CS 310 Stack Lab

Spring 2020

1 Introduction

In the last lab, we wrote a function called `printChar` that printed a single character to the terminal. Having a function to print a character is much better than setting up the registers and calling the BIOS for every character we print. Still, it's tedious to print messages one character at a time. In this lab, we will write some functions that can print longer messages without manually calling `printChar` for each character.

Function calls rely on a very simple data structure called a stack to keep track of who called them. The stack is like a trail of bread crumbs that the program can use to figure out where it should return to. The stack allows you to call the same function from many different places in your program, and it always knows how to get back to where it was called from when it returns.

The stack functions like a stack of plates. Every time we call a function, we put a new plate on the stack. The plate has written on it the address of the next instruction after the function call.

```
       main:
0x100   mov ax,6
0x102   mov bx,2
0x104   call add2nums
       loop:
0x106   jmp loop
       add2nums:
0x108   add ax,bx
0x10a   call check_sum
0x10c   ret
       check_sum:
0x10e   cmp ax,8
0x110   jne sum_wrong
0x112   mov ax,1
0x114   ret
       sum_wrong:
0x116   mov ax,1
0x118   ret
```

`0x106`	Return address pushed by the call to `add2nums`
`0x10c`	Return address pushed by the call to `check_sum`

When the program gets to address `0x114` in `check_sum` and it needs to return, the `ret` instruction will remove the address on the top of the stack (`0x10c`) and continue executing instructions at that address. That is the first instruction **after** the call.

2 Saving Registers

We can also use the stack to temporarily save the contents of the CPU registers in a function so we don't clobber them with local variables. The `push` and `pop` instructions can be used to add one number to the top of the stack.

```
printChar:
    push ax      ; Save AX on the stack
    push bx      ; Save BX on the stack
    ; Set up the registers for a BIOS call to print
    mov ah, 0x0e ; Write to terminal command
    xor bh,bh    ; Page 0
    mov bl,7     ; Foreground black
    mov al,'N'   ; Write an 'N' to the screen
    int 16       ; Call the BIOS!
    pop bx       ; Remove AX and BX in reverse order
    pop ax
    ret
```

	`Ret Addr`	Return address pushed by the call to `printChar`
	`AX`	Caller's AX
SP after `push BX`	`BX`	Caller's BX

3 Passing Parameters on the Stack

Below is an adaptation of the `putChar` function that takes its parameter on the stack, not in a register. Your job is to type this function in to emu8086 and call it from main to print a string. In order to call this function, you need to:

1. `push` that character you want to print.
2. `call putChar`
3. Clean up the stack after putChar returns, for example `add sp,2`

```
;
; Stack frame diagram for putChar:
;
; |-----------------------|
; | Character to print    |
; |-----------------------|
; | Return address        |
; |-----------------------|
; | Caller's BP           |  <- BP
; |-----------------------|
; | Caller's AX           |
; |-----------------------|
; | Caller's BX           | <- SP
; |-----------------------|
;
putChar:
    push bp      ; Save the caller's BP
    mov bp,sp    ; Point BP to our stack frame
    push ax      ; Save AX on the stack
    push bx      ; Save BX on the stack
    ; Set up the registers for a BIOS call to print
    mov ah, 0x0e ; Write to terminal command
    xor bh,bh    ; Page 0
    mov bl,7     ; Foreground black
    mov al,[bp+4]; Get character to print from the stack
    int 16       ; Call the BIOS!
    pop bx       ; Remove AX and BX in reverse order
    pop ax
    pop bp
    ret
```

3.1 Stack Frame Practice

```
int putChar(int c){
```

Draw the stack frame of this function below

Write the instructions needed to call this function in assembly. Pass the value in AX as `int c`

Write the prologue of this function and get the variable `int c` into AX

```
int printString(char *s){
    int i = 0;
```

Draw the stack frame of this function below

Write the instructions needed to call this function in assembly. Pass the value in AX as `char *s`

Write the prologue of this function and get the variable `char *s` into SI. Initialize `i` to 0.

```
int drawDot(int x, int y);
```

Draw the stack frame of this function below

Write the instructions needed to call this function in assembly. Pass `x = 10`, `y = 10`

Write the prologue of this function and get the variable `int y` into AX and `int x` into BX.

```
int drawRect(int x0, int y0, int w, int h);
    int currX, currY;
```

Draw the stack frame of this function below

Write the instructions needed to call this function in assembly. Pass `x0 = 10`, `y0 = 10`, `w = 20`, `h = 10`

Write the prologue of this function and get the variable `int x0` into AX and `int y0` into BX. Initialize `currX` and `currY` to `x0` and `y0` respectively.

```
int plotLine(int x0, int y0, int x1, int y1){
    int dx = x1 - x0;
    int dy = y1 - y0;
    int D = 2 * dy - dx;
```

Draw the stack frame of this function below

Write the instructions needed to call this function in assembly. Pass `x0 = 10`, `y0 = 10`, `x1 = 20`, `y1 = 30`

Write the prologue of this function and get the variable `int x0` into AX and `int y0` into BX.

4 Bresenham's Line Algorithm

The following pseudocode implements Bresenham's line algorithm to draw lines with a slope between 0 and 1. Implement this in assembly, passing parameters on the stack.

```
function PLOTLINE(x_0, x_1, y_0, y_1)
    dx ← x_1 − x_0
    dy ← y_1 − y_0
    D ← 2dy − dx
    y ← y_0
    for x from x_0 to x_1 do
        PLOT(x,y)
        if D > 0 then
            y ← y + 1
            D ← D − 2dx
        end if
        D ← D + 2dy
    end for
end function
```