88

Cal State University, San Bernardino
Midterm CSE 313– Machine Organization
Instructor: Taline Georgiou

Spring 2012
May 16, 2012

Closed books/notes. No calculators, cell phones, laptops, or any electronic device is allowed.. Only one sheet of paper with notes allowed. Must show all work for credit. Clearly indicate your answer. No cooperation. 4 problems. Total number of points is 100.

Name: James Small

25 pts 1. No show of work necessary for this problem. on back of page 1

-2

(a) How many memory locations are in LC-3?
(b) Give the decimal number 118 as a number in base 8.
(c) What is the width (in bits) of MAR and MDR in LC-3?
(d) What is the LC-3 assembly language instruction for this machine code: 0101|111|001|1|00011
(e) What is the opcode for LDI in LC-3?
(f) True or False. In a Von Neumann machine data and instructions both reside in memory.
(g) True or False. Control instructions affect the order in which instructions are executed.
(h) True or False. The ISA specifies items including the opcodes and addressing modes.
(i) True or False. MAR (Memory Address Register) holds the number of addresses in memory.
(j) True or False. Immediate instructions have one of their operands in the instruction itself.
(k) True or False. The FETCH phase is common for all instructions.
(l) True or False. The opcode is used in the DECODE phase.

25 pts 2. Suppose that register R1 holds a negative number in twos complement format. Write an assembly language program that multiplies the value in R1 by -2, and the result is placed in R2. (Hint: negation can be done by taking the twos complement and multiplication can be done by addition.)

```
      And R2, R2, #0   ; set R2 to 0
      not R1, R1
      ADD R1, R1, #1   ; convert negative number to positive in R1
      And R3, R3, #-2  ; puts negaitive 2 in R3
      not R3, R3
      Add R3, R3, #1   ; convert -2 to positive in R3

Loop  Add R3, R3, #0   ; set Branch codes on R3, which will be counter
      Brz Done         ; branch if zero
      Add R2, R2, R1
      Add R3, R3, #-1
      Br Loop
Done                   ; Once code gets here, answer is in R2
```

-10 X

Please go on to the next page . . .

25 pts 3. Read the following LC3 code and fill the blanks.

```
        .ORIG .x3200
        LD R0, num1        x3200    ; R0 = x_3301_     Addressing Mode
        LDI R1, num2       x3201    ; R1 = x_5390_     PC Relative (direct)
        LEA R2, num0       x3202    ; R2 = x_3207_     Indirect
        LDR R3, R2, #0     x3203    ; R3 = x_3300_     immediate
        HALT               x3204                       Base + offset
num1    .FILL x3301        x3205
num2    .FILL x3302        x3206
num0    .FILL x3300        x3207
        .END               x3208
```

-0

Contents of memory:

x3300	1010 1110 0011 0000	AE30
x3301	0001 0000 1100 1111	10CF
x3302	0101 0011 1001 0000	5390
x3303	1111 1111 0000 0000	FF00

25 pts 4. Convert the decimal number 10.625 to

(a) Binary number

(b) IEEE 754 32-bit floating point representation

-0

(a)

10 / 2 = 5 R 0

5 / 2 = 2 R 1

2 / 2 = 1 R 0

1 / 2 = 0 R 1

10 = 1010

.625 x 2 = 1.250

.250 x 2 = 0.5

.5 x 2 = 1.0

.625 = 101

Answer

10.625 = 1010.101

✓

(b) $1010.101 = (-1)^0 \times 1.010101 \times 2^3$

$3 = E - 127$

$E = 130$

Answer

0 10000010 01010100000000000000000

✓

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

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

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

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