Jade Yu Cheng
ICS 312
Homework #7
April 11, 2009

Exercise #3: Know your stack [10 pts]

Consider the following C code fragment:

```
#include <stdio.h>
void g(int n, int x, char y);
void f(int x, char y);

int main(int argc, char **argv) {
	f(4,5);
}

void f(int x, char y) {
	g(2,x,y);
	return;
}

void g(int n, int x, char y) {
	int z;
	if (n == 0) {
		printf("Values: [%d,%d]\n",x,y);
		return;
	}else {
		z = x+y;
		g(n-1, z, z+1);
	}
	return;
}
```

What is the content of the stack at the point when the code prints "Values: ...". Assume that there is nothing on the stack up to the call to function f. When showing values on the stack show integer values whenever possible as opposed to variable names (e.g., write "5" rather than "x"). Assume that only the EBP register is saved by these functions.

Answer:

```
%include "asm_io.inc"

segment .data
	msg	db	"Values: ",0	; the output message

segment .text
	global  asm_main
asm_main:
	push	ebp			; setup
	mov	ebp, esp		; setup
	pusha				; setup

	push	dword 4			; push the 1st param of f, 4
	push	dword 5			; push the 2nd param of f, 5
	call	f			; call function f
	add	esp, 8			; clean up the stack

	popa				; clean up
	mov	eax, 0			; clean up
	mov	esp, ebp		; clean up
	pop	ebp			; clean up
	ret				; clean up

;;; Code for function f
f:
	push	ebp			; setup
	mov	ebp, esp		; setup
	pusha				; setup

	push	dword 2			; push the 1st param of g, 2
	push	dword [ebp + 12]	; push the 2nd param of g, x
	push	dword [ebp + 8]		; push the 3rd param of g, y
	call	g			; call function g
	add	esp, 12			; clean up the stack

	popa				; clean up
	mov	eax, 0			; clean up
	mov	esp, ebp		; clean up
	pop	ebp			; clean up
	ret				; clean up

;;; Code for function g
g:
	push	ebp			; setup
	mov	ebp, esp		; setup
```

```
        pusha                           ; setup

        cmp     dword [ebp + 16], 0     ; if n == 0, print and end
        je      print
        mov     eax, [ebp + 12]         ; sum up x and y
        add     eax, [ebp + 8]          ; and store it in eax
        mov     ebx, [ebp + 16]         ; let ebx = n - 1
        dec     ebx
        push    ebx                     ; push the 1st param, n - 1
        push    eax                     ; push the 2nd param, x + y
        inc     eax                     ; let eax = x + y + 1
        push    eax                     ; push the 3rd param, x + y + 1
        call    g                       ; call function g
        add     esp, 12                 ; clean up stack
        jmp     g_end
print:
        mov     eax, msg                ; print the message
        call    print_string
        mov     eax, 91                 ; print [
        call    print_char
        mov     eax, [ebp + 12]         ; print x
        call    print_int
        mov     eax, 44                 ; print ,
        call    print_char
        mov     eax, [ebp + 8]          ; print y
        call    print_int
        mov     eax, 93                 ; print ]
        call    print_char
        call    print_nl                ; print a blank line

g_end:
        popa                            ; clean up
        mov     eax, 0                  ; clean up
        mov     esp, ebp                ; clean up
        pop     ebp                     ; clean up
        ret                             ; clean up
```

The assembly code above is converted from the C code given in the question. By executing this piece of code, I got the following standard output.

```
yucheng312@navet:~/private/hw7$ make
nasm -f elf hw7_ex3.asm -o hw7_ex3.o
gcc -m32 hw7_ex3.o driver.o asm_io.o -o hw7_ex3

yucheng312@navet:~/private/hw7$ ./hw7_ex3
Values: [19,20]
```

Contents of the stack are pretty clear when we examine them by looking at the assembly code. When the code prints “Values: [19, 20]”, the stack looks like:

```
                    ┆ 4                                                  ┆
                    ┆ 5                                                  ┆
                    ┆ return @ of function f                             ┆
      f's ebp →     ┆ main's ebp                                         ┆
                    ┆ :                                                  ┆
                    ┆ All registers                                      ┆
                    ┆ :                                                  ┆
                    ┆ 2                                                  ┆
                    ┆ 4                                                  ┆
                    ┆ 5                                                  ┆
                    ┆ return @ of the 1st function call of g (g1)        ┆
      g1's ebp →    ┆ f's ebp                                            ┆
                    ┆ :                                                  ┆
                    ┆ All registers                                      ┆
                    ┆ :                                                  ┆
                    ┆ 1                                                  ┆
                    ┆ 9                                                  ┆
                    ┆ 10                                                 ┆
                    ┆ return @ of the 2nd function call of g (g2)        ┆
      g2's ebp →    ┆ g1's ebp                                           ┆
                    ┆ :                                                  ┆
                    ┆ All registers                                      ┆
                    ┆ :                                                  ┆
                    ┆ 0                                                  ┆
                    ┆ 19                                                 ┆
                    ┆ 20                                                 ┆
                    ┆ return @ of the 3rd function call of g (g3)        ┆
      g3's ebp →    ┆ g2's ebp                                           ┆
                    ┆ :                                                  ┆
                    ┆ All registers                                      ┆
  current esp →     ┆ :                                                  ┆
```