ECE 120 Final Exam
Spring 2017

Tuesday, May 9, 2017

Name: SOLUTIONS

NetID: ____________

Discussion Section and TA name:

9:00 AM	[ ] AB1 Rui	
10:00 AM	[ ] AB2 Rui	
11:00 AM	[ ] AB3 Matt	
12:00 PM	[ ] AB4 Pawel	
1:00 PM	[ ] AB5 Pawel	
2:00 PM	[ ] AB6 Gowthami	[ ] ABA Huiren
3:00 PM	[ ] AB7 Gowthami	[ ] ABB Huiren
4:00 PM	[ ] AB8 Yu-Hsuan	[ ] ABC Sifan
5:00 PM	[ ] AB9 Yu-Hsuan	[ ] ABD Surya

- **Be sure that your exam booklet has 15 pages.**
- **Write your name, netid and check discussion section on the title page.**
- **Do not tear the exam booklet apart, except for the last 4 pages.**
- **Use backs of pages for scratch work if needed.**
- **This is a closed book exam. You may not use a calculator.**
- **You are allowed two handwritten 8.5 x 11" sheets of notes (both sides).**
- **Absolutely no interaction between students is allowed.**
- **Clearly indicate any assumptions that you make.**
- **The questions are not weighted equally. Budget your time accordingly.**

Problem 1	17 points	____________
Problem 2	14 points	____________
Problem 3	14 points	____________
Problem 4	14 points	____________
Problem 5	12 points	____________
Problem 6	15 points	____________
Problem 7	14 points	____________
Total	100 points	____________

Problem 1 (17 points): Binary Representation and Operations, Hamming codes

1. **(2 points)** A presidential term in the US lasts 1461 days. If the President of the US decided to refer to each day using fixed-length binary words, what is the **minimum number of bits** needed per day?

   Minimum number of bits: 11 (decimal number)

2. **(4 points)** Convert the following 24-bit pattern to hexadecimal:

   1010 1100 1100 1110 1101 1110_2 = x ACCEDE (hexadecimal number)

3. **(4 points)** Perform the following **bitwise** logical operations.

   a) 0110 XOR 0011 = 0101

   b) (NOT(0101)) NOR 1001 = 0100

4. **(4 points)** Perform the following operation in **4-bit 2's complement** representation.

   1001 + 10 = 0111

   Circle one: Carry out? [✓ **YES**] **NO**

   Circle one: Overflow? [✓ **YES**] **NO**

5. **(3 points)** You received the following 7-bit message encoded with a Hamming code: $X_7X_6X_5X_4X_3X_2X_1$ = 0010010. Does the message have an error or not?

   Circle one: [✓ **YES**] **NO**

   If you think there is an error, circle the bit below that is in error:

   [✓ 0] 0 1 0 0 1 0

Problem 2 (14 points): LC-3 Code and Datapath Control Signals

1. **(5 points)** The following fragment of assembly code is part of a larger program, in which all labels have been properly defined. Translate each line into LC-3 machine code. Some lines have been done for you. Use spaces to separate groups of bits.

Assembly Code	Machine Code
`LD R2, BITS`	0010 010 111111100
`ADD R3, R2, R2`	0001 011 010 0 00 010
`AND R2, R2, #-1`	0101 010 010 1 11111
`BRzp SKIP`	0000 011 000000001
`ADD R3, R3, #1`	0001 011 011 1 00001
`SKIP ST R3, BITS`	0011 011 111110111

What operation does this fragment of code perform on the bits stored at the memory address labeled BITS? **Circle one answer.**

Negation   Arithmetic shift left   Logical shift left   [✓ Cyclic (circular) shift left]

Bitwise NOT   Arithmetic shift right   Logical shift right   Cyclic (circular) shift right

2. **(9 points)** In Patt & Patel's LC-3 datapath, the implementation of some FSM states is not unique. For example, **state number 23** (MDR←SR), one of the states used to execute store instructions in the LC-3 FSM, can be implemented in two different ways. Implement both alternatives for state number 23 by completing the following table (with values 0, 1, or X). **If an answer is 'don't care' then you must write X.**

	LD.BEN	LD.MAR	LD.MDR	LD.IR	LD.PC	LD.REG	LD.CC	GateMARMUX	GateMDR	GateALU	GatePC	MARMUX	PCMUX(2)		ADDR1MUX	ADDR2MUX(2)		DRMUX(2)		SR1MUX(2)		ALUK(2)		MIO.EN	R.W
(a)	0	0	1	0	0	0	0	0	0	1	0	X	X	X	X	X	X	X	X	0	0	1	1	0	X
(b)	0	0	1	0	0	0	0	1	0	0	0	1	X	X	1	0	0	X	X	0	0	X	X	0	X

Problem 3 (14 points): LC-3 Assembly Programming

The program below waits for the user to type a single letter, flips the case of that letter, and then prints the message "`Your flipped-case letter is> `" followed by the flipped-case letter.

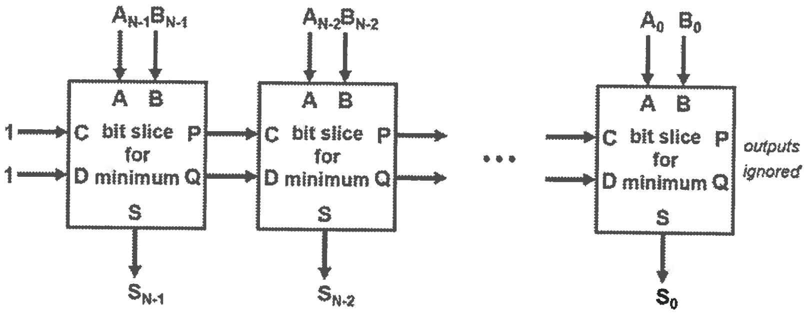
For example, if the user types '`A`' (ASCII #65), the program changes the case to '`a`' (ASCII #97). Similarly, if the user types '`z`' (ASCII #122), the program changes the case to '`Z`' (ASCII #90). (You can refer to the ASCII table provided on the last page of this exam.)

Write the missing lines of code. You must **write only one instruction per missing line**. Assume that the user only types letters.

LINE	PROGRAM		
1		.ORIG x3000	
2		**; Get character**	
3		GETC or IN	
4		**; Check case of character**	
5		LD	R1, Compare
6		ADD	R1, R0, R1
7		BRn	ToLowerCase
8		**; Change the case**	
9	ToUpperCase	LD	R2, Negative
10		BRnzp	ChangeCase
11	ToLowerCase	LD	R2, Positive
12	ChangeCase	ADD	R2, R0, R2
13		**; Print "Your flipped-case letter is> "**	
14		LEA	R0, Message
15		PUTS	
16		**; Print modified character**	
17		ADD	R0, R2, #0
18		OUT	
19		HALT	
20	Compare	.FILL	#-96
21	Message	.STRINGZ	"Your flipped-case letter is> "
22	Positive	.FILL	#32
23	Negative	.FILL	#-32
24		.END	

Problem 4 (14 points): Bit Slices and Abstraction

The bit-sliced design below finds the smaller of two N-bit unsigned numbers, $A=A_{N-1}A_{N-2}\ldots A_0$ and $B=B_{N-1}B_{N-2}\ldots B_0$. As shown in the diagram, information flows from the bit slice for the most significant bits to the bit slice for the least significant bits. The smaller number, $S=S_{N-1}S_{N-2}\ldots S_0$, (either A or B) comes out at the bottom.



The two bits passed between slices (and into the slice on the left) use the representation shown in the table to the right.

CD/PQ	meaning
00	not used
01	A < B
10	A > B
11	A = B

1. **(8 points)** The K-maps below represent the outputs P and S for the bit slice. For each K-map, find an expression with minimal area. You must consider both minimal SOP and minimal POS solutions, but circle loops for only the better of the two choices (SOP or POS) and write the corresponding expressions. Extra copies of each K-map are provided on the next page, but only the copies on this page will be graded.

P

CD

AB	00	01	11	10
00	x	0	1	1
01	x	0	0	1
11	x	0	1	1
10	x	0	1	1

S

CD

AB	00	01	11	10
00	x	0	0	0
01	x	0	0	1
11	x	1	1	1
10	x	1	0	0

P = $C(A+B'+D')$

S = $(A+D')(B+C')$

Problem 4 (14 points): Bit Slices and Abstraction, extra K-maps

Use the following K-maps as scratch copies. **We will not grade any work on this page.**

P

CD

AB	00	01	11	10
00	x	0	1	1
01	x	0	0	1
11	x	0	1	1
10	x	0	1	1

S

CD

AB	00	01	11	10
00	x	0	0	0
01	x	0	0	1
11	x	1	1	1
10	x	1	0	0

P

CD

AB	00	01	11	10
00	x	0	1	1
01	x	0	0	1
11	x	0	1	1
10	x	0	1	1

S

CD

AB	00	01	11	10
00	x	0	0	0
01	x	0	0	1
11	x	1	1	1
10	x	1	0	0

P

CD

AB	00	01	11	10
00	x	0	1	1
01	x	0	0	1
11	x	0	1	1
10	x	0	1	1

S

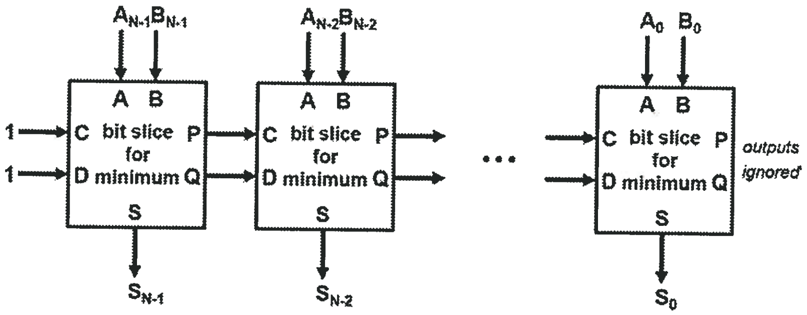
CD

AB	00	01	11	10
00	x	0	0	0
01	x	0	0	1
11	x	1	1	1
10	x	1	0	0

(Problem 4 continues on next page)

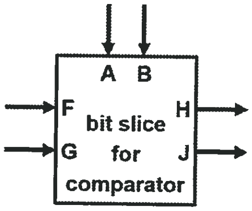
Problem 4 (14 points): Bit Slices and Abstraction, continued

(Figure and representation replicated for your convenience.)

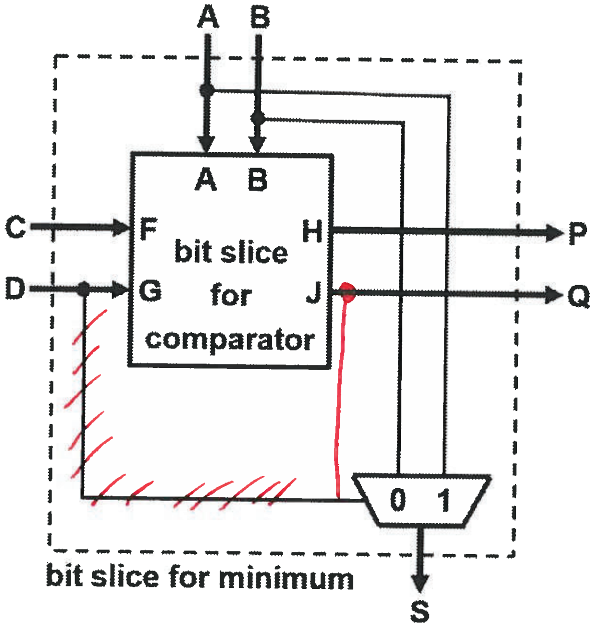


CD/PQ	meaning
00	not used
01	A < B
10	A > B
11	A = B

Your ECE120 lab partner suggests that rather than implementing the bit slice from gates, one can use a comparator bit slice and a mux, as shown below. The representation used for bits between comparator bit slices is the same as the representation used between the bit slices being implemented, as shown to the right.



FG/HJ	meaning
00	not used
01	A < B
10	A > B
11	A = B



2. **(3 points)** Unfortunately, your lab partner's implementation does not work correctly. Explain why, using **TEN WORDS OR FEWER**.

S based on previous rather than current bits AB.

3. **(3 points)** Indicate how to fix the implementation by marking on the diagram above. **You may not use any additional components nor gates.**

Problem 5 (12 points): LC-3 Instruction Control

In this problem, we introduce a new instruction ABC, with opcode 1101, to the LC-3 instruction set:

ABC BaseR1, BaseR2, imm5

After decode, the instruction ABC is defined by the following sequence of six RTL statements. **Note that register R6 is used as a temporary register.**

MAR ← BaseR1
MDR ← M[MAR]
R6 ← MDR
MDR ← R6 AND SEXT(imm5), Setcc
MAR ← BaseR2
M[MAR] ← MDR

1. **(4 points)** Express the functionality of ABC **in a single-line RTL expression**.

   RTL expression: M[BaseR2] ← M[BaseR1] AND sext(imm5), Setcc

2. **(6 points)** Provided below is the microinstruction corresponding to ABC's first RTL statement (MAR ← BaseR1). Give the control ROM address of this microinstruction.

   Control ROM address: 1101 = 13

IRD	COND(3)	J(6)	LD.BEN	LD.MAR	LD.MDR	LD.IR	LD.PC	LD.REG	LD.CC	GateMARMUX	GateMDR	GateALU	GatePC	MARMUX	PCMUX(2)	ADDR1MUX	ADDR2MUX(2)	DRMUX(2)	SR1MUX(2)	ALUK(2)	MIO.EN	R.W
0	000	111000	0	1	0	0	0	0	0	0	0	1	0	0	00	0	00	00	01	11	0	0

Fill in the 16 boxes below to give the binary encoding of the instruction **ABC R3, R2, #-5**. **Your answer MUST be consistent with the above MAR ← BaseR1 microinstruction.**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	1

3. **(2 points)** In **TWENTY WORDS OR FEWER**, explain your choice (0 or 1) of **bit 5** in the ABC instruction.

   IR[5] needs to be 1 so that the SR2MUX selects sext IR[4:0] as the B input to the ALU.

Problem 6 (15 points): Decoders and Counters

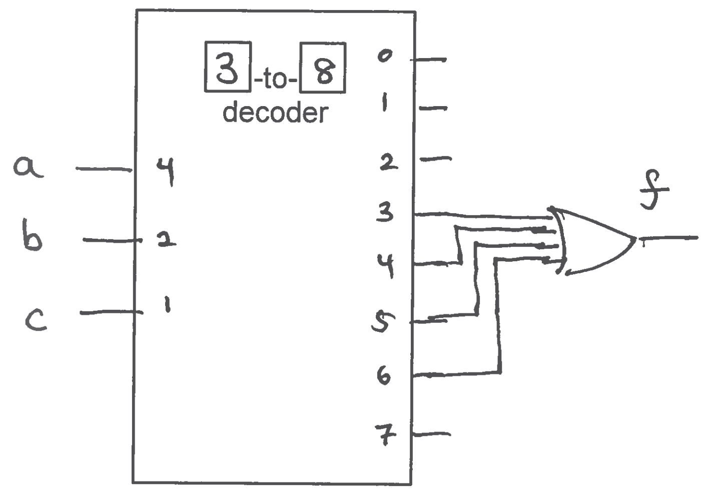
1. **(5 points)** Consider the 3-variable function **f(a,b,c) = a⊕(bc)**, where ⊕ denotes XOR. *Hint: you may wish to draw a truth table.*

$a \oplus bc = \bar{a}bc + a\overline{bc} = \bar{a}bc + a\bar{b} + a\bar{c}$

a) Express f(a,b,c) in canonical sum of products form.

Canonical SOP form f(a,b,c) = $\bar{a}bc + a\bar{b}\bar{c} + a\bar{b}c + ab\bar{c}$

b) Implement f(a,b,c) using the decoder shown below and **an OR gate**. Draw and label all decoder inputs and outputs. Label the size of the decoder by filling in the boxes.



2. **(4 points)** Complete the K-maps below corresponding to a standard 3-bit up-counter (counts the sequence 0, 1, 2 … 7, 0, 1 …) with state $Q_2Q_1Q_0$. Give minimal SOP expressions for Q_2^+, Q_1^+, and Q_0^+.

Q_2^+

Q_2 \ Q_1Q_0	00	01	11	10
0	0	0	1	0
1	1	1	0	1

Q_1^+

Q_2 \ Q_1Q_0	00	01	11	10
0	0	1	0	1
1	0	1	0	1

Q_0^+

Q_2 \ Q_1Q_0	00	01	11	10
0	1	0	0	1
1	1	0	0	1

Minimal SOP for Q_2^+ = $Q_2\bar{Q}_1 + Q_2\bar{Q}_0 + \bar{Q}_2Q_1Q_0$

Minimal SOP for Q_1^+ = $\bar{Q}_1Q_0 + Q_1\bar{Q}_0$

Minimal SOP for Q_0^+ = $\bar{Q}_0$

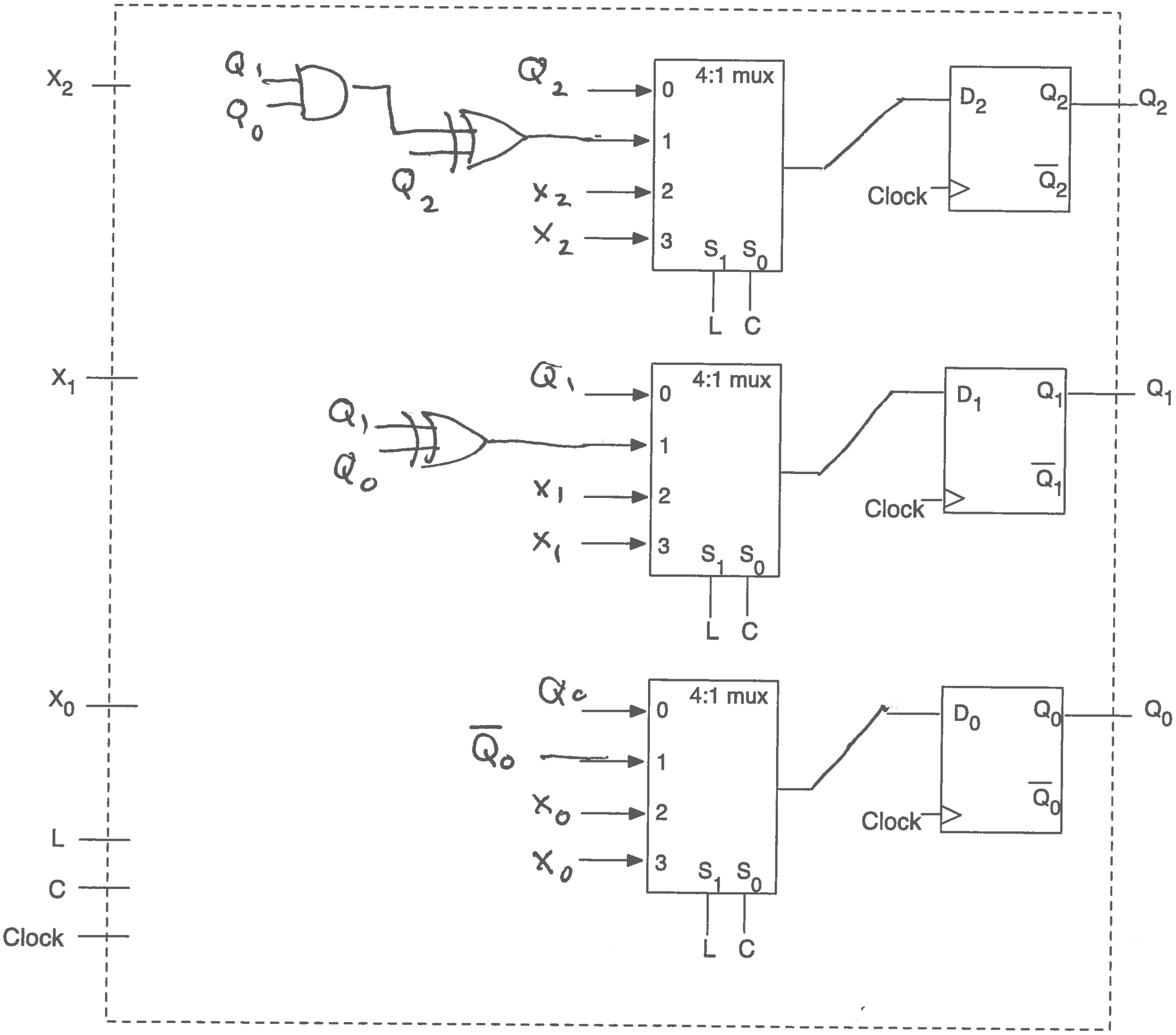
Problem 6 (15 points): Decoders and Counters, continued

3. **(6 points)** Implement a 3-bit up-counter with **parallel load** and **pause** operations. Your counter should have control inputs **L (load)** and **C (count);** data inputs X_2, X_1, X_0; outputs Q_2, Q_1, Q_0.

The L and C inputs operate as follows:

- If L = 1, then a parallel load is performed.
- If L = 0 and C = 1, the circuit works as a 3-bit up-counter.
- If L = 0 and C = 0, the counter pauses (that is, it stays in the same state.)

As shown below, the counter uses three D flip-flops and three 4:1 multiplexers. For your convenience, the L, C, and Clock lines are provided for you. Complete the counter implementation using as few gates (AND, OR, NOT, XOR) as possible. For full credit, use only **2 XOR gates and 1 AND gate. Inverted inputs are not available.** *Hint: Recall function f(a,b,c) from part 1 of this problem.*



Problem 7 (14 points): LC-3 Interpretation and Assembly

1. **(5 points)** Decode each of the following LC-3 instructions, writing the RTL in the box beside the instruction. For full credit, your RTL must include specific values for each operand (for example, "R4" rather than "DR"), and must be sign-extended when appropriate. You need not, however, perform calculations such as addition of the PC value.

   Write any immediate values either as **4-digit hexadecimal** (prefix them with "x") or as **decimal** (prefix them with "#"). DO NOT USE ANY OTHER NOTATION.

   Hint: Draw lines between bits to separate the instructions into appropriate fields.

instruction bits	RTL meaning
0001 1110 1011 0010	R7 ← R2 - #14, Setcc
1110 0101 1111 1000	R2 ← PC −#8, Setcc
0101 1100 0000 0101	R6 ← R0 AND R5, Setcc
0011 0110 1000 0000	M[PC+ #128] ← R3

2. **(9 points)** Sadly, Prof. Lumetta forgot to include comments in the program below. Complete the program's symbol table below to the right. Fill only as many rows as necessary.

```
        .ORIG x3000
        AND R5,R5,#0
        LD R1,SADDR
LOOP    LDR R4,R1,#0
        BRz DONE
        ADD R1,R1,#1
        ADD R5,R5,#1
        BRnzp LOOP
DONE    NOT R5,R5
        ADD R5,R5,#1
        LEA R3,RESULT
        STR R1,R3,#0
        STR R5,R3,#1
        HALT
RESULT  .BLKW #2
SADDR   .FILL x4000
        .END
```

Symbol Table

↙ x's optional

LOOP	x3002
DONE	x3007
RESULT	x300D
SADDR	x300F

Given that the string `"Why-ECEB?"` is placed in memory starting at address x4000, write the **4-digit hexadecimal values** held in the following memory locations when the program halts.

M[RESULT]= x4009

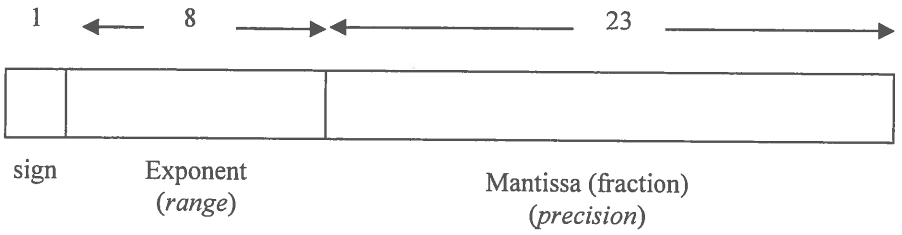
M[RESULT+#1]= xFFF7

M[R1] = x0000

Table of ASCII Characters

Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	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

IEEE 754 32-bit floating point format



The actual number represented in this format is:

$$(-1)^{\boxed{s}} \times 1.\boxed{\text{mantissa}} \times 2^{\boxed{\text{exp.}}\ -127}$$

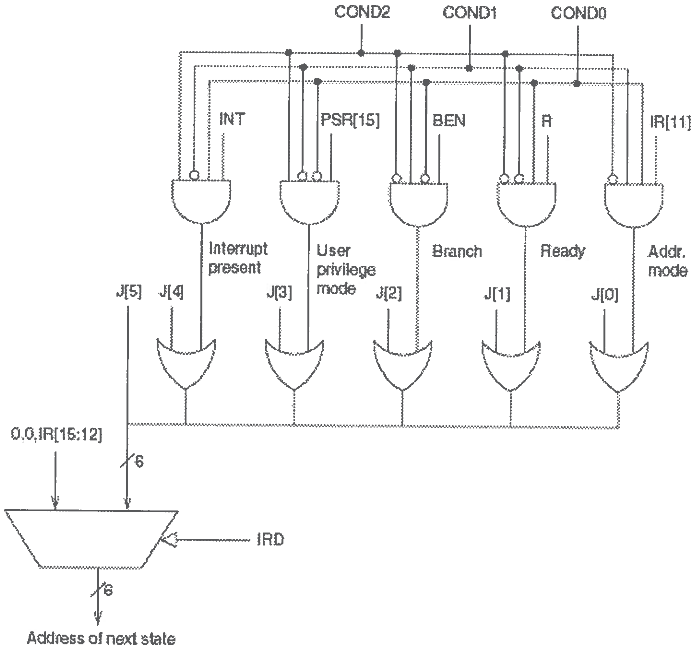
where 1≤ exponent ≤ 254 for normalized representation.

LC-3 Control Word Fields

IRD	COND[2:0]	J[5:0]	LD.BEN	LD.MAR	LD.MDR	LD.IR	LD.PC	LD.REG	LD.CC	GateMARMUX	GateMDR	GateALU	GatePC	MARMUX	PCMUX	ADDR1MUX	ADDR2MUX	DRMUX	SR1MUX	ALUK	MIO.EN	R.W

LC-3 Microsequencer Control

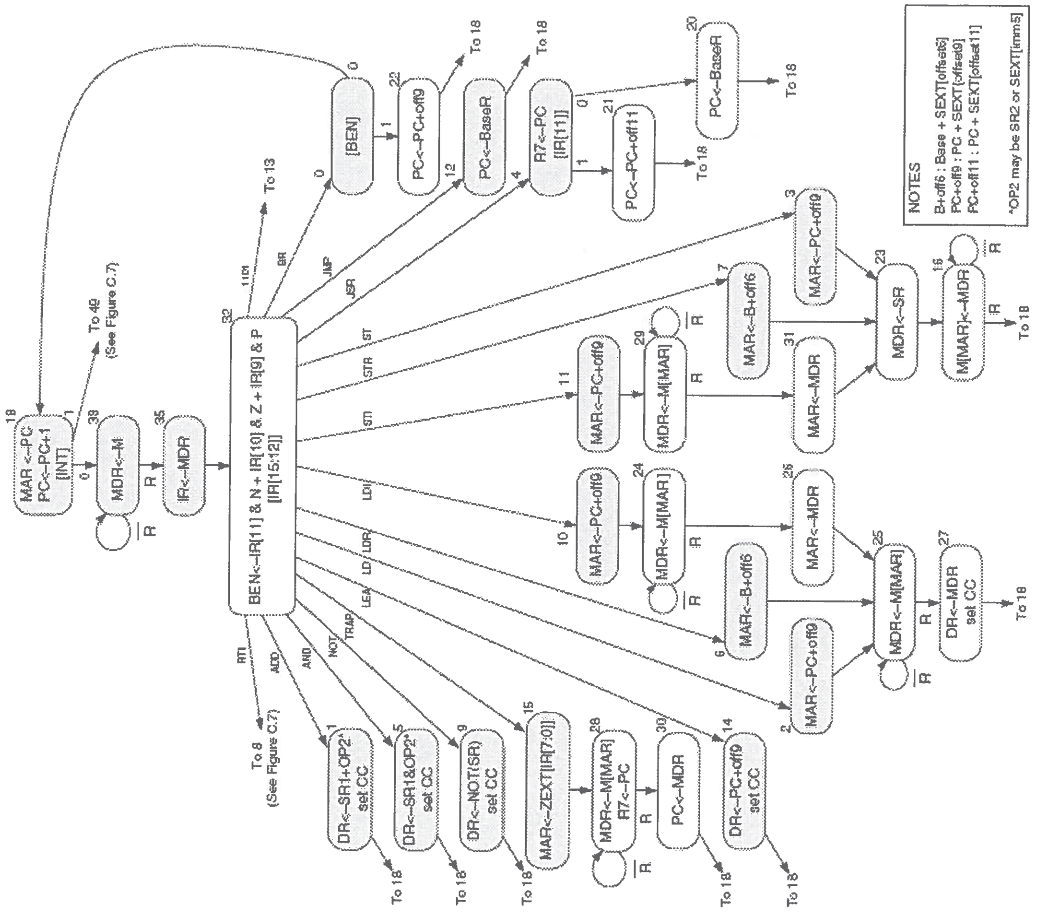
Signal	Description
IRD	= 1, CAR ← 00||opcode (opcode = IR[15:12]), only during decode = 0, CAR ← J (plus 1,2,4,8,16 depending on COND bits)
COND	= 000, CAR ← J = 001, IF (R=1 and J[1]=0) THEN (CAR ← J plus 2) ELSE (CAR ← J) = 010, IF (BEN=1 and J[2]=0) THEN (CAR ← J plus 4) ELSE (CAR ← J) = 011, IF (IR[11]=1 and J[0]=0) THEN (CAR ← J plus 1) ELSE (CAR ← J)
J	6-bit next value for CAR (plus modifications depending on COND bits)



LC-3 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.

LC-3 FSM



NOTES: RTL corresponds to execution (after fetch!); JSRR not shown

LC-3 Instructions

ADD	0001	DR	SR1	0	00	SR2	ADD DR, SR1, SR2

DR ← SR1 + SR2, Setcc

ADD	0001	DR	SR1	1	imm5	ADD DR, SR1, *imm5*

DR ← SR1 + SEXT(imm5), Setcc

AND	0101	DR	SR1	0	00	SR2	AND DR, SR1, SR2

DR ← SR1 AND SR2, Setcc

AND	0101	DR	SR1	1	imm5	AND DR, SR1, *imm5*

DR ← SR1 AND SEXT(imm5), Setcc

BR	0000	n	z	p	PCoffset9	BR{nzp} *PCoffset9*

((n AND N) OR (z AND Z) OR (p AND P)):
PC ← PC + SEXT(PCoffset9)

JMP	1100	000	BaseR	000000	JMP BaseR

PC ← BaseR

JSR	0100	1	PCoffset11	JSR *PCoffset11*

R7 ← PC, PC ← PC + SEXT(PCoffset11)

TRAP	1111	0000	trapvect8	TRAP *trapvect8*

R7 ← PC, PC ← M[ZEXT(trapvect8)]

LD	0010	DR	PCoffset9	LD DR, *PCoffset9*

DR ← M[PC + SEXT(PCoffset9)], Setcc

LDI	1010	DR	PCoffset9	LDI DR, *PCoffset9*

DR ← M[M[PC + SEXT(PCoffset9)]], Setcc

LDR	0110	DR	BaseR	offset6	LDR DR, BaseR, *offset6*

DR ← M[BaseR + SEXT(offset6)], Setcc

LEA	1110	DR	PCoffset9	LEA DR, *PCoffset9*

DR ← PC + SEXT(PCoffset9), Setcc

NOT	1001	DR	SR	111111	NOT DR, SR

DR ← NOT SR, Setcc

ST	0011	SR	PCoffset9	ST SR, *PCoffset9*

M[PC + SEXT(PCoffset9)] ← SR

STI	1011	SR	PCoffset9	STI SR, *PCoffset9*

M[M[PC + SEXT(PCoffset9)]] ← SR

STR	0111	SR	BaseR	offset6	STR SR, BaseR, *offset6*

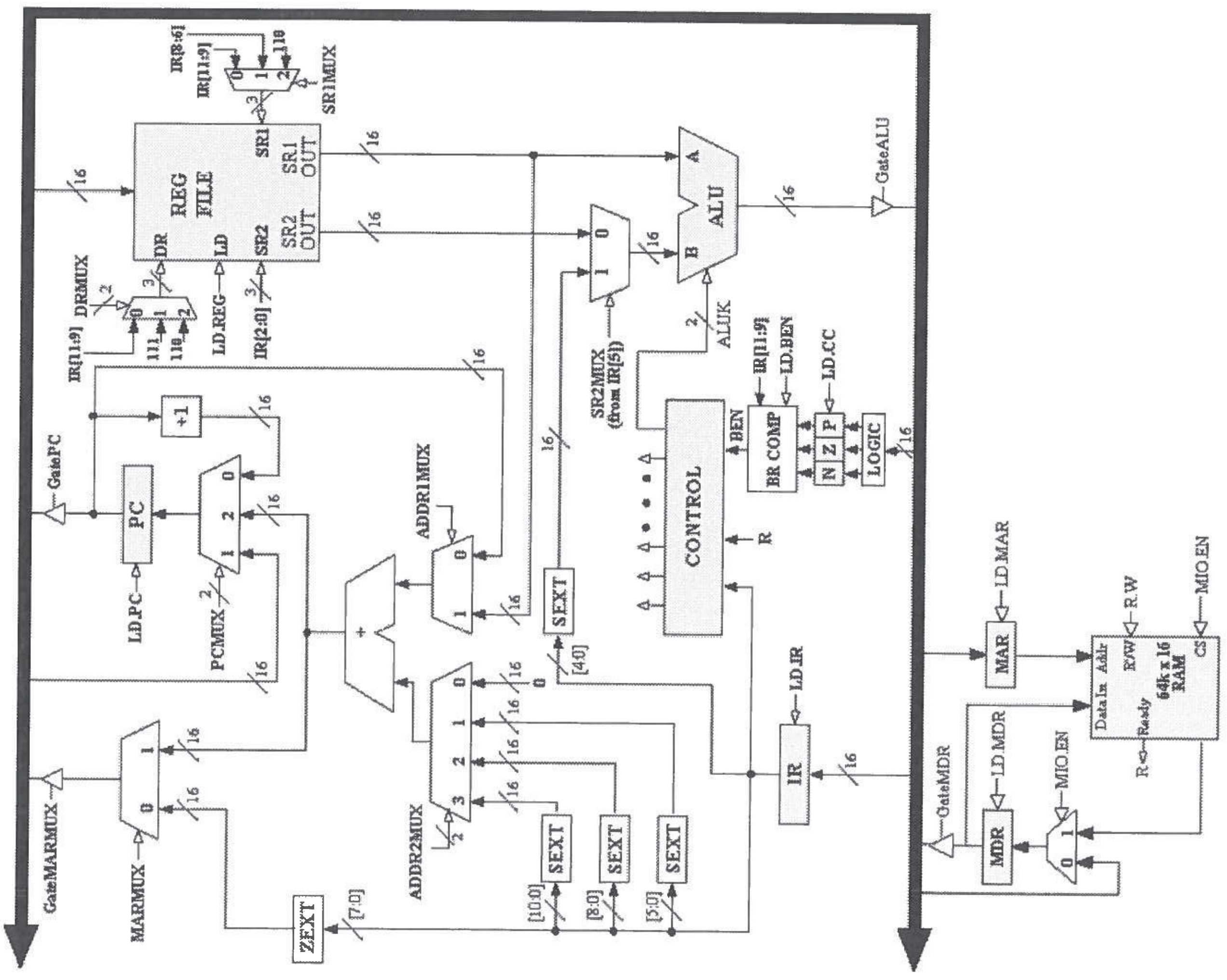
M[BaseR + SEXT(offset6)] ← SR

LC-3 Datapath Control Signals

Signal	Description
LD.MAR	= 1, MAR is loaded
LD.MDR	= 1, MDR is loaded
LD.IR	= 1, IR is loaded
LD.PC	= 1, PC is loaded
LD.REG	= 1, register file is loaded
LD.BEN	= 1, updates Branch Enable (BEN) bit
MARMUX	= 0, chooses ZEXT IR[7:0] = 1, chooses address adder output
ADDR1MUX	= 0, chooses PC = 1, chooses reg file SR1OUT
ADDR2MUX	= 00, chooses "0...00" = 01, chooses SEXT IR[5:0] = 10, chooses SEXT IR[8:0] = 11, chooses SEXT IR[10:0]
PCMUX	= 00, chooses PC + 1 = 01, chooses system bus = 10. chooses address adder output
SR1MUX	= 00, chooses IR[11:9] = 01, chooses IR[8:6] = 10, chooses "110"

Signal	Description
LD.CC	= 1, updates status bits from system bus
GateMARMUX	= 1, MARMUX output is put onto system bus
GateMDR	= 1, MDR contents are put onto system bus
GateALU	= 1, ALU output is put onto system bus
GatePC	= 1, PC contents are put onto system bus
MIO.EN	= 1, Enables memory, chooses memory output for MDR input = 0, Disables memory, chooses system bus for MDR input
R.W	= 1, M[MAR]<-MDR when MIO.EN = 1 = 0, MDR<-M[MAR] when MIO.EN = 1
ALUK	= 00, ADD = 01, AND = 10, NOT A = 11, PASS A
DRMUX	= 00, chooses IR[11:9] = 01, chooses "111" = 10, chooses "110"

LC-3 Datapath



IR[8:6]
IR[11:9]
110
SR1MUX
REG FILE
SR1
SR1 OUT
SR2 OUT
SR2
DR
LD
DRMUX
IR[11:9]
111
110
LD.REG
IR[2:0]
16
3
2
ALU
A
B
GateALU
ALUK
SR2MUX (from IR[5])
+1
GatePC
PC
LD.PC
PCMUX
ADDR1MUX
ADDR2MUX
MARMUX
GateMARMUX
SEXT
ZEXT
[10:0]
[8:0]
[5:0]
[7:0]
[4:0]
CONTROL
R
BEN
BR COMP
IR[11:9]
LD.BEN
N
Z
P
LD.CC
LOGIC
IR
LD.IR
MAR
LD.MAR
MDR
LD.MDR
GateMDR
MIO.EN
R.W
64k x 16 RAM
Addr
R/W
CS
DataIn
Ready
R