

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

EE 306, Fall 2017

Yale Patt, Instructor

Stephen Pruet, Siavash Zangeneh, Aniket Deshmukh, Zachary Susskind, Meiling Tang, Jiahan Liu

Exam 1, October 18, 2017

Name: Solutions

Problem 1 (20 points): _____

Problem 2 (15 points): _____

Problem 3 (20 points): _____

Problem 4 (20 points): _____

Problem 5 (25 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.

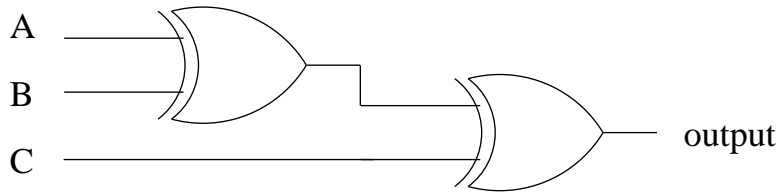
Signature

GOOD LUCK!

Name: Solutions

Problem 1. (20 points):

Part a. (5 points): The following logic circuits consists of two exclusive-OR gates. Construct the output truth table.



| A | B | C | output |
|---|---|---|--------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

Part b. (5 points): After these two instructions execute:

x3030 0001 000 001 0 00 010

x3031 0000 011 000000111

the next instruction to execute will be the instruction at x3039 if:

R1+R2 is zero or positive

Please be specific, but NOT unnecessarily wordy.

Name: Solutions

Part c. (5 points): We wish to know if R0 is being used as the Base Register for computing the address in an LDR instruction. Since the instruction is in memory, we can load it into R4. And, since the Base Register is identified in bits 8:6 of the instruction, we can Load R5 with 0000000111000000, and then execute AND R6,R5,R4. We would know that R0 is the base register if:

The result of AND is 0

Part d. (5 points): Three instructions all construct an address by sign-extending the low 9 bits of the instruction and adding it to the incremented PC.

| | | | | |
|----------------------------|--|-------------|-----|-------------|
| The Conditional Branch | <table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 2px;">0000</td> <td style="border: 1px solid black; padding: 2px;">111</td> <td style="border: 1px solid black; padding: 2px;">xxxxxxxxxxx</td> </tr> </table> | 0000 | 111 | xxxxxxxxxxx |
| 0000 | 111 | xxxxxxxxxxx | | |
| The Load Effective Address | <table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 2px;">1110</td> <td style="border: 1px solid black; padding: 2px;">111</td> <td style="border: 1px solid black; padding: 2px;">xxxxxxxxxxx</td> </tr> </table> | 1110 | 111 | xxxxxxxxxxx |
| 1110 | 111 | xxxxxxxxxxx | | |
| The LD Instruction | <table style="border-collapse: collapse; width: 100%;"> <tr> <td style="border: 1px solid black; padding: 2px;">0010</td> <td style="border: 1px solid black; padding: 2px;">111</td> <td style="border: 1px solid black; padding: 2px;">xxxxxxxxxxx</td> </tr> </table> | 0010 | 111 | xxxxxxxxxxx |
| 0010 | 111 | xxxxxxxxxxx | | |

The xxxxxxxxx represents the 9-bit offset that is sign-extended.

Where does the LC-3 microarchitecture put the result of adding the 9-bit sign-extended offset to the incremented PC?

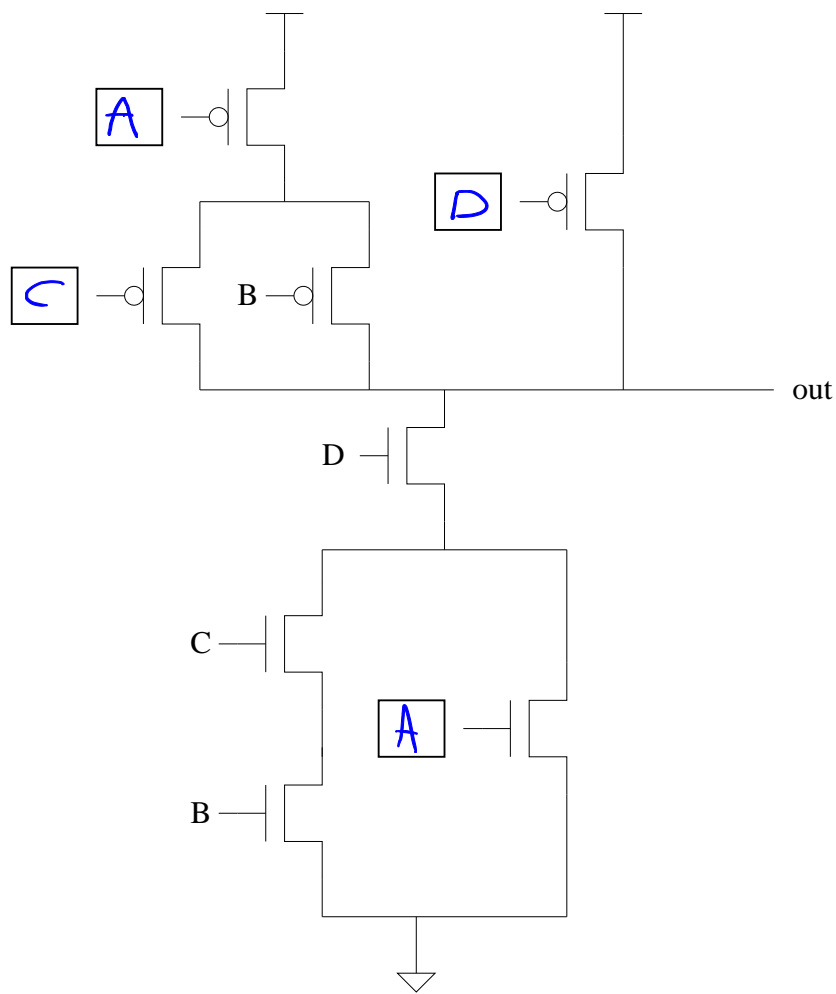
Conditional Branch PC LEA R7 LD MAR

Name: Solutions

Problem 2. (15 points):

Shown below is a transistor circuit, having four inputs (A,B,C,D) and one output (out). Also shown is the truth table for this circuit. The gates of some of the transistors are not labeled, and the outputs of some of the input combinations in the truth table are not shown.

Your job: Complete the transistor diagram by labeling the missing inputs to the gates, and by adding the missing outputs to the truth table. Every input combination produces an output of either 0 or 1. The result will be a transistor diagram and the truth table describing its behavior.



| A | B | C | D | out |
|---|---|---|---|-----|
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 |

Name: Solutions

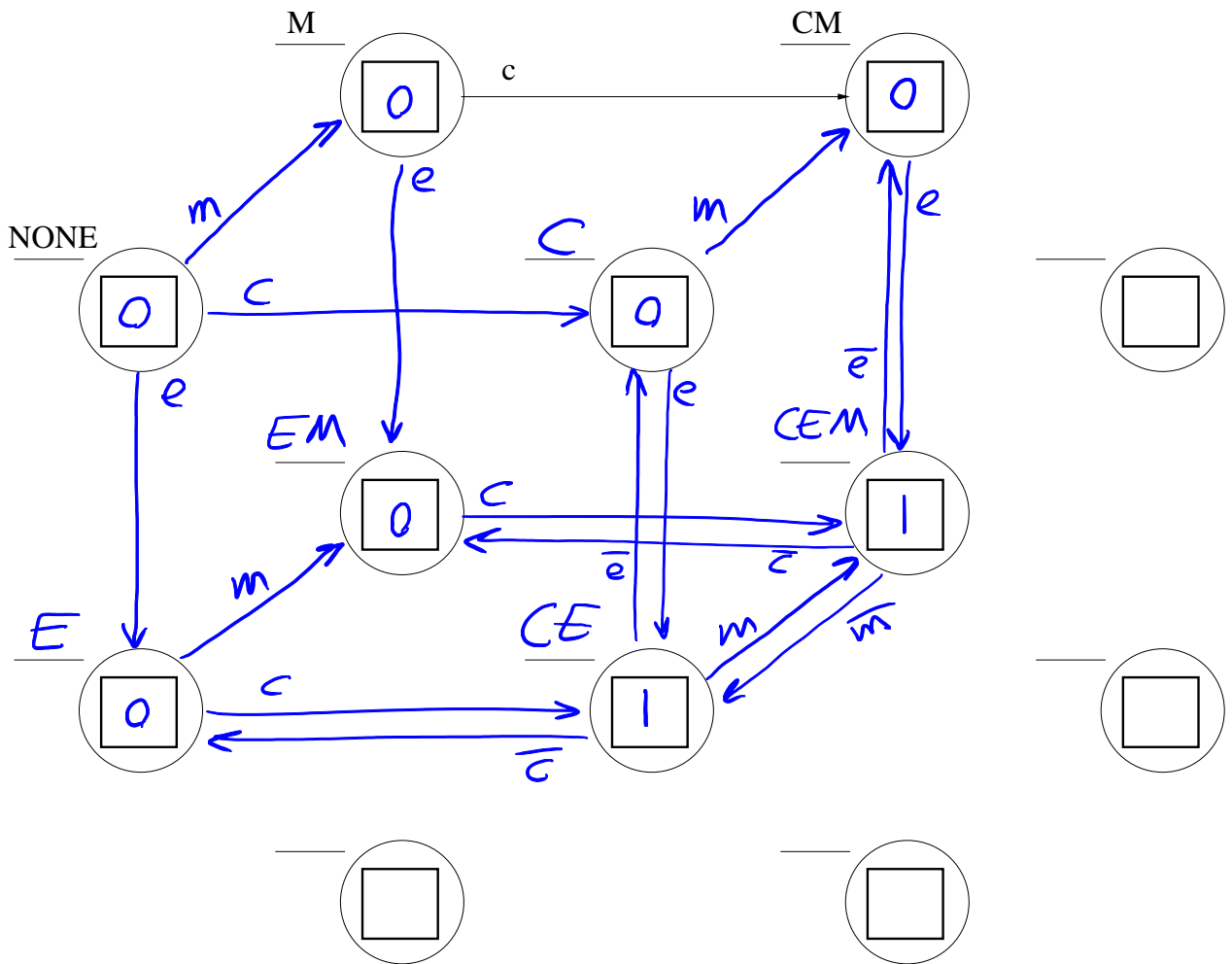
Problem 3. (20 points): You are taking three courses, one each in computing (C), engineering (E), and math (M). In each course, you periodically receive assignments. You never receive more than one assignment at a time. You also never receive another assignment in a course if you currently have an assignment in that course that has not been completed. You must procrastinate (i.e., do nothing) unless you have unfinished assignments in both computing and engineering.

Design a finite state machine to describe the state of the work you have to do and whether you are working or procrastinating.

Part a. (5 points): Label each state with the unfinished assignments (with letters C,E,M) for when you are in that state. There are far more states provided than you actually need. Use only what you need.

Part b. (10 points): There are six inputs: $c, e, m, \bar{c}, \bar{e}, \bar{m}$. c, e, m refer to you receiving an assignment. $\bar{c}, \bar{e}, \bar{m}$ refer to you completing an assignment. Draw the transition arc for each state/input pair. For example, if you had previously only had an unfinished assignment in Math and you received an assignment in computing, you would transition from state M to state CM, as shown below.

Part c. (5 points): The output of each state is your behavior, 1 if you are working on an assignment, 0 if you are procrastinating. Label the outputs of each state.



Name: Solutions

Problem 4. (20 points):

A warehouse is controlled by an electronic lock having an n-digit combination. The electronic lock has ten buttons, labeled 0 to 9 on its face. To open the lock, a user presses a sequence of n buttons. The corresponding ASCII characters get loaded into sequential locations of memory, starting at location x3150. After n buttons have been pressed, the null character x00 is loaded into the next sequential memory location.

The program shown below determines whether or not the lock should open, depending on whether the combination entered agrees with the combination stored in the n memory locations starting at x3100. If the lock should open, the program stores a 1 in location x3050. If the lock should not open, the program stores a 0 in location x3050.

Note that some of the instructions are missing.

Part a. (15 points): Complete the program by filling in the missing instructions.

| | | |
|-------|----------------------|------------------------------------|
| x3000 | 0101 101 101 1 00000 | ; R5 ← x0000 |
| x3001 | 0010 000 000001111 | ; R0 ← M[x3011] |
| x3002 | 0010 001 000001101 | ; R1 ← M[x3010] |
| x3003 | 0110 010 000 000000 | ; R2 ← M[R0] |
| x3004 | 0000 010 00001000 | ; Branch to x300D if Z is set |
| x3005 | 0110 011 001 000000 | ; R3 ← M[R1] |
| x3006 | 1001 011 011 111111 | ; NOT R3 |
| x3007 | 0001 011 011 1 00001 | ; R3 ← R3 + 1 |
| x3008 | 0001 011 011 000 010 | ; R3 ← R3 + R2 |
| x3009 | 0000 101 000000100 | ; Branch to x300E if N or P is set |
| x300A | 0001 000 000 100001 | ; R0 ← R0 + 1 |
| x300B | 0001 001 001 100001 | ; R1 ← R1 + 1 |
| x300C | 0000 111 111110110 | ; Branch always to x3003 |
| x300D | 0001 101 101 100001 | ; R5 ← R5 + 1 |
| x300E | 0011 101 001000001 | ; Store R5 in x3050 |
| x300F | 1111 0000 0010 0101 | ; HALT |
| x3010 | 0011 0001 0000 0000 | ; x3100 |
| x3011 | 0011 0001 0101 0000 | ; x3150 |

Part b. (5 points): A simple change to the contents of memory will allow us to eliminate the instructions at memory locations x3006 and x3007 in our program. What is the change?

Store the negatives of target combination in memory locations starting at x3100

Name: Solutions

Problem 5. (25 points):

Part a. (15 points): The PC is loaded with x3000, and the instruction at address x3000 is executed. In fact, execution continues and four more instructions are executed. The table below contains the contents of various registers at the end of execution for each of the five (total) instructions.

Your job: complete the table.

| | PC | MAR | MDR | IR | R0 | R1 |
|--------------------------|-------|-------|-------|-------|-------|-------|
| Before execution starts | x3000 | — | — | — | x0000 | x0000 |
| After the first finishes | x3001 | x3006 | xB333 | x2005 | xB333 | x0000 |
| After the 2nd finishes | x3002 | x3001 | x0601 | x0601 | xB333 | x0000 |
| After the 3rd finishes | x3003 | x3002 | x1261 | x1261 | xB333 | x0001 |
| After the 4th finishes | x3004 | x3003 | x1000 | x1000 | x6666 | x0001 |
| After the 5th finishes | x3001 | x3004 | x0BFC | x0BFC | x6666 | x0001 |

Part b. (10 points): Let's start execution again, starting with PC = x3000. First, we re-initialize R0 and R1 to 0, and set a breakpoint at x3004. We press RUN eleven times, each time the program executes until the breakpoint. What are the final values of R0 and R1?

R0 R1

Description of program

x3000: load Mem[x3006] = xB333 into R0

x3001: skip next instruction if number is not negative

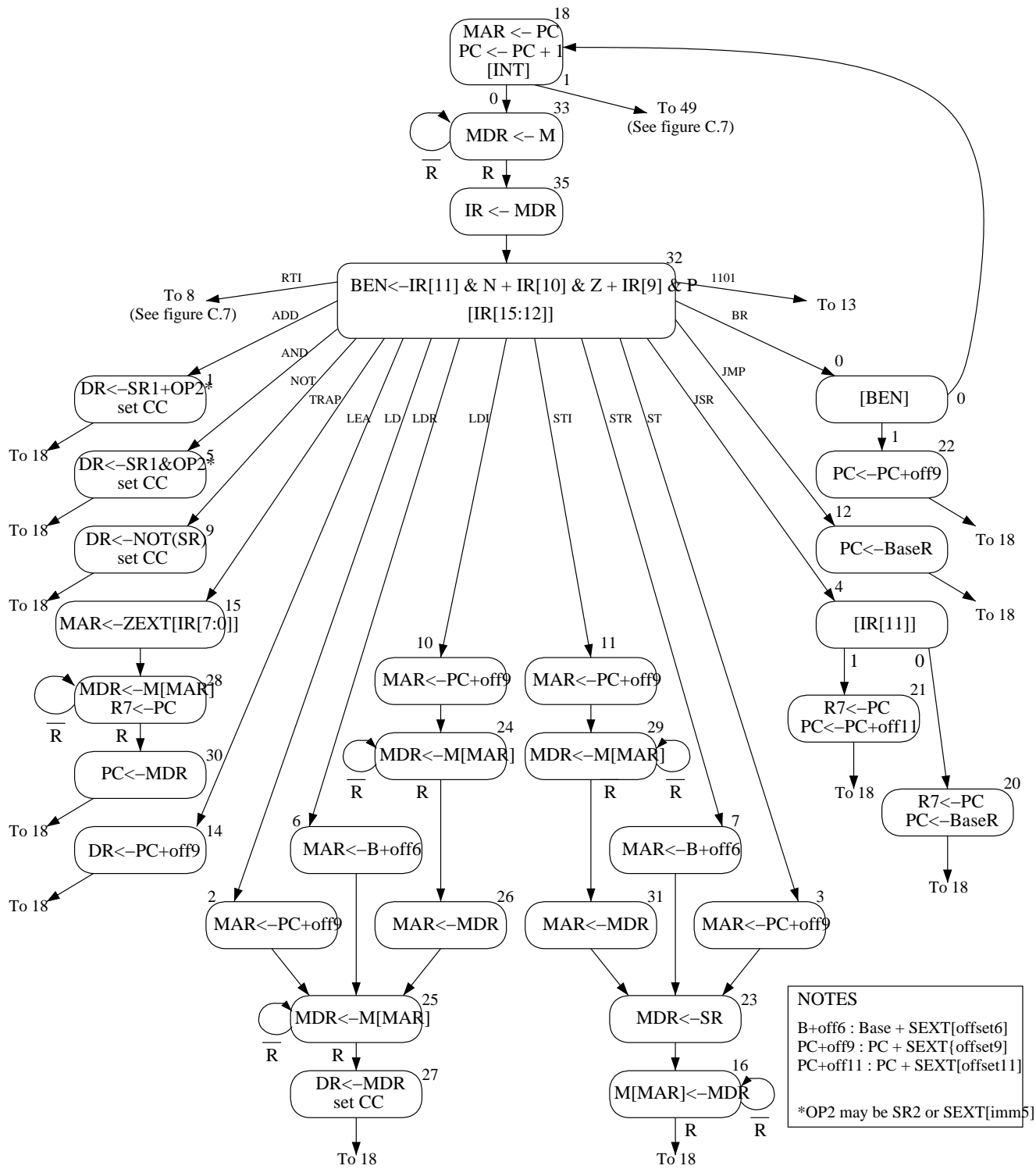
x3002: ADD 1 to R1 0001 001 001 10000 | x1261

x3003: left shift R0 by adding it to itself
0001 000 000 000 000

x3004: if R0 is not zero, branch back to x3001

Part b) After 11 iterations of executing the loop, R0 is shifted 11 times and R1 contains the number of 1s in the most significant 11 bits,

xB333 = 1011 0011 0011 0011
these 6



| | 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

Table 2.2 ASCII character set

| Oct | Dec | Hex | Char | Oct | Dec | Hex | Char | Oct | Dec | Hex | Char | Oct | Dec | Hex | Char |
|-----|-----|-----|------------|-----|-----|-----|--------------|-----|-----|-----|------|-----|-----|-----|------------|
| 000 | 0 | 00 | <i>NUL</i> | 040 | 32 | 20 | <i>SPACE</i> | 100 | 64 | 40 | @ | 140 | 96 | 60 | ` |
| 001 | 1 | 01 | <i>SOH</i> | 041 | 33 | 21 | ! | 101 | 65 | 41 | A | 141 | 97 | 61 | a |
| 002 | 2 | 02 | <i>STX</i> | 042 | 34 | 22 | " | 102 | 66 | 42 | B | 142 | 98 | 62 | b |
| 003 | 3 | 03 | <i>ETX</i> | 043 | 35 | 23 | # | 103 | 67 | 43 | C | 143 | 99 | 63 | c |
| 004 | 4 | 04 | <i>EOT</i> | 044 | 36 | 24 | \$ | 104 | 68 | 44 | D | 144 | 100 | 64 | d |
| 005 | 5 | 05 | <i>ENQ</i> | 045 | 37 | 25 | % | 105 | 69 | 45 | E | 145 | 101 | 65 | e |
| 006 | 6 | 06 | <i>ACK</i> | 046 | 38 | 26 | & | 106 | 70 | 46 | F | 146 | 102 | 66 | f |
| 007 | 7 | 07 | <i>BEL</i> | 047 | 39 | 27 | ' | 107 | 71 | 47 | G | 147 | 103 | 67 | g |
| 010 | 8 | 08 | <i>BS</i> | 050 | 40 | 28 | (| 110 | 72 | 48 | H | 150 | 104 | 68 | h |
| 011 | 9 | 09 | <i>HT</i> | 051 | 41 | 29 |) | 111 | 73 | 49 | I | 151 | 105 | 69 | i |
| 012 | 10 | 0A | <i>LF</i> | 052 | 42 | 2A | * | 112 | 74 | 4A | J | 152 | 106 | 6A | j |
| 013 | 11 | 0B | <i>VT</i> | 053 | 43 | 2B | + | 113 | 75 | 4B | K | 153 | 107 | 6B | k |
| 014 | 12 | 0C | <i>FF</i> | 054 | 44 | 2C | , | 114 | 76 | 4C | L | 154 | 108 | 6C | l |
| 015 | 13 | 0D | <i>CR</i> | 055 | 45 | 2D | - | 115 | 77 | 4D | M | 155 | 109 | 6D | m |
| 016 | 14 | 0E | <i>SO</i> | 056 | 46 | 2E | . | 116 | 78 | 4E | N | 156 | 110 | 6E | n |
| 017 | 15 | 0F | <i>SI</i> | 057 | 47 | 2F | / | 117 | 79 | 4F | O | 157 | 111 | 6F | o |
| 020 | 16 | 10 | <i>DLE</i> | 060 | 48 | 30 | 0 | 120 | 80 | 50 | P | 160 | 112 | 70 | p |
| 021 | 17 | 11 | <i>DC1</i> | 061 | 49 | 31 | 1 | 121 | 81 | 51 | Q | 161 | 113 | 71 | q |
| 022 | 18 | 12 | <i>DC2</i> | 062 | 50 | 32 | 2 | 122 | 82 | 52 | R | 162 | 114 | 72 | r |
| 023 | 19 | 13 | <i>DC3</i> | 063 | 51 | 33 | 3 | 123 | 83 | 53 | S | 163 | 115 | 73 | s |
| 024 | 20 | 14 | <i>DC4</i> | 064 | 52 | 34 | 4 | 124 | 84 | 54 | T | 164 | 116 | 74 | t |
| 025 | 21 | 15 | <i>NAK</i> | 065 | 53 | 35 | 5 | 125 | 85 | 55 | U | 165 | 117 | 75 | u |
| 026 | 22 | 16 | <i>SYN</i> | 066 | 54 | 36 | 6 | 126 | 86 | 56 | V | 166 | 118 | 76 | v |
| 027 | 23 | 17 | <i>ETB</i> | 067 | 55 | 37 | 7 | 127 | 87 | 57 | W | 167 | 119 | 77 | w |
| 030 | 24 | 18 | <i>CAN</i> | 070 | 56 | 38 | 8 | 130 | 88 | 58 | X | 170 | 120 | 78 | x |
| 031 | 25 | 19 | <i>EM</i> | 071 | 57 | 39 | 9 | 131 | 89 | 59 | Y | 171 | 121 | 79 | y |
| 032 | 26 | 1A | <i>SUB</i> | 072 | 58 | 3A | : | 132 | 90 | 5A | Z | 172 | 122 | 7A | z |
| 033 | 27 | 1B | <i>ESC</i> | 073 | 59 | 3B | ; | 133 | 91 | 5B | [| 173 | 123 | 7B | { |
| 034 | 28 | 1C | <i>FS</i> | 074 | 60 | 3C | < | 134 | 92 | 5C | \ | 174 | 124 | 7C | |
| 035 | 29 | 1D | <i>GS</i> | 075 | 61 | 3D | = | 135 | 93 | 5D |] | 175 | 125 | 7D | } |
| 036 | 30 | 1E | <i>RS</i> | 076 | 62 | 3E | > | 136 | 94 | 5E | ^ | 176 | 126 | 7E | ~ |
| 037 | 31 | 1F | <i>US</i> | 077 | 63 | 3F | ? | 137 | 95 | 5F | _ | 177 | 127 | 7F | <i>DEL</i> |