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

EE 306, Fall 2013
Yale Patt, Instructor
Ben Lin, Mochamad Asri, Ameya Chaudhari, Nikhil Garg, Lauren Guckert,
Jack Koenig, Saijel Mokashi, Sruti Nuthalapathi, Sparsh Singhai, Jiajun Wang
Exam 2, November 13, 2013

Name:_____

Problem 1 (20 points):_____

Problem 2 (20 points):_____

Problem 3 (20 points):_____

Problem 4 (20 points):_____

Problem 5 (20 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):

A student is debugging his program. His program does not have access to memory locations x0000 to x2FFF. Why that is the case we will discuss before the end of the semester. The term is ”privileged memory” but not something for you to worry about today.

He sets a breakpoint at x3050, and then starts executing the program. When the program stops, he examines the contents of several memory locations and registers, then hits single step. The simulator executes one instruction and then stops. He again examines the contents of the memory locations and registers. They are as follows:

	Before	After
PC	x3050	x3051
R0	x2F5F	xFFFF
R1	x4200	x4200
R2	x0123	x0123
R3	x2323	x2323
R4	x0010	x0010
R5	x0000	x0000
R6	x1000	x1000
R7	x0522	x0522
M[x3050]	**x6???**	**x6???**
M[x4200]	x5555	x5555
M[x4201]	xFFFF	xFFFF

Complete the contents of location x3050

x3050 | 0 | 1 | 1 | 0 | | | | | | | | | | | | |

Part b. (5 points):
A student is writing a program and needs to subtract the contents of R1 from the contents of R2, and put the result in R3. Instead of writing:

```
NOT   R3,R1
ADD   R3,R3,#1
ADD   R3,R3,R2
```

he writes:

```
NOT   R3,R1
.FILL x16E1
ADD   R3,R3,R2
```

He assembles the program and attempts to execute it. Does the subtract execute correctly? Why or why not?

Circle one: YES/NO. Explain in not more than fifteen words.

Name:_____

Part c. (5 points):
An assembly language program contains the following subroutine, which the LC-3 assembler stores in memory, starting at location x3070.

Construct the symbol table entries for the subroutine. **Hint: I am asking you to ONLY construct the symbol table entries for this subroutine, and nothing more.**

```
INPUT   ST R7 SAVER7
        LD  R1, MINUS
        LEA R5, BUFFER
        LD R0, LF
        TRAP x21
        LEA R0, PROMPT
        TRAP x22
        LD  R0, LF
        TRAP x21
AGAIN   TRAP x20
        STR R0, R5, #0
        ADD R0,R0,R1
        BRz NEXT
        ADD R5, R5, #1
        BRnzp AGAIN
NEXT    LD R7, SAVER7
        RET
SAVER7  .BLKW 1
MINUS   .FILL xFFDD
BUFFER  .BLKW x21
PROMPT  .STRINGZ "Type a word, then type #"
LF      .FILL x0A
```

Label	Address

Name:_____

Part d. (5 points):
An Aggie (always an Aggie!) modified the service routine that the operating system executes as a result of a user program executing the TRAP x20 instruction. The modification consists of inserting three lines of code into the trap service routine:

1. Adding the following instruction to the beginning of the service routine:

```
        LD  R2,MASK
```

2. Inserting the instruction AND R0,R1,R2 in the place shown in the original service routine:

```
AGAIN   LDI  R1,KBSR
        AND  R0,R1,R2
        BRzp AGAIN
        LDI  R0, KBDR
```

3. Inserting the following pseudo-op immediately after RET:

```
MASK    .FILL x7FFF
```

The complete TRAP service routine after adding the three changes:

```
        ST  R1, SaveR1
        ST  R2, SaveR2
;
        LD  R2,MASK
;
AGAIN   LDI  R1,KBSR
        AND  R0,R1,R2
        BRzp AGAIN
        LDI  R0, KBDR
;
        LD  R1, SaveR1
        LD  R2, SaveR2
;
        RET
MASK     .FILL x7FFF
KBSR    .FILL xFE00
KBDR    .FILL xFE02
SaveR1  .BLKW 1
SaveR2  .BLKW 1
```

Your job: Answer the question: Will the trap service routine still work, and explain why or why not in fifteen words or fewer.

Name:_____

Problem 2. (20 points):
The following program pushes elements onto a stack with JSR PUSH and pops elements off of the stack with JSR POP.

```
                .ORIG X3000
                LEA  R6, STACK_BASE

X               TRAP  x20           ;GETC
                TRAP  x21           ;OUT
                ADD  R1, R0, x-0A   ;x0A is ASCII code for line feed,
                                    ;x-0A is the negative of x0A
                BRz  Y
                JSR  PUSH
                BRnzp  X

Y               LEA  R2, STACK_BASE
                NOT  R2, R2
                ADD  R2, R2, #1
                ADD  R3, R2, R6
                BRz  DONE
                JSR  POP
                TRAP  x21           ;OUT
                BRnzp  Y

DONE            TRAP  x25           ;HALT
STACK           .BLKW  5
STACK_BASE      .FILL  x0FFF

PUSH            ADD  R6, R6, #-1
                STR  R0, R6, #0
                RET

POP             LDR  R0, R6, #0
                ADD  R6, R6, #1
                RET

                .END
```

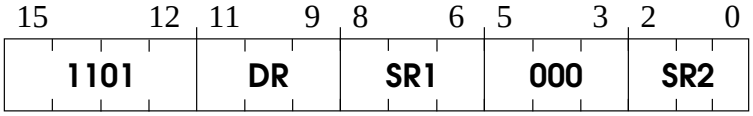
What will appear on the screen if a user, sitting at a keyboard, typed the three keys a, b, c, followed by the <Enter> key?

What will happen if a user, sitting at a keyboard, typed the eight keys a, b, c, d, e, f, g, h, followed by the <Enter> key? (Please, no more than fifteen words in the box.)

Name:_____

Problem 3. (20 points):

An aggressive young engineer decides to build and sell the LC-3, but is told that if he wants to succeed, he really needs a SUBTRACT instruction. Given the unused opcode 1101, he decides to specify the SUBTRACT instruction as follows:



The instruction is defined as: DR ← SR2 - SR1, and the condition codes are set.

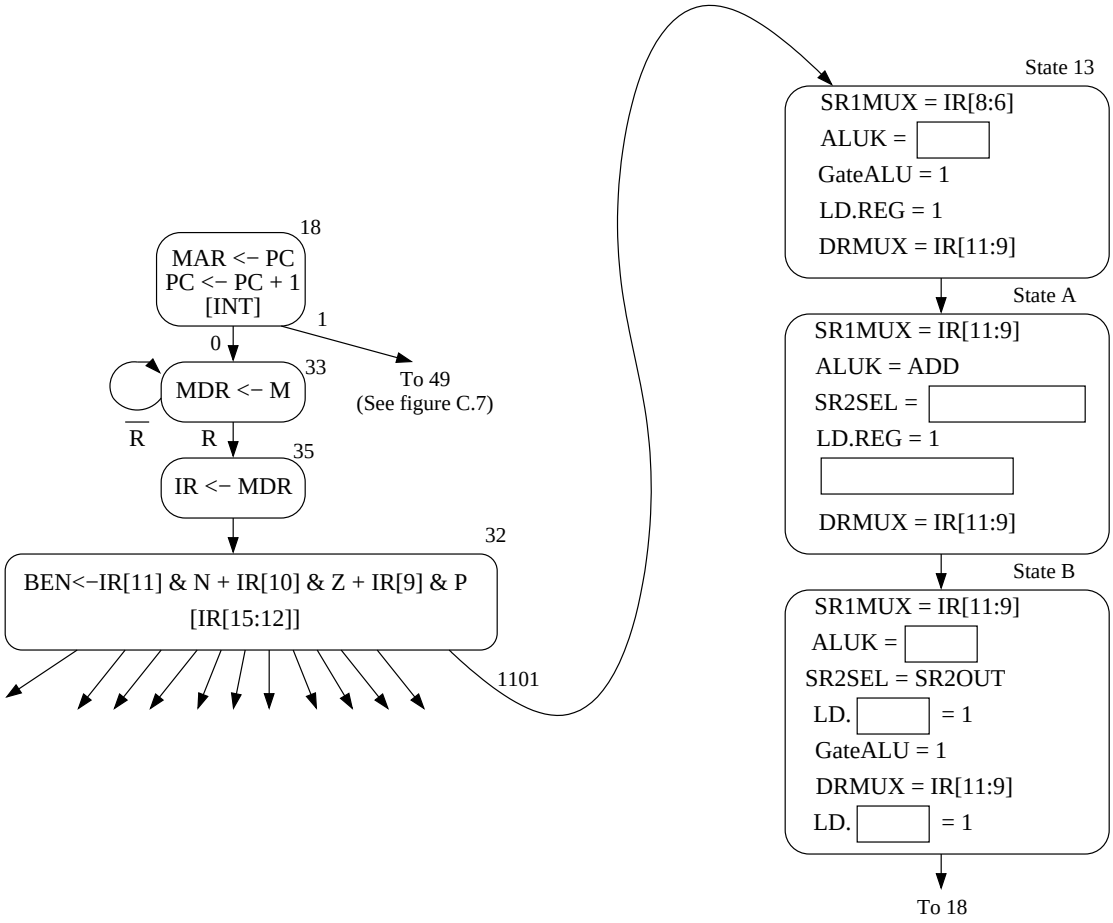
To accomplish this, the engineer needs to add three states to the state machine and a mux and register A to the data path. The modified state machine is shown below and the modified data path is shown on the next page. The mux is controlled by a new control signal SR2SEL which selects one of its two sources.

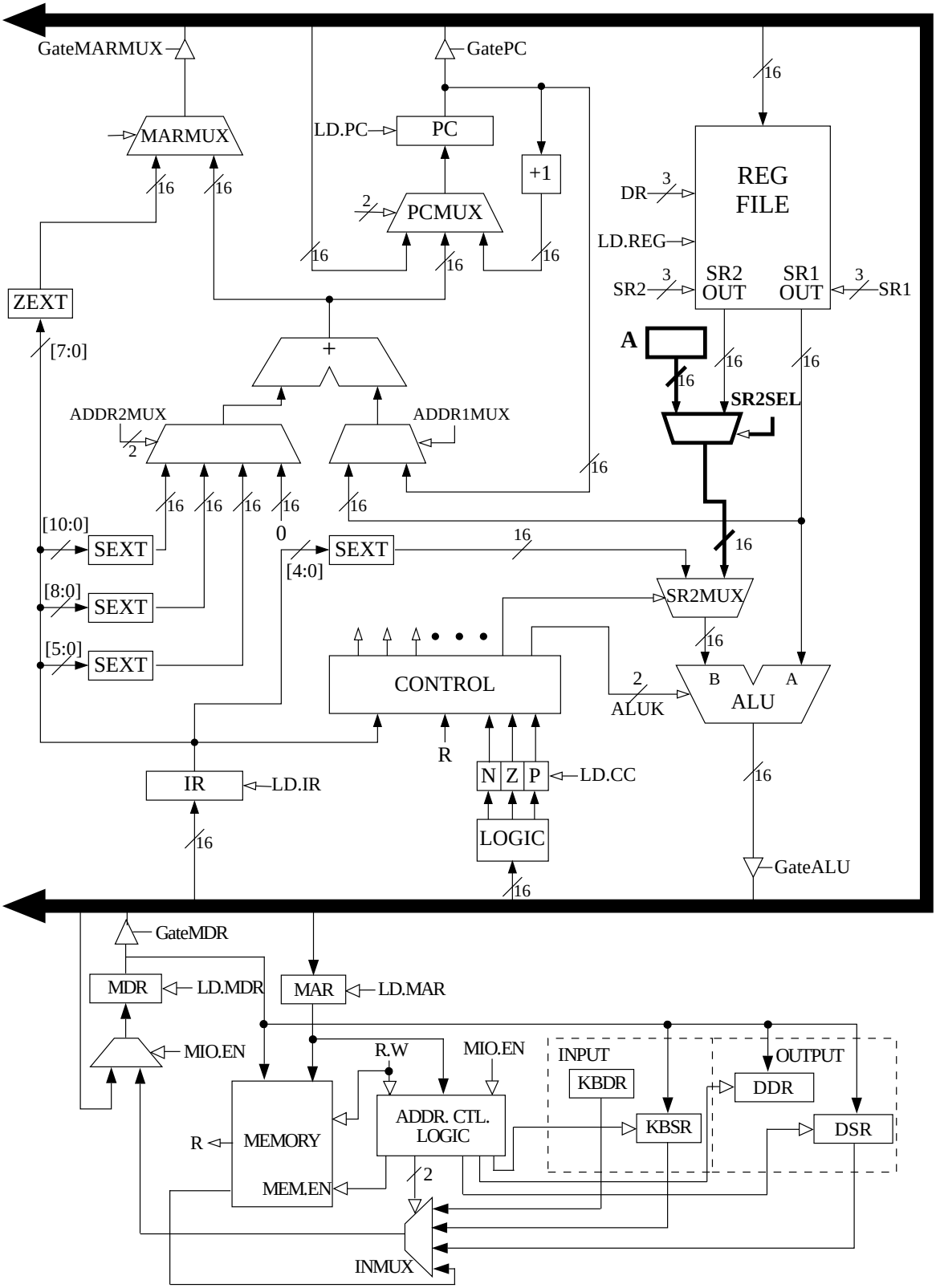
SR2SEL/1: SR2OUT, REGISTER_A

Your job:

For the state machine shown below, fill in the empty boxes with the control signals that are needed in order to implement the SUBTRACT instruction.

For the data path, fill in the value in register A.



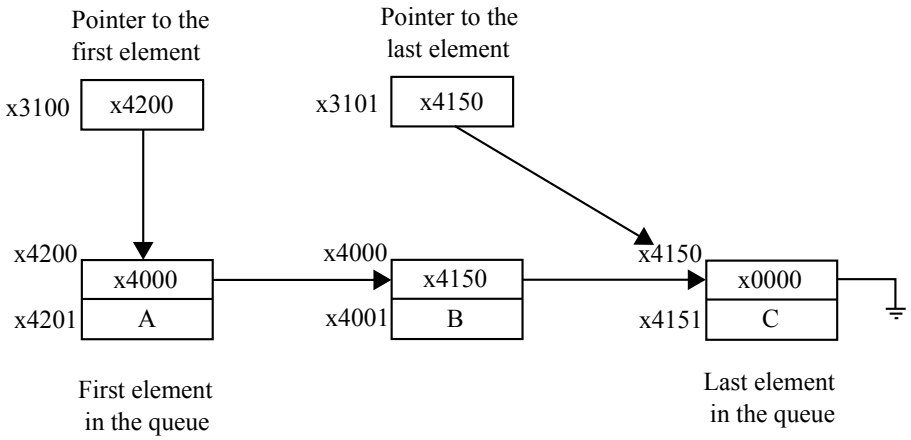

GateMARMUX
GatePC
16
MARMUX
LD.PC
PC
+1
REG
FILE
DR
3
16
16
2
PCMUX
16
16
16
LD.REG
SR2
3
SR2
OUT
SR1
OUT
3
SR1
ZEXT
[7:0]
+
A
16
16
16
SR2SEL
ADDR2MUX
2
ADDR1MUX
16
16
16
16
16
16
16
[10:0]
SEXT
0
[4:0]
SEXT
16
16
SR2MUX
[8:0]
SEXT
16
[5:0]
SEXT
CONTROL
2
ALUK
B
A
ALU
R
N
Z
P
LD.CC
IR
LD.IR
16
LOGIC
16
16
GateALU
GateMDR
MDR
LD.MDR
MAR
LD.MAR
MIO.EN
R.W
MIO.EN
INPUT
OUTPUT
KBDR
DDR
R
MEMORY
MEM.EN
ADDR. CTL.
LOGIC
KBSR
DSR
2
INMUX

Name:_____

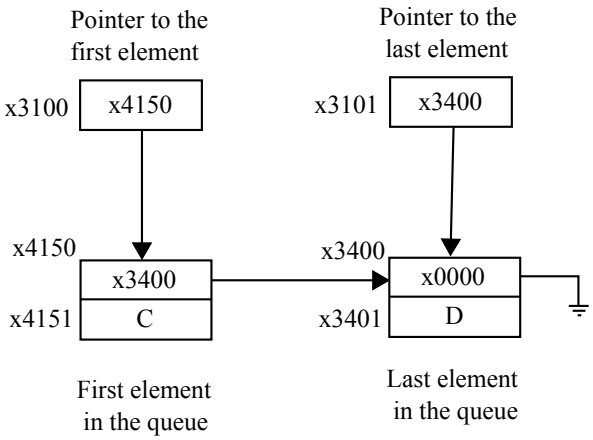
Problem 4. (20 points):
In class we talked about stacks, which are LIFO (Last In, First Out) mechanisms that allow us to push (insert) and pop (remove) elements from the top. Another data structure is a queue. It operates as a FIFO (First In, First Out) mechanism. Elements are removed from the front and inserted in the rear, much like the way a queue works in our daily lives.

Our queue is implemented as a linked list. Each element consists of two words: a pointer to the next element that entered the queue and a value. We need two pointers, one to the front of the queue which we use to remove elements, and one to the last element of the queue which we use to add another element. The last element points to NULL (x0000).

The figure below shows a queue with three elements, the first is A, the second is B, the last is C. M[x3100] contains the address of the front of the queue. M[x3101] contains the address of the last element of the queue.



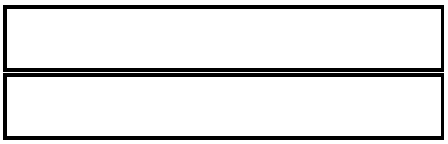
If we add an element D and we remove the elements A and B, the queue looks like this:



Name:_____

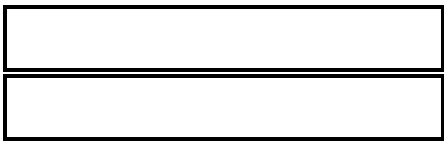
Your job: Complete the subroutines to dequeue (remove) the front element of the queue and enqueue (insert) a new element to the back of the queue. After the DEQUEUE subroutine is executed, R3 should contain the address of the element that was just dequeued; before the ENQUEUE subroutine is executed, R3 should contain the address of the new element to be enqueued. You do NOT have to worry about the case when the queue is(or becomes) empty; that is, you can assume the queue will always have at least one element before and after any operation.

```
DEQUEUE     ST  R0, A
            LDI R3, B
```



```
            LD  R0, A
            RET
A           .BLKW  1
B           .FILL  x3100
```

```
ENQUEUE     ST  R0, C
            LDI  R0, D
```



```
            LD  R0, C
            RET
C           .BLKW  1
D           .FILL  x3101
```

Name:_____

Problem 5. (20 points):

During the execution of an LC-3 program, an instruction in the program starts executing at clock cycle T and requires 15 cycles to complete.

The table below lists **ALL** five clock cycles during the processing of this instruction which require use of the bus. The table shows for each of those clock cycles: which clock cycle, the state of the state machine, the value on the bus, and the important control signals that are active during that clock cycle.

Note: In class on Monday, I gave an example where it took 18 clock cycles for memory to read or write. Part (d) of this problem asks you how many clock cycles it takes for memory to read or write in this example.

Cycle	State	Bus	Important Control Signals For This Cycle
T	18	x3010	LD.MAR = 1, LD.PC =1, PCMux = PC + 1, GatePC = 1
T + 4			
T + 6		x3013	
T + 10		x4567	
T + 14		x0000	LD.REG = 1, LD.CC = 1, GateMDR = 1, DR = 001

a. Fill in the missing entries in the table.

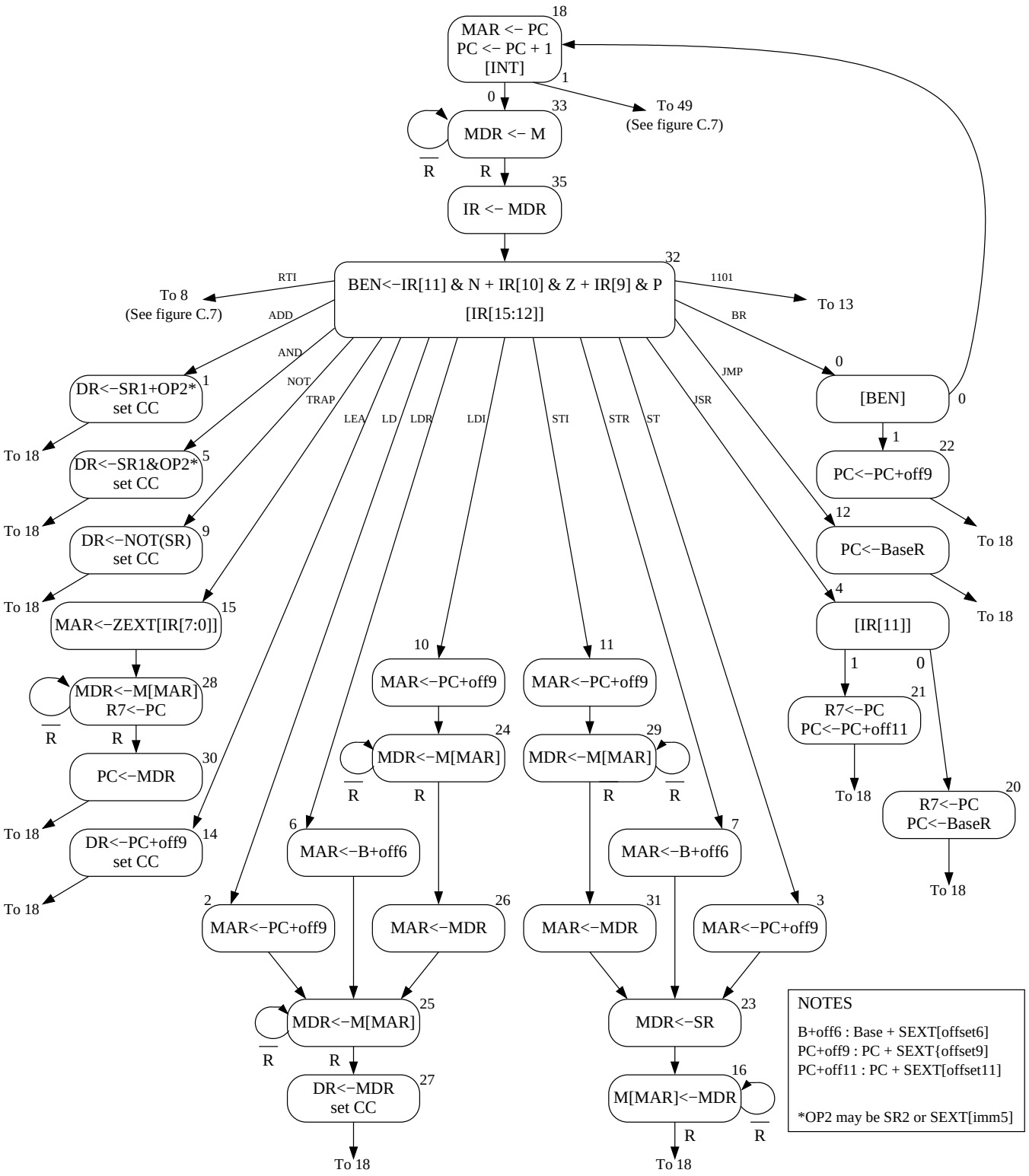
b. What is the instruction being processed?

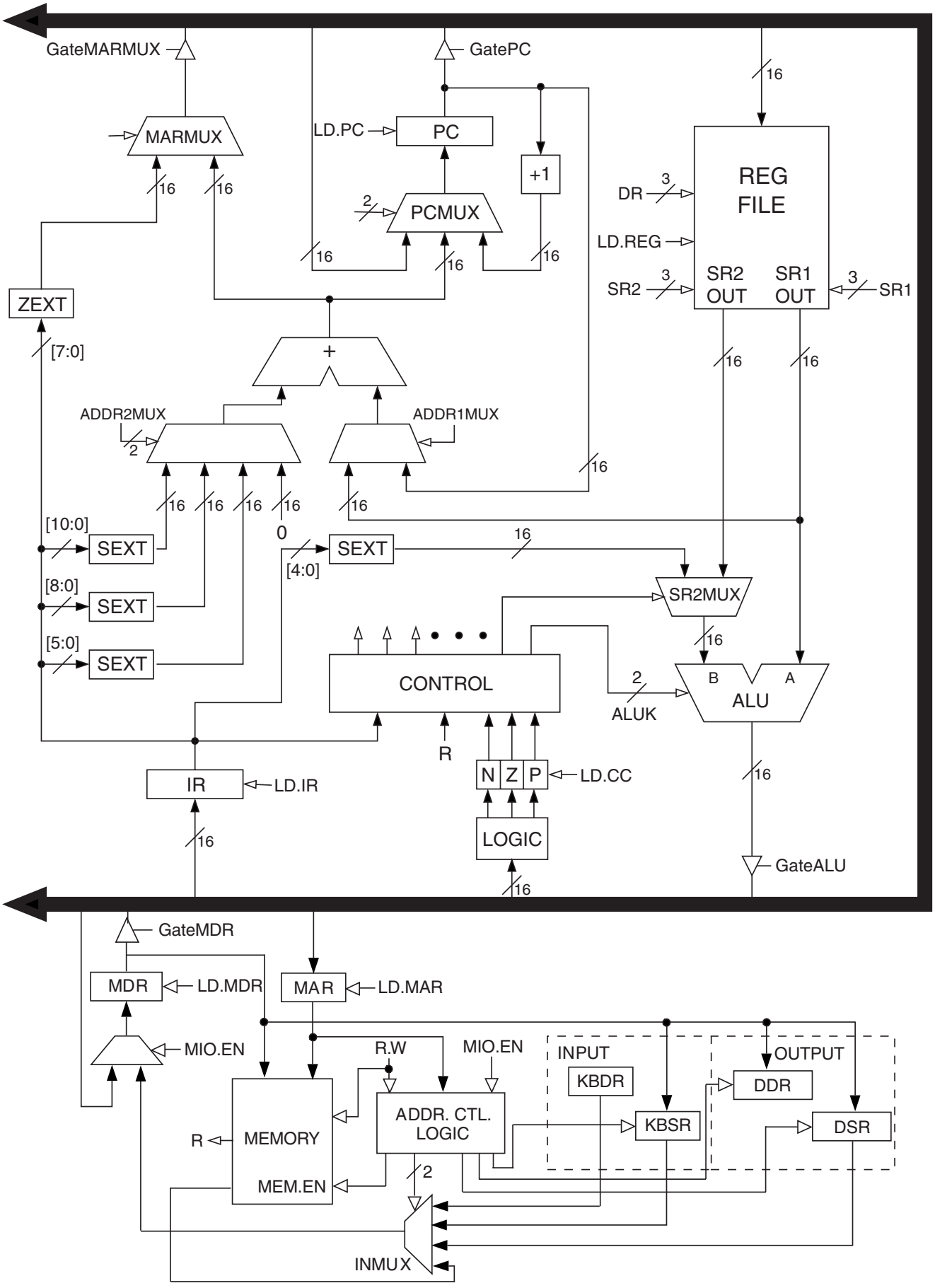
c. Where in memory is that instruction?

d. How many clock cycles does it take memory to read or write?

e. There is enough information above for you to know the contents of three memory locations. What are they and what are their contents?

Memory address	Contents


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
3
SR2
OUT
SR1
OUT
3
SR1
ZEXT
[7:0]
+
16
16
ADDR2MUX
ADDR1MUX
2
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

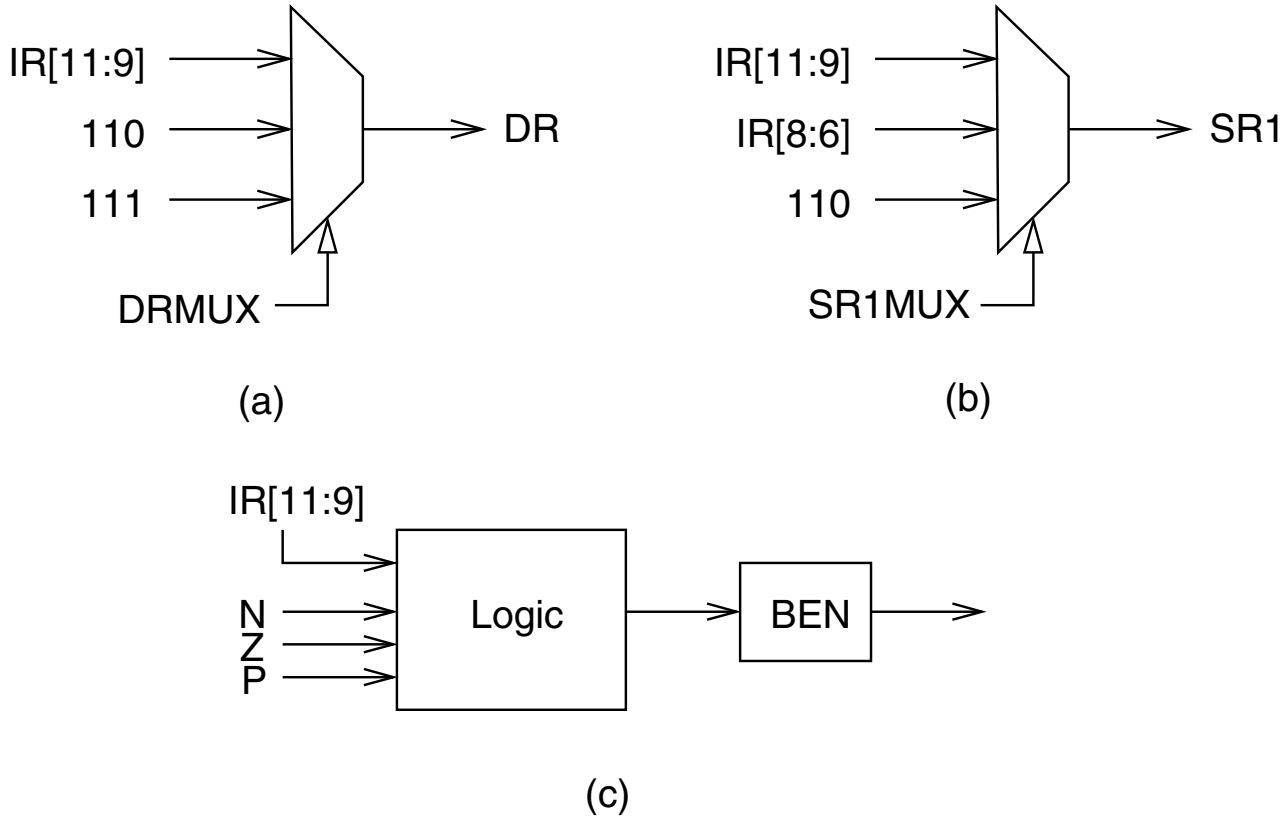

IR[11:9]
110
111
DR
DRMUX
(a)
IR[11:9]
IR[8:6]
110
SR1
SR1MUX
(b)
IR[11:9]
N
Z
P
Logic
BEN
(c)

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

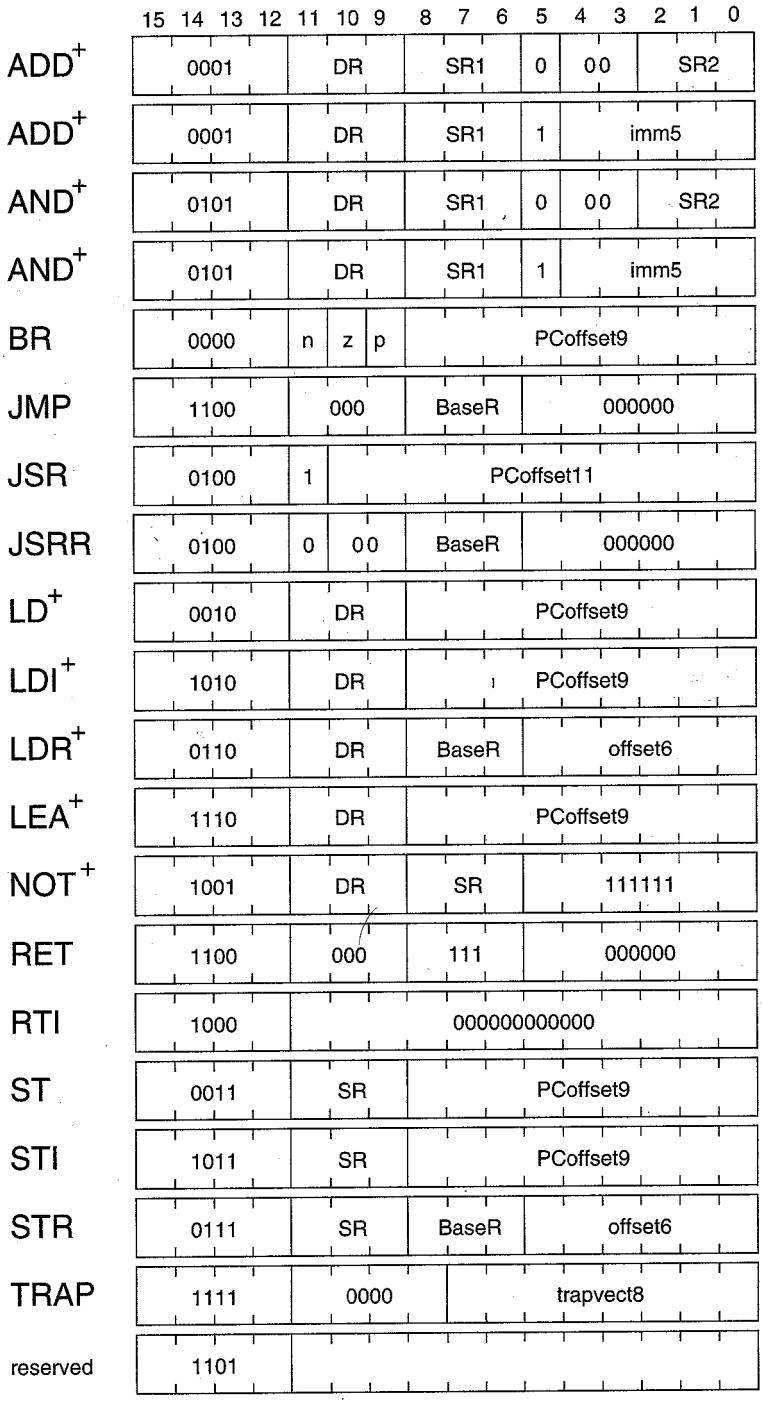


Figure A.2 Format of the entire LC-3 instruction set. **Note:** + indicates instructions that modify condition codes

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.