LC-3 Programming

CS 350: Computer Organization & Assembler Language Programming

Lab 7 due Tue Nov 8

A. Why?

- It's useful to know the issues involved in low-level programming

B. Outcomes

After this lab, you should

- Be able to write a short assembler program that reads and manipulates data

C. Programming Problem [50 points]

- You are to write an LC-3 program that reads a number from the keyboard, doubles it, and prints out the result. For example, you might have the following (user input is in `italics`)

```
Enter an integer: 1234
Two times 1234 is 2368
```

- In C terms, this would be equivalent to

```
int x; scanf("%d", &x); printf("%d\n", 2*x);
```

- Read the integer as a sequence of ASCII characters `'0'-'9'`, calculate the equivalent 16-bit 2's complement integer, double that integer, and then convert the result to a sequence of `'0'-'9'` so that you can print it out using `TRAP PUTS`.
- What to submit: Just your `.asm` file (no need for `.obj` etc., transcript of a run...)
- Here's some pseudocode for the program:

```
; READ IN A SEQUENCE OF DIGITS, CALCULATE
; THE EQUIVALENT INTEGER
; R0 = &read_buffer
; Call readstring
; val = 0
; pt = &read_buffer; c = *pt
; while c - '\n' != 0
;     val = val * 10
;     val = val + c - '0'
;     pt = pt + 1
;     c = *pt

; DOUBLE THE VALUE, BUILD STRING FOR IT
; val2 = val + val
; pt2 = &last char of print_buffer
; *pt2 = '\0'; pt--
; while val2 > 0
;   call divide subroutine with val and 10,
;      set val2 = quotient and rem = remainder
;   *pt2 = rem + '0'
;   pt2--
; print message containing original string and
; R0 = pt2; trap putstring  (print doubled value)

; SOME CONSTANTS
negative_of_return  .fill -10  ; (-'\n')
zero_char           .fill  48  ; ('0')
negative_zero_char  .fill -48  ; (-'0')

; DIVIDE SUBROUTINE (two arguments, num ≥ 0 and denom > 0 and
; two results: quotient and remainder.
; remainder = num
; quotient = 0
; neg_denom = -denom
; while remainder > 9
;   quotient ++
;   remainder = remainder + neg_denom
```

- Implementation notes: Use the `readstring` and `multiply` subroutines from class, write a `divide` subroutine to do the division.

Grading Guide

- **Read string of digits, convert to equivalent integer** (`scanf("%d", &value);`)
  - [2 points] Call readstring into buffer
  - [3 points] Initialize value, buffer ptr, get first buffer character

  - [2 points] while char ≠ `'0'`
    - [2 points] Call multiply and calc value = 10*value
    - [3 points] Convert char to 0-9, add to value
    - [1 point] Point to next buffer character and get it
- [2 points] Double the value
- **Build the string for the doubled integer**
  - [2 points] Initialize right end of output buffer
  - [2 points] Initialize buffer pointer, value2
  - [2 points] While value2 > 0
    - [2 points] Call divide subroutine with value2 and 10
    - [2 points] Set value2 = quotient, rem = remainder
    - [2 points] Convert rem to `'0'-'9'`, store in output buffer
    - [1 points] Point to previous char in output buffer
- [2 points] Print message with original and doubled values (using multiple I/O traps)
- **Readstring routine** [2 points] Reads integer correctly
- **Multiply(X,Y) routine** [2 points] Correctly calculates X * Y and returns the result
- **Divide(Numerator, Denominator) returning Quotient and Remainder routine**
  - [1 points] Save registers, initialize quotient and remainder
  - [3 points] Loop until remainder < denominator
    - [3 points] Increment quotient and decrement remainder by denominator
  - [2 points] Restore registers and return results
- **Program Structure & Comments**
  - [1 points] Name and section in comments
  - [2 points] Section comment for multiply, including register usage
  - [4 points] Sections well-organized, mnemonics, arguments, and line comments formatted and readable