EA0003: 시스템소프트웨어 실습

- 실습 번호: lab05

- 실습 디렉터리: ~/lab05 이름: 김동현

- 실습 날짜: 2018년 11 월 28 일

- 실습 제목: Subprogram 학번: 201511449

- 실습 내용:

1. (실습준비) 자신의 홈 디렉터리 아래에 실습 디렉터리 (~/lab05)를 만든다. 이번 실습에서 만들어지는 모든 파일들은 이 디렉터리에 두도록 한다.
2. 다음 두 개의 모듈로 이루어진 어셈블리 프로그램에 대해 물음에 답하시오.

main:

**…**

push dword 3

push dword 2

push dword 1

call subpr

add esp, 12

**…**

subpr:

push ebp

mov ebp, esp

sub esp, 8

;

; ①

;

mov esp, ebp

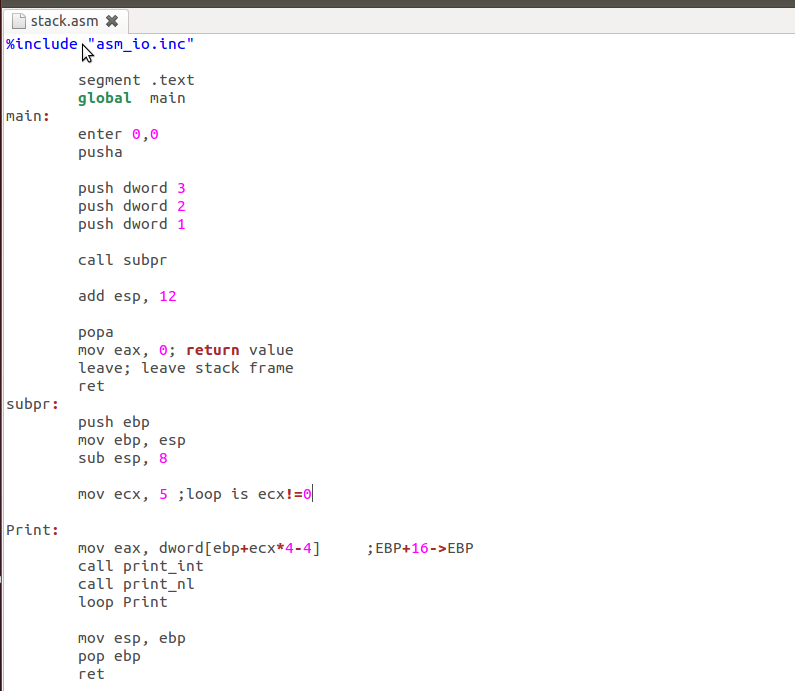
pop ebp

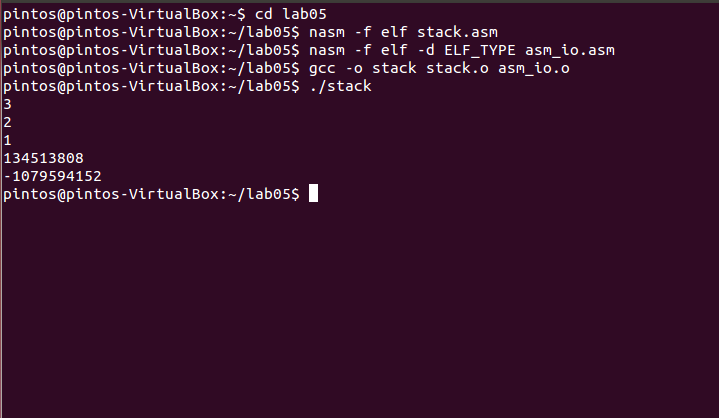
ret

* 1. 위 어셈블리 프로그램의 각 라인에 적절한 설명문을 붙이고, 서브프로그램 subpr 에서 call frame 이 형성된 직후 (① 부분)의 스택의 모습을 그리시오.

|  |  |  |
| --- | --- | --- |
| 3 | EBP+16  EBP+12  EBP+8  EBP+4  EBP | HIGH  Low |
| 2 |
| 1 |
| Return address |
| Saved EBP |

s

* 1. ① 부분에 call frame의 내용을 출력하는 코드를 추가한 프로그램 stack.asm을 작성하시오. 이 프로그램을 어셈블, 링크하여 실행 파일 stack 을 생성하고, 이를 실행시킨 결과를 화면 캡쳐하여 첨부하시오. 첨부된 출력값이 1.1.의 답과 같은지 확인하시오.



1.1과 같다.

* 1. 서브프로그램의 시작 시 call frame을 형성하는 부분과 종료 직전 call frame을 해제하는 부분은 각각 단일 명령어인 enter 와 leave 로 대체할 수 있다.
     1. 참고자료 NASM 매뉴얼의 appendix B.4.65 절과 B.4.136절을 참고하여 이 명령어들에 대한 사용 문법, 기능, 동작 과정 등을 조사하여 적으시오.

ENTER imm, imm

ENTER는 고급언어의 호출절차를 위한 스택프레임을 구성한다.

PUSH EBP ;or PUSH BP 16비트에서

MOV EBP, ESP ;or MOV BP, SP 16비트에서

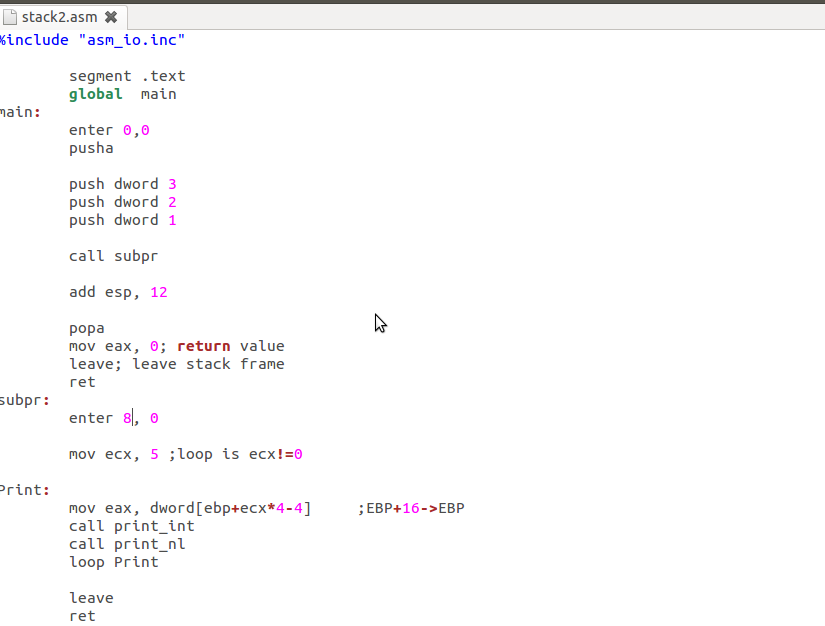
SUB ESP, operand1 ;or SUB SP, operand1 16비트에서

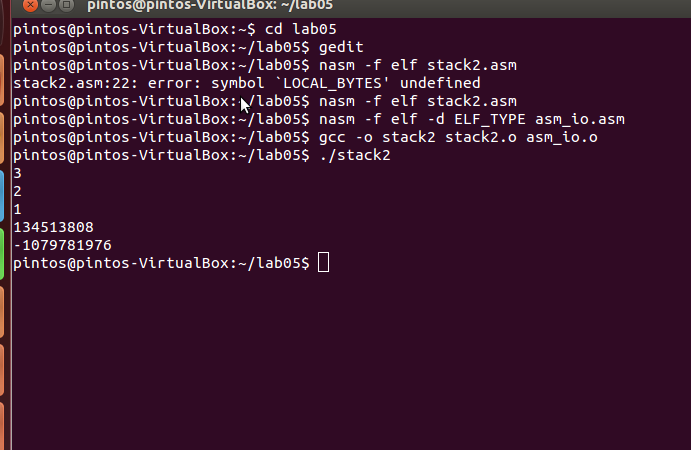
LEAVE

LEAVE 명령어는 ENTER 명령어에 의해 창조된 스택 프레임을 파괴한다.

MOV ESP, EBP(16비트에서는 MOV SP, BP)

POP EBP(16비트에서는 POP BP)

* + 1. 위 1.2에서 작성한 프로그램 stack.asm 에서 해당 부분을 enter 와 leave 명령어로 대체한 프로그램 stack2.asm 을 작성하시오. 이 프로그램을 어셈블, 링크하여 실행 파일 stack2를 생성하고, 이를 실행시킨 결과를 화면 캡쳐, 첨부하여 동일하게 동작하는지 확인하시오. 또한 stack.asm 과 stack2.asm 두 프로그램 중 어느 것이 더 효율적인지 생각해보고 아래에 의견을 적으시오.



Stack2.asm 이 더 효율적이다. 왜냐하면 ENTER, LEAVE 명령어가 고급 언어 명령어여서 컴퓨터의 관점에서 비휼적이지만 하드웨어용량과 성능이 좋아지면서 이러한 단점은 해결되었다.

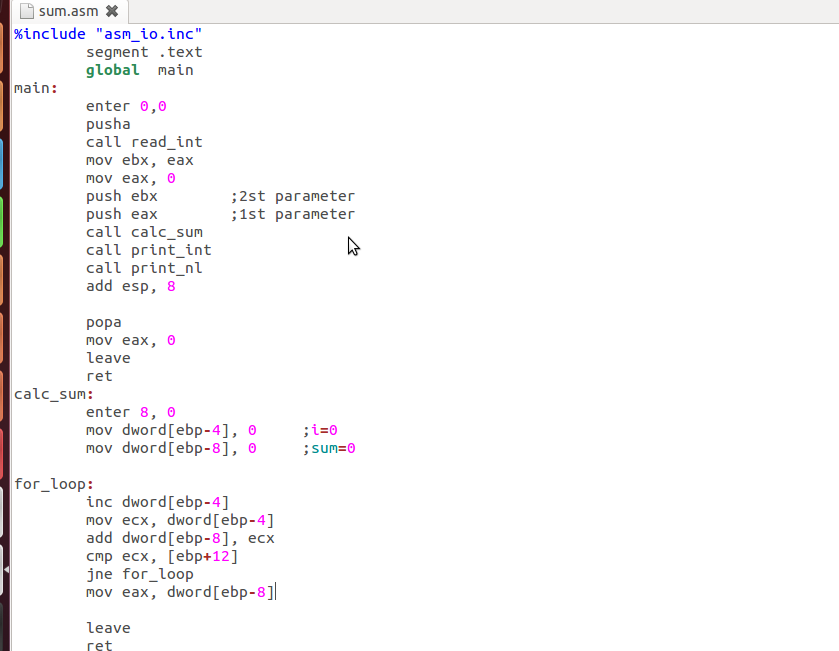
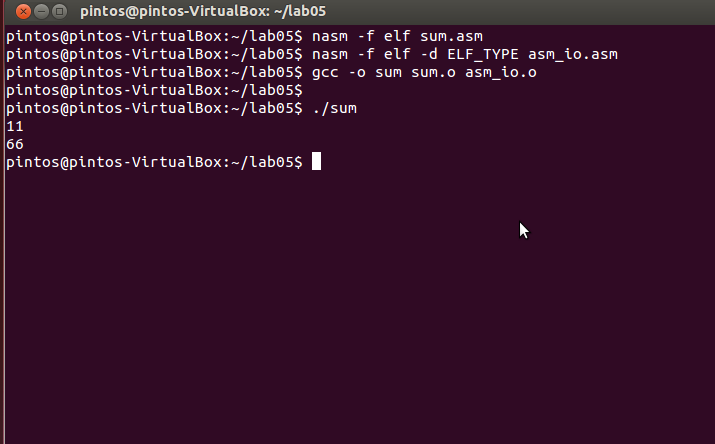
1. (sum.asm) 다음 조건을 만족하는 어셈블리 프로그램 sum.asm을 작성하시오. 이 프로그램을 어셈블, 링크하여 실행 파일 sum을 생성하고, 이를 실행시킨 결과를 화면 캡쳐하여 첨부하시오.

* 프로그램 sum 은 1 ~ n 의 정수들의 합을 계산하여 출력하는 프로그램이다. Sum 프로그램은 main 과 calc\_sum 두 개의 모듈로 구성된다.
* main 모듈은 (1) 사용자로부터 숫자 n 을 입력 받기, (2) 서브프로그램 calc\_sum 호출, (3) 결과값 출력 의 순서로 수행된다.
* calc\_sum 은 1 ~ n 의 덧셈을 계산하는 모듈이다. calc\_sum 은 main 에 의해 호출되며, 이 때 n 값과 합이 저장될 변수의 주소를 파라미터로 넘겨받는다. calc\_sum 은 수행 중에 두 개의 지역 변수를 사용하도록 한다 (아래 C 함수 참조)

void calc\_sum( int n, int \* sump)  
{  
    int i, sum= 0;

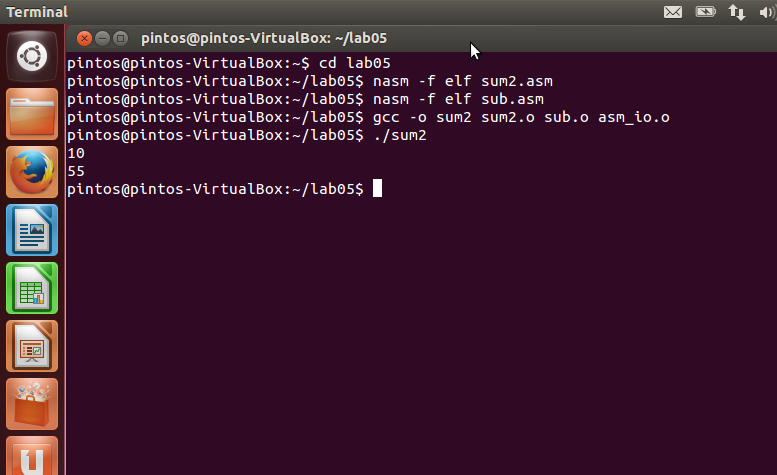
    for (i=1; i<=n; i++){  
        sum = sum + i;

\*sump = sum;  
}

* calc\_sum 호출 시의 calling convention은 반드시 수업 시간에 다룬 내용을 따라야 한다.
* 

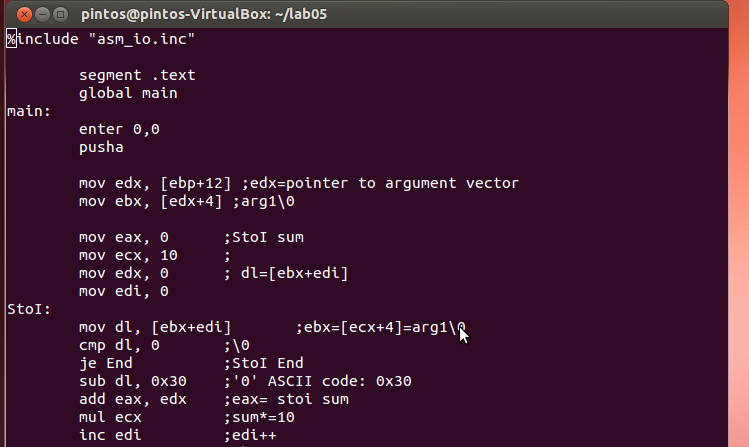
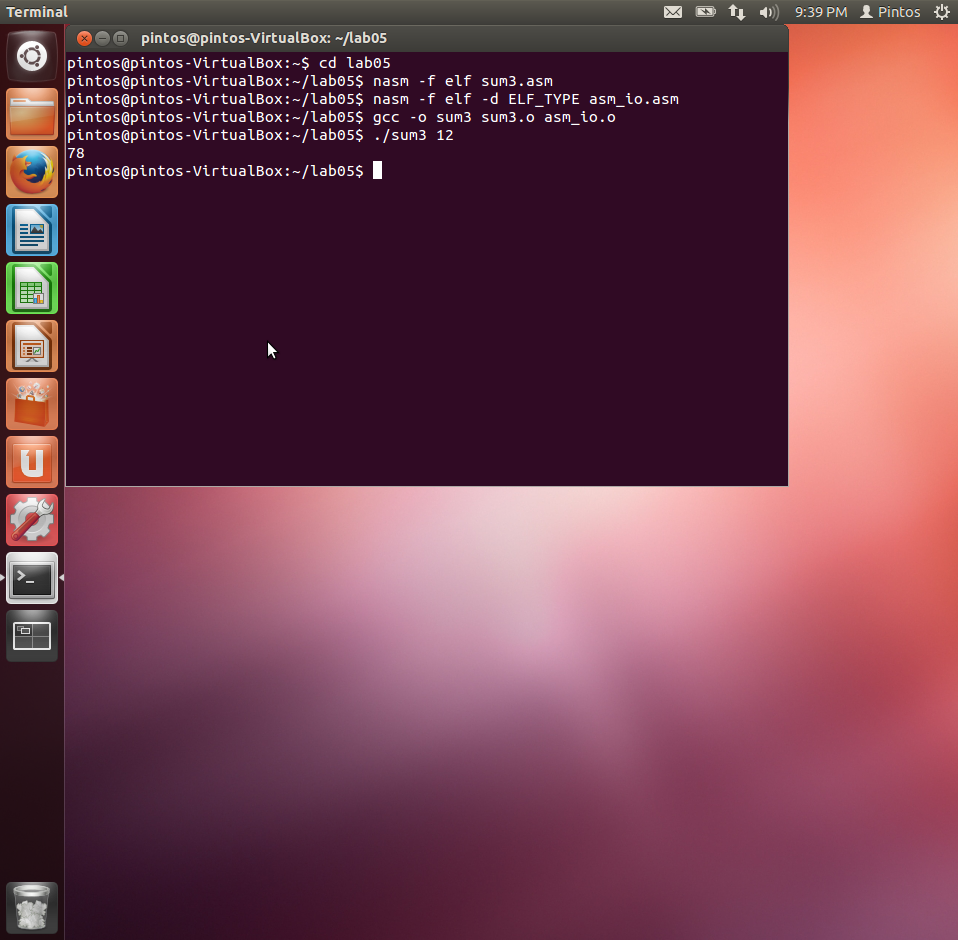
1. (sum2.asm/sub.asm) 위 2번의 “하나의” 소스 파일 sum.asm 에 정의된 두 개의 모듈 main 과 calc\_sum 을 각각 다른 소스 파일 sum2.asm 과 sub.asm 에 나누어 작성하시오. 두 어셈블리 소스 파일을 각각 어셈블하여 오브젝트 파일 sum2.o 와 sub.o 를 생성한 후, 이들을 링크하여 하나의 실행 파일 sum2 를 생성하시오.
   1. 링크 과정에서 실행 파일을 생성하기 위해 사용한 명령줄을 아래에 적으시오.

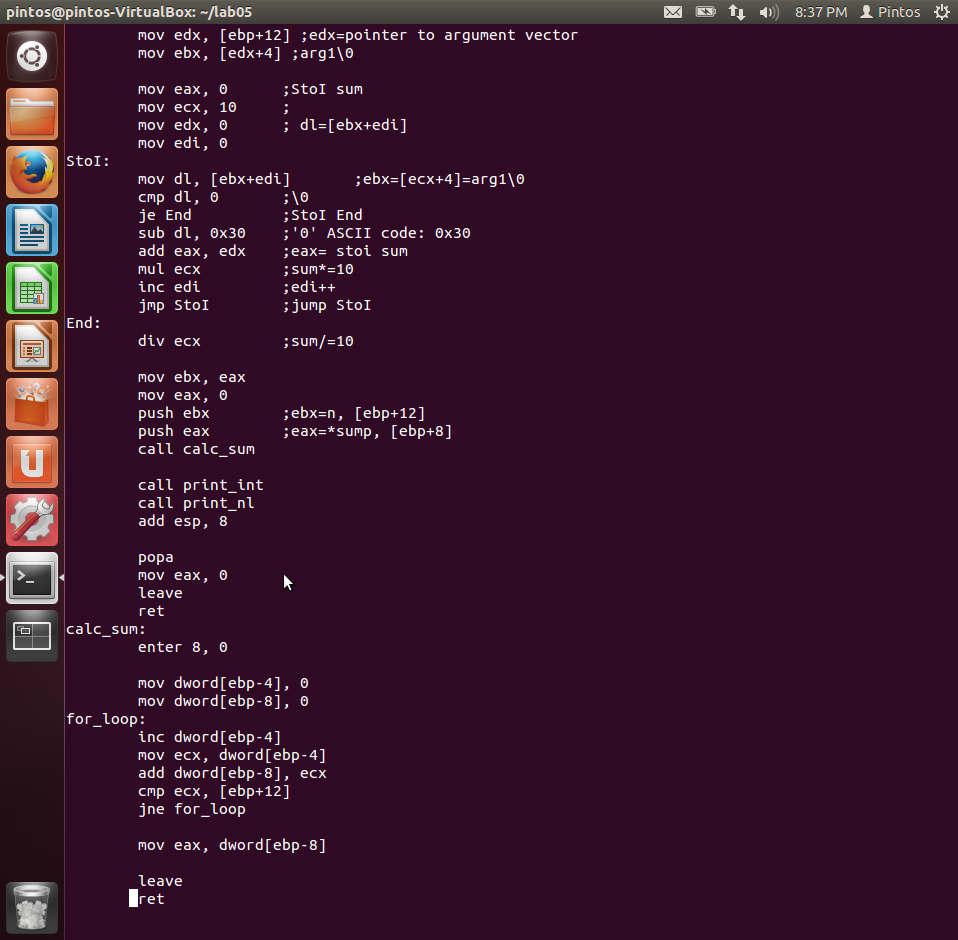
gcc -o sum2 sum2.o sub.o asm\_io.o

* 1. 프로그램 sum2 를 실행시킨 결과를 화면 캡쳐하여 첨부하고, 2번과 동일하게 동작하는지 확인하시오.

동일하게 작동합니다.

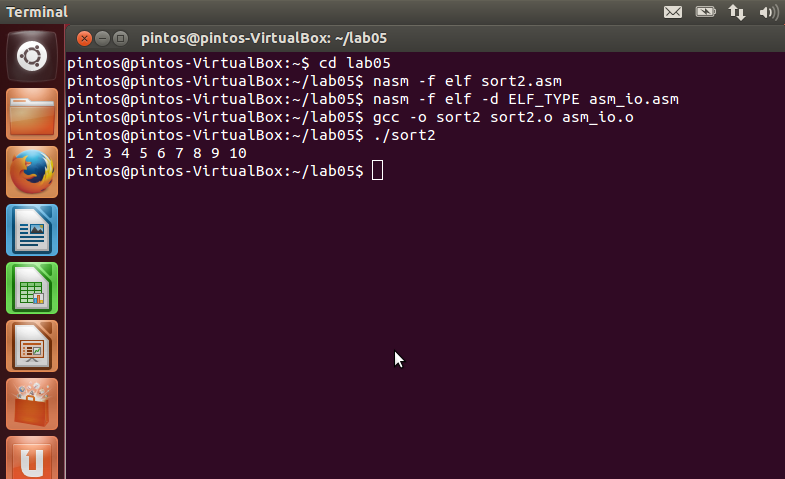
* 1. 두 개의 별도의 소스 파일로 나누는 경우 각 소스 코드에 추가되어야 할 어셈블리 지시어가 무엇인지 아래에 적으시오 (지난 실습 Lab2 의 3.2번에서 조사한 내용 참고). global, extern

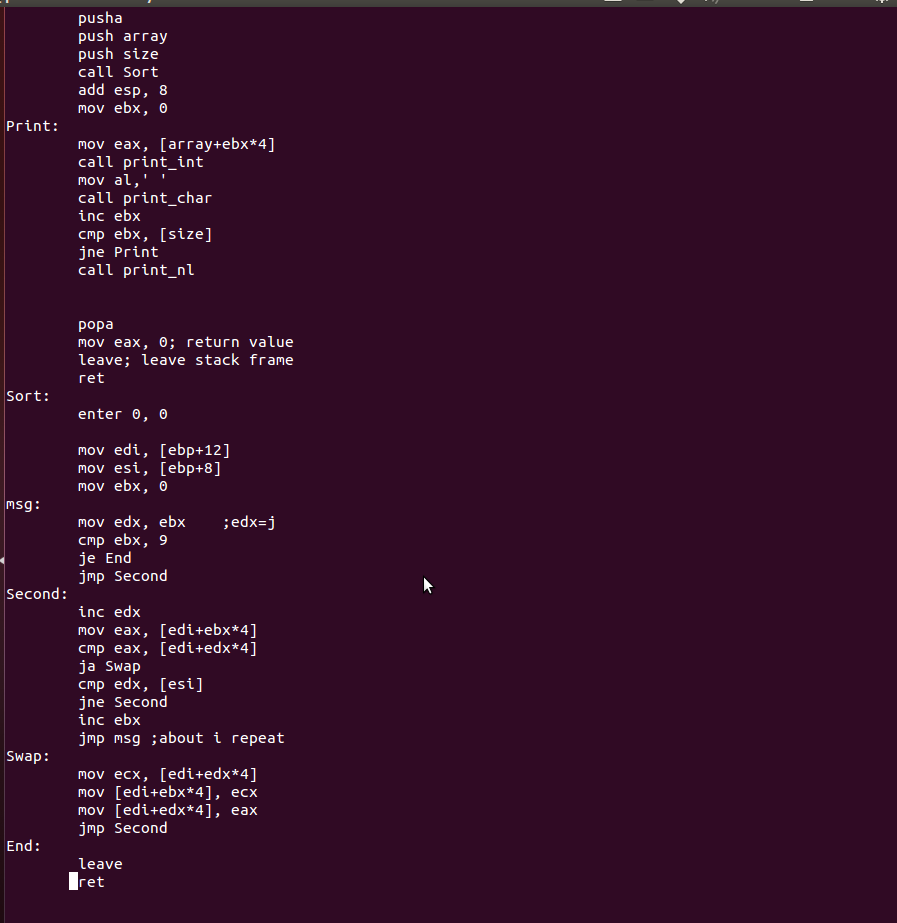
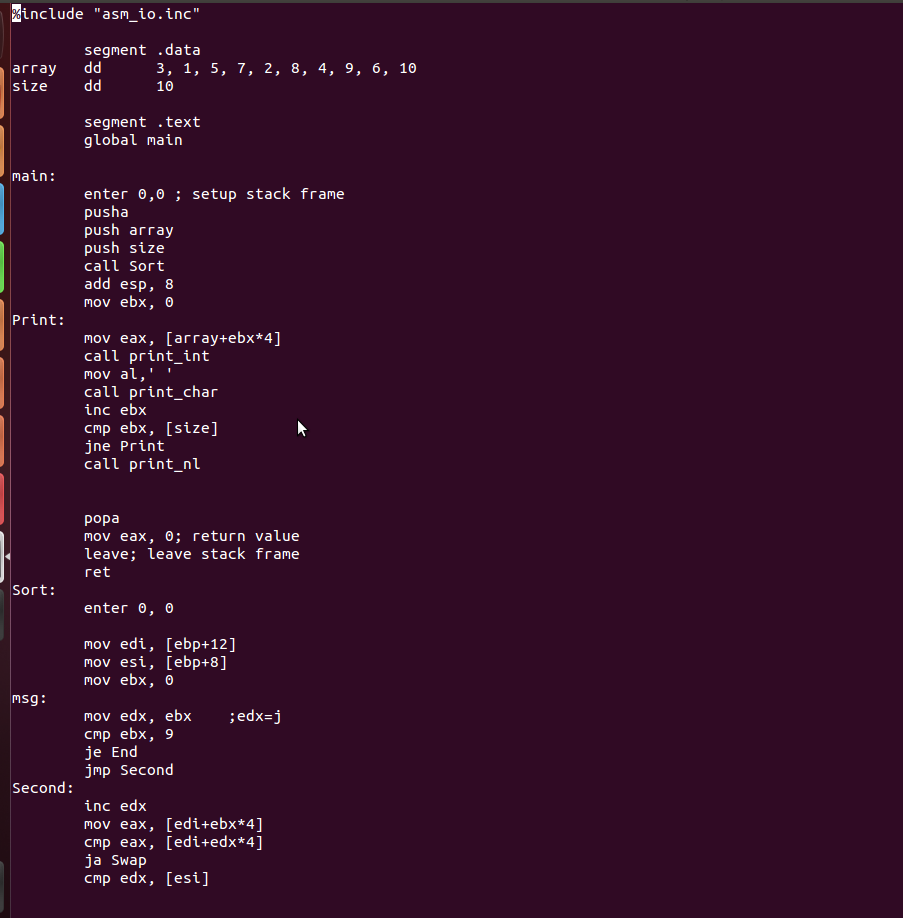
1. (sum3.asm) 위 3번의 프로그램 sum 을 수정하여, n을 프로그램 실행 후 사용자로부터 입력 받는 대신 명령줄에 명시하는 형태로 바꾼 프로그램 sum3 을 작성하고, 이를 실행시킨 결과를 화면 캡쳐하여 첨부하시오 (즉, 프로그램 sum3은 명령줄에서 “%sum3 10” 의 형태로 실행되어야 한다).



1. (sort2.asm) Lab4 에서 작성한 정렬 프로그램 sort.asm을 main 과 sort 두 개의 모듈로 이루어진 프로그램으로 수정하시오. 단 모듈 main 이 모듈 sort를 호출하는 형식을 취하도록 하며, 호출 시의 calling convention은 반드시 수업 시간에 다룬 내용을 따라야 한다.
   1. 모듈 sort 가 어떤 파라미터를 가지는지 아래에 적고, 간단히 설명하시오.

array와 size라는 파라미터를 가지고 있습니다. 왜냐하면 Sort 모듈(sub2.asm)에는 segment .data가 없어서 array와 size를 알 수 없기 때문이다..

* 1. 이 프로그램을 어셈블, 링크하여 실행 파일 sort를 생성하고, 이를 실행시킨 결과를 화면 캡쳐하여 첨부하시오.



끝.