Programming Language - HW7

2019155959 강상원

main.c

```c
#include <stdio.h>

float GLOBAL= 1.2f;

typedef struct {
    int i;
    float f;
} STRUCT;

float add(STRUCT s, float f, int i) {
    int j = i;
    return f + GLOBAL;
}

int main() {
    float f = 1.0f;
    float* fp = &f;
    STRUCT s = {1, 2.0f};
    STRUCT* sp = &s;

    float result = add(s, f, 1);
    printf("%lf\n", result);

    return 0;
}
```

main.S (assembly)

```
        .file   "main.c"
        .text
        .globl  GLOBAL
        .data
        .align 4
        .type   GLOBAL, @object
        .size   GLOBAL, 4
GLOBAL:
        .long   1067030938
        .text
        .globl  add
        .type   add, @function
add:
.LFB0:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        movq    %rdi, -24(%rbp)
        movss   %xmm0, -28(%rbp)
        movl    %esi, -32(%rbp)
        movl    -32(%rbp), %eax
        movl    %eax, -4(%rbp)
        movss   GLOBAL(%rip), %xmm0
        addss   -28(%rbp), %xmm0
        popq    %rbp
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE0:
        .size   add, .-add
        .section        .rodata
.LC2:
        .string "%lf\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB1:
        .cfi_startproc
        endbr64
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $48, %rsp
        movq    %fs:40, %rax
        movq    %rax, -8(%rbp)
        xorl    %eax, %eax
        movss   .LC0(%rip), %xmm0
        movss   %xmm0, -40(%rbp)
        leaq    -40(%rbp), %rax
        movq    %rax, -32(%rbp)
        movl    $1, -16(%rbp)
        movss   .LC1(%rip), %xmm0
        movss   %xmm0, -12(%rbp)
        leaq    -16(%rbp), %rax
        movq    %rax, -24(%rbp)
        movl    -40(%rbp), %edx
        movq    -16(%rbp), %rax
        movl    $1, %esi
        movd    %edx, %xmm0
        movq    %rax, %rdi
        call    add
        movd    %xmm0, %eax
        movl    %eax, -36(%rbp)
        cvtss2sd        -36(%rbp), %xmm0
        leaq    .LC2(%rip), %rdi
        movl    $1, %eax
        call    printf@PLT
        movl    $0, %eax
        movq    -8(%rbp), %rcx
        xorq    %fs:40, %rcx
        je      .L5
        call    __stack_chk_fail@PLT
.L5:
        leave
        .cfi_def_cfa 7, 8
        ret
        .cfi_endproc
.LFE1:
        .size   main, .-main
        .section        .rodata
        .align 4
.LC0:
        .long   1065353216
        .align 4
.LC1:
        .long   1073741824
        .ident  "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0"
        .section        .note.GNU-stack,"",@progbits
        .section        .note.gnu.property,"a"
        .align 8
        .long    1f - 0f
        .long    4f - 1f
        .long    5
0:
        .string  "GNU"
1:
        .align 8
        .long    0xc0000002
        .long    3f - 2f
2:
        .long    0x3
3:
        .align 8
4:
```

1. activation record 구조:

activation record는 함수가 호출될 때 생성되는 stack frame 구조를 나타낸다.

아래는 add 함수의 activation record 구조이다. 여기서 pushq %rbp는 stack frame을 설정하며, 이전 함수의 base pointer를 저장한다.

```
13 add:
14 .LFB0:
15         .cfi_startproc
16         endbr64
17         pushq   %rbp
18         .cfi_def_cfa_offset 16
19         .cfi_offset 6, -16
20         movq    %rsp, %rbp
21         .cfi_def_cfa_register 6
22         movq    %rdi, -24(%rbp)
23         movss   %xmm0, -28(%rbp)
24         movl    %esi, -32(%rbp)
25         movl    -32(%rbp), %eax
26         movl    %eax, -4(%rbp)
27         movss   GLOBAL(%rip), %xmm0
28         addss   -28(%rbp), %xmm0
29         popq    %rbp
30         .cfi_def_cfa 7, 8
31         ret
```

movq %rsp, %rbp는 새로운 base pointer를 설정하며,

그 이후의 코드들은 parameter 및 local 변수들을 stack에 저장하는 과정을 나타낸다.

현재 stack pointer를 base pointer로 설정

첫번째 인자 (STRUCT s) 주소를 activation record에 저장.

두번째 인자 (float f)를 activation record에 저장.

세 번째 인자 (int i)를 activation record에 저장.

local 변수 int j를 선언하고, 세 번째 인자 (int i)값을 j에 할당

할당된 j의 값을 activation record에 저장.

전역 변수 GLOBAL을 %xmm0 레지스터로 이동.

두 번째 인자 (float f)와 GLOBAL을 더하고 결과를 %xmm0에 저장.

Activation record 해제, 이전 base pointer 복원

결과 (%xmm0에 저장된 값)을 반환하고 함수 종료

2. Subprogram 상에서 parameter 및 local 변수, global 변수 참조 방법.

```
27         movss   GLOBAL(%rip), %xmm0
```

(main, add 전에 주소와 값 할당됨)

이 코드에서는 전역변수 GLOBAL을 참조한다. GLOBAL (%rip)를 통해 GLOBAL 값을 참조하고 있다.

또한 함수의 parameter는 stack에 저장된 후, 직접 참조된다. 예를 들어, add 함수의 float f parameter는 -28(%rbp) 위치에 저장된 후 참조된다. 즉 struct는 각각 변수들이 분할되어 전달되고, 실수는 xmm register, pointer : rsi register, 정수 : eax, ecx.. 등을 통해 전달된다.

3. 다양한 데이터 타입에 대한 Parameter Passing 방식.

기본적으로, C에서의 함수 호출 시 인자는 register 또는 stack을 통해 전달된다. Integer와 Float 형은 보통 register를 통해 전달되며, Array와 Structure는 메모리 주소 (포인터)를 통해 전달된다. 이번 예시에서는 STRUCT와 float, int가 인자로 사용되었는데, 각각 register %rdi, %xmm0, %esi에 저장되어 add 함수로 전달된다.

4. Return value 전달 방식.

함수의 반환 값은 보통 register를 통해 전달된다. 이번 예시에서는 %xmm0 register를 통해 float 값을 반환한다.

```
28         addss   -28(%rbp), %xmm0
```

```
68         call    add
69         movd    %xmm0, %eax
```