CompSci 210 Computer Systems 1 Assignment 2

Due Date: 5PM, Friday, Apr 30, 2010

A. Write an LC-3 assembly language programme that reads in an integer *n* and prints out the number 2^n.

Write an LC-3 assembly language programme that performs the following tasks:

- Prints out your name, UPI, student ID, then the text "CompSci 210 2010 Semester 1 Assignment 2". For example,

  ```
  Ian Chan
  icha052
  AUID 1234567
  CompSci 210 2010 Semester 1
  Assignment 2
  ```

  [Change the name, UPI, and ID to your own].

- Prints a message requesting input of an integer and specifying the maximum permissible size.
- Reads in the line of text, interprets an initial string of digits terminated by white space (spaces, tabs, and newlines) as an integer, discards the rest of the string.
- Prints out a message including the original number and its power and exits.

Sample Outputs

```
Please enter an integer between 1 and 9: 4
Power(2,4) = 16

Please enter an integer between 1 and 15: 14
Power(2,14) = 16,384

Please enter an integer between 1 and 50: 35
Power(2,35) = 34,359,738,368
```

The Programme

- Write an assembly language programme for the LC-3 to behave as described above. The programme must be documented so that it is easy to understand.
- Your programme can solve this problem using any method it chooses. For example, you can double an integer by adding it to itself, repeating *n* times. However, this method becomes difficult beyond *n*=15. You must print out the decimal value of the result. (Hint: there may be easier, more creative ways to

satisfy these requirements. A brute force method is also acceptable.)

- Your programme must handle integers up to n=9 to be considered correct. It will receive a small additional credit if it works for integers up to n=15, with a further small credit for integers up to n=50. This is not an easy assignment, so make sure your programme is working correctly for small integers before trying to handle larger numbers. You must check the input value of the input, and if it is larger than your programme can handle, print an error message and start over.

- To receive 100% credit your programme also must handle all input correctly and exit properly, indicating when an input is not understood or otherwise illegal. Your code should accurately indicate the largest integer for which the power is computed and displayed correctly.

- Your programme must first print out your name, UPI and AUID followed by information about the course, assignment number, and date. However, ***the digits of your ID must not be directly visible in their proper order in the assembly language code***. That is, it should not be as easy as looking for a sequence of digits (including labels) in the assembly code to determine your AUID. For example, you might print the digits of your UID as a series of ASCII characters, but rather than storing the characters, change each ASCII character by adding or subtracting a small constant from the previous value. Alternatively, your programme might embed your UID backwards in a string, then create a new string, copying the characters in reverse order and printing the new string. Requirement: the marker must not be able to determine your UID without running the programme or evaluating the code to determine how you generate it.

B. Changing the ISA for the LC-3.

The LC-3 has a very limited ISA, requiring some fairly common operations to be performed using multiple instructions. Suppose that the `add` instruction was changed to a `sub` instruction, that is, the format and operands would be specified exactly the same way, except that the second operand (SR2 or imm5) would be arithmetically *subtracted* from the first operand rather than added to it. (1) Describe an operation that would be made easier by this change. (2) Describe an operation that would be made more difficult by this change. (3) Do you think this would be a good idea? Explain your answer.

The `not` instruction can also be described as the 1's complement instruction, because for a computer using 1's complement representation, the instruction produces a result that is the additive inverse of the source operand. Suppose that the `not` instruction used bit 5 in the way the add and and instructions do, by which is to say, to change the operation from 1's complement to 2's complement. (4) Would this be useful? Explain. (5) How hard would it be to implement this new operation? (Hint: the ALU has two control bits that specify one of three operations: `add`, `and`, and 1's complement (`not`). It could easily be designed to have a fourth operation, 2's complement. How would that change the design of the ALU?)

Submit

1. An LC-3 programme, LC-3, satisfying the requirements of part A.
2. A txt, rtf, or pdf file answering the questions of part B.

Marking of Assignment

Part A: 80%

Your programme must satisfy minimal requirements stated above to receive nearly full credit. It should generally be well written and documented, making it easy to understand. Document by comments explaining high-level actions. Use meaningful identifiers for variables and labels.

Part B: 20%

Answer each part in complete sentences. Appendix A may be helpful for this part.

Submit electronically using the assignment drop box.

Due Date: 5PM, Friday, Apr 30, 2010