Programming Lauguage HW 6

20171645 박찬우

목차

2~6p : 코드 전문

7~14p : 각 코드 분석

15~16p : 정리

(

1. activation record 구조

2. subprogram상에서 parameter passing 방식

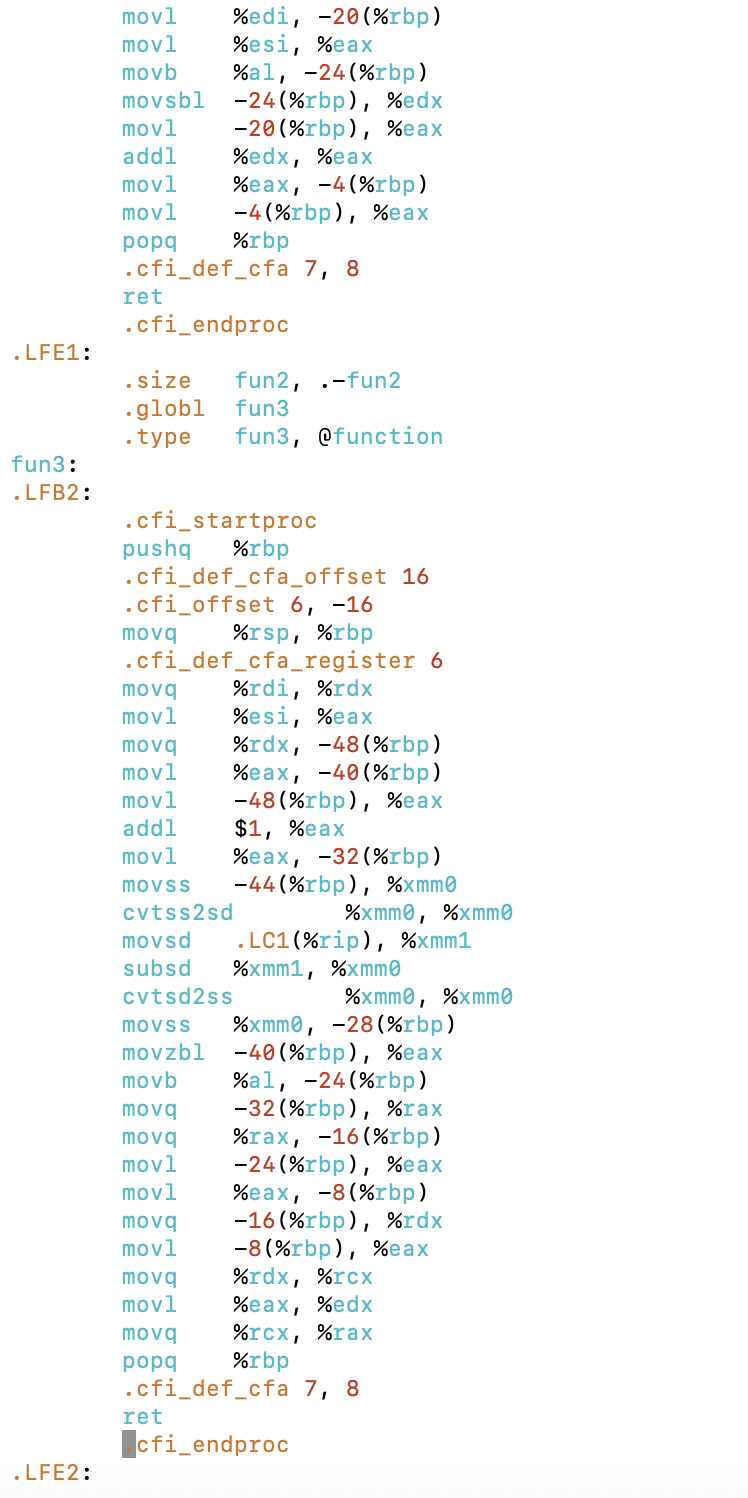
3. parameter passing 방식

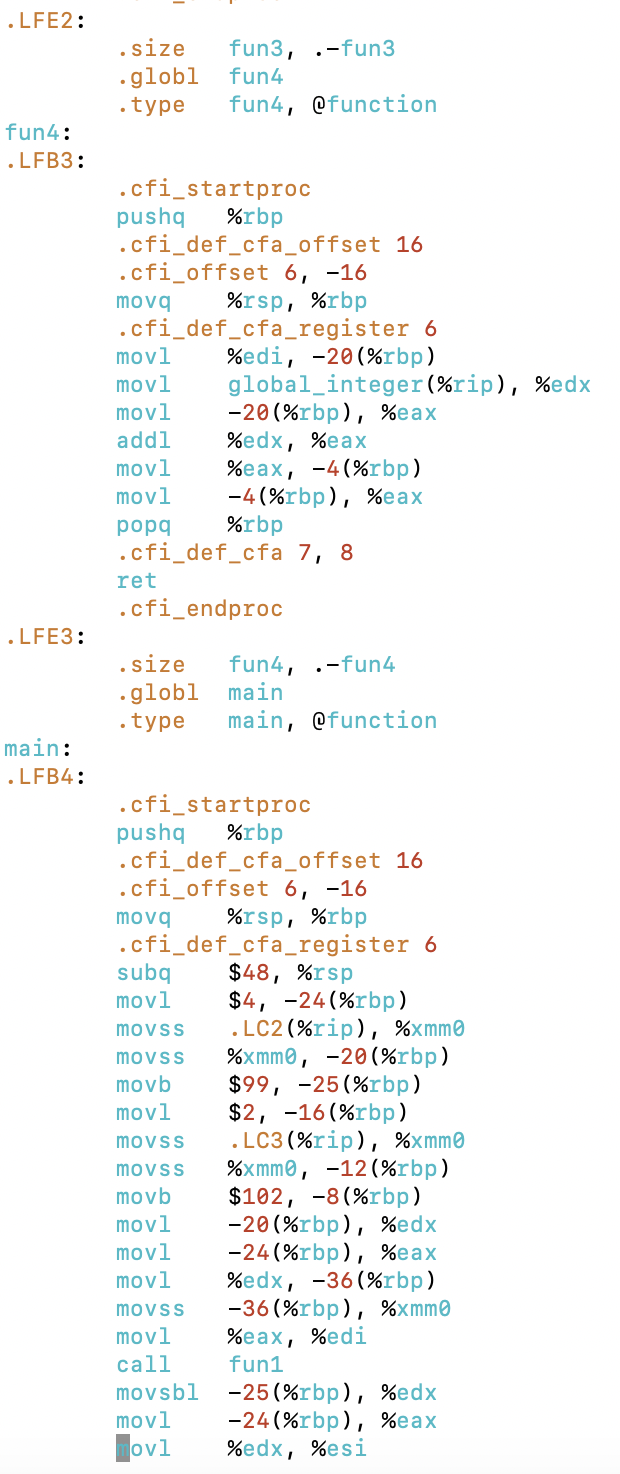
4. return value 전달방식

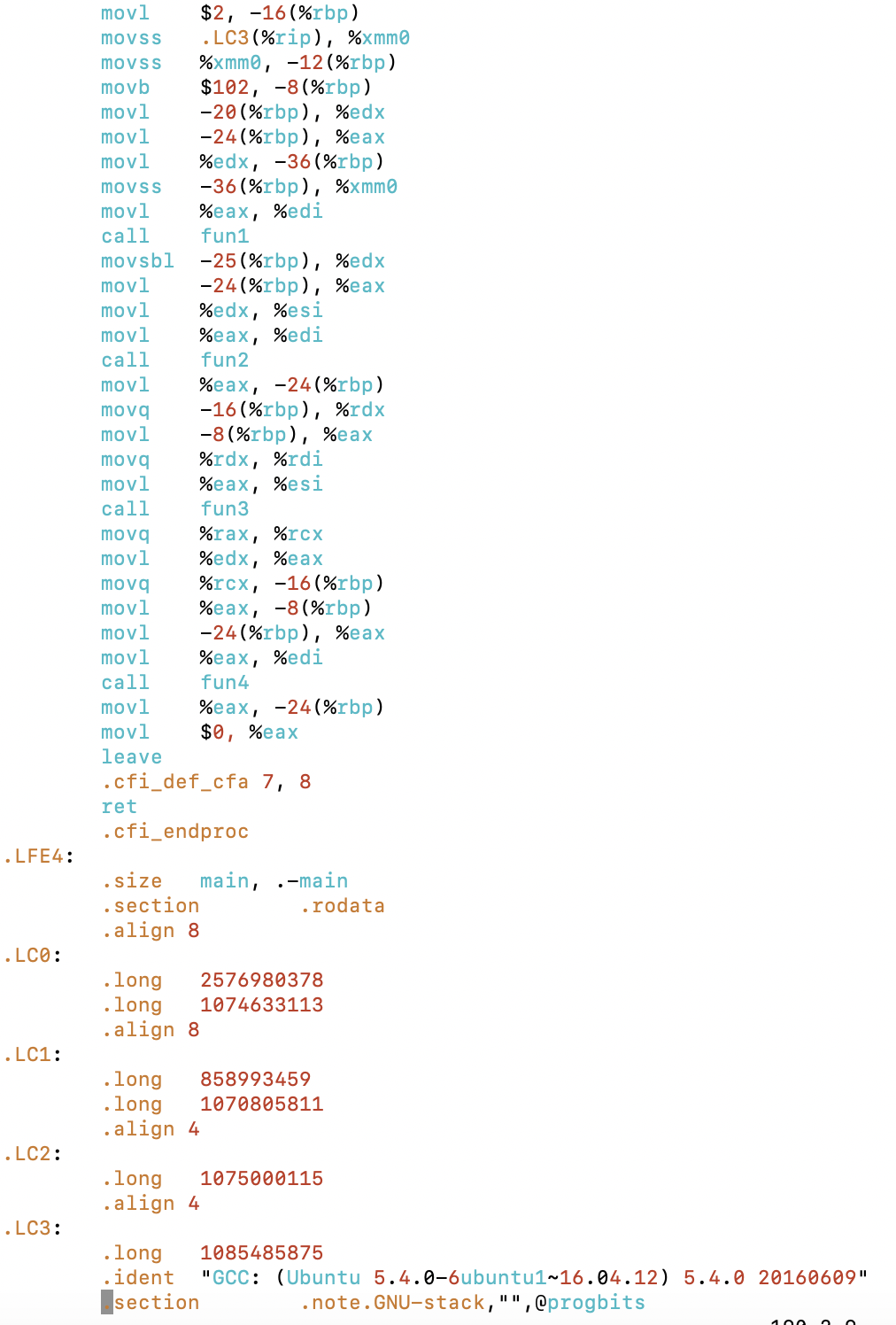
)





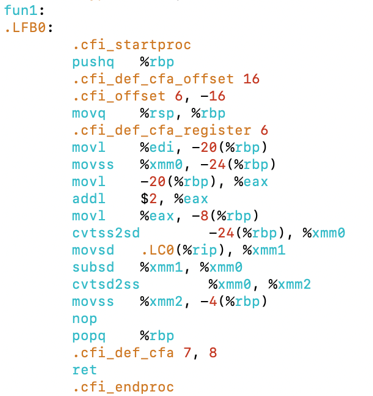






먼저 첫번째 함수인 fun1을 분석한다.





pushq %rbp를 통해 스택에 호출 함수의 stack frame을 저장하고,

movq %rsp, %rbp를 통해 현재 stack pointer를 가져와 호출된 함수의 frame으로 이용한다.

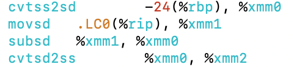
쉽게말해 pushq %rbp는 rbp레지스터에 스택을 push하는거고, movq %rsp, %rbp는 rsp값을 rsp에 저장해 현재 스택 위치를 저장한다고 생각하면 쉽다.

먼저, movl %edi, -20(%rbp), movl %edi, -24(%rbp)를 통해 fun1의 함수 parameter인 int integer, float floating이 각각 edi, xmm0 레지스터에서 스택 영역에 할당된다. 이를통해 C에서 함수 호출시 parameter는 integer type일경우 edi, esi등의 레지스터를 통해 들어오고, float type의 경우 xmm0, xmm1등의 레지스터를 통해 들어오는걸 확인할 수 있다.

그런 뒤, movl -20(%rbp), %eax 명령어를 통해 아까 옮겨줬던 stack 영역의 값을 eax 레지스터로 옮기고, addl $2, %eax 명령어를 통해 값을 2 더해준다. 이는 코드 영역의



에 해당한다.

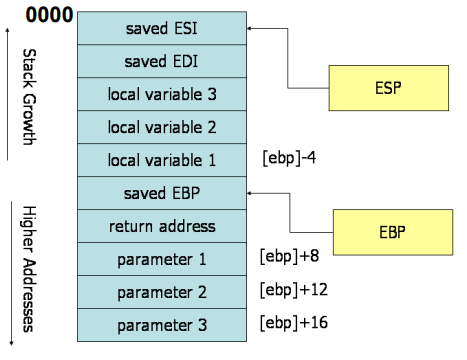


cvtss2sd는 one single-precision floating-point value를 one double-precision floation-point value로 변환해 옮겨주는 명령어로, 이전에 옮긴 스택 영역의 데이터 (floating) 를 다시 xmm0 레지스터로 옮기고, rip relative addressing에 따른 3.7이라는 값을 xmm1으로 옮기고, subsd 명령어로 floating 변수 값에서 3.7을 뺀다. 그 뒤 cvtsd2ss 명령어로 변경된 floating 변수의 값이 xmm2 레지스터에 저장된다.



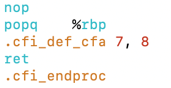
그런뒤, movss %xmm2, -4(%rbp) 명령어를 통해 계산된 값을 stack 영역에 다시 저장한다.

정리하면, 교재의 다음 그림처럼

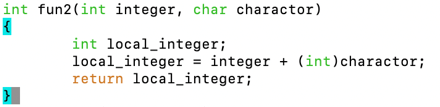


[rbp] – 4 위치에는 local variable 1인 local\_floating이 저장되고, [rbp] – 8 위치에는 local variable 2인 local\_integer 가 저장된다. 더 위로 올라가 [rbp] – 20 에는 prarameter인 edi 레지스터 값(변수 integer)가 저장되고, [rbp] - 24에는 parameter인 xmm0 레지스터 값(변수 floating)이 저장된다.

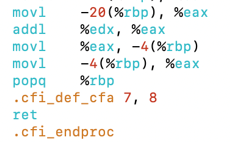
그 뒤,



popq를 통해 스택을 원래대로 되돌리고, ret 명령을 통해 return하는데, fun1과 다른 함수의 차이점이 있다.



fun2의 경우 fun1과 다르게 return값이 존재하는데, 어셈블리 코드의 마지막을 보면

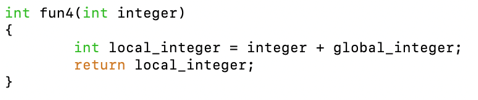


fun1과 달리 movl -4(%rbp), %eax 명령어가 존재한다. 바로 앞에 movl %eax, -4(%rbp)가 있음에도 반대로 다시 movl 시켜주는 이유는, 앞선 movl %eax, -4(%rbp) 는 eax 레지스터를 통해 연산한 결과를 스택 영역에 저장한 거지만, 뒤의 movl -4(%rbp), %eax 는 마지막

return local\_integer;

를 위한 명령어로, eax 레지스터에 local\_integer 값을 넣어준 뒤 return해 값을 넘겨주는 것이다.

추가로, 아래 fun4를 보면



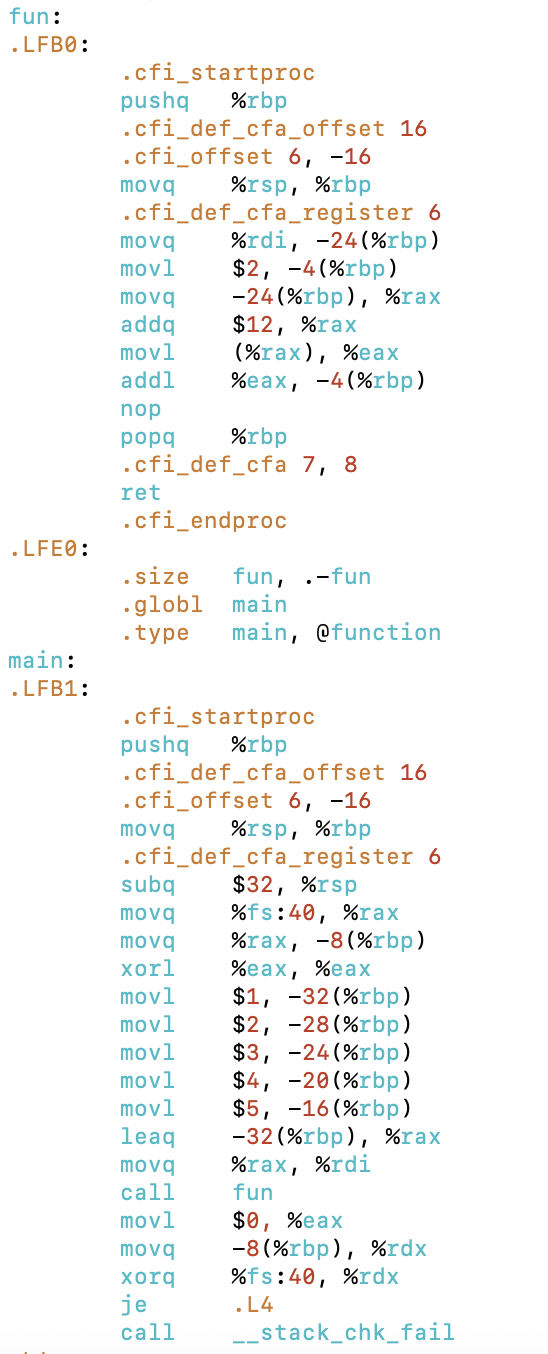
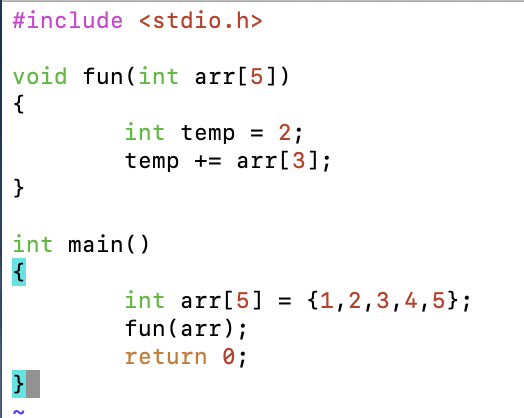


fun4는 global variable인 global\_integer를 사용하여 연산하는데, fun4의 어셈블리 코드를 보게되면 여타 코드는 앞선 코드와 동일한 방식이나,

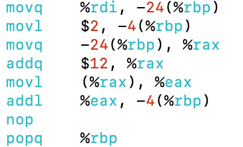
movl global\_integer(%rip), %edx

라는 명령어를 통해 global\_integer를 edx 레지스터에 저장하고 이를통해 연산하는걸 확인할 수 있다. 즉, subprogram에서 global 변수는 rip relative addressing을 통해 rip 레지스터를 사용해 접근한다고 볼 수 있다.

추가로,

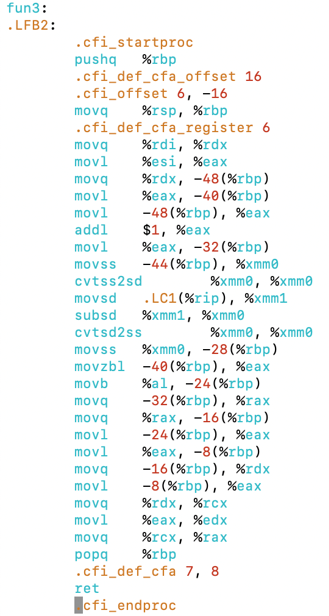
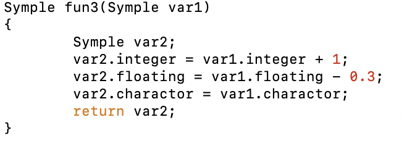


위 그림은 추가로 따로 작성한 코드로, array type의 변수가 어떻게 parameter passing되는지 확인할 수 있다.

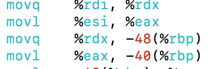


rdi 레지스터의 값을 [rbp] – 24로 옮기는데, 이는 앞선 다른 코드들처럼, parameter로 전달된 레지스터 값을 local variable 다음 위치에 저장하는 명령어로 볼 수 있다. 그 뒤, local variable을 할당해준뒤 (movl $2, -4(%rbp) ) rax 레지스터에 이전에 저장한 parameter인 array를 넘겨주고 (movq -24(%rbp), %rax ), 12를 더해 arr[3]의 값을 특정한 뒤, 그 값을 eax 레지스터에 옮긴 후 eax 레지스터를 local variable은 [rbp]-4에 더해주게 된다. 이를 통해 명령이 수행됨을 확인할 수 있다.

이와같은 과정을 통해, array type의 parameter는 rdi 레지스터를 통해 전달되는걸 확인할 수 있다.



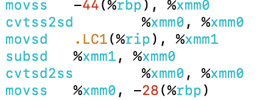
fun3은 structure data type이 어떻게 parameter passing되는지 확인할 수 있다.



를 통해 rdi, esi 레지스터를 통해 전달된 값을 [rbp]-48, [rbp]-40 에 저장하고,

는 앞서 확인한 fun1의 예시와 동일하게

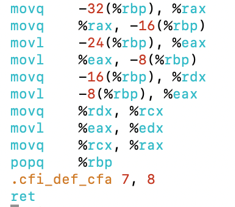
명령어를 수행하고,

는 동일하게

명령어를 수행한다.

각각의 계산된 값을 movl %eax, -32(%rbp) , movss %xmm0, -28(%rbp)를 통해 저장한다.

는를 수행한다.



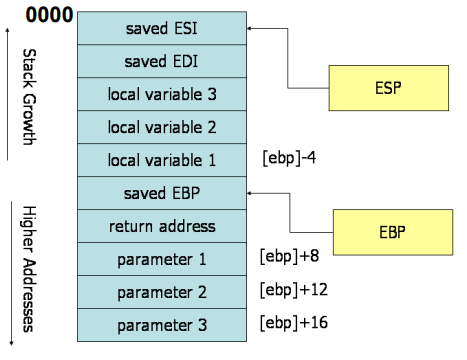
그런뒤, [rbp]-32에 위치한 값을 rax 레지스터 -> [rbp]-16으로 옮기고, [rbp]-24에 위치한 값을 eax 레지스터->[rbp]-8 위치로 옮긴다. 그 뒤 rdx, eax 레지스터가 각각 [rbp]-16, [rbp]-8 값을 갖고, rcx는 rdx를, edx는 eax레지스터를, rax는 rcx레지스터 값을 갖게 하고 return하게 된다.

이와같은 예시를 통해, 우선 structure type의 parameter는 내부의 성분으로 쪼개져 들어오게 된다. 이 예시에서 structure의 내용은 int type, float type, char type 하나씩이었기 때문에 각각 esi, xmm0, rdi 레지스터를 통해 parameter passing이 이루어졌다.

특기할점으로, [rbp]-40 ~ [rbp]-48 에는 전달받은 esi, xmm0, rdi 레지스터 값이 저장되고, [rbp]-24부터 [rbp]-32까지 구조체 각 성분의 값들이 저장되고, 이 값들은 [rbp]-8, [rbp]-16에 전달된 뒤 rax 레지스터는 rcx 레지스터를, rcx와 rdx는 [rbp]-16, eax, edx는 [rbp] – 8의 값을 가진채 return되는데, 이를 통해 rax,rdx 레지스터를 통해 구조체 데이터의 범위를 레지스터에 저장해 return 해준다는 걸 확인할 수 있다.

이를바탕으로 정리하면,

1. activation record 의 구조는 앞서 설명한 것처럼



다음 그림과 같은 구조를 띄며, 차이점은 esp, ebp 대신 rsp, rbp 레지스터가 stack의 시작과 끝을 가리킨다는 점이다. 함수가 호출되면, [rbp] + 4 위치에 return address가 위치하고, [rbp] - 4 부터 local variable이 위치하고, local variable들 뒤에 esi, edi, 혹은 xmm0, xmm1등이 저장된다. parameter는 이 함수 이전 함수의 변수들이므로, [rbp] + 8부터 위치해 있을테고, 함수 호출시 edi, esi, xmm0, xmm1등의 레지스터를 통해 전달된다. return value는 함수가 끝나면 ret 명령어 전에 eax 레지스터에 저장한 값이 return value가 되어 전달된다.

특기할 점으로, 명령어의 관점에서 보면 parameter 및 local variable은 순서대로 호출되지만, stack이라는 관점에서 봤을때 stack은 FILO 구조이므로, stack에서 데이터는 역순으로 저장된다고 생각할 수 있다. 예를들면, func(int x, int y)가 호출될때 stack에는 y를 넣은 뒤 x를 넣게 된다.

1. subprogram상에서 parameter 및 local 변수, global 변수 참조 방법 분석

앞서 설명했듯 subprogram상에서 parameter는 integer type의 parameter의 경우 edi, esi등의 레지스터를 통해 전달되고, float type parameter의 경우 xmm0, xmm1등의 레지스터를 통해 전달된다. local variable의 경우 subprogram 호출시 rbp를 rsp 위치로 옮긴 뒤, [rbp] - 4부터 local variable을 저장해 rbp로부터 떨어진 거리를 통해 local variable을 참조하거나 접근할 수 있다.

global variable의 경우 fun4을 통해 따로 설명했듯 rip relative addressing을 통해 rip 레지스터를 통해 참조하게 된다.

1. 다양한 데이터 타입에 대한 parameter passing 방식

fun1 및 fun3을 통해 알아봤듯 Integer type은 esi 등의 레지스터를 통해 전달되어 [rbp] 기준으로 local variables 다음 위치에 저장된다. Float type은 xmm0 등의 레지스터를 통해 전달되어 동일하게 [rbp] 기준으로 local variables 다음 위치에 저장된다. Array type은 따로 설명했듯 rdi 레지스터를 통해 parameter passing되어 들어오고, Structure type은 esi, rdi 레지스터를 통해 parameter passing이 일어나게 된다.

1. Return Value 전달 방식 분석

return value는 기본적으로 subprogram에서 마지막에 eax 레지스터, 혹은 rax 레지스터에 return할 값을 저장한 뒤 ret 하게되고,





이처럼 subprogram을 호출한 직후 eax 레지스터의 값을 return 값으로 활용하게된다.

만약 fun3 예처럼, subprogram의 return 값이 structure type일경우 rax, rdx 두 레지스터 쌍을 통해 return 하게된다.