ECE 120: Introduction to Computing Spring 2019 ZJUI

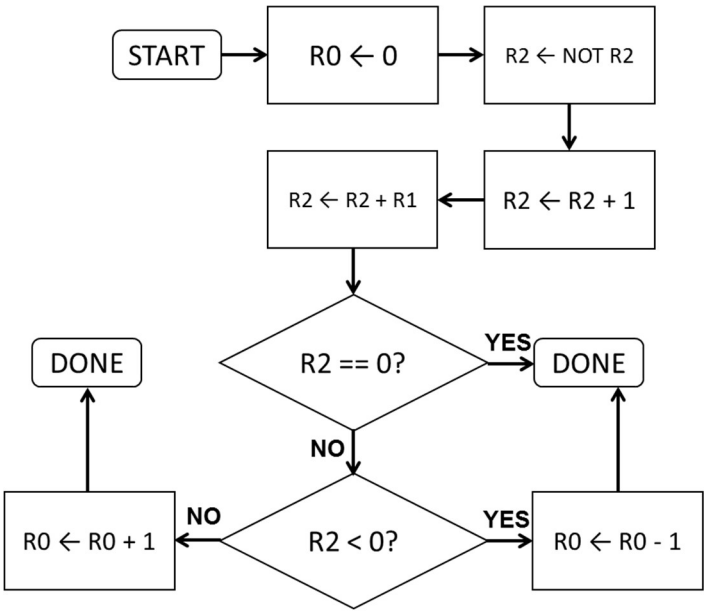
**due at start of section on Friday 17 May 2019**

# Homework 12: LC-3 Programming

**1. LC-3 Programming**

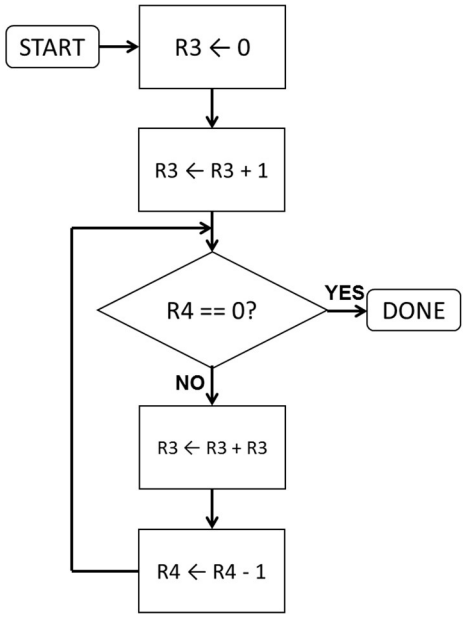
For each of the following tasks, systematically decompose the problem to the level of LC-3 instructions, then write LC-3 instructions to implement your solution. Turn in your flow chart and LC-3 instructions in binary. For credit, each instruction must be annotated with a comment in RTL or assembly.

a. Problem 6.4 from Patt and Patel. You may only change the contents of R0, R1, and R2.



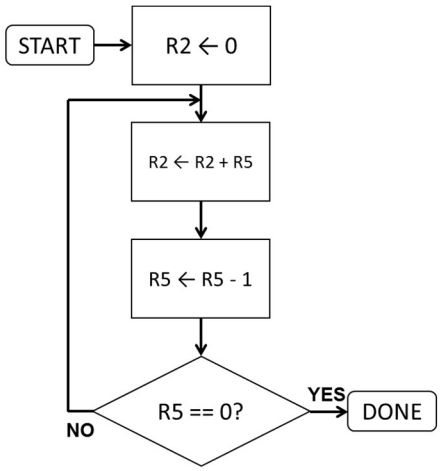

```
0101 000 000 1 00000  ; AND R0, R0, #0
1001 010 010 111111   ; NOT R2, R2
0001 010 010 1 00001  ; ADD R2, R2, #1
0001 010 010 0 00 001 ; ADD R2, R2, R1
0000 010 000000100    ; BRz #4
0000 001 000000010    ; BRp #2
0001 000 000 1 11111  ; ADD R0, R0, #-1
0000 111 000000001    ; BRnzp #1
0001 000 000 1 00001  ; ADD R0, R0, #1
```

b. Assuming that $0 \le R4 < 15$, put the 2's complement value $2^{R4}$ into R3. You may only change the contents of R3 and R4.




```
0101 011 011 1 00000  ; AND R3, R3, #0
0001 011 011 1 00001  ; ADD R3, R3, #1
0001 100 100 1 00000  ; ADD R4, R4, #0
0000 101 000000011    ; BRnp #3
0001 011 011 0 00 011 ; ADD R3, R3, R3
0001 100 100 1 11111  ; ADD R4, R4, #-1
0000 111 111111100    ; BRnzp #-4
```

c. Assuming that $R5 \ge 1$, compute the sum of integers from 1 to R5 and store the result in R2. You may only change the contents of R2 and R5.




```
0101 010 010 1 00000  ; AND R2, R2, #0
0001 010 010 0 00 101 ; ADD R2, R2, R5
0001 101 101 1 11111  ; ADD R5, R5, #1
0000 101 111111101    ; BRnp #-3
```

**2. LC-3 Program Execution**

Do problem 6.16 from Patt and Patel. Stop your execution trace when the PC reaches x3003, and do not fill in row x3003 of the table in your solution.

**All students have to do is fill in the table!** The following is an explanation of the final table.

| | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| x3000 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| x3001 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 |
| x3002 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| x3003 | (leave this row blank) | | | | | | | | | | | | | | | |
| x3004 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| x3005 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| x3006 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

The first MAR value reflects the instruction fetch from x3000. That instruction has opcode 0010, LD. The next MAR value must then be the address of the load. When the LD executes, PC has already been incremented (in fetch), so PC = x3001. The load address is x3005. Thus the LD instruction must use offset x3005 – x3001 = x0004 (sign-extended from the 9-bit offset in the instruction). The LD instruction's destination register (given) is R0. Looking at x3005, we see that R0 thus becomes x0030.

The next MAR value is x3001, which represents the instruction fetch from x3001. That instruction is an ADD (opcode 0001). Decoding it, we see that the ADD adds 1 to R0 and stores the result back into R0, so R0 now has x0031.

The next MAR value is x3002, which represents the instruction fetch from x3002. That instruction is an STI (opcode 1011). The next MAR value must be the address from which the address is obtained. When the STI executes, PC has already been incremented (in fetch), so PC = x3003. The address used is x3006. So the STI offset must be x3006 – x3003 = x0003 (sign-extended from the 9-bit offset in the instruction). The next MAR value (x4001) must be the bits stored in M[x3006], which is also where the STI instruction writes the value of R0 (the STI's SR). So M[x4001] now contains x0031 (from R0). And we can write x4001 into the table at location x3006.

The last instruction, at x3003, executes TRAP x21, but how this happens is a topic for ECE220. Read Ch. 9 of Patt and Patel if you want to understand it sooner.

**3. Analyzing LC-3 Code**

Do problem 7.16 from Patt and Patel.

Counts the number of even values in the sequence and stores the count in R4, and counts the number of odd values in the sequence and stores the count in R3.

4. **Data or Computation?**
Do problem 7.20 from Patt and Patel.

The programmer of module (a) uses the LC-3 processor to place the desired value in memory, whereas the programmer of module (b) includes the value as directly as part of the program image. Approach (a) generally gives a smaller but slower program, while (b) gives a larger but faster program. (The specific cases here are trivial, of course.)

5. **Buggy LC-3 Programs**
The code below is intended to divide R2 (R2 ≥ 0) by R4 (R4 > 0), leaving the quotient in R3 and the remainder in R1. Unfortunately, the code has a bug. Fix the code **by adding one new instruction.** ** For credit, do not rewrite the code in any other way. **

The author forgot to copy the difference back into R2 after calculating it and storing it in R0.

```
        .ORIG x3000

        ; registers R2 and R4 must be initialized first!
START   AND R3,R3,#0
        NOT R4,R4
        ADD R4,R4,#1
LOOP    ADD R0,R2,R4
        BRn BELOW
        ADD R3,R3,#1
        ADD R2,R0,#0 ; copy difference back into R2
        BRnzp LOOP
BELOW   ADD R1,R2,#0
        ; R1 and R3 values are correct here (HALT may change them
        ; in the simulator)
        HALT

        .END
```

**6. Comparing Algorithms**

Two friends are trying to print a line of periods bounded by asterisks to the console. The line is supposed to have N – 2 periods, where N ≥ 2. For example, when N = 8, they want to print, "***** . . . . . *****" to the console (without quotes).

Each friend has decomposed the task, but they have different results. Look at the two approaches, as represented by the flow charts below.

a. Explain which approach is the better of the two and why you believe it to be better.

The approach on the right executes more quickly than the one on the left. In the left approach, the checks for which character to produce are performed for each loop iteration, whereas on the right, these checks are implied by the use of a sequential decomposition, leaving the loop iterations much simpler.

b. Imagine that one must replace each box labeled "print '*'" with a complex algorithm requiring almost 1,000 LC-3 instructions to implement. Repeat your comparison between the two approaches after the replacement described, again explaining why you believe your choice to be the better one.

If we are required to replicate a large body of code, the approach on the left becomes better than the one on the right, since the left approach contains only one such copy, whereas the right contains two. In addition to increasing code size, replicating large chunks of code is error-prone (and replicates any bugs in the code!). In practice, we use subroutines (a technique that you will learn in ECE220) to enable our program to reuse such code (which becomes an abstraction layer!). If the new code is made into a subroutine, we can still get the benefits of the approach on the right without the drawbacks of replicating a large chunk of code.