**2일차 어셈블리/ 2/22(목)**

\* 32비트 버전

% eax (extended accumulator register)

% ecx (counter reg)

% edx (data reg) caller save register

% ebx (base reg)

% edi (destination index)

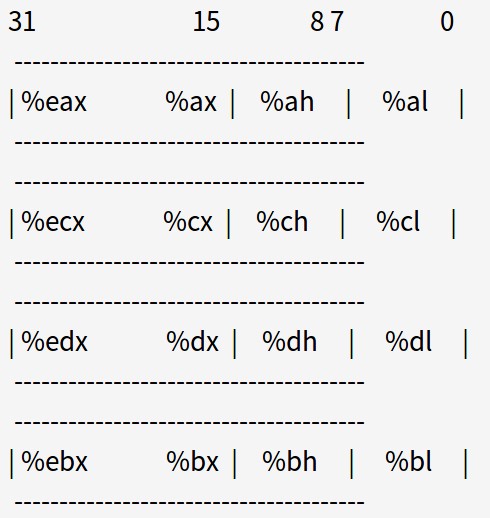
% esi (source Index) callee save register

% ebp (base pointer)

% esp (stack pointer) program stack 관리하는데 사용

\* 64 비트 버전은 r이 붙는다.

rax , rcx, rdx, rbx, rdi, rsi 등

일반적으로 레지스터는 32 비트 크기

그 중 eax, ecx, edx, ebx 는 이전 cpu 와 호환하기 위해 왼쪽 그림과 같은 구조이다.

**\* 스택프레임 구조**

프로시저는 함수라 지칭. 즉 Procedure call 은 function call, 같은 의미이다.

lA32 프로그램들은 procedure call 을 지원하기 위해서 프로그램 스택을 사용한다.

스택은 프로시저의 argument 들을 전달하거나, return information을 저장하거나, 마지막 저장된 정보를 레지스터로 전송하거나, 로컬 저장장치로 쓰이곤 한다. 여기서 하나의 procedure call 에 할당된 영역을 스택 프래임( stack frame ) 이라 고 부르며, 그 구조는 아래와 같다

**\* 스택 어디에 쓰이나?**

레지스터는 메모리 계층 구조를 봤을 때, 용량은 작고 속도는 빠르다. 레지스터 수천만개를 쓰면 참 좋겠지만, 단가가 너무 높아진다. 용량문제와 단가를 해결하는 메모리라는 녀석이 있다. 레지스터가 부족할 때 메모리에 값을 잠깐 저장하는 용도로 사용한다는 개념에서 stack을 만들게 된다.

**\* Stack**

스택은 값이 쌓이면 –의 주소를 갖고, 값이 빠지면 +의 주소를 갖는다.

\* 2진수 16진수

생략.

**\* 기계어 분석**

#includ <stdio.h>

Int func(int i) {return i+3;}

Int main(void){

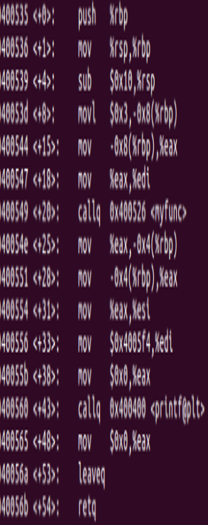
Int a =3, res;

res = func(a);

printf(“after func value is %d\n”,res);

return 0;

}

* gdb(디버그) 후 disas(디어셈블리) 코드확인

**1. push**: sp에 쌓아라, 이전함수의 base pointer에 저장

**2. mov**: rsp를 rbp에 옮기세요. 스택프레임 생성전 잔여물을 치워줌.

**3. sub**: 0x10을 rsp에서 빼세요.

**4. movl**: 0x03을 (rbp-8)주소로 이동. 복사의 개념.

**5. mov**: eax로 3(-0x8(%rbp))를 이동

**6. mov**: edi로 3(eax)를 이동.

**7. Callq주소,함수**: 주소를 호출하여 이 주소에 있는 명령을 실행하도록 한다. 그리고 , rsp에 주소값(0x400526)을 쌓아라 push+jmp(함수시행)

**15. Leaveq** push %ebp, mov %esp,%ebp와 같은 명령을 수행하게 되는데 이것을 함수 프롤로그 과정이라고 합니다. 이과정에서 stack frame을 맨 처음에 저장해 두고 mov %esp, %ebp 과정에서 다른 stack frame을 만들게 됩니다. leave 명령어에서는 함수 프롤로그와 반대로 mov %ebp, %esp, pop %ebp 과정을 거쳐서 stack frame 의 주소를 다시 바꿔준 후 pop 명령에서 base pointer를 복구하게 되는데, pop 명령으로 push 로 인해 줄어들었던 4byte를 다시 확장 시키게 됩니다. 이 과정을 함수 에필로그 과정이라고 합니다

**16. Retq**: pop rip 현재 값 유지하고, rip(복귀주소)로 돌아가라.

rbp2

rsp 최상위 값

OS 공간

rbp1

3

Main

Func

Mov 0x3

rsp1 – 8

0x08

Callq 0x4000526 <func>

0x02

rbp3

0x04

rbp1 🡪 push+jup 함수떄리고 주소로 복귀