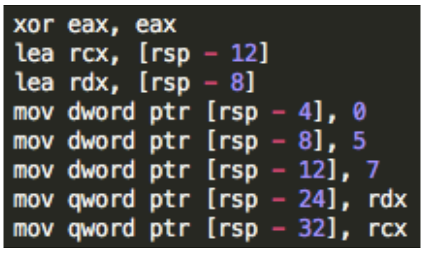
Jiwon Cha
CS 2150
11/01/2017

# Lab 8 report

## Parameter passing

To test how assembly treats different data types including int, float, char, pointer and objects, I wrote a .cpp file with functions that take in different data types and tested them inside the main function. The first thing that I noticed is that inside the main function, the assembly initializes 'normal' data types (int, char, float) with mov and pointers (references) with lea. Inside the stack, the assembly first moves the memory addresses at rsp-12 and rsp-8 to rcx and rdx. Another point that I noticed is that the assembly assigns rdx as the pointer that was declared first and rcx as the second pointer. Then, rsp grows downwards to include all of the local variables; 0, 5, 7 and two pointers.

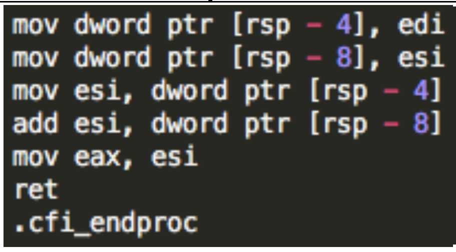


```
xor eax, eax
lea rcx, [rsp - 12]
lea rdx, [rsp - 8]
mov dword ptr [rsp - 4], 0
mov dword ptr [rsp - 8], 5
mov dword ptr [rsp - 12], 7
mov qword ptr [rsp - 24], rdx
mov qword ptr [rsp - 32], rcx
```

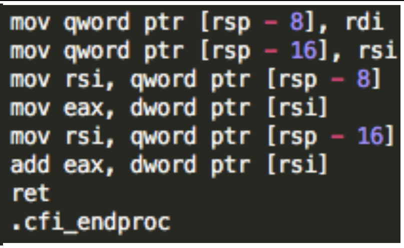
For my test cases, I wrote different functions to add the values of two parameters. First of all, the addition of two integers passed by value use eax, edi and esi as the registers. They are then pushed onto the stack and each spot is 4 bytes.

When the parameters are passed by reference, eax is used as the return value but rdi and rsi (64 bytes) are used to get the values of parameters, and the stack grows down by 8 bytes. Moreover, both values (stored in rsp-8 and rsp-16 respectively) are first stored in rsi and then added to the final eax value. The functions for pass by reference and pass by pointers were exactly the same, which makes sense because the assembly recognizes both reference and pointers as memory addresses. The functions written with char and float displayed similar behaviors as well.

**Pass by value** **Pass by pointer & reference**



```
mov dword ptr [rsp - 4], edi
mov dword ptr [rsp - 8], esi
mov esi, dword ptr [rsp - 4]
add esi, dword ptr [rsp - 8]
mov eax, esi
ret
.cfi_endproc
```

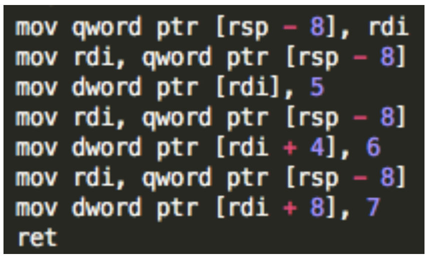


```
mov qword ptr [rsp - 8], rdi
mov qword ptr [rsp - 16], rsi
mov rsi, qword ptr [rsp - 8]
mov eax, dword ptr [rsi]
mov rsi, qword ptr [rsp - 16]
add eax, dword ptr [rsi]
ret
.cfi_endproc
```

When float values are passed by values, the registers used as parameters were xmm0 and xmm1, which are probably the registers used for floats and doubles (128 bits). Other than the registers used, the syntax is very similar to that of int function (passed by value) except xmm0 also acts as rax, and the commands used for floats are slightly different (movss, adds).

The function using char uses al, bl and cl as the registers (4 bits) because char is a small data type. The stack grows down by 1 byte. The syntax is also very similar to that of int function, except movsx is used to move the value stored in cl to eax (return value).

Finally, I wrote a C++ code to take in an array and assign 5,6, and 7 as its first three values. The assembly code for this is pretty simple; it first pushes the array that was passed in onto the stack at rsp-8. The first value (5) is stored at rdi, and two consecutive integers (6 and 7) are stored at rdi + 4 and rdi + 8 respectively.



```
mov qword ptr [rsp - 8], rdi
mov rdi, qword ptr [rsp - 8]
mov dword ptr [rdi], 5
mov rdi, qword ptr [rsp - 8]
mov dword ptr [rdi + 4], 6
mov rdi, qword ptr [rsp - 8]
mov dword ptr [rdi + 8], 7
ret
```