ECE 198JL Final Exam
Fall 2013

December 16th, 2013

Name: Austin McWilliams NetID: apmcwil2

Discussion Section:

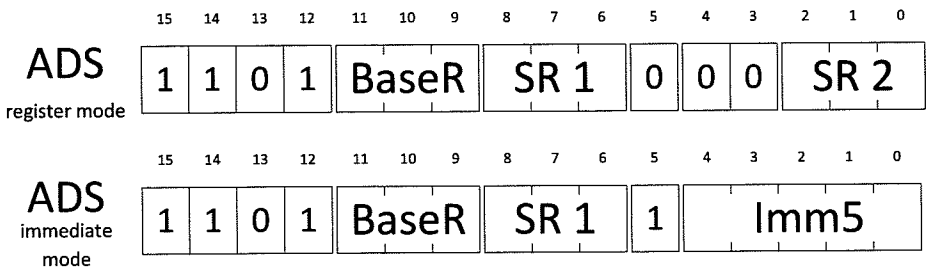
10:00 AM	[ ] JD1	
11:00 AM	[ ] JD2	
12:00 PM	[ ] JD7	
1:00 PM	[ ] JD9	[ ] JDA
2:00 PM	[ ] JDB	
3:00 PM	[X] JDC	
4:00 PM	[ ] JD8	

- **Be sure your exam booklet has 13 pages.**
- **Be sure to write your name and discussion section on the first page.**
- **Do not tear the exam booklet apart. You can only detach last page, if needed.**
- **We have provided LC-3 instructions set and other reference materials on separate pages.**
- **Use backs of pages for scratch work if needed.**
- **This is a closed book exam. You may not use a calculator.**
- **You are allowed two handwritten 8.5 x 11" sheets of notes.**
- **Absolutely no interaction between students is allowed.**
- **Be sure to clearly indicate any assumptions that you make.**
- **The questions are not weighted equally. Budget your time accordingly.**
- **Don't panic, and good luck!**

Problem 1	16 points:
Problem 2	10 points:
Problem 3	14 points:
Problem 4	17 points:
Problem 5	11 points:
Problem 6	24 points:
Problem 7	10 points:
Problem 8	13 points:
Problem 9	25 points:
Total	140 points:

Problem 1 (16 pts): LC-3 microinstructions

You are adding a new instruction, called **ADS** (add and store), to the LC-3 instruction set. This instruction adds two values, just like the ADD instruction does, and stores the result to the **memory** instead of the register file. The destination memory address is provided in the BaseR register specified by IR[11:9] bits. The binary encoding of this instruction is:



a) In RTL form, give a sequence of (at most 4) *microinstructions* that implement the execute phase of the **ADS** instruction. Make sure your implementation **does not modify any values in the general-purpose register file** and **does not set condition codes**.

MDR ← SR1 + OP2 (*) ✓ (*) OP2 may be SR2 or SEXT[imm5]

MAR ← BaseR ✓

M[MAR] ← MDR ✓

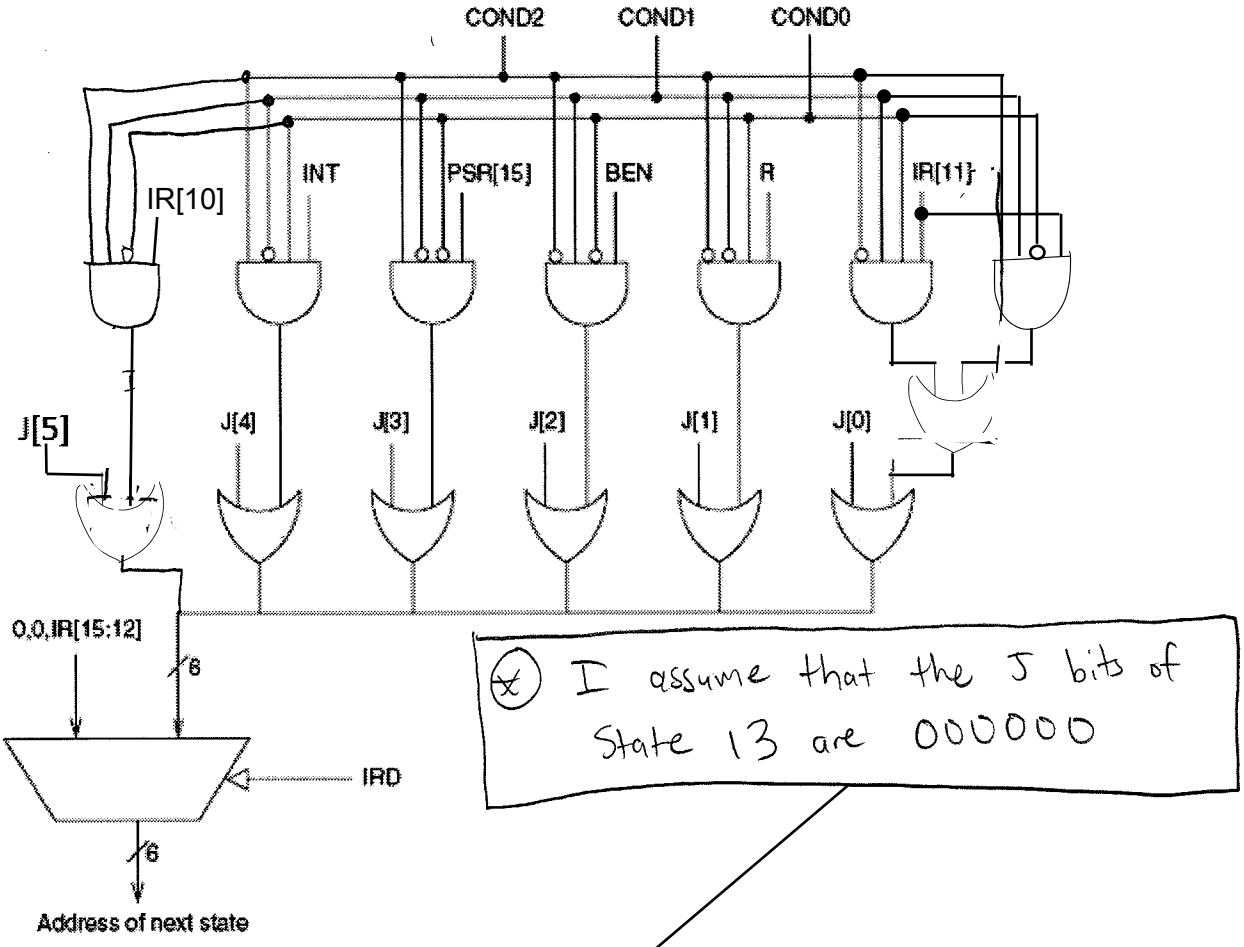
b) Determine the control ROM microinstructions that implement the RTL statements from part (a). Complete the table below by filling in 0, 1, or x as appropriate. Use don't cares wherever possible. Specify ROM addresses in **decimal**. When you need *additional* states, state numbers **55**, **56**, **57**, and **58** are available for your use.

ROM address	IRD ✓	COND(3) ✓	J(6) ✓	LD.BEN LD.MAR LD.MDR LD.IR LD.PC LD.REG ✓ LC.CC	GateMARMUX GateMDR GateALU ✓ GatePC	MARMUX PCMUX(2) ADDR1MUX ADDR2MUX(2) DRMUX(2) ✓ SR1MUX(2) ALUK(2)	MIO.EN R.W ✓
17	0	000	110111	0 0 1 0 0 0 0	0 0 1 0	x xx x xx xx 01 00	0 x
55	0	000	010000	0 1 0 0 0 0 0	0 0 1 0	x xx x xx xx 00 11	0 x
16	0	001	010000	**Do not fill in this space.** **Only fill in control word bits for the first two microinstructions.**			

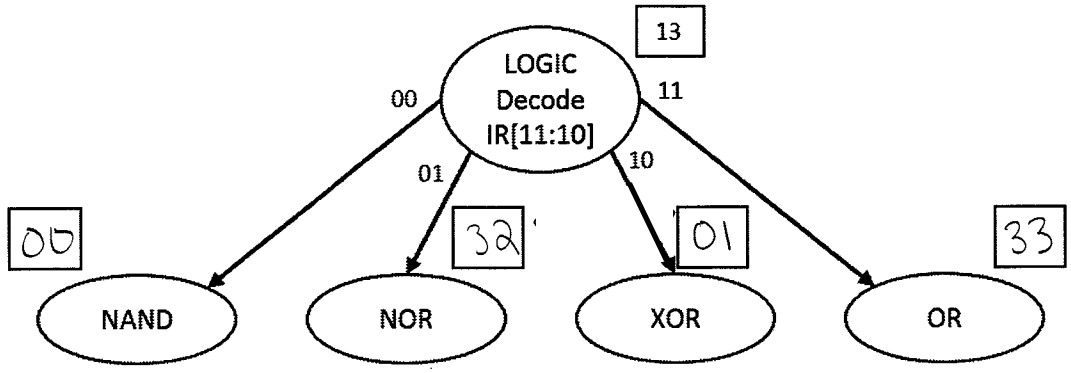
Problem 2 (10 pts): LC-3 microsequencer

Suppose we added a new instruction **LOGIC** with opcode 1101 to the LC-3 that will perform one of four logic operations (NAND, NOR, XOR, and OR) based on the IR[11:10] bits. To implement this new instruction, we need five new states in the LC-3 FSM and need to modify the microsequencer circuit. The first state performs a secondary decode phase before executing the four logic operations.

a) The microsequencer should use COND = 110 to determine the next state during the secondary decode phase of the logic operation. Adding at most 2 AND gates, 2 OR gates, and wires, modify the microsequencer circuit so that it can correctly decode the **LOGIC** instruction. A few modifications have already been made to give you a hint.



b) Based on your microsequencer circuit above, choose a set of *viable* next states for the execute phase states. Don't worry if the states are already in use by the LC-3 FSM. :)



Problem 3 (14 pts): LC-3 assembly program analysis

The following program prints a *sprite*. Sprite is a two-dimensional image of size 8x8 stored in memory as a one-dimensional array, row after row, with one symbol per memory location. For example, sprite shown on the right will occupy 64 memory locations starting from address labeled as SPRITE where each location contains ASCII value of the symbol in the sprite:

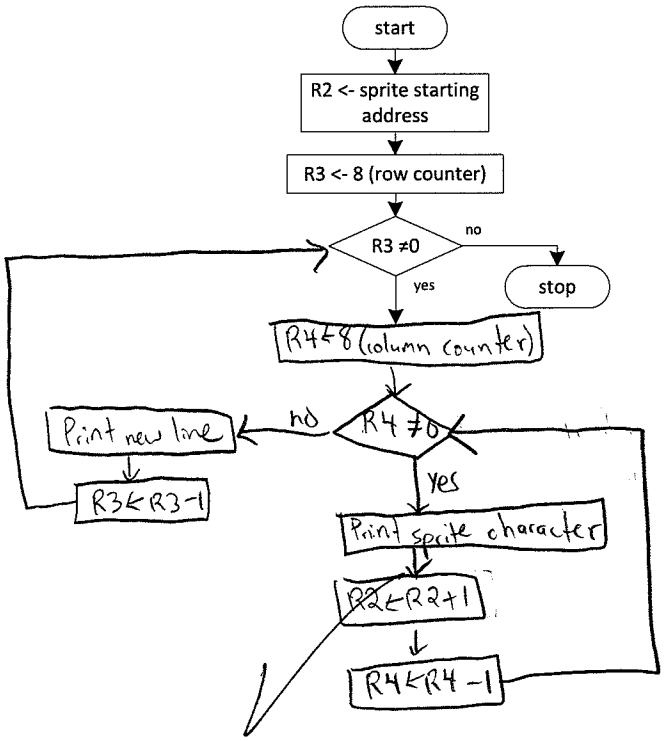
```
.FILL x2A ; *
.FILL x2A ; *
...
```

```
********
*      *
* ^  ^ *
* *  * *
*      *
* *  * *
*  **  *
********
```

a) Fill in missing instructions

```
.ORIG x3000
                LEA R2, SPRITE
                AND R3, R3, #0
                ADD R3, R3, #8
NEXT_ROW        ADD R3, R3, #0
                BRz DONE
                AND R4, R4, #0
                ADD R4, R4, #8
NEXT_COLUMN     ADD R4, R4, #0
                BRz DONE_ROW
                LDR R0, R2, #0
                OUT
                ADD R2, R2, #1
                ADD R4, R4, #-1
                BRnzp NEXT_COLUMN
DONE_ROW        LD R0, ASCII_NL
                OUT
                ADD R3, R3, #-1
                BRnzp NEXT_ROW
DONE            HALT
ASCII_NL .FILL xA
SPRITE
.FILL x2A ; *
.FILL x2A ; *
...
.END
```

b) Draw a flowchart for the program to the left. Be specific, use standard symbols only (ovals, rhombs, rectangles, etc.) The flowchart is already partially built to give you an example of what's expected.



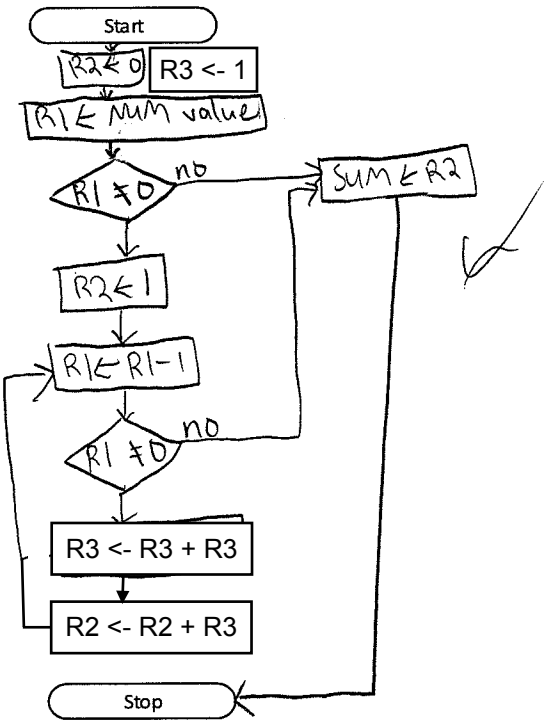
c) Complete the symbol table for the above program.

Symbol Name	Address
NEXT_ROW	x3003
NEXT_COLUMN	x3007
DONE_ROW	x300E
DONE	x3012
ASCII_NL	x3013
SPRITE	x3014

Problem 4 (17 pts): LC-3 assembly language programming

Write a program in LC-3 assembly language that computes sum 1 + 2 + 4 + 8 + ⋯ where sum terms are the successive powers of two. The number of terms is supplied by the user in memory at the location labeled as **NUM**. You can assume that this value is such that no overflow will occur. The result should be stored in memory at the location labeled as **SUM**.

a) Draw flowchart for the solution to the above problem.



; insert after initializing R2
AND R3, R3, #0;
ADD R3, R3, #1; initialize R3 to 1

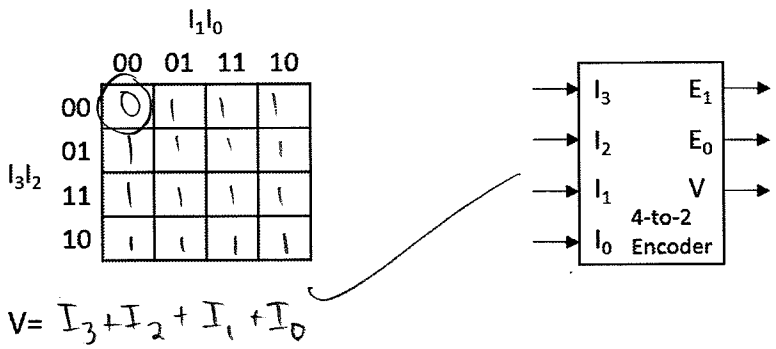
b) Write a program in LC-3 assembly language that corresponds to the above flowchart. The program must start at memory address x3000. The *number of terms* value must be initialized to 12. The program must terminate and it must be **well-documented** such that it can be graded by a human.

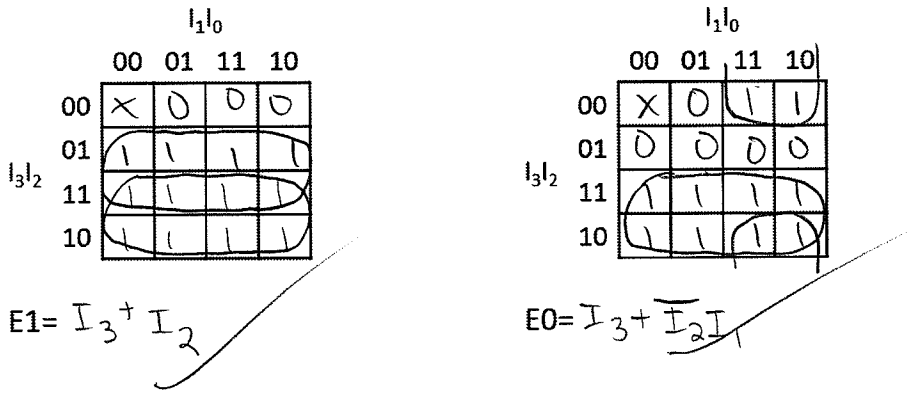
```
;       R1 is the term counter, R2 holds the sum, R3 holds the powers of 2
        .ORIG   x3000
        AND     R2, R2, #0  ; initialize R2 to 0
        LD      R1, NUM     ; initialize R1 to # of terms
        BRnz    DONE        ; finished if # of tems is <= 0
        ADD     R2, R3, #1  ; give R2 first term
CHECK   ADD     R1, R1, #-1 ; decrement term counter
        BRz     DONE        ; finished if term counter is 0
        ADD     R3, R3, R3  ; double the value of R3
        BRnzp   CHECK       ; go to see if there are more terms

; insert after doubling R3
ADD R2, R2, R3; compute sum

DONE    ST      R2, SUM
        HALT
NUM     .FILL   #12         ; memory location for # of terms
SUM     .BLKW   #1          ; memory location for sum
        .END
```

Problem 5 (11 pts): Priority Encoder

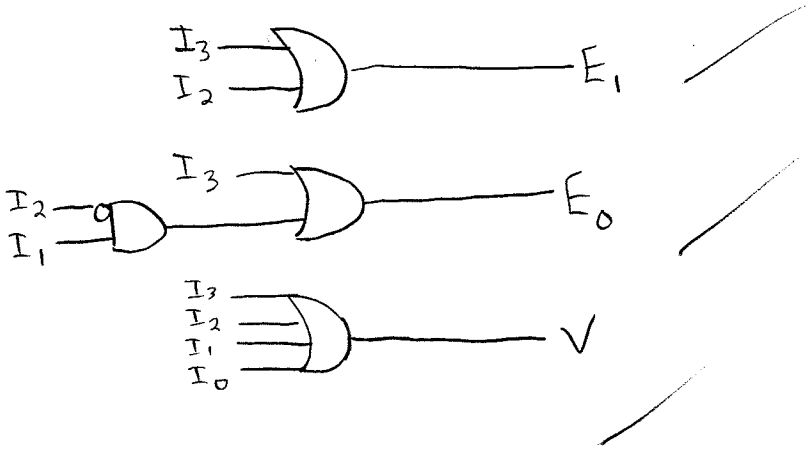
A 4-to-2 priority encoder encodes a set of four binary input bits into a binary code. The indices of the input bits I_3, I_2, I_1, and I_0 are decimal numbers that indicate which number should be encoded in unsigned binary representation in the output bits E_1E_0 (where the indices of the output bits represent the power of 2 encoded by each E bit). For example, for input $I_3I_2I_1I_0 = 0100$, the output is $E_1E_0 = 10$. If more than one input bit is one, the output bits give *priority* to the largest decimal number and encode the index of that input. For example, for input $I_3I_2I_1I_0 = 0101$, the output is $E_1E_0 = 10$. A third output V is 1 if and only if the input is valid. An input is valid only if at least one of the inputs is one. If none of the inputs is one, the input is invalid and the value of the encoded output does not matter.

a) Complete the Karnaugh maps for outputs E1, E0, and V, then derive **minimal** Boolean expressions for each output.





b) Implement the 4-to-2 priority encoder using as few gates as possible.



Problem 6 (24 pts): Control unit design

THIS IS NOT THE LC-3!!!!!!!!!! An additional copy of the datapath can be torn off from the backpage.

Each major part of this problem can be answered without a correct answer to the other parts.

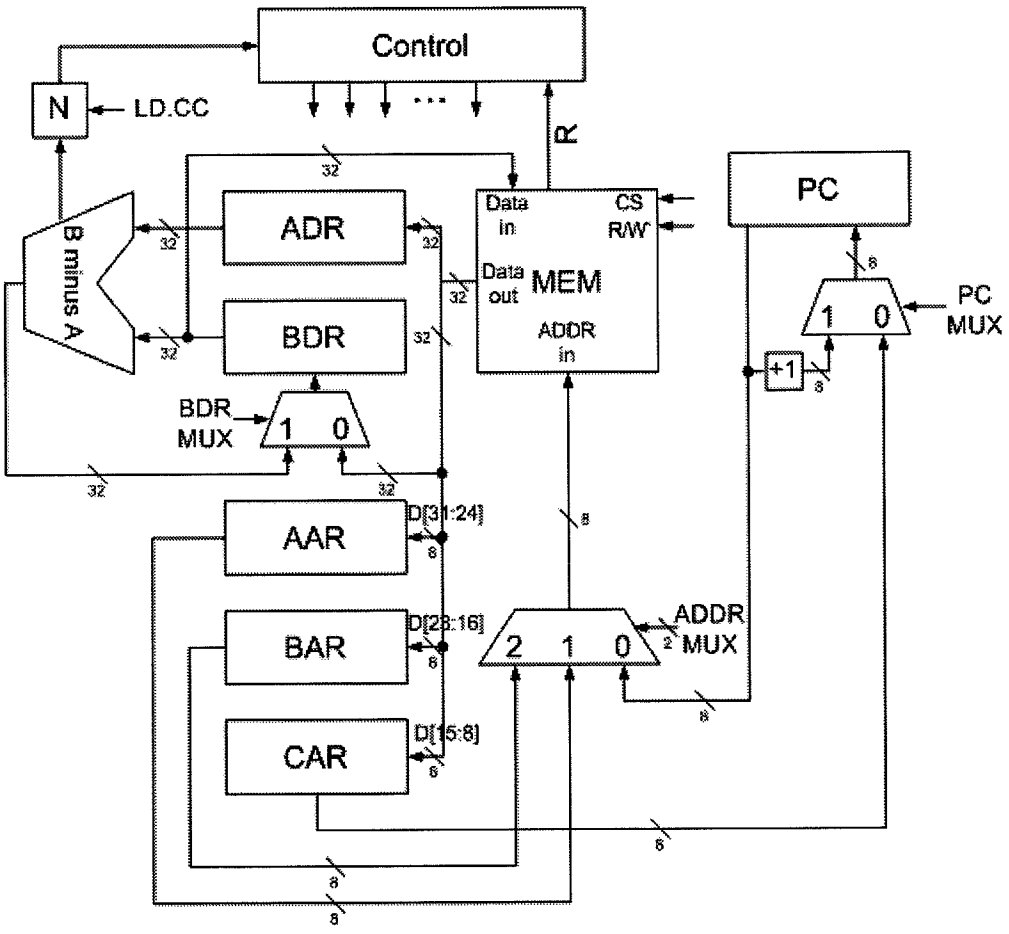
The Subtract and Branch if Negative (SBN) is a one instruction set computer. The SBN subtracts the contents at memory address *a* from the contents at memory address *b* and stores the result at address *b* (M[*b*]←M[*b*]-M[*a*]). If the result from the subtraction is negative, the program branches to the instruction at address *c* (PC ← *c*), else the program executes the next instruction in memory (PC ←PC+1).

The assembly code for the SBN instruction takes the following form.

SBN *a*, *b*, *c* ; M[*b*] ←M[*b*]-M[*a*]
; if (M[*b*]-M[*a*] ≤0) PC ← *c*, else PC ←PC+1

This ISA can be implemented with the following datapath and control unit. The datapath has 6 registers. Each register has a load signal that controls when it loads a new value; these signals are not shown. The subtractor is a 32-bit two's complement subtraction circuit.

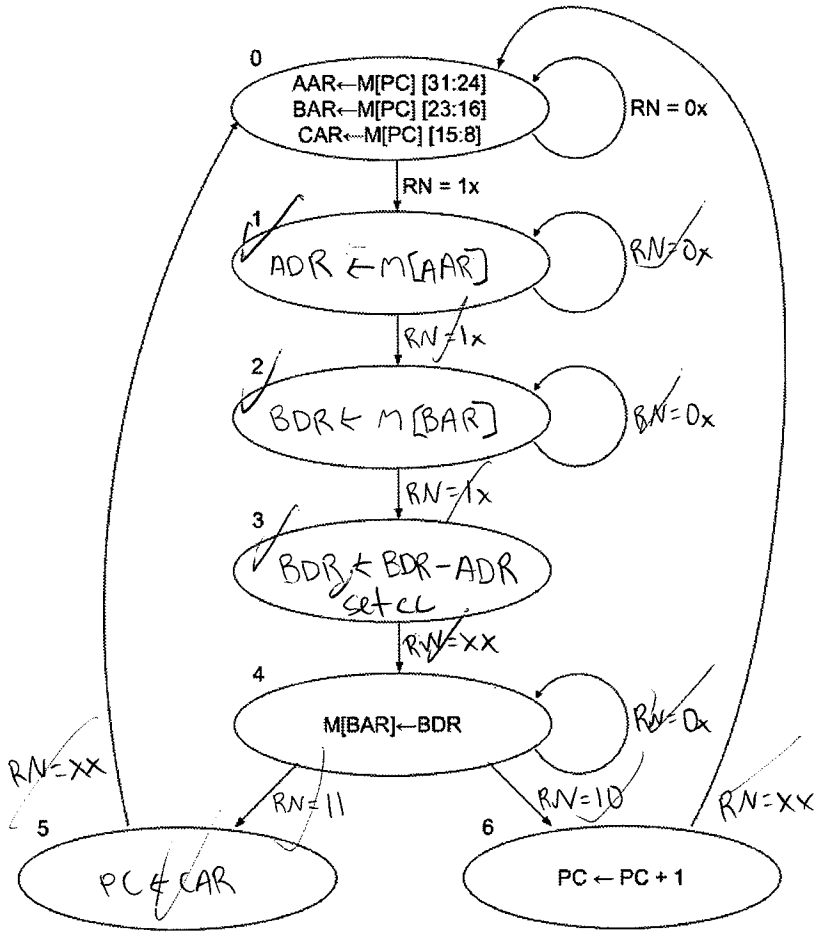
8-bit registers	PC – Program Counter	AAR – *a* addr register	BAR – *b* addr register	CAR – *c* addr register
32-bit registers	ADR – *a* data register		BDR – *b* data register	
Status flip-flop	N = 1, when subtraction yields a negative number			
Memory Control	CS = 1 when memory is active and 0 when inactive, R/W' = 1 for read and 0 for write.			



a) Below is the state diagram for the SBN architecture.

i. Place RTL instructions inside the empty states to finish the SBN state diagram.

ii. Indicate the values of the inputs denoted as R for the memory ready bit and N is the negative status flip-flop for **all unlabeled** state transitions.



b) Using the table below, fill in the values of the control signals for states 0, 4, and 6. Use don't cares when possible.

		32-bit reg		8-bit registers				MUXes			MEM	
Signal Name	LD.CC	LD.ADR	LD.BDR	LD.PC	LD.AAR	LD.BAR	LD.CAR	PCMUX	BDRMUX	ADDRMUX	CS	R/W'
State 0	0	0	0	0	1	1	1	X	X	00	1	1
State 4	0	0	0	0	0	0	0	X	X	10	1	0
State 6	0	0	0	1	0	0	0	1	X	XX	0	X

c) Based on the datapath and assembly instruction, draw the instruction format for the SBN instruction.

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Memory address a								Memory address b								Memory address c							0	0	0	0	0	0	0	0	0

Problem 7 (10 pts): Multiple-Choice Questions

For each multiple-choice question, you will receive **+1** point for a correct answer, **0** points for not answering, and **-1** point for a wrong answer. Circle your selected answer.

a) Select the expression that correctly compares the two numbers. The first number is encoded in IEEE 754 32-bit floating point representation and the second number is encoded in 8-bit 2's complement notation.

i. 01000010111110000000000000000000 > 01000000
ii. 01000010111110000000000000000000 = 01000000
iii. 01000010111110000000000000000000 < 01000000

b) Which statement is true about the two sets of numbers? [note the bases]

i. $(2.7)_{10} > (2.7)_{16}$ and $(1.3)_{10} > (1.3)_{16}$
ii. $(2.7)_{10} < (2.7)_{16}$ and $(1.3)_{10} < (1.3)_{16}$
iii. $(2.7)_{10} = (2.7)_{16}$ and $(1.3)_{10} = (1.3)_{16}$
iv. $(2.7)_{10} > (2.7)_{16}$ and $(1.3)_{10} < (1.3)_{16}$
v. $(2.7)_{10} < (2.7)_{16}$ and $(1.3)_{10} > (1.3)_{16}$

c) A random access memory (RAM) chip has 32 words and 64 bits per word.

i. How many address lines does the RAM have? Circle one: **5** **6** **32** **64**
ii. How many data lines does the RAM have? Circle one: **5** **6** **32** **64**

d) Which of the following 4-bit two's complement additions could result in overflow? Each variable (a, b, c, or d) is either 0 or 1 independent of the values of the other variables.

I)
$$\begin{array}{r} 00ab \\ +\ 1101 \\ \hline \end{array}$$

II)
$$\begin{array}{r} 00cd \\ +\ 0110 \\ \hline 1001 \end{array}$$

Circle one: **i. I only** **ii. II only** **iii. I and II** **iv. Neither**

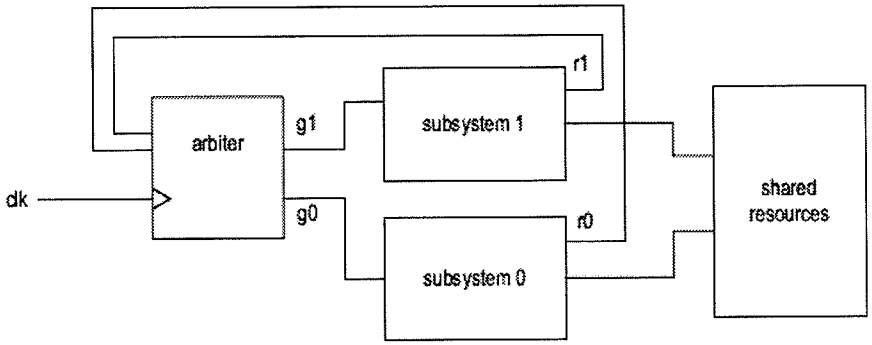
e) Use the K-map below to answer the following questions

ef \ gh	00	01	11	10
00	0	0	1	1
01	0	1	1	0
11	1	1	0	0
10	1	1	0	x

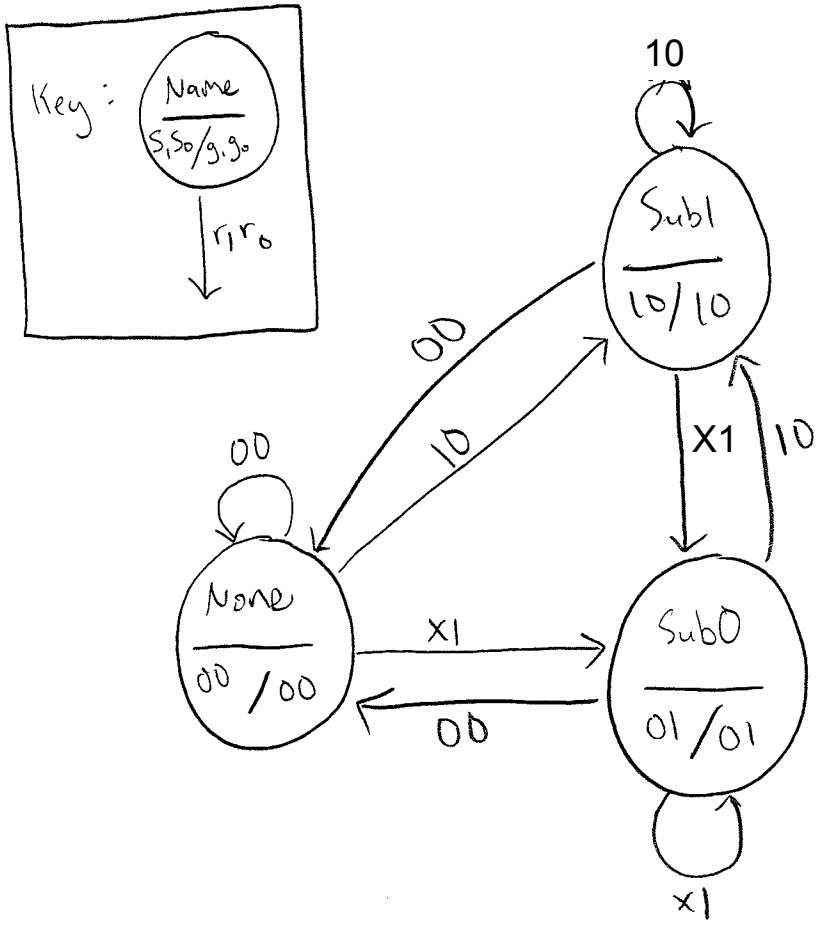
i. The K-map has how many prime implicants? Circle one: **3** **4** **5** **6** **7**
ii. $f\bar{g}h$ is an essential prime implicant (circle one): **True** or **False**
iii. The K-map has a unique minimal SOP Boolean expression: **True** or **False**
iv. A min SOP Boolean expression can be implemented using a 2-level NAND-NAND circuit: **True** or **False**
v. What is the minimum number of NOR gates that can implement the K-map? Circle one: **3** **4** **5**

Problem 8 (13 pts): FSM

In this assignment you will be designing FSM of an *arbiter* circuit. In many systems, some resources are shared by many subsystems. An arbiter is a circuit that coordinates access to these shared resources and resolves any conflicts. Consider example system shown below. It consists of two subsystems that both use the same shared resource. When a subsystem needs access to the shared resource, it activates (asserts) its *request* signal *r*. The arbiter monitors the use of the shared resource and the incoming request signals and grants access to the shared resource by activating the corresponding *grant* signal *g*. Once its grant signal is activated, a subsystem has permission to access the resource. After the task has been completed, the subsystem releases the resource by deactivating (clearing) its request signal *r*. When both subsystems simultaneously request access to the shared resource, priority is given to subsystem 0.



Draw a Moore state diagram for the above arbiter FSM. Assign states. Make sure to label each state with a name and output and each edge with an input. Make sure to label all parts, inputs, outputs, edges, etc. Points will be deducted for missing labels and messy drawing.

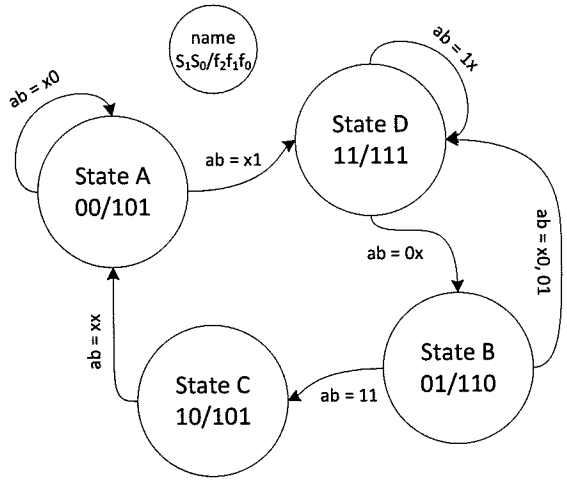


Problem 9 (25 pts): FSM implementation

Implement the FSM shown below using *negative-edge* triggered D flip-flops.

a) Based on the FSM shown below, drawn the next-state table. Use don't cares when possible.

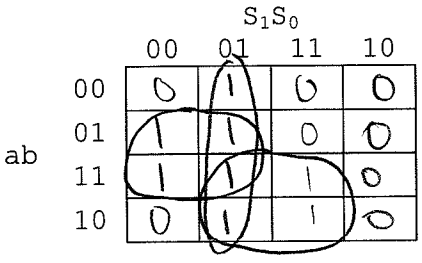
FSM



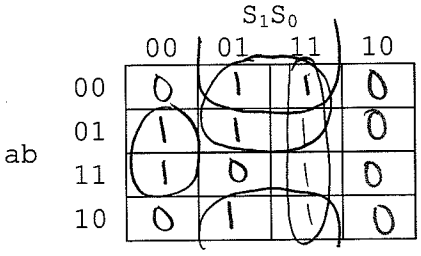
Next-state table

Current State		External Inputs		Next State		External Outputs		
S_1	S_0	a	b	S_1^+	S_0^+	f_2	f_1	f_0
0	0	x	0	0	0	1	0	1
		x	1	1	1			
0	1	x	0	1	1	1	1	0
		0	1	1	1			
		1	1	1	0			
1	0	x	x	0	0	1	0	1
1	1	0	x	0	1	1	1	1
		1	x	1	1			

b) Based on the next-state table from part (**a**), fill in K-maps and write **minimal SOP** Boolean expressions for flip-flop inputs D_1 and D_0.



$D_1 = b\bar{S}_1 + \bar{S}_1 S_0 + a S_0$



$D_0 = \bar{a} S_0 + \bar{b} S_0 + S_1 S_0 + b \bar{S}_1 \bar{S}_0$

c) Write **minimal** Boolean expressions for f_2, f_1, f_0.

$f_2 = 1$ (connect to power source)

$f_1 = S_0$

$f_0 = S_1 + \bar{S}_0$

d) Implement the FSM from part **(a)** using *negative-edge* triggered D flip-flops. Make sure to label all parts, inputs, and outputs. Points will be deducted for missing labels and messy drawing.