1주차 과제

가독성을 위해 Func1, Func2, Func3 (프롤로그) → Func3, Func2, Func1 (에필로그)로 작성하였습니다.

과제 (~5/12)

- Call stack을 개략적으로 구현해보고, 보고서(writeup) 작성하기
- 제공되는 함수 :func1, func2, func3, main, print_stack 함수
- 구현해야 하는 기능 : 함수 프롤로그, 함수 에필로그, push, pop (편하신대로)
- C/C++ 언어로만, x86(32bit) 환경으로 짜주세요

환경 설정

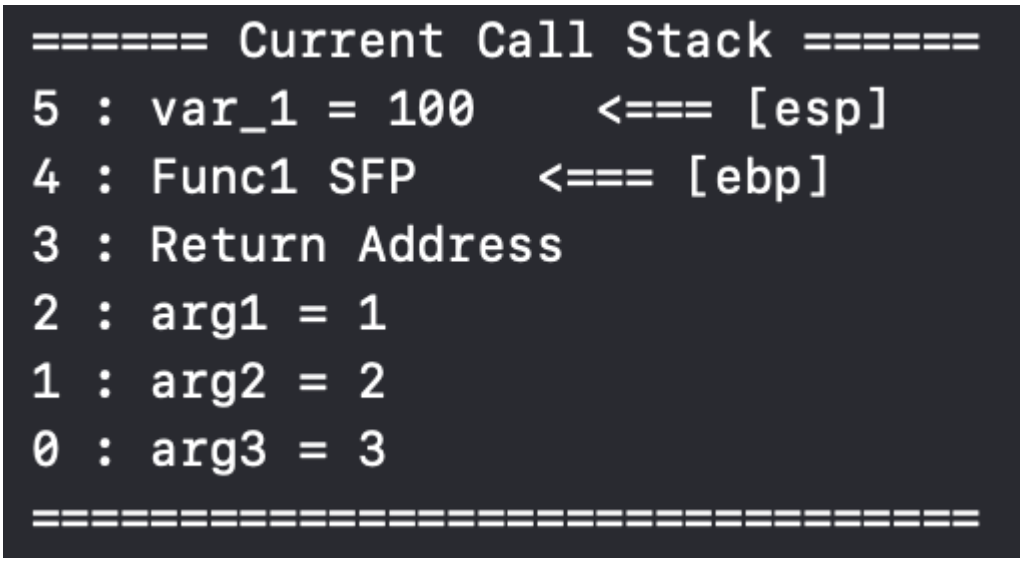
```
char    stack_info_list[STACK_SIZE][20] = {"arg1", "arg2", "arg3", "Return Addr
```

편의성을 위해 변수명들을 symbol처럼 미리 등록하여 복사하도록 하였습니다.

Func1 prologue

```
//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    // 인자 push
    for (int i = 3; i > 0; i--) {
        SP += 1;
        call_stack[SP] = i;
```

```
    strcpy(stack_info[SP], stack_info_list[i-1]);
}

// 반환 주소값
SP += 1;
call_stack[SP] = -1;
strcpy(stack_info[SP], stack_info_list[3]);

// SFP push
SP += 1;
call_stack[SP] = -1;
strcpy(stack_info[SP], stack_info_list[8]);
FP = SP;

// 지역변수
SP += 1;
call_stack[SP] = var_1;
strcpy(stack_info[SP], stack_info_list[4]);

print_stack();
func2(11, 13);
```



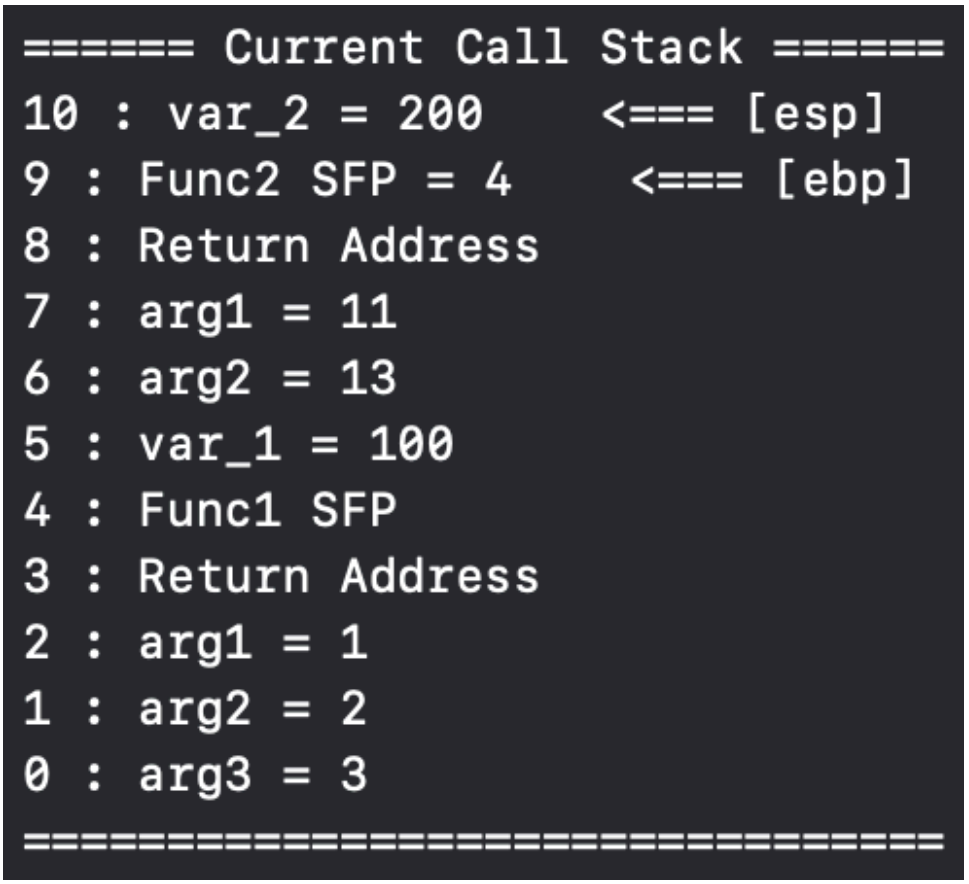
- 먼저 main 함수에서 Func1을 호출하였을 때, Func1의 파라미터들을 오른쪽에서 왼쪽으로 push 하였습니다. Func1은 인자가 단순해 반복문을 사용하여 push 하였습니다.
- 파라미터들을 push 한 후, return address의 값을 push 합니다.

- 다음으로, Func1의 SFP에 main 함수의 FP 값을 저장한 후, FP의 값을 SP의 값으로 변경합니다.
- 마지막으로, Func1의 지역변수 크기만큼 SP를 감소시킨 후, Func1의 지역변수를 넣습니다.

Func2 prologue

```
void func2(int arg1, int arg2)
{
    int var_2 = 200;

    // func2의 스택 프레임 형성 (함수 프롤로그 + push)
    // 인자 push
    SP += 1;
    call_stack[SP] = arg2;
    strcpy(stack_info[SP], stack_info_list[1]);
    SP += 1;
    call_stack[SP] = arg1;
    strcpy(stack_info[SP], stack_info_list[0]);

    // 반환 주소값
    SP += 1;
    call_stack[SP] = -1;
    strcpy(stack_info[SP], stack_info_list[3]);

    // SFP push
    SP += 1;
    call_stack[SP] = FP;
    strcpy(stack_info[SP], stack_info_list[9]);
    FP = SP;

    // 지역변수
    SP += 1;
    call_stack[SP] = var_2;
    strcpy(stack_info[SP], stack_info_list[5]);
```

```
    print_stack();
    func3(77);
```



- 마찬가지로, Func2의 파라미터 값을 오른쪽에서부터 왼쪽으로 push 해줍니다.
- 함수 호출이 끝나면 다시 돌아갈 코드의 주소도 Return Address로 저장합니다.
- Func2의 SFP에 Func1의 FP 값을 push한 후, FP의 값을 SP의 값으로 변경합니다.
- Func2의 지역변수 크기만큼 SP를 감소시킨 후, 지역변수를 넣습니다.

Func3 prologue

```
void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    // 인자 push
```

```
    SP += 1;
    call_stack[SP] = arg1;
    strcpy(stack_info[SP], stack_info_list[0]);

    // 반환 주소값
    SP += 1;
    call_stack[SP] = -1;
    strcpy(stack_info[SP], stack_info_list[3]);

    // SFP push
    SP += 1;
    call_stack[SP] = FP;
    strcpy(stack_info[SP], stack_info_list[10]);
    FP = SP;

    // 지역변수
    SP += 2;
    call_stack[SP-1] = var_3;
    strcpy(stack_info[SP-1], stack_info_list[6]);
    call_stack[SP] = var_4;
    strcpy(stack_info[SP], stack_info_list[7]);

    print_stack();
}
```

```
====== Current Call Stack ======
15 : var_4 = 400     <=== [esp]
14 : var_3 = 300
13 : Func3 SFP = 9     <=== [ebp]
12 : Return Address
11 : arg1 = 77
10 : var_2 = 200
9 : Func2 SFP = 4
8 : Return Address
7 : arg1 = 11
6 : arg2 = 13
5 : var_1 = 100
4 : Func1 SFP
3 : Return Address
2 : arg1 = 1
1 : arg2 = 2
0 : arg3 = 3
================================
```

- Func3의 파라미터 값을 오른쪽에서 왼쪽으로 push 해줍니다.
- Func3의 호출이 종료되고 Func2에서 실행할 코드 주소를 Return Address에 push 합니다.
- Func3 SFP에 Func2의 FP 값을 저장한 후, FP의 값을 현재 SP의 값으로 복사합니다.
- Func3의 지역변수 크기만큼 SP를 감소시킨 후, 지역변수를 넣습니다.

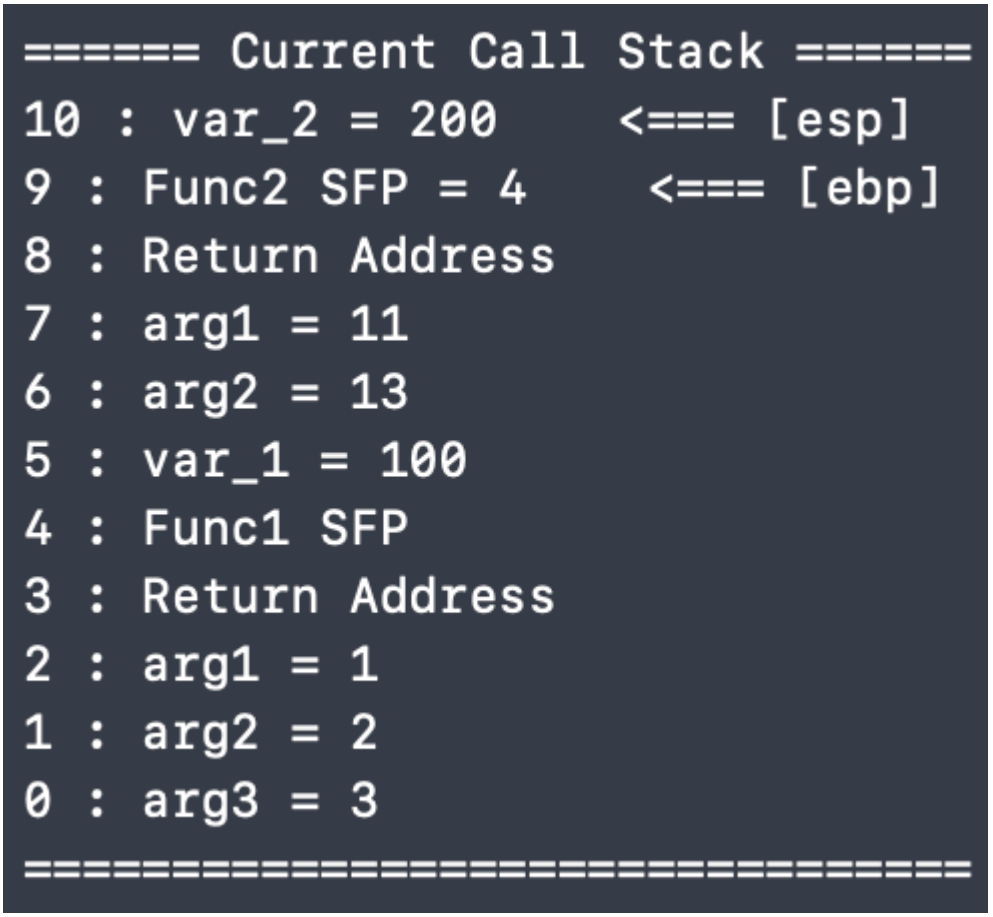
에필로그 방식을 구현하기에 앞서,

- 에필로그방식은 cdecl 방식으로 진행하였습니다.
- https://en.wikipedia.org/wiki/X86_calling_conventions#cdecl

- 위 x86 함수호출 규약에 따라 에필로그 과정에서 callee의 인자를 제거할 때 pop을 사용하지 않고 SP의 값을 증가시켜 제거했습니다.
- POP 어셈블리 언어는 메모리를 초기화하지 않고 값을 복사하고, 과제 진도에서 Return Address를 pop하는 과정을 구현하지 않기에, 단순히 SP의 값을 증가시켜 구현하였습니다.

Func3 epilogue

```
    func3(77);

    // func3의 스택 프레임 제거 (함수 에필로그 + pop)
    // 지역변수 및 SFP를 통한 FP 수정
    SP = FP;
    FP = call_stack[SP];
    SP -= 1;

    // Return Address pop
    SP -= 1;

    // func3 인자 제거
    SP -= 1;

    print_stack();
}
```

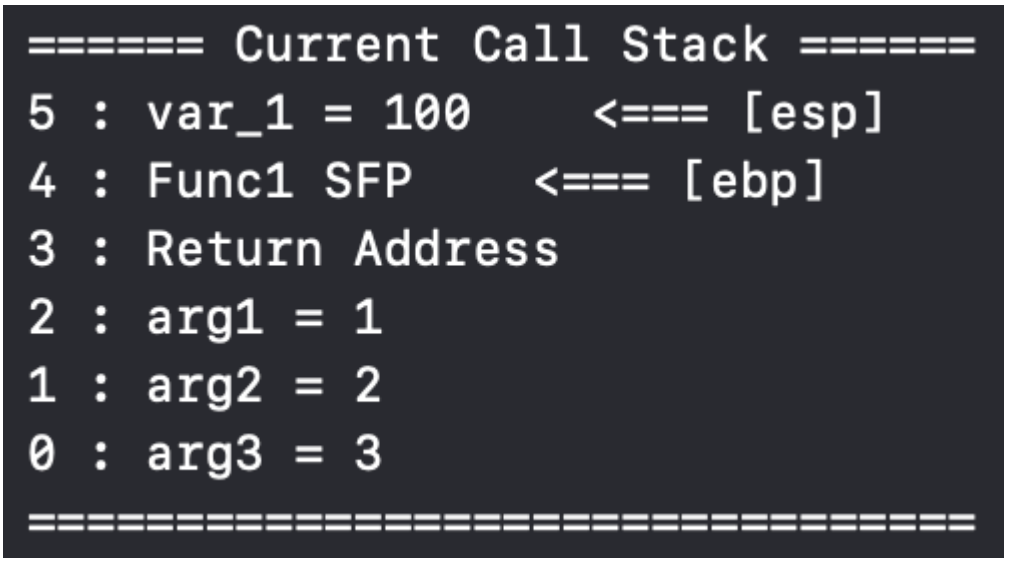


- Func3의 호출이 끝나면 Caller인 Func2에서 SP의 값을 FP로 복사해 지역 변수를 정리합니다.
- FP에 SFP에 들어있는 Func2의 값을 넣고, SP를 4바이트 더 증가시킵니다.
- Return Address의 값을 pop시켜 eip에 저장 합니다.
- x86 호출규약에 따라 Func3의 인자 크기만큼 SP를 증가시켜 Func3의 인자를 지웁니다.

Func2 epilogue

```
func2(11, 13);

// func2의 스택 프레임 제거 (함수 에필로그 + pop)
// 지역변수 및 SFP를 통한 FP 수정
SP = FP;
FP = call_stack[SP];
SP -= 1;
```

```
    // Return Address pop
    SP -= 1;

    // func2 인자 제거
    SP -= 2;

    print_stack();
}
```



- Func2의 Caller인 Func1에서 SP의 값을 FP로 복사해 지역 변수를 정리합니다.
- SP의 위치에 해당하는 SFP의 값을 FP로 복사한 후 SP의 값을 4바이트 증가시킵니다.
- Return Address의 값을 pop하여 eip에 복사합니다.
- Func2의 인자값의 크기만큼 SP의 값을 증가시켜 스택을 정리합니다.

Func1 epilogue

```
    func1(1, 2, 3);

    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    // 지역변수 및 SFP를 통한 FP 수정
    SP = FP;
    FP = call_stack[SP];
    SP -= 1;
```

```
    // Return Address pop
    SP -= 1;

    // func1 인자 제거
    SP -= 3;

    print_stack();
    return 0;
}
```

```
Stack is empty.
```

- Func1의 Caller인 main 함수에서 SP의 값을 FP로 넣어 Func1의 지역변수를 정리합니다.
- SP에 위치한 SFP의 값을 FP로 옮긴 후, SP를 4바이트 증가시킵니다.
- Return Address의 값을 pop시켜 eip에 저장 후, Func1의 인자값의 크기만큼 SP를 증가시켜 스택을 정리합니다.

전체 코드

```
/*  call_stack

    실제 시스템에서는 스택이 메모리에 저장되지만, 본 과제에서는 `int` 배열을 이용하여 ㄷ
    원래는 SFP와 Return Address에 실제 가상 메모리 주소가 들어가겠지만, 이번 과제어

    int call_stack[]      : 실제 데이터(`int 값`) 또는 `-1` (메타데이터 구분용)을 저장하는 i
    char stack_info[][]   : call_stack[]과 같은 위치(index)에 대한 설명을 저장하는 문자

    ============================call_stack 저장 규칙==============
    매개 변수 / 지역 변수를 push할 경우   : int 값 그대로
    Saved Frame Pointer 를 push할 경우  : call_stack에서의 index
    반환 주소값을 push할 경우       : -1
    ================================================================
```

```
   ============================stack_info 저장 규칙==============
   매개 변수 / 지역 변수를 push할 경우        : 변수에 대한 설명
   Saved Frame Pointer 를 push할 경우  : 어떤 함수의 SFP인지
   반환 주소값을 push할 경우                : "Return Address"
   ============================================================
*/
#include <stdio.h>
#include <string.h>
#define STACK_SIZE 50 // 최대 스택 크기

int     call_stack[STACK_SIZE];         // Call Stack을 저장하는 배열
char    stack_info[STACK_SIZE][20];     // Call Stack 요소에 대한 설명을 저장하는 배
char    stack_info_list[STACK_SIZE][20] = {"arg1", "arg2", "arg3", "Return Addr

/*  SP (Stack Pointer), FP (Frame Pointer)

   SP는 현재 스택의 최상단 인덱스를 가리킵니다.
   스택이 비어있을 때 SP = -1, 하나가 쌓이면 `call_stack[0]` → SP = 0, `call_stack[1

   FP는 현재 함수의 스택 프레임 포인터입니다.
   실행 중인 함수 스택 프레임의 sfp를 가리킵니다.
*/
int SP = -1;
int FP = -1;

void func1(int arg1, int arg2, int arg3);
void func2(int arg1, int arg2);
void func3(int arg1);



/*
   현재 call_stack 전체를 출력합니다.
   해당 함수의 출력 결과들을 바탕으로 구현 완성도를 평가할 예정입니다.
*/
void print_stack(void)
{
```

```
    if (SP == -1)
    {
        printf("Stack is empty.\n");
        return;
    }

    printf("====== Current Call Stack ======\n");

    for (int i = SP; i >= 0; i--)
    {
        if (call_stack[i] != -1)
            printf("%d : %s = %d", i ,stack_info[i], call_stack[i]);
        else
            printf("%d : %s", i, stack_info[i]);

        if (i == SP)
            printf("    ⇐= [esp]\n");
        else if (i == FP)
            printf("    ⇐= [ebp]\n");
        else
            printf("\n");
    }
    printf("================================\n\n");
}


//func 내부는 자유롭게 추가해도 괜찮으나, 아래의 구조를 바꾸지는 마세요
void func1(int arg1, int arg2, int arg3)
{
    int var_1 = 100;

    // func1의 스택 프레임 형성 (함수 프롤로그 + push)
    // 인자 push
    for (int i = 3; i > 0; i--) {
        SP += 1;
        call_stack[SP] = i;
        strcpy(stack_info[SP], stack_info_list[i-1]);
    }
```

```
    // 반환 주소값
    SP += 1;
    call_stack[SP] = -1;
    strcpy(stack_info[SP], stack_info_list[3]);

    // SFP push
    SP += 1;
    call_stack[SP] = -1;
    strcpy(stack_info[SP], stack_info_list[8]);
    FP = SP;

    // 지역변수
    SP += 1;
    call_stack[SP] = var_1;
    strcpy(stack_info[SP], stack_info_list[4]);

    print_stack();
    func2(11, 13);

    // func2의 스택 프레임 제거 (함수 에필로그 + pop)
    // 지역변수 및 SFP를 통한 FP 수정
    SP = FP;
    FP = call_stack[SP];
    SP -= 1;

    // Return Address pop
    SP -= 1;

    // func2 인자 제거
    SP -= 2;

    print_stack();
}


void func2(int arg1, int arg2)
{
```

```
int var_2 = 200;

// func2의 스택 프레임 형성 (함수 프롤로그 + push)
// 인자 push
SP += 1;
call_stack[SP] = arg2;
strcpy(stack_info[SP], stack_info_list[1]);
SP += 1;
call_stack[SP] = arg1;
strcpy(stack_info[SP], stack_info_list[0]);

// 반환 주소값
SP += 1;
call_stack[SP] = -1;
strcpy(stack_info[SP], stack_info_list[3]);

// SFP push
SP += 1;
call_stack[SP] = FP;
strcpy(stack_info[SP], stack_info_list[9]);
FP = SP;

// 지역변수
SP += 1;
call_stack[SP] = var_2;
strcpy(stack_info[SP], stack_info_list[5]);

print_stack();
func3(77);

// func3의 스택 프레임 제거 (함수 에필로그 + pop)
// 지역변수 및 SFP를 통한 FP 수정
SP = FP;
FP = call_stack[SP];
SP -= 1;

// Return Address pop
SP -= 1;
```

```
    // func3 인자 제거
    SP -= 1;

    print_stack();
}


void func3(int arg1)
{
    int var_3 = 300;
    int var_4 = 400;

    // func3의 스택 프레임 형성 (함수 프롤로그 + push)
    // 인자 push
    SP += 1;
    call_stack[SP] = arg1;
    strcpy(stack_info[SP], stack_info_list[0]);

    // 반환 주소값
    SP += 1;
    call_stack[SP] = -1;
    strcpy(stack_info[SP], stack_info_list[3]);

    // SFP push
    SP += 1;
    call_stack[SP] = FP;
    strcpy(stack_info[SP], stack_info_list[10]);
    FP = SP;

    // 지역변수
    SP += 2;
    call_stack[SP-1] = var_3;
    strcpy(stack_info[SP-1], stack_info_list[6]);
    call_stack[SP] = var_4;
    strcpy(stack_info[SP], stack_info_list[7]);

    print_stack();
```

```
}


//main 함수에 관련된 stack frame은 구현하지 않아도 됩니다.
int main(void)
{
    func1(1, 2, 3);

    // func1의 스택 프레임 제거 (함수 에필로그 + pop)
    // 지역변수 및 SFP를 통한 FP 수정
    SP = FP;
    FP = call_stack[SP];
    SP -= 1;

    // Return Address pop
    SP -= 1;

    // func1 인자 제거
    SP -= 3;

    print_stack();
    return 0;
}
```