SYSC 2001 LAB6

This week, you will convert the program from Lab5 to pass all function parameters using the stack, instead of using registers. We will also create a new subroutine, `getSalary`, that returns an employee's salary on the stack. Follow the instructions below. Write down your answers to any questions that appear **in bold**.

Part 0: Using the stack to pass parameters and returning values from functions

a) Returning values from functions

So far, our functions have had input parameters, but not output parameters (i.e. they have not "returned" any value to the calling code). Returning a value is achieved through output parameters either using registers, using the stack (see below), or using global variables. To return a value using a register, simply set the register to the appropriate value before returning from the function. Of course, the calling code must know in which register to look for the value. *Hint: do not "save and restore" the register that will be returned to the calling code -- it is meant to be modified by your subroutine!*

b) Parameter Passing Using the Stack

Instead of using registers to pass input and output parameters to/from a subroutine, we can use the stack. To do this for input parameters, simply push the values onto the stack before you invoke the subroutine. This will build a "stack frame". Once in the subroutine, you can access your input parameters by simply reading the values from the stack frame. This is only complicated by the fact that the `CALL` instruction will have pushed a return address onto the stack and you may have saved some registers by pushing them onto the stack. Both of these actions effectively "cover" your input parameter values. Therefore, you need to read from "further up" in the stack. To do this, you need to first make a copy of the stack pointer (`SP`) in the base pointer (`BP`), then use the displacement addressing mode to read from your stack frame. Your first parameter will be at (something like) `[BP+2]`, etc. **Important: you must remember to `POP` the input parameters back off the stack upon returning to the calling (main) code.**

To use the stack to pass return parameters, you use the same procedure, but first make "room on the stack" prior to calling the subroutine. This is typically accomplished by `PUSH`ing a 16-bit register onto the stack, then using the `[BP+x]` trick within the subroutine to overwrite this slot in the stack frame. When the subroutine returns, the main function simply `POP`s the return value back off the stack.

Here is some sample code for a function that uses the stack to pass in one 8-bit input parameter and return one 16-bit return value:

```
function:
       PUSH AX        ; Save AX register since we will modify it here
       MOV BP,SP      ; Load a copy of SP into BP
       MOV AL,[BP+4]  ; Load the input parameter from the stack frame into AL
       ...            ; Do SOMETHING USEFUL. Return value ends up in AX
       MOV [BP+6],AX  ; Save return val in slot reserved in the stack frame
       POP AX         ; Restore AX register
       RET
```

```
Main:
       MOV SP,0000h   ; Initialize SP at the start of your program
       ...
       PUSH AX       ; Make room on the stack for return value of 'function'
       PUSH CX        ; Push function's input parameter (from CX) onto stack
       CALL function  ; Call the function
       POP CX         ; Pop function's input parameter from stack
       POP DX         ; Get return value from function into DX
       ...
```

In the middle of '`function`', the stack frame will look like:

Address	Value
`FFFFh (BP+7)`	`upper byte for return value`
`FFFEh (BP+6)`	`lower byte for return value`
`FFFDh (BP+5)`	`(garbage) CH`
`FFFCh (BP+4)`	`(input parameter) CL`
`FFFBh (BP+3)`	`upper return address`
`FFFAh (BP+2)`	`lower return address`
`FFF9h (BP+1)`	`(saved) AH`
`FFF8h (BP+0)`	`(saved) AL`

Part 1: Passing input parameters using the stack

Download the source file **`lab6-P1.asm`** and save-as to your new "Lab6" directory on your M-drive. This file contains a number of subroutines that you will need to complete Part 2 below. The subroutines all work, but you must change them in the following way: **Instead of passing input parameters using registers, pass the input parameters using the stack.**

Add the appropriate commands to each subroutine to pass all input parameters via the stack. <u>For each subroutine, draw your stack frame showing pushed parameters, return addresses, and saved registers such as the table above</u>.

Complete the `main` code in `lab6-P1.asm` to call each of the subroutines to test them. Use the comments in the code as hints for what needs to be added to `main()`...

Set a breakpoint in the middle of the `printDigit` subroutine *(at the `ADD AL, '0'` line)*. **What is the value of the SP when the breakpoint is reached for the first time? Sketch the current**

contents of the stack (i.e. what register values are at which memory addresses on the stack). *Hint: the stack contents will be slightly LARGER than for lab5... why?*

Stop and demonstrate your answers above to the TA before proceeding to Part 2

Part 2: Adding a return value using the stack

Download the source file **`lab6-P2.asm`** and save-as to your new "Lab6" directory on your M-drive. This file contains a function with the following c-style declaration:

```
 short int sal = getSalary(Employee* startStructArray, short int recordNumber)
```

This function returns the salary field from the specified record number (`recordNumber`) from the specified array of `Employee` structures (`startStructArray`).

Before you start writing code, sketch your planned stack frame. Starting with the return address, what will be on your stack, in which order, when you enter your `getSalary` subroutine (including registers that you will be saving/restoring)?

You must copy and paste the 4 subroutines *(not the* `main` *code)* from your **`lab6-P1.asm`** file into the appropriate place holders in **`lab6-P2.asm`**.

Complete the `getSalary()` subroutine such that all input and output parameters are pass using the stack.

Add instructions to the `main()` code to call your `getSalary()` subroutine to implement the following C-style calls:

```
// Print the 1st day shift worker to screen
short int salary1 = getSalary(dayShiftDB, 0); printSalary(salary1)
printNewLine();

// Print the 4th day shift worker to screen
short int salary2 = getSalary(dayShiftDB, 3);
printSalary(salary2)
printNewLine();

// Print the 1st night shift worker to screen
short int salary3 = getSalary(nightShiftDB, 0);
printSalary(salary3)
printNewLine();
```

Demonstrate and explain your code to a TA. Take note that you must use BASED-INDEXED addressing as indicated in the comments!

Use the submit program to submit your final `lab6-P2.asm` file.