Department of Electrical and Computer Engineering
The University of Texas at Austin

EE 306, Fall 2015
Yale Patt, Instructor
Stephen Pruett, Siavash Zangeneh, Kamyar Mirzazad, Esha Choukse, Ali Fakhrzadegan, Zheng Zhao, Steven Flolid, Nico Garofano, Sabee Grewal, William Hoenig, Adeesh Jain, Matthew Normyle
Final Exam, December 11, 2015

Name: Solution

Part A:

Problem 1 (10 points):_______

Problem 2 (10 points):_______

Problem 3 (10 points):_______

Problem 4 (10 points):_______

Problem 5 (10 points):_______ **Part A (50 points):**

Part B:

Problem 6 (10 points):_______

Problem 7 (15 points):_______

Problem 8 (20 points):_______

Problem 9 (25 points):_______ **Part B (70 points):**

Note: Please be sure that your answers to all questions (and all supporting work that is required) are contained in the space provided.

Note: Please be sure your name is recorded on each sheet of the exam.

I will not cheat on this exam.

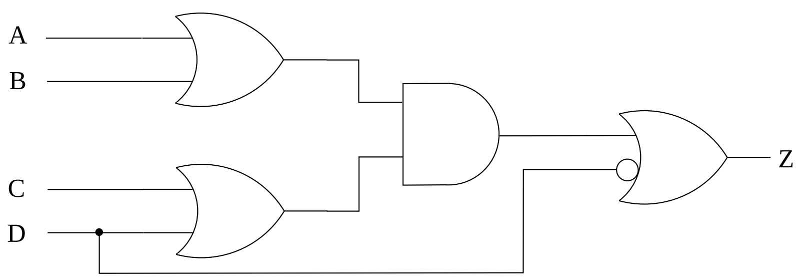
____
Signature

GOOD LUCK!
(HAVE A GREAT SEMESTER BREAK)

Name: Solution

Problem 1. (10 points):

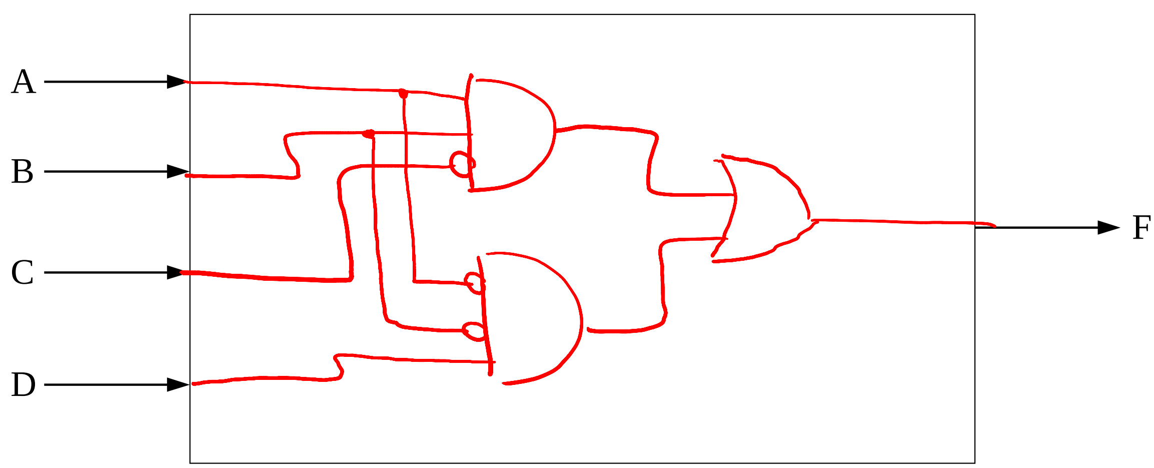
Part a. (5 points):
Construct the truth table for the following logic circuit.



A	B	C	D	Z
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
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

Part b. (5 points):
Cosntruct a logic circuit to implement the function:

```
f = (A AND B AND (NOT C)) OR ((NOT A) AND (NOT B) AND D)
```



Name: Solution

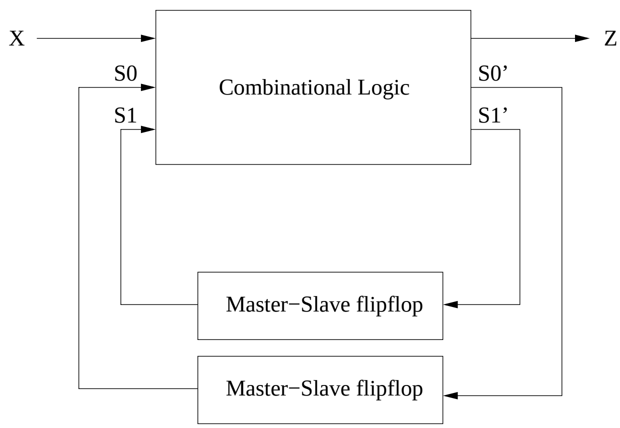
Problem 2. (10 points):
Assemble the following program. You may not need every space provided.

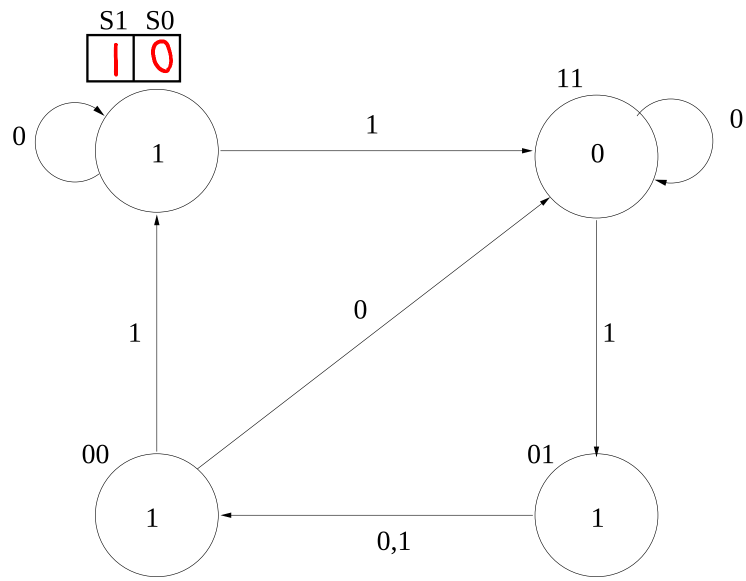
```
        .ORIG x3000
        LEA  R1, BANNER
        AND  R2, R2, #0
 AGAIN  ADD R2, R2, #1
        LDR  R0, R1, #0
        BRnp AGAIN
  DONE  ST   R2, RESULT
        HALT
;
BANNER  .STRINGZ "Bevo"
RESULT  .BLKW #3
MASK    .FILL x0030
        .END
```

x3000	1	1	1	0	0	0	1	0	0	0	0	0	0	1	1	0
x3001	0	1	0	1	0	1	0	0	1	0	1	0	0	0	0	0
x3002	0	0	0	1	0	1	0	0	1	0	1	0	0	0	0	1
x3003	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0
x3004	0	0	0	0	1	0	1	1	1	1	1	1	1	1	0	1
x3005	0	0	1	1	0	1	0	0	0	0	0	0	0	1	1	0
x3006	1	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1
x3007	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0
x3008	0	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1
x3009	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	0
x300A	0	0	0	0	0	0	0	0	0	1	1	0	1	1	1	1
x300B	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
x300C																
x300D																
x300E																
x300F	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0
x3010																
x3011																
x3012																
x3013																

Name: Solution

Problem 3. (10 points): Shown below are the block diagram and the state diagram for a simple four state machine.

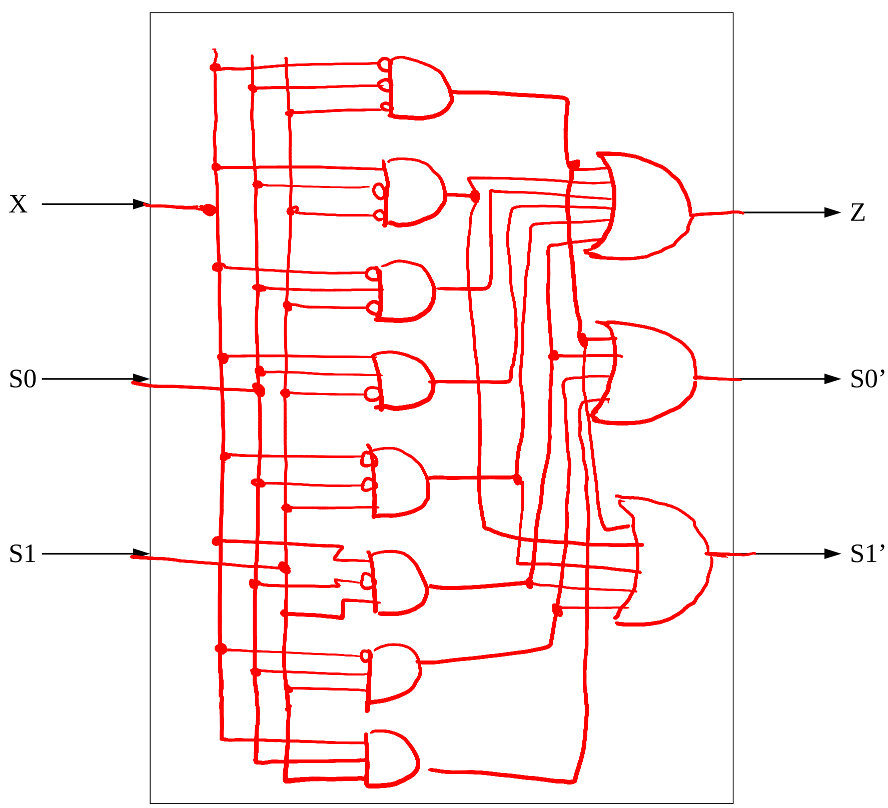




Note that upper left hand state is missing its label. Write it in the box provided.

Fill in the state table, and draw the logic circuits required to implement this state machine.

S1	S0	X	S1'	S0'	Z
0	0	0	1	1	1
0	0	1	1	0	1
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	1	1	1
1	1	0	1	1	0
1	1	1	0	1	0



Name: Solution

Problem 4. (10 points):

The instruction cycle of only one of the LC-3 instructions requires all three of the states 18, 32, and 15 to execute that instruction.

Part a. (1 points): What is the assembly language name for that instruction's opcode?

TRAP

Part b. (9 points): The table below consists of three rows, one each for states 18, 32, and 15. The columns identify the control signals of the LC-3.

Your job: Fill in the entries in the table.

If it does not matter what value is in that entry, put an x in that entry.

Note: Table C.1 at the back of your exam lists the signal values for each control signal. Use the signal value names specified in Table C.1 as entries in the table you fill in below.

State	LD.PC	LD.MAR	LD.MDR	LD.REG	GatePC	GateMARMUX	GateMDR	MARMUX	PCMUX	DRMUX	MIO.EN	R.W
18	1	1	0	0	1	0	0	X	PC+1	X	0	X
32	0	0	0	0	0	0	0	X	X	X	0	X
15	0	1	0	0	0	1	0	0	X	X	0	X

Name: Solution

Problem 5. (10 points):
The following program, after you insert the two missing instructions, will examine a list of positive integers stored in consecutive sequential memory locations and store the smallest one in location x4000. The number of integers in the list is contained in memory location x4001. The list itself starts at memory location x4002. Assume the list is not empty (i.e., the contents of x4001 is not zero.)

```
                .ORIG x3000
                LDI R1, SIZE
                LD  R2, LISTPOINTER
                LDR R0, R2, #0
                ADD R1, R1, #-1
                BRz ALMOSTDONE   ; Only one element in the list

AGAIN           ADD R2,R2,#1
                LDR R3,R2,#0
                NOT R4,R3
                ADD R4,R4,#1
                ADD R4,R0,R4
                BRnz  SKIP
                ADD R0,R3,#0

SKIP            ADD R1,R1,#-1

                BRp AGAIN

ALMOSTDONE      LD R5,MIN
                STR R0,R5,#0
                HALT

MIN             .FILL x4000
SIZE            .FILL x4001
LISTPOINTER     .FILL x4002
                .END
```

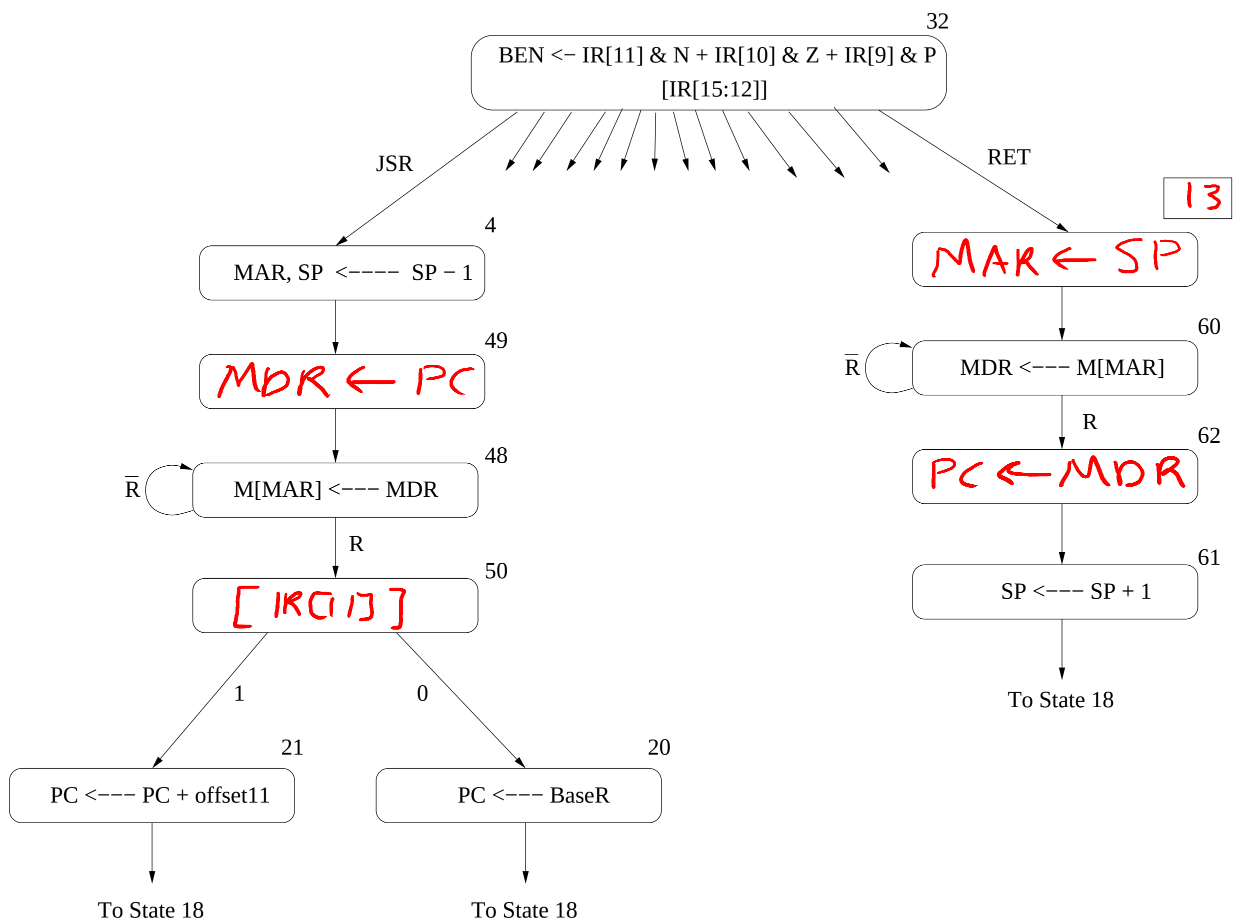
Your job: Insert the two the missing instructions.

Name: Solution

Problem 6. (10 points):
As you know, the LC-3 ISA specifies that the JSR instruction saves the return linkage in R7, and JMP R7 returns to the calling program. Some ISAs prefer to save the return linkage on the stack. There are actually pluses and minuses of doing it that way, which you will learn before you graduate.

Suppose we decide to do that, have JSR cause the return linkage to be pushed on the stack and use the unused LC-3 opcode as the RET to pop the return linkage from the stack.

Your job: Fill in the boxes in the state machine below to implement JSR(R), RET if we implement the call/return mechanism by saving the return linkage on the stack.



Name: Solution

Problem 7. (15 points): The following user program (priority 0) is assembled and loaded into memory.

```
        .ORIG x8000
        LD R0, Z
AGAIN   ADD R0, R0, #-1
        BRnp AGAIN
        LD R0, W
        BRp L1
        LD R0, X
        TRAP x21
        BRnzp DONE
L1      LEA R0, Y
        TRAP x22
DONE    HALT

X      .FILL x34
Y      .STRINGZ "OOOOPS!"
Z      .FILL x100
W      .BLKW #1
       .END
```

Before this code executes, two things happen: (a) another program loads a value into W, and (b) a breakpoint is set at the address DONE.

Then the run switch is hit and the program starts executing. Before the computer stops due to the breakpoint, several interrupts occur and their corresponding service routines are executed. Finally, the LC-3 stops due to the breakpoint. We examine the memory shown, and R6, the supervisor stack pointer.

	Memory
x2FF8	x0601
x2FF9	x0601
x2FFA	x0500
x2FFB	x0504
x2FFC	x0204
x2FFD	x0201
x2FFE	x8004
x2FFF	x8002
x3000	x8010
x3001	x8012

R6 x3000

Question: What does the user program write to the monitor? How do you know that?

The user program writes "4" to the monitor
Looking at the system stack, PC = x8004 and PSR = x8002
right before the last user interrupt was taken. Therefore, last
interrupt returns to "BRp L1", but the branch is not taken since
p bit in PSR is not set

Name: Solution

Problem 8. (20 points):
Your job in this problem will be to add the missing instructions to a program that detects palindromes. Recall a palindrome is a string of characters that are identical when read from left to right or from right to left. For example, racecar and 112282211. In this program, we will have no spaces and no capital letters in our input string – just a string of lower case letters.

The program will make use of both a stack and a queue. The subroutines for accessing the stack and queue are shown below. Recall that elements are PUSHed (added) and POPped (removed) from the stack. Elements are ENQUEUEd (added) to the back of a queue, and DEQUEUEd (removed) from the front of the queue.

```
        .ORIG x3050
PUSH    ADD R6, R6, #-1
        STR R0, R6, #0
        RET
POP     LDR R0, R6, #0
        ADD R6, R6, #1
        RET
STACK   .BLKW #20
        .END
```

```
        .ORIG x3080
ENQUEUE ADD R5, R5, #1
        STR R0, R5, #0
        RET
DEQUEUE LDR R0, R4, #0
        ADD R4, R4, #1
        RET
QUEUE   .BLKW #20
        .END
```

The program is carried out in two phases. Phase 1 enables a user to input a character string one keyboard character at a time. The character string is terminated when the user types the enter key (line feed). In Phase 1, the ASCII code of each character input is pushed on a stack, and its negative value is inserted at the back of a queue. Inserting an element at the back of a queue we call enqueuing.

In Phase 2, the characters on the stack and in the queue are examined by removing them, one by one from their respective data structures (i.e., stack and queue). If the string is a palindrome, the program stores a 1 in memory location RESULT. If not, the program stores a zero in memory location RESULT. The PUSH and POP routines for the stack as well as the ENQUEUE and DEQUEUE routines for the queue are shown below. You may assume the user never inputs more than 20 characters.

The program for detecting palindromes (with some instructions missing) are on the next page.

Your job, as stated earlier, is to fill in the missing instructions.

Name: Solution

```
        .ORIG X3000
        LEA   R4, QUEUE
        LEA   R5, QUEUE
        ADD   R5, R5, #-1
        LEA   R6, ENQUEUE        ; Initialize SP
        LD    R1, ENTER
        AND   R3, R3, #0
;
        LEA R0, PROMPT
        TRAP x22
PHASE1  TRAP x20
        ADD R2, R0, R1
        BRz PHASE2
        JSR PUSH
        NOT R0, R0
        ADD R0, R0, #1
        JSR ENQUEUE
        ADD R3, R3, #1
        BRnzp PHASE1
;
PHASE2  JSR POP
        ADD R1, R0, #0
        JSR DEQUEUE
        ADD R1, R0, R1
        BRnp FALSE
        ADD R3, R3, #-1
        BRz TRUE
        BRnzp PHASE2
;
TRUE    AND R0, R0, #0
        ADD R0, R0, #1
        ST R0, RESULT
        HALT
FALSE   AND R0, R0, #0
        ST R0, RESULT
        HALT
RESULT  .BLKW #1
ENTER   .FILL x-0A
PROMPT  .STRINGZ "Enter an input string: "
        .END
```

Name: Solution

Problem 9. (25 points):
Recall Problem 5 on Midterm 2. Dr. Patt liked that type of problem so much, we are going to try it again. We have a program with some missing instructions, and we have a table consisting of some information and some missing information associated with five specific clock cycles of the program's execution. Your job is to complete both!

Part a: As on the second midterm, insert the missing instructions in the program and the missing information in the table. Cycle numbering starts at 1. That is, cycle 1 is the first clock cycle of the processing of LD R0,A. Note that we have not said anything about the number of clock cycles a memory access takes. You do have enough information to figure that out for yourself. Note that we are asking for the value of the registers DURING each clock cycle.

```
        .ORIG x3000
        LD   R0, A
        LD   R1, B
        NOT  R1, R1
        ADD  R1, R1, #1
        AND  R2, R2, #0
AGAIN   [ADD R2,R2,#1]
        [ADD R0,R1,R0]
        BRn DONE
        BRnzp AGAIN
DONE    ST   R2, C
        HALT
A       .FILL #5
B       .FILL [#2]
C       .BLKW #1
        .END
```

Cycle Number	State Number	Information					
11	27	LD.REG:	1	DRMUX:	IR[11:9]	GateMDR:	1
		LD.CC:	1	GateALU:	0	GatePC:	0
16	35	LD.MDR:	0	MDR:	x220A	IR:	x200A
		LD.IR:	1				
50	1	LD.REG:	1	MDR:	x14A1	DRMUX:	IR[11:9]
		BUS:	x0001			GateMDR:	0
57	1	PC:	x3007	IR:	x1040	GateALU:	1
		BUS:	x0003			GatePC:	0
123	22	ADDR1MUX:	PC	ADDR2MUX:	PCoffset9		
		LD.PC:	1	PC:	x3008	PCMUX:	ADDER

Name: Solution

Part b: What is stored in C at the end of execution for the specific operands given in memory locations A and B?

#3

Part c: Actually, the program was written by an Aggie, so as expected, he did not get it quite right. Almost, but not quite! Your final task on this problem is to examine the code, figure out what the Aggie was trying to do, point out where he messed up, and how you would fix it. It is not necessary to write any code, just explain briefly how you would fix it.

What was the Aggie trying to do?

Trying to divide A by B, put result in C
Throws away the remainder

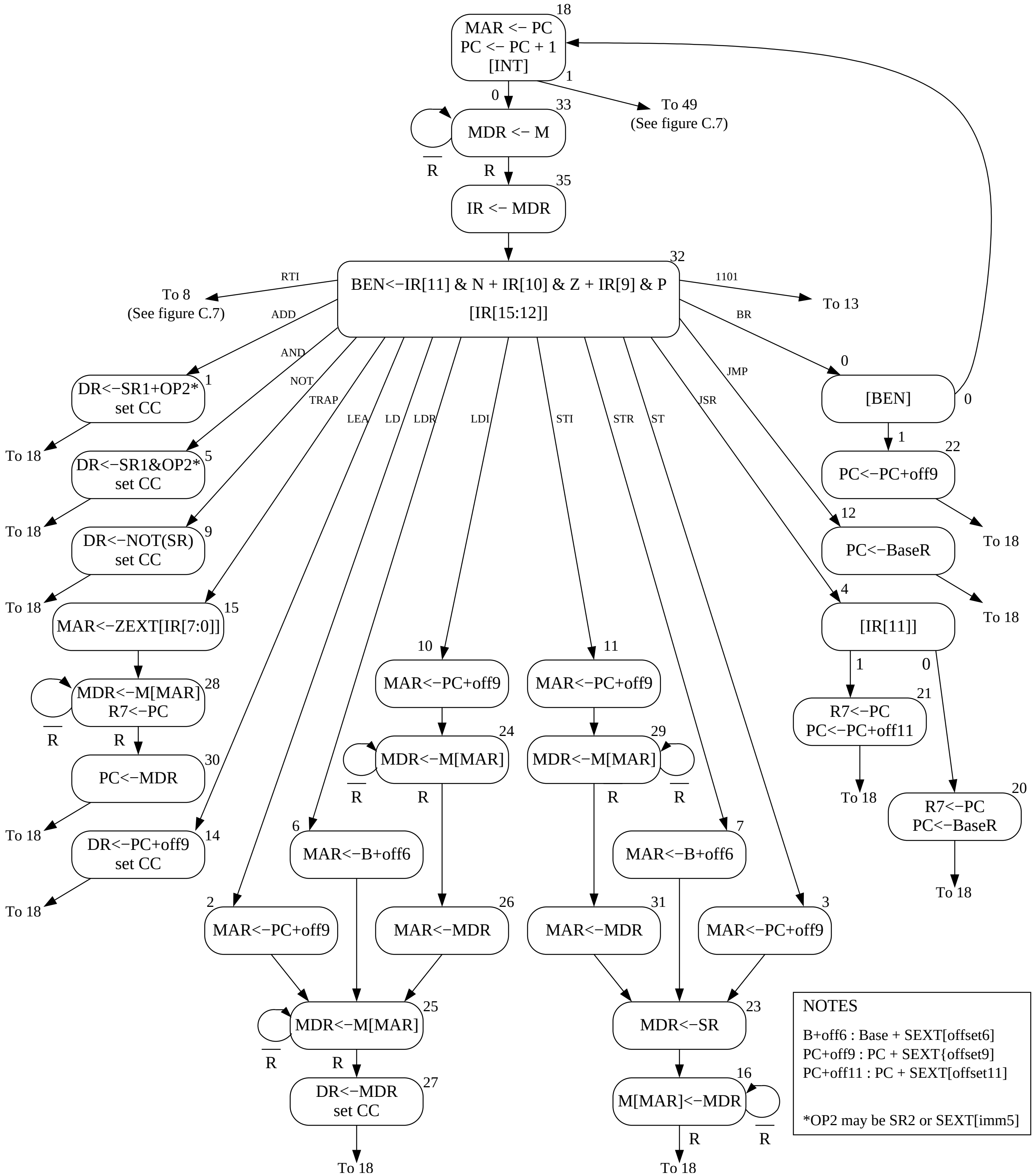
How did the Aggie mess up?

Output is always off by 1

How would you fix his program?

Subtract 1 before storing the result
OR: Move "ADD R2, R2, #1" to the instruction after "BRn DONE" and before "BRnzp AGAIN"

NOTE: There are many valid answers for this question



18
MAR <− PC
PC <− PC + 1
[INT]
1
To 49
(See figure C.7)
0
33
MDR <− M
R
R
35
IR <− MDR
32
BEN<−IR[11] & N + IR[10] & Z + IR[9] & P
[IR[15:12]]
RTI
To 8
(See figure C.7)
1101
To 13
ADD
AND
NOT
TRAP
LEA
LD
LDR
LDI
STI
STR
ST
JSR
JMP
BR
1
DR<−SR1+OP2*
set CC
To 18
5
DR<−SR1&OP2*
set CC
To 18
9
DR<−NOT(SR)
set CC
To 18
15
MAR<−ZEXT[IR[7:0]]
28
MDR<−M[MAR]
R7<−PC
R
R
30
PC<−MDR
To 18
14
DR<−PC+off9
set CC
To 18
0
[BEN]
0
1
22
PC<−PC+off9
To 18
12
PC<−BaseR
To 18
4
[IR[11]]
1
0
21
R7<−PC
PC<−PC+off11
To 18
20
R7<−PC
PC<−BaseR
To 18
10
MAR<−PC+off9
24
MDR<−M[MAR]
R
R
11
MAR<−PC+off9
29
MDR<−M[MAR]
R
R
6
MAR<−B+off6
7
MAR<−B+off6
2
MAR<−PC+off9
26
MAR<−MDR
31
MAR<−MDR
3
MAR<−PC+off9
25
MDR<−M[MAR]
R
R
27
DR<−MDR
set CC
To 18
23
MDR<−SR
16
M[MAR]<−MDR
R
R
To 18
NOTES
B+off6 : Base + SEXT[offset6]
PC+off9 : PC + SEXT{offset9]
PC+off11 : PC + SEXT[offset11]
*OP2 may be SR2 or SEXT[imm5]

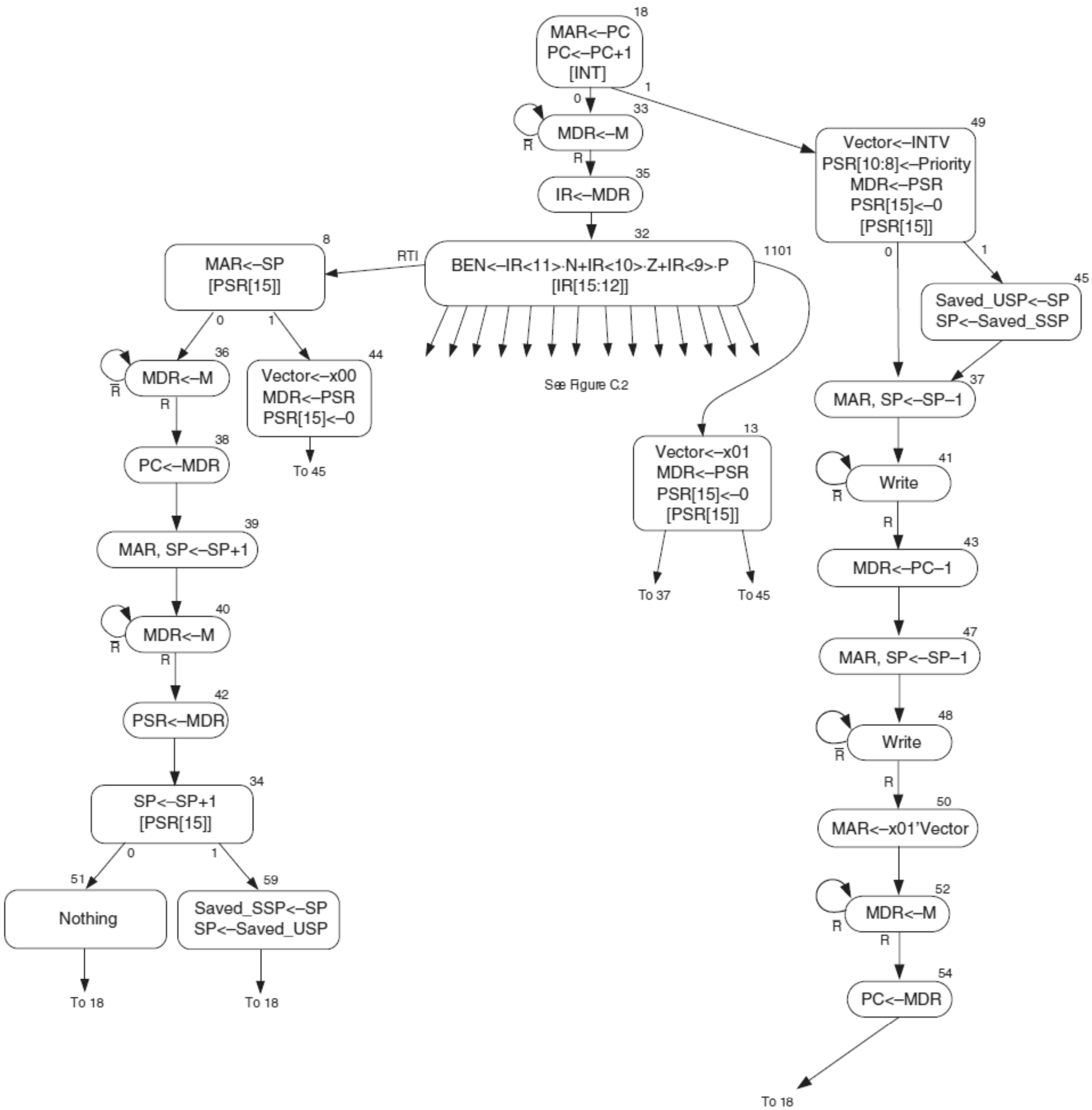
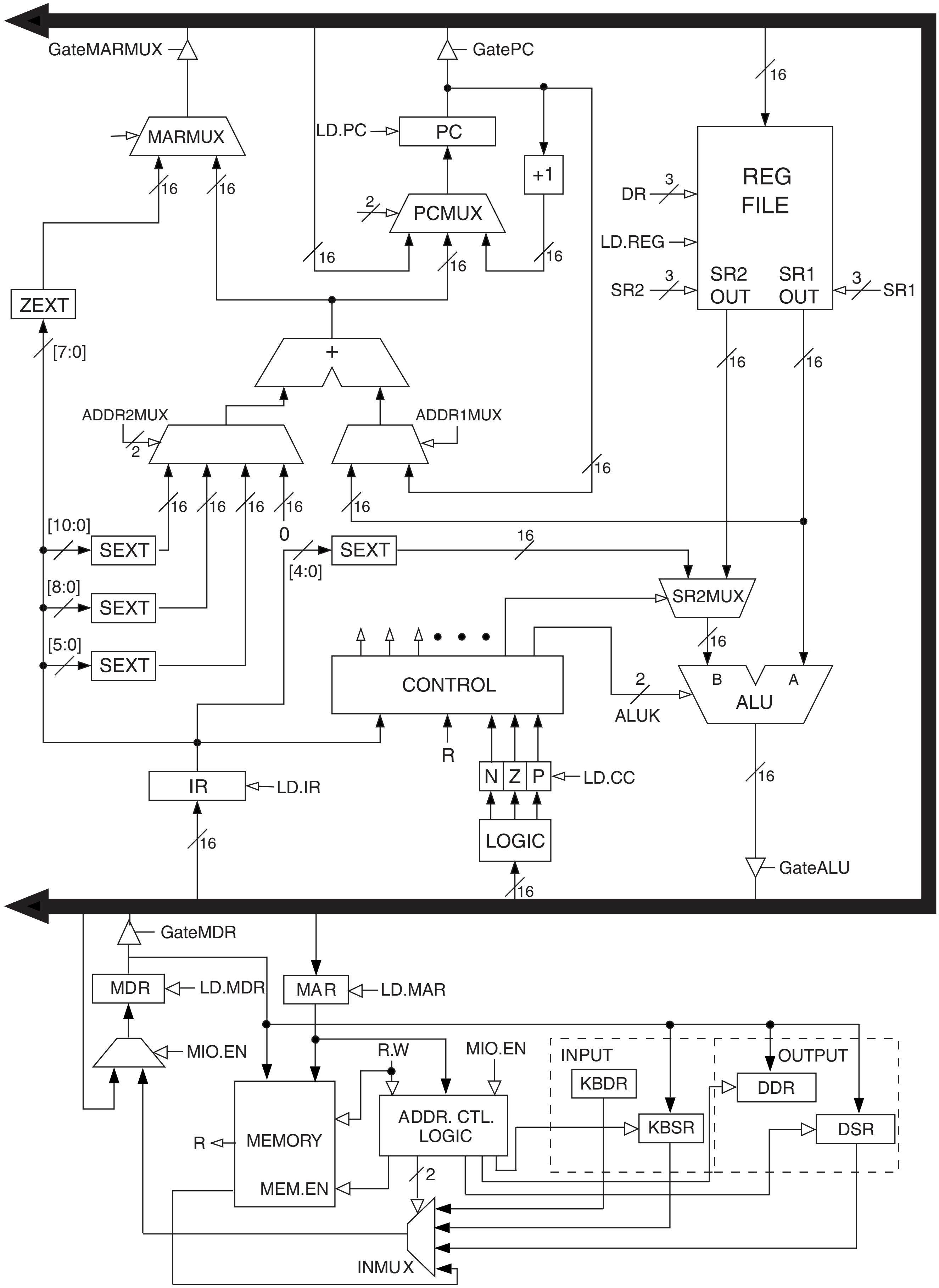
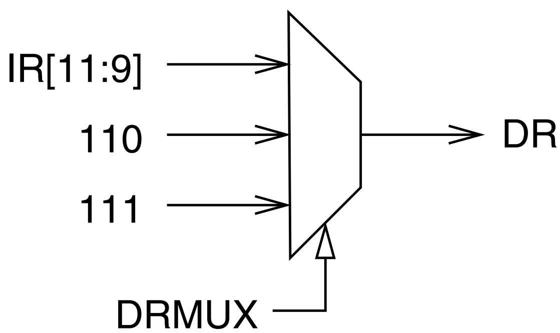
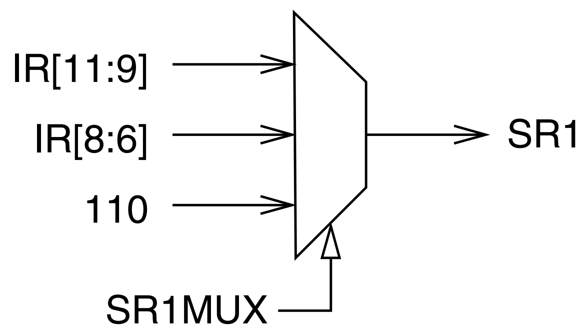


Figure C.7 LC-3 state machine showing interrupt control

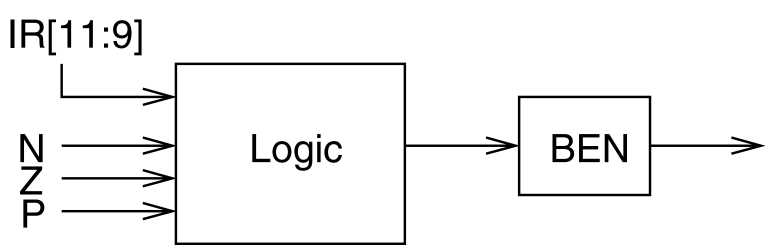

GateMARMUX
GatePC
MARMUX
LD.PC
PC
+1
PCMUX
REG
FILE
DR
LD.REG
SR2
SR2
OUT
SR1
OUT
SR1
16
3
2
ZEXT
[7:0]
ADDR2MUX
ADDR1MUX
+
[10:0]
SEXT
[8:0]
SEXT
[5:0]
SEXT
0
SEXT
[4:0]
SR2MUX
CONTROL
ALU
B
A
ALUK
R
IR
LD.IR
N
Z
P
LD.CC
LOGIC
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
MIO.EN
R.W
MIO.EN
MEMORY
MEM.EN
R
ADDR. CTL.
LOGIC
INPUT
KBDR
KBSR
OUTPUT
DDR
DSR
INMUX


IR[11:9]
110
111
DR
DRMUX

(a)


IR[11:9]
IR[8:6]
110
SR1
SR1MUX

(b)


IR[11:9]
N
Z
P
Logic
BEN

(c)

Table C.1 Data Path Control Signals

Signal Name	Signal Values	
LD.MAR/1:	NO, LOAD	
LD.MDR/1:	NO, LOAD	
LD.IR/1:	NO, LOAD	
LD.BEN/1:	NO, LOAD	
LD.REG/1:	NO, LOAD	
LD.CC/1:	NO, LOAD	
LD.PC/1:	NO, LOAD	
LD.Priv/1:	NO, LOAD	
LD.SavedSSP/1:	NO, LOAD	
LD.SavedUSP/1:	NO, LOAD	
LD.Vector/1:	NO, LOAD	
GatePC/1:	NO, YES	
GateMDR/1:	NO, YES	
GateALU/1:	NO, YES	
GateMARMUX/1:	NO, YES	
GateVector/1:	NO, YES	
GatePC-1/1:	NO, YES	
GatePSR/1:	NO, YES	
GateSP/1:	NO, YES	
PCMUX/2:	PC+1	;select pc+1
	BUS	;select value from bus
	ADDER	;select output of address adder
DRMUX/2:	11.9	;destination IR[11:9]
	R7	;destination R7
	SP	;destination R6
SR1MUX/2:	11.9	;source IR[11:9]
	8.6	;source IR[8:6]
	SP	;source R6
ADDR1MUX/1:	PC, BaseR	
ADDR2MUX/2:	ZERO	;select the value zero
	offset6	;select SEXT[IR[5:0]]
	PCoffset9	;select SEXT[IR[8:0]]
	PCoffset11	;select SEXT[IR[10:0]]
SPMUX/2:	SP+1	;select stack pointer+1
	SP−1	;select stack pointer−1
	Saved SSP	;select saved Supervisor Stack Pointer
	Saved USP	;select saved User Stack Pointer
MARMUX/1:	7.0	;select ZEXT[IR[7:0]]
	ADDER	;select output of address adder
VectorMUX/2:	INTV	
	Priv.exception	
	Opc.exception	
PSRMUX/1:	individual settings, BUS	
ALUK/2:	ADD, AND, NOT, PASSA	
MIO.EN/1:	NO, YES	
R.W/1:	RD, WR	
Set.Priv/1:	0	;Supervisor mode
	1	;User mode

	15 14 13 12	11	10	9	8	7 6	5	4 3	2 1 0
ADD+	0001	DR			SR1		0	00	SR2
ADD+	0001	DR			SR1		1	imm5	
AND+	0101	DR			SR1		0	00	SR2
AND+	0101	DR			SR1		1	imm5	
BR	0000	n	z	p	PCoffset9				
JMP	1100	000			BaseR		000000		
JSR	0100	1	PCoffset11						
JSRR	0100	0	00		BaseR		000000		
LD+	0010	DR			PCoffset9				
LDI+	1010	DR			PCoffset9				
LDR+	0110	DR			BaseR		offset6		
LEA+	1110	DR			PCoffset9				
NOT+	1001	DR			SR		111111		
RET	1100	000			111		000000		
RTI	1000	000000000000							
ST	0011	SR			PCoffset9				
STI	1011	SR			PCoffset9				
STR	0111	SR			BaseR		offset6		
TRAP	1111	0000				trapvect8			
reserved	1101								

Figure A.2 Format of the entire LC-3 instruction set. **Note:** + indicates instructions that modify condition codes

The Standard ASCII Table

ASCII			ASCII			ASCII			ASCII		
Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex	Character	Dec	Hex
nul	0	00	sp	32	20	@	64	40	`	96	60
soh	1	01	!	33	21	A	65	41	a	97	61
stx	2	02	"	34	22	B	66	42	b	98	62
etx	3	03	#	35	23	C	67	43	c	99	63
eot	4	04	$	36	24	D	68	44	d	100	64
enq	5	05	%	37	25	E	69	45	e	101	65
ack	6	06	&	38	26	F	70	46	f	102	66
bel	7	07	'	39	27	G	71	47	g	103	67
bs	8	08	(	40	28	H	72	48	h	104	68
ht	9	09	)	41	29	I	73	49	i	105	69
lf	10	0A	*	42	2A	J	74	4A	j	106	6A
vt	11	0B	+	43	2B	K	75	4B	k	107	6B
ff	12	0C	,	44	2C	L	76	4C	l	108	6C
cr	13	0D	-	45	2D	M	77	4D	m	109	6D
so	14	0E	.	46	2E	N	78	4E	n	110	6E
si	15	0F	/	47	2F	O	79	4F	o	111	6F
dle	16	10	0	48	30	P	80	50	p	112	70
dc1	17	11	1	49	31	Q	81	51	q	113	71
dc2	18	12	2	50	32	R	82	52	r	114	72
dc3	19	13	3	51	33	S	83	53	s	115	73
dc4	20	14	4	52	34	T	84	54	t	116	74
nak	21	15	5	53	35	U	85	55	u	117	75
syn	22	16	6	54	36	V	86	56	v	118	76
etb	23	17	7	55	37	W	87	57	w	119	77
can	24	18	8	56	38	X	88	58	x	120	78
em	25	19	9	57	39	Y	89	59	y	121	79
sub	26	1A	:	58	3A	Z	90	5A	z	122	7A
esc	27	1B	;	59	3B	[	91	5B	{	123	7B
fs	28	1C	<	60	3C	\	92	5C	|	124	7C
gs	29	1D	=	61	3D	]	93	5D	}	125	7D
rs	30	1E	>	62	3E	^	94	5E	~	126	7E
us	31	1F	?	63	3F	_	95	5F	del	127	7F

Table A.2 Trap Service Routines

Trap Vector	Assembler Name	Description
x20	GETC	Read a single character from the keyboard. The character is not echoed onto the console. Its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x21	OUT	Write a character in R0[7:0] to the console display.
x22	PUTS	Write a string of ASCII characters to the console display. The characters are contained in consecutive memory locations, one character per memory location, starting with the address specified in R0. Writing terminates with the occurrence of x0000 in a memory location.
x23	IN	Print a prompt on the screen and read a single character from the keyboard. The character is echoed onto the console monitor, and its ASCII code is copied into R0. The high eight bits of R0 are cleared.
x24	PUTSP	Write a string of ASCII characters to the console. The characters are contained in consecutive memory locations, two characters per memory location, starting with the address specified in R0. The ASCII code contained in bits [7:0] of a memory location is written to the console first. Then the ASCII code contained in bits [15:8] of that memory location is written to the console. (A character string consisting of an odd number of characters to be written will have x00 in bits [15:8] of the memory location containing the last character to be written.) Writing terminates with the occurrence of x0000 in a memory location.
x25	HALT	Halt execution and print a message on the console.

Table A.3 Device Register Assignments

Address	I/O Register Name	I/O Register Function
xFE00	Keyboard status register	Also known as KBSR. The ready bit (bit [15]) indicates if the keyboard has received a new character.
xFE02	Keyboard data register	Also known as KBDR. Bits [7:0] contain the last character typed on the keyboard.
xFE04	Display status register	Also known as DSR. The ready bit (bit [15]) indicates if the display device is ready to receive another character to print on the screen.
xFE06	Display data register	Also known as DDR. A character written in the low byte of this register will be displayed on the screen.
xFFFE	Machine control register	Also known as MCR. Bit [15] is the clock enable bit. When cleared, instruction processing stops.