Assignment 2 - Assembly Language

Due Feb 15 at 11:59pm **Points** 33 **Questions** 5
Available Feb 8 at 8:45pm - Feb 15 at 11:59pm **Time Limit** None
Allowed Attempts Unlimited

Instructions

Please complete this short assignment, which is due NEXT WEDNESDAY

Take the Quiz Again

Attempt History

	Attempt	Time	Score
LATEST	Attempt 1	63 minutes	3 out of 33 *

* Some questions not yet graded

Score for this attempt: **3** out of 33 *
Submitted Feb 13 at 6:04pm
This attempt took 63 minutes.

Question 1

1 / 1 pts

Which of these is NOT a good reason to include BCD arithmetic in an instruction set?

○ Is minimimes conversion errors

Correct! ● It is faster than binary arithmetic

○ Converting to and from decimal can be slow

○ It is required by some language standards

Question 2

1 / 1 pts

If a language standard does not specify a maximum precision for floating point data types:

○ It can be assumed the the precision is as great as needed.

○ It can be assumed that the maximum precision is the same as the minimum precision.

Correct!

● There is a risk of portability issues from proggrams that depend on a higher-than-specified precision.

○ Floating point arithmetic must be implemented in software rather than hardware.

Question 3

1 / 1 pts

The 8-bit Extended ASCII standard:

Correct!

● Is a set of standards that support several European character sets.

○ Supports 137,994 glyphs and control characters.

○ Allows programs to use the 8th bit however they wish.

○ Is based on the EBCDIC character set.

Question 4

Not yet graded / 15 pts

Using *informal* pseudocode write an algorithm for the 1st pass of an assembler.

Assume someone has provided subroutines which you can call to:

- SCANLINE: Scan a line and return the tokens for $label, $opcode, $argument, and $comment - or an error
- LOOKUPOPCODE: Look up the $opcode and return the instruction length - or an error

Your algorithm should assume that the first instruction is at memory location 1000.

Your algorithm should assume that instructions have zero or one argument, and if the instruction has one argument, it may be an address (corresponding to a label), or an integer value referring to the number of bytes to reserve for a declaration. In other words, you **don't** need to handle statements like:

BYTE   C‘widget’

Your algorithm will need to track the $currentaddress as you process lines of the program. The $currentaddress will be affected by the instruction length returned by LOOKUPOPCODE, or the number of bytes specified in declarations.

Your algorithm should build a symbol table and a reference table.

The goal of this assignment is to be sure you understand the *concepts* of how a two-pass assembler is written (though this assignment is only for the 1st pass). *Make your pseudocode clear and specific enough for me to understand what you are doing* - but I am not trying to trip you up over specific syntax.

If you make simplifying assumptions, include those as comments in your response.

Your Answer:

Here is an algorithm for the first pass of an assembler.

```
//Start of the algorithm
//Set the current address to 1000
currAddr = 1000;

//Initialize the symbol table and reference table
symbolTable = {};
referenceTable = {};

//Repeat the below steps for all the lines
while there is a line in the program
{
    //Call the SCANLINE subroutine to get the tokens for label, opcode, argument, and comment
    tokens = SCANLINE();
    label = tokens[0];
    opcode = tokens[1];
    argument = tokens[2];
    comment = tokens[3];

    //If the label is not empty, add it to the symbol table with the current address as its value
    if label is not empty
    {
        symbolTable[label] = currAddr;
    }

    //If the opcode is not an error, call the LOOKUPOPCODE subroutine to get the instruction length
    if opcode is not an error
    {
        instrLen = LOOKUPOPCODE(opcode);

        //If the argument is not an error
        if argument is not an error
        {
            //If it is also not a number, it is a reference to a label
            if argument is not a member
            {
                //Add the reference to the reference table
                referenceTale[currAddr] = argument;
            }
            //If the argument is a number, it represents the # of bytes to reserve
            else
            {
                //Update the current address with the number of bytes to reserve
                currAddr = currAddr + argument;
            }
        }
        currAddr = currAddr + instrLen;
    }
}
//End of the algorithm
```

Question 5

Not yet graded / 15 pts

Write a SIC/XE program to computer the first 15 terms of a Fibonacci sequence, and store them in an array at the top of your program.

Fibonacci Sequence:

- Each term is the sum of the previous two terms
- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34…

Upload your program as a text file

Fibonacci15Q5.txt (https://sjsu.instructure.com/files/71844173/download)

Quiz Score: **3** out of 33