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

EE 306, Fall 2009
Yale Patt, Instructor
Aater Suleman, Chang Joo Lee, Ameya Chaudhari, Antonius Keddis, Arvind Chandrababu, Bhargavi Narayanasetty, Eshar Ben-dor, Faruk Guvenilir, Marc Kellermann, RJ Harden, TAs
Final Exam, December 15, 2009

Name: Solution

Part A:

Problem 1 (10 points): 10

Problem 2 (10 points): 10

Problem 3 (10 points): 10

Problem 4 (10 points): 10

Problem 5 (10 points): 10

Part A (50 points): 50

Part B:

Problem 6 (20 points): 20

Problem 7 (20 points): 20

Problem 8 (20 points): 20

Problem 9 (20 points): 20

Total (130 points): 130

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.

Solution
Signature

GOOD LUCK!
(HAVE A GREAT SEMESTER BREAK)

Name: Solution

Problem 1. (10 points): The following program is written in LC-3 Assembly Language. To generate the binary, the LC-3 Assembler must first create a symbol table corresponding to the program.

Your job: Create the symbol table. Use as many entries in the table as you need.

```
          .ORIG x3000
MAIN    0 LEA   R0, S1
        1 LEA   R1, BUF
        2 LD    R2, NEGo
AGAIN   3 LDR   R3, R0, #0
        4 ADD   R4, R3, R2
        5 BRnp  SAVE
        6 LD    R3, ZERO
SAVE    7 STR   R3, R1, #0
        8 ADD   R0, R0, #1
        9 ADD   R1, R1, #1
        A ADD   R3, R3, #0
        B BRnp  AGAIN
        C HALT
NEGo    D .FILL x-6F          x300E + x30 → x303E
BUF     E .BLKW x30
S1     3E .STRINGZ ``Good luck''
ZERO   48 .FILL x30
          .END
```

Symbol	Address
MAIN	x3000
AGAIN	x3003
SAVE	x3007
NEGo	x300D
BUF	x300E
S1	x303E
ZERO	x3048

Name: Solution

Problem 2. (10 points): Recall programming lab 5 where you wrote a keyboard interrupt service routine which displayed a typed character ten times on the screen, followed by a line feed. One student who is preoccupied with other things submitted the following as his keyboard interrupt service routine. Five instructions in his code are incorrect.

Your job: For each of the incorrect instructions, enter the correct instruction in the table on the right in the same row as the corresponding incorrect instruction. For example, **STR R0, SaveR0** should be **ST R0, SaveR0**, as shown.

Complete the table on the right by adding ONLY the four correct instructions that correct the four remaining bugs. Please do not copy any instruction into the table on the right if they are already correct.

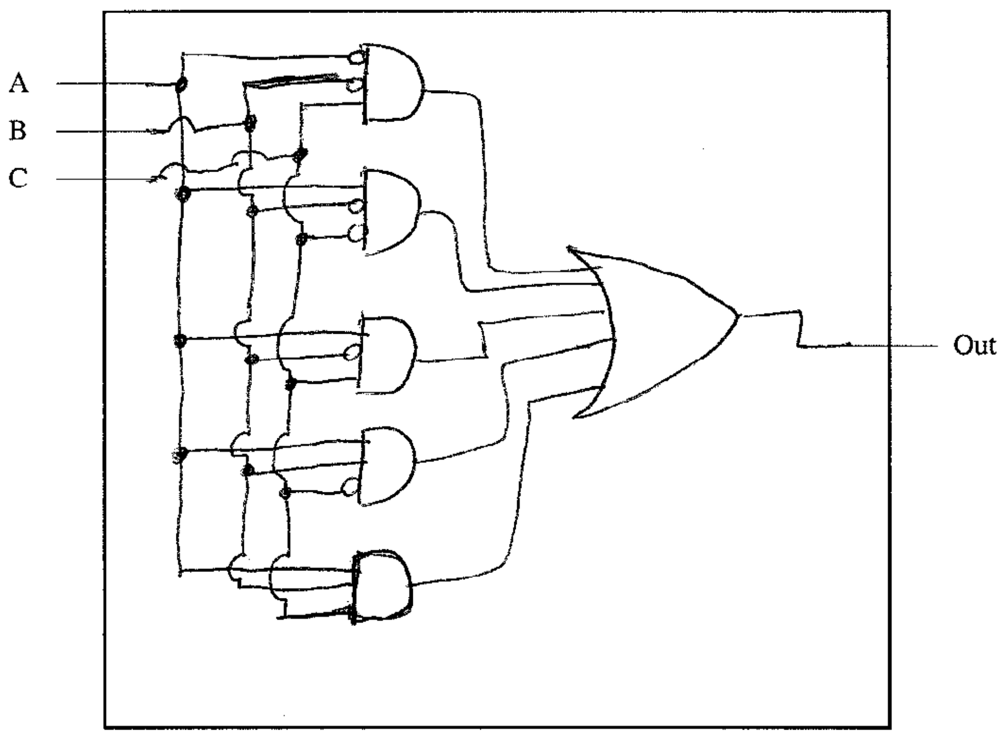
Incorrect code			Corrected instructions (ONLY)
	.ORIG	x2000	
	STR	R0, SaveR0	ST R0, SaveR0
	ST	R1, SaveR1	
	ST	R2, SaveR2	
	LD	R0, KBDR	LDI R0,KBDR
	AND	R2, R2, #0	
	ADD	R2, R2, #10	
DSP_RDY	LDI	R1, DSR	
	BRn	DSP_RDY	BRzp DSP_RDY
	STI	R0, DDR	
	ADD	R2, R2, #-1	
	BRp	DSP_RDY	
ENTER	LD	R0, LF	
DSP_RDY1	LDI	R1, DSR	
	BRn	DSP_RDY1	BRzp DSP_RDY1
	STI	R0, DDR	
	LD	R0, SaveR0	
	LD	R1, SaveR1	
	LD	R2, SaveR2	
	RET		RTI
SaveR0	.BLKW	#1	
SaveR1	.BLKW	#1	
SaveR2	.BLKW	#1	
KBDR	.FILL	xFE02	
DSR	.FILL	xFE04	
DDR	.FILL	xFE06	
LF	.FILL	x000A	
	.END		

Name: Solution

Problem 3. (10 points): Design a digital logic circuit that implements the following truth table.

A	B	C	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Please draw the logic circuit inside the box below. Connect the inputs and the output of your circuit to the wires labeled A, B, C, and Out. You can use **only** AND, OR, and NOT gates. You can use as many of them as you need.



Name: Solution

Problem 4. (10 points): One algorithm for dividing a positive (non-zero) **even** number by 2 is to load the even number into one register, load a second register with 0, and then continually decrement the first and increment the second, until you have the same value in both registers. That value is your original even number divided by 2.

Example: Take the value 10: (10,0) → (9,1) → (8,2) → (7,3) → (6,4) → (5,5). Hooray!

The subroutine shown below, with the two missing instructions, performs this algorithm.

Your job: Insert the two missing instructions.

```
        LDI   R0, INPUT
        AND   R1, R1, #0
AGAIN   ADD   R0, R0, #-1
        ADD   R1, R1, #1
        NOT   R2, R0
        ADD R2,R2,#1
        ADD   R2, R2, R1
        BRnp AGAIN
        STI   R0, OUTPUT
        RET
INPUT   .FILL x3100
OUTPUT  .FILL x3101
```

Name: Solution

Problem 5. (10 points): This problem tests your knowledge of the instruction cycle for processing the NOT instruction. You are asked to show the values of several control signals in every clock cycle of the sequence that is used to process the NOT instruction.

The instruction cycle starts with state 18 as shown in the table below.

Your job: Identify each state in the sequence, and show the values of the control signals listed during each state in the sequence. Use the convention specified below. For a particular state, if the value of a control signal does not matter, fill it with an X. You may not have to use all the rows.

Note: Assume a memory access takes one clock cycle.

Cycle	State	LD.PC	LD.MAR	LD.MDR	LD.REG	LD.CC	GateALU	GatePC	ALUK	PCMUX
1	18	1	1	0	0	0	0	1	X	00
2	33	0	0	1	0	0	0	0	X	X
3	35	0	0	0	0	0	0	0	X	X
4	32	0	0	0	0	0	0	0	X	X
5	9	0	0	0	1	1	1	0	10	X
6										
7										
8										
9										
10										

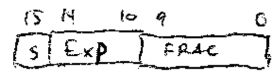
```
LD.PC     0: load not enabled
          1: load enabled

LD.MAR    0: load not enabled
          1: load enabled

LD.MDR    0: load not enabled
          1: load enabled

LD.REG    0: load not enabled
          1: load enabled

LD.CC     0: load not enabled
          1: load enabled

GateALU   0: do not pass signal
          1: pass signal

GatePC    0: do not pass signal
          1: pass signal

ALUK      00: ADD
          01: AND
          10: NOT
          11: Pass input A

PCMUX     00: PC+1
          01: BUS
          10: from adder
```

Name: Solution

Problem 6. (20 points): In the spirit of the IEEE Floating Point standard, we have specified a 16-bit floating point data type. Bit[15] is the sign, bits[14:10] contains an excess-15 code for the exponent, bits[9:0] contains the fraction.

The subroutine shown below tests the floating point value contained in R0 and returns 0 in R5 if it **is** an integer, and returns 1 in R5 if it **is not** an integer. Three instructions in the subroutine have been omitted. Your job: insert the missing three instructions.

Note: This subroutine calls another subroutine `RightShift10` (not shown) which right shifts the contents of R1 by 10 bits, and returns the result in R1.

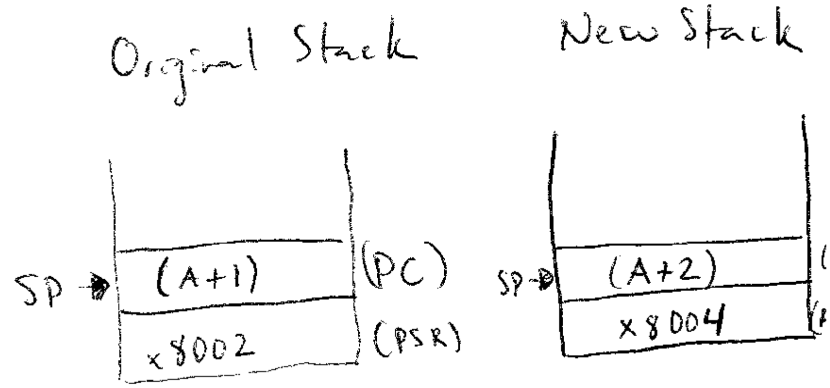



```
CHECK   AND R5, R5, #0

; left shift the floating point number 6 bits, moving fraction bits into R2[15:6]
        ADD R2, R0, #0
        AND R3, R3, #0
        ADD R3, R3, #6
LOOP1   BRz EXP
        ADD R2, R2, R2
        ADD R3, R3, #-1
        BRnzp LOOP1

; move exponent into R1[4:0]
EXP     LD  R1, MASK1
        AND R1, R1, R0
        JSR RightShift10
        ADD R1, R1, #-15

; determine if floating point number is an integer

        BRzp LOOP2

        ADD R5, R5, #1
        BRnzp END

LOOP2   BRz NEXT

        ADD R2, R2, R2

        ADD R1, R1, #-1
        BRnzp LOOP2

; report the result

NEXT    ADD R2, R2, #0

        BRz END

        ADD R5, R5, #1

END     RET
MASK1   .FILL   x7C00
```

Name: Solution

Problem 7. (20 points): The interrupt service routine shown below is loaded into the LC-3 memory, and then the user program shown below is loaded into the LC-3 memory. Then, the run button is pressed.

Interrupt service routine




```
        .ORIG    x2000
        ST        R0, SaveR0
        ST        R1, SaveR1
        LDR       R0, R6, #0
        ADD       R0, R0, #1
        STR       R0, R6, #0
        LDR       R0, R6, #1
        LD        R1, Mask
        AND       R0, R1, R0
        ADD       R0, R0, #4
        STR       R0, R6, #1
        LD        R0, SaveR0
        LD        R1, SaveR1
        RTI
SaveR0  .BLKW     1
SaveR1  .BLKW     1
Mask    .FILL     xFFF8
```

User Program

```
        .ORIG  x4000
         ... ;initialize keyboard interrupt handler as x2000

        LEA    R0, S1
AGAIN   TRAP   x22
A       AND    R2, R2, #0  ;  <=================== Interrupt during this instruction
        BRz    SKIP
        BRz    SET
        ADD    R2, R2, #-5
SET     ADD    R2, R2, #10
AGAIN2  BRnz   DONE
        LEA    R0, S2
        TRAP   x22
        ADD    R2, R2, #-1
        BRnzp  AGAIN2
SKIP    BRnzp  AGAIN
DONE    HALT

S1      .STRINGZ "UT "
S2      .STRINGZ "Rules "
        .END
```

In the absence of any keyboard input, what does the User Program do (in no more than 10 words)?

Prints "UT UT -----" forever.

At some point during the execution of the User Program, a key on the keyboard is pressed causing an interrupt. This happens while the LC-3 is executing the instruction at location A.

What does this program do **after** the key is pressed (in no more than 10 words)?

Prints "UT Rules Rules Rules Rules Rules".

Name: Solution

Problem 8. (20 points): The table shows the contents of all the relevant (and some irrelevant) registers at the completion of three SUCCESSIVE instructions (I1, I2, and I3) of a program. Complete the table by filling in the missing entries. Ignore entries that contain dashes.

Notes:
1. None of the three instructions is an LD, LDR, LDI, or LEA.
2. All interrupts are disabled during the execution.
3. R7 is not modified by any instruction in the program except I3.

	I1	I2	I3
PC	————	x5110	x1000
MAR	————	x510F	x0022
MDR	————	x1B7E	x1000
R0	x389A	x389A	x389A
R1	x01B1	x01B1	x01B1
R2	x1234	x1234	x1234
R3	x2222	x2222	x2222
R4	x2345	x2345	x2345
R5	x0000	xFFFE	xFFFE
R6	x2678	x2678	x2678
R7	x4764	x4764	x5111
N	0	1	1
Z	1	0	0
P	0	0	0

Identify instructions I2 and I3.

I2: ADD R5, R5, #-2

1 | B | 7 | E
0001 | 101 | 101 | 1 | 11110

I3: TRAP x22

Name: Solution

Problem 9. (20 points): A program running in privilege mode (PSR[15]=0) suddenly stops due to a breakpoint set at location x2000. The operator immediately pushes the run button.

The table lists in order the next nine memory accesses (MAR and MDR of each).

Your job: Complete the missing entries in the table.

Note: Do not make any assumptions about the values stored in registers or memory locations except what can be deduced from the trace.

MAR	MDR	
x2000	x8000	RTI
x2C02	x1050	(PC)
x2C03	x0004	(PSR)
x1050	xBCAE	1011 | 110 | 0 1010 1110 → STI R6, #AE
x10FF	x2800	
x2800	x2C04	← R6
x1051	x1 - - - x1DA6	ADD R6, R6, #6 → 0001 | 110 | 110 | 1 0 0110 (x1 | D | A | 6)
x1052	x3C4D	0011 | 110 | 0 0100 1101 → ST R6, x4D
x10A0	x2C0A	← R6

x1050 + xAE = 10FF

x1053 + x4D = 10A0

	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

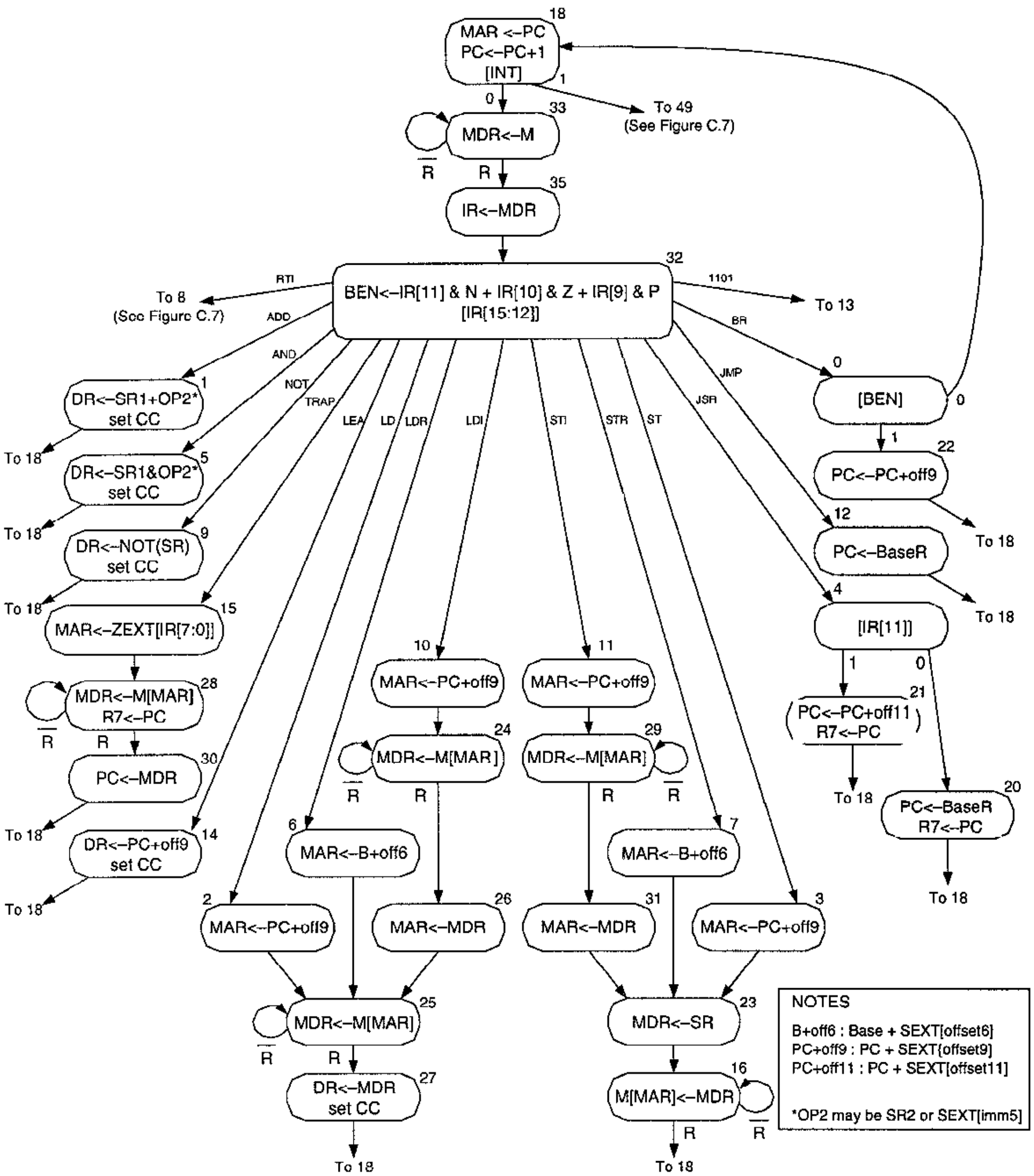


Figure C.2 A state machine for the LC-3

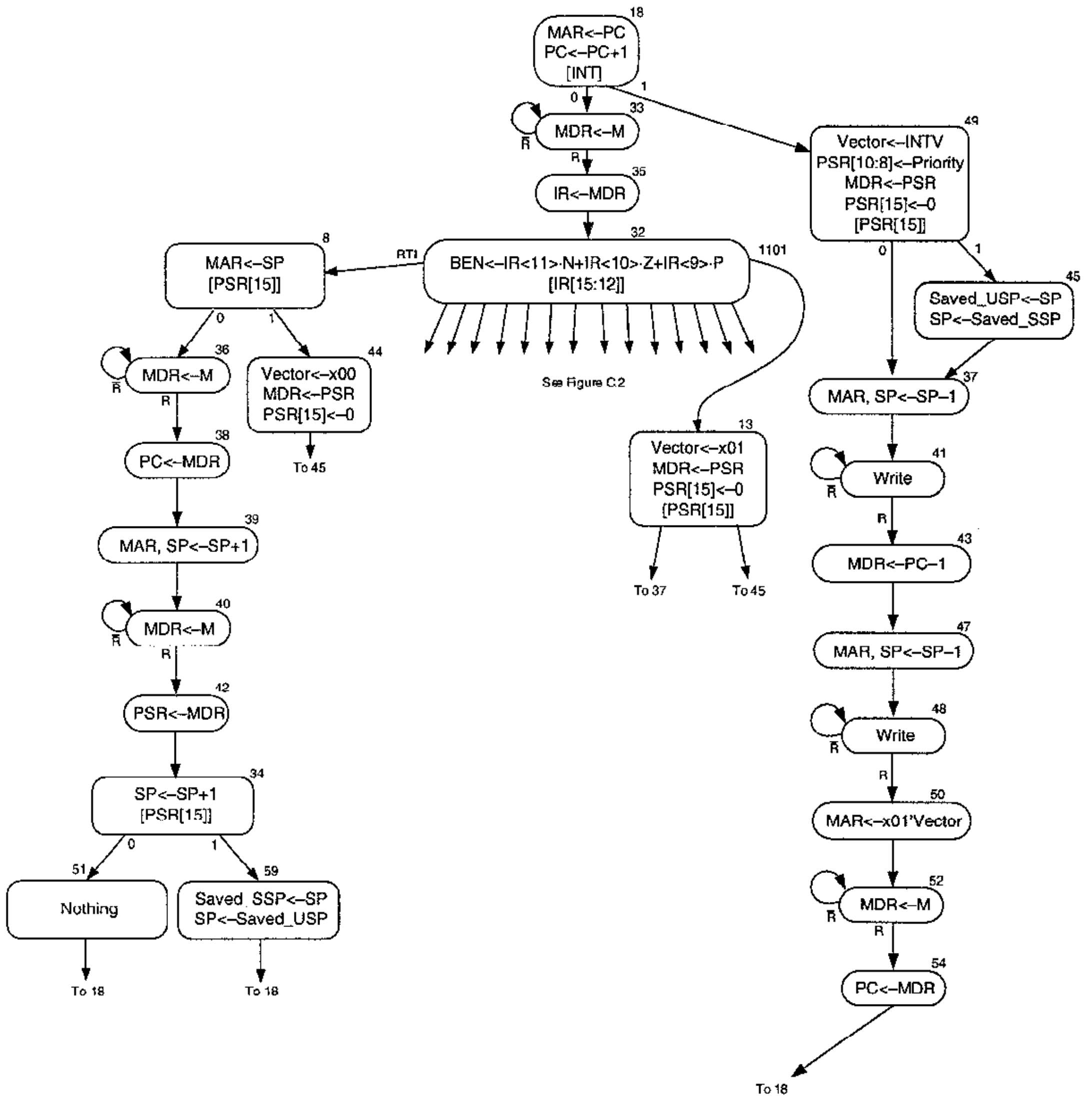


Figure C.7 LC-3 state machine showing interrupt control

of one of the two exceptions specified by the ISA. The two exceptions are a privilege mode violation and an illegal opcode. Figure C.7 shows the state machine that carries these out. Figure C.8 shows the data path, after adding the additional structures to Figure C.3 that are needed to make interrupt and exception processing work.

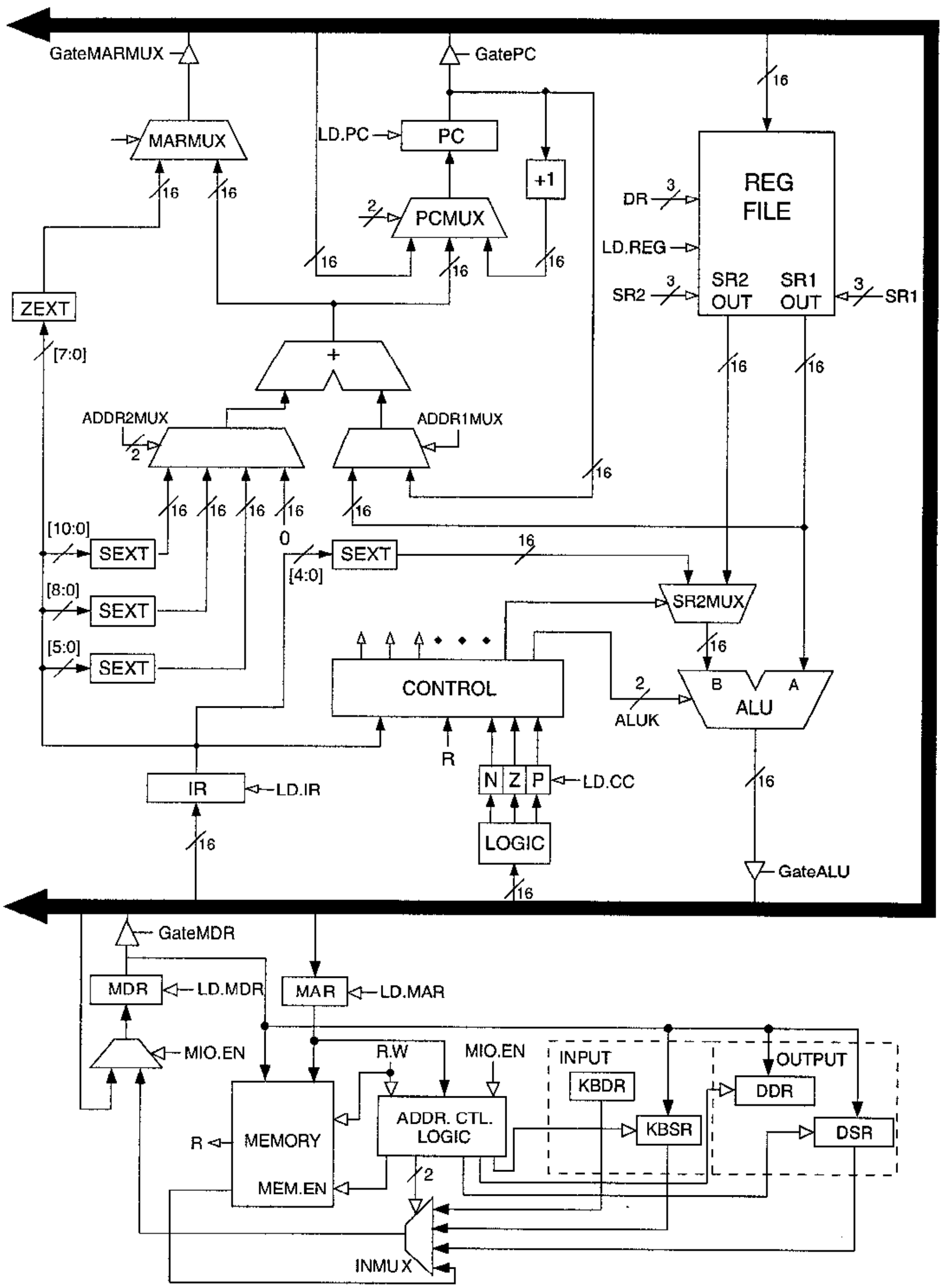


Figure C.3 The LC-3 data path

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.

A.4 Interrupt and Exception Processing

Events external to the program that is running can interrupt the processor. A common example of an external event is interrupt-driven I/O. It is also the case that the processor can be interrupted by exceptional events that occur while the program is running that are caused by the program itself. An example of such an "internal" event is the presence of an unused opcode in the computer program that is running.

Associated with each event that can interrupt the processor is an 8-bit vector that provides an entry point into a 256-entry *interrupt vector table.* The starting address of the interrupt vector table is x0100. That is, the interrupt vector table

E.2 Standard ASCII codes

Table E.2 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