

One problem in bioinformatics is to see which of a given set of patterns of DNA bases occurs in one long DNA sequence. Each strand of DNA is a sequence of chemical units called bases which come in 4 varieties, labelled A, G, T and C. The pattern occurs when it appears contiguously as a subsequence in the long sequence.

We will address a simplified problem in which there are only two bases (0 and 1) and the long sequence is a circular ring of 16 bases.

Another simplification, which makes this problem amenable as the first LC-3 programming project, affects what it means for a pattern to occur:

The given long sequence L contains the pattern P if and only if after you rotate L left a number of times r , $0 \leq r \leq 15$, and then shift it right a number of times s , $0 \leq s \leq 15$, you get a result that equals the pattern P .

The number of times the pattern occurs is the number of combinations of r and s for which rotating L by r bits followed by a right shift of s bits gives a result equal to P . (Therefore, some patterns will be counted multiple times when L contains subsequences of 0's.) This way patterns P and sequences L can always be 16-bit bit strings; you do NOT have to deal with any bit string length beside 16.

The LC-3 machine language program that you will finish for Project 4 should operate as follows:

- It will start at address 0x3000.
- It will use the long sequence that the user must store at address 0x30FF.
- The user must store the patterns at even addresses 0x3100, 0x3102, 0x3104, etc. Each memory location (with odd address) just after each pattern must contain 0 initially.
- The data at the even address after the last pattern must be 0 to mark it as the last pattern.

When the program is started as above, it must calculate, and store in memory after each pattern, the number of times that pattern occurs within the given long sequence, where that sequence is considered to be circular, i.e., a ring. After storing each number after each pattern, it must halt the LC-3 by executing the HALT instruction.

1 Learning Objectives and Project Plan

The actual objective is not to do some real bioinformatics. The real objectives are (1) To practice problem solving within situations where the kind of details that occur in our discipline occur. (2) Become familiar with the low-level operations computers do. (3) Learn to identify and then solve small sub-problems that will be used or combined to solve bigger problems. (4) Learn and practice how to complete the solution of a bigger problem by combining the smaller solutions so they work together.

Project 3 is to program and test sequences of LC-3 instructions to do things that are useful for the big problem but are not provided by single LC-3 instructions. It emphasizes the idea of implementing new operations by combining fixed given operations. This is the main idea to

demonstrate that a computing model is “universal,” as discussed by PP. This is also the idea behind “procedural abstraction” as done by programmers.

Project 4 is to utilize the operations that you implemented in project 3 within a program to solve our simplified DNA matching problem. The hard (interesting) part is planning how to make the computer try all the possibilities systematically, count all the ways a match occurs, and program it using machine language. Planning tip, ask: How many nested loops, and what are they for?

We will introduce **function linkage** by teaching how to code function bodies and function calls in machine language. It will be important to carefully record in writing how each piece uses the LC-3 registers, so you can program each piece not to interfere with the other pieces.

2 Project 3

Study the above to understand what we mean by the sequence L containing pattern P . Try it on examples until you are sure of what we mean!

Project 3 includes implementing the operations to (1) rotate left the bits of a 16-bit LC-3 word a variable number of times, (2) shift right those bits a variable number of times and (3) test if two 16-bit words are equal. The LC-3 ISA does not have instructions for these operations!

All right shifts are “logical” which means that the new bits shifted in from the left are always 0. (In an “arithmetic” shift, the new bits are copies of the sign bit.)

Although shifting right by even a single bit is in the LC-3 ISA, shifting left by one bit position can be done with the ADD instruction. You figure out how (it’s easy because $1+1=2$, $2+2=4$, $4+4=8$, etc.)

Once you have a way to shift left by one bit, you can implement shift left by a variable number of bits with a loop. There are plenty of examples of loops in PP’s chapters on LC-3 machine language programming and program design.

Rotating left is a problem because the top bit gets lost after a pure left shift. Therefore, your solution must keep track of the original top bit so a copy of it (0 or 1) can be added after each shift. Note that many instructions set the N condition code, the N condition code can be tested before the shift, and the test result can be used to choose the data (0 or 1) to be added later. (It helps to be aware of what happens when 1 is added to an even number, and be aware that even numbers have 0 in the low bit.)

Once you can make the LC-3 do left rotation, you can make it do right rotation by doing left rotation by a (usually) different amount. That amount can be computed by subtraction. (You will have to learn how to make the LC-3 subtract given its repertoire of ADD and bitwise NOT only.)

But we still have to do right shift! Left or right rotation can be used to put the bits that should be shifted into their destination positions. However the bits to their left will not always be zero. Another left and then right shift will take care of them! (What do I mean by “take care of” in this context?) You might implement a faster strategy that uses 15 addressible constants in memory and the AND instruction.

Your job: Write a series of instructions to load relevant registers with test data and do test calls to the functions you will write. Lay out a gap after the test calls and write as functions the LC-3 code sequences for the tasks listed below. (A template will be provided.)

In all cases, the input number(s) should be in registers and the result should be in a register. The code sequences should be written in an ascii text file using 16 0 or 1s at the beginning of each

line to represent a value in binary. After **EACH** 0-1 sequence, there must be a semicolon and then a **comment** that explains each instruction.

1. Subtract two numbers.
2. Test if two numbers are equal and store a 1 in a register if they are and a 0 in the same register otherwise.
3. Shift a number one bit to the left.
4. Shift a number one bit to the left the number of times given in a register.
5. Rotate a number one bit to the left.
6. Rotate a number one bit to the left the number of times given in a register.
7. Rotate a number one bit to the right the number of times given in a register.
8. Shift a number to the right the number of times given in a register.

Very important: Preceding the code for each operation, there **MUST** be lines of comments without code that:

1. Document the operation (by repeating or rewriting our explanation);
2. EXPLAIN which registers the operation uses for input and how;
3. EXPLAIN which register will hold the result;
4. and WARN people about which registers and/or memory locations will be messed up by the code.

YOU will have to invent a scheme to systematically choose which registers are used for each operation's input and output! **DOCUMENT** that scheme **AND** each operation. You **WILL** be sorry otherwise: (1) You will be awfully confused about which register to use and which will be messed up when you try to write the Project 4 program. (2) Errors will occur when you try to use one operation within other operations without correctly heeding register usage. (3) We will give undocumented operation an almost **ZERO** grade since we won't be able to easily know which registers to use for test data and to see the results (we won't try).

Each operation must be implemented as a LC-3 function. Making a function return is easy: Don't use R7 (or save it in memory before use) because the return address is put there by the JSR instruction, and use the RET operation at the end, which jumps to the address given in R7. (If R7 had been saved, use a LD instruction to copy the saved address back to R7 before the RET.)

To call the function is also easy: Make sure R7 is not needed and then code a JSR instruction (beginning with 01001). You must calculate the "offset" so when the offset is added to the PC value pointing to the next instruction (the one just after the JSR) the result is the address of the first instruction of the function.

The first line of your programs must specify the memory address of the first instruction of your program. The LC-3 simulator will copy your program into its simulated memory starting at that address. For this assignment, you should place your program starting at x3000. The first non-comment line of your program should be

0011000000000000 ; Load address for the code and data below

If you are using a Windows machine, use the LC3Edit program to type in your programs and/or convert them to binary files that the LC-3 simulator can load. Consult the TA or instructor in the lab for information about using other systems for this project.

3 Project 3 Due Sept 25 (1 week)

The on-time due time is 11:00 PM, Sept. 25. Submit the ACSII file (with the comments) to Project 3 via WebCT. Late submissions will be penalized at the rate of about 17% per day for the next 72 hours, so the maximum penalty will be 50%.

Tip: Do the pieces one by one as you read Chapters 4, 5 and 6 in PP to learn machine language programming and LC-3 details. Systematically save separately each version of your work in which you solved something successfully.

4 Project 4 Due Oct 2 (2 weeks)

The on-time due time is 11:00 PM, Sept. 2. Submit the ACSII file (with the comments) to Project 4 via WebCT. Late submissions will be penalized at the rate of about 17% per day for the next 72 hours, so the maximum penalty will be 50%.

The program should start at 0x3000. It will be tested by us putting some test data in the LC-3 simulator beginning at address 0x30ff for the long sequence and 0x3100, 0x3102, etc. for the patterns. We will then start the simulator at 0x3000 and watch for the correct answers appearing at addresses 0x3101, 0x3103, etc.

Please address any questions not addressed by the material above to me or the TA early enough so you know what you need to complete the project on time, even if there are some unexpected debugging delays!