Lab 1: Stack and Stack Frame

CSC 472/583

Kate Nguyen

Lab performed on 09/27/2021

Introduction

The purpose of this lab is to understand and get familiarize with the Stack and Stack frame while using gdb to debug the program and by executing specific functions such as mov, add, push, etc… to determine use of these functions and how it affects the stack.

Question 1: Pointing out the assembler code (including their memory address) which are used for creating the stack frame of the main() function (1 point). To create a new stack frame or mark the beginning of a new stack frame the code for the function is **mov ebp, esp**; the red circle indicates where it starts. Figure A. is from my terminal (memory address: 0x08049aa) and Figure B. is from the lab instruction (memory address: 0x080491a0).

Figure A.

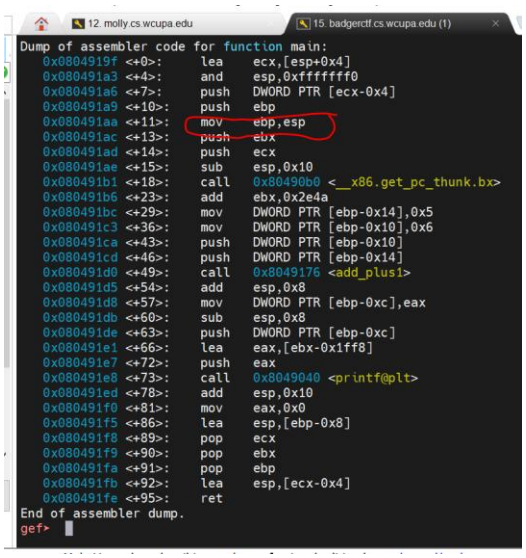
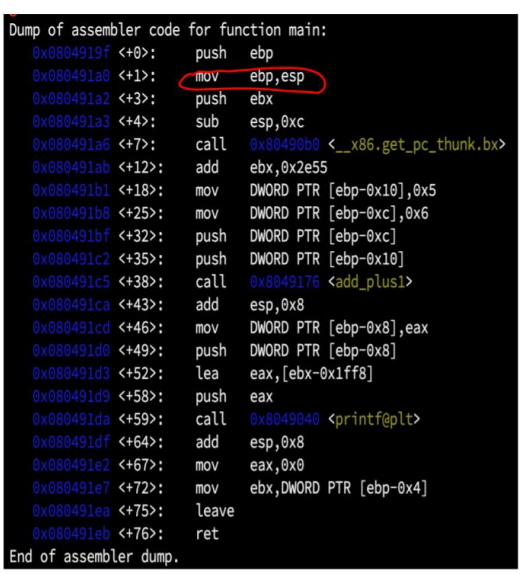


Figure B.



Question 2: What's the meaning of these two lines (1 point): Each two lines shows a specific memory address of ebp and the values that comes after, 0x5 / 0x6, are the values that will be stored there. The first line, the value of 0x5 will be moved to the memory address [ebp-0x10]. The second line, the value 0x6 will be moved into the memory address [ebp-0xc].

Figure C.

```
0x080491b1 <+18>:     mov     DWORD PTR [ebp-0x10],0x5
0x080491b8 <+25>:     mov     DWORD PTR [ebp-0xc],0x6
```

Question 3: Before calling add_plus1() function, the stack contains 5,6,5,6 (see the picture below), why there are two sets of "5,6" instead of just one?(1 point). Before calling the add plus1() function, the stack contains 5,6,5,6 ; where there are two sets of "5,6" because the first set of "5,6" which is moving 0x5 into memory address [ebp-0x10] and moving 0x6 into memory address [ebp-0xc]. After that, it pushes the memory address of [ebp-0xc] and then [ebp-0x10] into the stack frame which is the second set of "5,6".

Question 4: What's the meaning of the first two lines (1 point). The first line, push ebp is getting added to the stack frame. The second line, mov ebp, esp shows the value of esp getting stored into ebp.

Figure D.

```
0x08049176 <+0>:      push    ebp
0x08049177 <+1>:      mov     ebp,esp
```

Question 5: Which register is being used to store the final summation result? (1 point). The register used to store the final summation result is eax, which adds 0x1 onto the value that is in eax.

Figure E.

```
gef➤  disas add_plus1
Dump of assembler code for function add_plus1:
   0x08049176 <+0>:     push   ebp
   0x08049177 <+1>:     mov    ebp,esp
   0x08049179 <+3>:     sub    esp,0x10
   0x0804917c <+6>:     call   0x80491ff <__x86.get_pc_thunk.ax>
   0x08049181 <+11>:    add    eax,0x2e7f
   0x08049186 <+16>:    mov    eax,DWORD PTR [ebp+0x8]
   0x08049189 <+19>:    mov    DWORD PTR [ebp-0x8],eax
   0x0804918c <+22>:    mov    eax,DWORD PTR [ebp+0xc]
   0x0804918f <+25>:    mov    DWORD PTR [ebp-0x4],eax
   0x08049192 <+28>:    mov    edx,DWORD PTR [ebp-0x8]
   0x08049195 <+31>:    mov    eax,DWORD PTR [ebp-0x4]
   0x08049198 <+34>:    add    eax,edx
   0x0804919a <+36>:    add    eax,0x1
   0x0804919d <+39>:    leave
   0x0804919e <+40>:    ret
End of assembler dump.
```

Figure F.

```
Dump of assembler code for function add_plus1:
   0x08049176 <+0>:     push   ebp
   0x08049177 <+1>:     mov    ebp,esp
   0x08049179 <+3>:     sub    esp,0x8
   0x0804917c <+6>:     call   0x80491ec <__x86.get_pc_thunk.ax>
   0x08049181 <+11>:    add    eax,0x2e7f
   0x08049186 <+16>:    mov    eax,DWORD PTR [ebp+0x8]
   0x08049189 <+19>:    mov    DWORD PTR [ebp-0x8],eax
   0x0804918c <+22>:    mov    eax,DWORD PTR [ebp+0xc]
   0x0804918f <+25>:    mov    DWORD PTR [ebp-0x4],eax
   0x08049192 <+28>:    mov    edx,DWORD PTR [ebp-0x8]
   0x08049195 <+31>:    mov    eax,DWORD PTR [ebp-0x4]
   0x08049198 <+34>:    add    eax,edx
   0x0804919a <+36>:    add    eax,0x1
   0x0804919d <+39>:    leave
   0x0804919e <+40>:    ret
End of assembler dump.
```

Discussion & Conclusion

In conclusion, this lab fulfills the understanding and basic concepts of the stack and stack frame. The specific functions that were called such as move, push, add, etc… provided a great visual for CS students trying to learn the stack frame. The lecture provided a foundation of how these functions are used, however the visual demonstration in class helps echoes the knowledge and the work in lab facilitates this into memory.