

Prob.	Points:
1	10
2	10
3	10
4	10
5	10
6	10
7	10
8	10
9	10
10	10
11	10
12	10
13	10
14	10
15	10
16	10
17	10
18	10
19	10
20	10
21	10
22	10
23	10
24	10
25	10
26	10
27	10
28	10
29	10
30	10
31	10
32	10
33	10
34	10
35	10
36	10
37	10
38	10
39	10
40	10
41	10
42	10
43	10
44	10
45	10
46	10
47	10
48	10
49	10
50	10

1		(10)
2		(10)
3		(10)
4		(14)
5		(14)
6		(15)
7		(15)
8		(12)

Total:

(100)

[illegible]

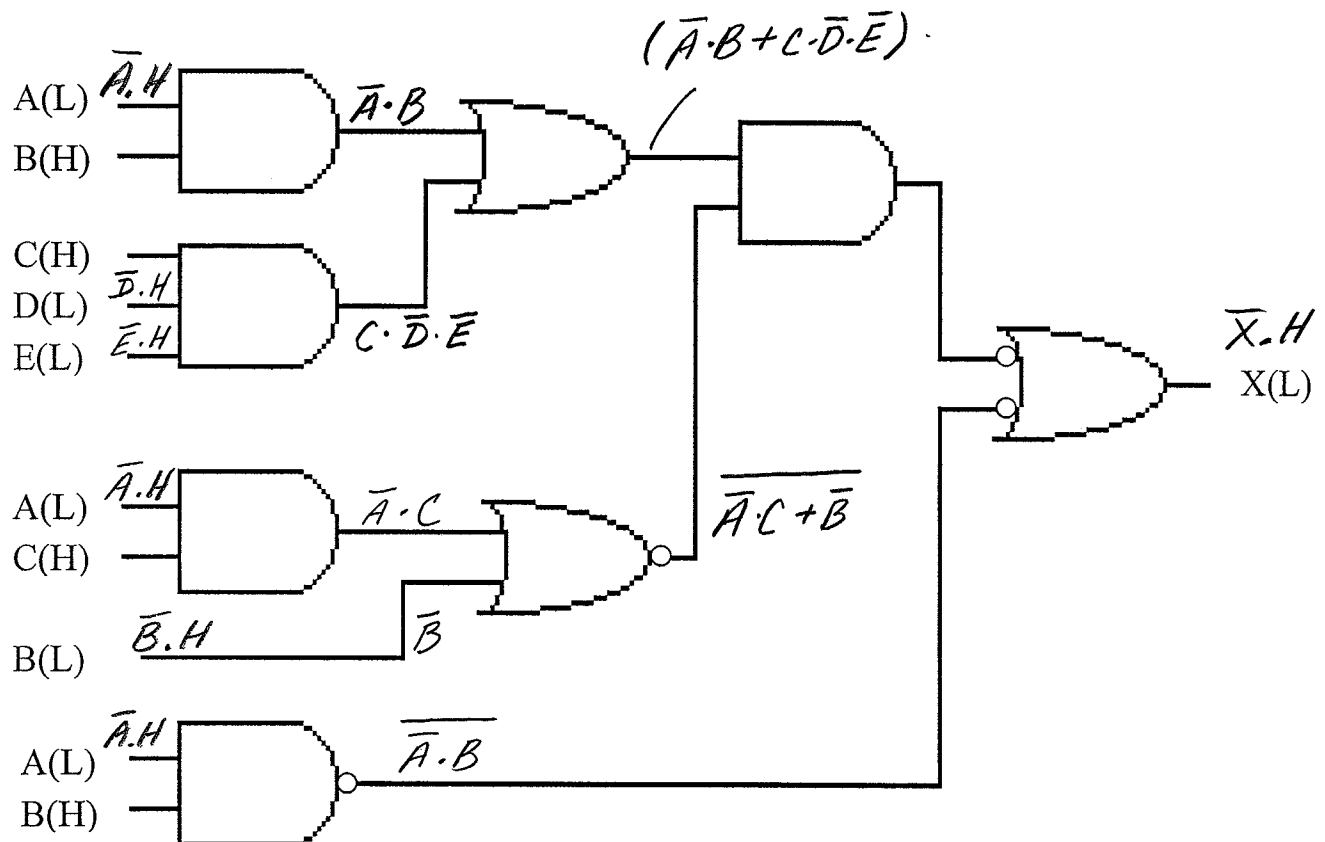
Remember to show ALL work here and in EVERY problem on this exam.

[10%] 1. Circuit Analysis

What is the logic equation for X in the given circuit? Do not simplify or transform it into an SOP or POS form. Leave the logic expression as it is after analysis. Also, draw the intermediate expression at the input to each gate.

- Notation reminder: $A(H)$ is the same as $A.H$
- Boolean expression answers must be in **lexical order**, i.e., /A before A, A before B, etc.

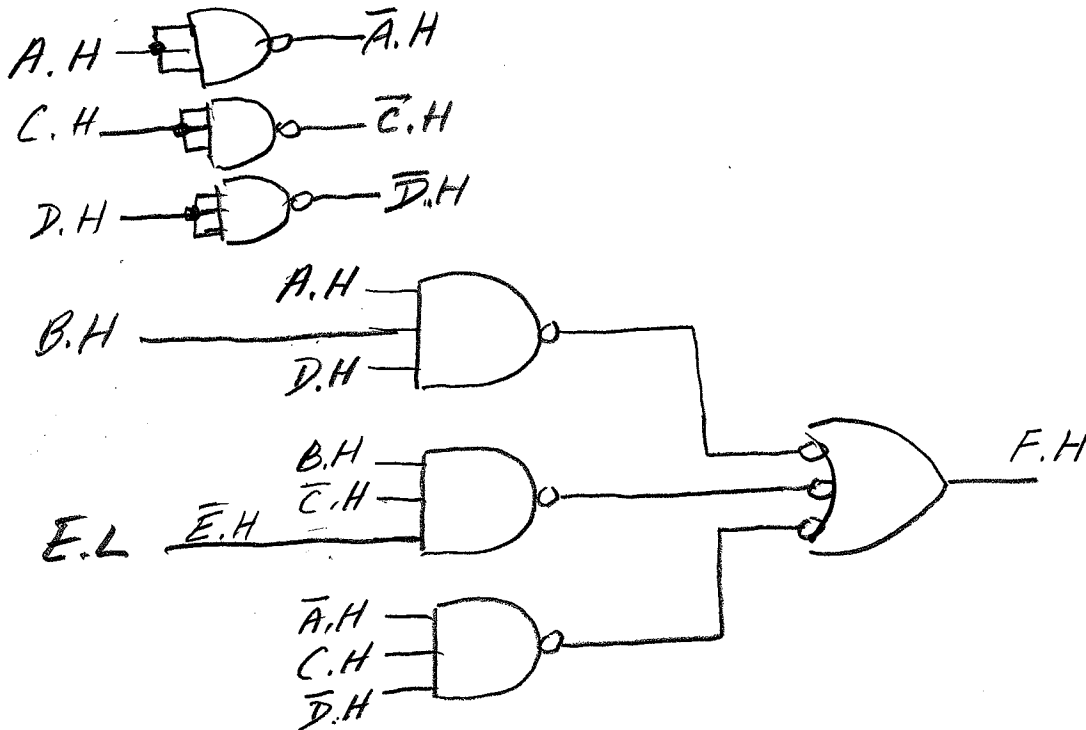
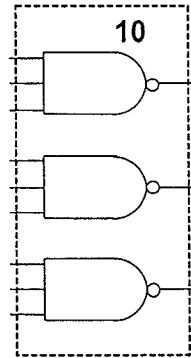
EQUATION: $X = (\bar{A} \cdot B + C \cdot \bar{D} \cdot \bar{E}) \cdot (\bar{A} \cdot C + \bar{B}) + \bar{A} \cdot B$

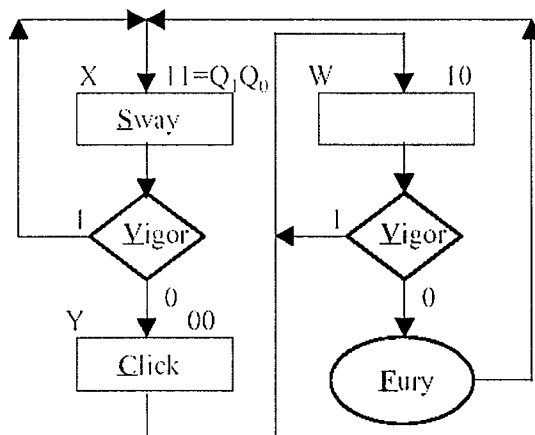


[10%] 2. Circuit Synthesis

Draw a mixed-logic circuit diagram (with the minimum number of gates) to **directly implement** the below equation. All inputs and the output can be of any activation-level desired. Be sure to **specify the desired activation levels**. Do **not** simplify this equation. You may **only use** gates available on 74HC10 chips (shown). Use as many 74HC10 chips as you need, but use the minimum number required to solve this problem.

$$F = A * B * D + B * /C * /E + /A * C * /D$$

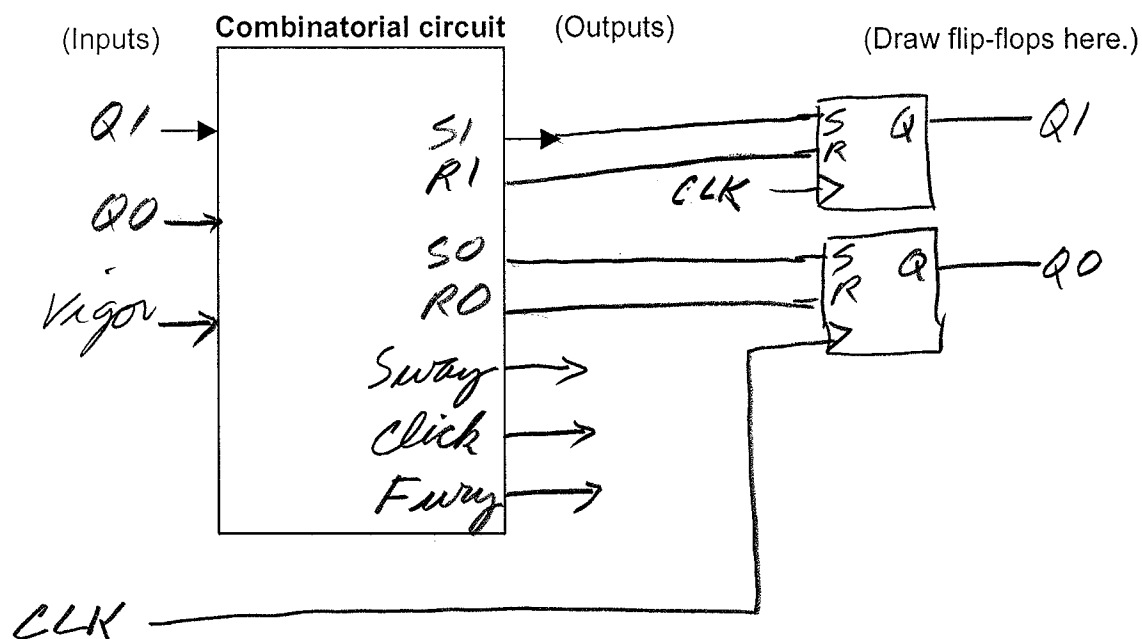


[10%] 3. Implementation of an ASM chart using clocked S-R FFs**S-R characteristic table:**

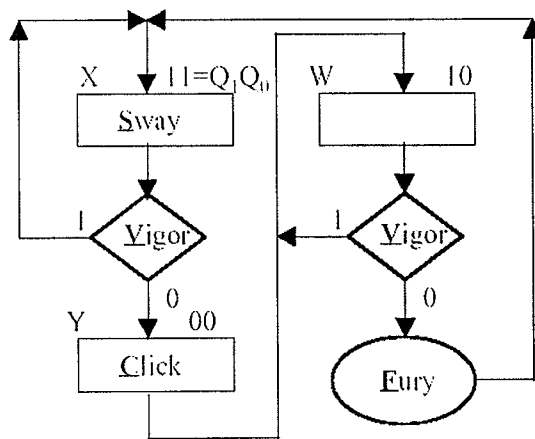
S	R	Q	Q+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	?

(a) Given the above ASM chart, complete the following block diagram of its implementation using the minimum number of **clocked** S-R flip-flops: (2%)

- Determine how many clocked S-R flip-flops that are needed.
- Draw in all the inputs and outputs of the combinational circuit.
- Make all necessary connections to complete the block diagram (between the combinational circuit and the clocked S-R flip-flops).



3. (continued) (ASM chart is repeated here for your convenience)



Excitation Table

Q	Q^+	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

S-R characteristic table:

S	R	Q	Q^+
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	?
1	1	1	?

- (b) Finish the implementation of the ASM by determining the minimum sum-of-products (MSOP) logic expressions for all the output signals: (8%)

(If necessary, use the bottom/back of the previous page to do your work.)

Q_1	Q_0	V	Q_1^+	Q_0^+	S	R	S	C	F
0	0	0	1	0	1	0	0	1	0
0	0	1	1	0	1	0	0	1	0
0	1	0	X	X	X	X	X	X	X
0	1	1	X	X	X	X	X	X	X
1	0	0	1	1	X	0	0	0	1
1	0	1	1	0	X	0	0	0	0
1	1	0	0	0	0	1	1	0	0
1	1	1	1	1	X	0	1	0	0

$$S_1 = \overline{Q_1}$$

$$R_1 = Q_0 \overline{V}$$

$$S_0 = Q_1 \overline{Q_0} \overline{V}$$

$$R_0 = Q_0 \overline{V}$$

$$S = Q_0$$

$$C = \overline{Q_1}$$

$$F = Q_1 \overline{Q_0} \overline{V}$$

$Q_1 \backslash Q_0$

$Q_0 \backslash Q_1$	0	1
0	0	0
1	0	0
X	X	0
X	1	1

R_1

$Q_1 \backslash Q_0$

$Q_0 \backslash Q_1$	0	1
0	0	1
1	0	0
X	X	X
X	0	0

S_0

$Q_1 \backslash Q_0$

$Q_0 \backslash Q_1$	0	1
0	X	0
1	X	X
X	X	0
X	1	1

R_0

$Q_1 \backslash Q_0$

$Q_0 \backslash Q_1$	0	1
0	0	1
1	0	0
X	X	0
X	0	0

- [14%] 4. Assume the below has already been run, and the code that follows in a, b, c, d, e, f, g will follow. Hand assemble the following instructions and fill in the blanks. EA is the 16-bit effective address. If there is no effective address, write none. (Use the G-CPU instruction set attached to this test).

	ORG	\$0032		ORG	\$0048
Data0	DC.B	\$66	Data6	DC.B	\$3E
	ORG	\$0028		ORG	\$0058
Data1	DC.B	\$A3	Data7	DC.B	\$9E
	ORG	\$002A		ORG	\$2A42
Data2	DC.B	\$74	Data8	DC.B	\$99, \$AC
	ORG	\$0038			
Data3	DC.B	\$EC		ORG	\$0000
	ORG	\$003A		LDX	#\$000A
Data4	DC.B	\$EC		LDAA	#3
	ORG	\$0040		LDAB	#37
Data5	DC.B	\$AB		SUM_AB	

$X = \$000A \leftarrow$
 $A = \$03 \leftarrow$
 $B = \$25$
 $B = \$28 \leftarrow$

	ADDRESS INSTRUCTION	HEX ADDRESS	HEX VALUE
a) $48 = \$30$	\$0008 STAB 48, X $X = 000A$	0008	12
	EA = $\$003A$	0009	30
	Value stored = $\$28$	000A	
		000B	
b) $40 = \$28$	\$0008 LDAA 40 $LDAA \$28$	0008	04
	EA = $\$0028$	0009	28
	A = $\$A3$	000A	00
		000B	
c)	\$0008 LDX #Data8 $LDX \#\$2A42$	0008	08
	EA = $\$0009$	0009	42
	X = $\$2A42$	000A	2A
		000B	
d)	\$0008 TBA	0008	01
	EA = $no EA$	0009	
	A = 28	000A	
		000B	
e)	\$0008 LDAB Data8 $LDAB \$2A42$	0008	05
	EA = $\$2A42$	0009	42
	B = $\$99$	000A	2A
		000B	
f)	\$0008 BN \$EF	0008	22
	EA = <u>NONE</u>	0009	EF
	PC after this instruction = $\$000A$	000A	
		000B	
g)	\$0008 BNE \$EF	0008	21
	EA = <u>NONE</u>	0009	EF
	PC after this instruction = $\$00EF$	000A	
		000B	

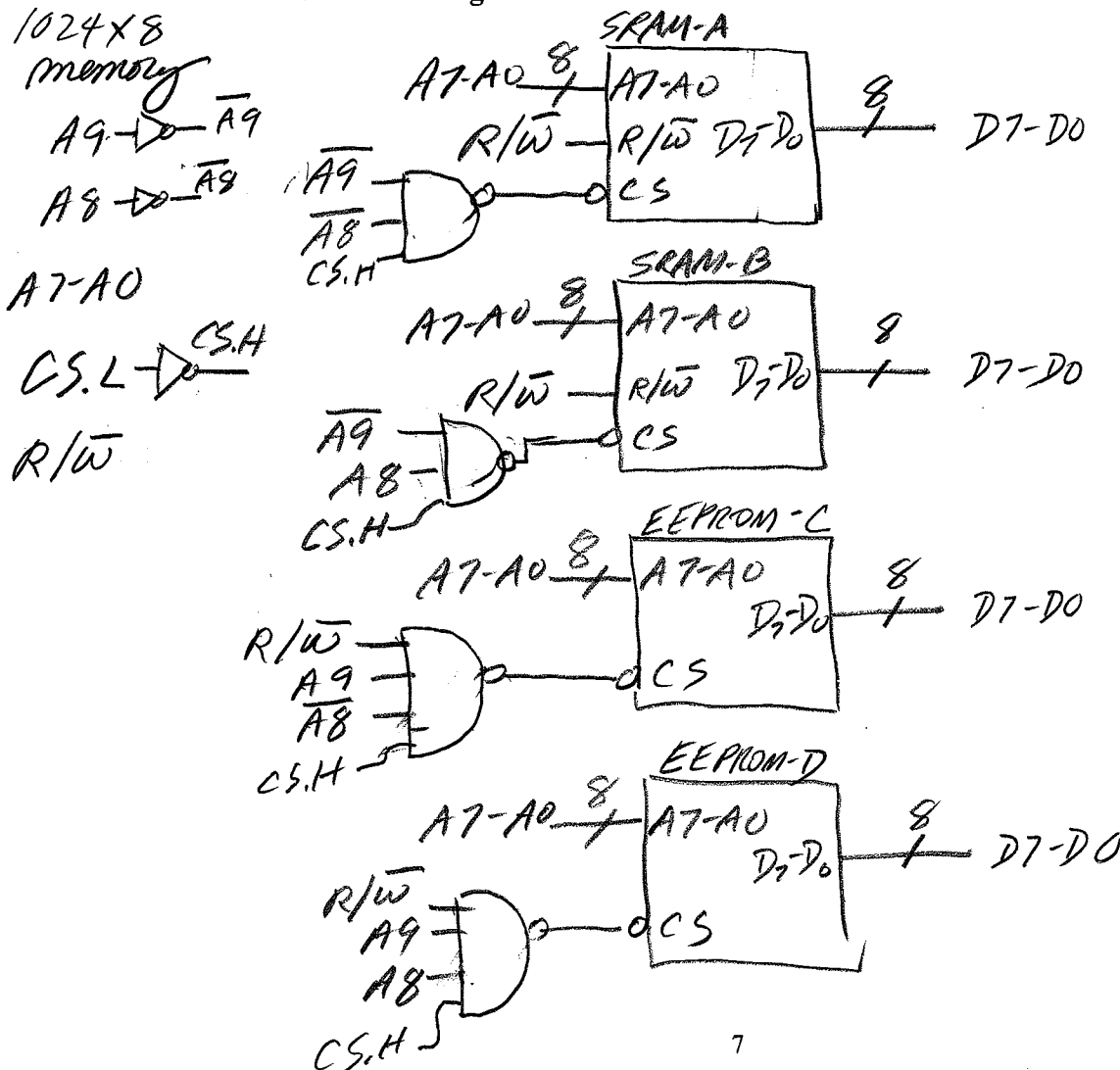
[14%] 5. EEPROM and SRAM

Given as many 256x8 EEPROM chips and 256x8 static RAM chips as needed, design a 1024x8 memory module (with a CS) that has 512x8 of RAM at the lowest addresses and 512x8 of EEPROM at the highest addresses. The 512x8 of RAM must start at address 0 and the first address of the 512x8 of EEPROM must immediately follow the last RAM address. Add the minimum number of additional components required. Make sure the EEPROM is NEVER enabled during a write cycle and the RAM is enabled for both read and write cycles. The EEPROM and SRAM devices have active low CS and the SRAM devices have a R/W control signal.

- (2%) a) What is the address range for each of the memory components (in binary and in hex)?

SRAM-A $00_2 0000_2 0000 - 00_2 1111_2 1111$ \$000-\$0FF
 SRAM-B $01_2 0000_2 0000 - 01_2 1111_2 1111$ \$100-\$1FF
 EEPROM-C $10_2 0000_2 0000 - 10_2 1111_2 1111$ \$200-\$2FF
 EEPROM-D $11_2 0000_2 0000 - 11_2 1111_2 1111$ \$300-\$3FF

- (12%) b) Design the required memory device below. Make sure you show the memory module's inputs and outputs and all the individual memory component devices. Use labels instead of wires in the design.



[15%] **6. HAND Assembly** The following program finds all occurrences of \$37 in a table with an \$FF end of table marker and replaces them with \$AA. It uses the G-CPU instruction set attached to this test. Hand assemble the program in the space given.

	Address (Hex)	Data (Hex)		Program
TABLE	0100	10		ORG \$100
	0101	37	Table	DC.B \$10,\$37,\$CC,\$37,\$15
	0102	CC	EOT	DC.B \$FF
	0103	37	NEW	EQU \$AA
	0104	15	OLD	EQU \$C8
EOT	0105	FF		ORG \$200
			STRT	LDX #Table
STRT	0200	08	LOOP	LDAB EOT
	0201	00		COMB
	0202	01		LDAA 0,X
LOOP	0203	05		SUM_BA
	0204	05		BEQ QUIT
	0205	01		LDAA 0,X
	0206	1B		LDAB #OLD <i>LDAB #C8</i>
	0207	0C		SUM_BA
	0208	00		BNE NOSW
	0209	14		LDAA #NEW <i>LDAA #AA</i>
	020A	20		STAA 0,X
	020B	1A	NOSW	INX
	020C	0C		BNE LOOP
	020D	00	QUIT	BEQ QUIT
	020E	03		
	020F	C8		
	0210	14		
	0211	21		
	0212	17		
	0213	02		
	0214	AA		
	0215	10		
	0216	00		
NOSW	0217	30		
	0218	21		
	0219	03		
QUIT	021A	20		
	021B	1A		
	021C			

Labels	Instructions	Comments
TAB1	ORG \$3000	} already in memory beginning of SRAM 1200 decimal }
TAB2	ORG \$7000	
	ORG \$4000	
COUNT	DC.B 200	
	ORG \$0000	COUNT EQU \$4000
	LDX #TAB1	LDAA #200
	LDY #TAB2	STAA COUNT
LOOP	LDAA 0,X	Load next byte
	BN SKIP	} if positive, store in TAB2 and INY
	STAA 0,Y	
	INY	next byte in TAB1
SKIP	INX	
	LDAA COUNT	Load counter
	LDAB #\$FF	} subtract 1
	SUB-BA	
	STAA COUNT	store counter
	BNE LOOP	
FIN	BEQ FIN	

[12%] 8. GCPU Instruction Design (See the G-CPU Next State Table attached to this test)

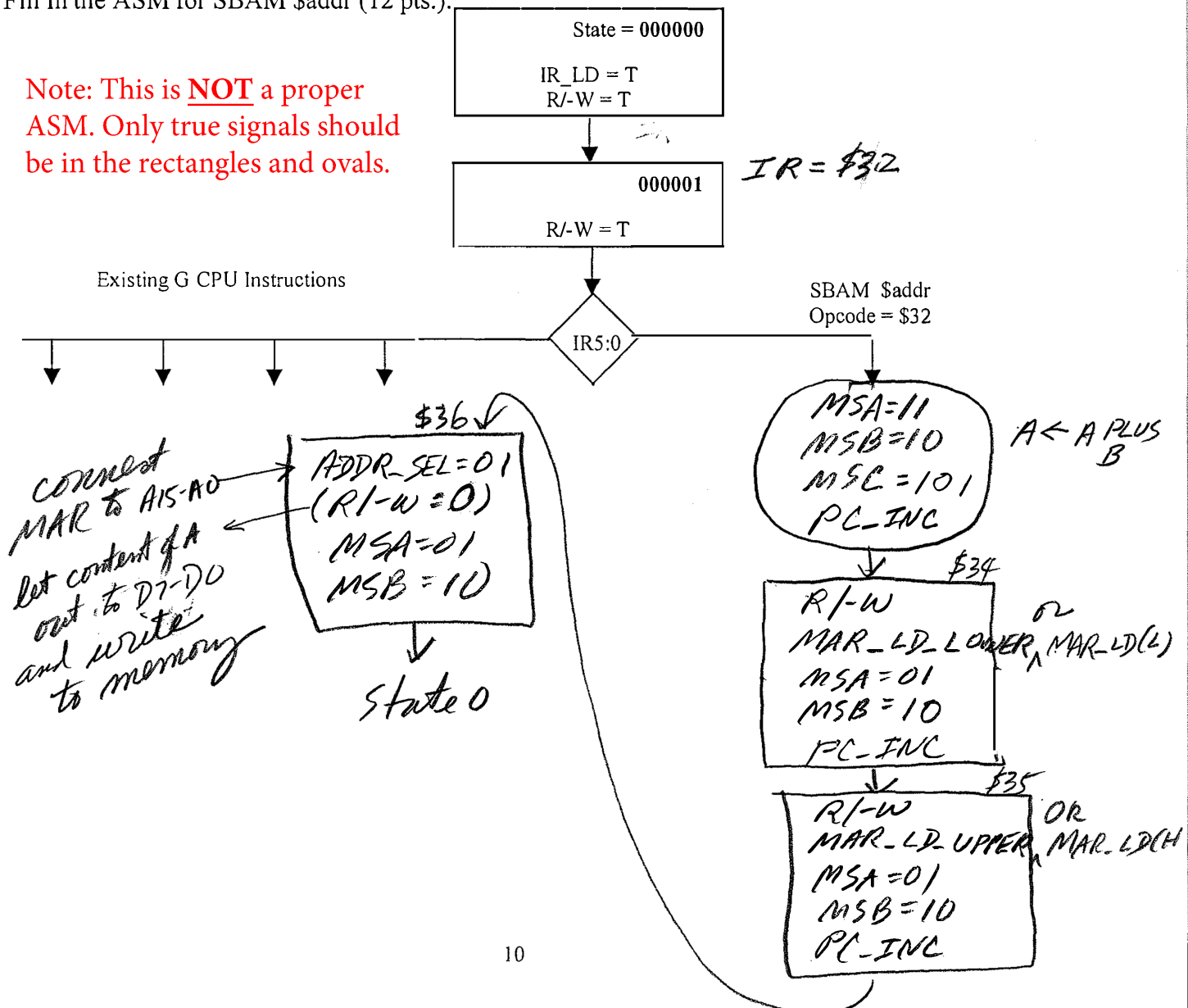
We want to implement a new instruction for the GCPU. The new instruction will be denoted as **SBAM addr**. It added the content of register B to register A (like **SUM_BA**). But, it also stores the result into memory at location "addr". The **new opcode** for this instruction will be designated as \$32 and the **next state available** in the Controller's ASM is \$34. Show the additional states required to implement this new instruction in the Controller's ASM below and show all controller output signals that must be true in each new state for this new instruction:

Assume: R/-W is True for **Read** and **False** for **Write**.

Control Signals: PC_INC, PC_LD_Upper, PC_LD_Lower, MAR_INC, MAR_LD_Upper, MAR_LD_Lower, X_INC, X_LD_Upper, X_LD_Lower, Y_INC, Y_LD_Upper, Y_LD_Lower, IR_LD, R/-W, ADDR_SEL1:0, XD_LD, YD_LD

Note1: The controller output signals are also shown in the G CPU Block Diagram for reference.

Fill in the ASM for SBAM \$addr (12 pts.):



GCPU Instructions**Data Movement Instructions:**

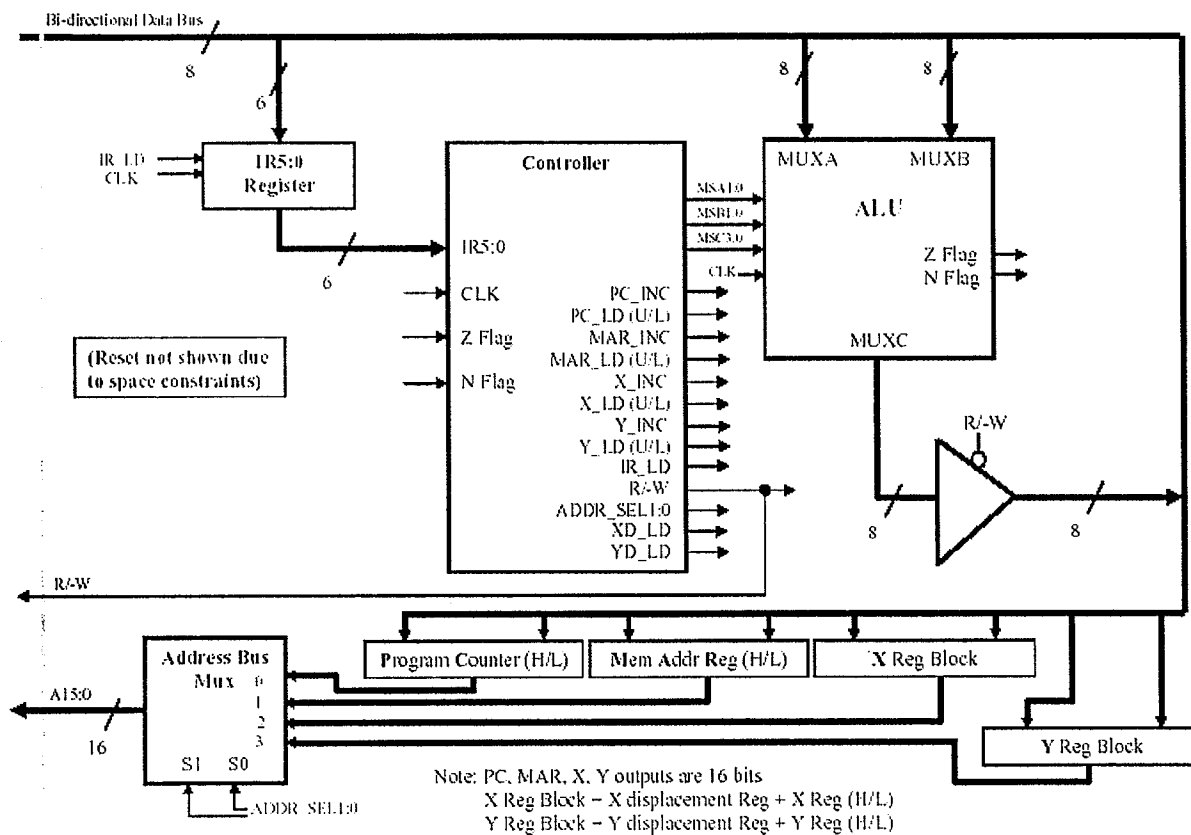
Opcode	Instruction	Operand	Description	# of States
0	TAB	none	Transfer A to B (inherent addressing)	2
1	TBA	none	Transfer B to A (inherent addressing)	2
2	LDAA #data	8 bit data	Load A with immediate data (immediate addr.)	3
3	LDAB #data	8 bit data	Load B with immediate data (immediate addr.)	3
4	LDAA addr	16 bit address	Load A with data from memory location addr (extended addressing)	5
5	LDAB addr	16 bit address	Load B with data from memory location addr (extended addressing)	5
6	STAA addr	16 bit address	Store data in A to memory location addr (extended addressing)	5
7	STAB addr	16 bit address	Store data in B to memory location addr (extended addressing)	5
8	LDX #data	16 bit data	Load X with immediate data (immediate addr.)	4
9	LDY #data	16 bit data	Load Y with immediate data (immediate addr.)	4
A	LDX addr	16 bit addr	Load X with data from memory location addr. (extended addressing)	6
B	LDY addr	16 bit addr	Load Y with data from memory location addr. (extended addressing)	6
C	LDAA dd,X	8 bit displacement	Load A with data from memory location pointed to by X + dd (indexed addressing)	4
D	LDAA dd,Y	8 bit displacement	Load A with data from memory location pointed to by Y + dd (indexed addressing)	4
E	LDAB dd,X	8 bit displacement	Load B with data from memory location pointed to by X + dd (indexed addressing)	4
F	LDAB dd,Y	8 bit displacement	Load B with data from memory location pointed to by Y + dd (indexed addressing)	4
10	STAA dd,X	8 bit displacement	Store data in A to memory location pointed to by X + dd (indexed addressing)	4
11	STAA dd,Y	8 bit displacement	Store data in A to memory location pointed to by Y + dd (indexed addressing)	4
12	STAB dd,X	8 bit displacement	Store data in B to memory location pointed to by X + dd (indexed addressing)	4
13	STAB dd,Y	8 bit displacement	Store data in B to memory location pointed to by Y + dd (indexed addressing)	4

14	SUM BA	none	Sum A, B and place in A (inherent addressing)	2
15	SUM AB	none	Sum A, B and place in B (inherent addressing)	2
16	AND BA	none	AND A, B and place in A (inherent addressing)	2
17	AND AB	none	AND A, B and place in B (inherent addressing)	2
18	OR BA	none	OR A, B and place in A (inherent addressing)	2
19	OR AB	none	OR A, B and place in B (inherent addressing)	2
1A	COMA	none	Complement contents in A (inherent addressing)	2
1B	COMB	none	Complement contents in B (inherent addressing)	2
1C	SHFA L	none	Shift A left by one bit (inherent addressing)	2
1D	SHFA R	none	Shift A right by one bit (inherent addressing)	2
1E	SHFB L	none	Shift B left by one bit (inherent addressing)	2
1F	SHFB R	none	Shift B right by one bit (inherent addressing)	2
30	INX	none	Increment X (inherent addressing)	2
31	INY	none	Increment Y (inherent addressing)	2

20	BEQ	addrL	Branch if A = 0, i.e., Z Flag = 1 (absolute addressing)	3
21	BNE	addrL	Branch if A ≠ 0, i.e., Z Flag = 0 (absolute addressing)	3
22	BN	addrL	Branch if A is negative, i.e., N Flag = 1 (absolute addressing)	3
23	BP	addrL	Branch if A is positive (or zero), i.e., N Flag = 0 (absolute addressing)	3

Special Notes

1. Z flag and N flag are only set and cleared by the contents in register A.
2. A branch is accomplished by moving the operand address "addrL" to the lower byte of the PC. The upper byte of the PC remains unchanged after a branch.
3. The Branch Instructions use absolute addressing where only the low byte of the address is used as an operand. If the branch condition is met, the high byte of the PC is unchanged and the low byte takes the value of the operand (addrL).

GCPU Block Diagram

MSA1/ MSB1	MSA0/ MSB0	Bus Selected as Input to REGA/REGB
0	0	INPUT Bus
0	1	REGA Output Bus
1	0	REGB Output Bus
1	1	OUTPUT Bus

MSC2:0	Action
000	REGA Bus to OUTPUT Bus
001	REGB Bus to OUTPUT Bus
010	complement of REGA Bus to OUTPUT Bus
011	bit wise AND REGA/REGB Bus to OUTPUT Bus
100	bit wise OR REGA/REGB Bus to OUTPUT Bus
101	sum of REGA Bus & REGB Bus to OUTPUT Bus
110	shift REGA Bus left one bit to OUTPUT Bus
111	shift REGA Bus right one bit to OUTPUT Bus (without sign extension)

Partial GCPU Next State Table

Prev State	Opcode	Flags	Next State	Mux Select			Control		REG INC	ADDR SEL	PC	MAR	X,Y Loading		Disp Regs	
Q[5..0]	IR[5..0]	Z N	D[5..0]	MSA [1..0]	MSB [1..0]	MSC [3..0]	IR LD	R /V	PC MAR X Y	ADDR SEL [1..0]	PC LD L/U	MAR LD L/U	X LD L/U	Y LD L/U	XD_LD YD_LD	Present State Function
000000	XXXXXX	XX	000001	01	10	0000	1	1	0000	00	00	00	00	00	00	generic instruction fetch
000001	000000	XX	000000	01	01	0000	0	1	1000	00	00	00	00	00	00	Transfer A to B (TAB)
000001	000001	XX	000000	10	10	0000	0	1	1000	00	00	00	00	00	00	Transfer B to A (TBA)
000001	000010	XX	000010	01	10	0000	0	1	1000	00	00	00	00	00	00	LDAA #data, state 1
000010	XXXXXX	XX	000000	00	10	0000	0	1	1000	00	00	00	00	00	00	LDAA #data, state 2
000001	000011	XX	000011	01	10	0000	0	1	1000	00	00	00	00	00	00	LDAB #data, state 1
000011	XXXXXX	XX	000000	01	00	0001	0	1	1000	00	00	00	00	00	00	LDAB #data, state 3
000001	000100	XX	000100	01	10	0000	0	1	1000	00	00	00	00	00	00	LDAA addr, state 1
000100	XXXXXX	XX	000101	01	10	0000	0	1	1000	00	00	10	00	00	00	LDAA addr, state 4
000101	XXXXXX	XX	000110	01	10	0000	0	1	1000	00	00	01	00	00	00	LDAA addr, state 5
000110	XXXXXX	XX	000000	00	10	0000	0	1	0000	01	00	00	00	00	00	LDAB addr, state 6
000001	000101	XX	000111	01	10	0000	0	1	1000	00	00	00	00	00	00	LDAB addr, state 1
000111	XXXXXX	XX	001000	01	10	0000	0	1	1000	00	00	10	00	00	00	LDAB addr, state 7
001000	XXXXXX	XX	001001	01	10	0000	0	1	1000	00	00	01	00	00	00	LDAB addr, state 8
001001	XXXXXX	XX	000000	01	00	0000	0	1	0000	01	00	00	00	00	00	LDAB addr, state 9
000001	000110	XX	001010	01	10	0000	0	1	1000	00	00	00	00	00	00	STAA addr, state 1
001010	XXXXXX	XX	001011	01	10	0000	0	1	1000	00	00	10	00	00	00	STAA addr, state A
001011	XXXXXX	XX	001100	01	10	0000	0	1	1000	00	00	01	00	00	00	STAA addr, state B
001100	XXXXXX	XX	000000	01	10	0000	0	0	0000	01	00	00	00	00	00	STAA addr, state C
000001	000111	XX	001101	01	10	0000	0	1	1000	00	00	00	00	00	00	STAB addr, state 1
001101	XXXXXX	XX	001110	01	10	0000	0	1	1000	00	00	10	00	00	00	STAB addr, state D
001110	XXXXXX	XX	001111	01	10	0000	0	1	1000	00	00	01	00	00	00	STAB addr, state E
001111	XXXXXX	XX	000000	01	10	0001	0	0	0000	01	00	00	00	00	00	STAB addr, state F
000001	010100	XX	000000	11	10	0010	0	1	1000	00	00	00	00	00	00	SUM BA state 1
000001	010101	XX	000000	01	11	0010	0	1	1000	00	00	00	00	00	00	SUM AB state 1
000001	010110	XX	000000	11	10	0011	0	1	1000	00	00	00	00	00	00	AND BA state 1
000001	010111	XX	000000	01	11	0011	0	1	1000	00	00	00	00	00	00	AND AB state 1
000001	100011	N0	110010	01	10	0000	0	1	1000	00	00	00	00	00	00	BP addr state 1
000001	100011	N1	110011	01	10	0000	0	1	1000	00	00	00	00	00	00	BP addr state 1
110010	XXXXXX	XX	000000	01	10	0000	0	1	0000	00	10	00	00	00	00	BP addr state 32
110011	XXXXXX	XX	000000	01	10	0000	0	1	1000	00	00	00	00	00	00	BP addr state 33
000001	110000	XX	000000	01	10	0000	0	1	1010	00	00	00	00	00	00	Increment X (INX)
000001	110001	XX	000000	01	10	0000	0	1	1001	00	00	00	00	00	00	Increment Y (INY)