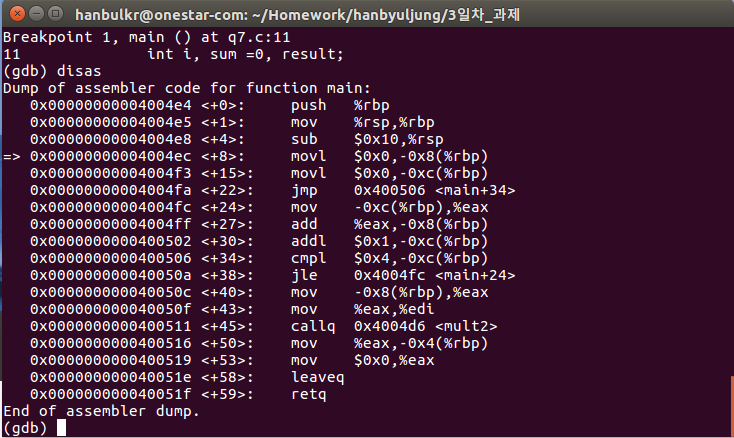
**<3일차 \_주말 숙제\_7번, 11 >**

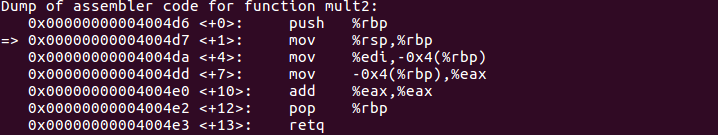
7. C로 함수를 만들 때, Stack이란 구조가 생성된다.

이 구조가 어떻게 동작하는지 Assembly Language를 해석하며 기술해보시오.

esp, ebp, eip등의 Register에 어떤 값이 어떻게 들어가는지 등등

메모리에 어떤 값들이 들어가는지 등을 자세히 기술하시오.





1. main 루틴으로 들어가며 rbp(stack의 기준점을 가리키는 범용레지스터)와

rsp(현 stack의 최상위점 주소 값을 가짐)가 잡힌다.

2. 처음 루틴인 push %rbp를 하여 스텍 최상위 주소에 rsp에 rbp 주소를 밀어 넣는다.

3. mov는 rsp의 값을 rbp에 값에 복사 한다.

4. sub 를 이용해 stack의 크기를 할당한다.

0x10 =16 byte를 빼는 모양새,

rsp에서 0x10을 빼면 밑으로 자르는 스텍은 최상위가 16byte를 확보한다.(이유는 밑에)

rbp 는 2.번에서의 값과 아직 같다.

5. movl (long type) , $0x0, -0x8(%rbp) 는 rbp로 부터 8byte 공간을 할당 받아 그 곳에0을 복사한다.

6. movl (long type) , $0x0, -0xc(%rbp) 는 rbp로 부터 12byte 공간을 할당 받아 그 곳에0을 복사한다.

7.jmp 0x400506 <main+34>로 이동(점프)하라는 뜻이다. (여기서 부터 반복문 시작)

8. move에서 eax값에 -0xc(%rbp) 값을 저장.

9. add 는 두인자를 더하고 뒤에 있는 인자에 초기화 즉 -0x8(%rbp) 에 eax 값을 넣는다.

10. add 는 두인자를 더하고 뒤에 있는 인자에 초기화 즉 -0xc(%rbp) 에 0x1 값을 넣는다.

11. cmp는 두 인자사이의 뺄셈을 하는 것이다.(이 결과를 이용해 플래그를 업데이트 한고 결과를 폐기함)

12. jle 는 jump (if) less (or) equal의 약자로 <=이며 이조건을 인자가 만족하는지 본다.

13. 밑에 callq는 push +jump이다. (함수 호출시에 사용을 한다.) push를 하면서 복귀주소를 저장하고

jump을 통해서 함수의 주소로 이동한다.

나머진 겹치는 부분이 상당수 이기에 생략.

**<리눅스 디버깅 하는 방법 정리>**

**---> gcc –g –O0 –o 생성이름(디버깅파일) 디버깅할c파일**

\_-g 옵션: 디버깅을 분석 할 수 있도록 미리 설정하는 옵션.

\_-O0 옵션: 영어 대문자 O ,숫자 0을 한 것으로 디버깅이 자동최적화가 되지

않게 컴파일 한다.(최적화 되면 디버깅시 너무 왔다갔다.해서 확인이 어려움)

\_-o 옵션: c언어프로그램 컴파일 시, 이름을 지정하기 위한 옵션.

**<gdb 명령어 정리>**

1. gdb 실행파일 : 실행파일을 디버깅 모드(gdb)로 집입시킨다.

2. b main : main에 break 포인트를 걸어 둔다. (b는 브레이크 포인트 설정하는 것)

3. r : run debug를 run하여 debugging을 확인을 시작 시킨다.

4. disas : display assembly 어셈블리를어(기계어로)된 현재 debug 상태를 보여준다.

5. b\* 주소값 : 주소값을 이용해 break point를 추가 할 수 있다.

6. info b : 브레이크 포인트를 확인 할 수 있다.

7. q : 디버깅 끝내기.

8. p/x $rsp : 16 진수의 주소를 보기위해 사용,(rsp의 16진수 주소를 보여준다.)

9. p/c : 문자형을 보기위해 사용

10. p &변수 : 변수 하나의 값을 보여준다.

11. l : 엘은 디버깅 하는 함수 원본 소스코드를 보여준다.

12. ni : (next)어셈블리 소스 한줄 실행 (함수 호출도 한줄)\_\_(n3: 이러면 3줄 실행)

13. si : (step)어셈블리 소스 한줄 실행 (함수 호출 같이, 함수 루틴안으로 들어감)

14. c : 브레이크 포인트를 만날 때까지 계속 진행한다.

15. bt : 오류가 발생한 함수를 역으로 찾아간다.

<디버깅 하는 방법>

1. 먼저 gcc로 c파일을 컴파일 하여 디버깅할 실행 파일을 만든다.

2. b 명령어를 사용해서 break point를 걸어둔다.

3. r을 이용해서 한 번 debugging 실행 시켜 최신화 한다.

4. disas를 통해 현재 기계어 진행 상태를 display 한다.

5. b \*를 이용하여 break pointer를 맨위 주소를 추가하거나 보고 싶은 부분에 break pointer를 걸어주어 확인을 시작한다.

6. info b 를 통해 브레이크 포인팅된 값을 확인해 본다.

7. 다시 r을 눌려 최신화를 시켜준다.

8. disas 로 진행 상태를 확인한다.

9. 어셈블리 명령어에 따라 값이 이동 복사 초기화 등의 상태를 p/x , p, x 등을 통해 확인 해 본다.

10. 다음 단계 시 변화를 보기 위해 si, ni를 이용하여 한줄을 실행 한다.

11. 8~10 과정을 반복하면서 어셈블리 명령어에 따라 범용 레지스터와 그 값들이 제대로 움직이고 있는지를 확인한다. ( 레지스터 한번에 보기 : info register)

12. 모든 과정이 끝났으면 q 를 통해 gdb 모드를 나올 수 있다.