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
Exam 2, November 11, 2015

Name:_____

Problem 1 (20 points):_____

Problem 2 (15 points):_____

Problem 3 (15 points):_____

Problem 4 (25 points):_____

Problem 5 (25 points):_____

Total (100 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!

Name:_____

Problem 1. (20 points):

Part a. (5 points): Part of the state of the computer is as follows:

R3: x3000	Mem[x4000]: x1234
R4: x4000	Mem[x4001]: x2345
R5: x5000	Mem[x4002]: x3456
	Mem[x4003]: x4567

Then, LDR R5,R4,#2 is executed.

After this instruction is executed, R5 contains [ ]

Part b. (5 points): The program below adds the absolute value of the integer in A to the absolute value of the integer in B, and stores the sum in C. We decide to use the subroutine ABS to take as input the contents of R0, and return its absolute value in R0.

```
        .ORIG x3000
        LD R0, A
        JSR ABS
        ADD R4, R0, #0
        LD R0, B
        JSR ABS
        ADD R0, R4, R0
        ST R0, C
        HALT
   A    .BLKW 1
   B    .BLKW 1
   C    .BLKW 1

  ABS   ADD R4, R0, #0
        BRzp DONE
 SKIP   NOT R4, R4
        ADD R0, R4, #1
 DONE   RET
        .END
```

Why will the above program not work correctly? Please answer in 20 words or fewer.

Name:_____

Part c. (5 points): The following program is assembled, loaded into LC-3 memory, and executed.

```
        .ORIG x3000
        LD R0, A
        LD R1, B
        ADD R0, R1, R0
        ST R0, B
    A   .STRINGZ "%"
    B   .FILL xF000
        .END
```

Does the program halt? If yes, explain what causes the program to halt. If no, explain why the program doesn't halt. Please answer in 20 words or fewer.

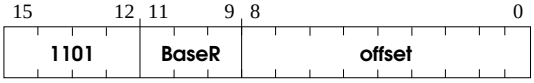
Part d. (5 points): Create the Symbol Table for this piece of code that an Aggie wrote one night when he was drunk.

```
        .ORIG x4000
        LEA   R1, X
AGAIN   ADD   R2, R1, R1
        ST    R2, X
        ADD   R2, R1, R1
        ST    R2, Y
        BRz   AGAIN
        HALT
PROMPT  .STRINGZ "EE306 ROCKS!"
     X  .BLKW 10
     Y  .BLKW 1
     Z  .FILL xAE00
        .END
```

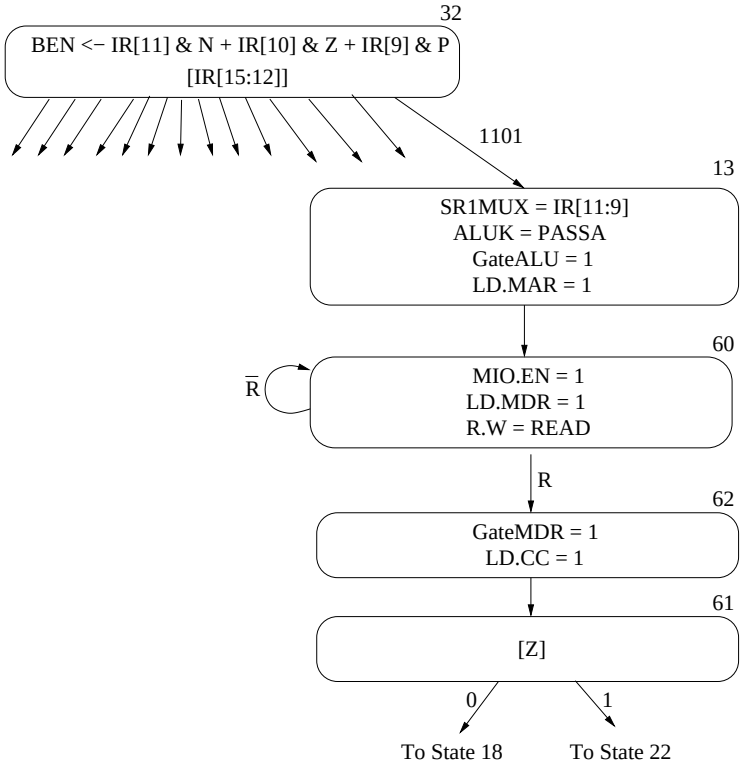
Symbol	Address

Name:_____

Problem 2. (15 points): We want to add a new instruction to the LC-3, using the unused opcode 1101. It will have the following format:



To implement this instruction we add four new states, shown below.



We show in each state the control signals that are needed to implement the processing for that clock cycle. All control signals not shown in a state are assumed to be 0.

Note that from state 61, we branch either to state 18 or state 22.

What does this new instruction do? Be concise, but complete in your answer.

Name:_____

Problem 3. (15 points): We want to support 8 input keyboards instead of 1. To do this we need 8 ready bits in KBSR, and 8 separate KBDRs. We will use the 8 odd-numbered bits in the KBSR as ready bits for the 8 keyboards, as shown below. We will set the other 8 bits in the KBSR to 0.

The 8 memory-mapped keyboard data registers and their corresponding ready bits are as follows:

FE04:	KBSR	
FE06:	KBDR1,	Ready bit is KBSR[1]
FE08:	KBDR2,	Ready bit is KBSR[3]
FE0A:	KBDR3,	Ready bit is KBSR[5]
FE0C:	KBDR4,	Ready bit is KBSR[7]
FE0E:	KBDR5,	Ready bit is KBSR[9]
FE10:	KBDR6,	Ready bit is KBSR[11]
FE12:	KBDR7,	Ready bit is KBSR[13]
FE14:	KBDR8,	Ready bit is KBSR[15]

We wish to write a program that polls the keyboards and loads the ASCII code typed by the highest priority keyboard into R0. That is, if someone had previously typed a key on keyboard 1, we want to load the ASCII code in KBDR1 into R0. If no key was typed on keyboard 1, but a key had been typed on keyboard 2, we want to load the ASCII code in KBDR2 into R0. ...and so on. That is, KB1 has higher priority than KB2, which has higher priority than KB3, which has higher priority than KB4, etc. KB8 has the lowest priority.

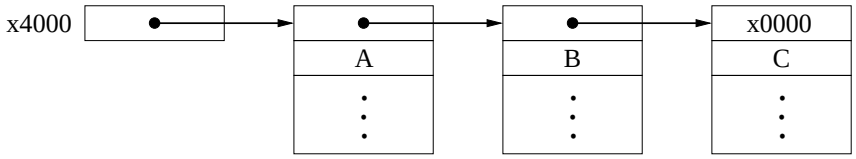
The following program will do the job AFTER you fill in the missing instructions:

```
        .ORIG X3000
        LD    R0, KBDR1
POLL    LDI   R1, KBSR
        BRz   POLL
        AND   R2, R2, #0
        ADD   R2, R2, #2
AGAIN   [                    ]
        BRnp    FOUND
        ADD     R0, R0, #2
        [                    ]
        [                    ]
        BRnp    AGAIN
        HALT
FOUND   [                    ]
        HALT
KBSR    .FILL   xFE04
KBDR1   .FILL   xFE06
        .END
```

Your job: fill in the missing instructions.

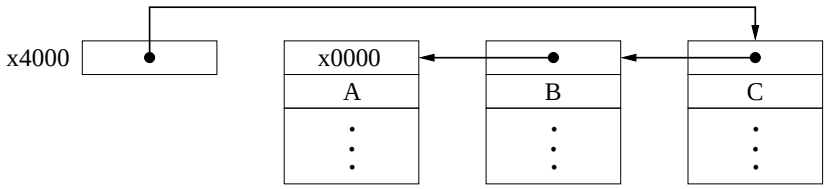
Name:_____

Problem 4. (25 points):
You are given a linked list, consisting of at most 20 elements, as shown below.



Note the listhead is at location x4000.

We want to reverse the nodes of the linked list. For the above linked list, the result would be:



The program on the following page (with missing instructions filled in) does the job, using subroutines PUSH and POP.

Your job: fill in the missing instructions.

Name:_____

```
        .ORIG X3000
        LEA   R6, BASE
        LD    R0, START

PHASE1  LDR   R0, R0, #0
        [                          ]

        JSR   PUSH
        BRnzp PHASE1

PHASE2  LD    R1, START

AGAIN   JSR   POP
        [                          ]

        BRnp  DONE
        [                          ]
        [                          ]

        BRnzp AGAIN

DONE    AND   R0, R0, #0
        [                          ]

        HALT

START   .FILL x4000
STACK   .BLKW #20

BASE .FILL [          ]

PUSH    [                          ]

        STR   R0, R6, #0
        RET

POP     AND   R5, R5, #0
        LD    R0, BASE
        ADD   R0, R0, R6
        BRz   EMPTY
        LDR   R0, R6, #0
        ADD   R6, R6, #1
        RET
EMPTY   ADD   R5, R5, #1
        RET

        .END
```

Name:_____

Problem 5. (25 points): Consider the following program:

```
        .ORIG x3000
        LD  R0, A
        LD  R1, B
        BRz DONE
        [                    ]
        [                    ]
        BRnzp AGAIN
DONE    ST  R0, A
        HALT
A .FILL x0  [          ]

   A    .FILL x0___
   B    .FILL x0001
        .END
```

The program uses only R0 and R1. Note the boxes to indicate two missing instructions. Note also that one of the instructions in the program must be labeled AGAIN and that label is missing.

After execution of the program, the contents of A is x1800.

PROBLEM IS CONTINUED ON THE NEXT PAGE!!!

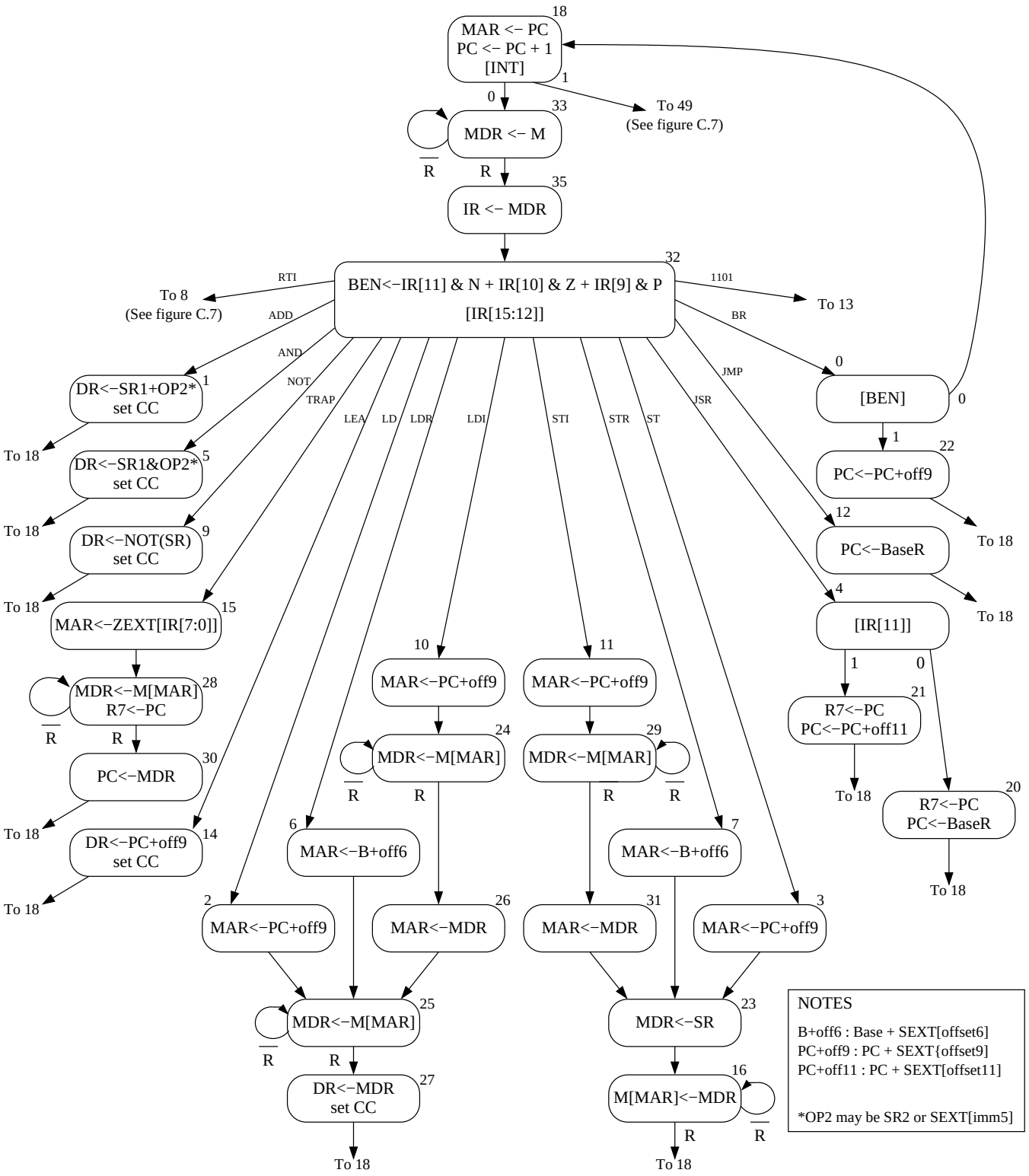
Name:_____

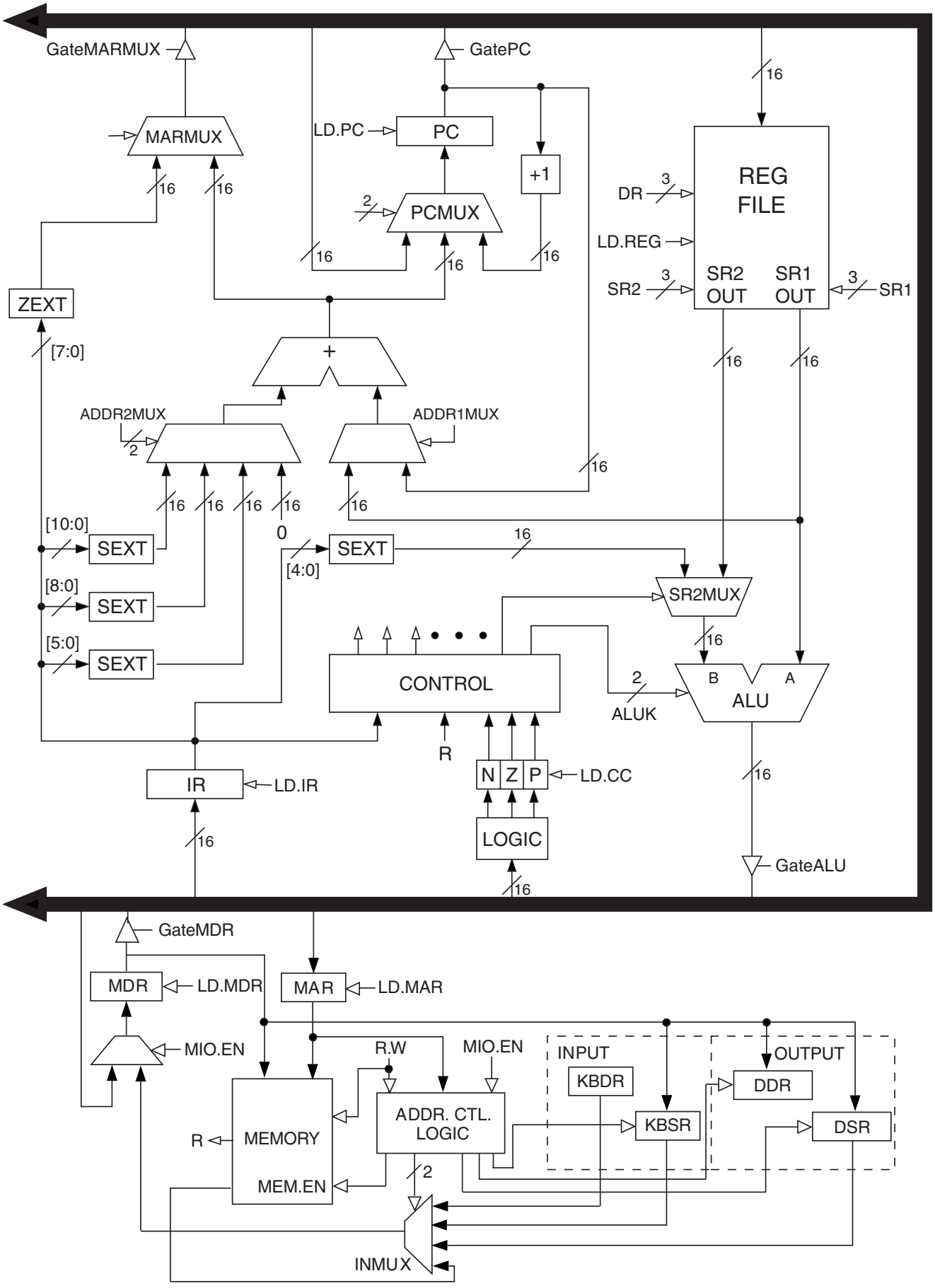
During execution, we examined the computer during each clock cycle, and recorded some information for certain clock cycles, producing the table shown below. The table is ordered by the cycle number in which the information was collected. Note that each memory access takes 5 clock cycles.

Cycle Number	State Number	Control Signals
1	18	LD.MAR: ☐ LD.REG: ☐ GateMDR: ☐ LD.PC: ☐ PCMUX: ☐ GatePC: ☐
☐	0	LD.MAR: ☐ LD.REG: ☐ BEN ☐ LD.PC: ☐ LD.CC: ☐
☐	☐	LD.REG: 1 DR: 000 GateMDR: ☐ GateALU: ☐ GateMARMUX: ☐
57	1	LD.MAR: ☐ ALUK: ☐ GateALU: ☐ LD.REG: ☐ DR: ☐ GatePC: ☐
77	22	ADDR1MUX: ☐ ADDR2MUX: ☐ LD.PC: ☐ LD.MAR ☐ PCMUX: ☐
101	15	

Part a: Fill in the missing instructions in the program, and complete the program by labeling the appropriate instruction AGAIN. Also, fill in the missing information in the table.

Part b: Given values for A and B, what does the program do?


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
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
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
10
MAR<−PC+off9
24
MDR<−M[MAR]
R
R
6
MAR<−B+off6
2
MAR<−PC+off9
26
MAR<−MDR
25
MDR<−M[MAR]
R
R
27
DR<−MDR
set CC
To 18
11
MAR<−PC+off9
29
MDR<−M[MAR]
R
R
7
MAR<−B+off6
31
MAR<−MDR
3
MAR<−PC+off9
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]


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

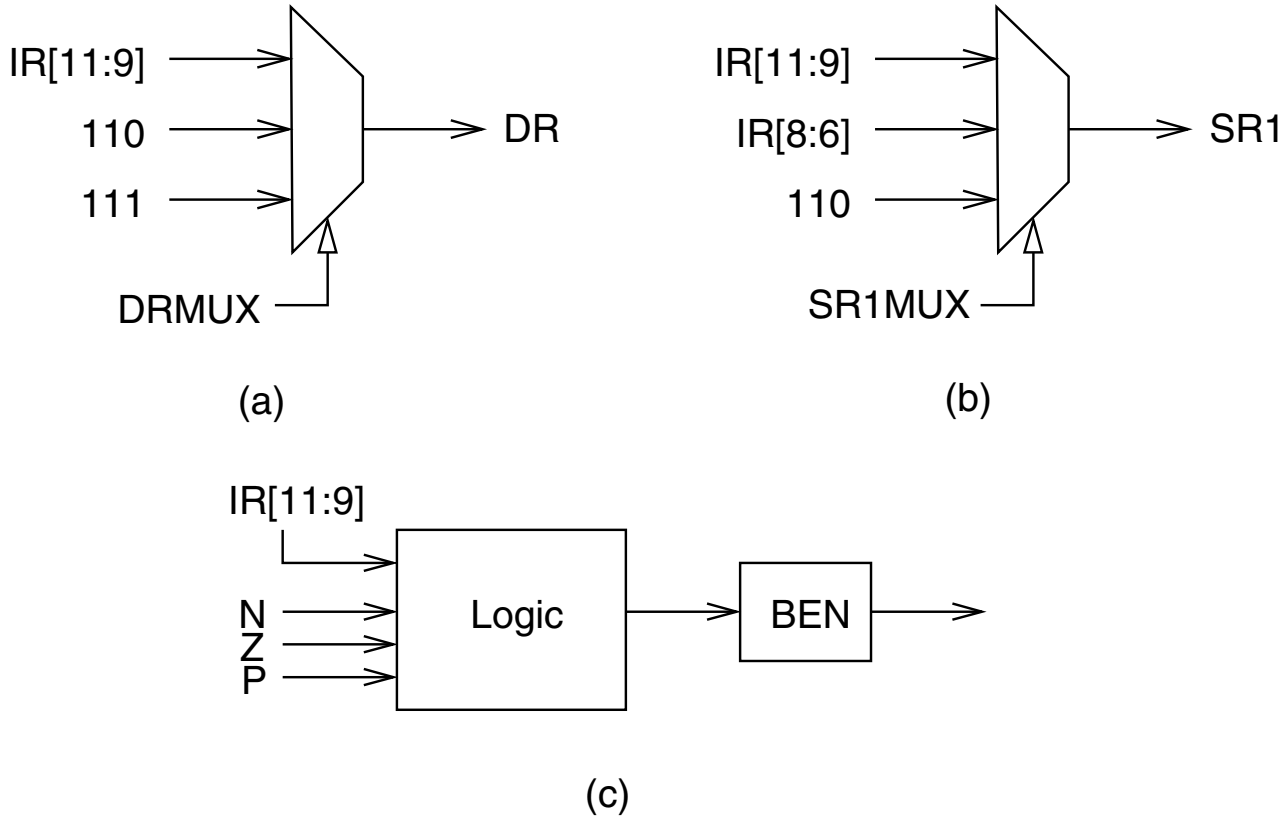

IR[11:9]
110
111
DR
DRMUX
(a)
IR[11:9]
IR[8:6]
110
SR1
SR1MUX
(b)
IR[11:9]
N
Z
P
Logic
BEN
(c)

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

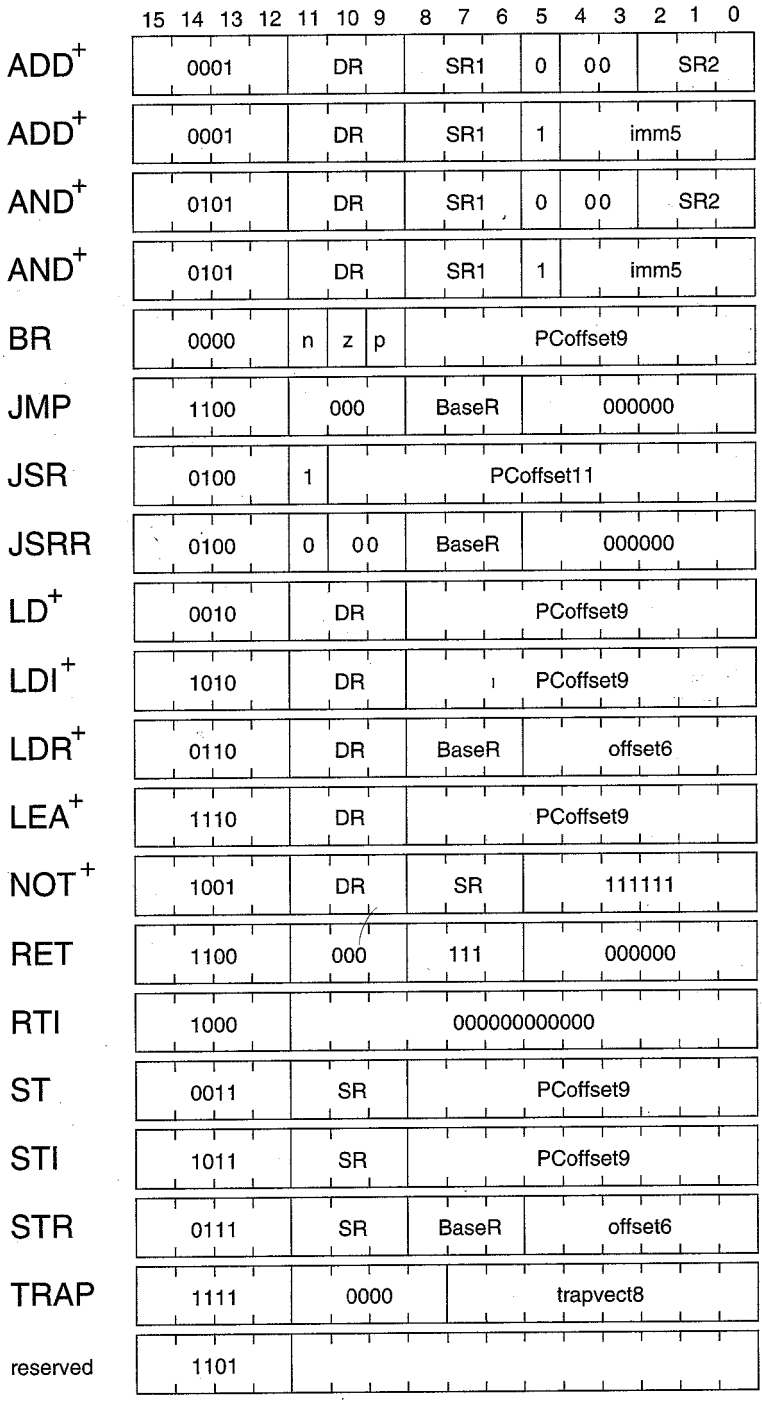


Figure A.2 Format of the entire LC-3 instruction set. **Note:** + indicates instructions that modify condition codes

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.