**CIS 21JA - Lab 7: Procedures and macros**

**Overview**  
Change the code of lab 5 into a program with procedures.

(If you have feedback comments or point deduction in lab 5, make sure to fix the code of lab 5 before starting lab 7. Don't get points deducted in both labs for the same error.)

**Details**  
Download lab7.asm, which already has some code, and add code to it so that there 3 procedures and 1 macro.  
This lab is practice for reading someone else’s code (mine) as well as writing new code.

lab7.asm works the same way as lab 5:

* Ask the user for the start hour, start minute, end hour, end minute.  
  Check for valid hour and minute or re-prompt in the same way as lab 5.
* Find the difference between the 2 times and calculate the difference in hours and minutes.  
  Use all 8-bit data for the calculation as lab 5.
* Print the difference or print an error message and loop back to the top.

All variables are already defined in lab7.asm:

* *Strings* to prompt the user or for error messages
* An *array* of 2 bytes: to store the start total minutes and end total minutes
* A *variable* to store the difference in hours, and another *variable* to store the difference in minutes

Here are the parts you need to fill in lab7.asm:

1. Write a macro that accepts a string address as input and prints the string.
2. Fill in the code for the main procedure that’s already in lab7.asm, shown below.   
   The code to be filled in are highlighted in blue:

main PROC

top :

; 1. read time: pass arguments through \*registers\*

;; call readTime proc and pass first 3 strings and timeArr

; 2. find difference: pass arguments through \*the stack\*

;; call findDiff proc and pass timeArr, diffHr, diffMin

; 3. based on return value, either:

; a) print result

movzx eax, diffHr

call writeDec

;; write code to use macro to print hrOutStr

movzx eax, diffMin

call writeDec

;; write code to use macro to print minOutStr

jmp theEnd

; or b) print error message

invalidDiff :

;; write code to use macro to print diffErrStr

theEnd:

jmp top ; create infinite loop for testing

exit

main ENDP

1. Write the readTime procedure:  
   -- Accept the timeArr, and 3 strings through registers  
   -- *Loop 2 times*, in each iteration of the loop:
2. Prompt and read in the hour, check for 0-23 range, and print the error message and re-prompt if needed.
3. Prompt and read in the minute, check for 0-59 range, and print the error message and re-prompt if needed.
4. Calculate the total minutes and print the error message and re-prompt if needed.
5. Store the total minutes into the timeArr (the 2 elements in the array are for start and end time)

Note that the loop and steps a, b, c are straight from Lab 5.

* This means use 32-bit data in steps a and b, and use only 8 bit data in step c.
* If you don’t use 8-bit data in step c, you could run out of registers. Needless to say, you should not use variables in this procedure.

This exercise is to demo the disadvantage of passing arguments through registers.  
(What’s the advantage of passing arguments through registers?)

1. Write the findDiff procedure:  
   -- Accept the timeArr, diffHr, diffMin variables through the stack
2. Subtract to find the difference and divide by 60 to find the difference in hours and minutes.
3. If the time difference is valid, store the hours and minutes into the diffHr and diffMin variables without using these variable names (use indirect addressing only)
4. Return a Boolean value (your choice) to indicate valid or not valid time difference.

Note that:

* All calculations are with 8-bit data, but all addresses are still 32 bits
* The return value of step c should be returned through the stack

**Additional requirements, don't miss them:**

* + Make sure you set up and clean out the stack frame for your procedures. We use the STDCALL convention, so the called procedure (callee) must clear out input parameters on the stack.
  + Make sure to save and restore registers that the findDiff procedure and the macro use.
  + Other than main, *each procedure must use data that are passed in through the stack or through registers*.   
    Don't access variable names directly (*1/2pt off for each instance of variable name that your code uses in readTime or findDiff*)

**Testing and program output**

The testing and program output should be the same as for lab 5, but there’s no asking the user for continue. Instead, the main procedure is already written to keep looping back. To end the program, click X to close the window or use control-c to end the loop.