**Chapter 5**

**5.4** Say we have a memory consisting of 256 locations, and each location contains 16 bits.

(a) How many bits are required for the address?

(b) If we use the PC-relative addressing mode, and want to allow control transfer between instructions 20 locations away, how many bits of a branch instruction are needed to specify the PC-relative offset?

(c) If a control instruction is in location 3, what is the PC-relative offset of address 10? Assume that the control transfer instructions work the same way as in the LC-3.

**5.13** (a) Write an LC-3 instruction that clears the contents of R0.

(b) The LC-3 has no subtract instruction. How could one perform the following operation using only three LC-3 instructions:

R1 ← R1 – R3

(c) Using only one LC\_3 instruction and without changing the content of any register, how might one set the condition codes based on the value that resides in R1?

(d) Is there a sequence of LC-3 instructions that will cause the condition codes at the end of the sequence to be N = 1, Z = 1, and P = 0.

(e) How might one use a single LC-3 instruction to move the value in R2 into R4?

**5.15**  State the contents of R1, R2, R3, and R4 after the program starting at location x3100 halts.

|  |  |
| --- | --- |
| Address | Data |
| 0011 0001 0000 0000 | 1110 001 000100000 |
| 0011 0001 0000 0001 | 0010 010 000100000 |
| 0011 0001 0000 0010 | 1010 011 000100000 |
| 0011 0001 0000 0011 | 0110 100 010 000001 |
| 0011 0001 0000 0100 | 1111 0000 0010 0101 |
| .. | .. |
| .. | .. |
| 0011 0001 0010 0010 | 0100 0101 0110 0110 |
| 0011 0001 0010 0011 | 0100 0101 0110 0111 |
| .. | .. |
| .. | .. |
| 0100 0101 0110 0111 | 1010 1011 1100 1101 |
| 0100 0101 0110 1000 | 1111 1110 1101 0011 |

**5.23** Suppose the following LC-3 program is loaded into memory staring at location x30FF:

x30FF 1110 0010 0000 0001

x3100 0110 0100 0100 0010

x3101 1111 0000 0010 0101

x3102 0001 0100 0100 0001

x3103 0001 0100 1000 0010

If the program is executed, what is the value in R2 at the end of execution?

**5.31** The following figure shows a snapshot of the 8 registers of the LC-3 before and after the instruction at location x1000 is executed. Fill in the bits of the instruction at location x1000.

|  |  |  |  |
| --- | --- | --- | --- |
|  | BEFORE |  | AFTER |
| R0 | x0000 | R0 | x0000 |
| R1 | x1111 | R1 | x1111 |
| R2 | x2222 | R2 | x2222 |
| R3 | x3333 | R3 | x3333 |
| R4 | x4444 | R4 | x4444 |
| R5 | x5555 | R5 | xFFF8 |
| R6 | x6666 | R6 | x6666 |
| R7 | x7777 | R7 | x7777 |

0x1000:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 |  |  |  |  |  |  |  |  |  |  |  |  |

**Chapter 6**

**6.4** Write a short LC-3 program that compares the two numbers in R1 and R2 and puts the larger number in R1 and the smaller one in R2.

(Must load 3102-3105 with x1, xA, x64, and x3E8 prior to running this program)

.ORIG x3000

LD R1, x0FF

LD R2, x0FF

LEA R3, x0FF

ADD R6, R1, #0

ADD R7, R2, #0

LDR R4, R3, #0

AND R6, R6, R4

AND R7, R7, R4

ADD R6, R6, #0

BRz #7

ADD R7, R7, #0

BRz #2

ADD R3, R3, #1

BRnzp #-9

ADD R5, R1, #0

ADD R1, R2, #0

ADD R2, R5, #0

ADD R5, R2, #0

ADD R2, R1, #0

ADD R1, R5, #0

HALT

.END

**6.7** What does the following LC-3 program do?

This program adds an array of numbers together.

|  |  |
| --- | --- |
| x3001 | 1110 0000 0000 1100 |
| x3002 | 1110 0010 0001 0000 |
| x3003 | 0101 0100 1010 0000 |
| x3004 | 0010 0100 0001 0011 |
| x3005 | 0110 0110 0000 0000 |
| x3006 | 0110 1000 0100 0000 |
| x3007 | 0001 0110 1100 0100 |
| x3008 | 0111 0110 0000 0000 |
| x3009 | 0001 0000 0010 0001 |
| X300A | 0001 0010 0110 0001 |
| X300B | 0001 0100 1011 1111 |
| X300C | 0000 0011 1111 1000 |
| X300D | 1111 0000 0010 0101 |
| X300E | 0000 0000 0000 0101 |
| X300F | 0000 0000 0000 0100 |
| X3010 | 0000 0000 0000 0011 |
| X3011 | 0000 0000 0000 0110 |
| X3012 | 0000 0000 0000 0010 |
| X3013 | 0000 0000 0000 0100 |
| X3014 | 0000 0000 0000 0111 |
| X3015 | 0000 0000 0000 0110 |
| X3016 | 0000 0000 0000 1000 |
| X3017 | 0000 0000 0000 0111 |
| X3018 | 0000 0000 0000 0101 |

**6.13** Notice that we can shift a number to the left by one bit position by adding it to itself. For example, when the binary number 0011 is added to itself, the result is 0110. Shifting a number one bit pattern to the right is not as easy. Devise a routine in LC-3 machine code to shift the contents of memory location x3100 to the right by three bits.