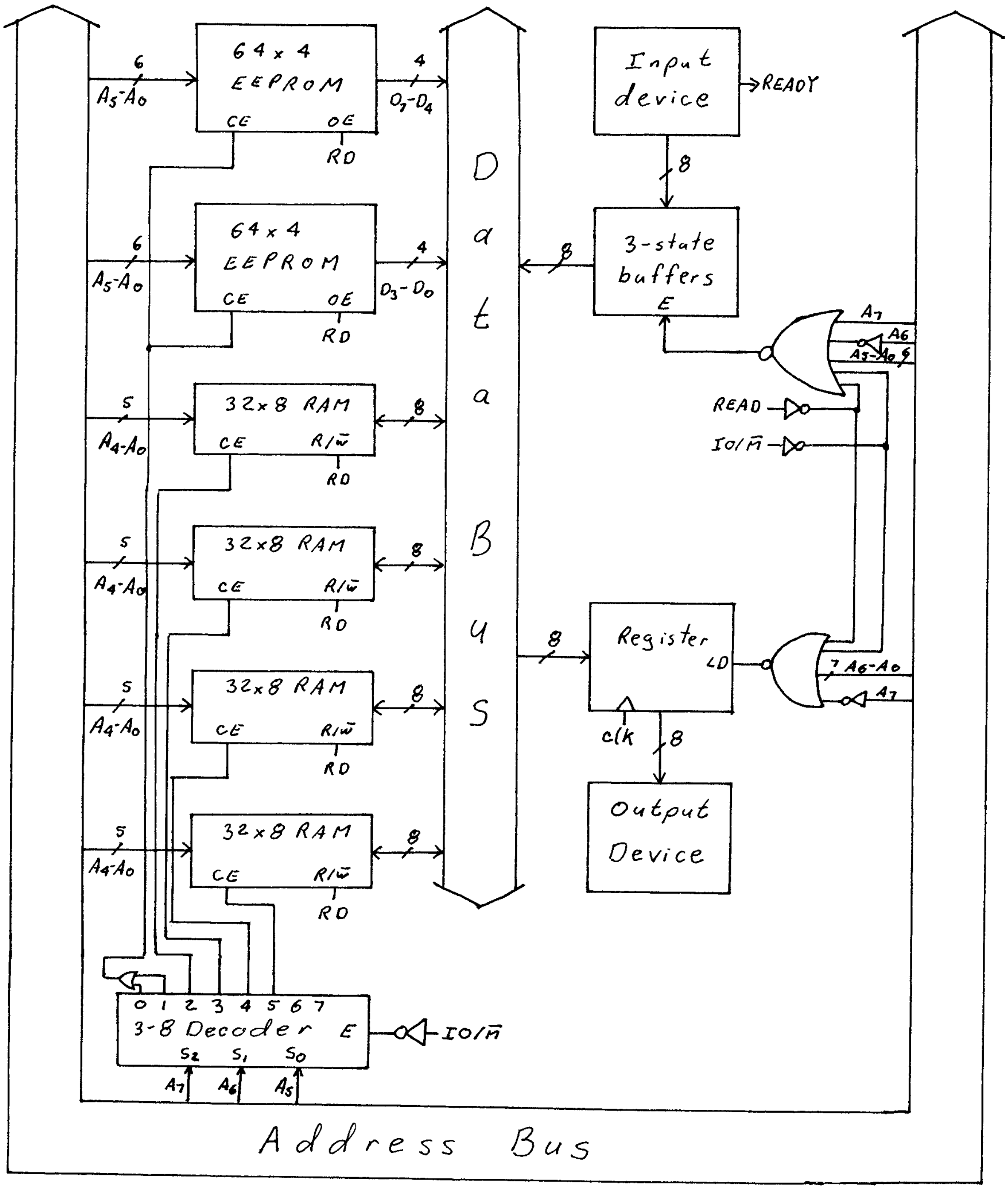
17.



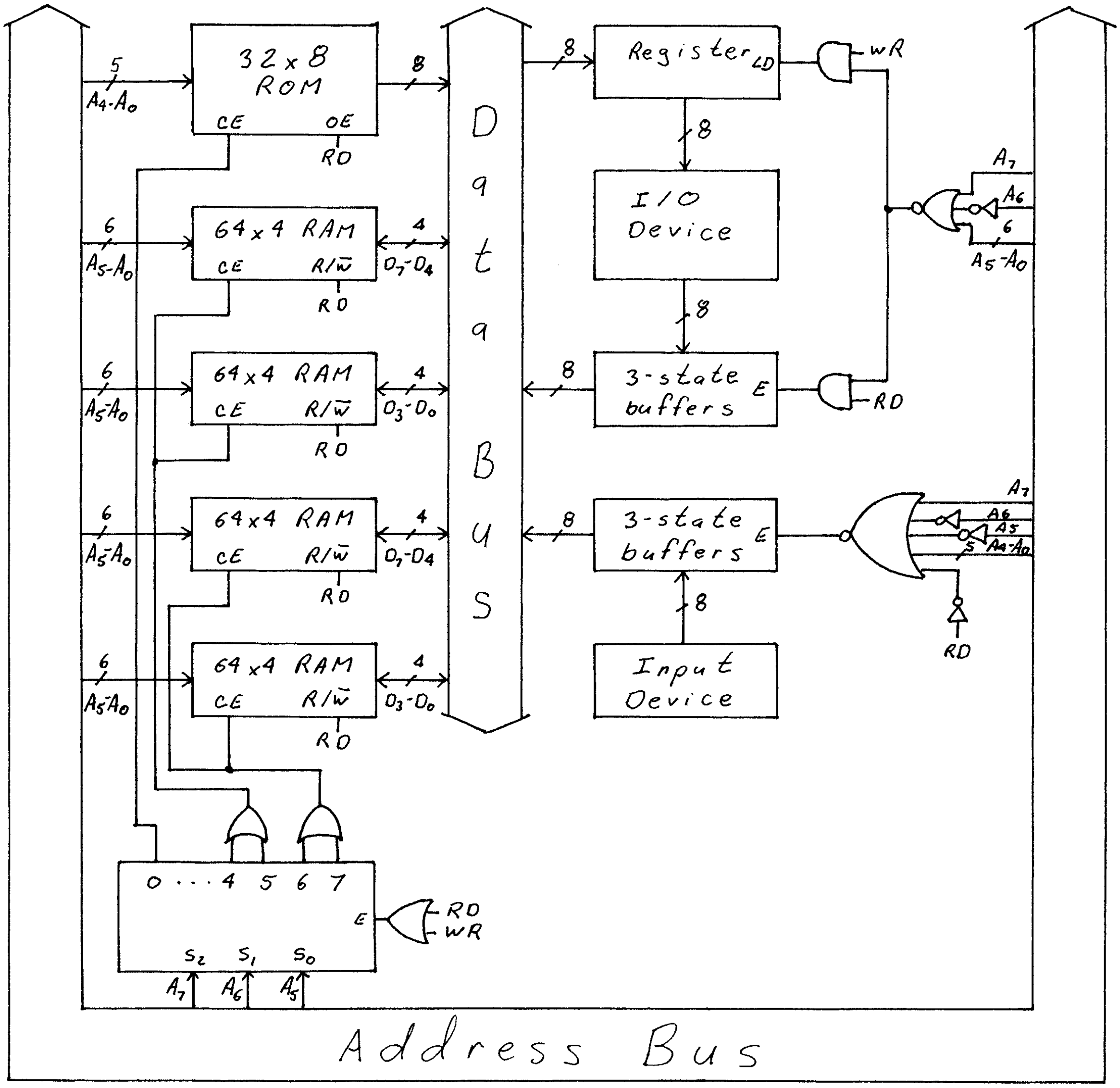
18. This is the same as the previous problem, except $IO/\overline{M}$ is not included.



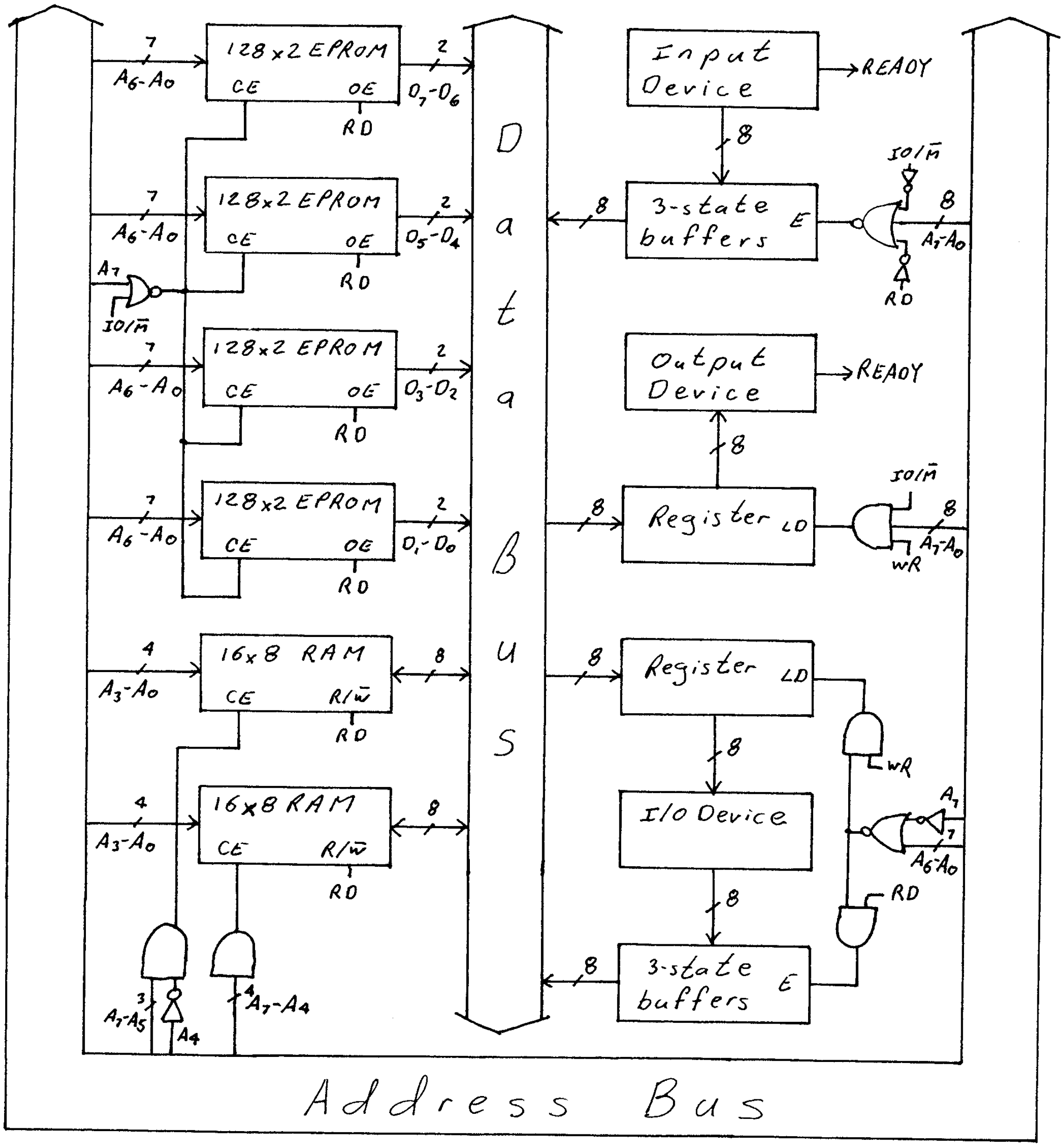
19.


64 x 4
EEPROM
6
A5-A0
CE
OE
RD
4
D7-D4
64 x 4
EEPROM
6
A5-A0
CE
OE
RD
4
D3-D0
32x8 RAM
5
A4-A0
CE
R/W
RD
8
32x8 RAM
5
A4-A0
CE
R/W
RD
8
32x8 RAM
5
A4-A0
CE
R/W
RD
8
32x8 RAM
5
A4-A0
CE
R/W
RD
8
Data Bus
Input device
READY
8
3-state buffers
E
8
A7
A6
A5-A0 6
READ
IO/M
Register
LD
clk
8
8
7 A6-A0
A7
Output Device
0 1 2 3 4 5 6 7
3-8 Decoder
E
IO/M
S2 S1 S0
A7 A6 A5
Address Bus

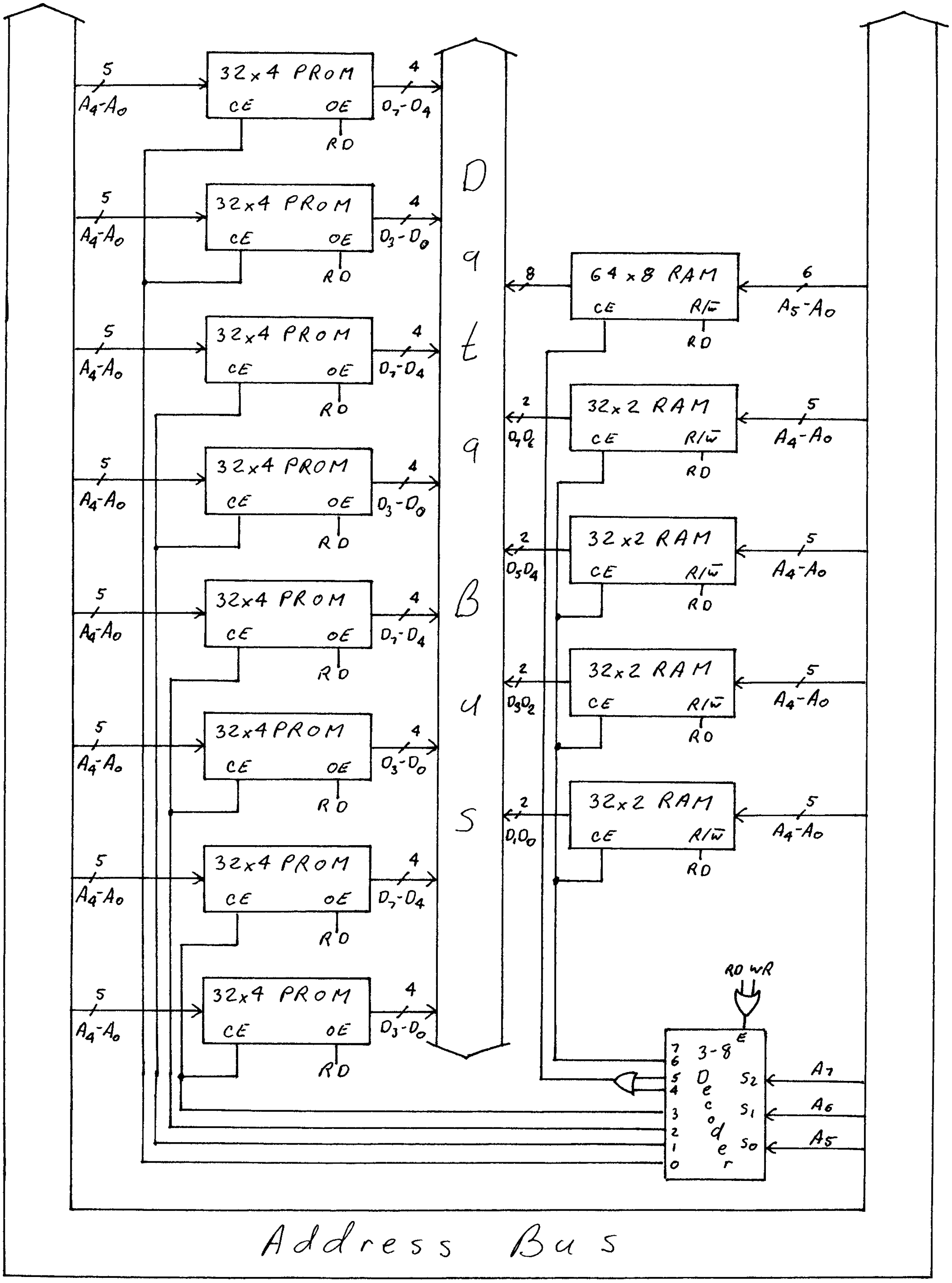
20.


32 x 8
ROM
CE
OE
RD
5
A4-A0
8
64 x 4 RAM
CE
R/W̄
6
A5-A0
4
O7-O4
O3-O0
RD
Data Bus
Register
LD
WR
8
I/O Device
3-state buffers
E
RD
A7
A6
6
A5-A0
A5
5
A4-A0
Input Device
0 · · · 4 5 6 7
E
RD
WR
S2 S1 S0
A7 A6 A5
Address Bus

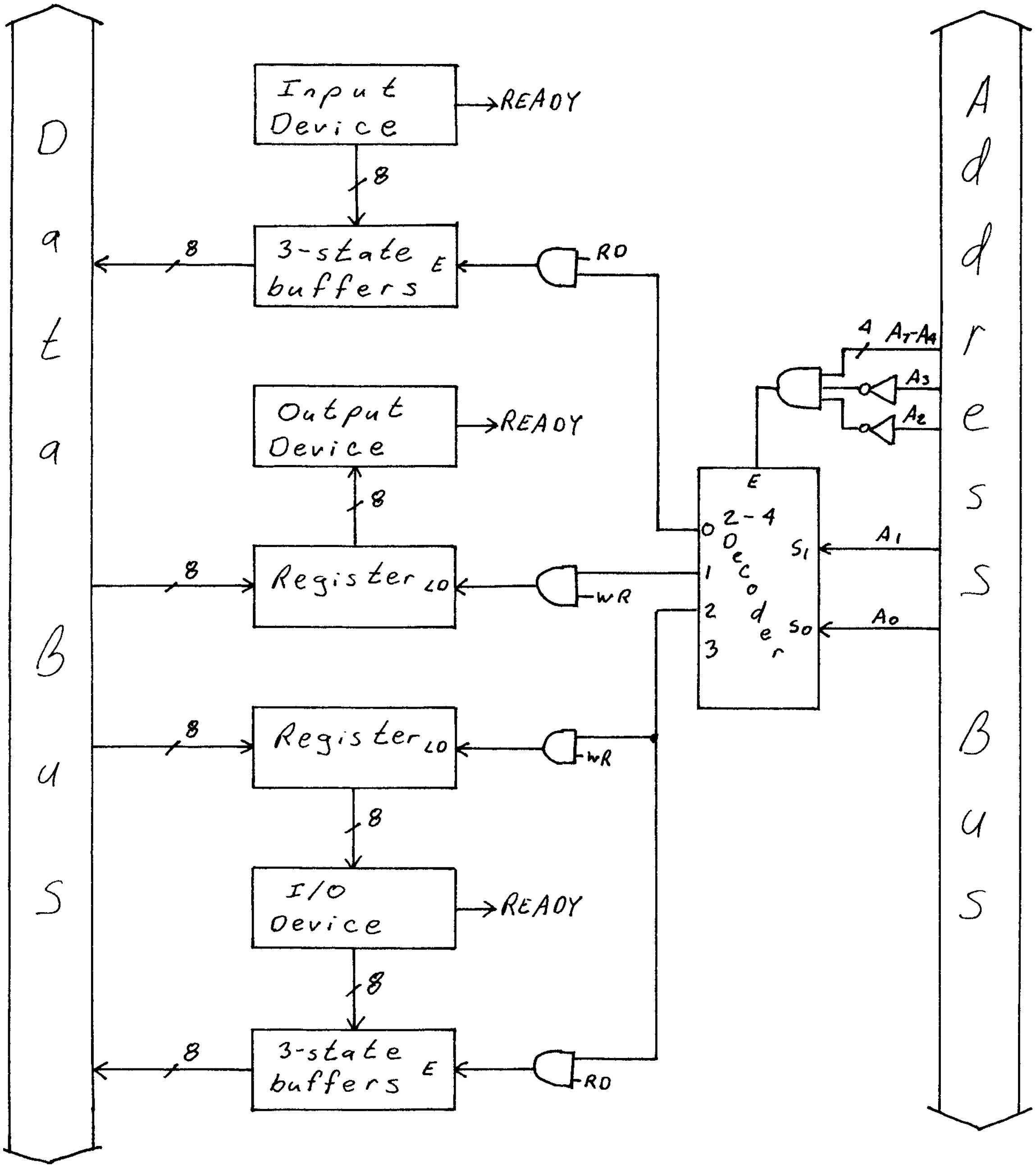
21.


7
A6-A0
128x2 EPROM
CE
OE
RD
2
O7-O6
7
A6-A0
128x2 EPROM
CE
OE
RD
2
O5-O4
A7
IO/M
7
A6-A0
128x2 EPROM
CE
OE
RD
2
O3-O2
7
A6-A0
128x2 EPROM
CE
OE
RD
2
O1-O0
4
A3-A0
16x8 RAM
CE
R/W
RD
8
4
A3-A0
16x8 RAM
CE
R/W
RD
8
3
A7-A5
A4
4
A7-A4
Data Bus
Input Device
READY
8
3-state buffers
E
8
IO/M
8
A7-A0
RD
Output Device
READY
8
8
Register
LD
IO/M
8
A7-A0
WR
8
Register
LD
WR
8
I/O Device
A7
7
A6-A0
8
RD
8
3-state buffers
E
Address Bus

22. Memory subsystem:



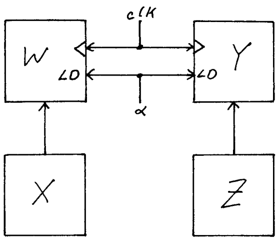
22 (continued). I/O subsystem:



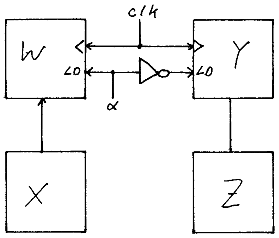
Chapter 5

1. a) α: $W \leftarrow X, Y \leftarrow Z$
   b) α: $W \leftarrow X$
   α': $Y \leftarrow Z$
   c) α': $W \leftarrow X$

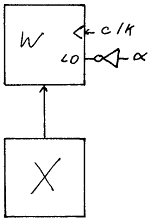
2. a)



b)

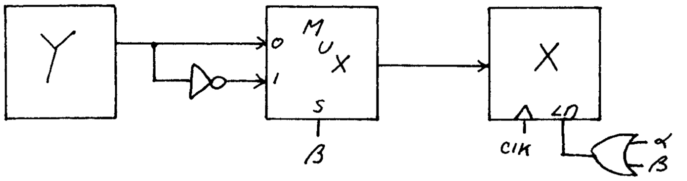


c)

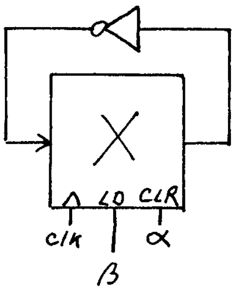


3. a) α: $X \leftarrow Y$
   β: $X \leftarrow Y'$
   b) α: $X \leftarrow 0$
   β: $X \leftarrow X'$

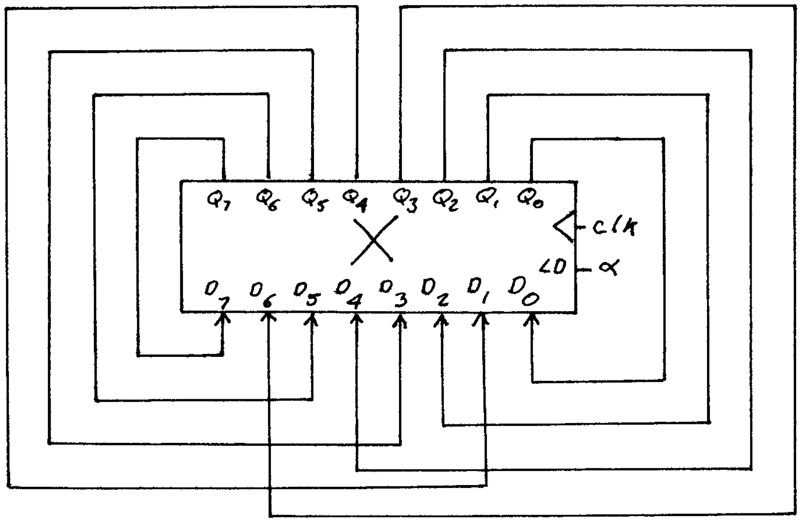
4. a)



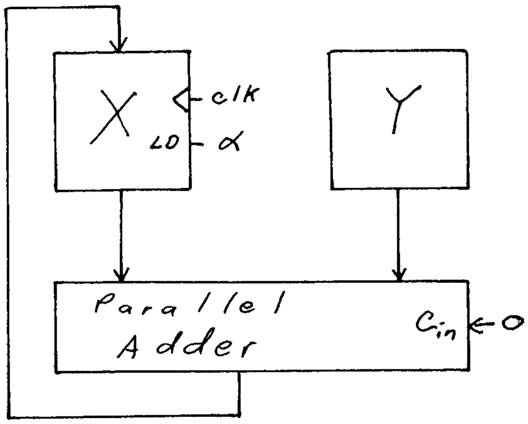
b)



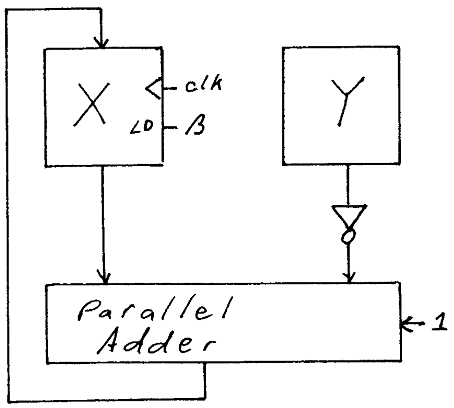
5.



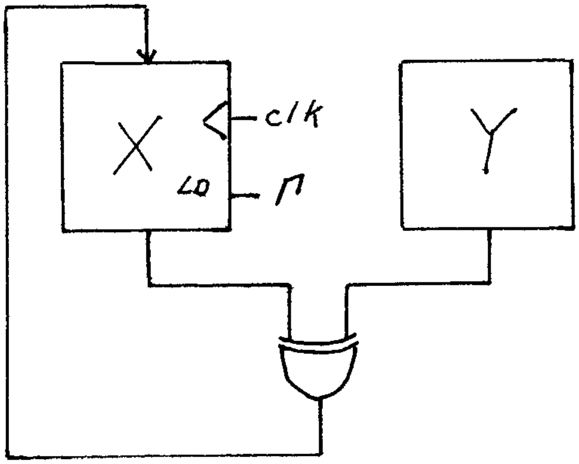
6. a)


X
clk
LD
α
Y
Parallel
Adder
C_{in}
0

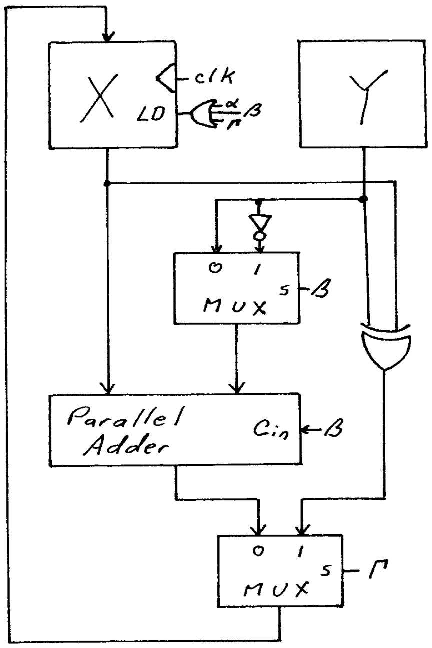
b)


X
clk
LD
β
Y
Parallel
Adder
1

c)


X
clk
LD
Γ
Y

7.

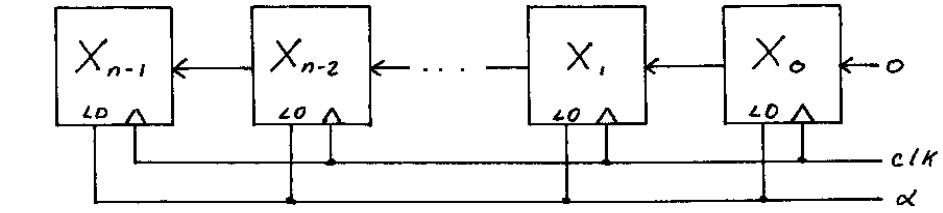

X
clk
LD
α
Γ
β
Y
0
1
MUX
s
β
Parallel
Adder
C_{in}
β
0
1
MUX
s
Γ

8. a) 0011 0010 0000 0100
   b) 0100 1100 1000 0001
   c) 0011 0010 0000 0101
   d) 0100 1100 1000 0001
   e) 1011 0010 0000 0100
   f) 1100 1100 1000 0001
   g) 1001 0000 0010 0000
   h) 0000 1001 1001 0000

9. a) 0000 0111 0010 1010
   b) 0100 0001 1100 1010
   c) 0000 0111 0010 1011
   d) 1100 0001 1100 1010
   e) 1000 0111 0010 1010
   f) 1100 0001 1100 1010
   g) 0011 1001 0101 0000
   h) 0000 1000 0011 1001

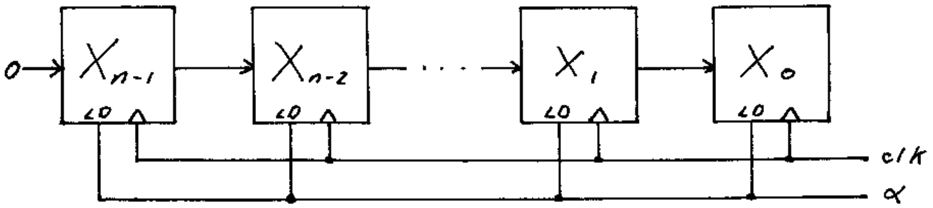
10. a) 1011 0010 1111 0000
    b) 0010 1100 1011 1100
    c) 1011 0010 1111 0000
    d) 0010 1100 1011 1100
    e) 0011 0010 1111 0000
    f) 0010 1100 1011 1100
    g) 1001 0111 1000 0000
    h) 0000 0101 1001 0111

11. a)



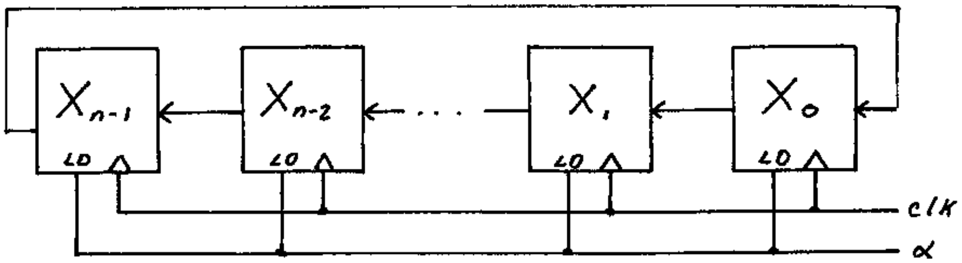
X_{n-1}
X_{n-2}
X_1
X_0
LD
0
clk
α

b)



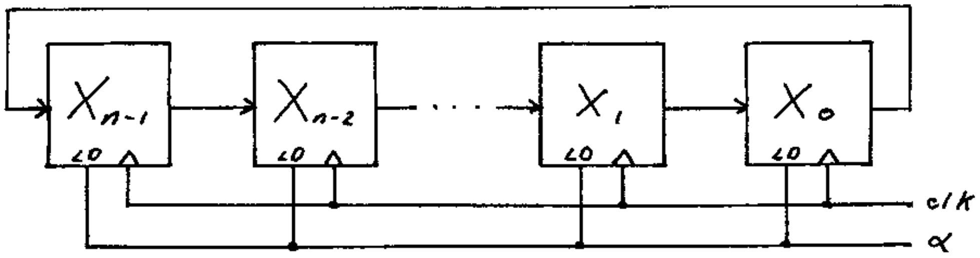
0
X_{n-1}
X_{n-2}
X_1
X_0
LD
clk
α

c)



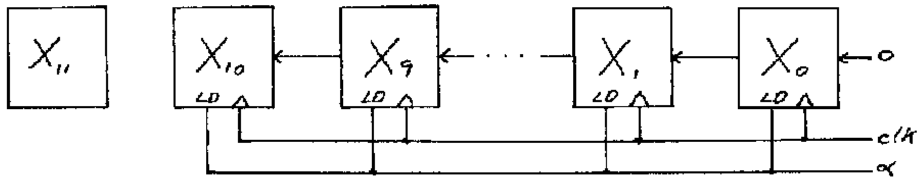
X_{n-1}
X_{n-2}
X_1
X_0
LD
clk
α

d)

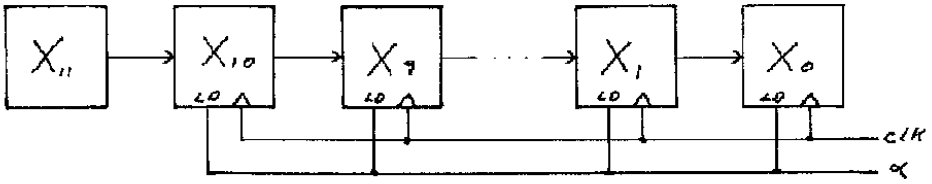


X_{n-1}
X_{n-2}
X_1
X_0
LD
clk
α

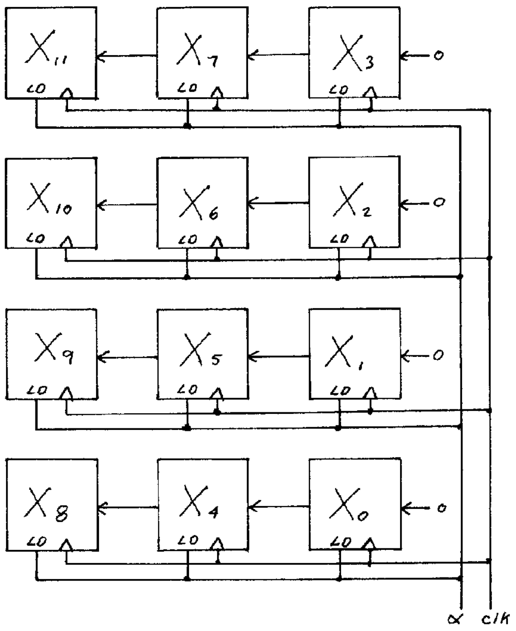
12. a)



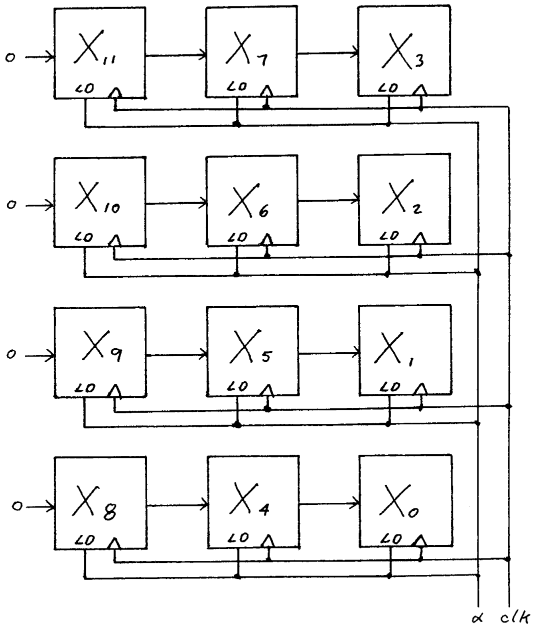
b)



c)



d)

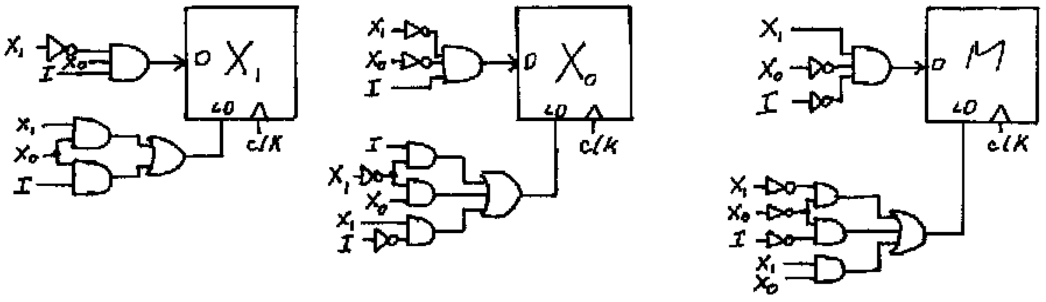


13. a) $X \leftarrow 0,X[(n\text{-}2)\text{-}1]$
    b) $X \leftarrow X[(n\text{ -}2)\text{-}0,(\ n\text{ -}1)]$
    c) $X \leftarrow X[0,(\ n\text{ -}1)\text{-}1]$
    d) $X[(n\text{ -}2)\text{-}0] \leftarrow X[(n\text{ -}3)\text{-}0],0$
    e) $X[(n\text{ -}2)\text{-}0] \leftarrow X[(n\text{ -}1)\text{-}1]$
    f) $X \leftarrow X[(n\text{ -}5)\text{-}0],0000$
    g) $X \leftarrow 0000,X[(n\text{ -}1)\text{-}4]$

14. Define X_1X_0 = 00 (S_0), 01 (S_1), 10 (S_2), 11 (S_3). I is the input bit.

$X_1'X_0'I'$: $M \leftarrow 0$
$X_1'X_0'I$: $X_0 \leftarrow 1, M \leftarrow 0$
$X_1'X_0I'$: $X_0 \leftarrow 0$
$X_1'X_0I$: $X_1 \leftarrow 1, X_0 \leftarrow 0$
$X_1X_0'I'$: $X_0 \leftarrow 1, M \leftarrow 1$
X_1X_0I': $X_1 \leftarrow 0, X_0 \leftarrow 0, M \leftarrow 0$
X_1X_0I: $X_1 \leftarrow 0, M \leftarrow 0$

15.



16. Define X_1X_0 = 00 (S_0), 01 (S_1), 10 (S_2), 11 (S_3). I is the input bit.

Brute force solution (one RTL statement per input value per state)

$X_2'X_1'X_0'I'$: $M \leftarrow 0$
$X_2'X_1'X_0'I$: $X_0 \leftarrow 1, M \leftarrow 0$
$X_2'X_1'X_0I'$: $X_1 \leftarrow 1, X_0 \leftarrow 0$
$X_2'X_1'X_0I$: $X_1 \leftarrow 1, X_0 \leftarrow 1$
$X_2'X_1X_0'I'$: $X_2 \leftarrow 1, X_1 \leftarrow 0$
$X_2'X_1X_0'I$: $X_2 \leftarrow 1, X_1 \leftarrow 0, X_0 \leftarrow 1$
$X_2'X_1X_0I'$: $X_2 \leftarrow 1, X_0 \leftarrow 0, M \leftarrow 1$
$X_2'X_1X_0I$: $X_2 \leftarrow 1$
$X_2X_1'X_0'I'$: $X_2 \leftarrow 0$
$X_2X_1'X_0'I$: $X_2 \leftarrow 0, X_0 \leftarrow 1$
$X_2X_1'X_0I'$: $X_2 \leftarrow 0, X_1 \leftarrow 1, X_0 \leftarrow 0$
$X_2X_1'X_0I$: $X_2 \leftarrow 0, X_1 \leftarrow 1$
$X_2X_1X_0'I'$: $X_1 \leftarrow 0, M \leftarrow 0$
$X_2X_1X_0'I$: $X_1 \leftarrow 0, X_0 \leftarrow 1, M \leftarrow 0$
$X_2X_1X_0I'$: $X_0 \leftarrow 0$

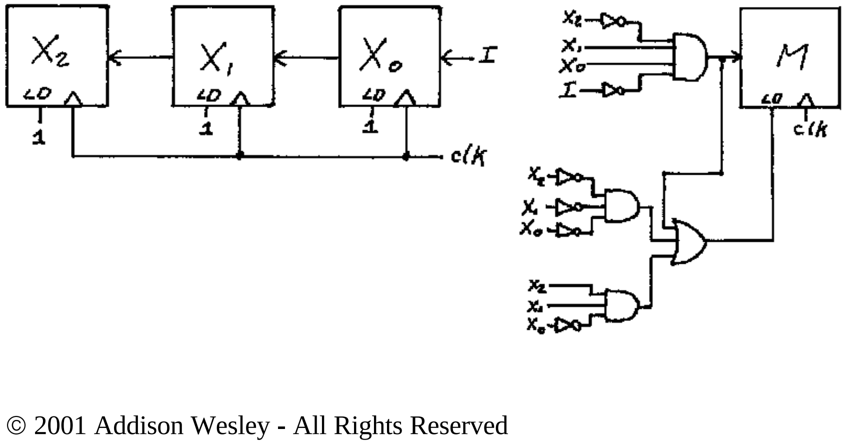
Simpler solution (one RTL statement per state)

$X_2'X_1'X_0'$: $X_0 \leftarrow I, M \leftarrow 0$
$X_2'X_1'X_0$: $X_1 \leftarrow 1, X_0 \leftarrow I$
$X_2'X_1X_0'$: $X_2 \leftarrow 1, X_1 \leftarrow 0, X_0 \leftarrow I$
$X_2'X_1X_0$: $X_2 \leftarrow 1, X_0 \leftarrow I, M \leftarrow I'$
$X_2X_1'X_0'$: $X_2 \leftarrow 0, X_0 \leftarrow I$
$X_2X_1'X_0$: $X_2 \leftarrow 0, X_1 \leftarrow 1, X_0 \leftarrow I$
$X_2X_1X_0'$: $X_1 \leftarrow 0, X_0 \leftarrow I, M \leftarrow 0$
$X_2X_1X_0$: $X_0 \leftarrow I, M \leftarrow I'$

Simplest solution (combining states)

1: $X_2 \leftarrow X_1, X_1 \leftarrow X_0, X_0 \leftarrow I$
$X_2'X_1'X_0' + X_2'X_1X_0\, I' + X_2X_1X_0'$: $M \leftarrow X_2'X_1X_0\, I'$

17.



18.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity string_checker is
   port(
      I,clk: in std_logic;
         M: out std_logic
   );
end string_checker;

architecture a_string_checker of string_checker is
   type states is (S0, S1, S2, S3);
   signal present_state, next_state: states;

begin
   state_check_string: process(present_state,I)
      begin
         case present_state is
            when S0 => M<='0';
               if (I='0') then next_state <= S0;
                  else next_state <= S1;
               end if;
            when S1 => M<='0';
               if (I='0') then next_state <= S0;
                  else next_state <= S2;
               end if;
            when S2 => M<='0';
               if (I='0') then next_state <= S3;
                  else next_state <= S2;
               end if;
            when S3 => M<='1';
               if (I='0') then next_state <= S0;
                  else next_state <= S1;
               end if;
         end case;
      end process state_check_string;

   state_transition: process(clk)
      begin
         if rising_edge(clk) then present_state <= next_state;
         end if;
      end process state_transition;

end a_string_checker;
```

19.
```
library IEEE;
use IEEE.std_logic_1164.all;

entity string_checker is
   port(
      I,clk: in std_logic;
      X1,X0: buffer std_logic;
         M: out std_logic
   );
end string_checker;

architecture a_string_checker of string_checker is
begin
   cct_string_checker: process(X1,X0,I,clk)
      begin
         if rising_edge(clk) then
            X1 <= (X1 and (not X0)) or
                 ((not X1) and X0 and I);
            X0 <= ((not X1) and (not X0) and I) or
                 (X1 and (not X0) and (not I)) or
                 (X1 and X0 and I);
         end if;
          M <= X1 AND X0;
      end process cct_string_checker;
end a_string_checker;
```

20.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity string_checker is
   port(
      I,clk: in std_logic;
         M: out std_logic
   );
end string_checker;

architecture a_string_checker of string_checker is
   type states is (S0, S1, S2, S3, S4, S5, S6, S7);
   signal present_state, next_state: states;

begin
   state_check_string: process(present_state,I)
      begin
         case present_state is
            when S0 => M<='0';
               if (I='0') then next_state <= S0;
                  else next_state <= S1;
               end if;
            when S1 => M<='0';
               if (I='0') then next_state <= S2;
                  else next_state <= S3;
               end if;
            when S2 => M<='0';
               if (I='0') then next_state <= S4;
                  else next_state <= S5;
               end if;
            when S3 => M<='0';
               if (I='0') then next_state <= S6;
                  else next_state <= S7;
               end if;
            when S4 => M<='0';
               if (I='0') then next_state <= S0;
                  else next_state <= S1;
               end if;
            when S5 => M<='0';
               if (I='0') then next_state <= S2;
                  else next_state <= S3;
               end if;
            when S6 => M<='1';
               if (I='0') then next_state <= S4;
                  else next_state <= S5;
               end if;
            when S7 => M<='0';
               if (I='0') then next_state <= S6;
                  else next_state <= S7;
               end if;
         end case;
      end process state_check_string;

   state_transition: process(clk)
      begin
         if rising_edge(clk) then present_state <= next_state;
         end if;
      end process state_transition;

end a_string_checker;
```

21.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity string_checker is
   port(
         I,clk: in std_logic;
      X2,X1,X0: buffer std_logic;
            M: out std_logic
   );
end string_checker;

architecture a_string_checker of string_checker is
begin
   cct_string_checker: process(X1,X0,I,clk)
      begin
         if rising_edge(clk) then
            X2 <= X1;
            X1 <= X0;
            X0 <= I;
         end if;
          M <= X2 and X1 and (not X0);
      end process cct_string_checker;
end a_string_checker;
```

22.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity toll_booth_controller is
   port(
      I1,I0,C,clk: in std_logic;
            R,G,A: out std_logic
   );
end toll_booth_controller;

architecture a_toll_booth_controller of toll_booth_controller is
   type states is (SN, S0, S5, S10, S15, S20, S25, S30, SP, SC);
   signal present_state, next_state: states;

begin
   state_toll_booth_controller: process(present_state,I1,I0)
      begin
         case present_state is
            when SN => R<='1'; G<='0'; A<='0';
               if (C='1') then next_state <= S0;
                  else next_state <= SN;
               end if;
            when S0 => R<='1'; G<='0'; A<='0';
               if (C='0') then next_state <= SC;
                  elsif (I1 = '0' AND I0 = '1') then next_state <= S5;
                  elsif (I1 = '1' AND I0 = '0') then next_state <= S10;
                  elsif (I1 = '1' AND I0 = '1') then next_state <= S25;
                  else next_state <= S0;
               end if;
            when S5 => R<='1'; G<='0'; A<='0';
               if (C='0') then next_state <= SC;
                  elsif (I1 = '0' AND I0 = '1') then next_state <= S10;
                  elsif (I1 = '1' AND I0 = '0') then next_state <= S15;
                  elsif (I1 = '1' AND I0 = '1') then next_state <= S30;
                  else next_state <= S5;
               end if;
            when S10 => R<='1'; G<='0'; A<='0';
               if (C='0') then next_state <= SC;
                  elsif (I1 = '0' AND I0 = '1') then next_state <= S15;
                  elsif (I1 = '1' AND I0 = '0') then next_state <= S20;
                  elsif (I1 = '1' AND I0 = '1') then next_state <= SP;
                  else next_state <= S10;
               end if;
            when S15 => R<='1'; G<='0'; A<='0';
               if (C='0') then next_state <= SC;
                  elsif (I1 = '0' AND I0 = '1') then next_state <= S20;
                  elsif (I1 = '1' AND I0 = '0') then next_state <= S25;
                  elsif (I1 = '1' AND I0 = '1') then next_state <= SP;
                  else next_state <= S15;
               end if;
            when S20 => R<='1'; G<='0'; A<='0';
               if (C='0') then next_state <= SC;
                  elsif (I1 = '0' AND I0 = '1') then next_state <= S25;
                  elsif (I1 = '1' AND I0 = '0') then next_state <= S30;
                  elsif (I1 = '1' AND I0 = '1') then next_state <= SP;
                  else next_state <= S20;
               end if;
            when S25 => R<='1'; G<='0'; A<='0';
               if (C='0') then next_state <= SC;
                  elsif (I1 = '0' AND I0 = '1') then next_state <= S30;
                  elsif (I1 = '1') then next_state <= SP;
                  else next_state <= S25;
               end if;
            when S30 => R<='1'; G<='0'; A<='0';
               if (C='0') then next_state <= SC;
                  elsif (I1 = '1' OR I0 = '1') then next_state <= SP;
                  else next_state <= S30;
               end if;
            when SP => R<='0'; G<='1'; A<='0';
               if (C='0') then next_state <= SN;
                  else next_state <= SP;
               end if;
            when SC => R<='1'; G<='0'; A<='1';
               if (C='1') then next_state <= S0;
                  else next_state <= SC;
               end if;
         end case;
      end process state_toll_booth_controller;

   state_transition: process(clk)
      begin
         if rising_edge(clk) then present_state <= next_state;
         end if;
      end process state_transition;

end a_toll_booth_controller;
```

23.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity toll_booth_controller is
   port(
       I1,I0,C,clk: in std_logic;
       X3,X2,X1,X0: buffer std_logic;
            R,G,A: out std_logic
   );
end toll_booth_controller;

architecture a_toll_booth_controller of toll_booth_controller is
begin
   cct_toll_booth_controller: process(X3,X2,X1,X0,I1,I0,C,clk)
       begin
          if rising_edge(clk) then
             X3 <= not C;
             X2 <= ((not X3) and C and I1 and I0) or
                   ((not X3) and (X2 or X1) and C and I1 and (not I0)) or
                   ((not X3) and (X2 or (X1 and X0)) and C and (not I1) and I0)
                     or (X2 and C and (not I1) and (not I0));
             X1 <= ((not X3) and (X2 or X1 or X0) and C and I1 and I0)or
                   ((not X3) and (X2 or (not X1)) and C and I1 and (not I0)) or
                   ((not X3) and (((not X1) and X0) or (X1 and (not X0)) or
                    (X2 and X1 and X0)) and C and (not I1) and I0) or
                   (X1 and X0 and C and (not I1) and (not I0));
             X0 <= ((not X3) and (X2 or X1 or (not X0)) and C and I1 and I0) or
                   ((not X3) and (X0 or (X2 and X1)) and C and I1 and (not I0))
                    or ((not X3) and ((not X0) or (X2 and X1)) and C and
                    (not I1) and I0) or ((not X3) and X0 and C and (not I1) and
                    (not I0)) or (X3 and X0 and (not C)) or ((not X3) and
                    ((not X2) or (not X1) or (not X0)) and (not C));
          end if;
           R <= X3 or (not X2) or (not X1) or (not X0);
           G <= (not X3) and X2 and X1 and X0;
           A <= X3 and (not X2) and (not X1) and X0;
       end process cct_toll_booth_controller;
end a_toll_booth_controller;
```

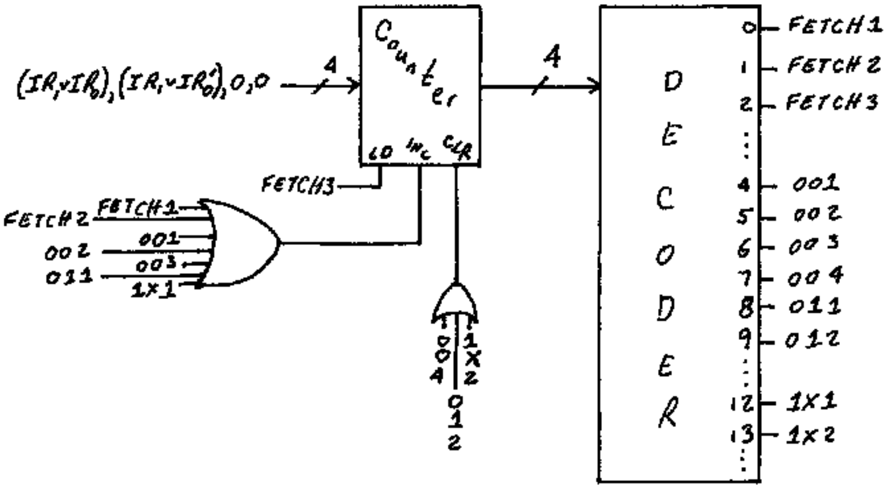
Chapter 6

1. JMP11: $PC \leftarrow AR$
   JMP12: $PC \leftarrow PC + 1$
   INC21: $AC \leftarrow AC + 1$
   INC22: $AC \leftarrow AC + 1$
   ADD11: $DR \leftarrow M, AC \leftarrow AC + 1$
   ADD12: $AC \leftarrow AC + DR$
   SKIP1: $PC \leftarrow PC + 1$

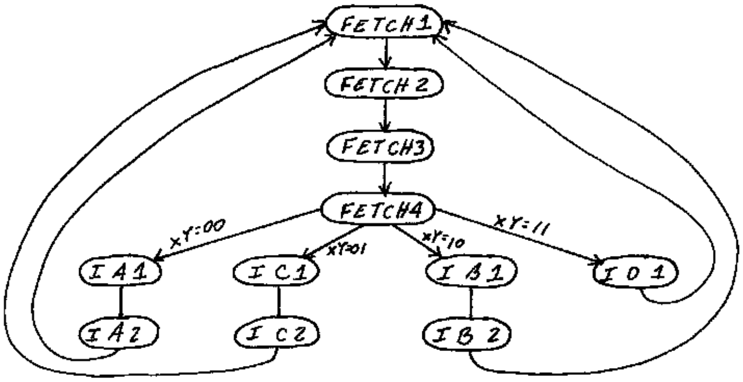
2.

Instruction	Instruction Code	Operation
ADDADD	00AAAAAA	$AC \leftarrow AC$ + M[AAAAAA] + M[AAAAAA + 1]
ANDSKIP	01AAAAAA	$AC \leftarrow AC$ ^ M[AAAAAA], $PC \leftarrow PC + 1$
INCAND	1XAAAAAA	$AC \leftarrow (AC + 1)$ ^ M[AAAAAA]

3. This is one of many possible solutions.

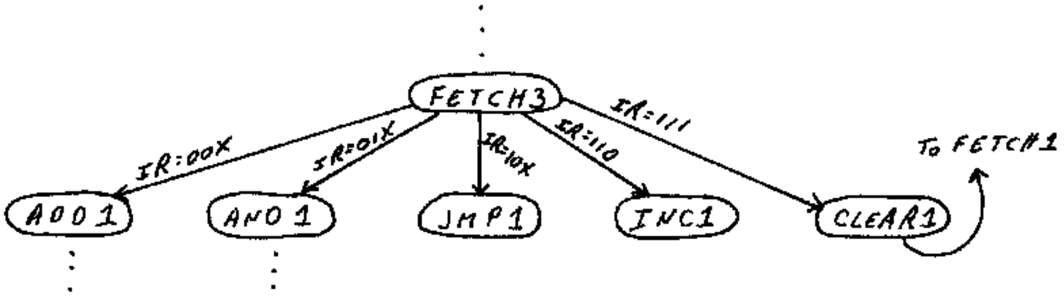


4.



5. Change the input to the counter to X, X', Y, $X \vee Y$.
Change *INC* input IA3 to IA2.
Change *CLR* input IA2 to IA3.

6. (*IR* must have 3 bits instead of 2.)



FETCH3: $IR \leftarrow DR[7..5]$, $AR \leftarrow DR[5..0]$
CLEAR1: $AC \leftarrow 0$

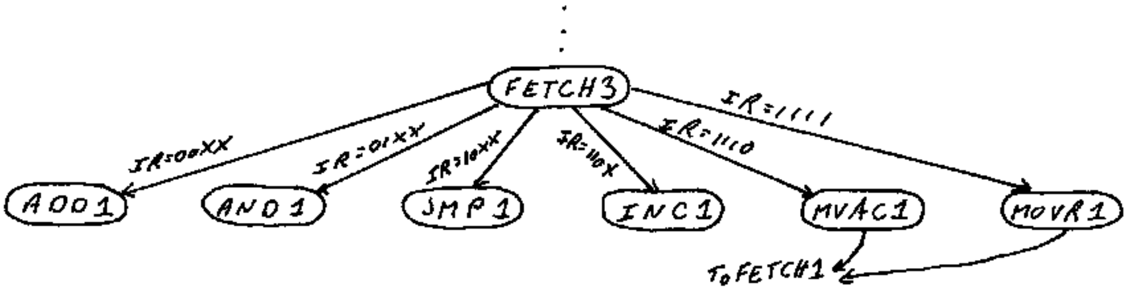
7. i) *IR* must have 3 bits instead of 2. It receives bus bits 7..5 as its inputs. During FETCH3, bit 5 of *DR* is sent to both *IR* and *AR*.
ii) *AC* needs a *CLR* input (*ACCLR* is a new control signal which connects to the new *CLR* input.)

8. Arbitrarily assign *CLEAR1* to decoder output 15.
i) New input to counter: 1,*IR*[2..1],($IR_2 \wedge IR_1 \wedge IR_0$).
ii) Add *CLEAR1* to the inputs of the OR gate driving counter *CLR*.
iii) New control signal *ACCLR* = *CLEAR1*.

9. Test program: 0: CLEAR

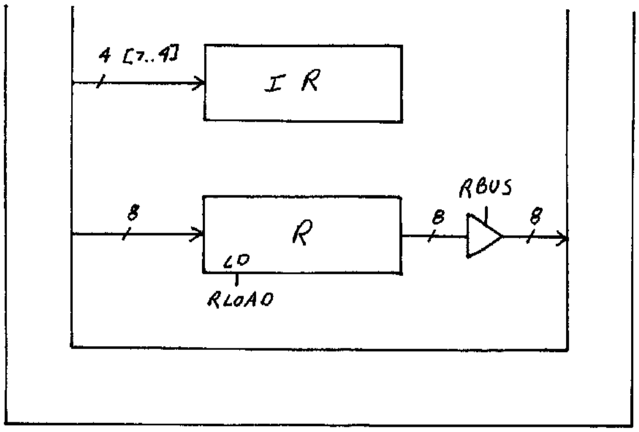
Instruction	State	Operations performed	Next state
CLEAR	FETCH1	$AR \leftarrow 0$	FETCH2
	FETCH2	$DR \leftarrow$ E0H, $PC \leftarrow 1$	FETCH3
	FETCH3	$IR \leftarrow 111$, $AR \leftarrow$ 20H	CLEAR1
	CLEAR1	$AC \leftarrow 0$	FETCH1

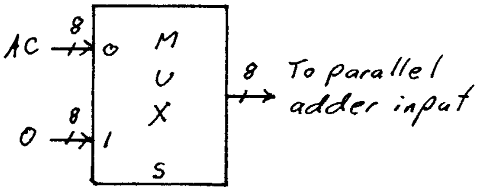
10. (*IR* must have 4 bits instead of 2.)



FETCH3: $IR \leftarrow DR[7..4]$, $AR \leftarrow DR[5..0]$
MVAC1: $R \leftarrow AC$
MOVR1: $AC \leftarrow R$

11. i.) *IR* must have 4 bits instead of 2. It receives bus bits 7..4 as its inputs. During FETCH3, bit *DR*[5..4] is sent to both *IR* and *AR*. This is shown below.
ii) Register *R* is added to the CPU. It receives data from the bus and sends data to the bus through tri-state buffers. It requires only a *LD* signal. This is shown below.
iii) The ALU is modified as shown below.





12. Arbitrarily assign *MVAC1* and *MOVR1* to decoder outputs 6 and 7, respectively.

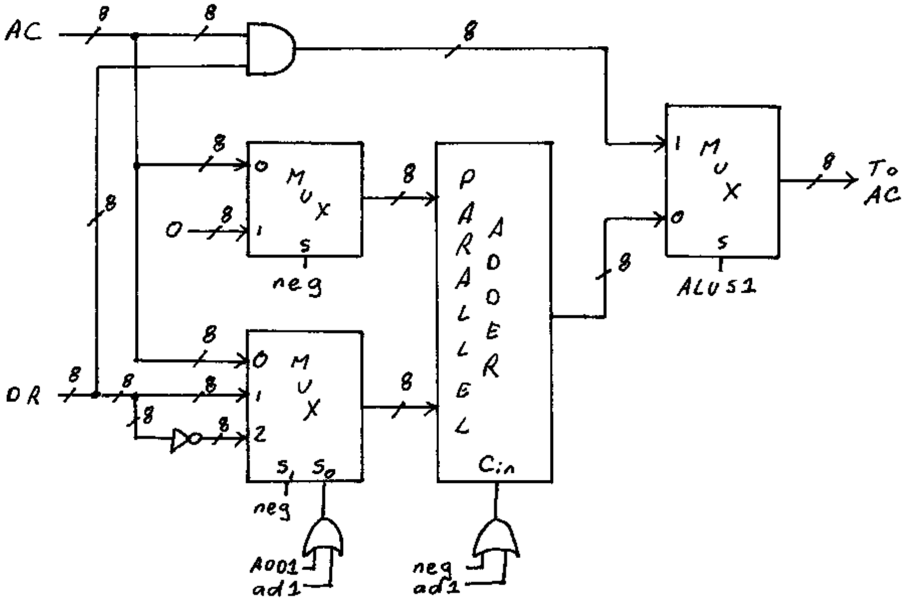
i) New input to counter: $(IR_3 \wedge IR_2 \wedge IR_1)'$, *IR*[3..2],$(IR_3 \wedge IR_2 \wedge IR_1 \wedge IR_0.)$.
ii) Add *MVAC1* and MOVR1 to the inputs of the OR gate driving counter *CLR*.
iii) New control signals *RLOAD* = *MVAC1* and *RBUS* = *MOVR1*.
iv) Add *MOVR1* to the inputs of the OR gate generating *ACLOAD*.

13. Test program: 0: MVAC
1: MOVR

Instruction	State	Operations performed	Next state
MVAC	FETCH1	*AR* ← 0	FETCH2
	FETCH2	*DR* ← E0H, *PC* ← 1	FETCH3
	FETCH3	*IR* ← 1110, *AR* ← 20H	MVAC1
	MVAC1	*R* ← 1	FETCH1
MOVR	FETCH1	*AR* ← 1	FETCH2
	FETCH2	*DR* ← F0H, *PC* ← 2	FETCH3
	FETCH3	*IR* ← 1111, *AR* ← 30H	MOVR1
	MOVR1	*AC*← 1	FETCH1

14. All operations except AND are performed by the parallel adder.

Micro-operation	**Adder inputs**
ADD1	*AC* + *BUS* + 0
shl	*AC* + *AC* + 0
neg	0 + *BUS'* + 0
ad1	*AC* + *BUS* + 1



15. *PCLOAD* = *JUMP3* ∨ *JMPZY3* ∨ *JPNZY3*

PCINC = *FETCH2* ∨ *LDAC1* ∨ *LDAC2* ∨ *STAC1* ∨ *STAC2* ∨ *JMPZN1* ∨ *JMPZN2* ∨ *JPNZN1* ∨ *JPNZN2*

DRLOAD = *FETCH2* ∨ *LDAC1* ∨ *LDAC2* ∨ *LDAC4* ∨ *STAC1* ∨ *STAC2* ∨ *STAC4* ∨ *JUMP1* ∨ *JUMP2* ∨ *JMPZY1* ∨ *JMPZY2* ∨ *JPNZY1* ∨ *JPNZY2*

TRLOAD = *LDAC2* ∨ *STAC2* ∨ *JUMP2* ∨ *JMPZY2* ∨ *JPNZY2*

IRLOAD = *FETCH3*

16. *RLOAD* = *MVAC1*

ACLOAD = *LDAC5* ∨ *MOVR1* ∨ *ADD1* ∨ *SUB1* ∨ *INAC1* ∨ *CLAC1* ∨ *AND1* ∨ *OR1* ∨ *XOR1* ∨ *NOT1*

ZLOAD = *ADD1* ∨ *SUB1* ∨ *INAC1* ∨ *CLAC1* ∨ *AND1* ∨ *OR1* ∨ *XOR1* ∨ *NOT1*

17.

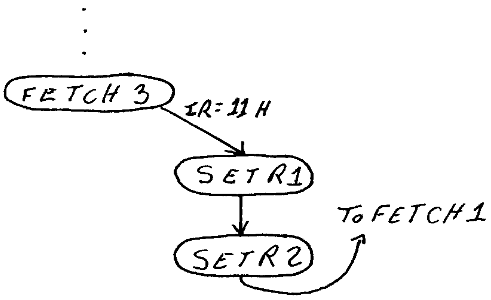
State	**ALUS[1..7]**
LDAC5	0 0 1 0 X X 0
MOVR1	0 0 1 0 X X 0
ADD1	1 0 1 0 X X 0
SUB1	1 1 0 0 X X 0
INAC1	1 0 0 1 X X 0
CLAC1	0 0 0 0 X X 0
AND1	X X X X 0 0 1
OR1	X X X X 0 1 1
XOR1	X X X X 1 0 1
NOT1	X X X X 1 1 1

ALUS1 = *ADD1* ∨ *SUB1* ∨ *INAC1*
ALUS2 = *SUB1*
ALUS3 = *LDAC5* ∨ *MOVR1* ∨ *ADD1*
ALUS4 = *SUB1* ∨ *INAC1*
ALUS5 = *XOR1* ∨ *NOT1*
ALUS6 = *OR1* ∨ *NOT1*
ALUS7 = *AND1* ∨ *OR1* ∨ *XOR1* ∨ *NOT1*

18. The student can execute the following program using the Relatively Simple CPU simulator to verify that each instruction performs properly.

```
0:  LDAC 0000     (AC ← 1)
    NOP
    MVAC          (R ← 1)
    ADD           (AC ← 2, Z ← 0)
    INAC          (AC ← 3, Z ← 0)
    XOR           (AC ← 2, Z ← 0)
    AND           (AC ← 0, Z ← 1)
9:  JMPZ 000D     (jump is taken)
    NOP           (skipped by JMPZ 000D)
D:  JPNZ 0009     (jump is not taken)
    NOT           (AC ← FF, Z ← 0)
    JMPZ 0009     (jump is not taken)
    JPNZ 0018     (jump is taken)
    NOP           (skipped by JMPZ 0018)
18: CLAC          (AC ← 0, Z ← 1)
    OR            (AC ← 1, Z ← 0)
    SUB           (AC ← 0, Z ← 1)
    MOVR          (AC ← 1)
    STAC 0030     (M[30] ← 1)
    AND           (AC ← 1, Z ← 0)
    JUMP 0000     (start again)
```

19. *SETR1*: $R \leftarrow 0$
SETR2: $R \leftarrow R - 1$



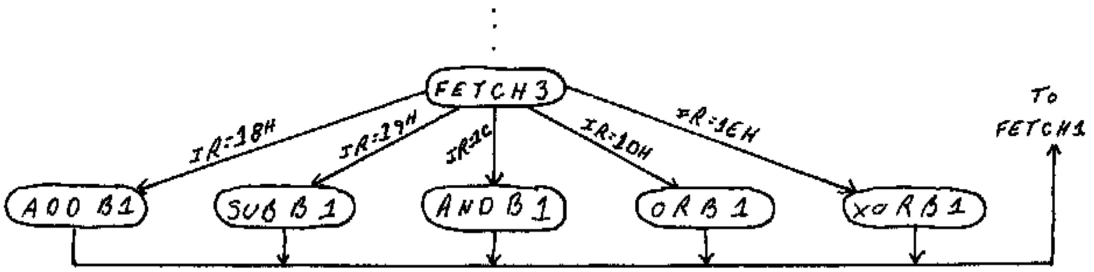
20. *R* needs two additional inputs: *CLR*, driven by new control signal *RCLR*, and *DCR*, driven by new control signal *RDCR*.

21. i.) Add hardware to generate $ISETR = I_7' \wedge I_6' \wedge I_5' \wedge I_4 \wedge I_3' \wedge I_2' \wedge I_1' \wedge I_0'$, $SETR1 = ISETR \wedge T3$, and $SETR2 = ISETR \wedge T4$.
ii) Add *SETR1* to the OR gate driving *INC* of the time counter and *SETR2* to the OR gate driving *CLR* of the time counter.
iii) New control signals *RCLR* = *SETR1* and *RDCR* = *SETR2*.

22. Test program: 0: SETR

Instruction	**State**	**Operations performed**	**Next state**
SETR	FETCH1	$AR \leftarrow 0$	FETCH2
	FETCH2	$DR \leftarrow$ 11H, $PC \leftarrow 1$	FETCH3
	FETCH3	$IR \leftarrow 11$, $AR \leftarrow 1$	SETR1
	SETR1	$R \leftarrow 0$	SETR2
	SETR2	$R \leftarrow$FF	FETCH1

23.

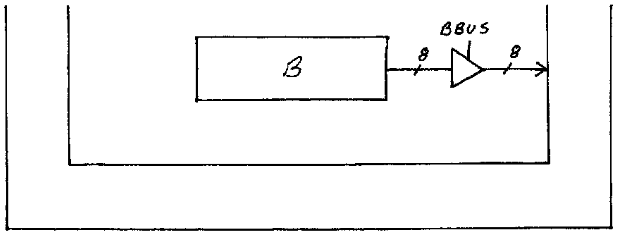


ADDB1: *AC* ← *AC* + *B*
SUBB1: *AC* ← *AC* - *B*
ANDB1: *AC* ← *AC* ^ *B*
ORB1: *AC* ← *AC* ∨ *B*
XORB1: *AC* ← *AC* ⊕ *B*

24.

i.) No ALU changes are needed!

ii) Register *B* is added to the CPU. It sends data to the bus through tri-state buffers but does not receive data from the bus (since it is never loaded). This is shown below.



25.

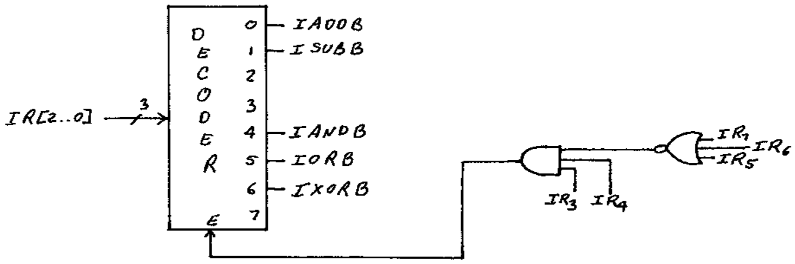
i.) Add the hardware shown below to generate *IADDB*, *ISUBB*, *IANDB*, *IORB*, and *IXORB*, and add hardware to generate *ADDB1 = IADDB* ^ *T3*, *SUBB1 = ISUBB* ^ *T3*, *ANDB1 = IANDB* ^ *T3*, *ORB1 = IORB* ^ *T3*, and *XORB1 = IXORB* ^ *T3*.

ii) OR together *ADDB1*, *SUBB1*, *ANDB1*, *ORB1*, and *XORB1* to generate *BBUS*.

iii) Add the same five signals to the OR gate driving *CLR* of the counter.

iv) Change *ALUS*[1..7] such that *ADD1* is replaced by *ADD1* ∨ *ADDB1*, and so on for *SUB1*, *AND1*, *OR1*, and *XOR1*, yielding:

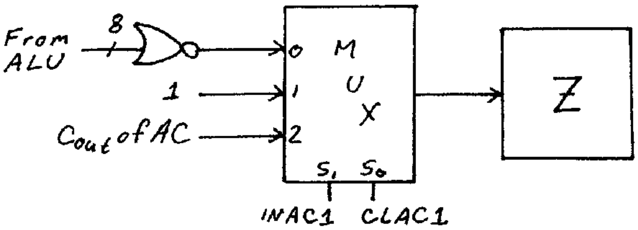
ALUS1 = ADD1 ∨ *ADDB1* ∨ *SUB1* ∨ *SUBB1* ∨ *INAC1*
ALUS2 = SUB1 ∨ *SUBB1*
ALUS3 = LDAC5 ∨ *MOVR1* ∨ *ADD1*∨ *ADDB1*
ALUS4 = SUB1 ∨ *SUBB1*∨ *INAC1*
ALUS5 = XOR1 ∨ *XORB1* ∨ *NOT1*
ALUS6 = OR1 ∨ *ORB1* ∨ *NOT1*
ALUS7 = AND1 ∨ *ANDB1* ∨ *OR1* ∨ *ORB1* ∨ *XOR1* ∨ *XORB1* ∨ *NOT1*



26. Initially *AC* = 1 and *B* = 2. Fetch cycles not shown.

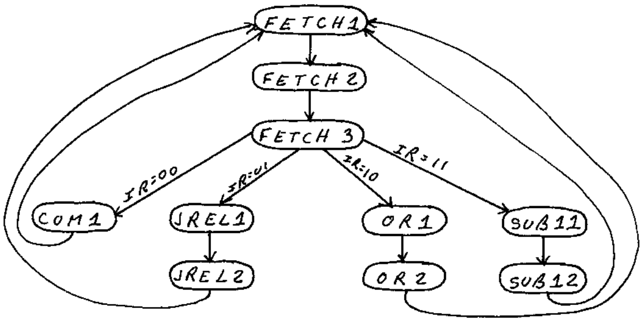
Instruction	State	Operations performed
ORB	ORB1	*AC* ← 1 ∨ 2 = 3
ADDB	ADDB1	*AC* ← 3 + 2 = 5
ANDB	ANDB1	*AC* ← 5 ^ 2 = 0
XORB	XORB1	*AC* ← 0 ⊕ 2 = 2
SUBB	SUBB1	*AC* ← 2 - 2 = 0

27. i.) Remove *CLAC1* and *INAC1* as inputs to the OR gate which generates *ACLOAD*.
ii) Add control inputs to AC: *CLR* = *CLAC1*, and *INC* = *INAC1*.
iii) Change the input to *Z* as shown below. *ZLOAD* is unchanged.

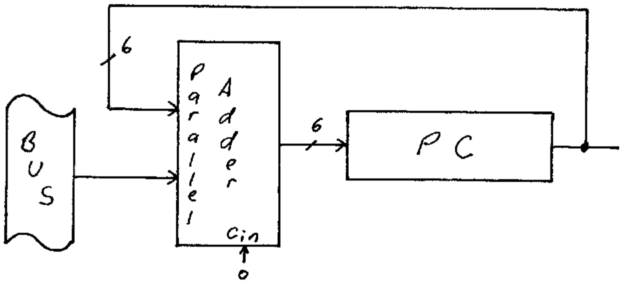


28. State diagram and RTL code:

FETCH1:	*AR* ← *PC*
FETCH2:	*DR* ← *M*, *PC* ← *PC* + 1
FETCH3:	*IR* ← *DR*[7..6], *AR* ← *DR*[5..0]
COM1:	*AC* ← *AC'*
JREL1:	*DR* ← *M*
JREL2:	*PC* ← *PC* + *DR*[5..0]
OR1:	*DR* ← *M*
OR2:	*AC* ← *AC* ∨ *DR*
SUB11:	*DR* ← *M*
SUB12:	*AC* ← *AC* + *DR'*



The register section is the same as Figure 6.6, except for the data input to *PC*, shown below.



Control signals:

$ARLOAD = FETCH1 \vee FETCH3$

$PCLOAD = JREL2$

$PCINC = FETCH2$

$PCBUS = FETCH1$

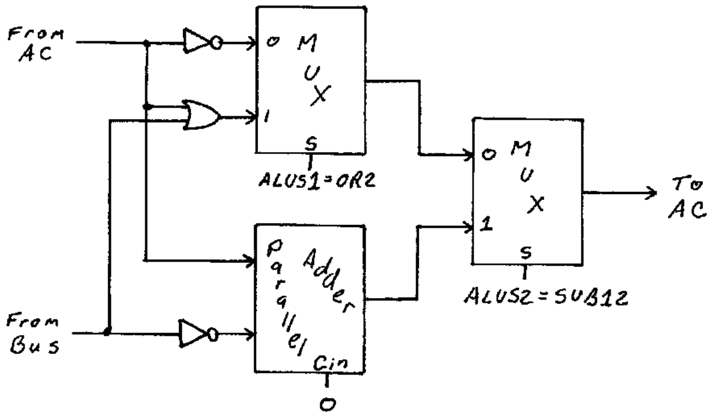
$DRLOAD = MEMBUS = READ = FETCH2 \vee JREL1 \vee OR1 \vee SUB11$

$DRBUS = FETCH3 \vee JREL2 \vee OR2 \vee SUB12$

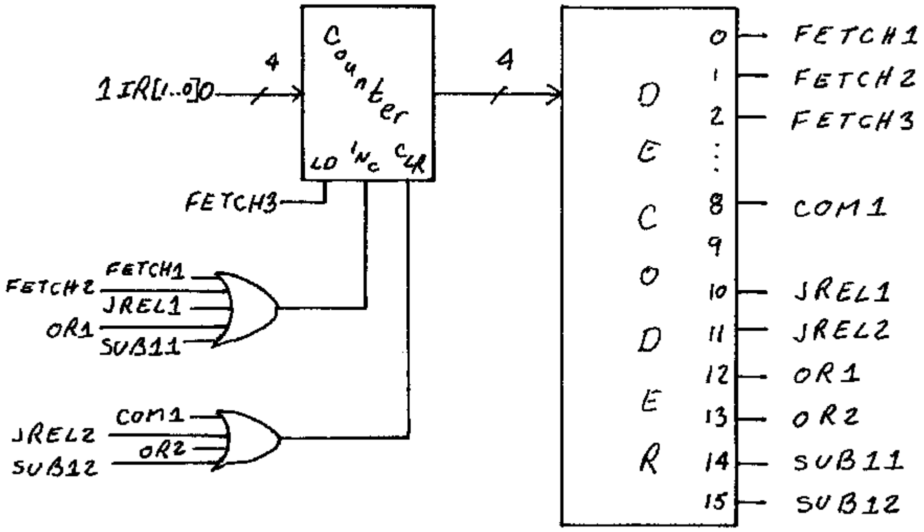
$ACLOAD = COM1 \vee OR2 \vee SUB12$

$IRLOAD = FETCH3$

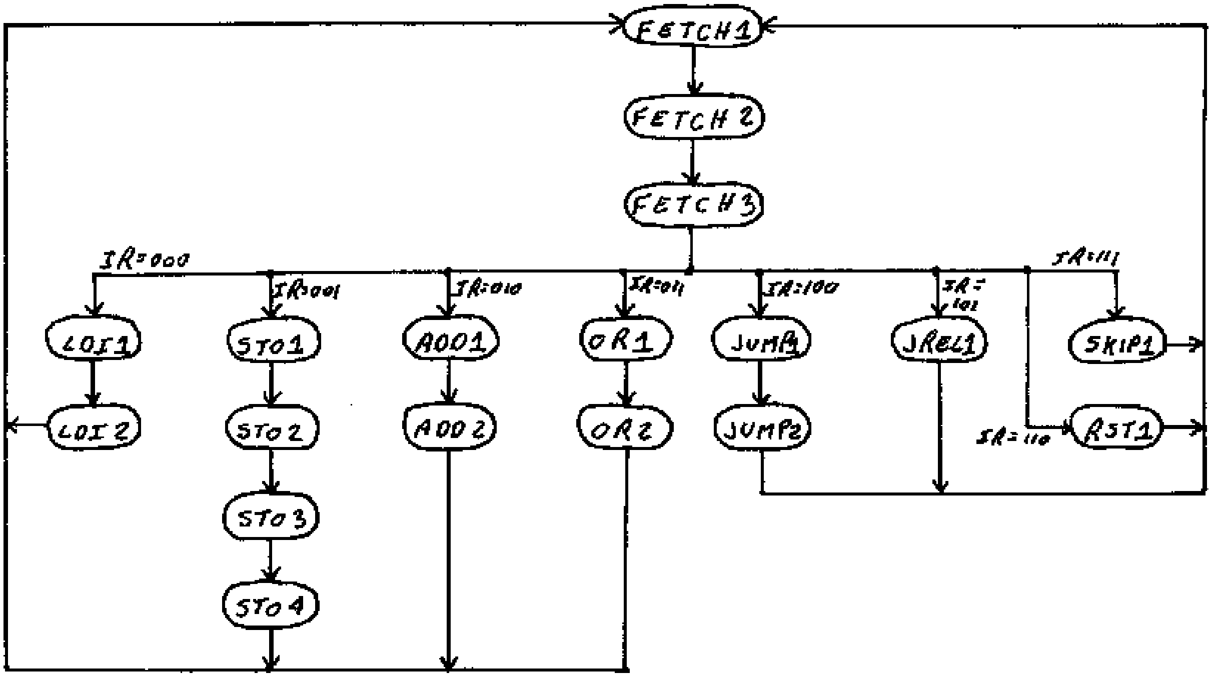
ALU:



Control unit:



29. State diagram and RTL code:

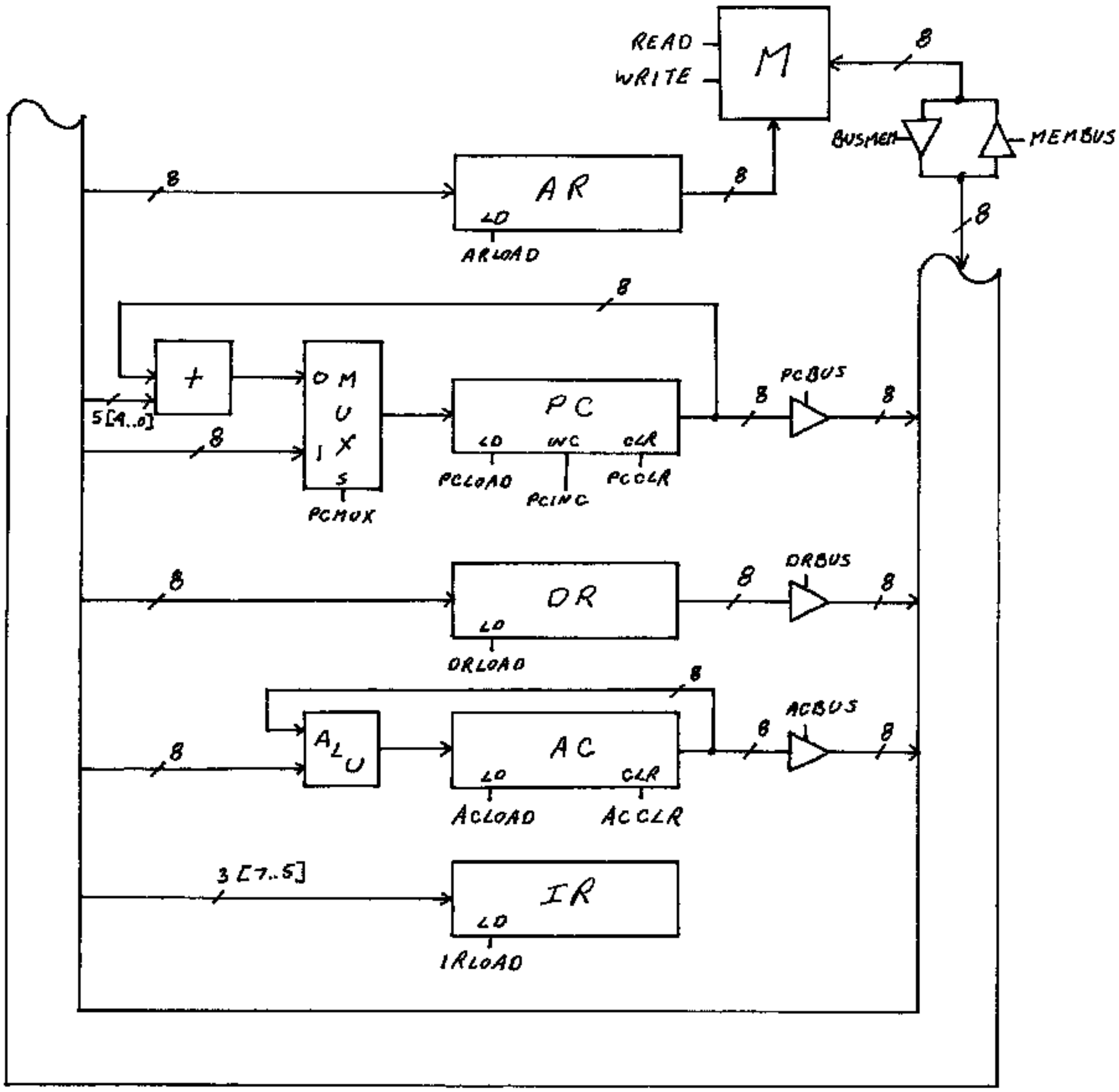


FETCH1:	$AR \leftarrow PC$	ADD1:	$DR \leftarrow M, PC \leftarrow PC + 1$
FETCH2:	$DR \leftarrow M, PC \leftarrow PC + 1$	ADD2:	$AC \leftarrow AC + DR$
FETCH3:	$IR \leftarrow DR[7..5], AR \leftarrow PC$	OR1:	$DR \leftarrow M, PC \leftarrow PC + 1$
LDI1:	$DR \leftarrow M, PC \leftarrow PC + 1$	OR2:	$AC \leftarrow AC \vee DR$
LDI2:	$AC \leftarrow DR$	JUMP1:	$DR \leftarrow M$
STO1:	$DR \leftarrow M, PC \leftarrow PC + 1$	JUMP2:	$PC \leftarrow DR$
STO2:	$AR \leftarrow DR$	JREL1:	$PC \leftarrow PC + 000DR[4..0]$
STO3:	$DR \leftarrow AC$	SKIP1:	$PC \leftarrow PC + 1$
STO4:	$M \leftarrow DR$	RST1:	$PC \leftarrow 0, AC \leftarrow 0$

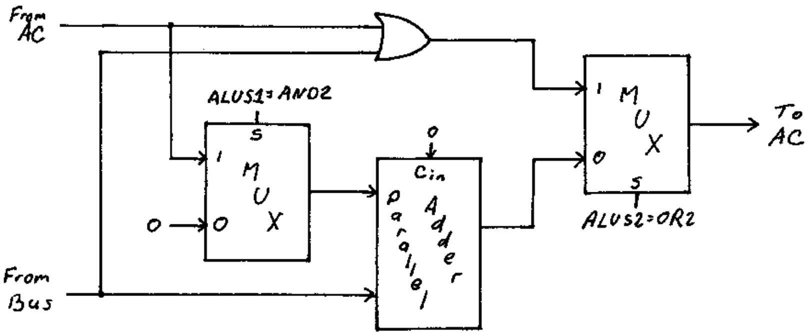
Control signals:

$ARLOAD = FETCH1 \vee FETCH3 \vee STO2$
$PCLOAD = JUMP2 \vee JREL2$
$PCINC = FETCH2 \vee LDI1 \vee STO1 \vee ADD1 \vee OR1 \vee SKIP1$
$PCCLR = RST1$
$PCBUS = FETCH1 \vee FETCH3$
$PCMUX = JUMP2$
$DRLOAD = FETCH2 \vee LDI1 \vee STO1 \vee STO3 \vee ADD1 \vee OR1 \vee JUMP1$
$DRBUS = FETCH3 \vee LDI2 \vee STO2 \vee STO4 \vee ADD2 \vee OR2 \vee JUMP2 \vee JREL1$
$ACLOAD = LDI2 \vee ADD2 \vee OR2$
$ACCLR = RST1$
$ACBUS = STO3$
$IRLOAD = FETCH3$
$MEMBUS = READ = FETCH2 \vee LDI1 \vee STO1 \vee ADD1 \vee OR1 \vee JUMP1$
$BUSMEM = WRITE = STO4$

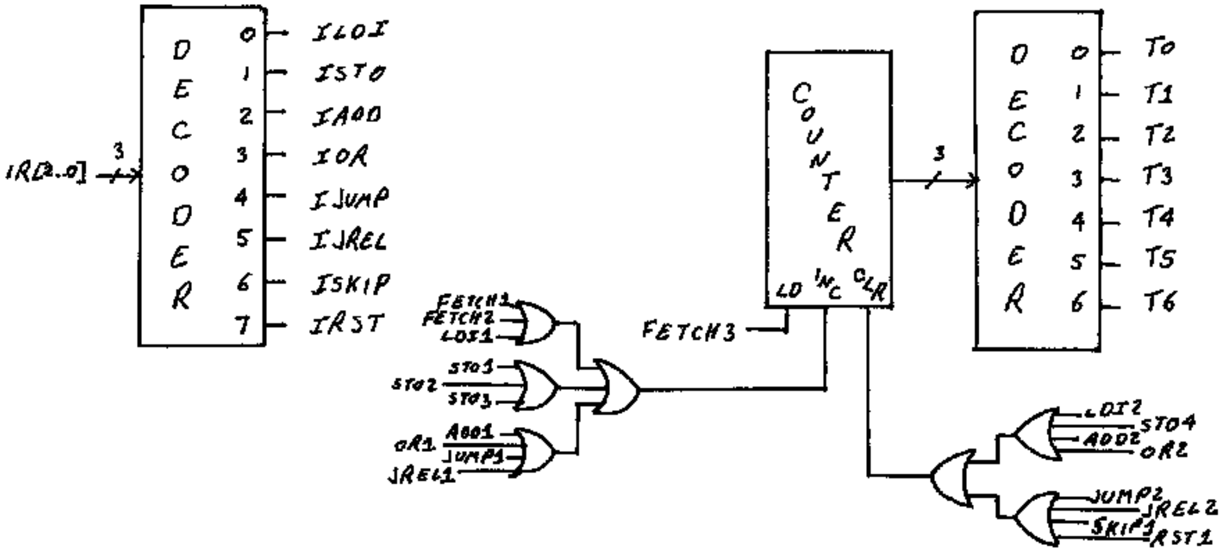
Register section:



ALU:

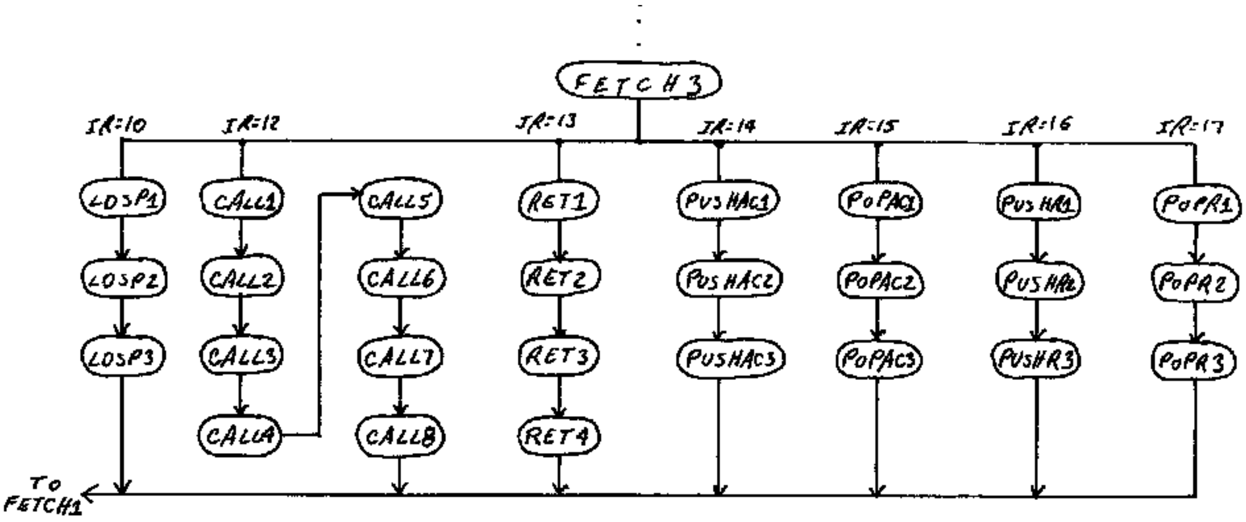


Control unit:



FETCH1 =	*T0*	*ADD1 =*	*IADD ^ T3*
FETCH2 =	*T1*	*ADD2 =*	*IADD ^ T4*
FETCH3 =	*T2*	*OR1 =*	*IOR ^ T3*
LDI1 =	*ILDI ^ T3*	*OR2 =*	*IOR ^ T4*
LDI2 =	*ILDI ^ T4*	*JUMP1 =*	*IJUMP ^ T3*
STO1 =	*ISTO ^ T3*	*JUMP2 =*	*IJUMP ^ T4*
STO2 =	*ISTO ^ T4*	*JREL1 =*	*IJREL ^ T3*
STO3 =	*ISTO ^ T5*	*JREL2 =*	*IJREL ^ T4*
STO4 =	*ISTO ^ T6*	*SKIP1 =*	*ISKIP ^ T3*
		RST1 =	*IRST ^ T3*

30. Modified state diagram:



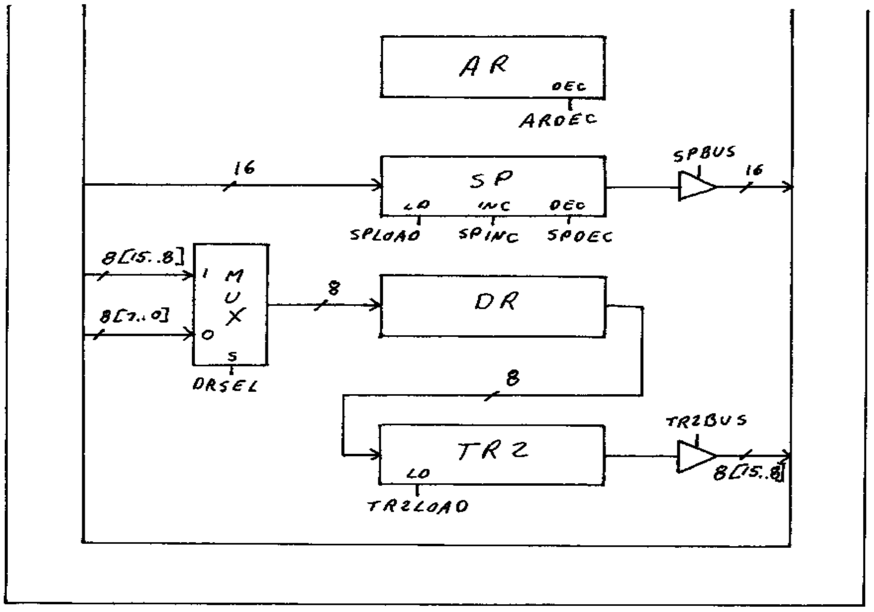
Modified RTL code:

LDSP1: *DR* ← *M*, *AR* ← *AR* + 1, *PC* ← *PC* + 1
LDSP2: *TR* ← *DR*, *DR* ← *M*, *PC* ← *PC* + 1
LDSP3: *SP* ← *DR*,*TR*
CALL1: *DR* ← *M*, *AR* ← *AR* + 1, *PC* ← *PC* + 1
CALL2: *TR* ← *DR*, *DR* ← *M*, *PC* ← *PC* + 1
CALL3: *TR*2 ← *DR*, *DR* ← *PC*[15..8], *SP* ← *SP* - 1
CALL4: *AR* ← *SP*
CALL5: *M* ← *DR*, *AR* ← *AR* - 1, *SP* ← *SP* - 1
CALL6: *DR* ← *PC*[7..0]
CALL7: *M* ← *DR*
CALL8: *PC* ← *TR2*,*DR*
RET1: *AR* ← *SP*
RET2: *DR* ← *M*, *SP* ← *SP* + 1, *AR* ← *AR* + 1
RET3: *TR* ← *DR*, *DR* ← *M*, *SP* ← *SP* + 1
RET4: *PC* ← *DR*,*TR*

PUSHAC1: *SP* ← *SP* - 1, *DR* ← *AC*
PUSHAC2: *AR* ← *SP*
PUSHAC3: *M* ← *DR*
POPAC1: *AR* ← *SP*
POPAC2: *DR* ← *M*, *SP* ← *SP* + 1
POPAC3: *AC* ← *DR*
PUSHR1: *SP* ← *SP* - 1, *DR* ← *R*
PUSHR2: *AR* ← *SP*
PUSHR3: *M* ← *DR*
POPR1: *AR* ← *SP*
POPR2: *DR* ← *M*, *SP* ← *SP* + 1
POPR3: *R* ← *DR*

Modified register section: (shown below)

- New registers: *SP* (with *LD*, *DEC*, *INC*), *TR2* (with *LD*, receives data directly from *DR*)
- New control signal: *AR* adds a *DEC* signal
- New data path: *DR* can receive data from *BUS*[15..8] or *BUS*[7..0]
- All other connections remain the same as shown in Figure 6.15.



New control signals:

ARDEC = *CALL5*
SPLOAD = *LDSP3*
SPINC = *RET2* ∨ *RET3* ∨ *POPAC2* ∨ *POPR2*
SPDEC = *CALL3* ∨ *CALL5* ∨ *PUSHAC1* ∨ *PUSHR1*
SPBUS = *CALL4* ∨ *RET1* ∨ *PUSHAC2* ∨ *POPAC1* ∨ *PUSHR2* ∨ *POPR1*
DRSEL = *CALL3*
TR2LOAD = *CALL3*
TR2BUS = *CALL8*

Modified control signals:

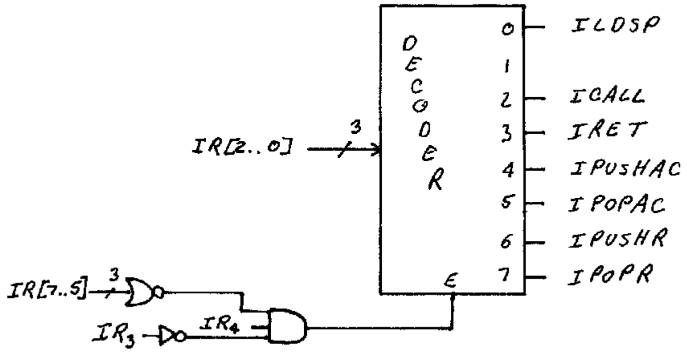
ARLOAD = (original value) ∨ *CALL4* ∨ *RET1* ∨ *PUSHAC2* ∨ *POPAC1* ∨ *PUSHR2* ∨ *POPR1*
ARINC = (original value) ∨ *LDSP1* ∨ *CALL1* ∨ *RET2*
PCLOAD = (original value) ∨ *CALL8* ∨ *RET4*
PCINC = (original value) ∨ *LDSP1* ∨ *LDSP2* ∨ *CALL1* ∨ *CALL2*
PCBUS = (original value) ∨ *CALL3* ∨ *CALL6*
DRLOAD = (original value) ∨ *LDSP1* ∨ *LDSP2* ∨ *CALL1* ∨ *CALL2* ∨ *CALL3* ∨ *CALL6* ∨ *RET2* ∨ *RET3* ∨ *PUSHAC1* ∨ *POPAC2* ∨ *PUSHR1* ∨ *POPR2*
DRHBUS = (original value) ∨ *LDSP3* ∨ *RET4*
DRLBUS = (original value) ∨ *CALL5* ∨ *CALL7* ∨ *PUSHAC3* ∨ *POPAC3* ∨ *PUSHR3* ∨ *POPR3*
TRLOAD = (original value) ∨ *LDSP2* ∨ *CALL2* ∨ *RET3*
TRBUS = (original value) ∨ *LDSP3* ∨ *CALL8* ∨ *RET4*
RLOAD = (original value) ∨ *POPR3*
RBUS = (original value) ∨ *PUSHR1*
ACLOAD = (original value) ∨ *POPAC3*
ACBUS = (original value) ∨ *PUSHAC1*
ALUS1 = (original value) ∨ *POPAC3*
MEMBUS = (original value) ∨ *LDSP1* ∨ *LDSP2* ∨ *CALL1* ∨ *CALL2* ∨ *RET2* ∨ *RET3* ∨ *POPAC2* ∨ *POPR2*
BUSMEM = (original value) ∨ *CALL5* ∨ *CALL7* ∨ *PUSHAC3* ∨ *PUSHR3*
WRITE = (original value) ∨ *CALL5* ∨ *CALL7* ∨ *PUSHAC3* ∨ *PUSHR3*

Control unit modifications:

- Increase the counter size to 4 bits. The decoder now outputs T_0 - T_{10}.
- Add a second instruction decoder as shown below.
- Modify the *INC* and *CLR* inputs to the counter as follows:

INC = (original value) ∨ *LDSP1* ∨ *LDSP2* ∨ *CALL1* ∨ *CALL2* ∨ *CALL3* ∨ *CALL4* ∨ *CALL5* ∨ *CALL6* ∨ *CALL7* ∨ *RET1* ∨ *RET2* ∨ *RET3* ∨ *PUSHAC1* ∨ *PUSHAC2* ∨ *POPAC1* ∨ *POPAC2* ∨ *PUSHR1* ∨ *PUSHR2* ∨ *POPR1* ∨ *POPR2*

CLR = (original value) ∨ *LDSP3* ∨ *CALL8* ∨ *RET4* ∨ *PUSHAC3* ∨ *POPAC3* ∨ *PUSHR3* ∨ *POPR3*



LDSP1 =	*ILDSP* ^ *T3*	*PUSHAC1* =	*IPUSHAC* ^ *T3*
LDSP2 =	*ILDSP* ^ *T4*	*PUSHAC2* =	*IPUSHAC* ^ *T4*
LDSP3 =	*ILDSP* ^ *T5*	*PUSHAC3* =	*IPUSHAC* ^ *T5*
CALL1 =	*ICALL* ^ *T3*	*POPAC1* =	*IPOPAC* ^ *T3*
CALL2 =	*ICALL* ^ *T4*	*POPAC2* =	*IPOPAC* ^ *T4*
CALL3 =	*ICALL* ^ *T5*	*POPAC3* =	*IPOPAC* ^ *T5*
CALL4 =	*ICALL* ^ *T6*	*PUSHR1* =	*IPUSHR* ^ *T3*
CALL5 =	*ICALL* ^ *T7*	*PUSHR2* =	*IPUSHR* ^ *T4*
CALL6 =	*ICALL* ^ *T8*	*PUSHR3* =	*IPUSHR* ^ *T5*
CALL7 =	*ICALL* ^ *T9*	*POPR1* =	*IPOPR* ^ *T3*
CALL8 =	*ICALL* ^ *T10*	*POPR2* =	*IPOPR* ^ *T4*
RET1 =	*IRET* ^ *T3*	*POPR3* =	*IPOPR* ^ *T5*
RET2 =	*IRET* ^ *T4*		
RET3 =	*IRET* ^ *T5*		
RET4 =	*IRET* ^ *T6*		

Chapter 7

1.

IR	MAP
00	0011
01	0101
10	0111
11	1000

MAP = $IR_1 \wedge IR_0, IR_1 \oplus IR_0, IR_0', (IR_1 \wedge IR_0)'$

2.

State	Address	SEL	ARPC	AIDR	PCIN	PCDR	DRM	PLUS	AND	ACIN	ADDR
FETCH1	0000 (0)	0	1	0	0	0	0	0	0	0	0001
FETCH2	0001 (1)	0	0	0	1	0	1	0	0	0	0010
FETCH3	0010 (2)	1	0	1	0	0	0	0	0	0	XXXX
ADD1	0011 (3)	0	0	0	0	0	1	0	0	0	0100
ADD2	0100 (4)	0	0	0	0	0	0	1	0	0	0000
AND1	0101 (5)	0	0	0	0	0	1	0	0	0	0110
AND2	0110 (6)	0	0	0	0	0	0	0	1	0	0000
JMP1	0111 (7)	0	0	0	0	1	0	0	0	0	0000
INC1	1000 (8)	0	0	0	0	0	0	0	0	1	0000

3.

M1	M2
NOP	NOP
$DR \leftarrow M$	$PC \leftarrow PC + 1$
$AC \leftarrow AC'$	$AC \leftarrow AC + 1$
$DR \leftarrow DR + 1$	$PC \leftarrow PC + DR[5..0]$
$AR \leftarrow PC$	
$IR,AR \leftarrow DR$	
AC ← *AC* Å *DR*	
$M \leftarrow DR$	

Required
Micro-operations in these two rows must be allocated the same relative to each other
The remaining operations are assigned arbitrarily

4. Test program:

```
0:  ADD 4
1:  AND 5
2:  INC
3:  JMP 0
4:  27H
5:  39H
```

Instruction	State	Address	Micro-operations	Operations performed	Next Address
ADD 4	FETCH1	0000	ARPC	*AR* ← 0	0001
	FETCH2	0001	DRM, PCIN	*DR* ← 04H, *PC* ← 1	0010
	FETCH3	0010	AIDR	*IR* ← 00, *AR* ← 04H	1000
	ADD1	1000	DRM	*DR* ← 27H	1001
	ADD2	1001	PLUS	*AC* ← 0 + 27H = 27H	0000
AND 5	FETCH1	0000	ARPC	*AR* ← 1	0001
	FETCH2	0001	DRM, PCIN	*DR* ← 45H, *PC* ← 2	0010
	FETCH3	0010	AIDR	*IR* ← 01, *AR* ← 05H	1010
	AND1	1010	DRM	*DR* ← 39H	1011
	AND2	1011	AND	*AC* ← 27H ^ 39H = 31H	0000
INC	FETCH1	0000	ARPC	*AR* ← 2	0001
	FETCH2	0001	DRM, PCIN	*DR* ← C0H, *PC* ← 3	0010
	FETCH3	0010	AIDR	*IR* ← 11, *AR* ← 00H	1110
	INC1	1110	ACIN	*AC* ← 21H + 1 = 22H	0000
JMP 0	FETCH1	0000	ARPC	*AR* ← 3	0001
	FETCH2	0001	DRM, PCIN	*DR* ← 80H, *PC* ← 4	0010
	FETCH3	0010	AIDR	*IR* ← 10, *AR* ← 00H	1100
	JMP1	1100	PCDR	*PC* ← 0	0000

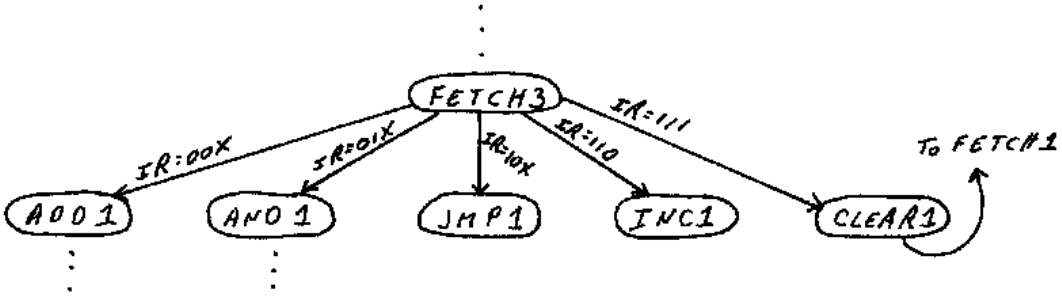
5. Use the same test program as in problem 4.

Instruction	State	Address	M1	M2	Operations performed	Next Address
ADD 4	FETCH1	0000	ARPC	NOP	*AR* ← 0	0001
	FETCH2	0001	DRM	PCIN	*DR* ← 04H, *PC* ← 1	0010
	FETCH3	0010	AIDR	NOP	*IR* ← 00, *AR* ← 04H	1000
	ADD1	1000	DRM	NOP	*DR* ← 27H	1001
	ADD2	1001	PLUS	NOP	*AC* ← 0 + 27H = 27H	0000
AND 5	FETCH1	0000	ARPC	NOP	*AR* ← 1	0001
	FETCH2	0001	DRM	PCIN	*DR* ← 45H, *PC* ← 2	0010
	FETCH3	0010	AIDR	NOP	*IR* ← 01, *AR* ← 05H	1010
	AND1	1010	DRM	NOP	*DR* ← 39H	1011
	AND2	1011	AND	NOP	*AC* ← 27H ^ 39H = 31H	0000
INC	FETCH1	0000	ARPC	NOP	*AR* ← 2	0001
	FETCH2	0001	DRM	PCIN	*DR* ← C0H, *PC* ← 3	0010
	FETCH3	0010	AIDR	NOP	*IR* ← 11, *AR* ← 00H	1110
	INC1	1110	ACIN	NOP	*AC* ← 21H + 1 = 22H	0000
JMP 0	FETCH1	0000	ARPC	NOP	*AR* ← 3	0001
	FETCH2	0001	DRM	PCIN	*DR* ← 80H, *PC* ← 4	0010
	FETCH3	0010	AIDR	NOP	*IR* ← 10, *AR* ← 00H	1100
	JMP1	1100	PCDR	NOP	*PC* ← 0	0000

6. Use the same test program as in problem 4.

Instruction	**State**	**Address**	**Control Signals**	**Operations performed**	**Next Address**
ADD 4	FETCH1	0000	PCBUS, ARLOAD	*AR* ← 0	0001
	FETCH2	0001	READ, MEMBUS, DRLOAD, PCINC	*DR* ← 04H, *PC* ← 1	0010
	FETCH3	0010	DRBUS, ARLOAD, IRLOAD	*IR* ← 00, *AR* ← 04H	1000
	ADD1	1000	READ, MEMBUS, DRLOAD	*DR* ← 27H	1001
	ADD2	1001	DRBUS, ACLOAD	*AC* ← 0 + 27H = 27H	0000
AND 5	FETCH1	0000	PCBUS, ARLOAD	*AR* ← 1	0001
	FETCH2	0001	READ, MEMBUS, DRLOAD, PCINC	*DR* ← 45H, *PC* ← 2	0010
	FETCH3	0010	DRBUS, ARLOAD, IRLOAD	*IR* ← 01, *AR* ← 05H	1010
	AND1	1010	READ, MEMBUS, DRLOAD	*DR* ← 39H	1011
	AND2	1011	DRBUS, ALUSEL, ACLOAD	*AC* ← 27H ^ 39H = 31H	0000
INC	FETCH1	0000	PCBUS, ARLOAD	*AR* ← 2	0001
	FETCH2	0001	READ, MEMBUS, DRLOAD, PCINC	*DR* ← C0H, *PC* ← 3	0010
	FETCH3	0010	DRBUS, ARLOAD, IRLOAD	*IR* ← 11, *AR* ← 00H	1110
	INC1	1110	ACINC	*AC* ← 21H + 1 = 22H	0000
JMP 0	FETCH1	0000	PCBUS, ARLOAD	*AR* ← 3	0001
	FETCH2	0001	READ, MEMBUS, DRLOAD, PCINC	*DR* ← 80H, *PC* ← 4	0010
	FETCH3	0010	DRBUS, ARLOAD, IRLOAD	*IR* ← 10, *AR* ← 00H	1100
	JMP1	1100	DRBUS, PCLOAD	*PC* ← 0	0000

7. Modified state diagram: (same as for problem 6.6)



Modified RTL code: (same as for problem 6.6)

FETCH3: *IR* ← *DR*[7..5], *AR* ← *DR*[5..0]
CLEAR1: *AC* ← 0

Microsequencer modifications:

Change the mapping hardware so that its inputs are *IR*[2..0] and its outputs are 1,*IR*[2..1],($IR_2 \wedge IR_1 \wedge IR_0$).

Register modifications: (same as for problem 6.7)

i) *IR* must have 3 bits instead of 2. It receives bus bits 7..5 as its inputs. During FETCH3, bit 5 of *DR* is sent to both *IR* and *AR*.
ii) *AC* needs a *CLR* input (*ACCLR* is a new control signal which connects to the new *CLR* input.)

Microcode modifications:

i) Add mico-operation *ACCL*, which sets *AC* ← 0. Connect this bit of the microsequencer to the *CLR* input of *AC*.
ii) Add the following to microcode memory. Set the *ACCL* field to 0 for all other microinstructions.

State	**Address**	**S**	**A**	**A**	**P**	**P**	**D**	**P**	**A**	**A**	**A**	**ADDR**
		E	**R**	**I**	**C**	**C**	**R**	**L**	**N**	**C**	**C**	
		L	**P**	**D**	**I**	**D**	**M**	**U**	**D**	**I**	**C**	
			C	**R**	**N**	**R**		**S**		**N**	**L**	
CLEAR1	1111 (15)	0	0	0	0	0	0	0	0	0	1	0000

Verification: Test program: 0: CLEAR

Instruction	**State**	**Address**	**Micro-operations**	**Operations performed**	**Next Address**
CLEAR	FETCH1	0000	ARPC	*AR* ← 0	0001
	FETCH2	0001	DRM, PCIN	*DR* ← 04H, *PC* ← 1	0010
	FETCH3	0010	AIDR	*IR* ← 111, *AR* ← 04H	1111
	CLEAR1	1111	ACCL	*AC* ← 00	0000

8. The modifications are the same as in problem 7 with the following exceptions.

i) Label output 0 of the M1 decoder *ACCL*. (This can be done because the NOP of M1 is never used. If it was used, a new micro-operation code would have to be created.) The code for *ACCL* is M1 = 000.
ii) Connect *ACCL* to *ACCLR*.
iii) Add the following to microcode memory.

State	Address	SEL	M1	M2	ADDR
CLEAR1	1111	0	000	0	0000

Verification: Test program: 0: CLEAR

Instruction	State	Address	M1	M2	Operations performed	Next Address
CLEAR	FETCH1	0000	ARPC	NOP	*AR* ← 0	0001
	FETCH2	0001	DRM	PCIN	*DR* ← 04H, *PC* ← 1	0010
	FETCH3	0010	AIDR	NOP	*IR* ← 111, *AR* ← 04H	1111
	CLEAR1	1111	ACCL	NOP	*AC* ← 00	0000

9. The modifications are the same as in problem 7 with the following exceptions.

i) Add control signal output *ACCLR* to the control signals in microcode memory. Set it to 0 for all microinstructions except the microinstruction at address 1111.
ii) Add the following to microcode memory.

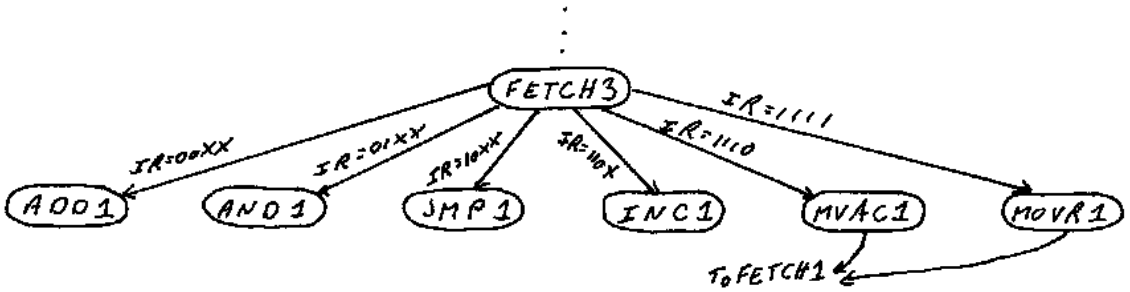
State	Address	SEL	ARLOAD	PCLOAD	PCINC	DMR	ACLOAD	ACINC	IRLOAD	ALUSEL	PCBUS	DRBUS	ACCLR	ADDR
CLEAR1	1111 (15)	0	0	0	0	0	0	0	0	0	0	0	1	0000

Verification: Test program: 0: CLEAR

Instruction	State	Address	Control Signals	Operations performed	Next Address
CLEAR	FETCH1	0000	PCBUS, ARLOAD	*AR* ← 0	0001
	FETCH2	0001	READ, MEMBUS, DRLOAD, PCINC	*DR* ← 04H, *PC* ← 1	0010
	FETCH3	0010	DRBUS, ARLOAD, IRLOAD	*IR* ← 111, *AR* ← 04H	1111
	CLEAR1	1111	ACCLR	*AC* ← 00	0000

10. Modified state diagram: (same as for problem 6.10)

(*IR* must have 4 bits instead of 2.)



Modified RTL code: (same as for problem 6.10)

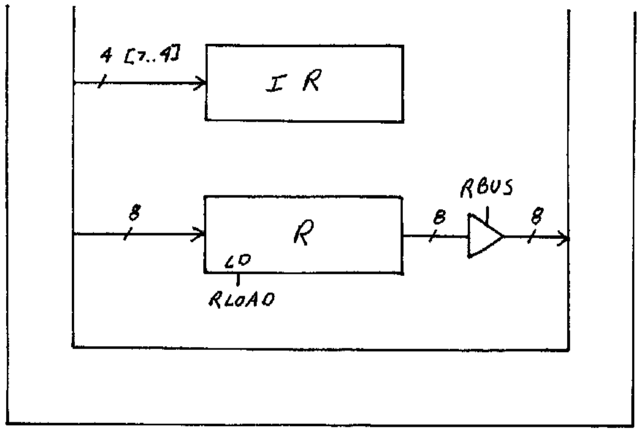
FETCH3: *IR* ← *DR*[7..4], *AR* ← *DR*[5..0]
MVAC1: *R* ← *AC*
MOVR1: *AC* ← *R*

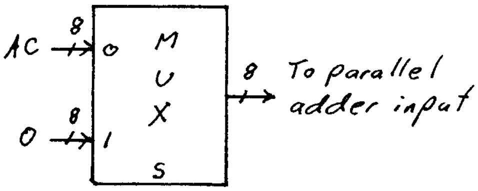
Microsequencer modifications:

i) Change the mapping hardware so that its inputs are *IR*[3..0] and its outputs are ($IR_3 \wedge IR_2 \wedge IR_1$),*IR*[3..2],($IR_3 \wedge IR_2 \wedge IR_1 \wedge IR_0$).
ii) Add micro-operation *RAC* (*R* ← *AC*); connect it to an OR gate that generates *ACBUS* and have it directly drive *RLOAD*.
iii) Add micro-operation *ACR* (*AC* ← *R*); connect it to directly to *RBUS* and connect it to an OR gate that generates *ACLOAD*.

Register and ALU modifications: (same as for problem 6.11)

i.) *IR* must have 4 bits instead of 2. It receives bus bits 7..4 as its inputs. During FETCH3, bit *DR*[5..4] is sent to both *IR* and *AR*. This is shown below.
ii) Register *R* is added to the CPU. It receives data from the bus and sends data to the bus through tri-state buffers. It requires only a *LD* signal. This is shown below.
iii) The ALU is modified as shown below.





Microcode modifications:

Add the following to microcode memory. Set the *RAC* and *ACR* fields to 0 for all other microinstructions.

State	**Address**	**SEL**	**ARPC**	**AIDR**	**PCIN**	**PCDR**	**DRM**	**PLUS**	**AND**	**ACIN**	**RAC**	**ACR**	**ADDR**
MVAC1	0110 (6)	0	0	0	0	0	0	0	0	0	1	0	0000
MOVR1	0111 (7)	0	0	0	0	0	0	0	0	0	0	1	0000

Verification: Test program: 0: MVAC (Initially *AC* = 1)
1: MOVR

Instruction	**State**	**Address**	**Micro-operations**	**Operations performed**	**Next Address**
MVAC	FETCH1	0000	ARPC	*AR* ← 0	0001
	FETCH2	0001	DRM, PCIN	*DR* ← E0H, *PC* ← 1	0010
	FETCH3	0010	AIDR	*IR* ← 1110, *AR* ← 20H	0110
	MVAC1	0110	RAC	*R* ← 01H	0000
MOVR	FETCH1	0000	ARPC	*AR* ← 1	0001
	FETCH2	0001	DRM, PCIN	*DR* ← F0H, *PC* ← 2	0010
	FETCH3	0010	AIDR	*IR* ← 1111, *AR* ← 30H	0111
	MOVR1	0111	ACR	*AC* ← 01H	0000

11. The modifications are the same as in problem 10 with the following exceptions.

i) Add micro-operations *RAC* (*R* ← *AC*) and *ACR* (*AC* ← *R*) to M2 with the following assignments.

M2	**Micro-operation**
00	NOP
01	PCIN
10	RAC
11	ACR

ii) Use a 2-to-4 decoder to generate the control signals for M2.
iii) Modify the existing microinstructions to accommodate the new values for M2 (0 → 00, 1 → 01).
iv) Add the following to microcode memory.

State	**Address**	**SEL**	**M1**	**M2**	**ADDR**
MVAC1	0110	0	000	10	0000
MOVR1	0111	0	000	11	0000

Verification: Test program: 0: MVAC (Initially *AC* = 1)
1: MOVR

Instruction	**State**	**Address**	**M1**	**M2**	**Operations performed**	**Next Address**
MVAC	FETCH1	0000	ARPC	NOP	*AR* ← 0	0001
	FETCH2	0001	DRM	PCIN	*DR* ← E0H, *PC* ← 1	0010
	FETCH3	0010	AIDR	NOP	*IR* ← 1110, *AR* ← 20H	0110
	MVAC1	0110	NOP	RAC	*R* ← 01H	0000
MOVR	FETCH1	0000	ARPC	NOP	*AR* ← 1	0001
	FETCH2	0001	DRM	PCIN	*DR* ← F0H, *PC* ← 2	0010
	FETCH3	0010	AIDR	NOP	*IR* ← 1111, *AR* ← 30H	0111
	MOVR1	0111	NOP	ACR	*AC* ← 01H	0000

12. The modifications are the same as in problem 10 with the following exceptions.

i) Add control signals *RLOAD*, *RBUS*, *ACBUS*, and *ALUS2* to the control signals in microcode memory. Set them to 0 for all microinstructions except those at addresses 0110 and 0111.

ii) Add the following to microcode memory.

State	**Address**	**SEL**	**ARLOAD**	**PCLOAD**	**PCINC**	**DMR**	**ACLOAD**	**ACINC**	**IRLOAD**	**ALUSEL**	**PCBUS**	**DRBUS**	**RLOAD**	**RBUS**	**PCBUS**	**ALUS2**	**ADDR**
MVAC1	0110 (6)	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0000
MOVR1	0111 (7)	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0000

<u>Verification</u>: Test program: 0: MVAC (Initially *AC* = 1)
1: MOVR

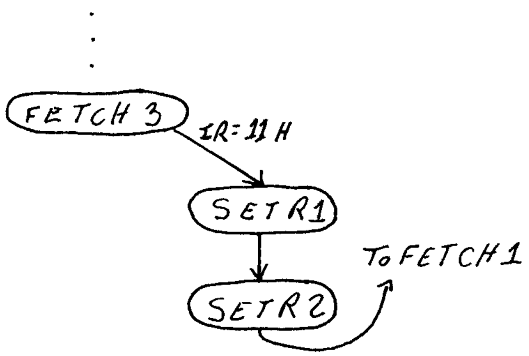
Instruction	**State**	**Address**	**Control Signals**	**Operations performed**	**Next Address**
MVAC	FETCH1	0000	PCBUS, ARLOAD	*AR* ← 0	0001
	FETCH2	0001	READ, MEMBUS, DRLOAD, PCINC	*DR* ← E0H, *PC* ← 1	0010
	FETCH3	0010	DRBUS, ARLOAD, IRLOAD	*IR* ← 1110, *AR* ← 20H	0110
	MVAC1	0110	ACBUS, RLOAD	*R* ← 01H	0000
MOVR	FETCH1	0000	ARPC	*AR* ← 1	0001
	FETCH2	0001	DRM, PCIN	*DR* ← F0H, *PC* ← 2	0010
	FETCH3	0010	AIDR	*IR* ← 1111, *AR* ← 30H	0111
	MOVR1	0111	RBUS, ACLOAD, ALUS2	*AC* ← 01H	0000

13. Some points that might be included:

- The mapping hardware change is equivalent to changing the inputs to the counter in the hardwired controller.
- The changes in the state diagram and register hardware are the same for either control unit.
- The microcode may require less hardware changes in the control unit than the hardwired control unit, especially if no new micro-operations are needed.

14. Modified state diagram and RTL code: (same as for problem 6.19)

SETR1: $R \leftarrow 0$
SETR2: $R \leftarrow R - 1$



Register and ALU modifications: (same as for problem 6.20)

R needs two additional inputs: *CLR*, driven by new control signal *RCLR*, and *DCR*, driven by new control signal *RDCR*. There are no ALU modifications.

Microcode and microsequencer modifications:

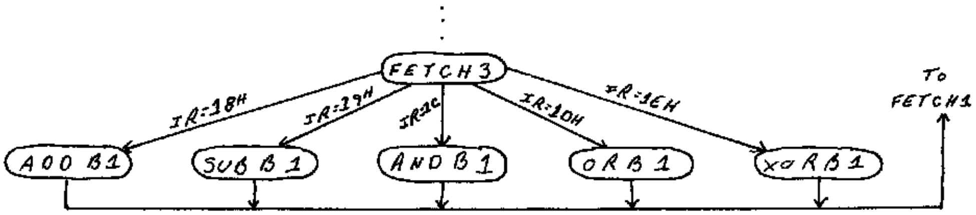
i) Add micro-operations *CLRR* ($R \leftarrow 0$) and *DECR* ($R \leftarrow R - 1$) to the microcode. These fields are set to zero for all existing microinstructions.
ii) Change the mapping function to $(IR_4 \vee IR_3)$,$(IR_4 \vee IR_2)$,$(IR_4 \vee IR_1)$, $(IR_4 \vee IR_0)$,IR_4,0.
iii) Add the following microinstructions to the microprogram.

State	Address	Condition	BT	All other micro-operations	CLRR	DECR	ADDR
SETR1	62	1	J	0	1	0	63
SETR2	63	1	J	0	0	1	1

Verification: Test program: 0: SETR

Instruction	State	Active signals	Operations performed	Next state
SETR	FETCH1	PCBUS, ARLOAD	$AR \leftarrow 0$	FETCH2
	FETCH2	READ, MEMBUS, DRLOAD, PCINC	$DR \leftarrow$ 10H, $PC \leftarrow 1$	FETCH3
	FETCH3	DRBUS, ARLOAD, IRLOAD	$IR \leftarrow$ 10H, $AR \leftarrow$ 01H	SETR1
	SETR1	CLRR	$R \leftarrow$ 00H	SETR2
	SETR2	DECR	$R \leftarrow$ FFH	FETCH1

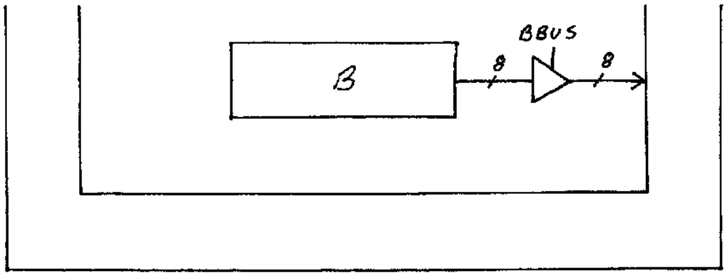
15. Modified state diagram and RTL code: (same as for problem 6.23)



ADDB1:	$AC \leftarrow AC + B$
SUBB1:	$AC \leftarrow AC - B$
ANDB1:	$AC \leftarrow AC \wedge B$
ORB1:	$AC \leftarrow AC \vee B$
XORB1:	$AC \leftarrow AC \oplus B$

Register and ALU modifications: (same as for problem 6.24)

i.) No ALU changes are needed!
ii) Register *B* is added to the CPU. It sends data to the bus through tri-state buffers but does not receive data from the bus (since it is never loaded). This is shown below.



Microcode and microsequencer modifications:

i) Add micro-operations *BPLU* ($AC \leftarrow AC + B$), *BMIN* ($AC \leftarrow AC - B$), *BAND* ($AC \leftarrow AC \wedge B$), *BOR* ($AC \leftarrow AC \vee B$), and *BXOR* ($AC \leftarrow AC \oplus B$) to the microcode. These fields are set to zero for all existing microinstructions.
ii) Change the mapping function to $IR_3,IR_2,IR_1,IR_0,0,IR_4$.
iii) Change *ALUS*[1..7] such that *ADD1* is replaced by *ADD1* ∨ *ADDB1*, and so on for *SUB1*, *AND1*, *OR1*, and *XOR1*, yielding:

$ALUS1 = ADD1 \vee ADDB1 \vee SUB1 \vee SUBB1 \vee INAC1$
$ALUS2 = SUB1 \vee SUBB1$
$ALUS3 = LDAC5 \vee MOVR1 \vee ADD1 \vee ADDB1$
$ALUS4 = SUB1 \vee SUBB1 \vee INAC1$
$ALUS5 = XOR1 \vee XORB1 \vee NOT1$
$ALUS6 = OR1 \vee ORB1 \vee NOT1$
$ALUS7 = AND1 \vee ANDB1 \vee OR1 \vee ORB1 \vee XOR1 \vee XORB1 \vee NOT1$

iv) Add the following microinstructions to the microprogram.

State	Address	Cond.	BT	All other μ-ops	BPLU	BMIN	BAND	BOR	BXOR	ADDR
ADDB1	33	1	J	0	1	0	0	0	0	1
SUBB1	37	1	J	0	0	1	0	0	0	1
ANDB1	49	1	J	0	0	0	1	0	0	1
ORB1	53	1	J	0	0	0	0	1	0	1
XORB1	57	1	J	0	0	0	0	0	1	1

Verification: Test program shown below. Fetch cycles not shown

Instruction	State	Micro-operations	Operations performed
ORB	ORB1	*BOR*	*AC* ← 1 ∨ 2 = 3
ADDB	ADDB1	*BPLU*	*AC* ← 3 + 2 = 5
ANDB	ANDB1	*BAND*	*AC* ← 5 ^ 2 = 0
XORB	XORB1	*BXOR*	*AC* ← 0 ⊕ 2 = 2
SUBB	SUBB1	*BMIN*	*AC* ← 2 - 2 = 0

16.

PCLOAD = PCDT
TRLOAD = TRDR
PCBUS = ARPC
DRHBUS = ARDT ∨ PCDT
DRLBUS = ACDR ∨ MDR
ACBUS = DRAC ∨ RAC
READ = DRM
WRITE = MDR
MEMBUS = DRM
BUSMEM = MDR

RBUS = ACR ∨ PLUS ∨ MINU ∨ AND ∨ OR ∨ XOR
ALUS1 = PLUS ∨ MINU ∨ ACIN
ALUS2 = MINU
ALUS3 = ACDR ∨ ACR ∨ PLUS
ALUS4 = MINU ∨ ACIN
ALUS5 = XOR ∨ NOT
ALUS6 = OR ∨ NOT
ALUS7 = AND ∨ OR ∨ XOR ∨ NOT
ACLOAD = ACDR ∨ ACR ∨ PLUS ∨ MINU ∨ ACIN ∨ ACZO ∨ AND ∨ OR ∨ XOR ∨ NOT

17. The subroutine now consists only of its last two instructions:

State	Address	Condition	BT	ARPC	ARIN	ARDT	PCIN	PCDT	DRM	DRAC	IRDR	RAC	ZALU	TRDR	ACDR	ACR	PLUS	MINU	ACIN	ACZO	AND	OR	XOR	NOT	MDR	ADDR
SUB1	61	1	J	0	0	0	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	62
SUB2	62	1	R	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	X

18. i) The microsequencer is the same as shown in Figure 7.8, except the micro-operations fields are input to decoders which generate the micro-operations.

ii) From the microcode of Table 7.17, the following groups of micro-operations must occur simultaneously at least once, and therefore must be located in different fields:

- PCIN, DRM, and ARIN
- ARPC, and IRDR
- PCIN, DRM, and TRDR
- ZALU, and each of the arithmetic and logic micro-operations (PLUS, MINU, ACIN, ACZO, AND, OR, XOR, and NOT)

iii) Since some microinstructions (such as NOP1) perform no micro-operations, each field requires a NOP.

One possible partitioning of the micro-operations, and its resultant microcode, are shown below.

M1		M2	M3
NOP	PLUS	NOP	NOP
TRDR	MINU	PCIN	DRM
ARIN	ACIN	IRDR	
ARPC	ACZO	ZALU	
ARDT	AND	ACDR	
PCDT	OR	ACR	
DRAC	XOR	MDR	
RAC	NOT		

State	**Address**	**Condition**	**BT**	**M1**	**M2**	**M3**	**ADDR**
FETCH1	1	1	J	ARPC	NOP	NOP	2
FETCH2	2	1	J	NOP	PCIN	DRM	3
FETCH3	3	X	M	ARPC	IRDR	NOP	X
NOP1	0	1	J	NOP	NOP	NOP	1
LDAC1	4	1	J	ARIN	PCIN	DRM	5
LDAC2	5	1	J	TRDR	PCIN	DRM	6
LDAC3	6	1	J	ARDT	NOP	NOP	7
LDAC4	7	1	J	NOP	NOP	DRM	33
LDAC5	33	1	J	NOP	ACRD	NOP	1
STAC1	8	1	J	ARIN	PCIN	DRM	9
STAC2	9	1	J	TRDR	PCIN	DRM	10
STAC3	10	1	J	ARDT	NOP	NOP	11
STAC4	11	1	J	DRAC	NOP	NOP	34
STAC5	34	1	J	NOP	MDR	NOP	1
MVAC1	12	1	J	NOP	NOP	NOP	1
MOVR1	16	1	J	NOP	NOP	NOP	1
JUMP1	20	1	J	ARIN	NOP	DRM	21
JUMP2	21	1	J	TRDR	NOP	DRM	22
JUMP3	22	1	J	PCDT	NOP	NOP	1

State	**Address**	**Condition**	**BT**	**M1**	**M2**	**M3**	**ADDR**
JMPZ1	24	*Z*¢	J	NOP	NOP	NOP	41
JMPZY1	25	1	J	ARIN	NOP	DRM	26
JMPZY2	26	1	J	TRDR	NOP	DRM	27
JMPZY3	27	1	J	PCDT	NOP	NOP	1
JMPZN1	41	1	J	NOP	PCIN	NOP	42
JMPZN2	42	1	J	NOP	PCIN	NOP	1
JPNZ1	28	*Z*	J	NOP	NOP	NOP	45
JPNZY1	29	1	J	ARIN	NOP	DRM	30
JPNZY2	30	1	J	TRDR	NOP	DRM	31
JPNZY3	31	1	J	PCDT	NOP	NOP	1
JPNZN1	45	1	J	NOP	PCIN	NOP	46
JPNZN2	46	1	J	NOP	PCIN	NOP	1
ADD1	32	1	J	PLUS	ZALU	NOP	1
SUB1	36	1	J	MINU	ZALU	NOP	1
INAC1	40	1	J	ACIN	ZALU	NOP	1
CLAC1	44	1	J	ACCL	ZALU	NOP	1
AND1	48	1	J	AND	ZALU	NOP	1
OR1	52	1	J	OR	ZALU	NOP	1
XOR1	56	1	J	XOR	ZALU	NOP	1
NOT1	60	1	J	NOT	ZALU	NOP	1

19. The microsequencer is the same as shown in Figure 7.8, except the micro-operations fields outputs the control signals directly, as shown in the following table.

State	**Address**	**Condition**	**BT**	**Active control signals**	**ADDR**
FETCH1	1	1	J	ARLOAD, PCBUS	2
FETCH2	2	1	J	PCINC, DRLOAD, MEMBUS, READ	3
FETCH3	3	X	M	ARLOAD, IRLOAD, PCBUS	X
NOP1	0	1	J	None	1
LDAC1	4	1	J	ARINC, PCINC, DRLOAD, MEMBUS, READ	5
LDAC2	5	1	J	PCINC, DRLOAD, TRLOAD, MEMBUS, READ	6
LDAC3	6	1	J	ARLOAD, DRHBUS, TRBUS	7
LDAC4	7	1	J	DRLOAD, MEMBUS, READ	33
LDAC5	33	1	J	ACLOAD, ALUS3, DRLBUS	1
STAC1	8	1	J	ARINC, PCINC, DRLOAD, MEMBUS, READ	9
STAC2	9	1	J	PCINC, DRLOAD, TRLOAD, MEMBUS, READ	10
STAC3	10	1	J	ARLOAD, DRHBUS, TRBUS	11
STAC4	11	1	J	DRLOAD, ACBUS	34
STAC5	34	1	J	DRLBUS, BUSMEM, WRITE	1
MVAC1	12	1	J	RLOAD, ACBUS	1
MOVR1	16	1	J	ACLOAD, ALUS3, RBUS	1
JUMP1	20	1	J	ARINC, DRLOAD, MEMBUS, READ	21
JUMP2	21	1	J	DRLOAD, TRLOAD, MEMBUS, READ	22
JUMP3	22	1	J	PCLOAD, DRHBUS, TRBUS	1
JMPZ1	24	*Z¢*	J	None	41
JMPZY1	25	1	J	ARINC, DRLOAD, MEMBUS, READ	26
JMPZY2	26	1	J	DRLOAD, TRLOAD, MEMBUS, READ	27
JMPZY3	27	1	J	PCLOAD, DRHBUS, TRBUS	1
JMPZN1	41	1	J	PCINC	42
JMPZN2	42	1	J	PCINC	1
JPNZ1	28	*Z*	J	None	45
JPNZY1	29	1	J	ARINC, DRLOAD, MEMBUS, READ	30
JPNZY2	30	1	J	DRLOAD, TRLOAD, MEMBUS, READ	31
JPNZY3	31	1	J	PCLOAD, DRHBUS, TRBUS	1
JPNZN1	45	1	J	PCINC	46
JPNZN2	46	1	J	PCINC	1
ADD1	32	1	J	ACLOAD, ZALU, ALUS1, ALUS3, RBUS	1
SUB1	36	1	J	ACLOAD, ZALU, ALUS1, ALUS2, ALUS4, RBUS	1
INAC1	40	1	J	ACLOAD, ZALU, ALUS1, ALUS4	1
CLAC1	44	1	J	ACLOAD, ZALU	1
AND1	48	1	J	ACLOAD, ZALU, ALUS7, RBUS	1
OR1	52	1	J	ACLOAD, ZALU, ALUS6, ALUS7, RBUS	1
XOR1	56	1	J	ACLOAD, ZALU, ALUS5, ALUS7, RBUS	1
NOT1	60	1	J	ACLOAD, ZALU, ALUS5, ALUS6, ALUS7	1

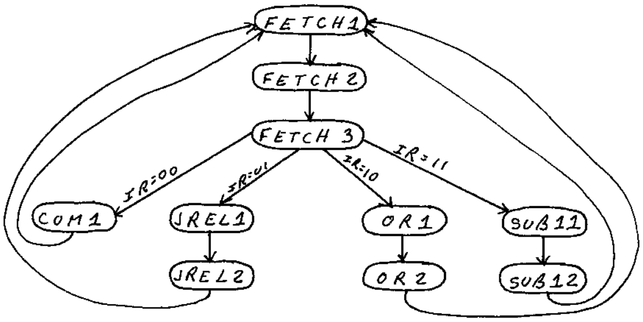
20. This solution is the same as for Problem 6.18. The student can execute the following program using the Relatively Simple CPU simulator to verify that each instruction performs properly.

```
0:  LDAC 0000     (AC ← 1)
    NOP
    MVAC          (R ← 1)
    ADD           (AC ← 2, Z ← 0)
    INAC          (AC ← 3, Z ← 0)
    XOR           (AC ← 2, Z ← 0)
    AND           (AC ← 0, Z ← 1)
9:  JMPZ 000D     (jump is taken)
    NOP           (skipped by JMPZ 000D)
D:  JPNZ 0009     (jump is not taken)
    NOT           (AC ← FF, Z ← 0)
    JMPZ 0009     (jump is not taken)
    JPNZ 0018     (jump is taken)
    NOP           (skipped by JMPZ 0018)
18: CLAC          (AC ← 0, Z ← 1)
    OR            (AC ← 1, Z ← 0)
    SUB           (AC ← 0, Z ← 1)
    MOVR          (AC ← 1)
    STAC 0030     (M[30] ← 1)
    AND           (AC ← 1, Z ← 0)
    JUMP 0000     (start again)
```

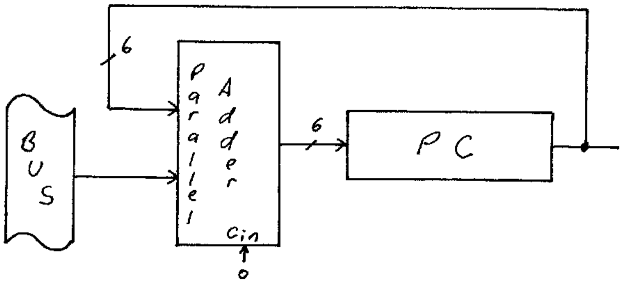
21. The state diagram, RTL code, register section, and ALU are the same as for Problem 6.28.

State diagram and RTL code:

FETCH1:	$AR \leftarrow PC$
FETCH2:	$DR \leftarrow M, PC \leftarrow PC + 1$
FETCH3:	$IR \leftarrow DR[7..6], AR \leftarrow DR[5..0]$
COM1:	$AC \leftarrow AC'$
JREL1:	$DR \leftarrow M$
JREL2:	$PC \leftarrow PC + DR[5..0]$
OR1:	$DR \leftarrow M$
OR2:	$AC \leftarrow AC \vee DR$
SUB11:	$DR \leftarrow M$
SUB12:	$AC \leftarrow AC + DR'$



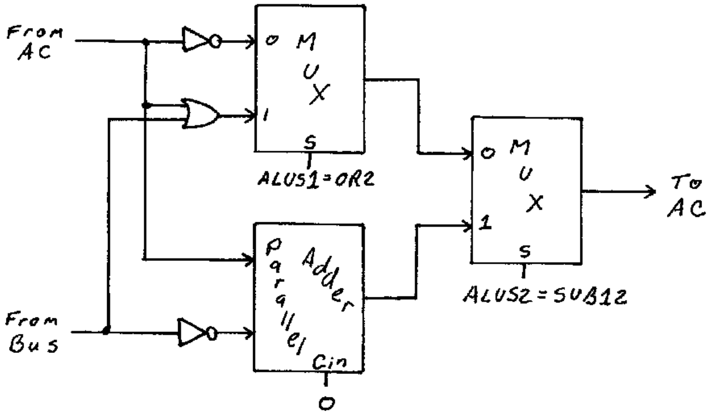
The register section is the same as Figure 6.6, except for the data input to *PC*, shown below.



Control signals:

$$\begin{aligned}
ARLOAD &= FETCH1 \vee FETCH3 \\
PCLOAD &= JREL2 \\
PCINC &= FETCH2 \\
PCBUS &= FETCH1 \\
DRLOAD = MEMBUS = READ &= FETCH2 \vee JREL1 \vee\ OR1 \vee\ SUB11 \\
DRBUS &= FETCH3 \vee JREL2 \vee OR2 \vee SUB12 \\
ACLOAD &= COM1 \vee OR2 \vee SUB12 \\
IRLOAD &= FETCH3
\end{aligned}$$

ALU:



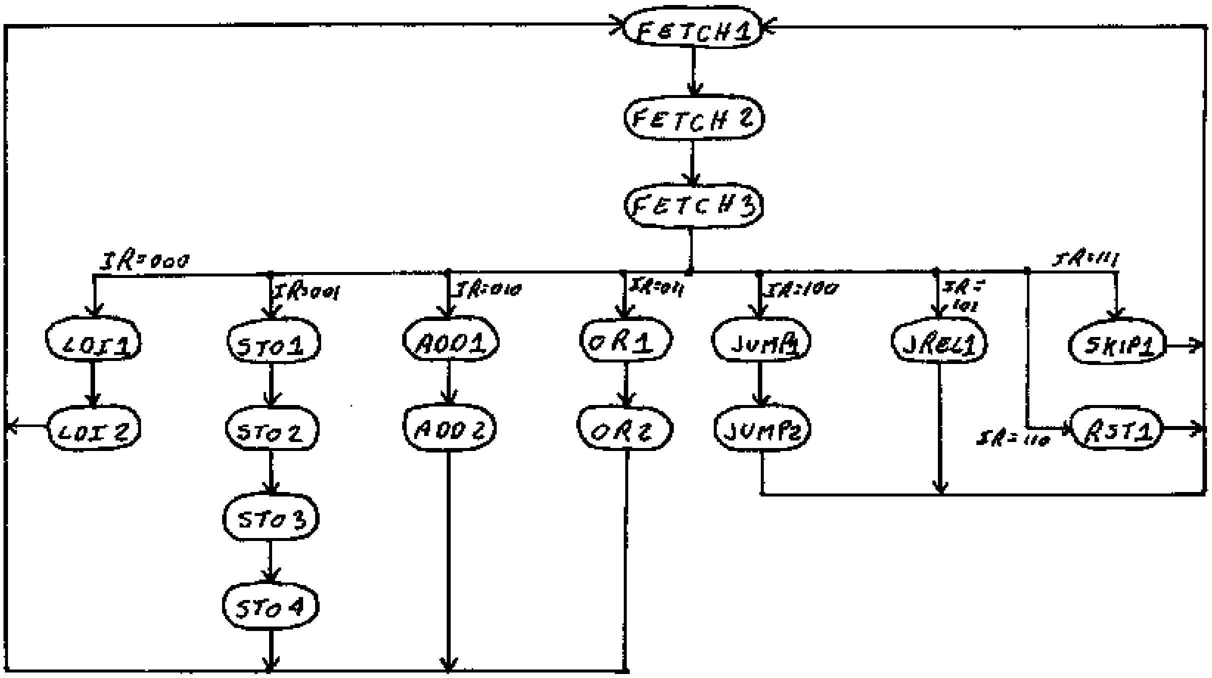
The microsequencer hardware is the same as shown in Figures 7.3 and 7.4, except the micro-operations are replaced by control signals.

Microcode:

State	Address	SEL	ARLOAD	PCLOAD	PCINC	PCBUS	DMR	DRBUS	ACLOAD	IRLOAD	ALUS1	ALUS2	ADDR
FETCH1	0000 (0)	0	1	0	0	1	0	0	0	0	0	0	0001
FETCH2	0001 (1)	0	0	0	1	0	1	0	0	0	0	0	0010
FETCH3	0010 (2)	1	1	0	0	0	0	1	0	1	0	0	XXXX
COM1	1000 (8)	0	0	0	0	0	0	0	1	0	0	0	0000
JREL1	1010 (10)	0	0	0	0	0	1	0	0	0	0	0	1011
JREL2	1011 (11)	0	0	1	0	0	0	1	0	0	0	0	0000
OR1	1100 (12)	0	0	0	0	0	1	0	0	0	0	0	1101
OR2	1101 (13)	0	0	0	0	0	0	1	1	0	1	0	0000
SUB11	1110 (14)	0	0	0	0	0	1	0	0	0	0	0	1111
SUB12	1111 (15)	0	0	0	0	0	0	1	1	0	0	1	0000

22. The state diagram, RTL code, register section, and ALU are the same as for Problem 6.29.

State diagram and RTL code:

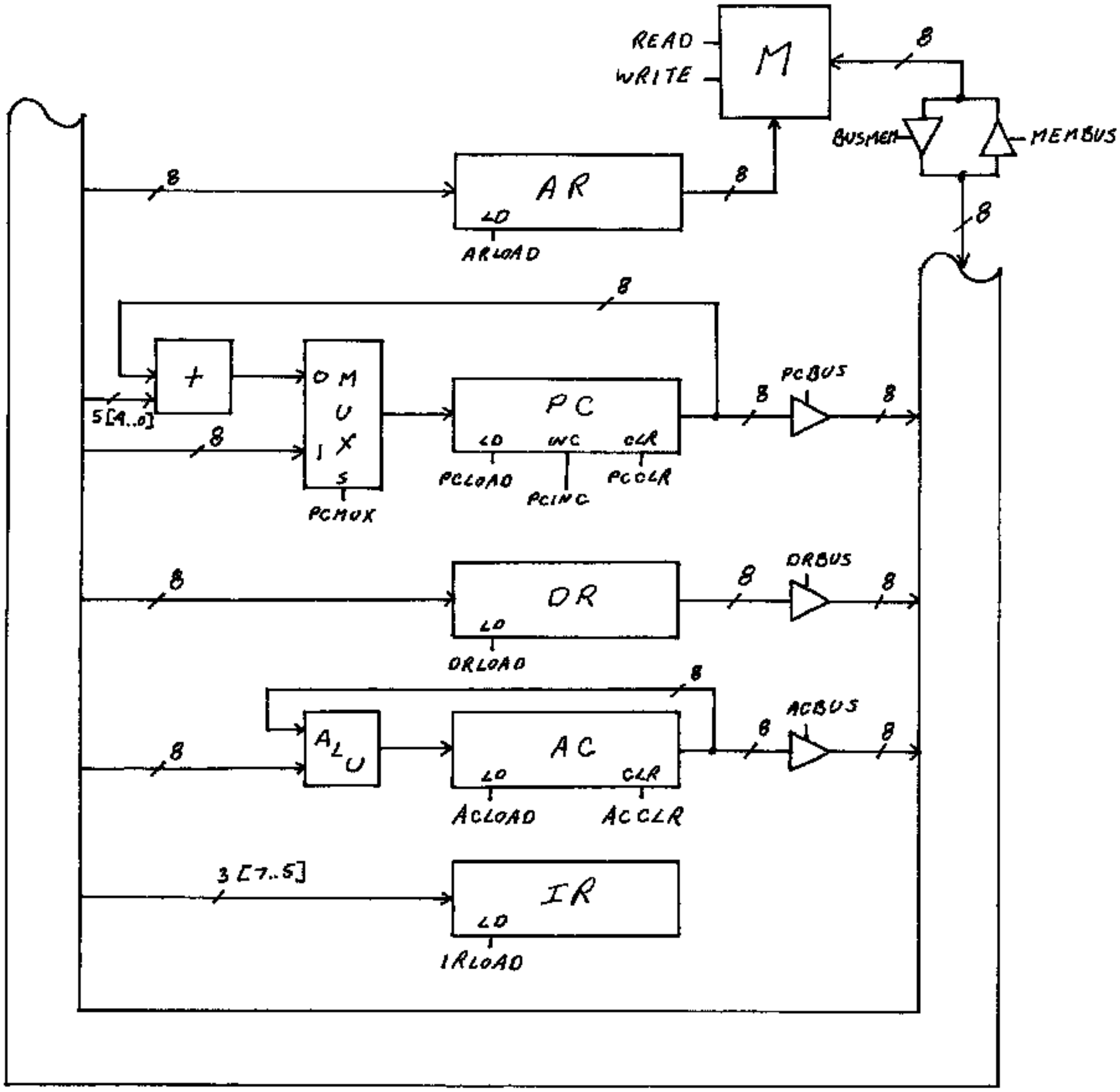


FETCH1:	$AR \leftarrow PC$	ADD1:	$DR \leftarrow M, PC \leftarrow PC + 1$
FETCH2:	$DR \leftarrow M, PC \leftarrow PC + 1$	ADD2:	$AC \leftarrow AC + DR$
FETCH3:	$IR \leftarrow DR[7..5], AR \leftarrow PC$	OR1:	$DR \leftarrow M, PC \leftarrow PC + 1$
LDI1:	$DR \leftarrow M, PC \leftarrow PC + 1$	OR2:	$AC \leftarrow AC \vee DR$
LDI2:	$AC \leftarrow DR$	JUMP1:	$DR \leftarrow M$
STO1:	$DR \leftarrow M, PC \leftarrow PC + 1$	JUMP2:	$PC \leftarrow DR$
STO2:	$AR \leftarrow DR$	JREL1:	$PC \leftarrow PC + 000DR[4..0]$
STO3:	$DR \leftarrow AC$	SKIP1:	$PC \leftarrow PC + 1$
STO4:	$M \leftarrow DR$	RST1:	$PC \leftarrow 0, AC \leftarrow 0$

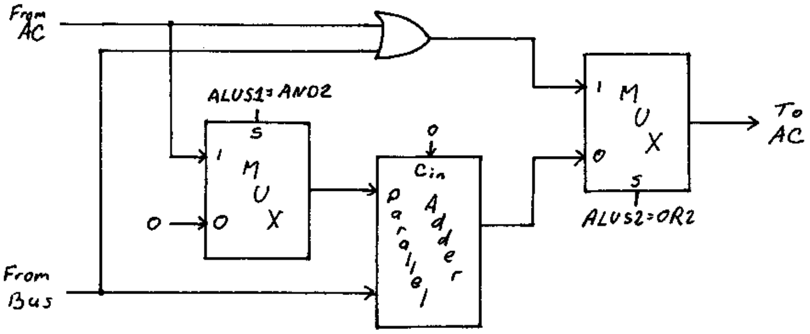
Control signals*:*

$ARLOAD = FETCH1 \vee FETCH3 \vee STO2$
$PCLOAD = JUMP2 \vee JREL2$
$PCINC = FETCH2 \vee LDI1 \vee STO1 \vee ADD1 \vee OR1 \vee SKIP1$
$PCCLR = RST1$
$PCBUS = FETCH1 \vee FETCH3$
$PCMUX = JUMP2$
$DRLOAD = FETCH2 \vee LDI1 \vee STO1 \vee STO3 \vee ADD1 \vee OR1 \vee JUMP1$
$DRBUS = FETCH3 \vee LDI2 \vee STO2 \vee STO4 \vee ADD2 \vee OR2 \vee JUMP2 \vee JREL1$
$ACLOAD = LDI2 \vee ADD2 \vee OR2$
$ACCLR = RST1$
$ACBUS = STO3$
$IRLOAD = FETCH3$
$MEMBUS = READ = FETCH2 \vee LDI1 \vee STO1 \vee ADD1 \vee OR1 \vee JUMP1$
$BUSMEM = WRITE = STO4$

Register section:



ALU:



The microsequencer hardware is the same as shown in Figures 7.3 and 7.4, except the micro-operations are output to decoders to generate the actual micro-operation signals, and the mapping function is 1,*IR*[2..0],0.

Micro-operation field assignments:

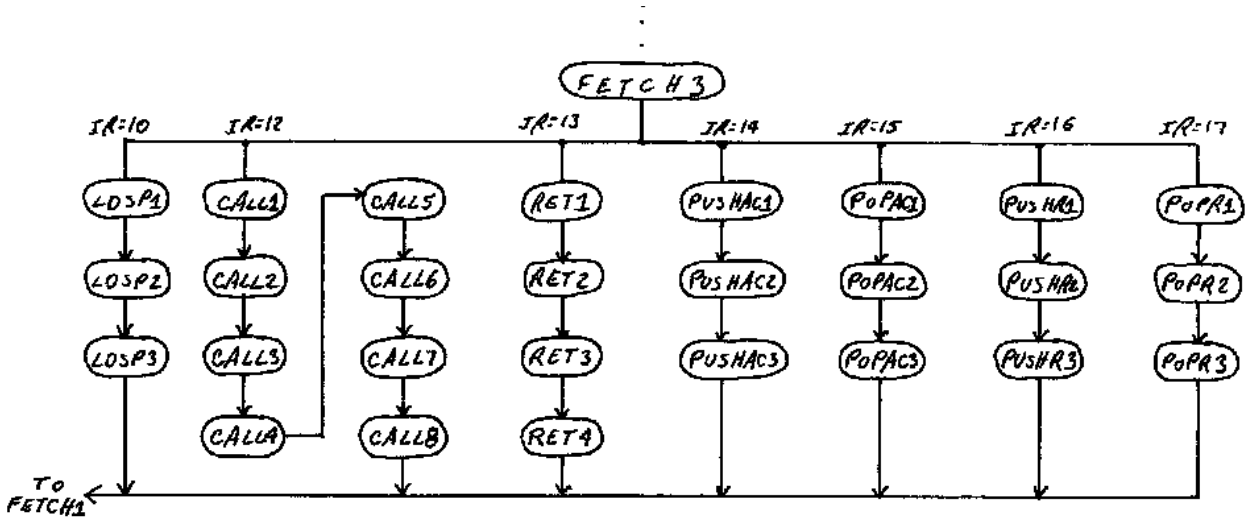
	M1		M2
0	NOP	0	NOP
1	$AR \leftarrow PC$	1	$IR \leftarrow DR[7..5]$
2	$DR \leftarrow M$	2	$PC \leftarrow PC + 1$
3	$AC \leftarrow DR$	3	$PC \leftarrow 0$
4	$AR \leftarrow DR$	4	$PC \leftarrow DR$
5	$DR \leftarrow AC$	5	$PC \leftarrow PC + 000DR[4..0]$
6	$M \leftarrow DR$	6	$AC \leftarrow \mathrm{AC} + DR$
7	$AC \leftarrow 0$	7	$AC \leftarrow AC \vee DR$

Microcode:

State	**Address**	**SEL**	**M1**	**M2**	**ADDR**
FETCH1	00000	0	001	000	00001
FETCH2	00001	0	101	010	00010
FETCH3	00010	1	001	001	XXXX
STO3	00100	0	101	000	00101
STO4	00101	0	110	000	00000
LDI1	10000	0	010	010	10001
LDI2	10001	0	011	000	00000
STO1	10010	0	010	010	10011
STO2	10011	0	100	000	00100
ADD1	10100	0	010	010	10101
ADD2	10101	0	000	110	00000
OR1	10110	0	010	010	10111
OR2	10111	0	000	111	00000
JUMP1	11000	0	010	000	11001
JUMP2	11001	0	000	100	00000
JREL1	11010	0	000	101	00000
SKIP1	11100	0	000	010	00000
RST1	11110	0	111	011	00000

23. The state diagram, RTL code, and register section are the same as in Problem 6.30.

Modified state diagram:



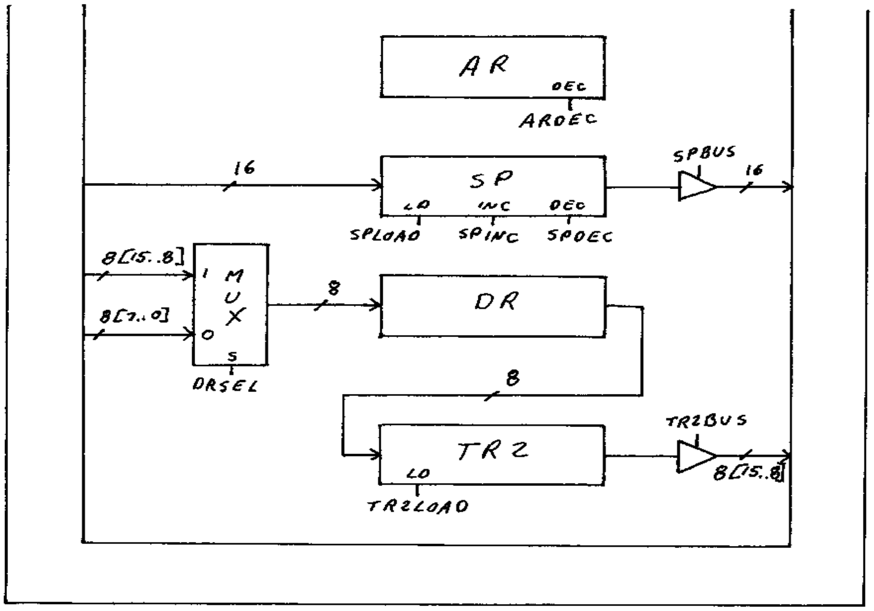
Modified RTL code:

LDSP1: $DR \leftarrow M, AR \leftarrow AR + 1, PC \leftarrow PC + 1$
LDSP2: $TR \leftarrow DR, DR \leftarrow M, PC \leftarrow PC + 1$
LDSP3: $SP \leftarrow DR,TR$
CALL1: $DR \leftarrow M, AR \leftarrow AR + 1, PC \leftarrow PC + 1$
CALL2: $TR \leftarrow DR, DR \leftarrow M, PC \leftarrow PC + 1$
CALL3: $TR2 \leftarrow DR, DR \leftarrow PC[15..8], SP \leftarrow SP - 1$
CALL4: $AR \leftarrow SP$
CALL5: $M \leftarrow DR, AR \leftarrow AR - 1, SP \leftarrow SP - 1$
CALL6: $DR \leftarrow PC[7..0]$
CALL7: $M \leftarrow DR$
CALL8: $PC \leftarrow TR2,DR$
RET1: $AR \leftarrow SP$
RET2: $DR \leftarrow M, SP \leftarrow SP + 1, AR \leftarrow AR + 1$
RET3: $TR \leftarrow DR, DR \leftarrow M, SP \leftarrow SP + 1$
RET4: $PC \leftarrow DR,TR$

PUSHAC1: $SP \leftarrow SP - 1, DR \leftarrow AC$
PUSHAC2: $AR \leftarrow SP$
PUSHAC3: $M \leftarrow DR$
POPAC1: $AR \leftarrow SP$
POPAC2: $DR \leftarrow M, SP \leftarrow SP + 1$
POPAC3: $AC \leftarrow DR$
PUSHR1: $SP \leftarrow SP - 1, DR \leftarrow R$
PUSHR2: $AR \leftarrow SP$
PUSHR3: $M \leftarrow DR$
POPR1: $AR \leftarrow SP$
POPR2: $DR \leftarrow M, SP \leftarrow SP + 1$
POPR3: $R \leftarrow DR$

Modified register section: (shown below)

- New registers: *SP* (with *LD*, *DEC*, *INC*), *TR2* (with *LD*, receives data directly from *DR*)
- New control signal: *AR* adds a *DEC* signal
- New data path: *DR* can receive data from *BUS*[15..8] or *BUS*[7..0]
- All other connections remain the same as shown in Figure 6.15.



New micro-operations:

SPDT:	*SP* ← *DR,TR*	DRPL:	*DR* ← *PC*[7..0]
T2DR:	*TR2* ← *DR*	PCTR:	*PC* ← *TR2,TR*
DRPH:	*DR* ← *PC*[15..8]	SPIN:	*SP* ← *SP* + 1
SPDC:	*SP* ← *SP* - 1	DRR:	*DR* ← *R*
ARSP:	*AR* ← *SP*	RDR:	*R* ← *DR*
ARDC:	*AR* ← *AR* - 1		

New mapping function: *IR*[7,3..0]00 (The extra MSB in ADDR is 0 for all current instructions.)

ARDEC =	ARDC
SPLOAD =	SPDT
SPINC =	SPIN
SPDEC =	SPDC
SPBUS =	ARSP
DRSEL =	DRPH
TR2LOAD =	T2DR
TR2BUS =	PCTR

Modified control signals:

ARLOAD =	(original value) ∨ ARSP
ARINC =	(original value) ∨ PCTR
PCLOAD =	(original value) ∨ DRPH ∨ DRPL ∨ DRR
DRLOAD =	(original value) ∨ DRPH ∨ DRPL ∨ DRR
DRHBUS =	(original value) ∨ SPDT
DRLBUS =	(original value) ∨ T2DR ∨ RDR
TRLOAD =	(original value) ∨ LDSP2 ∨ CALL2 ∨ RET3
TRBUS =	(original value) ∨ PCTR
RLOAD =	(original value) ∨ RDR
RBUS =	(original value) ∨ DRR

New microinstructions:

State	**Address**	**SEL**	**Micro-operations**	**ADDR**
LDSP1	100 0000	0	DRM, ARIN, PCIN	100 0001
LDSP2	100 0001	0	TRDR, DRM, PCIN	100 0010
LDSP3	100 0010	0	SPDT	000 0001
CALL1	100 1000	0	DRM, ARIN, PCIN	100 1001
CALL2	100 1001	0	TRDR, DRM, PCIN	100 1010
CALL3	100 1010	0	T2DR, DRPH, SPDC	100 1011
CALL4	100 1011	0	ARSP	100 0100
CALL5	100 0100	0	MDR, ARDC, SPDC	100 0101
CALL6	100 0101	0	DRPL	100 0110
CALL7	100 0110	0	MDR	100 0111
CALL8	100 0111	0	PCTR	000 0001
RET1	100 1100	0	ARSP	100 1101
RET2	100 1101	0	DRM, SPIN, ARIN	100 1110
RET3	100 1110	0	TRDR, DRM, SPIN	100 1111
RET4	100 1111	0	PCDT	000 0001
PUSHAC1	101 0000	0	SPDC, DRAC	101 0001
PUSHAC2	101 0001	0	ARSP	101 0010
PUSHAC3	101 0010	0	MDR	000 0001
POPAC1	101 0100	0	ARSP	101 0101
POPAC2	101 0101	0	DRM, SPIN	101 0110
POPAC3	101 0110	0	ACDR	000 0001
PUSHR1	101 1000	0	SPDC, DRR	101 1001
PUSHR2	101 1001	0	ARSP	101 1010
PUSHR3	101 1010	0	MDR	000 0001
POPR1	101 1100	0	ARSP	101 1101
POPR2	101 1101	0	DRM, SPIN	101 1110
POPR3	101 1110	0	RDR	000 0001

Chapter 8

1. a) 64 = 01000000 64' = 1100 0000
   b) 33 = 0010 0001 33' = 1101 1111
   c) -1 = 1111 1111 -1' = 0000 0001

2.

		Non-negative	Unsigned two's-complement
a)	29 =	0001 1101	0001 1101
b)	-128 =	N/A	1000 0000
c)	199 =	1100 0111	N/A

3.

		Signed-Magnitude	Signed-Two's Complement
a)	-63 =	1011 1111	1100 0001
b)	147 =	N/A	N/A
c)	85 =	0101 0101	0101 0101

4. a) 0011 1101 (180 - 119 = 61)
   b) N/A (56 + 205 = 261, overflow)
   c) 1111 1111 (139 + 116 = 255)
   d) N/A (116 - 139 = -23, negative number)

5. a) N/A (-76 - 119 = -193, overflow)
   b) 0000 0101 (56 + -51 = 5)
   c) 1111 1111 (-117 + 116 = -1)
   d) N/A (116 - -117 = 233, overflow)

6. The worst cases are +127 + (-1) = +126 and +0 + (-128) = -128, both of which produce valid results.

7.

Conditions	*Micro-operations*	*i*	*C*	*U*	*V*	*Y*	*Z*	FINISH
START		x	x	xxxx	xxxx	1110		0
1	$U \leftarrow 0$, $i \leftarrow 4$	4		0000			0	
2	$i \leftarrow i$ - 1	3					0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0000	0xxx	0111		
Y_0***2**,**2***	$CU \leftarrow U + X$, $i \leftarrow i$ - 1	2	0	1001			0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0100	10xx	1011		
Y_0***2**,**2***	$CU \leftarrow U + X$, $i \leftarrow i$ - 1	1	0	1101			0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0110	110x	1101		
Y_0***2**,**2***	$CU \leftarrow U + X$, $i \leftarrow i$ - 1	0	0	1111			1	
3**,Z**3	shr(*CUV*), cir(*Y*), *FINISH* ← 1		0	0111	1110	1110		1

Result: 9 * 14 = 126, or 1001 * 1110 = 0111 1110

8.

Conditions	*Micro-operations*	*i*	*C*	*U*	*V*	*Z*	FINISH
START		x	x	xxxx	1110		0
$\mathbf{1}$	$U \leftarrow 0, i \leftarrow 4$	4		0000		0	
$\mathbf{2}$	$i \leftarrow i - 1$	3				0	
$\mathbf{3},Z'\mathbf{3}$	shr(CUV), goto $\mathbf{2}$		0	0000	0111		
$V_0\mathbf{2},\mathbf{2}$	$CU \leftarrow U + X, i \leftarrow i - 1$	2	0	1001		0	
$\mathbf{3},Z'\mathbf{3}$	shr(CUV), goto $\mathbf{2}$		0	0100	1011		
$V_0\mathbf{2},\mathbf{2}$	$CU \leftarrow U + X, i \leftarrow i - 1$	1	0	1101		0	
$\mathbf{3},Z'\mathbf{3}$	shr(CUV), goto $\mathbf{2}$		0	0110	1101		
$V_0\mathbf{2},\mathbf{2}$	$CU \leftarrow U + X, i \leftarrow i - 1$	0	0	1111		1	
$\mathbf{3},Z\mathbf{3}$	shr(CUV), $FINISH \leftarrow 1$		0	0111	1110		1

Result: 9 * 14 = 126, or 1001 * 1110 = 0111 1110

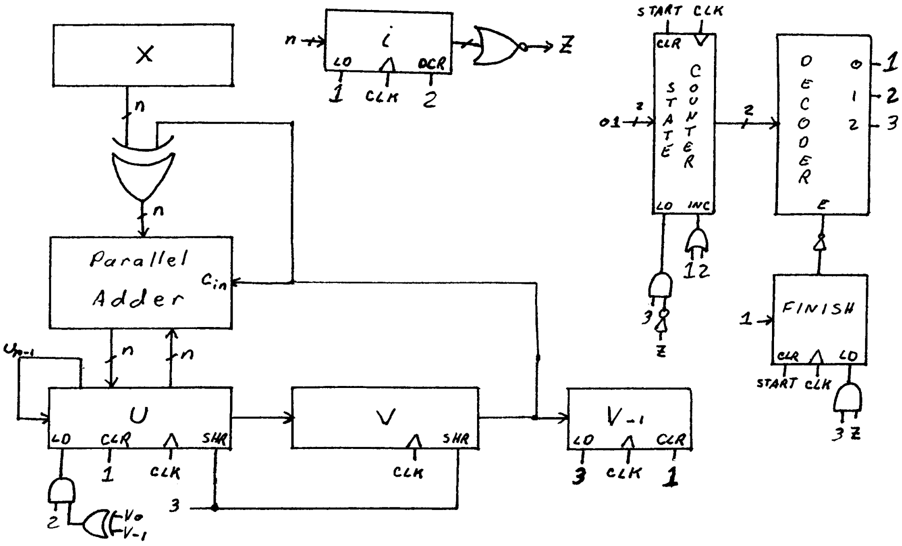
9.

Conditions	*Micro-operations*	*i*	*U*	*V*	*Y*	Y_{-1}	*Z*	FINISH
START		x	xxxx	xxxx	1011	x		0
$\mathbf{1}$	$U \leftarrow 0, Y_{-1} \leftarrow 0, i \leftarrow 4$	4	0000			0	0	
$Y_0Y_{-1}'\mathbf{2},\mathbf{2}$	$U \leftarrow U + X' + 1, i \leftarrow i - 1$	3	1010				0	
$\mathbf{3},Z'\mathbf{3}$	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$, goto $\mathbf{2}$		1101	0xxx	1101	1		
$\mathbf{2}$	$i \leftarrow i - 1$	2					0	
$\mathbf{3},Z'\mathbf{3}$	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$, goto $\mathbf{2}$		1110	10xx	1110	1		
$Y_0'Y_{-1}\mathbf{2},\mathbf{2}$	$U \leftarrow U + X, i \leftarrow i - 1$	1	0100				0	
$\mathbf{3},Z'\mathbf{3}$	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$, goto $\mathbf{2}$		0010	010x	0111	0		
$Y_0Y_{-1}'\mathbf{2},\mathbf{2}$	$U \leftarrow U + X' + 1, i \leftarrow i - 1$	0	1100				1	
$\mathbf{3},Z\mathbf{3}$	ashr(UV), cir(Y), $Y_{-1} \leftarrow Y_0$, $FINISH \leftarrow 1$		1110	0010	1011			1

Result: 6 * -5 = -30, or 0110 * 1011 = 1110 0010

10.

$\mathbf{1}$: $U \leftarrow 0, V_{-1} \leftarrow 0, i \leftarrow n$
$V_0V'_{-1}\mathbf{2}$: $U \leftarrow U + X' + 1$
$V'_0V_{-1}\mathbf{2}$: $U \leftarrow U + X$
$\mathbf{2}$: $i \leftarrow i - 1$
$\mathbf{3}$: ashr(UVV_{-1})
$Z'\mathbf{3}$: goto $\mathbf{2}$
$Z\mathbf{3}$: $FINISH \leftarrow 1$



11.

Conditions	*Micro-operations*	*i*	*C*	*U*	*V*	*Y*	*Z*	OVERFLOW	FINISH
START		x	x	0110	1011	xxxx			0
$\mathbf{1}_1$	$CU \leftarrow U + X\text{¢} + 1$		0	1100					
$\mathbf{1}_2$	$U \leftarrow U + X$			0110					
$\mathbf{2}$	$Y \leftarrow 0$, $OVERFLOW \leftarrow 0$, $i \leftarrow 4$	4				0000	0	0	
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	3	0	1101	0110	0000	0		
$C'\mathbf{4}_1$	$CU \leftarrow U + X\text{¢} + 1$		1	0011					
$C\mathbf{4}_2, Z'\mathbf{4}_2$	$Y_0 \leftarrow 1$, goto $\mathbf{3}$					0001			
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	2	0	0110	1100	0010	0		
$C'\mathbf{4}_1$	$CU \leftarrow U + X\text{¢} + 1$		0	1100					
$C'\mathbf{4}_2, Z'\mathbf{4}_2$	$U \leftarrow U + X$, goto $\mathbf{3}$		0	0110					
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	1	0	1101	1000	0100	0		
$C'\mathbf{4}_1$	$CU \leftarrow U + X\text{¢} + 1$		1	0011					
$C\mathbf{4}_2, Z'\mathbf{4}_2$	$Y_0 \leftarrow 1$, goto $\mathbf{3}$					0101			
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	0	0	0111	0000	1010	1		
$C'\mathbf{4}_1$	$CU \leftarrow U + X\text{¢} + 1$		0	1101					
$C'\mathbf{4}_2, Z\mathbf{4}_2$	$U \leftarrow U + X$, $FINISH \leftarrow 1$			0111					1

Result: 107 ÷ 10 = 10 R 7, or 0110 1011 ÷ 1010 = 1010 R 0111

12.

Conditions	*Micro-operations*	*i*	*C*	*U*	*V*	*Y*	*Z*	FINISH
START		x	x	0110	1011	xxxx		0
$\mathbf{1}$	*NONE*							
$\mathbf{2}$	$Y \leftarrow 0$, $C \leftarrow 0$, $OVERFLOW \leftarrow 0$, $i \leftarrow 4$	4	0			0000	0	
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	3	0	1101	0110	0000	0	
$(C + G)\mathbf{4}, Z'\mathbf{4}$	$Y_0 \leftarrow 1$, $U \leftarrow U + X\text{¢} + 1$, goto $\mathbf{3}$			0011		0001		
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	2	0	0110	1100	0010	0	
$Z'\mathbf{4}$	goto $\mathbf{3}$							
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	1	0	1101	1000	0100	0	
$(C + G)\mathbf{4}, Z'\mathbf{4}$	$Y_0 \leftarrow 1$, $U \leftarrow U + X\text{¢} + 1$, goto $\mathbf{3}$			0011		0101		
$\mathbf{3}$	shl CUV, shl Y, $i \leftarrow i - 1$	0	0	0111	0000	1010	1	
$Z\mathbf{4}$	$FINISH \leftarrow 1$							1

Result: 107 ÷ 10 = 10 R 7, or 0110 1011 ÷ 1010 = 1010 R 0111

13.

$\mathbf{1}$: $OVERFLOW \leftarrow G$

$G\mathbf{1}$: $FINISH \leftarrow 1$

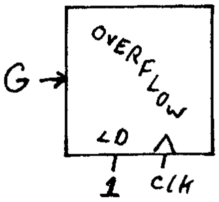
$\mathbf{2}$: $Y \leftarrow 0$, $C \leftarrow 0$, $i \leftarrow n$

$\mathbf{3}$: shl CUV, shl Y, $i \leftarrow i - 1$

$(C + G)\mathbf{4}$: $Y_0 \leftarrow 1$, $U \leftarrow U + X\text{¢} + 1$

$Z'\mathbf{4}$: goto 3

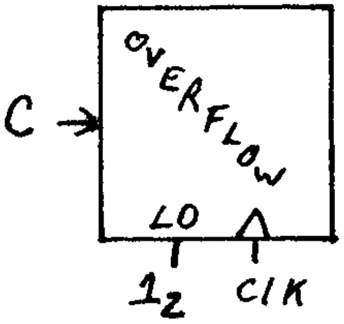
$Z\ \mathbf{4}$: $FINISH \leftarrow 1$



(Only **1**, *G***1**, and **2**, and the OVERFLOW hardware are changed; the rest is the same as in the chapter.)

14.

```
      $\mathbf{1}_1$: $CU \leftarrow U + X¢ + 1$
      $\mathbf{1}_2$: $U \leftarrow U + X$, $OVERFLOW$ ¬ $C$
     $C\mathbf{1}_2$: $FINISH \leftarrow 1$
         $\mathbf{2}$: $Y \leftarrow 0$, $i \leftarrow n$
         $\mathbf{3}$: shl $CUV$, shl $Y$, $i \leftarrow i - 1$
     $C\mathbf{4}_1$: $U \leftarrow U + X¢ + 1$
    $C´\mathbf{4}_1$: $CU \leftarrow U + X¢ + 1$
     $C\mathbf{4}_2$: $Y_0 \leftarrow 1$
    $C´\mathbf{4}_2$: $U \leftarrow U + X$
     $Z\ \mathbf{4}_2$: $FINISH \leftarrow 1$
    $Z´\mathbf{4}_2$: goto $\mathbf{3}$
```



(Only $\mathbf{1}_2$, and $\mathbf{2}$, and the OVERFLOW hardware are changed; the rest is the same as in the chapter.)

15.

```
              $\mathbf{1}_0$: $NU \leftarrow U_{n-1}$, $NX \leftarrow X_{n-1}$
      $U_{n-1}\mathbf{1}_0$: $UV \leftarrow (UV)' + 1$
      $X_{n-1}\mathbf{1}_0$: $X \leftarrow X' + 1$
             $G\mathbf{1}$: $FINISH \leftarrow 1$, $OVERFLOW \leftarrow 1$
           $NUG\mathbf{1}$: $UV \leftarrow (UV)' + 1$
           $NXG\mathbf{1}$: $X \leftarrow X' + 1$
                $\mathbf{2}$: $Y \leftarrow 0$, $C \leftarrow 0$, $OVERFLOW \leftarrow 0$, $i \leftarrow n$
                $\mathbf{3}$: shl $CUV$, shl $Y$, $i \leftarrow i - 1$
       $(C + G)\mathbf{4}$: $Y_0 \leftarrow 1$, $U \leftarrow CU + X' + 1$
            $Z´\mathbf{4}$: goto $\mathbf{3}$
  $(NU \oplus NX)\mathbf{5}$: $Y \leftarrow Y' + 1$, $U \leftarrow U' + 1$
           $NX\mathbf{5}$: $X \leftarrow X' + 1$
                $\mathbf{5}$: $FINISH \leftarrow 1$
```

16.

```
              $\mathbf{1}_0$: $NU \leftarrow U_{n-1}$, $NX \leftarrow X_{n-1}$
      $U_{n-1}\mathbf{1}_0$: $UV \leftarrow (UV)' + 1$
      $X_{n-1}\mathbf{1}_0$: $X \leftarrow X' + 1$
              $\mathbf{1}_1$: $CU \leftarrow U + X¢ + 1$
              $\mathbf{1}_2$: $U \leftarrow U + X$
            $C\mathbf{1}_2$: $FINISH \leftarrow 1$, $OVERFLOW \leftarrow 1$
          $NUC\mathbf{1}_2$: $UV \leftarrow (UV)' + 1$
          $NXC\mathbf{1}_2$: $X \leftarrow X' + 1$
                $\mathbf{2}$: $Y \leftarrow 0$, $C \leftarrow 0$, $OVERFLOW \leftarrow 0$, $i \leftarrow n$
                $\mathbf{3}$: shl $CUV$, shl $Y$, $i \leftarrow i - 1$
            $C\mathbf{4}_1$: $U \leftarrow U + X¢ + 1$
           $C´\mathbf{4}_1$: $CU \leftarrow U + X¢ + 1$
            $C\mathbf{4}_2$: $Y_0 \leftarrow 1$
           $C´\mathbf{4}_2$: $U \leftarrow U + X$
           $Z´\mathbf{4}_2$: goto $\mathbf{3}$
  $(NU \oplus NX)\mathbf{5}$: $Y \leftarrow Y' + 1$, $U \leftarrow U' + 1$
           $NX\mathbf{5}$: $X \leftarrow X' + 1$
                $\mathbf{5}$: $FINISH \leftarrow 1$
```

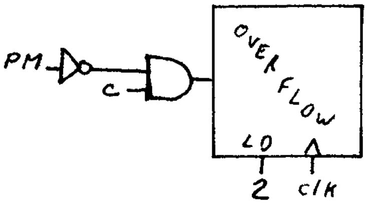
17. a) *PM**1***: *CU* ← 0 1110, *OVERFLOW* ← 0
*C´PM**2**,**2***: U_s ← 1, U ← 0010, *FINISH* ← 1
Result: U_sU = 1 0010 = -2

b) *PM´**1***: U_s ← 0, *CU* ← 1 0000
*PM´**2**,**2***: *OVERFLOW* ← 1, *FINISH* ← 1
Result: Overflow

c) *PM**1***: CU ← 0 1110, *OVERFLOW* ← 0
*C´PM**2**,**2***: U_s ← 0, U ← 0010, *FINISH* ← 1
Result: U_sU = 0 0010 = +2

d) *PM´**1***: U_s ← 0, *CU* ← 1 0000
*PM´**2**,**2***: *OVERFLOW* ← 1, *FINISH* ← 1
Result: Overflow

18.
*PM´**1***: U_s ← X_s, *CU* ← *X* + *Y*
*PM**1***: *CU* ← *X* + *Y*¢ + 1, *OVERFLOW* ← 0
*CZ´PM**2***: U_s ← X_s
*CZPM**2***: U_s ← 0
*C´PM**2***: U_s ← X_s¢, *U* ← *U*¢ + 1
2: *OVERFLOW* ← *PM'* ^ *C*, *FINISH* ← 1



(Only *PM**1***, *PM'**2*** (deleted), and ***2***, and the OVERFLOW hardware are changed; the rest is the same as in the chapter.)

19. a)

Conditions	*Micro-operations*	*i*	*C*	*U*	*V*	*Y*	*Z*	FINISH
START		x	x	xxxx	xxxx	1001		0
1	U_s ← 1, V_s ← 1, *U* ← 0, *i* ← 4	4		0000			0	
Y_0***2**,**2***	*CU* ← *U* + *X*, *i* ← *i* - 1	3	0	0111			0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0011	1xxx	1100		
2	*i* ← *i* - 1	2					0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0001	11xx	0110		
2	*i* ← *i* - 1	1					0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0000	111x	0011		
Y_0***2**,**2***	*CU* ← *U* + *X*, *i* ← *i* - 1	0	0	0111			1	
3**,Z**3	shr(*CUV*), cir(*Y*), *FINISH* ← 1		0	0011	1111	1001		1

Result: +7 × -9 = -63, or 0 0111 × 1 1001 = 1 0011 1111

b)

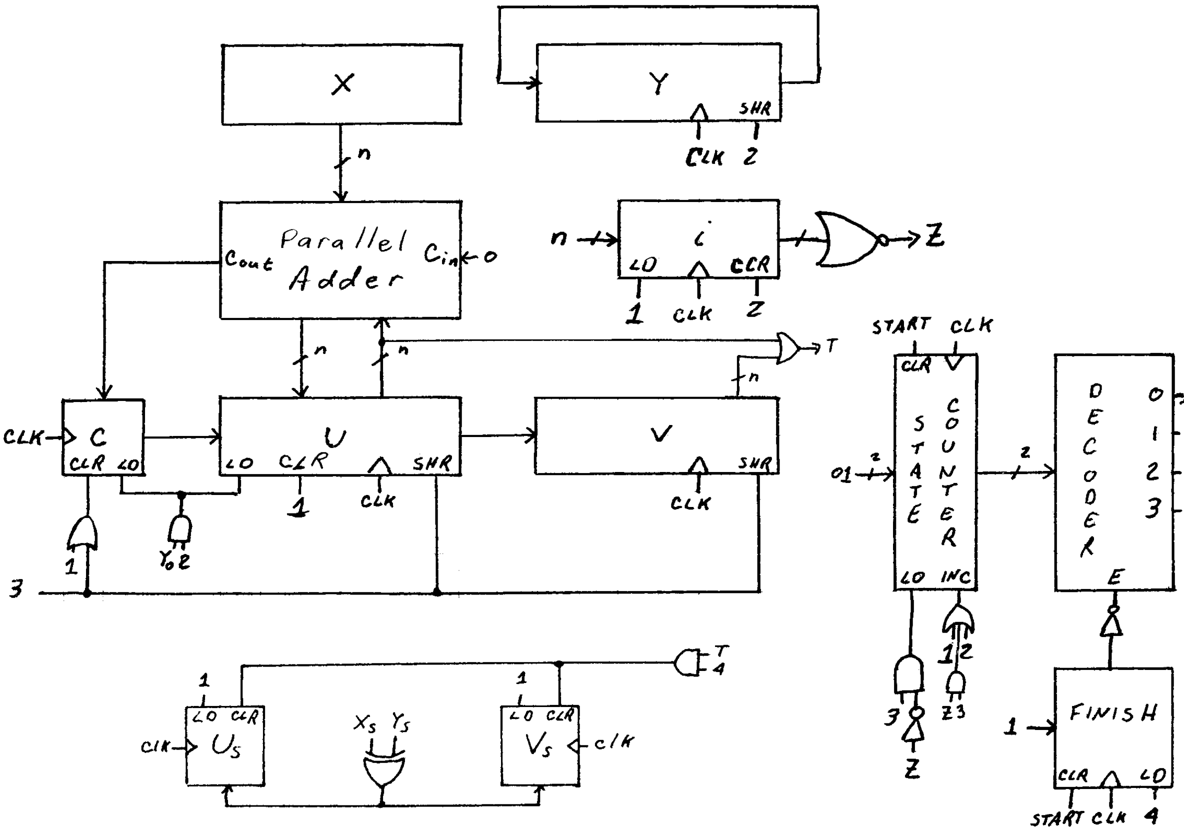
Conditions	*Micro-operations*	*i*	*C*	*U*	*V*	*Y*	*Z*	FINISH
START		x	x	xxxx	xxxx	0000		0
1	U_s ← 1, V_s ← 1, *U* ← 0, *i* ← 4	4		0000			0	
2	*i* ← *i* - 1	3					0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0000	0xxx	0000		
2	*i* ← *i* - 1	2					0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0000	00xx	0000		
2	*i* ← *i* - 1	1					0	
3**,Z´**3	shr(*CUV*), cir(*Y*), goto ***2***		0	0000	000x	0000		
2	*i* ← *i* - 1	0	0	0000			1	
*ZT**3**,**3**,Z**3***	U_s ← 0, V_s ← 0, shr(*CUV*), cir(*Y*), *FINISH* ← 1		0	0000	0000	0000		1

Result: -13 × +0 = +0, or 1 1101 × 0 0000 = 0 0000 0000 c)

Conditions	*Micro-operations*	*i*	*C*	*U*	*V*	*Y*	*Z*	FINISH
START		x	x	xxxx	xxxx	1111		0
1	$U_s \leftarrow 0$, $V_s \leftarrow 0$, $U \leftarrow 0$, $i \leftarrow 4$	4		0000			0	
Y_0***2***,***2***	$CU \leftarrow U + X$, $i \leftarrow i - 1$	3	0	1111			0	
3,Z'***3***	shr(*CUV*), cir(*Y*), goto ***2***		0	0111	1xxx	1111		
Y_0***2***,***2***	$CU \leftarrow U + X$, $i \leftarrow i - 1$	2	1	0110			0	
3,Z'***3***	shr(*CUV*), cir(*Y*), goto ***2***		0	1011	01xx	1111		
Y_0***2***,***2***	$CU \leftarrow U + X$, $i \leftarrow i - 1$	1	1	1010			0	
3,Z'***3***	shr(*CUV*), cir(*Y*), goto ***2***		0	1101	001x	1111		
Y_0***2***,***2***	$CU \leftarrow U + X$, $i \leftarrow i - 1$	0	1	1100			1	
3,*Z****3***	shr(*CUV*), cir(*Y*), *FINISH* ← 1		0	1110	0001	1111		1

Result: +15 × +15 = +225, or 0 1111 × 0 1111 = 0 1110 0001

20.



21. Add the following RTL statement. The rest of the algorithm is unchanged.

$G'\mathbf{1}$: $U_s \leftarrow U_s \oplus X_s$, $Y_s \leftarrow U_s \oplus X_s$

22. Add the following RTL statement. The rest of the algorithm is unchanged.

$C'\mathbf{1}_2$: $U_s \leftarrow U_s \oplus X_s$, $Y_s \leftarrow U_s \oplus X_s$

23. a) $PM\mathbf{1}$: $CU \leftarrow$ 0 98, $OVERFLOW \leftarrow 0$
$C'PM\mathbf{2},\mathbf{2}$: $U_s \leftarrow 1$, U ← 02, $FINISH \leftarrow 1$
Result: U_sU = 1 02 = -2

b) $PM\mathbf{1}$: $CU \leftarrow$ 0 64, $OVERFLOW \leftarrow 0$
$C'PM\mathbf{2},\mathbf{2}$: $U_s \leftarrow 1$, U ← 36, $FINISH \leftarrow 1$
Result: U_sU = 1 36 = -36

c) $PM'\mathbf{1}$: $CU \leftarrow$ 0 30
$PM'\mathbf{2},\mathbf{2}$: $U_s \leftarrow 0$, $OVERFLOW \leftarrow 0$, $FINISH \leftarrow 1$
Result: U_sU = 0 30 = +30

24. a)

Conditions	*Micro-operations*	*i*	C_d	*U*	*V*	*Y*	Z_{Y0}	*Z*	FINISH
START		x	x	xx	xx	23	0		0
1	$U_s \leftarrow 0$, $V_s \leftarrow 0$, $U \leftarrow 00$, $i \leftarrow 2$, $C_d \leftarrow 0$	2	0	00				0	
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	17		22	0		
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	34		21	0		
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	51		20	1		
Z_{Y0}**2**	$i \leftarrow i - 1$	1						0	
3,Z'**3**	dshr(C_dUV), dshr(Y), goto **2**		0	05	1x	02	0		
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	22		01	0		
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	39		00	1		
Z_{Y0}**2**	$i \leftarrow i - 1$	0						1	
3,Z**3**	dshr(C_dUV), dshr(Y), $FINISH \leftarrow 1$			03	91				1

Result: +17 × +23 = +391

b)

Conditions	*Micro-operations*	*i*	C_d	*U*	*V*	*Y*	Z_{Y0}	*Z*	FINISH
START		x	x	xx	xx	32	0		0
1	$U_s \leftarrow 1$, $V_s \leftarrow 1$, $U \leftarrow 00$, $i \leftarrow 2$, $C_d \leftarrow 0$	2	0	00				0	
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	71		31	0		
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		1	42		30	1		
Z_{Y0}**2**	$i \leftarrow i - 1$	1						0	
3,Z'**3**	dshr(C_dUV), dshr(Y), goto **2**		0	14	2x	03			
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	85		02	0		
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$,		1	56		01	0		

	$Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**								
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		2	27		00	1		
Z_{Y0}**2**	$i \leftarrow i - 1$	0						1	
3,*Z***3**	dshr(C_dUV), dshr(Y), $FINISH \leftarrow 1$			22	72				1

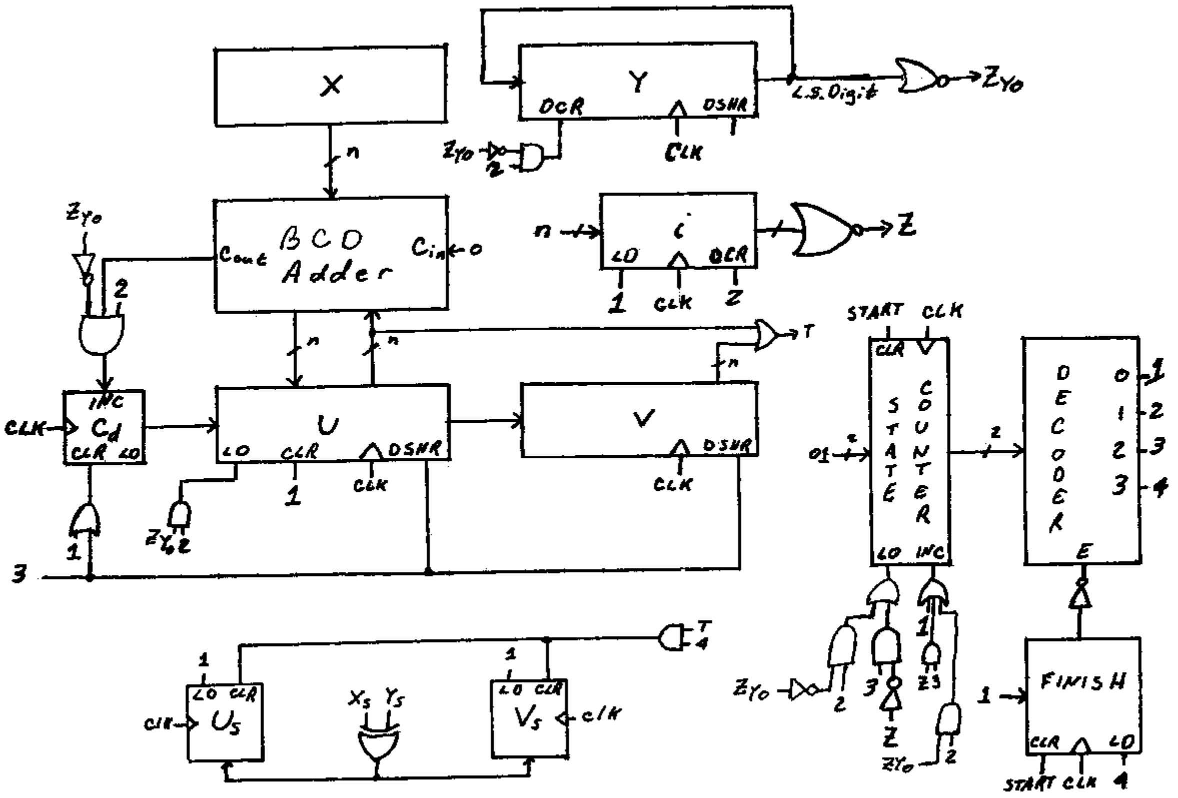
Result: +71× -32 = -2272

c)

Conditions	*Micro-operations*	*i*	C_d	*U*	*V*	*Y*	Z_{Y0}	Z	FINISH
START		x	x	xx	xx	10	0		0
1	$U_s \leftarrow 0$, $V_s \leftarrow 0$, $U \leftarrow 00$, $i \leftarrow 2$, $C_d \leftarrow 0$	2	0	00				0	
Z_{Y0}**2**	$i \leftarrow i - 1$	1						0	
3,*Z'***3**	dshr(C_dUV), dshr(Y), goto **2**		0	00	0x	01			
Z'_{Y0} **2**	$C_dU \leftarrow C_dU + X$, $Y_{d0} \leftarrow Y_{d0} - 1$, goto **2**		0	39		00	1		
Z_{Y0}**2**	$i \leftarrow i - 1$	0						1	
3,*Z***3**	dshr(C_dUV), dshr(Y), $FINISH \leftarrow 1$		0	03	90				1

Result: -39× -10 = +390

25.



26.

$$\begin{aligned}
G'\mathbf{1}&: U_s \leftarrow U_s \oplus X_s,\ Y_s \leftarrow U_s \oplus X_s\\
\mathrm{G}\mathbf{1}&: FINISH \leftarrow 1,\ OVERFLOW \leftarrow 1\\
\mathbf{2}&: Y \leftarrow 0,\ C_d \leftarrow 0,\ OVERFLOW \leftarrow 0,\ i \leftarrow n\\
\mathbf{3}&: \text{dshl } C_dUV,\ \text{dshl } Y,\ i \leftarrow i - 1\\
(Z'_{cd} + G)\mathbf{4}&: Y_0 \leftarrow Y_0 + 1,\ C_dU \leftarrow C_dU + X' + 1,\ \text{goto } \mathbf{4}\\
Z_{cd}G'Z'\mathbf{4}&: \text{goto } \mathbf{3}\\
Z_{cd}G'Z\mathbf{4}&: FINISH \leftarrow 1
\end{aligned}$$

27.

$$\begin{aligned}
\mathbf{1}_1&: C_dU \leftarrow C_dU + X' + 1 \qquad \text{(Note: } X' + 1 = 967\text{, not 67)}\\
\mathbf{1}_2&: U \leftarrow U + X\\
Z'_{cd}\mathbf{1}_2&: FINISH \leftarrow 1,\ OVERFLOW \leftarrow 1\\
Z_{cd}\mathbf{1}_2&: U_s \leftarrow U_s \oplus X_s,\ Y_s \leftarrow U_s \oplus X_s\\
\mathbf{2}&: Y \leftarrow 0,\ OVERFLOW \leftarrow 0,\ i \leftarrow n\\
\mathbf{3}&: \text{dshl } C_dUV,\ \text{dshl } Y,\ i \leftarrow i - 1\\
\mathbf{4}_1&: CC_dU \leftarrow C_dU + X' + 1\\
C\mathbf{4}_2&: Y_0 \leftarrow Y_0 + 1,\ C_dU \leftarrow C_dU + X' + 1,\ \text{goto } \mathbf{4}_2\\
C'\mathbf{4}_2&: C_dU \leftarrow C_dU + X\\
Z'C'\mathbf{4}_2&: \text{goto } \mathbf{3}\\
ZC'\mathbf{4}_2&: FINISH \leftarrow 1
\end{aligned}$$

28. a) 20 ns

b) $S_\infty = \dfrac{T_1}{T_k} = \dfrac{15+10+15}{20} = 2$

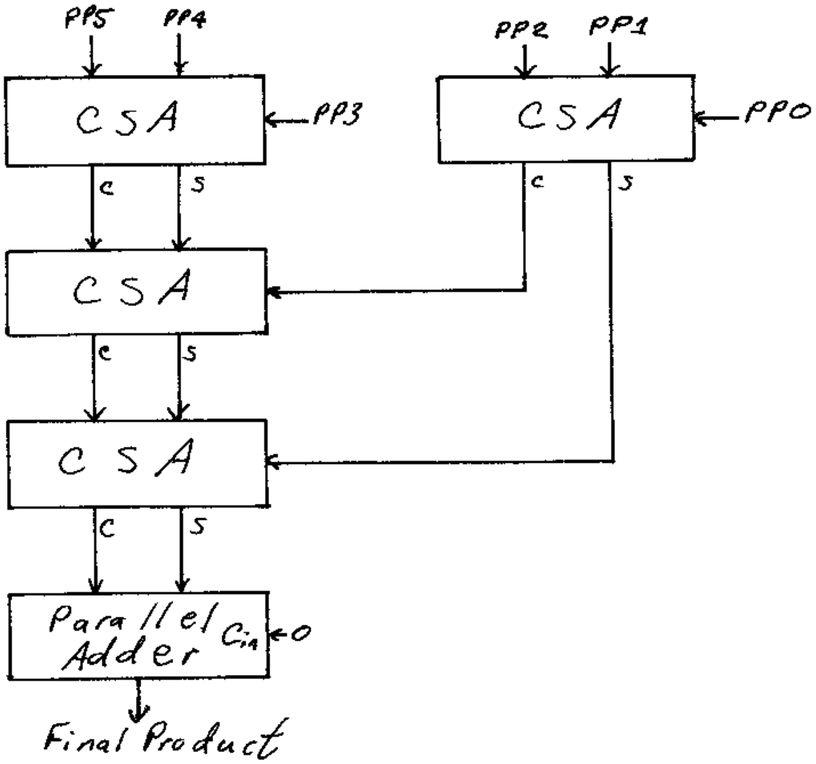
c) $n * 40 > 20 * (n + 2)$, which yields $n > 2$

d) $\dfrac{n * 40}{2(n+2)} = 1.5$, which yields $n = 6$

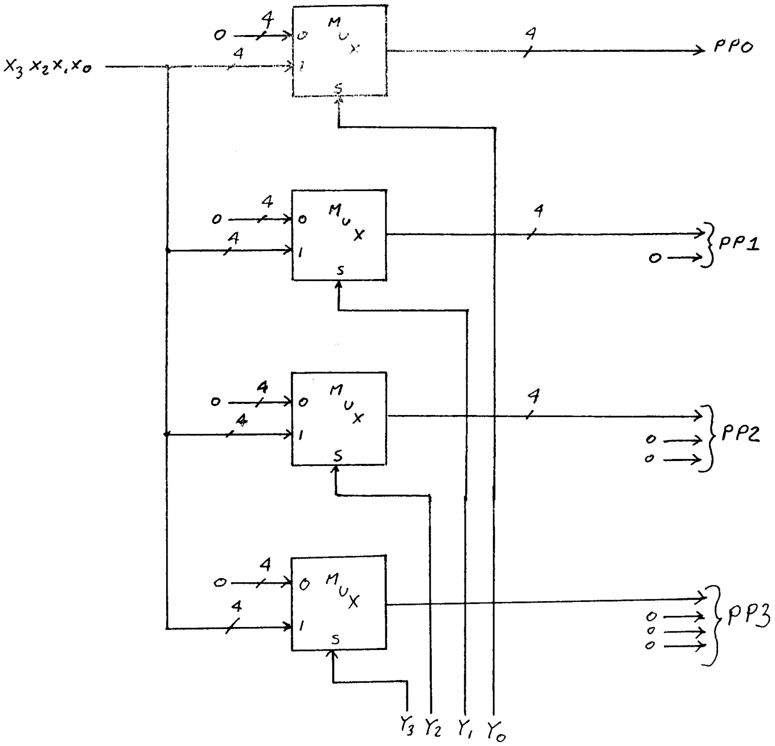
29.

Addresses	xxx000	xxx001	xxx010	xxx011	xxx100	xxx101	xxx110	xxx111
0-7	0	0	0	0	0	0	0	0
8-15	0	1	2	3	4	5	6	7
16-23	0	2	4	6	8	10	12	14
24-31	0	3	6	9	12	15	18	21
32-39	0	4	8	12	16	20	24	28
40-47	0	5	10	15	20	25	30	35
48-55	0	6	12	18	24	30	36	42
56-63	0	7	14	21	28	35	42	49

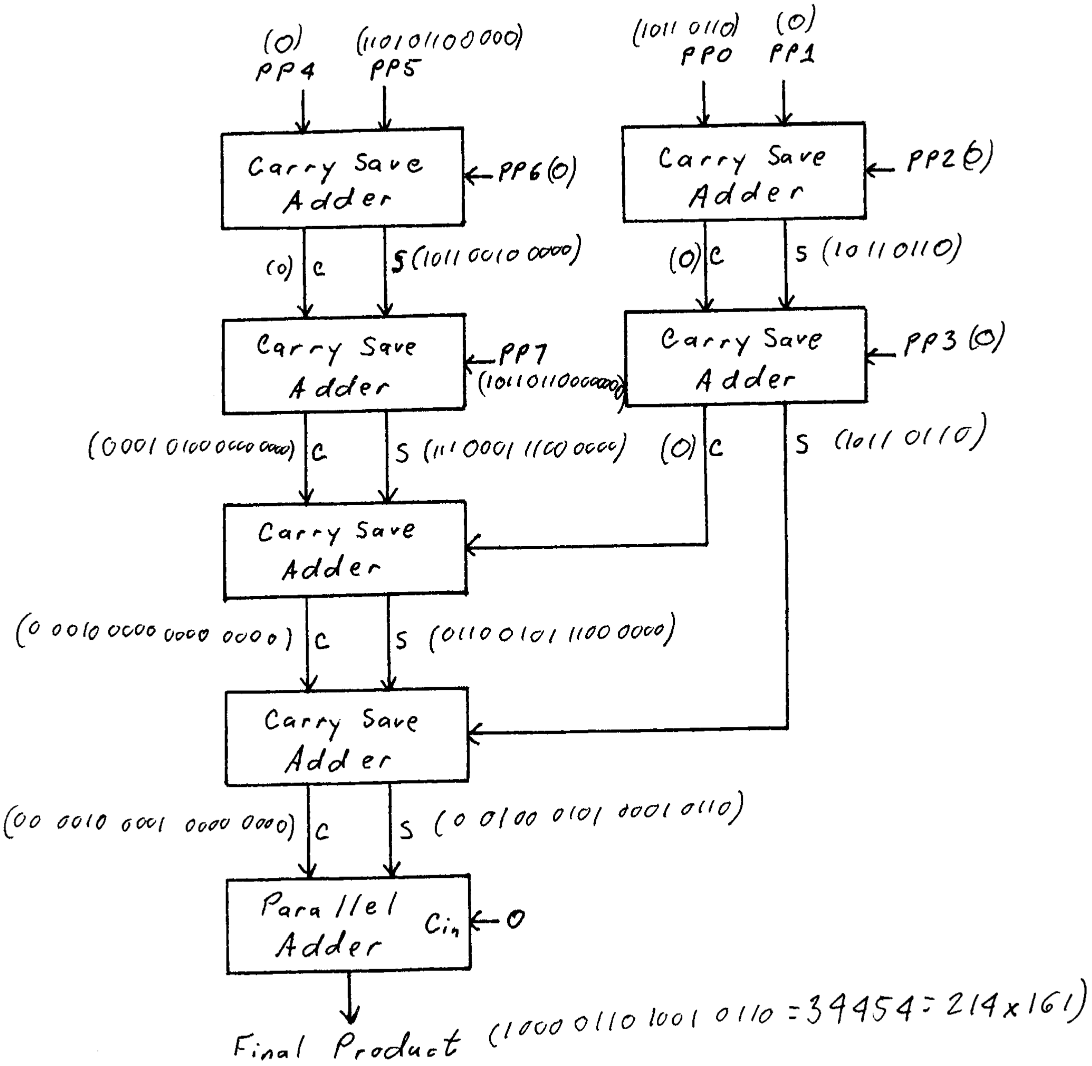
30.


PP5
PP4
PP2
PP1
CSA
PP3
CSA
PP0
C
S
C
S
CSA
C
S
CSA
C
S
Parallel Adder
C_{in}
0
Final Product

31.

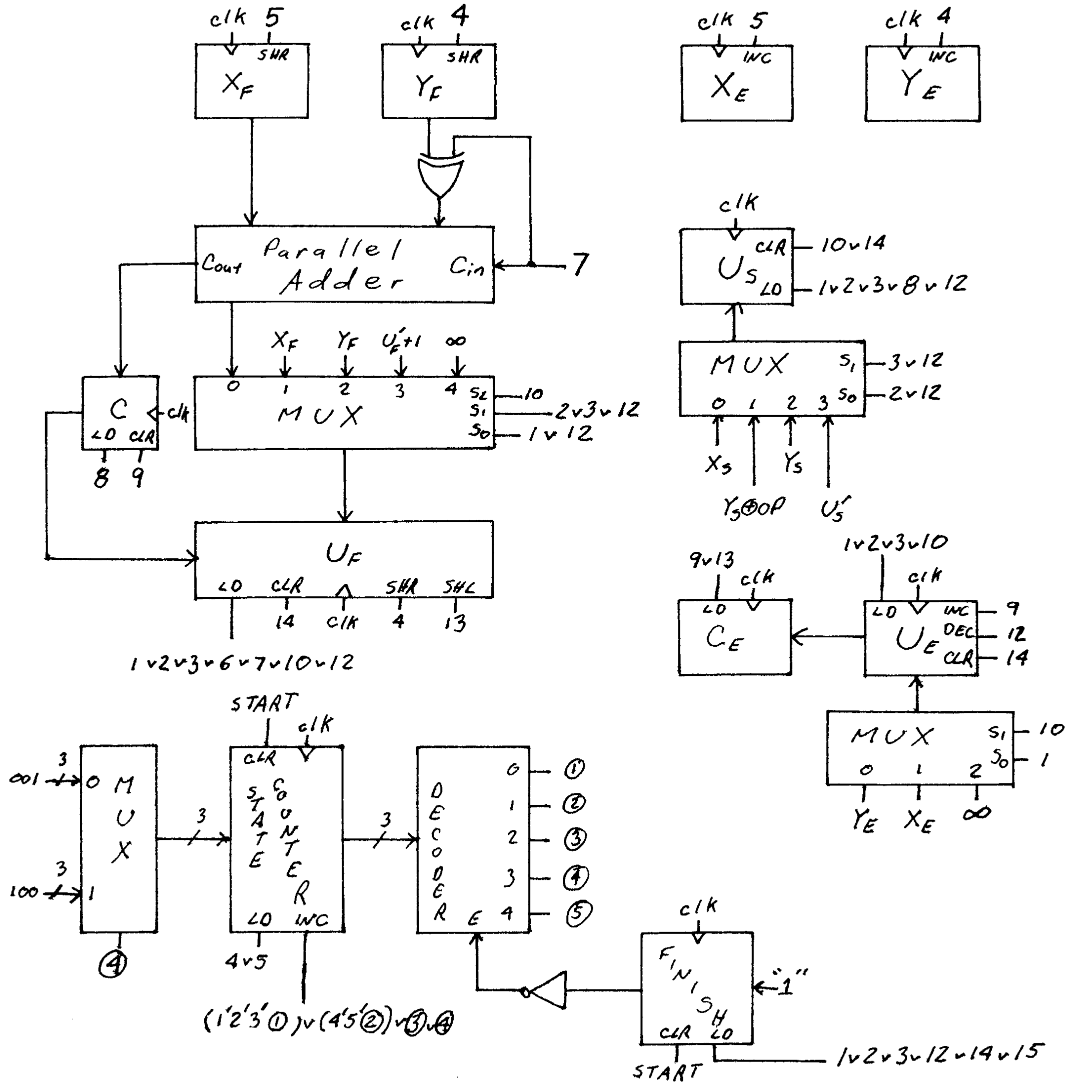

0
4
4
$X_3 X_2 X_1 X_0$
MUX
0
1
S
4
PP0
MUX
PP1
PP2
PP3
Y_3
Y_2
Y_1
Y_0

32.


(0)
PP4
(110101100000)
PP5
(1011 0110)
PP0
(0)
PP1
Carry Save Adder
PP6(0)
Carry Save Adder
PP2(0)
(0) C
S(1011 0010 0000)
(0) C
S (10110110)
Carry Save Adder
PP7
(101101100000000)
Carry Save Adder
PP3 (0)
(0001 0100 0000 0000) C
S (1110001 1100 0000)
(0) C
S (1011 0110)
Carry Save Adder
(0 0010 0000 0000 0000) C
S (0110 0101 1100 0000)
Carry Save Adder
(00 0010 0001 0000 0000) C
S (0 0100 0101 0001 0110)
Parallel Adder
Cin
0
Final Product (1000 0110 1001 0110 = 34454 = 214 x 161)

33. The following symbols are used in this design.

Condition	Symbol	Condition	Symbol
$(I_X N'_Y + N_X + Z_Y)\mathbf{1}$:	**1**	$CPM'\mathbf{4}$:	**9**
$(N'_Y Z_X Z'_Y + I'_X I_Y N'_X)\mathbf{1}$:	**2**	$C_E PM'\mathbf{5}$:	**10**
$(I_Y N_X + N'_X N_Y)\mathbf{1}$:	**3**	$PM'\mathbf{5}$:	**11**
$E_{XY}\mathbf{2}$:	**4**	$C'PM\mathbf{4}$:	**12**
$E_{YX}\mathbf{2}$:	**5**	$Z'_U C'_E U'_{F(n-1)} PM\mathbf{5}$:	**13**
$PM'\mathbf{3}$:	**6**	$(Z_U + C_E)PM\ \mathbf{5}$:	**14**
$PM\mathbf{3}$:	**7**	$C'_E U_{F(n-1)} PM\ \mathbf{5}$:	**15**
$\mathbf{3}$:	**8**		



Chapter 9

1. Mask: 1111 1111 0000 0000
   Data: 1111 0000 xxxx xxxx

2. a) The fifth location from the top
   b) The third, seventh, and eighth locations from the top
   c) No locations match this criteria

3. a)

20 bit tag	16 bit data	Valid bit

37 bits

b)

7 bit tag	16 bit data	Valid bit

24 bits

c)

8 bit tag	16 bit data	Valid bit	8 bit tag	16 bit data	Valid bit
	Way 1			Way 2	

50 bits

d)

9 bit tag	16 bit data	Valid bit	9 bit tag	16 bit data	Valid bit	9 bit tag	16 bit data	Valid bit	9 bit tag	16 bit data	Valid bit
	Way 1			Way 2			Way 3			Way 4	

104 bits

4. a)

18 bit tag	8 bit data	Valid bit

27 bits

b)

4 bit tag	8 bit data	Valid bit

13 bits

c)

5 bit tag	8 bit data	Valid bit	5 bit tag	8 bit data	Valid bit
	Way 1			Way 2	

28 bits

d)

6 bit tag	8 bit data	Valid bit	6 bit tag	8 bit data	Valid bit	6 bit tag	8 bit data	Valid bit	6 bit tag	8 bit data	Valid bit
	Way 1			Way 2			Way 3			Way 4	

60 bits

5. a) 32 or 33 bits:
   15 for the address tag
   8 for the first data value
   8 for the second data value
   1 for the valid bit
   1 for the dirty bit (only if the cache uses write-back)

   b) Assuming the bits are ordered as listed in part a: 111 1111 1111 1111 0000 0000 0000 0000

6. Only changes shown

Instruction	Address bits	Data 1	Data 2	Valid	Comments
LDAC 4234	000 0000 0000 0000	01	34	1	
	000 0000 0000 0001	42	0B	1	
	010 0001 0001 1010	55	29	1	
CLAC	No changes				Cache hit
JMPZ 000A	000 0000 0000 0010	06	0A	1	
	000 0000 0000 0011	00	05	1	
INAC	000 0000 0000 0101	0A	03	1	
MVAC	No changes				Cache hit
ADD	000 0000 0000 0110	08	02	1	
STAC 0927					Cache hit
	000 0000 0000 0111	29	09	1	(opcode)
	000 0100 1001 0011	--	02	1	
JUMP 0000	000 0000 0000 1000	05	00	1	
	000 0000 0000 1001	00	--	1	

7. Only changes shown

Instruction	Address bits	Tag	Data	Valid	Dirty	Comments
LDAC 4234	0	000	01	1	0	
	1	000	34	1	0	
	2	000	42	1	0	
	4	423	55	1	0	
CLAC	3	000	0B	1	0	
JMPZ 000A	4	000	06	1	0	Replace data
	5	000	0A	1	0	
	6	000	00	1	0	
INAC	A	000	0A	1	0	
MVAC	B	000	03	1	0	
ADD	C	000	08	1	0	
STAC 0927	D	000	02	1	0	
	E	000	27	1	0	
	F	000	09	1	0	
	7	092	02	1	1	
JUMP 0000	0	001	05	1	1	Replace data
	1	001	00	1	1	Replace data
	2	001	00	1	1	Replace data

8. Only changes shown

Instruction	Address bits	Tag	Data 0	Data 1	Data 2	Data 3	Valid	Dirty	Comments
LDAC 4234	0	000	01	34	42	0B	1	0	
	1	423	55	29	--	--	1	0	
CLAC	No changes								Cache hit
JMPZ 000A	1	000	06	0A	00	05	1	0	Replace data
INAC	2	000	00	00	0A	03	1	0	
MVAC	No changes								Cache hit
ADD	3	000	08	02	27	09	1	0	
STAC 0927									Cache hit (instr)
	1	092	--	--	--	02	1	1	Replace data
JUMP 0000	0	001	05	00	00	--	1	0	Replace data

9. Only changes shown

Instruction	Address bits	Tag 1	Data 1	Valid 1	Dirty 1	Tag 2	Data 2	Valid 2	Dirty 2	Comments
LDAC 4234	0	000	01	1	0					
	1	000	34	1	0					
	2	000	42	1	0					
	4	423	55	1	0					
CLAC	3	000	0B	1	0					
JMPZ 000A	4	423	55	1	0	000	06	1	0	
	5	000	0A	1	0					
	6	000	00	1	0					
INAC	A	000	0A	1	0					
MVAC	B	000	03	1	0					
ADD	C	000	08	1	0					
STAC 0927	D	000	02	1	0					
	E	000	27	1	0					
	F	000	09	1	0					
	7	092	02	1	1					
JUMP 0000	0	000	01	1	0	001	05	1	0	
	1	000	34	1	0	001	00	1	0	
	2	000	42	1	0	001	00	1	0	

10. Only changes shown

Instruction	Address bits	Tag 1	Data 1	Valid 1	Dirty 1	Tag 2	Data 2	Valid 2	Dirty 2	Comments
LDAC 4234	0	000	01/34	1	0					
	1	000	42/0B	1	0					
	2	423	55/29	1	0					
CLAC	No change									Cache hit
JMPZ 000A	2	423	55/29	1	0	000	06/0A	1	0	
	3	000	00/05	1	0					
INAC	5	000	0A/03	1	0					
MVAC	No change									Cache hit
ADD	6	000	08/02	1	0					
STAC 0927										Hit (instr)
	7	000	27/09	1	0					
	3	000	00/05	1	0	092	--/02	1	1	
JUMP 0000	0	000	01/34	1	0	001	05/00	1	0	
	1	000	42/0B	1	0	001	00/--	1	0	

11. Only changes shown

Note: Both LRU and FIFO replacement policies replace the same values for this program.

Instruction	Address	Tag1	Data1	Valid1	Dirty1	Tag2	Data2	Valid2	Dirty2	Comments
LDAC 4234	0	000	01/34/42/0B	1	0					
	1	423	55/29/--/--	1	0					
CLAC	No change									Cache hit
JMPZ 000A	1	423	55/29/--/--	1	0	000	06/0A/00/05	1	0	
INAC	2	000	00/00/0A/03	1	0					
MVAC	No change									Cache hit
ADD	3	000	08/02/27/09	1	0					
STAC 0927		423	55/29/--/--	1	0		--/--/--/02			Hit (instr)
	1					092		1	1	Replace
JUMP 0000	0	000	01/34/42/0B	1	0	001	05/00/00/--	1	0	

12. Hit ratio = 4.5%

Instruction	Tag	Data	Valid	Dirty	Comments
LDAC 4234	0000	01	1	0	
	0001	34	1	0	
	0002	42	1	0	
	4234	55	1	0	
STAC 4235	0003	0B	1	0	
	0004	35	1	0	
	0005	42	1	0	
	4235	55	1	1	
MVAC	0006	03	1	0	
INAC	0007	0A	1	0	
ADD	0008	08	1	0	
JPNZ 0020	0009	07	1	0	
	000A	20	1	0	
	000B	00	1	0	
LDAC 4235	0020	01	1	0	
	0021	35	1	0	
	0022	42	1	0	
	4235	55	1	1	Cache hit
JUMP 0029	0023	05	1	0	
	0024	29	1	0	
	0025	00	1	0	
AND	0029	0C	1	0	

13. Hits in italics Hit ratio = 40.9%

Instruction	Tag	Data	Valid	Dirty	Comments
LDAC 4234	0000	01/*34*	1	0	
	0001	42/*0B*	1	0	
	211A	55/*55*	1	0	
STAC 4235					Hit (instr)
	0002	35/42	1	0	
	211A	55/*55*	1	1	Hit (4235)
MVAC	0003	03/*0A*	1	0	
INAC					Cache hit
ADD	0004	08/*07*	1	0	
JPNZ 0020					Hit (instr)
	0005	20/*00*	1	0	
			1	0	Hit (00)
LDAC 4235	0010	01/*35*	1	0	Hit (instr)
	0011	42/*05*	1	0	Hit (05) - replaces data
	211A				Hit (55) - replaces data
JUMP 0029					Hit (instr)
	0012	29/*00*	1	0	Replaces data
AND	0014	00/0C	1	0	Replaces data

14. Hit ratio = 4.5%

Instruction	Address	Tag	Data	Valid	Dirty	Comments
LDAC 4234	0	000	01	1	0	
	1	000	34	1	0	
	2	000	42	1	0	
	4	423	55	1	0	
STAC 4235	3	000	0B	1	0	
	4	000	35	1	0	Replaces data
	5	000	42	1	0	
	5	423	55	1	1	Replaces data
MVAC	6	000	03	1	0	
INAC	7	000	0A	1	0	
ADD	8	000	08	1	0	
JPNZ 0020	9	000	07	1	0	
	A	000	20	1	0	
	B	000	00	1	0	
LDAC 4235	0	002	01	1	0	Replaces data
	1	002	35	1	0	Replaces data
	2	002	42	1	0	Replaces data
						Hit (read from 4235)
JUMP 0029	3	002	05	1	0	Replaces data
	4	002	29	1	0	Replaces data
	5	002	00	1	0	Replaces data
AND	9	002	0C	1	0	Replaces data

15. Hits in italics Hit ratio = 50.0%

Instruction	Address	Tag	Data	Valid	Dirty	Comments
LDAC 4234	0	000	01/*34*	1	0	Hit (34)
	1	000	42/*0B*	1	0	
	2	423	55/--	1	0	
STAC 4235						Hit (STAC)
	2	000	35/*42*	1	0	Hit (42) - Replaces data
	2	423	55/*55*	1	1	Hit (4235) Replaces data
MVAC	3	000	03/*0A*	1	0	
INAC						Hit (INAC)
ADD	4	000	08/*07*	1	0	
JPNZ 0020						Hit (STAC)
	5	000	20/*00*	1	0	Hit (00)
LDAC 4235	0	002	01/*35*	1	0	Hit (35) - Replaces data
	1	002	42/*05*	1	0	Replaces data
						Hit (4235)
JUMP 0029						Hit (JUMP)
	2	002	29/*00*	1	0	Hit (00) - Replaces data
AND	4	002	00/0C	1	0	Replaces data

16. Hits in italics LRU value underlined Hit ratio = 4.5%

Instruction	Address	Tag1	Data1	V1	D1	Tag2	Data2	V2	D2	Tag3	Data3	V3	D3	Tag4	Data4	V4	D4
LDAC 4234	0	0000	01	1	0	108D	55	1	0								
	1	0000	34	1	0												
	2	0000	42	1	0												
STAC 4235	3	0000	0B	1	0												
	0	0000	01	1	0	108D	55	1	0	0001	35	1	0				
	1	0000	34	1	0	0001	42	1	0	108D	55	1	1				
MVAC	2	0000	42	1	0	0001	03	1	0								
INAC	3	0000	0B	1	0	0001	0A	1	0								
ADD	0	0000	01	1	0	108D	55	1	0	0001	35	1	0	0002	08	1	0
JPNZ 0020	1	0000	34	1	0	0001	42	1	0	108D	55	1	1	0002	07	1	0
	2	0000	42	1	0	0001	03	1	0	0002	20	1	0				
	3	0000	0B	1	0	0001	0A	1	0	0002	00	1	0				
LDAC 4235	0	0008	01	1	0	108D	55	1	0	0001	35	1	0	0002	08	1	0
	1	0008	35	1	0	0001	42	1	0	108D	*55*	1	1	0002	07	1	0
	2	0000	42	1	0	0001	03	1	0	0002	20	1	0	0008	42	1	0
JUMP 0029	3	0000	0B	1	0	0001	0A	1	0	0002	00	1	0	0008	05	1	0
	0	0008	01	1	0	0009	29	1	0	0001	35	1	0	0002	08	1	0
	1	0008	01	1	0	0009	00	1	0	108D	55	1	0	0002	07	1	0
AND	1	0008	01	1	0	0009	00	1	0	000A	0C	1	0	0002	07	1	0

17. Hits in italics LRU value underlined Hit ratio = 40.9%

Instruction	Address	Tag1	Data1	V1	D1	Tag2	Data2	V2	D2
LDAC 4234	0	0000	01/*34*	1	0				
	1	0000	42/*02*	1	0				
	2	211A	55/--	1	0				
STAC 4235									
	2	211A	55/*55*	1	0	0000	35/*42*	1	1
MVAC	3	0000	03/*0A*	1	0				
INAC									
ADD	0	0000	01/34	1	0	0001	08/*07*	1	0
JPNZ 0020									
	1	0000	42/02	1	0	0002	20/*00*	1	0
LDAC 4235	0	0008	01/*35*	1	0	0001	08/07	1	0
	1	0008	42/*05*	1	0	0002	20/00	1	0
	1	0008	42/05	1	0	211A	29/*00*	1	0
JUMP 0029									
	2	0009	29/*00*	1	0	0000	35/42	1	1
AND	0	0008	01/35	1	0	0009	0C/--	1	0

18. $T_M = hT_C + (1 - h)T_P$ = (.75 * 8 ns) + (.25 * 65 ns) = 22.25 ns

19. $T_C = (T_M - (1 - h)T_P)/h$ = (39.9 ns - .35 * 75 ns) / .65 = 21 ns

20. $T_P = (T_M - hT_C)/(1 - h)$ = (24 ns - .8 * 10 ns) / .2 = 80 ns

21. $h = (T_M - T_P)/(T_C - T_P)$ = (40 ns - 55 ns) / (10 ns - 55 ns) = 0.333 ns

22. The next jump instruction is always overwritten by its predecessor; *h* = 0.

23. (Only changes shown)

	Address	Frame	Valid	Comments
LDAC 4234	0	0	1	
	4	1	1	
JUMP 1000				No changes
STAC 4235	1	2	1	4235 already in memory
JUMP 2000				No changes
JUMP 0010	2	3	1	
JUMP 3000				No changes
JUMP 0100	3	0	1	
	0	0	0	
JUMP 1100	0	1	0	

24.

LDAC 4234 JUMP 1000	STAC 4235 JUMP 2000	JUMP 0010	JUMP 3000	JUMP 0100	JUMP 1100
P F V	P F V	P F V	P F V	P F V	P F V
0 0 1	1 2 1	1 2 1	0 0 1	3 0 1	3 0 1
4 1 1	4 1 1	2 3 1	2 3 1	2 3 1	0 1 1

25.

LDAC 4234 JUMP 1000	STAC 4235 JUMP 2000	JUMP 0010	JUMP 3000	JUMP 0100	JUMP 1100
P F V	P F V	P F V	P F V	P F V	P F V
0 0 1	1 2 1	1 2 1	0 0 1	3 0 1	3 0 1
4 1 1	4 1 1	2 3 1	2 3 1	2 3 1	0 1 1

26. a) 1554H
b) 2000H
c) Fault

27. a) F231H
b) Fault
c) 4401H

28. a) 1C35H
b) 0A38H
c) Fault

29. a) C543H
b) 4077H
c) 8401H

30. a) 2000H
b) 0D61H
c) 3FFFH

31. a) 9512H
    b) 3456H
    c) 63EDH

32. a)

2	C	1

b)

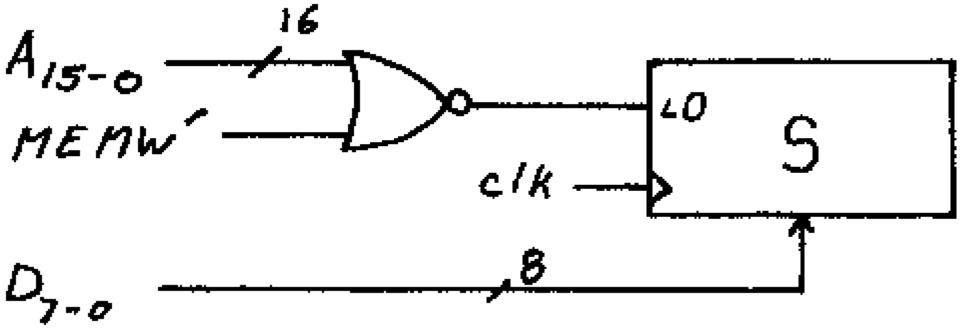
9	6	1

c)

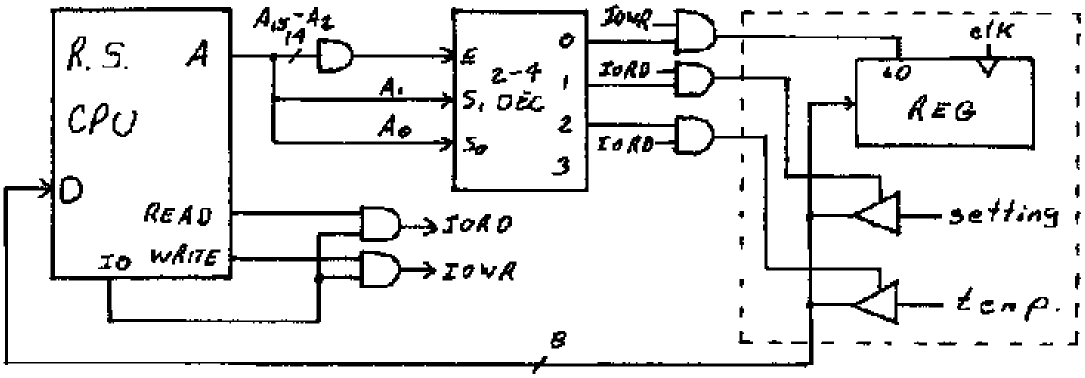
E	A	1

Chapter 10

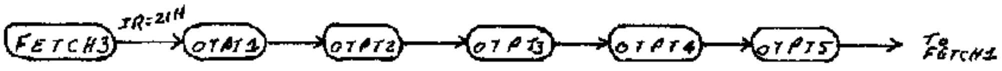
1.



2.

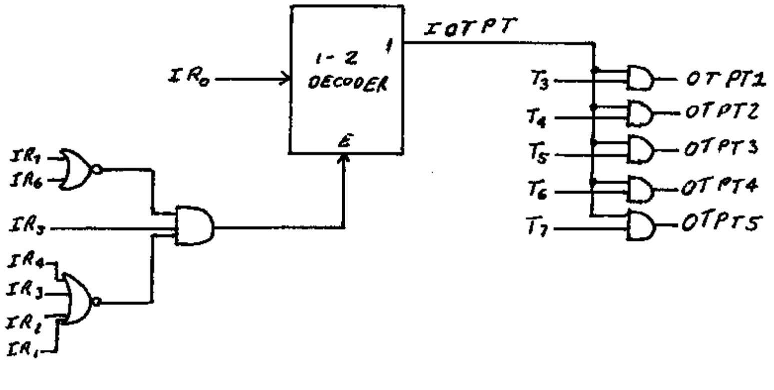


3.



OTPT1: *DR* ← *M*, *PC* ← *PC* + 1, *AR* ← *AR* + 1
OTPT2: *TR* ← *DR*, *DR* ← *M*, *PC* ← *PC* + 1
OTPT3: *AR* ← *DR*,*TR*
OTPT4: *DR* ← *AC*
OTPT5: *Output port* ← *DR*

3. Add the following connections using the same decoder used to generate the states of the INPT execute routine. The rest of the circuit is unchanged.



5.

MEMBUS	=	(old value of MEMBUS) ∨ INPT1 ∨ INPT2 ∨ INPT4
PCINC	=	(old value of PCINC) ∨ INPT1 ∨ INPT2
TRLOAD	=	(old value of TRLOAD) ∨ INPT2
ARLOAD	=	(old value of ARLOAD) ∨ INPT3
DRHBUS	=	(old value of DRHBUS) ∨ INPT3
TRBUS	=	(old value of TRBUS) ∨ INPT3
DRLBUS	=	(old value of DRLBUS) ∨ INPT5

6. The control unit changes are the same as for Problem 10.4. The control signal changes are as follows.

ARINC	=	(old value of ARINC) ∨ OTPT1
MEMBUS	=	(old value of MEMBUS) ∨ OTPT1 ∨ OTPT2
BUSMEM	=	(old value of BUSMEM) ∨ OTPT5
PCINC	=	(old value of PCINC) ∨ OTPT1 ∨ OTPT2
TRLOAD	=	(old value of TRLOAD) ∨ OTPT2
ARLOAD	=	(old value of ARLOAD) ∨ OTPT3
DRHBUS	=	(old value of DRHBUS) ∨ OTPT3
TRBUS	=	(old value of TRBUS) ∨ OTPT3
DRLBUS	=	(old value of DRLBUS) ∨ OTPT5

7.
i) Modify the mapping function to map instruction code 0010 0000 to microcode address 100 0000.
ii) Add microcode signal *IO*, which is 1 only for the microinstruction at address 67.
iii) Add the following microinstructions to memory. (Only active control signals are shown.)

```
64:  DRLOAD, MEMBUS, PCINC, ARINC         U J 65
65:  TRLOAD, DRLOAD, MEMBUS, PCINC        U J 66
66:  ARLOAD, DRHBUS, TRBUS                U J 67
67:  DRLOAD, MEMBUS                       U J 68
68:  ACLOAD, DRLBUS                       U J 01
```

8.
i) Modify the mapping function to map instruction code 0010 0001 to microcode address 100 1000.
ii) Add microcode signal *IO*, which is 1 for the microinstruction at address 76.
iii) Add the following microinstructions to memory. (Only active control signals are shown.)

```
72:  DRLOAD, MEMBUS, PCINC, ARINC         U J 73
73:  TRLOAD, DRLOAD, MEMBUS, PCINC        U J 74
74:  ARLOAD, DRHBUS, TRBUS                U J 75
75:  DRLOAD, ACBUS                        U J 76
76:  BUSMEM, DRLBUS                       U J 01
```

9.

Time (ns)	0	10	20	40	45	60	80	85	90	100
Routine	MAIN	IRQ1	IRQ2	IRQ1	IRQ3	IRQ4	IRQ3	IRQ1	MAIN	

10.

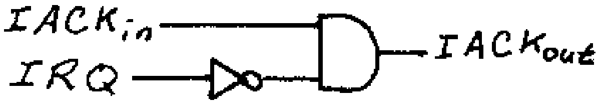
Time (ns)	0	10	30	50	70	90	100
Routine	MAIN	IRQ6	IRQ5	IRQ4	IRQ3	MAIN	

11.

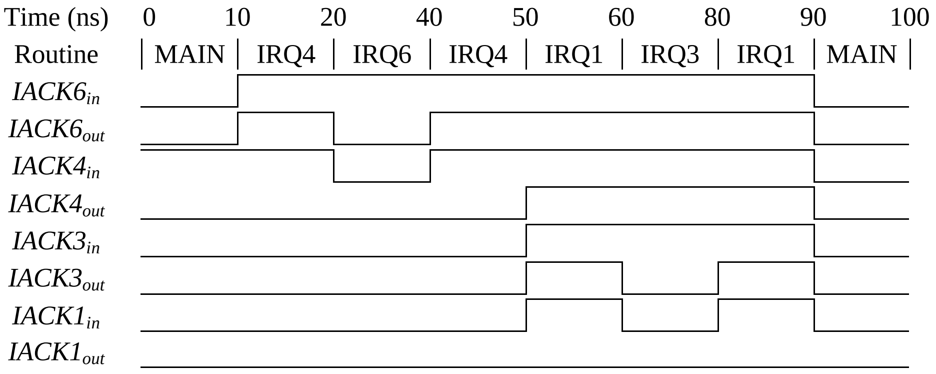
Time (ns)	0	10	20	40	50	60	80	90	100
Routine	MAIN	IRQ4	IRQ6	IRQ4	IRQ1	IRQ3	IRQ1	MAIN	

12. Daisy chaining is easier to modify when it is necessary to add peripherals to a computer system. It also requires fewer pins on the CPU. Other points may also be considered for this question.

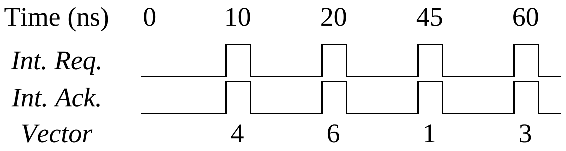
13. $IACK_{out} = IACK_{in} \wedge IRQ'$



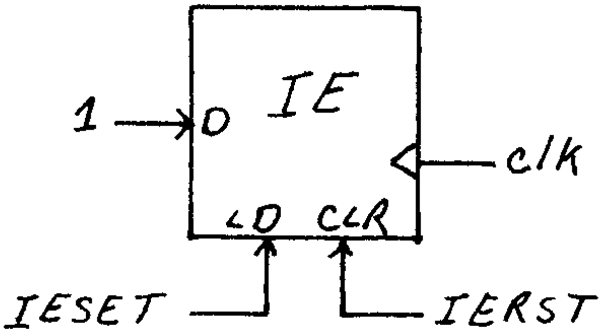
14.



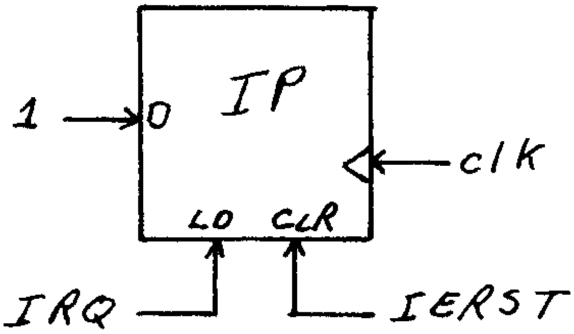
15.



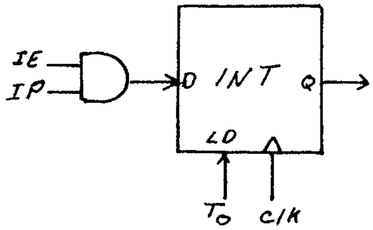
16.



17.



18. i) Add the following hardware to implement *INT*.



ii) Change the following state signals:

FETCH1 = *T0* ^ (*IP'* ∨ *IE'*)
FETCH2 = *T1* ^ *INT'*
FETCH3 = *T2* ^ *INT'*

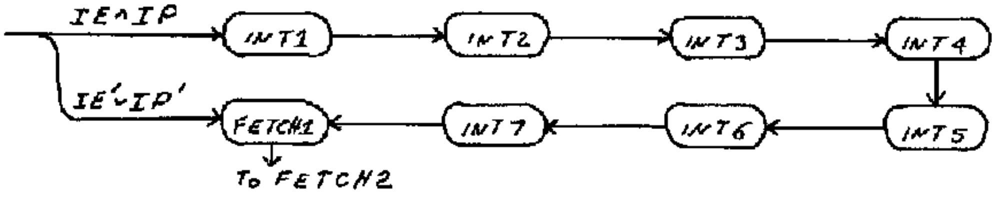
iii) Add hardware to generate the following new state signals:

INT1 = *T0* ^ (*IP* ^ *IE*)
INT2 = *T1* ^ *INT*
INT3 = *T2* ^ *INT*
INT4 = *T3* ^ *INT*
INT5 = *T4* ^ *INT*
INT6 = *T5* ^ *INT*
INT7 = *T6* ^ *INT*

iv) Modify the following internal control unit signals:

Decoder enable = (Old value of Decoder Enable) ^ *INT'*
Time counter CLR = (Old value of Time counter CLR) ^ *INT7*

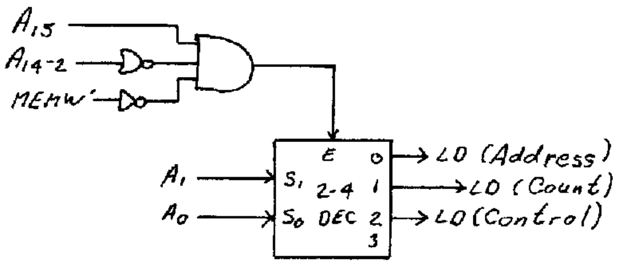
19. Replace state FETCH1 and its input and output arcs with the following.



20. *IE* ^ *IP* ^ *FETCH1*: *AR* ← *SP*
(*IE'* ∨ *IP'*) ^ *FETCH1*: *AR* ← *PC*
INT2 - INT7 are the same as in the chapter text.

21. *ARLOAD* = (Original value of *ARLOAD*) ∨ *INT1*
ARDEC = (Original value of *ARDEC*) ∨ *INT3*
SPBUF = (Original value of *SPBUF*) ∨ *INT1*
SPDEC = (Original value of *SPDEC*) ∨ *INT2* ∨ *INT3*
DRLOAD = (Original value of *DRLOAD*) ∨ *INT2* ∨ *INT4* ∨ *INT6*
DRLBUS = (Original value of *DRLBUS*) ∨ *INT3* ∨ *INT5* ∨ *INT7*
PCHBUS = (Original value of *PCHBUS*) ∨ *INT2*
PCLBUS = (Original value of *PCLBUS*) ∨ *INT4*
PCLOAD = (Original value of *PCLOAD*) ∨ *INT7*
BUSMEM = (Original value of *BUSMEM*) ∨ *INT3* ∨ *INT5*
WRITE = (Original value of *WRITE*) ∨ *INT3* ∨ *INT5*
MEMBUS = (Original value of *MEMBUS*) ∨ *INT6*
IACK = *INT5* ∨ *INT6*

22.



23. LDAC 2000
OTPT 8000
LDAC 2001
OTPT 8001
LDAC 2002
OTPT 8002

24. Replace the FETCH1 input to the OR gate which drives the INC signal of the Time Counter with FETCH1 ^ *BR'*.

25. No changes are required, since the CPU does not interact with the data bus while a DMA transfer is active.

26. i) COH 10H 00H 00H 20H 99H 00H
ii) C1H 11H 00H 08H 28H 99H 01H
iii) C0H 10H 80H 00H 01H 99H 02H

27. a) 1 start + 0 parity + 1½ stop bits = 2½ bits overhead; 2½ ÷ (2½ + 8) = 23.8%.
b) 1 start + 1 parity + 2 stop bits = 4 bits overhead; 4 ÷ (4 + 7) = 36.4%.
c) 1 start + 1 parity + 1 stop bits = 3 bits overhead; 3 ÷ (3 + 5) = 37.5%.

28. a) Asynchronous: 2 ÷ (2 + 8) = 20.0%; HDLC: 48 ÷ (48 + 96) = 33.3%; Asynchronous has less overhead
b) Asynchronous: 2½ ÷ (2½ + 7) = 26.3%; HDLC: 48 ÷ (48 + 168) = 22.2%; HDLC has less overhead
c) Asynchronous: 2 ÷ (2 + 8) = 20.0%; HDLC: 48 ÷ (48 + 192) = 20.0%; Both have the same overhead.

29. LDAC 1111
OTPT 9800H
LDAC 1112H
OTPT 9801H

30. a)

Token packet:	24 bits
Data packet:	6168 bits
Handshaking packet:	8 bits
TOTAL:	6200 bits

b) 56 ÷ 6200 = 0.90%

c) 1536 × (1 start bit + 8 data bits + 1 stop bit) = 15,360 bits

Chapter 11

1. a) 25 ns × 100% vs. (24 ns × 99%) + (24 ns × 4 instructions × 1%) average time
   25 ns vs. 23.76 ns + 0.96 ns
   25 ns > 24.72 ns, therefore the second CPU has better performance
   b) 25 ns × 100% = (24 ns × (100 - x)%) + (96 ns × x%); x = 1.389%
   c) 25 ns × 100% = (24 ns × 99%) + (24 ns × x instructions × 1%); x = 5.167 instructions
   d) x ns × 100% = (24 ns × 99%) + (96 ns × 1%); x = 24.72 ns
   e) 25 ns × 100% = (x ns × 99%) + (4x ns × 1%); x = 24.272 ns

2. a) 15 ns × 100% vs. (12 ns × 98%) + (12 ns × 6 instructions × 2%) average time
   15 ns > 13.2 ns, therefore the second CPU has better performance
   b) 15 ns × 100% = (12 ns × (100 - x)%) + (72 ns × x%); x = 5%
   c) 15 ns × 100% = (12 ns × 98%) + (12 ns × x instructions × 2%); x = 13.5 instructions
   d) x ns × 100% = (12 ns × 98%) + (72 ns × 2%); x = 13.2 ns
   e) 15 ns × 100% = (x ns × 98%) + (6x ns × 2%); x = 13.636 ns

3. a) 18 ns × 100% vs. (16 ns × 96%) + (16 ns × 5 instructions × 4%) average time
   18 ns > 18.56 ns, therefore the first CPU has better performance
   b) 18 ns × 100% = (16 ns × (100 - x)%) + (80 ns × x%); x = 3.125%
   c) 18 ns × 100% = (16 ns × 96%) + (16 ns × x instructions × 4%); x = 4.125 instructions
   d) x ns × 100% = (16 ns × 96%) + (80 ns × 4%); x = 18.56 ns
   e) 18 ns × 100% = (x ns × 96%) + (5x ns × 4%); x = 15.517 ns

4. a) Clock period = 80 ns
   Steady state speedup = (40 + 80 + 50)/80 = 2.125
   b) 160/80 = 2

5. a) Clock period = 60 ns
   Steady state speedup = (30 + 25 + 60 + 40)/60 = 2.583
   b) 150/60 = 2.5

6. a) Clock period = 70 ns
   Steady state speedup = (20 + 25 + 20 + 70 + 40)/70 = 2.5
   b) 160/70 = 2.286

7. a) Clock period = 50 ns
   Steady state speedup = (40 + 45 + 35 + 50)/50 = 3.4
   b) 160/50 = 3.2

8. a) Clock period = 40 ns
   Steady state speedup = (30 + 25 + 20 + 40 + 40)/40 = 3.875
   b) 150/40 = 3.75

9. a) Clock period = 45 ns
   Steady state speedup = (20 + 25 + 20 + 25 + 45 + 40)/45 = 3.889
   b) 160/45 = 3.556
   c) Combine stages 1 and 2, and combine stages 3 and 4

10. 10 global registers + 8 windows × (4 input registers + 10 local registers) = 122 registers
Note: Common output registers are not counted since they are already counted as common input registers of the next window.

11. 160 registers = 16 global registers + W windows × (8 input registers + 16 local registers); W = 6 windows

12. 192 registers = 12 global registers + 10 windows × (6 input registers + L local registers);
L = 12 local registers

13. 188 registers = 20 global registers + 12 windows × (C input registers + 10 local registers);
C = 4 common input (and common output) registers

14. a) 2 no-ops
b) 2 no-ops

15. a)

1:*R1 ← R2 + R3*
N1:*No-op*
2:*R4 ← R1 + R2*
N2:*No-op*
3:*R3 ← R1 + R4*
4:*R5 ← R2 + R6*
5:*R6 ← R1 + R2*
N3:*No-op*
6:*R7 ← R5 + R6*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11
1	1	N1	2	N2	3	4	5	N3	6		
2	-	1	N1	2	N2	3	4	5	N3	6	
3	-	-	1	N1	2	N2	3	4	5	N3	6

b)

1:*R1 ← R2 + R3*
4:*R5 ← R2 + R6*
2:*R4 ← R1 + R2*
5:*R6 ← R1 + R2*
3:*R3 ← R1 + R4*
6:*R7 ← R5 + R6*

Stage\Cycle	1	2	3	4	5	6	7	8
1	1	4	2	5	3	6		
2	-	1	4	2	5	3	6	
3	-	-	1	4	2	5	3	6

c)

1:*R1 ← R2 + R3*
2:*R4 ← R1 + R2*
3:*R3 ← R1 + R4*
4:*R5 ← R2 + R6*
5:*R6 ← R1 + R2*
6:*R7 ← R5 + R6*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11
1	1	2	S	3	S	4	5	S	6		
2	-	1	2	S	3	S	4	5	S	6	
3	-	-	1	2	S	3	S	4	5	S	6

d)

1:*R1 ← R2 + R3*
2:*R4 ← R1 + R2*
3:*R3 ← R1 + R4*
4:*R5 ← R2 + R6*
5:*R6 ← R1 + R2*
6:*R7 ← R5 + R6*

Stage\Cycle	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6		
2	-	1	2	3	4	5	6	
3	-	-	1	2	3	4	5	6

16. a)

1: $R1 \leftarrow R2 + R3$
N1: *No-op*
N2: *No-op*
2: $R4 \leftarrow R1 + R2$
N3: *No-op*
N4: *No-op*
3: $R3 \leftarrow R1 + R4$
4: $R5 \leftarrow R2 + R6$
5: $R6 \leftarrow R1 + R2$
N5: *No-op*
N6: *No-op*
6: $R7 \leftarrow R5 + R6$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6			
2	-	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6		
3	-	-	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6	
4	-	-	-	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6

b)

1: $R1 \leftarrow R2 + R3$
4: $R5 \leftarrow R2 + R6$
N1: *No-op*
2: $R4 \leftarrow R1 + R2$
5: $R6 \leftarrow R1 + R2$
N2: *No-op*
3: $R3 \leftarrow R1 + R4$
6: $R7 \leftarrow R5 + R6$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11
1	1	4	N1	2	5	N2	3	6			
2	-	1	4	N1	2	5	N2	3	6		
3	-	-	1	4	N1	2	5	N2	3	6	
4	-	-	-	1	4	N1	2	5	N2	3	6

c)

1: $R1 \leftarrow R2 + R3$
2: $R4 \leftarrow R1 + R2$
3: $R3 \leftarrow R1 + R4$
4: $R5 \leftarrow R2 + R6$
5: $R6 \leftarrow R1 + R2$
6: $R7 \leftarrow R5 + R6$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	S	S	2	S	S	3	4	5	S	S	6			
2	-	1	S	S	2	S	S	3	4	5	S	S	6		
3	-	-	1	S	S	2	S	S	3	4	5	S	S	6	
4	-	-	-	1	S	S	2	S	S	3	4	5	S	S	6

d)

1: $R1 \leftarrow R2 + R3$
2: $R4 \leftarrow R1 + R2$
3: $R3 \leftarrow R1 + R4$
4: $R5 \leftarrow R2 + R6$
5: $R6 \leftarrow R1 + R2$
6: $R7 \leftarrow R5 + R6$

Stage\Cycle	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6			
2	-	1	2	3	4	5	6		
3	-	-	1	2	3	4	5	6	
4	-	-	-	1	2	3	4	5	6

17. a)

1:*R1* ← *R2* + *R3*
N1:*No-op*
N2:*No-op*
2:*R4* ← *R1* + *R2*
N3:*No-op*
N4:*No-op*
3:*R3* ← *R1* + *R4*
4:*R5* ← *R2* + *R6*
5:*R6* ← *R1* + *R2*
N5:*No-op*
N6:*No-op*
6:*R7* ← *R5* + *R6*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6				
2	-	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6			
3	-	-	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6		
4	-	-	-	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6	
5	-	-	-	-	1	N1	N2	2	N3	N4	3	4	5	N5	N6	6

b)

1:*R1* ← *R2* + *R3*
4:*R5* ← *R2* + *R6*
N1:*No-op*
2:*R4* ← *R1* + *R2*
5:*R6* ← *R1* + *R2*
N2:*No-op*
3:*R3* ← *R1* + *R4*
6:*R7* ← *R5* + *R6*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12
1	1	4	N1	2	5	N2	3	6				
2	-	1	4	N1	2	5	N2	3	6			
3	-	-	1	4	N1	2	5	N2	3	6		
4	-	-	-	1	4	N1	2	5	N2	3	6	
5	-	-	-	-	1	4	N1	2	5	N2	3	6

c)

1:*R1* ← *R2* + *R3*
2:*R4* ← *R1* + *R2*
3:*R3* ← *R1* + *R4*
4:*R5* ← *R2* + *R6*
5:*R6* ← *R1* + *R2*
6:*R7* ← *R5* + *R6*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	S	S	2	S	S	3	4	5	S	S	6				
2	-	1	S	S	2	S	S	3	4	5	S	S	6			
3	-	-	1	S	S	2	S	S	3	4	5	S	S	6		
4	-	-	-	1	S	S	2	S	S	3	4	5	S	S	6	
5	-	-	-	-	1	S	S	2	S	S	3	4	5	S	S	6

d)

1:*R1* ← *R2* + *R3*
2:*R4* ← *R1* + *R2*
3:*R3* ← *R1* + *R4*
4:*R5* ← *R2* + *R6*
5:*R6* ← *R1* + *R2*
6:*R7* ← *R5* + *R6*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6				
2	-	1	2	3	4	5	6			
3	-	-	1	2	3	4	5	6		
4	-	-	-	1	2	3	4	5	6	
5	-	-	-	-	1	2	3	4	5	6

18. a)

1: $R1 \leftarrow R2 + R3$
N1: *No-op*
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
N2: *No-op*
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10
1	1	N1	2	3	4	N2	5	6		
2	-	1	N1	2	3	4	N2	5	6	
3	-	-	1	N1	2	3	4	N2	5	6

b)

1: $R1 \leftarrow R2 + R3$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
2: $R1 \leftarrow R1 + R2$
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8
1	1	3	4	2	5	6		
2	-	1	3	4	2	5	6	
3	-	-	1	3	4	2	5	6

c)

1: $R1 \leftarrow R2 + R3$
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10
1	1	2	3	S	4	5	S	6		
2	-	1	2	3	S	4	5	S	6	
3	-	-	1	2	3	S	4	5	S	6

d)

1: $R1 \leftarrow R2 + R3$
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6		
2	-	1	2	3	4	5	6	
3	-	-	1	2	3	4	5	6

19. a)

1: $R1 \leftarrow R2 + R3$
N1: *No-op*
N2: *No-op*
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
N3: *No-op*
N4: *No-op*
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	N1	N2	2	3	4	N3	N4	5	6			
2	-	1	N1	N2	2	3	4	N3	N4	5	6		
3	-	-	1	N1	N2	2	3	4	N3	N4	5	6	
4	-	-	-	1	N1	N2	2	3	4	N3	N4	5	6

b)

1: $R1 \leftarrow R2 + R3$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
N1: *No-op*
2: $R1 \leftarrow R1 + R2$
5: $R5 \leftarrow R5 + R7$
N2: *No-op*
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11
1	1	3	4	N1	2	5	N2	6			
2	-	1	3	4	N1	2	5	N2	6		
3	-	-	1	3	4	N1	2	5	N2	6	
4	-	-	-	1	3	4	N1	2	5	N2	6

c)

1: $R1 \leftarrow R2 + R3$
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
1	1	2	S	S	3	4	S	S	5	6			
2	-	1	2	S	S	3	4	S	S	5	6		
3	-	-	1	2	S	S	3	4	S	S	5	6	
4	-	-	-	1	2	S	S	3	4	S	S	5	6

d)

1: $R1 \leftarrow R2 + R3$
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6			
2	-	1	2	3	4	5	6		
3	-	-	1	2	3	4	5	6	
4	-	-	-	1	2	3	4	5	6

20. a)

1: $R1 \leftarrow R2 + R3$
N1: *No-op*
N2: *No-op*
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
N3: *No-op*
N4: *No-op*
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	N1	N2	2	3	4	N3	N4	5	6				
2	-	1	N1	N2	2	3	4	N3	N4	5	6			
3	-	-	1	N1	N2	2	3	4	N3	N4	5	6		
4	-	-	-	1	N1	N2	2	3	4	N3	N4	5	6	
5	-	-	-	-	1	N1	N2	2	3	4	N3	N4	5	6

b)

1: $R1 \leftarrow R2 + R3$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
N1: *No-op*
2: $R1 \leftarrow R1 + R2$
5: $R5 \leftarrow R5 + R7$
N2: *No-op*
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12
1	1	3	4	N1	2	5	N2	6				
2	-	1	3	4	N1	2	5	N2	6			
3	-	-	1	3	4	N1	2	5	N2	6		
4	-	-	-	1	3	4	N1	2	5	N2	6	
5	-	-	-	-	1	3	4	N1	2	5	N2	6

c)

1: $R1 \leftarrow R2 + R3$
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14
1	1	2	S	S	3	4	S	S	5	6				
2	-	1	2	S	S	3	4	S	S	5	6			
3	-	-	1	2	S	S	3	4	S	S	5	6		
4	-	-	-	1	2	S	S	3	4	S	S	5	6	
5	-	-	-	-	1	2	S	S	3	4	S	S	5	6

d)

1: $R1 \leftarrow R2 + R3$
2: $R1 \leftarrow R1 + R2$
3: $R2 \leftarrow R3 + R4$
4: $R5 \leftarrow R6 + R7$
5: $R5 \leftarrow R5 + R7$
6: $R6 \leftarrow R1 + R2$

Stage\Cycle	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6				
2	-	1	2	3	4	5	6			
3	-	-	1	2	3	4	5	6		
4	-	-	-	1	2	3	4	5	6	
5	-	-	-	-	1	2	3	4	5	6

21.

Stage\Cycle	1	2	3	4	5	6	7	8
1	1	2	3	S	S	10		
2	-	1	2	3	S	S	10	
3	-	-	1	2	3	S	S	10

22.

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	2	3	4	5	N	N	2	3	4	5	N	N		
2	-	1	2	3	4	5	N	N	2	3	4	5	N	N	
3	-	-	1	2	3	4	5	N	N	2	3	4	5	N	N

23.

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11
1	1	4	5	2	3	4	5	2	3		
2	-	1	4	5	2	3	4	5	2	3	
3	-	-	1	4	5	2	3	4	5	2	3

24. a)

1:*R1* ← *R1* + *R2*
2:*R3* ← *R3* + *R4*
3:*R5* ← *R1* + *R5*
4:JUMP 9
N1:*No-op*
N2:*No-op*
9:*R2* ← *R1* + *R3*

Stage\Cycle	1	2	3	4	5	6	7	8	9
1	1	2	3	4	N1	N2	9		
2	-	1	2	3	4	N1	N2	9	
3	-	-	1	2	3	4	N1	N2	9

b)

1:*R1* ← *R1* + *R2*
4:JUMP 9
2:*R3* ← *R3* + *R4*
3:*R5* ← *R1* + *R5*
9:*R2* ← *R1* + *R3*

Stage\Cycle	1	2	3	4	5	6	7
1	1	4	2	3	9		
2	-	1	4	2	3	9	
3	-	-	1	4	2	3	9

c)

1:*R1* ← *R1* + *R2*
2:*R3* ← *R3* + *R4*
3:*R5* ← *R1* + *R5*
4:JUMP 9
9:*R2* ← *R1* + *R3*

Stage\Cycle	1	2	3	4	5	6	7	8	9
1	1	2	3	4	S	S	9		
2	-	1	2	3	4	S	S	9	
3	-	-	1	2	3	4	S	S	9

25. a)

1:*R1* ← *R1* + *R2*
2:*R3* ← *R3* + *R4*
N1:*No-op*
3:*R5* ← *R1* + *R5*
4:JUMP 9
N2:*No-op*
N3:*No-op*
9:*R2* ← *R1* + *R3*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11
1	1	2	N1	3	4	N2	N3	9			
2	-	1	2	N1	3	4	N2	N3	9		
3	-	-	1	2	N1	3	4	N2	N3	9	
4	-	-	-	1	2	N1	3	4	N2	N3	9

b)

1:*R1* ← *R1* + *R2*
4:JUMP 9
2:*R3* ← *R3* + *R4*
3:*R5* ← *R1* + *R5*
N:*No-op*
9:*R2* ← *R1* + *R3*

Stage\Cycle	1	2	3	4	5	6	7	8	9
1	1	4	2	3	N	9			
2	-	1	4	2	3	N	9		
3	-	-	1	4	2	3	N	9	
4	-	-	-	1	4	2	3	N	9

c)

1:*R1* ← *R1* + *R2*
2:*R3* ← *R3* + *R4*
3:*R5* ← *R1* + *R5*
4:JUMP 9
9:*R2* ← *R1* + *R3*

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11
1	1	2	3	S	4	S	S	9			
2	-	1	2	3	S	4	S	S	9		
3	-	-	1	2	3	S	4	S	S	9	
4	-	-	-	1	2	3	S	4	S	S	9

26. a)

```
 1: $R1 \leftarrow 3$
 2: $R2 \leftarrow R2 + R3$
 3: $R3 \leftarrow R3 + R4$
 4: $R4 \leftarrow R1 + R2$
 5: $R1 \leftarrow R1 - 1$
N1: No-op
 6: IF ($R1 \neq 0$) THEN GOTO 2
N2: No-op
N3: No-op
 7: $R5 \leftarrow R6 + R7$
 8: $R6 \leftarrow R7 + R8$
```

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	1	2	3	4	5	N1	6	N2	N3	2	3	4	5	N1	6	N2	N3	2	3	4	5	N1	6	N2	N3	7	8		
2	-	1	2	3	4	5	N1	6	N2	N3	2	3	4	5	N1	6	N2	N3	2	3	4	5	N1	6	N2	N3	7	8	
3	-	-	1	2	3	4	5	N1	6	N2	N3	2	3	4	5	N1	6	N2	N3	2	3	4	5	N1	6	N2	N3	7	8

b)

```
1: $R1 \leftarrow 3$
N: No-op
5: $R1 \leftarrow R1 - 1$
2: $R2 \leftarrow R2 + R3$
6: IF ($R1 \neq 0$) THEN GOTO 2
3: $R3 \leftarrow R3 + R4$
4: $R4 \leftarrow R1 + R2$
7: $R5 \leftarrow R6 + R7$
8: $R6 \leftarrow R7 + R8$
```

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1	1	N	5	2	6	3	4	5	2	6	3	4	5	2	6	3	4	7	8		
2	-	1	N	5	2	6	3	4	5	2	6	3	4	5	2	6	3	4	7	8	
3	-	-	1	N	5	2	6	3	4	5	2	6	3	4	5	2	6	3	4	7	8

c)

```
1: $R1 \leftarrow 3$
2: $R2 \leftarrow R2 + R3$
3: $R3 \leftarrow R3 + R4$
4: $R4 \leftarrow R1 + R2$
5: $R1 \leftarrow R1 - 1$
6: IF ($R1 \neq 0$) THEN GOTO 2
7: $R5 \leftarrow R6 + R7$
8: $R6 \leftarrow R7 + R8$
```

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
1	1	2	3	4	5	S	6	S	S	2	3	4	5	S	6	S	S	2	3	4	5	S	6	S	S	7	8		
2	-	1	2	3	4	5	S	6	S	S	2	3	4	5	S	6	S	S	2	3	4	5	S	6	S	S	7	8	
3	-	-	1	2	3	4	5	S	6	S	S	2	3	4	5	S	6	S	S	2	3	4	5	S	6	S	S	7	8

27.

1:*R1* ← 3
2:*R2* ← *R2* + *R3*
3:*R3* ← *R3* + *R4*
4:*R4* ← *R1* + *R2*
5:*R1* ← *R1* – 1
N:*No-op*
6:IF (*R1* ≠ 0) THEN GOTO 2
7:*R5* ← *R6* + *R7*
8:*R6* ← *R7* + *R8*

Underlined instructions are annulled

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	1	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8		
2	-	1	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8	
3	-	-	1	2	3	4	5	N	6	$\underline{7}$	$\underline{8}$	2	3	4	5	N	6	$\underline{7}$	$\underline{8}$	2	3	4	5	N	6	7	8

28. a)

1:*R1* ← 3
2:*R2* ← *R2* + *R3*
3:*R3* ← *R3* + *R4*
4:*R4* ← *R1* + *R2*
5:*R1* ← *R1* – 1
N:*No-op*
6:IF (*R1* ≠ 0) THEN GOTO 2
7:*R5* ← *R6* + *R7*
8:*R6* ← *R7* + *R8*

Underlined instructions are annulled

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1	1	2	3	4	5	N	6	2	3	4	5	N	6	2	3	4	5	N	6	2	3	7	8		
2	-	1	2	3	4	5	N	6	2	3	4	5	N	6	2	3	4	5	N	6	2	3	7	8	
3	-	-	1	2	3	4	5	N	6	2	3	4	5	N	6	2	3	4	5	N	6	$\underline{2}$	$\underline{3}$	7	8

b)

1:*R1* ← 3
2:*R2* ← *R2* + *R3*
3:*R3* ← *R3* + *R4*
4:*R4* ← *R1* + *R2*
5:*R1* ← *R1* – 1
N:*No-op*
6:IF (*R1* ≠ 0) THEN GOTO 2
7:*R5* ← *R6* + *R7*
8:*R6* ← *R7* + *R8*

Underlined instructions are annulled

Stage\Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	1	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8		
2	-	1	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8	2	3	4	5	N	6	7	8	
3	-	-	1	2	3	4	5	N	6	$\underline{7}$	$\underline{8}$	2	3	4	5	N	6	$\underline{7}$	$\underline{8}$	2	3	4	5	N	6	7	8

Chapter 12

1. a) diameter = 1; bandwidth = 500 Mb/s; bisection bandwidth = 500 Mb/s
   b) diameter = 32; bandwidth = 32000 Mb/s; bisection bandwidth = 1000 Mb/s
   c) diameter = 10; bandwidth = 31000 Mb/s; bisection bandwidth = 500 Mb/s
   d) diameter = 16; bandwidth = 56000 Mb/s; bisection bandwidth = 4000 Mb/s
   e) diameter = 8; bandwidth = 64000 Mb/s; bisection bandwidth = 8000 Mb/s
   f) diameter = 6; bandwidth = 96000 Mb/s; bisection bandwidth = 16000 Mb/s
   g) diameter = 1; bandwidth = 1008000 Mb/s; bisection bandwidth = 512000 Mb/s

2. a) diameter = 1; bandwidth = 10 Mb/s; bisection bandwidth = 10 Mb/s
   b) diameter = 8; bandwidth = 160 Mb/s; bisection bandwidth = 20 Mb/s
   c) diameter = 6; bandwidth = 150 Mb/s; bisection bandwidth = 10 Mb/s
   d) diameter = 8; bandwidth = 240 Mb/s; bisection bandwidth = 40 Mb/s
   e) diameter = 4; bandwidth = 320 Mb/s; bisection bandwidth = 80 Mb/s
   f) diameter = 4; bandwidth = 320 Mb/s; bisection bandwidth = 80 Mb/s
   g) diameter = 1; bandwidth = 1200 Mb/s; bisection bandwidth = 640 Mb/s

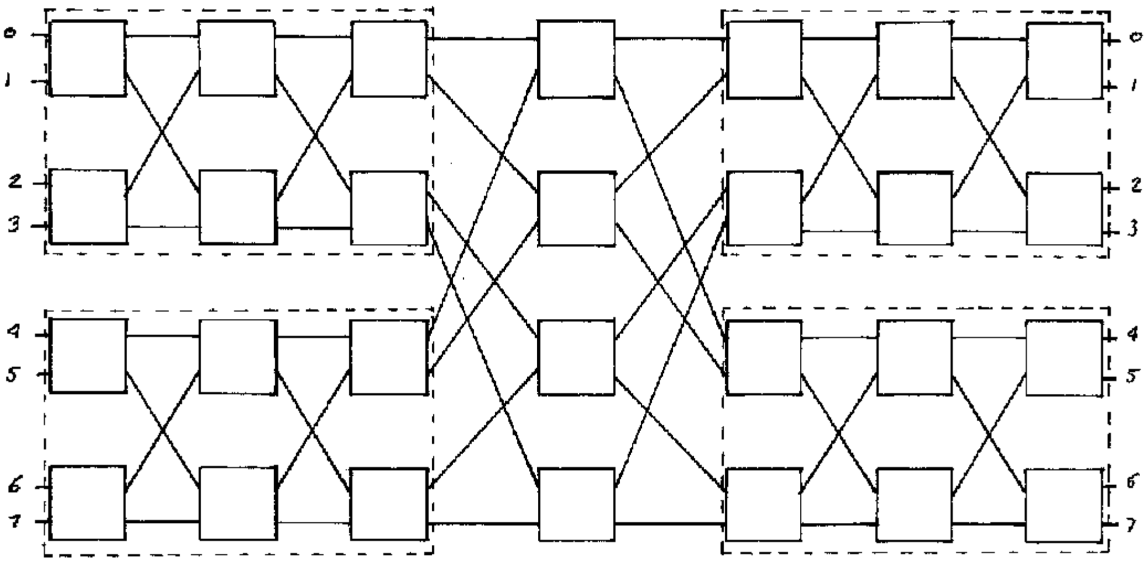
3. d(tree) = $2*\lfloor \lg n \rfloor$, d(ring) = $n/2$; $2*\lfloor \lg n \rfloor < n/2$ for $13 \le n \le 15$ and $n \ge 17$

4. d(mesh) = $\sqrt{n}$, d(tree) = $2*\lfloor \lg n \rfloor$; $\sqrt{n} < 2*\lfloor \lg n \rfloor$ for $2 \le n \le 196$

5. b(hc) = $(n/2) * \lg n * l$, b(cc) = $(\lfloor n/2 \rfloor * \lceil n/2 \rceil) * l$; $(n/2) * \lg n * l < (\lfloor n/2 \rfloor * \lceil n/2 \rceil) * l$ for $n \ge 5$

6. a) bb(mesh) = $2\sqrt{16} * 100$ Mb/s = 800 Mb/s = $(\lfloor 16/2 \rfloor * \lceil 16/2 \rceil) * l_{cc}$; $l_{cc} =$ 12.5 Mb/s
   b) 800 Mb/s = $16/2 * l_{hc}$; l_{hc} = 100 Mb/s
   c) 800 Mb/s = $2\lceil \sqrt{16} / 2 \rceil * l_{mesh}$; l_{mesh} = 200 Mb/s

7. The hardware complexity is $O(n^2)$.



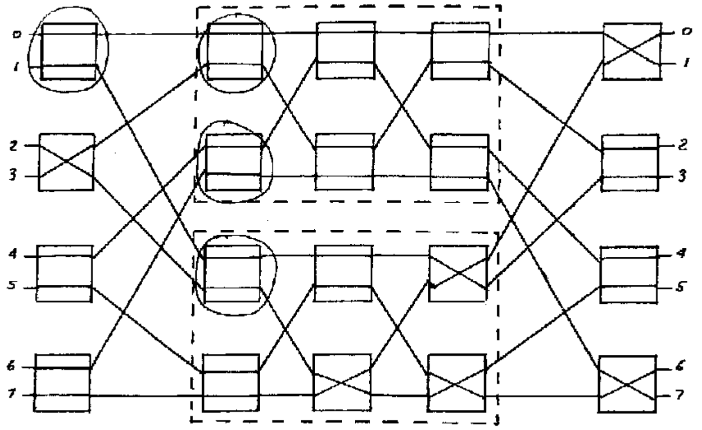
8. $$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 0 & 6 & 2 & 4 & 7 & 5 & 3 \end{pmatrix}$$

9. $$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 6 & 7 & 5 & 2 & 1 & 0 & 3 \end{pmatrix}$$

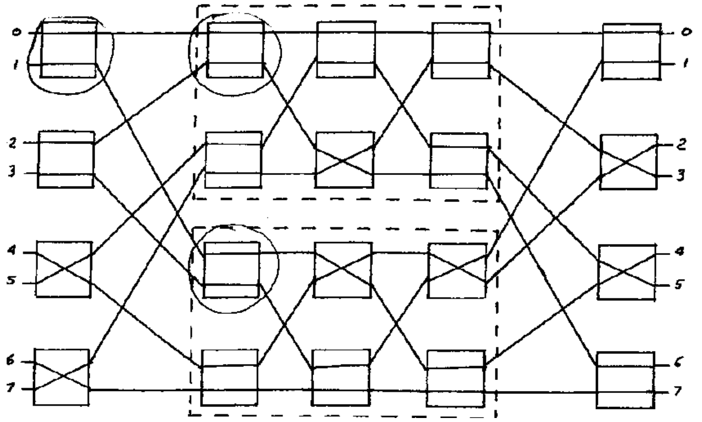
10. $$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 4 & 3 & 7 & 2 & 6 & 0 & 1 & 5 \end{pmatrix}$$

11. Circled switches were set randomly.

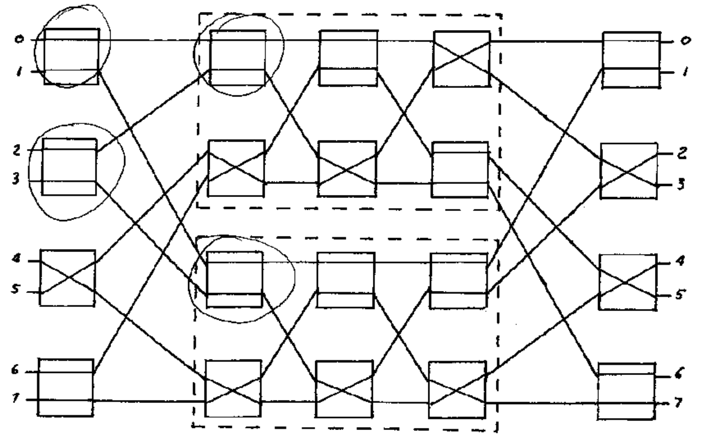
a)



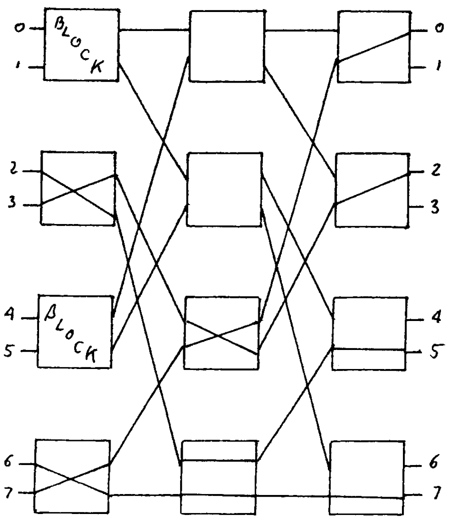
b)



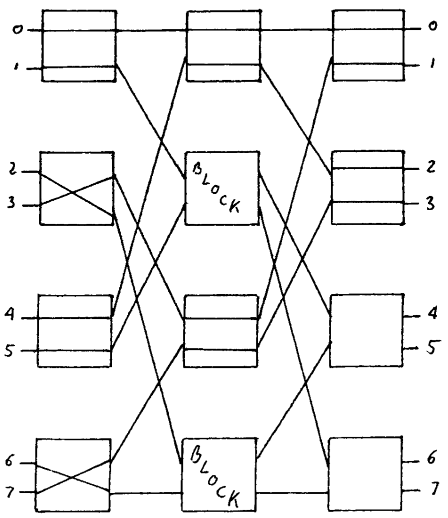
c)



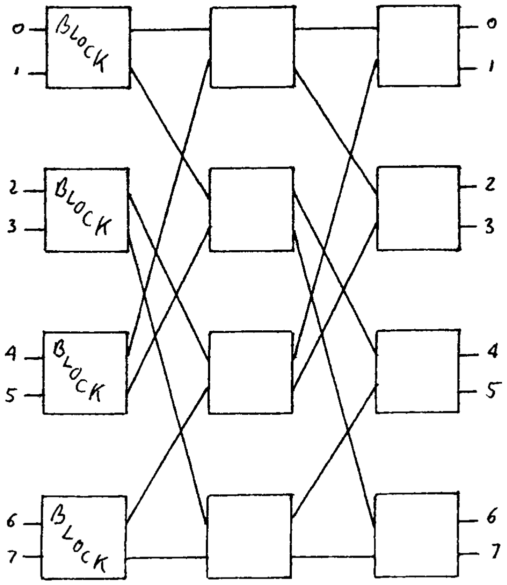
12. a)



b)



c)



13. a)

Module	Address Range	Addresses
M_0	0 to 8M – 1	00 0XXX XXXX XXXX XXXX XXXX XXXX
M_1	8M to 16M – 1	00 1XXX XXXX XXXX XXXX XXXX XXXX
M_2	16M to 24M – 1	01 0XXX XXXX XXXX XXXX XXXX XXXX
M_3	24M to 32M – 1	01 1XXX XXXX XXXX XXXX XXXX XXXX
M_4	32M to 40M – 1	10 0XXX XXXX XXXX XXXX XXXX XXXX
M_5	40M to 48M – 1	10 1XXX XXXX XXXX XXXX XXXX XXXX
M_6	48M to 56M – 1	11 0XXX XXXX XXXX XXXX XXXX XXXX
M_7	56M to 64M – 1	11 1XXX XXXX XXXX XXXX XXXX XXXX

b)

Module	Address Range	Addresses
M_0	0 to 32M – 1	00X XXXX XXXX XXXX XXXX XXXX XXXX
M_1	32M to 64M – 1	01X XXXX XXXX XXXX XXXX XXXX XXXX
M_2	64M to 96M – 1	10X XXXX XXXX XXXX XXXX XXXX XXXX
M_3	96M to 128M – 1	11X XXXX XXXX XXXX XXXX XXXX XXXX

c)

Module	Address Range	Addresses
M_0	0 to 4M – 1	0 00XX XXXX XXXX XXXX XXXX XXXX
M_1	4M to 8M – 1	0 01XX XXXX XXXX XXXX XXXX XXXX
M_2	8M to 12M – 1	0 10XX XXXX XXXX XXXX XXXX XXXX
M_3	12M to 16M – 1	0 11XX XXXX XXXX XXXX XXXX XXXX
M_4	16M to 20M – 1	1 00XX XXXX XXXX XXXX XXXX XXXX
M_5	20M to 24M – 1	1 01XX XXXX XXXX XXXX XXXX XXXX
M_6	24M to 28M – 1	1 10XX XXXX XXXX XXXX XXXX XXXX
M_7	28M to 32M – 1	1 11XX XXXX XXXX XXXX XXXX XXXX

14. a)

Module	Address Range	Addresses
M_0	i mod 8 = 0 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X000
M_1	i mod 8 = 1 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X001
M_2	i mod 8 = 2 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X010
M_3	i mod 8 = 3 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X011
M_4	i mod 8 = 4 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X100
M_5	i mod 8 = 5 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X101
M_6	i mod 8 = 6 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X110
M_7	i mod 8 = 7 ($0 \le i \le 64M - 1$)	XX XXXX XXXX XXXX XXXX XXXX X111

b)

Module	Address Range	Addresses
M_0	i mod 4 = 0 ($0 \le i \le 128M - 1$)	XXX XXXX XXXX XXXX XXXX XXXX XX00
M_1	i mod 4 = 1 ($0 \le i \le 128M - 1$)	XXX XXXX XXXX XXXX XXXX XXXX XX01
M_2	i mod 4 = 2 ($0 \le i \le 128M - 1$)	XXX XXXX XXXX XXXX XXXX XXXX XX10
M_3	i mod 4 = 3 ($0 \le i \le 128M - 1$)	XXX XXXX XXXX XXXX XXXX XXXX XX11

c)

Module	Address Range	Addresses
M_0	i mod 8 = 0 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X000
M_1	i mod 8 = 1 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X001
M_2	i mod 8 = 2 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X010
M_3	i mod 8 = 3 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X011
M_4	i mod 8 = 4 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X100
M_5	i mod 8 = 5 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X101
M_6	i mod 8 = 6 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X110
M_7	i mod 8 = 7 ($0 \le i \le 32M - 1$)	X XXXX XXXX XXXX XXXX XXXX X111

15.

Action	Result	Cache 0	Cache 1	Cache 2	Cache 3	Shared
P0 read	Read miss	1000:K_1 E				1000:K_1
P2 write	Write miss	1000:XX I		1000:K_2 M		1000:K_1
P1 read	Read miss		1000:K_2 S	1000:K_2 S		1000:K_2
P0 write	Write miss	1000:K_3 M	1000:XX I	1000:XX I		1000:K_2
P3 read	Read miss	1000:K_3 S			1000:K_3 S	1000:K_3
P1 write	Write miss	1000:XX I	1000:K_4 M		1000:XX I	1000:K_3
P1 read	Read hit		1000:K_4 M			1000:K_4

16.

Action	**Result**	**Cache 0**	**Cache 1**	**Cache 2**	**Cache 3**	**Shared**
P2 write	Write miss			1100:K_1 M		1100:K_0
P1 read	Read miss		1100:K_1 S	1100:K_1 S		1100:K_1
P3 write	Write miss		1100:XX I	1100:XX I	1100:K_2 M	1100:K_1
P2 read	Read miss			1100:K_2 S	1100:K_2 S	1100:K_2
P0 read	Read miss	1100:K_2 S		1100:K_2 S	1100:K_2 S	1100:K_2
P1 write	Write miss	1100:XX I	1100:K_3 M	1100:XX I	1100:XX I	1100:K_2
P2 write	Write miss		1100:XX I	1100:K_2 M		1100:K_3

17. Data dependencies: 1 → 3 (A); 2 → 3 (D); 2 → 4 (D)
Data anti-dependencies 3 → 4 (A)
Data output dependencies: 1 → 4 (A)

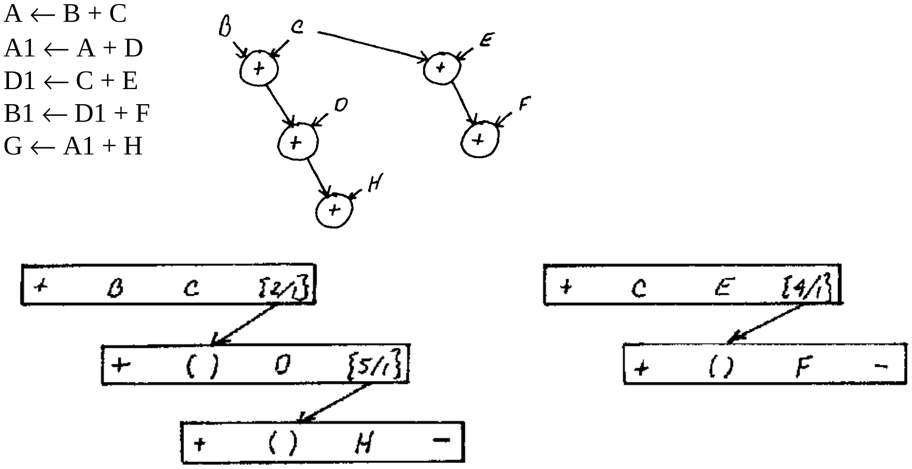
18. Data dependencies: 1 → 2 (A); 2 → 5 (A); 3 → 4 (D)
Data anti-dependencies 2 → 4 (B); 1 → 6 (C); 2 → 3 (D) ; 3 → 6 (F)
Data output dependencies: 1 → 2 (A)

19. Data dependencies: 1 → 2 (A); 1 → 3 (A); 2 → 3 (B); 2 → 4 (B); 2 → 7 (B); 3 → 5 (C);
3 → 7 (C); 4 → 6 (A); 6 → 8 (A); 7 → 8 (E)
Data anti-dependencies 1 → 2 (B); 1 → 3 (C); 2 → 3 (C); 2 → 4 (A); 2 → 6 (A); 3 → 4 (A);
3 → 6 (A); 4 → 5 (D); 5 → 7 (E); 6 → 8 (F)
Data output dependencies: 1 → 4 (A); 1 → 6 (A); 4 → 6 (A);

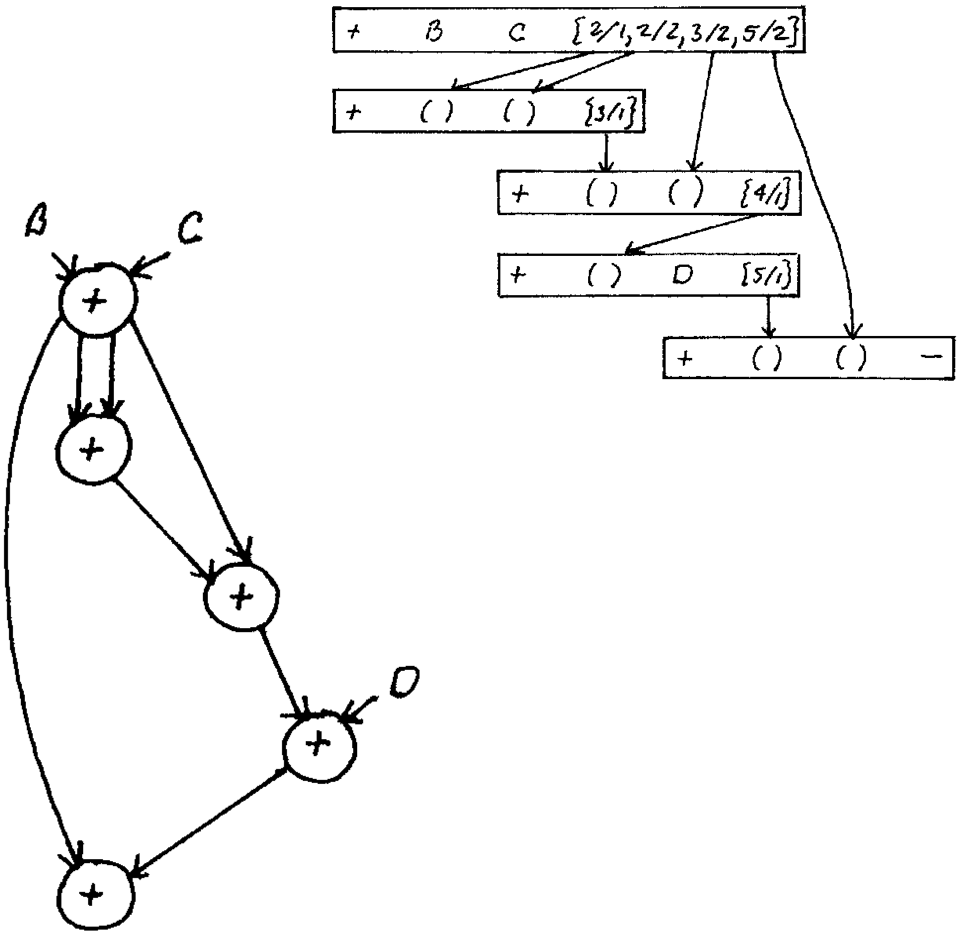
20.

i	*j*	*k*	*C*
1..3	1..3	1	$\begin{bmatrix} 2 & 0 & 1 \\ 2 & 0 & 2 \\ 0 & 2 & 0 \end{bmatrix}$
1..3	1..3	2	$\begin{bmatrix} 2+0 & 0+0 & 1+2 \\ 2+2 & 0+1 & 2+0 \\ 0+4 & 2+4 & 0+0 \end{bmatrix}$
1..3	1..3	3	$\begin{bmatrix} 2+2 & 0+1 & 3+0 \\ 4+2 & 1+4 & 2+1 \\ 4+2 & 6+0 & 0+4 \end{bmatrix}$
-	-	-	$\begin{bmatrix} 4 & 1 & 3 \\ 6 & 5 & 3 \\ 6 & 6 & 4 \end{bmatrix}$

21.
1. A ← B + C
2. A1 ← A + D
3. D1 ← C + E
4. B1 ← D1 + F
5. G ← A1 + H



22.
1. A ← B + C
2. B1 ← A + A
3. C1 ← B1 + A
4. B2 ← C1 + D
5. B3 ← B2 + A


+ B C {2/1, 2/2, 3/2, 5/2}
+ ( ) ( ) {3/1}
+ ( ) ( ) {4/1}
+ ( ) D {5/1}
+ ( ) ( ) –
B
C
D
+