Solutions

ECE 120 Final Exam
Spring 2016

Friday, May 13, 2016

Name: ______________________ NetID: ______________

Discussion Section:

9:00 AM	[ ] AB1	
10:00 AM	[ ] AB2	
11:00 AM	[ ] AB3	
12:00 PM	[ ] AB4	
1:00 PM	[ ] AB5	[ ] ABA
2:00 PM	[ ] AB6	
3:00 PM	[ ] AB7	[ ] ABB
4:00 PM	[ ] AB8	[ ] ABC
5:00 PM	[ ] AB9	[ ] ABD

- **Be sure that your exam booklet has 12 pages.**
- **Write your name, netid and check discussion section on the title page.**
- **Do not tear the exam booklet apart.**
- **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" sheet 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	16 points	________
Problem 2	11 points	________
Problem 3	19 points	________
Problem 4	15 points	________
Problem 5	22 points	________
Problem 6	20 points	________
Problem 7	22 points	________
Total	125 points	________

Problem 1 (16 points): Binary Representation and Operations

1. (12 points) Suppose an **8-bit** processor performs **2's complement arithmetic addition and subtraction** and generates the following one-bit condition codes: **P** for *positive*, **N** for *negative*, **O** for *overflow*, and **C** for *carryout* bit. For each of the operations below, give the result of the operation (**in hexadecimal with correct number of digits**) as performed by this processor and give the value of the condition code bits (**in binary**) after the operation is complete.

Operation	Result of arithmetic operation (in hexadecimal)	P	N	O	C
10000000 - 00011011	x65	1	0	1	1
10101010 + 11101110	x98	0	1	0	1
11000101 + 00111011	x00	0	0	0	1

2. (4 points) Suppose a 24-bit instruction takes the following format:

OPCODE	DR	SR1	SR2	UNUSED

a. If there are 40 opcodes and 24 registers, what is the **minimum** number of bits required to represent:

the opcodes? 6 bits

the source register 1 (SR1)? 5 bits

b. What is the **maximum** number of UNUSED bits in this instruction encoding?

Answer: 3 bits

c. If during a complete re-design of the ISA there are 3 times the number of opcodes we had before, how many **additional bits** would be required to represent the opcodes?

Answer: 1 additional bits

Problem 2 (11 points): Synchronous Counter

Using D flip-flops, design a 3-bit counter that counts in the following sequence: 1, 5, 2, 6, 7, 1 …

1. (8 points) The current state of the counter is denoted by $S_2S_1S_0$. Fill in the K-maps for S_2^+, S_1^+ and S_0^+ **using don't cares whenever possible**.

S_2^+		S_1S_0			
		00	01	11	10
S_2	0	X	1	X	1
	1	X	0	0	1

S_1^+		S_1S_0			
		00	01	11	10
S_2	0	X	0	X	1
	1	X	1	0	1

S_0^+		S_1S_0			
		00	01	11	10
S_2	0	X	1	X	0
	1	X	0	1	1

2. (3 points) Write **minimal SOP** Boolean expressions for S_2^+, S_1^+, and S_0^+.

$S_2^+ =$ $\overline{S_0} + \overline{S_2}$

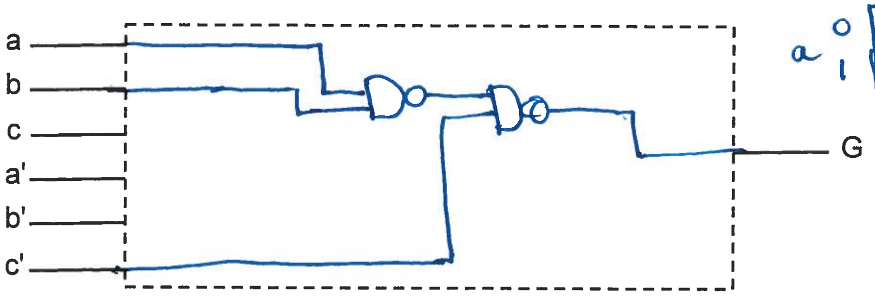
$S_1^+ =$ $\overline{S_0} + S_2\overline{S_1}$

$S_0^+ =$ $S_2S_1 + \overline{S_2}S_0 = S_2S_1 + \overline{S_2}\,\overline{S_1}$

Problem 3 (19 points): Combinational logic structures

Consider the Boolean function **F(a,b,c) = OR(m_1, m_3, m_6, m_7)**, where m_i is minterm *i*.

1. (4 points) Let G(a,b,c) be equal to F(a,b,c), **except that input abc=101 will never arise**. Give a **minimal 2-level NAND gate** implementation of G. *Complemented inputs are available.*



a \ bc	00	01	11	10
0	0	1	1	0
1	0	0	1	1

F

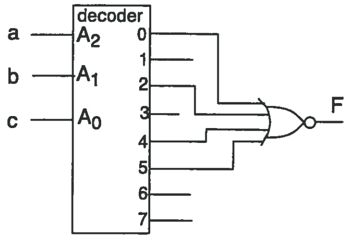
0	1	1	0
0	x	1	1

G

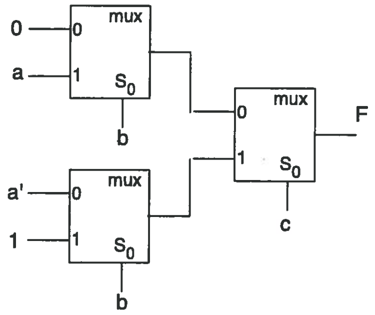
2. (15 points) For each of the following circuits, does it implement **F(a,b,c) = OR(m_1, m_3, m_6, m_7)**? For each circuit, circle 'Yes' or 'No'.

Note: There is a *guessing penalty*: +3 for correct, 0 for blank, -2 for wrong.

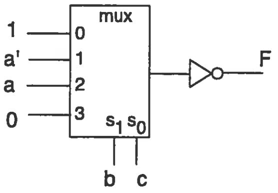
(a) [✓ Yes] No



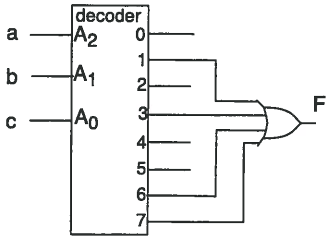
(b) [✓ Yes] No



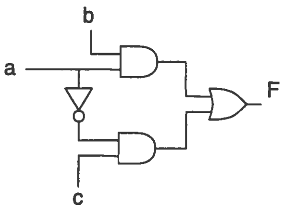
(c) Yes [✓ No]



(d) [✓ Yes] No



(e) [✓ Yes] No



Problem 4 (15 points): The LC-3 microprocessor

1. (3 points) List the state numbers that implement the entire instruction cycle of the LC-3 *STI instruction*.

Answer: 18, 33, 35, 32, 11, 29, 31, 23, 16

2. (12 points) Complete the following table by entering values (0, 1, or X) for the LC-3 microinstructions at ROM addresses 4 and 7.
Note: If the answer is '*don't care*' then you **must use 'X'**.

ROM Address	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
4	0	011	010100	0	0	0	0	0	1	0	0	0	0	1	x	xx	x	xx	01	xx	xx	0	x
7	0	000	010111	0	1	0	0	0	0	0	1	0	0	0	1	xx	1	01	xx	01	xx	0	x

Problem 5 (22 points): The LC-3 microprocessor

In this problem we introduce a new instruction to the LC-3 instruction set, called **`ANDMI`**: AND from Memory with Immediate data.

`ANDMI   DR,   SR,   imm5`

`ANDMI` has opcode 1101. It computes the bitwise AND of sext(**`imm5`**) and the memory word whose address is in register SR, and puts the result in the DR. **`ANDMI`** then sets the condition codes. The RTL is:

DR ← M[R(IR[8:6])] AND sext(imm5), Setcc

1. (4 points) Give the binary encoding of the instruction **`ANDMI   R3,   R4,   #-15`**

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

2. (2 points) Why would **IR[5]=0** in the **`ANDMI`** instruction not work?

Answer: IR[5] must be 1 in order that the sext(imm5)
operand is selected by the SR2MUX.

3. (8 points) In RTL form, give the sequence of **4 microinstructions** that implement the **`ANDMI`** instruction **after the decode state**. If needed, you may use register R6 as a temporary register.

Answer: MAR ← R(IR[8:6])
MDR ← M[MAR]
R6 ← MDR
DR ← R6 AND sext(imm5), setCC

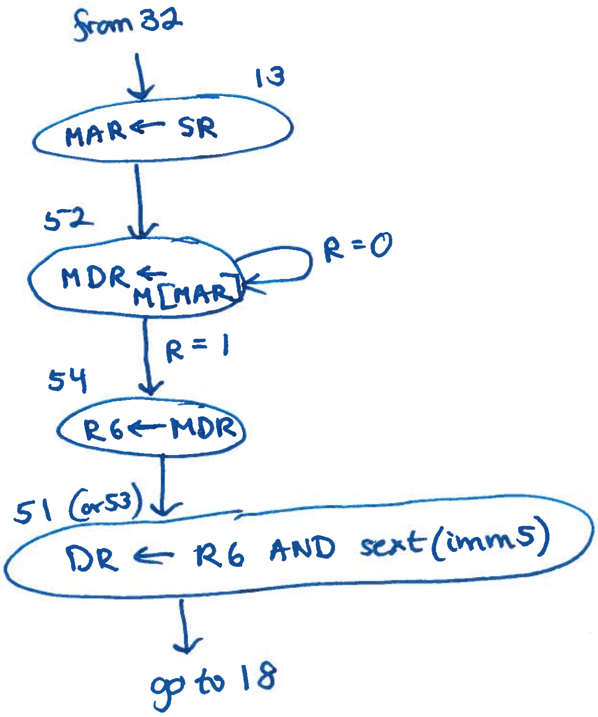
4. (2 points) The execute phase of **`ANDMI`** starts at what control ROM address?

Answer: 001101 (Answer in **binary** with the **correct number of bits**)

Problem 5 (continued)

5. (6 points) Draw the state diagram for **ANDMI**, *including the state numbers* **after the decode state**. When you need *additional* states, state numbers 51, 52, 53, and 54 are available for your use.

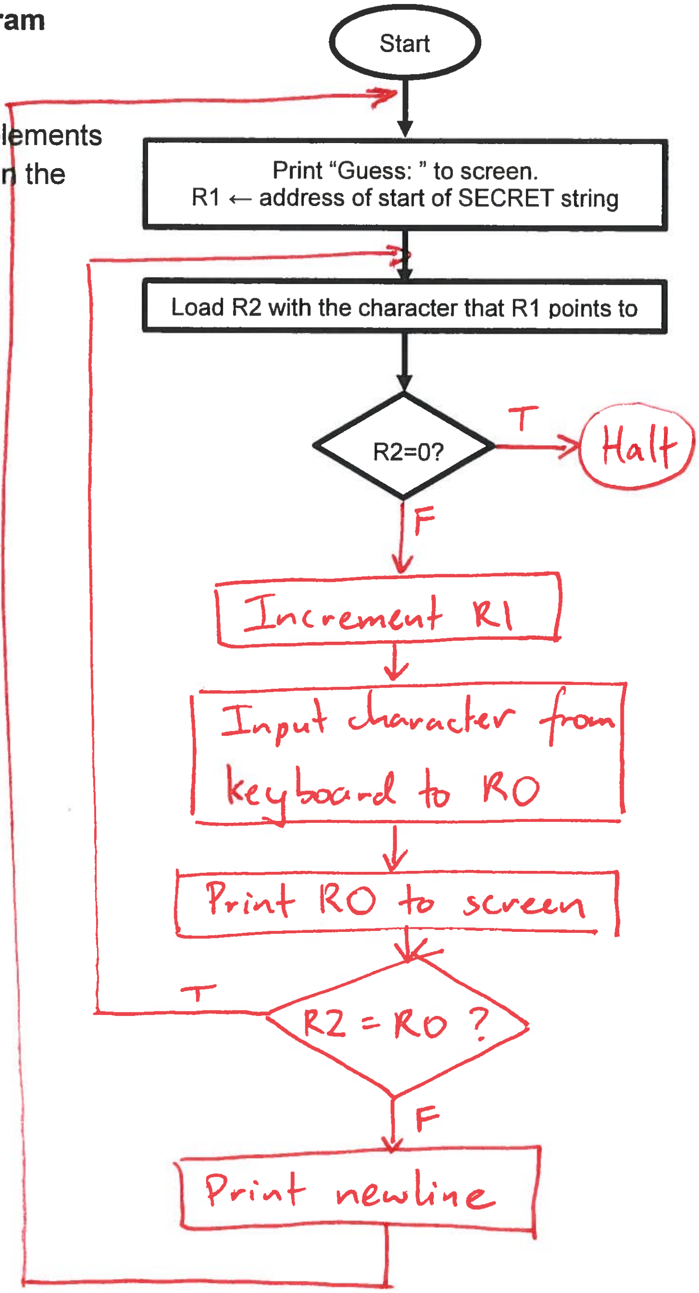
Note: $51_{10} = 110011_2$, $52_{10} = 110100_2$, $53_{10} = 110101_2$, $54_{10} = 110110_2$.



Problem 6 (20 points): LC-3 Assembly Program Analysis

1. (10 points) The following LC-3 program implements a guessing game. Complete the flowchart on the right for the same program.

```
            .ORIG       x3000
OUTER       LEA         R0, GUESS
            PUTS
            LEA         R1, SECRET
INNER       LDR         R2, R1, #0
            BRz         DONE
            ADD         R1, R1, #1
            GETC
            OUT
            NOT         R0, R0
            ADD         R0, R0, #1
            ADD         R0, R0, R2
            BRz         INNER
            LD          R0, NEWLINE
            OUT
            BRnzp       OUTER
DONE        HALT
NEWLINE     .FILL       x0A
SECRET      .STRINGZ    "BYTE"
GUESS       .STRINGZ    "Guess: "
            .END
```



Start
Print "Guess: " to screen.
R1 ← address of start of SECRET string
Load R2 with the character that R1 points to
R2=0?
T
Halt
F
Increment R1
Input character from keyboard to R0
Print R0 to screen
T
R2 = R0 ?
F
Print newline

2. (10 points) Write down exactly what this program prints to screen when the user runs the program and enters **BITBYTE** using a keyboard. Note that GETC inputs a character from the keyboard but does not print anything to the screen.

Guess: BI
Guess: T
Guess: BYTE

Problem 7 (22 points): LC-3 Machine Code Debugging

1. (2 points) Define the Hamming weight of a value to be the number of 1s in its binary representation.

   Write down the Hamming weight of the 16-bit value xECEB as a decimal number: _____

2. (20 points) The snippet of LC-3 machine code shown below is intended to compute the Hamming weight of a 16-bit value that is already provided in R1. The register usage is described now.

   R1: initially contains the 16-bit value
   R2: is used as a loop variable
   R3: will contain the result (Hamming weight of the initial value in R1)

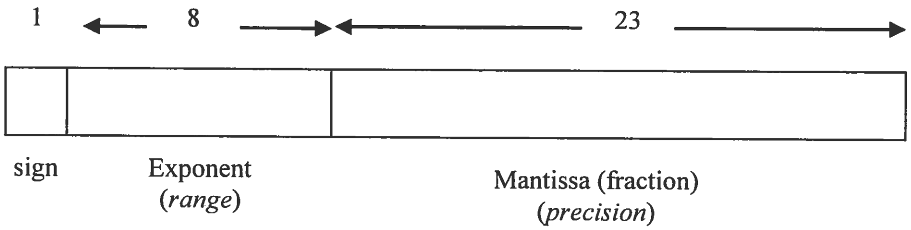
   Unfortunately, there is exactly 1 bit in error in each LC-3 machine code instruction. First translate the original machine code exactly into original assembly code (leaving the errors in). Then debug the machine code by circling the bit error in each line and writing down the corrected assembly code. The first three lines and the last line have been completed for you.

Original Machine Code	Original Assembly Code	Corrected Assembly Code
0001 010 010 1 00000	ADD R2,R2,#0	AND R2,R2,#0
0001 010 010 1 11111	ADD R2,R2,#-1	ADD R2,R2,#15
0101 111 011 1 00000	AND R7,R3,#0	AND R3,R3,#0
0101 001 001 1 00000	AND R1,R1, #0	ADD R1,R1, #0
0000 011 000000000	BRzp #0	BRzp #1
0001 011 010 1 00001	ADD R3,R2,#1	ADD R3,R3, #1
0001 001 001 1 00001	ADD R1,R1,#1	ADD R1,R1,R1
0001 010 010 1 11101	ADD R2,R2, #-3	ADD R2,R2,#-1
0000 111 111111010	BRnzp #-6	BRzp #-6

Fundamental Laws of Boolean Algebra

Commutativity	$x \cdot y = y \cdot x$	$x + y = y + x$
Associativity	$(x \cdot y) \cdot z = x \cdot (y \cdot z)$	$(x + y) + z = x + (y + z)$
Distributivity	$x \cdot (y + z) = xy + xz$	$x + yz = (x + y)(x + z)$
Idempotence	$x \cdot x = x$	$x + x = x$
Identity	$x \cdot 1 = x$	$x + 0 = x$
Null	$x \cdot 0 = 0$	$x + 1 = 1$
Complementarity	$x \cdot x' = 0$	$x + x' = 1$
Involution	$(x')' = x$	
DeMorgan's	$(x \cdot y)' = x' + y'$	$(x + y)' = x' \cdot y'$
Absorption	$x \cdot (x + y) = x$	$x + x \cdot y = x$
No-Name	$x \cdot (x' + y) = x \cdot y$	$x + x' \cdot y = x + y$
Consensus	$(x+y) \cdot (y+z) \cdot (x'+z) = (x+y) \cdot (x'+z)$	$x \cdot y + y \cdot z + x' \cdot z = x \cdot y + x' \cdot z$

IEEE 754 32-bit floating point format

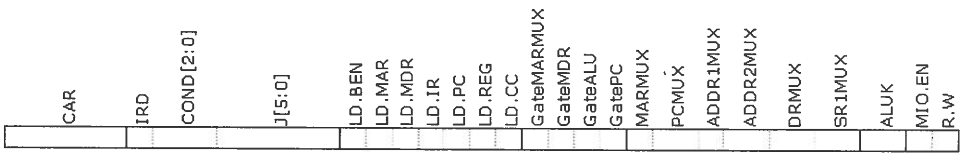


The actual number represented in this format is:

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

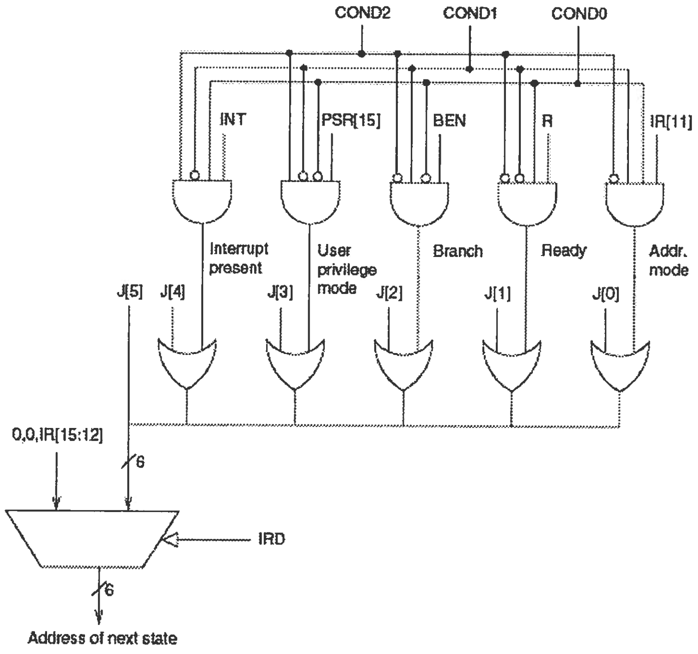
where $1 \leq$ exponent ≤ 254 for normalized representation.

LC-3 Control Word Fields

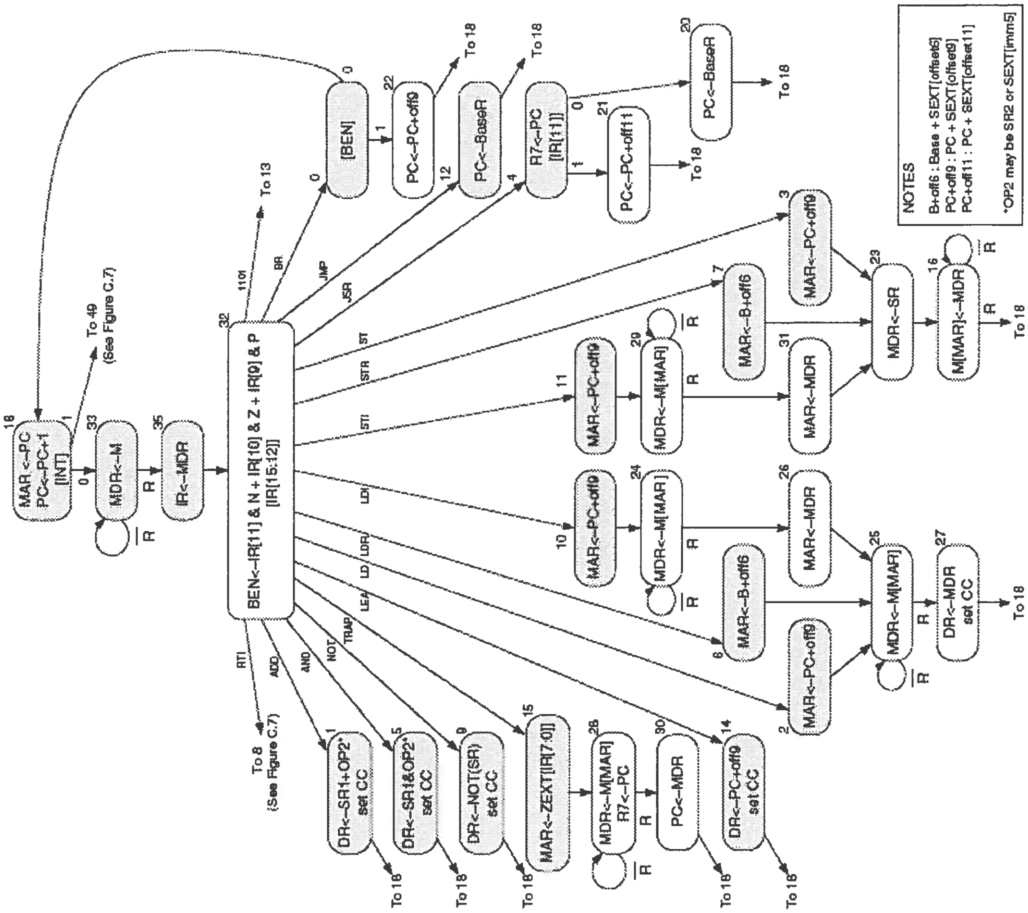


LC-3 Microsequencer Control

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 FSM



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

LC-3 Instructions

Instruction	Format	Syntax	RTL
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)]

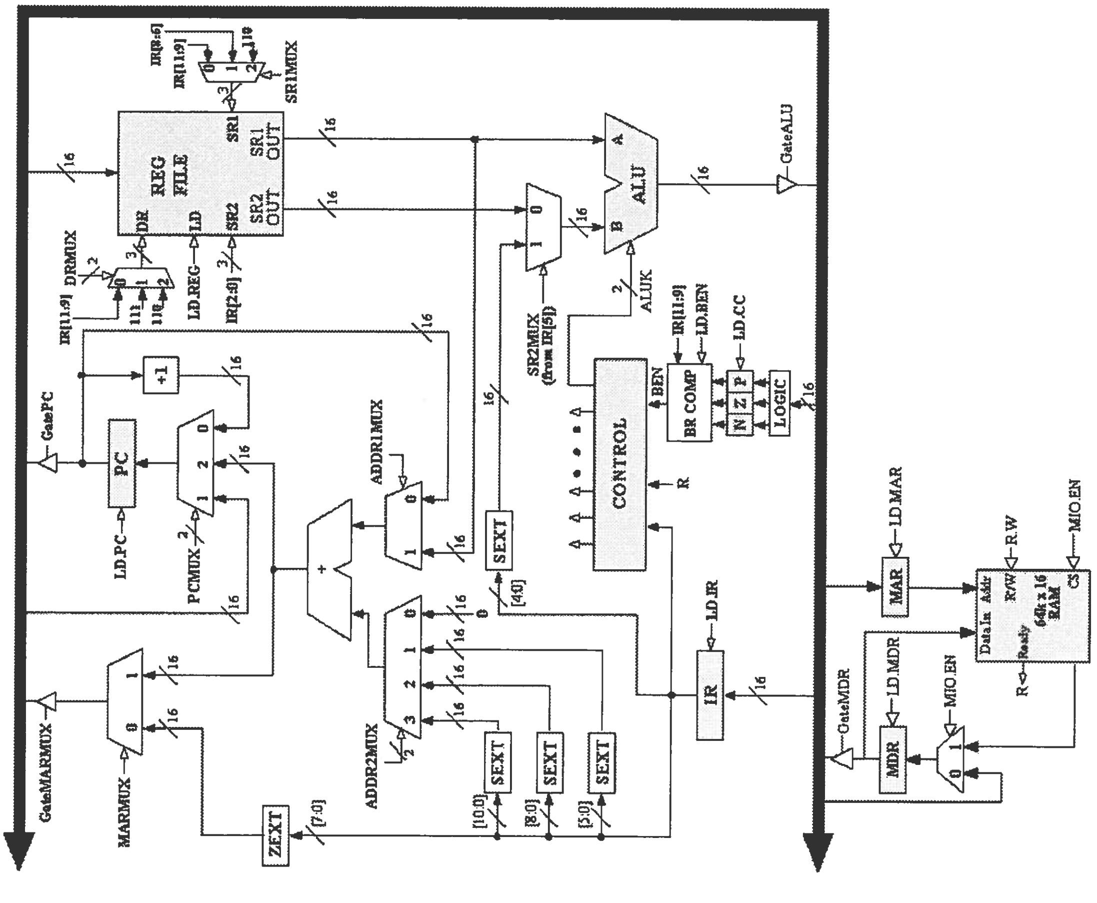
Instruction	Format	Syntax	RTL
LD	0010 | DR | PCoffset9	LD DR, *PCoffset9*	DR ← M[PC + SEXT(PCoffset9)], Setcc
LDI	1010 | DR | PCoffset9	LDI DR, *PCoffset9*	DR ← M[M[PC + SEXT(PCoffset9)]], Setcc
LDR	0110 | DR | BaseR | offset6	LDR DR, BaseR, *offset6*	DR ← M[BaseR + SEXT(offset6)], Setcc
LEA	1110 | DR | 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 SR1 OUT
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
SR2 OUT
DR
LD
DRMUX
IR[11:9]
111
110
LD.REG
IR[2:0]
16
3
2
ALU
A
B
GateALU
SR2MUX (from IR[5])
ALUK
IR[11:9]
LD.BEN
LD.CC
BEN
BR COMP
N
Z
P
LOGIC
CONTROL
R
GatePC
PC
+1
LD.PC
PCMUX
ADDR1MUX
SEXT
[4:0]
ADDR2MUX
SEXT
SEXT
SEXT
[10:0]
[8:0]
[5:0]
ZEXT
[7:0]
MARMUX
GateMARMUX
IR
LD.IR
MAR
LD.MAR
R.W
MIO.EN
Addr
DataIn
R/W
64K x 16 RAM
CS
Ready
R
MDR
LD.MDR
GateMDR
MIO.EN