ECE120 Final Review - Cramming Carnival Solutions

Author: Members of HKN

Spring 2024

Answers are in blue

1 Bitwise Bonanza

(a) **Twos Complement Sign Extension**: Given the binary numbers in two's complement give the respective value as a 8-bit two's complement representation (If given a 5-bit number, write the same value with 8-bits)

(i) 010100 00010100
(ii) 1010100 11010100
(iii) 0001010 00001010
(iv) 0100011 00100011
(v) 111000 11111000
(vi) 10 11111110
(vii) 00 00000000
(viii) 01 00000001
(ix) 1 11111111

(b) **Converting to Hexadecimal Notation**: Given the binary number below convert to hexadecimal (Sign extend if necessary)

(i) 1010100111010000 A9D0
(ii) 0101001111000001 53C1
(iii) 1111011110001010 F78A
(iv) 0100001110100101 23A5

(c) **Converting From Unsigned Representation**: Given the 5-bit unsigned binary string, what is its decimal equivalent?

(i) 10101 21
(ii) 01010 10
(iii) 11110 30
(iv) 01000 8
(v) 11011 27

(d) **Converting from IEEE 754 32-bit Floating Point**: Given the IEEE 754 32-bit Floating Point, what is its decimal equivalent?

(i) 1 10000000 00010000000000000000000 -2.125
(ii) 0 01111111 10010000000000000000000 1.5625
(iii) 1 10000011 11000100000000000000000 -28.25

(e) **Converting from Hexadecimal Notation**: Given the hexadecimal notation below, what is the binary representation?

(i) 109F 0001000010011111

(ii) 74BC 0111010010111100

(iii) E235 1110001000110101

(iv) 86AD 1000011010101101

(f) **Representation Integer Range**: What is the minimum number of bits needed to represent the values (which are in decimal) below?

(i) 343 9 [10 if 2's complement]

(ii) 1709 11 [12 if 2's complement]

(iii) -123 8

(iv) -2 2

(v) 0 1

(g) **Representing a Number Using Two's Complement Representation**: Give the 6-bit two's complement representation of the value of the following decimal values

(i) -9 110111

(ii) -4 111100

(iii) 5 000101

(iv) 7 000111

(v) -16 110000

(h) **Converting to IEEE 754 32-bit Floating Point**: Enter the IEEE 754 32-bit floating point representation of each of the following decimal fractions.

(i) $6\frac{15}{16}$ 0 10000001 10111100000000000000000

(ii) $4\frac{1}{8}$ 0 10000001 00001000000000000000000

(iii) $7\frac{3}{4}$ 0 10000001 11110000000000000000000

(i) **Two's Complement Subtraction**: If these numbers are in two's complement representation subtract the numbers.

(i) 101001 - 111111 101010

(ii) 001101 - 001010 0000011

(iii) 110010 - 110110 100

(iv) 111100 - 000110 110110

(j) **Two's Complement Addition Overflow**: For the following binary values, which of them have an overflow in 2's complement.

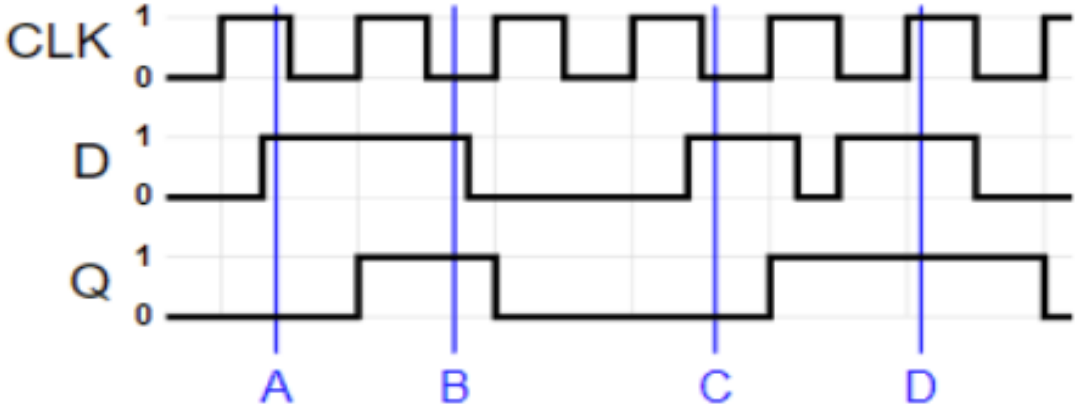
(i) 110001 + 100011 Yes

(ii) 100100 + 001010 No

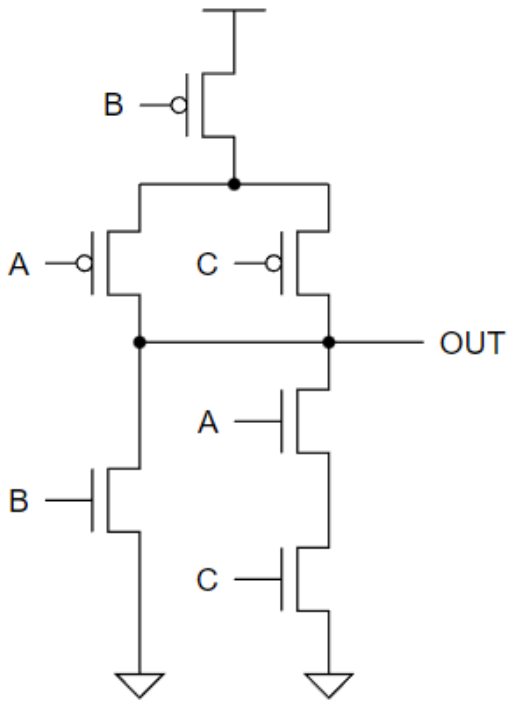
(iii) 110010 + 111111 No

2 Thrilling Toggles and Terrific Truth Tables

(a) **D Flip-Flop Timing Diagram**: Given this image below of a timing diagram for a positive edge-triggered D flip-flop sketch the values for Q at each time. Solution Below:



(b) **CMOS Gate Truth Table (3 Variables)**: For the CMOS gate shown below, enter in the truth table output.



A	B	C	Out
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

(c) **Boolean Expression to Truth Table**: Fill in the truth table given by the functions below.

(i) F(A,B,C,D) = (A+B'+C)'D'
(ii) G(A,B,C,D) = (A+B')(CD+A)'
(iii) H(A,B,C,D) = (CB)+A'D'B+B'CD
(iv) I(A,B,C,D) = ACD+AB+(BC)'

Inputs				Outputs			
A	B	C	D	F	G	H	I
0	0	0	0	0	1	0	1
0	0	0	1	0	1	0	1
0	0	1	0	0	1	0	1
0	0	1	1	0	0	1	1
0	1	0	0	1	0	1	1
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
0	1	1	1	0	0	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	0	0	1
1	0	1	0	0	0	0	1
1	0	1	1	0	0	1	1
1	1	0	0	0	0	0	1
1	1	0	1	0	0	0	1
1	1	1	0	0	0	1	1
1	1	1	1	0	0	1	1

(d) **Truth Table to SOP and POS**: Using the truth table below, express outputs F1 and F2 as a minimal sum of products (SOP) and a minimal product of sums (POS)

A	B	C	D	F1	F2
0	0	0	0	1	1
0	0	0	1	1	0
0	0	1	0	0	0
0	0	1	1	x	1
0	1	0	0	0	1
0	1	0	1	0	0
0	1	1	0	1	1
0	1	1	1	x	x
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	1
1	0	1	1	1	1
1	1	0	0	0	x
1	1	0	1	1	1
1	1	1	0	0	1
1	1	1	1	x	0

F1 (SOP) = A'B'C'+CD+A'BC+B'C'D'+ABD

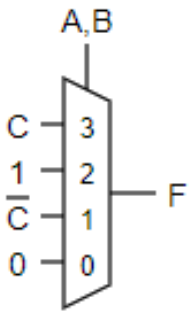
F1 (POS) = (B'+C+D)(A+B'+D)(A'+B+C+D')
(A+B+C')(A'+C'+D)

F2 (SOP) = A'C'D'+ABC'+B'CD+A'BC+ACD'

F2 (POS) = (A+C+D')(A'+B'+C'+D')
(A'+B+C')(A+B+C'+D)

3 Multiplexer Mania

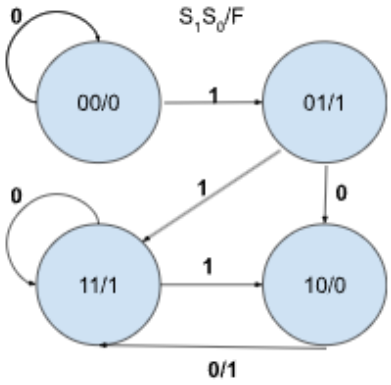
Multiplexer to Boolean Formula: Give the minimal SOP for F for the multiplexer below (express F in terms of the inputs)



F(A,B,C) =A'BC'+AC+AB'

4 Funky FSMs

(a) **FSM Design Application**: Given the FSM below:



(i) Find the K-Maps for S1+,S0+, and F

S1S0	I=0	I=1
00	0	0
01	1	1
11	1	1
10	1	1

Table 1: S1+ Solutions

S1S0	I=0	I=1
00	0	1
01	0	1
11	1	0
10	1	1

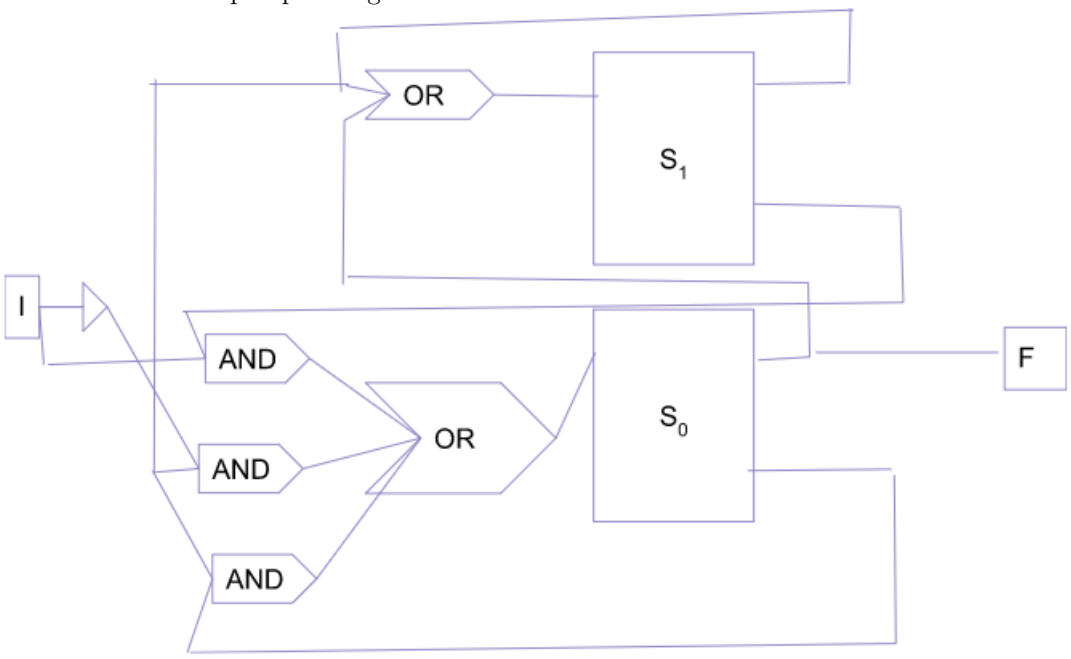
Table 2: S2+ Solutions

S1 / S0-	F
00	0
01	1
11	1
10	0

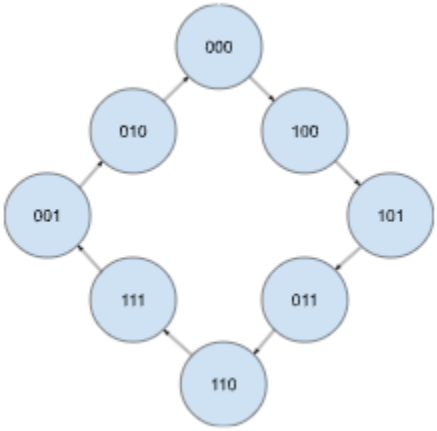
Table 3: F Solutions

(ii) Give their minimal SOP Boolean logic expressions
S1+ = S0+S1
S0+ = S1'I+S1I'+S1S0'
F = S0

(iii) Implement the FSM as D flip-flops and gates



(b) **FSM Counter**: Given the FSM below:



(i) Find the K-Maps for S2+,S1+, and S0+:

S2	I
00	0
01	1
11	0
10	1

Table 4: S2+ Solutions

S1S0	I=0	I=1
00	0	0
01	1	1
11	0	1
10	1	0

Table 5: S1+ Solutions

S1S0	I=0	I=1
00	0	0
01	0	1
11	1	1
10	1	1

Table 6: S0+ Solutions

(ii) Give their minimal SOP Boolean logic expressions
S2+ = S1'S0'+S2'S1S0+S2S0'
S1+ = S1'S0+S2'S0+S2S1S0'
S0+ = S2

(c) **FSM Sequence Detector**: Generate an FSM that detects the sequence 001:

Input sequence M (starting from left to right): 100110100101

Output sequence R (Starting from left to right): 00010000010

(i) State declaration table

State Names	**States (S1S0)**	**State Meanings**
Start	00	None of pattern elements have been found yet
A	01	First '0' in the pattern '001' has been found
B	10	Second '0' in the pattern '001' has been found
C	11	Pattern '001' identified

Table 7: State declaration table

(ii) The K-Maps for S1+,S0+, and F

S1S0	M=0	M=1
00	0	0
01	1	0
11	0	0
10	1	1

Table 8: S1+ Solutions

S1S0	M=0	M=1
00	1	0
01	0	0
11	1	0
10	0	1

Table 9: S0+ Solutions

S1S0	R
00	0
01	0
11	1
10	0

Table 10: F Solution

(iii) Give their minimal SOP Boolean logic expressions

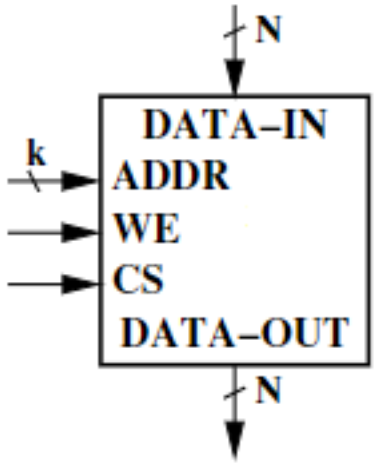
S1+ = S1S0'+M'S1'S0

S0+ = M'S1'S0'+M'S1S0+MS1S0'

R = S1S0

5 Rambunctious RAM

RAM Concepts: Use the RAM Module Below to answer the questions:



(a) If there are k ADDR wires and N DATA-IN wires, what size memory does this chip store? (Hint: AxB)
$2^k \times N$

(b) After entering the chip and before being input to a cell, what do the k wires enter?
A decoder so it can access specific cells

(c) If given four 4x4 RAM blocks and any decoder possible, what sizes of RAM could be made?
4x4, 4x8, 4x16, 16x4, 8x4, 8x8

(d) In a 16x8 RAM do the following tasks

(i) Write 0x12 into address 0x578A
Into WE and CS 1
Into ADDR 0101 0111 1000 1010
Into DATA-IN 0001 0010

(ii) Write 0x72 into address 0xECEB
Into WE and CS 1
Into ADDR 1110 1100 1110 1011
Into DATA-IN 0111 0010

(iii) Read from address 0x893F
Into WE 0 and CS 1
Into ADDR 1000 1001 0011 1111
DATA-OUT reads the data from that cell

6 Legendary LC3

(a) **LC-3 bits to RTL**: For each of the following 16-bit sequences, give the RTL equivalent:

(i) 0001 1000 0100 0100 ADD R4, R1, R4

(ii) 0101 0000 0011 1111 AND R0, R0, -1

(iii) 0000 1101 1111 1100 BR110 #-4

(iv) 0010 1000 0000 0111 LD R4, #7

(v) 1001 1010 0011 1111 NOT R5, R0

(b) **LC-3 FSM Concepts (Von Neumann model)**: Answer the conceptual questions below:

(i) What are the first 3 FSM instructions the LC-3 executes with each line of code in memory?
$MAR \leftarrow PC, PC \leftarrow PC + 1$
$MDR \leftarrow M$
$IR \leftarrow MDR$

(ii) If the ALUK value is 00, what is the opcode of the line currently being performed?
ADD

(iii) How many gates (that go to BUS) can be active at a time?
1

(iv) Which opcodes set CC?
ADD, AND, NOT, LD, LDI, LDR, LEA

(v) In the von Neumann model, where is the program stored?
Memory

(vi) How many phases are there in an instruction cycle?
3 phases – Fetch, Decode, Execute. Each of these might go through multiple states, as described by the control FSM.

(c) **LC-3 Programming**: Write the following functions in RTL (For each part, assume your first line starts at x3000 and all registers are x3000. Each part is independent of the others.)

(i) Given a value in x3005, set its 2's complement in x3006
LD R0, #4
NOT R0, R0
ADD R0, R0, #1
ST R0, #2

(ii) Get an input from the keyboard and store it at x3030 (GETC = x20)
GETC
ST R0, x0028

(iii) Given two values in x3004 and x3005 store their sum in x3006
LD R0, #3
LD R1, #3
ADD R0, R0, R1
ST R0, #2

(iv) Given two values in x3004 and x3005 store their product in x3006
LD R0, #9
LD R1, #9
ADD R2, R1, #0
ADD R1, R1, R2
ADD R0, R0, #-1
BRp #-3
ST R0, #5

(d) **String Programming**: After the following code executes, what will be printed to the console? Assume that the data in the table has already been loaded into memory when the code is run.

.ORIG x3000
LD R0, STRING
PUTS
STRING
.FILL 0x3047
.END
Hi (Null Terminated)

Address	Data	ASCII Representation
x3046	x0021	'!'
x3047	x0048	'H'
x3048	x0069	'i'
x3049	x0000	
x304A	x0074	't'
x304B	x0068	'h'
x304C	x0065	'e'
x304D	x0072	'r'
x304E	x0065	'e'
x304F	x0021	'!'
x3050	x0000	

(i) How many memory locations will the following string use?
.STRINGZ "AMONG US"
9 (1 Extra for null termination)

(e) **Executing ALU Instructions**: The following sequence of instructions is executed with the initial register values shown below.

x3000: 0001000001000011
x3001: 0101011011000001
x3002: 0101001000100000
R0: xA25B R1: x1B2A R2: x4875 R3: x6227

(i) Give the contents of the following registers after the instruction sequence above finishes executing. Convert your answers to hexadecimal. R0: x7D51 R1: x0000 R2: x4875 R3: x0222

(f) **LC3 Missing Instruction**

(i) An LC-3 computer starts with the following register and memory contents. In addition, the word at address x3000, which is not shown, contains an instruction, one of ADD, AND, BR, LD, LDI, LDR, LEA, NOT, ST, STI, or STR. Note: The LEA instruction does not change the CC register (3rd edition behavior).

Registers	**Memory**	
R0: x619F		
R4: x48AA		
R1: x4295	x3001: x4333	
R5: x49FC	x3002: x1BDC	
R2: x259D		
R6: x54C2		
R3: x4000	x4000: x2911	
R7: xF914	x4001: x68A8	
PC: x3000		
CC: b001		

Table 11: Before: Registers and Memory

All other memory locations start with x0000. After the instruction at address x3000 is executed, the contents of registers and memory are:

Registers	**Memory**	
R0: x619F		
R4: x48AA		
R1: x4295	x3001: x4333	
R5: x49FC	x3002: x1BDC	
R2: x259D		
R6: x48B8		
R3: x4000	x4000: x2911	
R7: xF914	x4001: x68A8	
PC: x3001		
CC: b001		

Table 12: After: Registers and Memory

What value could be stored at address x3000? Give your answer in hexadecimal.

x3000 = x1D2E; ADD R6, R4, #14

(ii) An LC-3 computer starts with the following register and memory contents. In addition, the word at address x3000, which is not shown, contains an instruction, one of ADD, AND, BR, LD, LDI, LDR, LEA, NOT, ST, STI, or STR. Note: The LEA instruction does not change the CC register (3rd edition behavior).

Registers	**Memory**	
R0: x619F		
R4: x48AA		
R1: x4295	x3001: x4333	
R5: x49FC	x3002: x1BDC	
R2: x259D		
R6: x54C2		
R3: x4000	x4000: x2911	
R7: xF914	x4001: x619F	
PC: x3000		
CC: b001		

Table 13: Before: Registers and Memory

All other memory locations start with x0000. After the instruction at address x3000 is executed, the contents of registers and memory are:

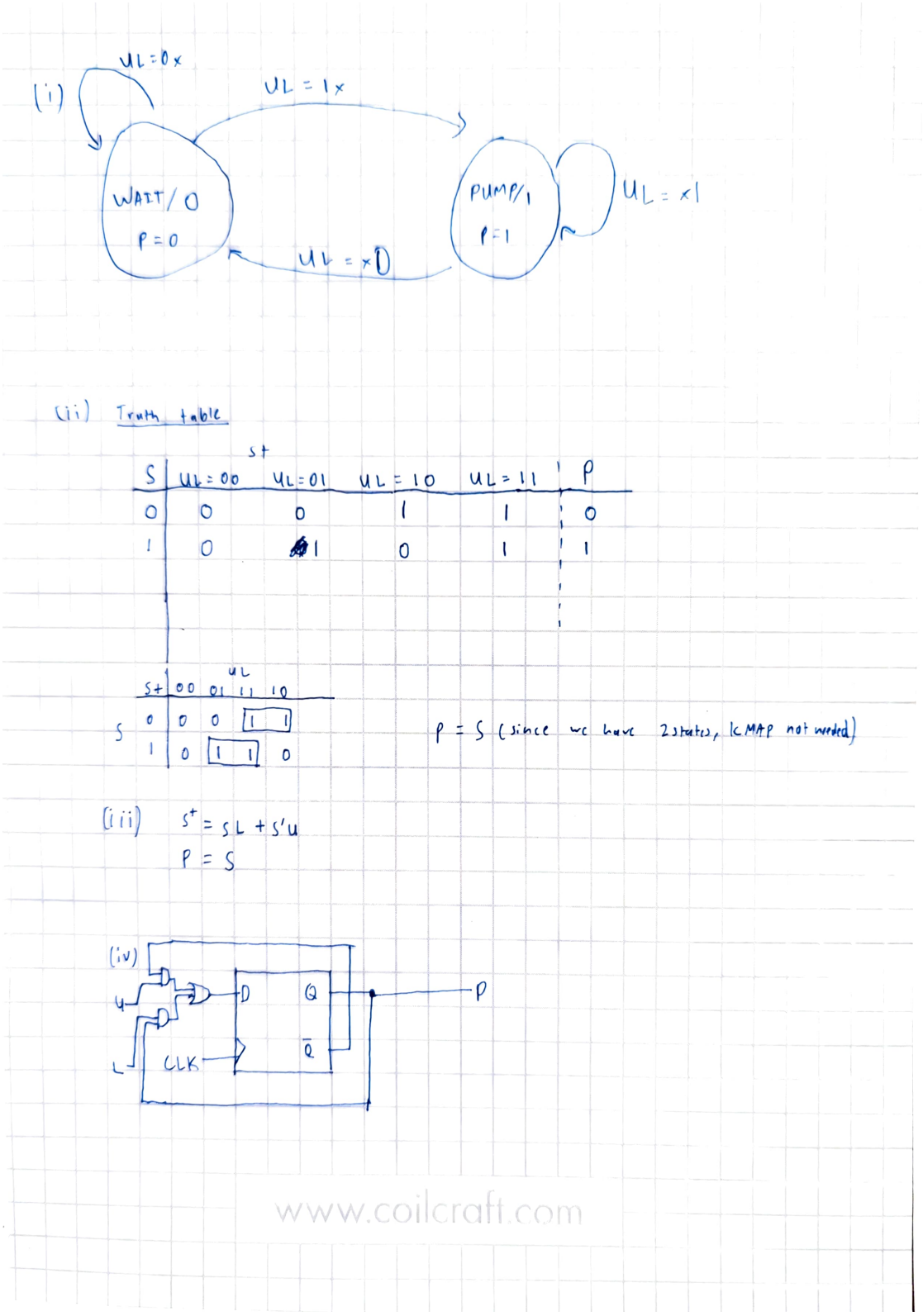
Registers	**Memory**	
R0: x619F		
R4: x48AA		
R1: x4295	x3001: x4333	
R5: x49FC	x3002: x1BDC	
R2: x259D		
R6: x48B8		
R3: x4000	x4000: x2911	
R7: xF914	x4001: x619F	
PC: x3001		
CC: b001		

Table 14: After: Registers and Memory

What value could be stored at address x3000? Give your answer in hexadecimal.

x3000 = x70C1; STR R0, R3, #1

7 Fortuitous FSMs



8 Liberating Loading

(a) A memory address, pointing to the start of an array is stored at x30AD. Which load instruction type would you use to access the first array value? Write the single LC-3 assembly instruction to accomplish this when the PC is at x3090.

Instruction: Load Indirect (LDI)

LDI R1, #28; R1 = M[M[PC + 0x1C]]

(b) What state numbers of the LC-3 FSM are used for a LDR instruction? Which of these states are unique only to LDR?

LDR States: 18, 33, 35, 32, 6, 25, 27

Only State 6 is unique to the LDR instruction.

(c) Fill in the following table with the correct control signals on the LC-3 data path for the LDR Instruction. Note: This table doesn't have all the states for LDR.

States	**18**	**25**	**27**
LD.BEN	0	0	0
LD.MAR	1	0	0
LD.MDR	0	1	0
LD.IR	0	0	0
LD.PC	1	0	0
LD.REG	0	0	1
LD.CC	0	0	1
GateMARMux	0	0	0
GateMDR	0	0	1
GateALU	0	0	0
GatePC	1	0	0
MARMUX	X	X	X
PCMUX	00	XX	XX
ADDR1MUX	X	X	X
ADDR2MUX	XX	XX	XX
DRMUX	XX	XX	00
SR1MUX	XX	XX	XX
ALUK	XX	XX	XX
MIO.EN	0	1	0
R.W	X	0	X

9 Manic Multiplying

You are going to complete a program that multiplies two numbers from memory together and saves the result back into memory. Below are the memory locations for the inputs and output. While you don't have to specify a .ORIG instruction for this problem, it is relevant that the program begins at x3000.

Meaning	Address
Multiplicand 1	x3FFE
Multiplicand 2	x3FFF
Product	x4000

(a) Load Multiplicand 1 and Multiplicand 2 into R1 and R2, respectively. This will require 2 LC-3 instruction lines and 2 pre-processor instruction lines (ie .STRINGZ, .FILL, etc).

LDI R1, MULTI1 ; R1 = M[M[MULTI1]]
LDI R2, MULTI2 ; R2 = M[M[MULTI2]]
MULTI1 .FILL x3FFE ;
MULTI2 .FILL x3FFF ; Load pointers to the multiplicands

You cannot use the LD instruction here since x3FFE and x3FFF are beyond the 9 bit offset (+/- 255) from the PC. As a reminder, the PC is located around x3000.

(b) Create a loop to multiply the multiplicands together and place the result in R3. You are allowed to modify R1 and R2 as needed.

AND R3, R3, #0 ; R3 = R3 AND 0 — Zero out R3
LOOP ADD R3, R3, R1 ; R3 = R3 + R1
ADD R2, R2, #-1 ; R2 = R2 - 1
BRp LOOP ; PC = LOOP — Loops to make next interactive add

(c) Store R3 into the product memory location. this will require 1 LC-3 instruction line and 1 pre-processor instruction line.

STI R3, PRODUCT ; M[M[PRODUCT]] = R3
PRODUCT .FILL x4000 ; Load pointer to the product location

10 Disorienting Decoding

(a) In the LC-3, if the IRD control signal is set to 1, what happens to the contents of the instruction register (IR) during the decoding process? Specifically, how are the bits IR[15:12] utilized?

When IRD is set to 1, it indicates that the microsequencer is in the decode stage of the instruction cycle. The bits IR[15:12], which represent the opcode of the instruction, are directly used to determine the next state in the microsequencer. This state will be one where the specific type of instruction (determined by the opcode) starts its execution.

(b) Given a situation where COND is set to 001, which represents checking for a positive condition code (P), and the instruction's execution previously set the P flag to 1, what would be the value of BEN (branch enable)?

If COND is 001, it checks for the positive (P) condition code. If the P flag is set to 1 (true), BEN would also be set to 1. This means the branch condition is satisfied, and if the instruction is a branch instruction, the branch will take place.

(c) Suppose the current instruction has an opcode of 1101 in IR[15:12], corresponding to a LEA instruction. If the J bits in the microinstruction are set to 011111, explain how the next state of the microsequencer is determined.

In the case of an opcode 1101 for a LEA instruction, the J bits being 011111 indicates complex conditions for state transition. The LEA instruction typically does not depend on condition flags; instead, the instruction's behavior is largely straightforward. The J bits set to 011111 essentially direct the microsequencer to jump to the next sequential state unconditionally, ignoring any condition codes.

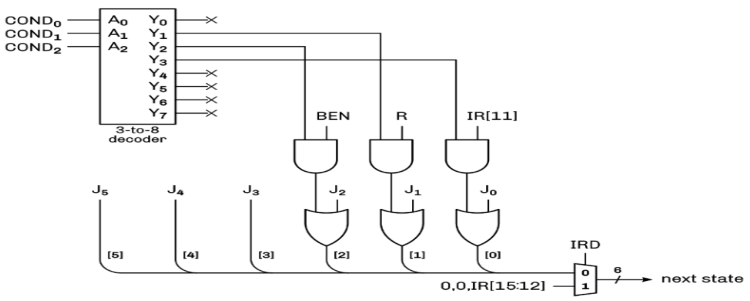


Figure 1: LC-3 Microsequencer

11 Conspicuous Control Signals

(a) Complete the following table by entering values (0, 1, or X) for the LC-3 microinstructions at ROM addresses 4 and 7

ROM Address	IRD	COND(3)	J(6)	LD.BEN	LD.MAR	LD.MDR	LD.IR	LD.PC	LD.REG	LD.CC	GateMARMUX	GateMDR	GateALU	GatePC	MARMUX	PCMUX(2)	ADDR1MUX	ADDR2MUX(2)	DRMUX(2)	SR1MUX(2)	ALUK(2)	MIO.EN	R.W
4	0	011	010100	0	0	0	0	0	1	0	0	0	0	1	x	xx	x	xx	01	xx	xx	0	x
7	0	000	010111	0	1	0	0	0	0	0	1	0	0	0	1	xx	1	01	xx	01	xx	0	x

(b) If GateALU=1, what should the other Gate signals be
All other Gate signals should be 0

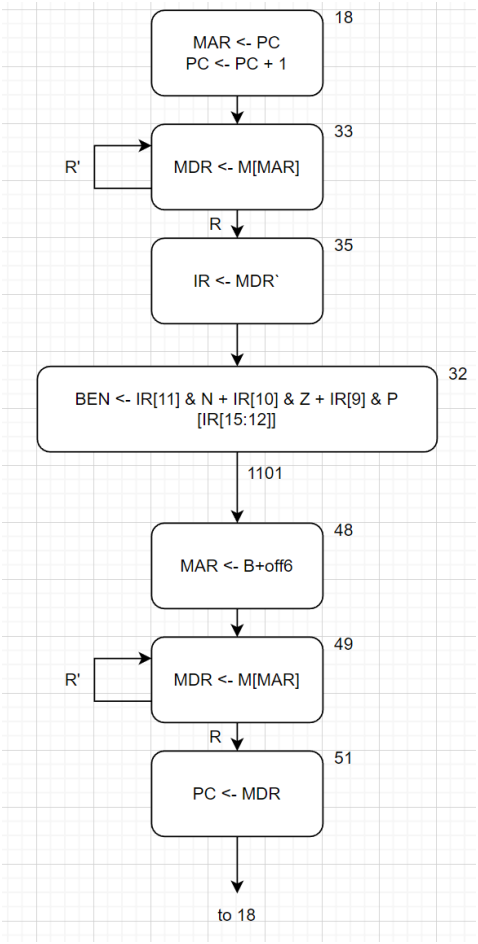
12 Novel New Instructions

We want to implement the following instruction:
PC ⟵ PC + M[BaseR + PCOffset6] + 1

Note: the +1 is to accommodate state 18.
Note 2: In the very rare case that you actually read the textbook, pretend Appendix C.6.3 did not exist.

(a) We know that LC-3 has one unused opcode, the opcode 1101. When this opcode is decoded, suppose that the LC-3 FSM enters state 48 (110000). How many FSM states will this instruction need? What will each state do? Assign them in the 48-55 range.
We assign the states as follows:
48 (0x30): $MAR \longleftarrow B + off6$
49 (0x31): $MDR \longleftarrow M[MAR]$
51 (0x33): $PC \longleftarrow MDR$

(b) How will each instruction transition to each other state? Draw a state diagram.



(c) Using K-maps or another method, write an expression for the next state transition that can be used after entering your first state and before entering state 18. (Hint: $S_5S_4S_3 = 110$, and memory ready is the signal R. you can work with the 4 bits $RS_2S_1S_0$)
We can first write out our desired transitions:
110000 $\longrightarrow$ 110001
110001 $\longrightarrow$ 110011 if R
110001 $\longrightarrow$ 110001 if R'
110011 $\longrightarrow$ 010010
Then, one possible solution is:
$S_5' = (S_5S_4S_3'S_2'S_1S_0)'$
$S_4' = 1$
$S_3' = 0$
$S_2' = 0$
$S_1' = S_1 + RS_0$
$S_0' = S_1'$

(d) Using a similar table as in Problem 8, write down the control signals for each state.
Only relevant signals are shown. Assume other signals are 0 or X.

State	GateMARMUX	GateMDR	LD.MAR	LD.MDR	LD.PC	MARMUX
48	1	0	1	0	0	1
49	0	0	0	1	0	0
51	0	1	0	0	1	0

State	ADDR1MUX	ADDR2MUX	SR1MUX	PCMUX	MIO.EN	R.W
48	1	01	01	XX	0	X
49	X	XX	XX	XX	1	0
51	X	XX	XX	01	0	X

(e) Trace through your instruction with the following instruction:
1101 000 101 000111
Assume PC = x3001, R5 = x2000, and M[x2007] = x8EED.
What is PC after the instruction completes and before the next execution of state 18?
We want to perform the operation PC $\longleftarrow$ PC + M[BaseR + PCOffset6] + 1.
We recognize that BaseR = R5, PCOffset6 = 000111 = x07. Then,
PC $\longleftarrow$ x3001 + M[x2000 + x07] + 1
PC $\longleftarrow$ x3001 + x8EED + 1
PC $\longleftarrow$ xBEEF

13 Challenging Conundrums (Just For Fun)

Unless otherwise specified, you may use any registers you would like and store results in any register

(a) Using only three lines of code, set R0 = 0, R1 = 4, and R2 = -3
AND R0, R0, #0
ADD R1, R0, #4
ADD R2, R0, #-3

(b) Implement R1 NOR R2 in only 3 lines of code
NOT R1, R1
NOT R2, R2 ; By DeMorgan's Law,
AND R3, R2, R1 ; (R1 + R2)' = R1'·R2'

Result is stored in R3.

(c) Implement R3 XOR R4 in only 5 lines of code
NOT R5, R3 ; R5 ← R3'
AND R5, R4, R5 ; R5 ← R3'·R4
NOT R6, R4 ; R6 ← R4'
AND R6, R3, R6 ; R6 ← R3·R4'
ADD R6, R5, R6 ; R6 ← R3'·R4 + R3·R4' = R3 XOR R4

We are guaranteed not to encounter overflow (a 1 + 1 addition) in the final step (prove this to yourself!)
Result is stored in R6

(d) Implement R5 OR R6 in only 4 lines of code
NOT R5, R5 ; R5 ← R5'
NOT R6, R6 ; R6 ← R6'
AND R4, R5, R6 ; R4 ← R5'·R6' = R5 NOR R6
NOT R4, R4 ; R4 ← (R5'·R6')' = R5 OR R6

Result is stored in R4