HOMEWORK

이름	문지희
날짜	2018/2/22
수업일수	2일차
담당교수	이상훈

목차

0. 용어정리

-메모리

: 컴퓨터에 사용되는 장치 중 저장기능을 가지고 있는 것들
중앙처리장치(CPU)로부터 가까운 레지스터, 캐시, 메모리(RAM과 ROM),
하드디스크가 있다.
메모리는 레지스터-캐시-메모리-디스크 순으로 속도가 빠르고,
디스크-메모리-캐시-레지스터 순으로 용량이 크다.

-범용 레지스터

: 범용 레지스터란 CPU내에 있는 레지스터 중 계산결과의 임시저장,
산술 및 논리 연산, 주소 색인 등 여러 목적으로 사용될 수 있는 레지스터이다.
: x86 범용 레지스터
1) ax: 함수의 return 값을 저장한다.
2) cx: 무언가를 반복하고자 할 때 사용한다
3) bp: 스택의 기준점
4) sp: 스택의 최상위점
5) ip: 다음에 실행할 명령어의 주소

-스택(stack)

: 지역변수가 저장되는 장소
스택은 값이 쌓이면 -의 주소를 가지고 값이 빠지면 +의 주소를 가지게 된다.
스택을 제외한 나머지는 쌓이면 + 빠지면 -의 주소를 가진다.

: 스택의 사용이유
레지스터는 속도가 빠르지만 용량이 작아 빠른 성능을 가지도록 여러 레지스터를
사용하기엔 비용면에서 문제가 있다. 빠른 속도와 많은 용량을 가지는 조건을
충족시키기 위해 레지스터에 저장할 수 없는 정보를 메모리에 값을 잠깐
저장했다가 필요할 때 부르는 개념으로 스택이 만들어졌다.
-ALU(산술 논리 장치)
중앙처리장치 속에서 산술연산, 논리연산 및 시프트(shift)를 수행하는 회로장치이다.

1. 기계어 분석

*프로그램 디버깅 및 어셈블리어 분석을 위한 명령어와 과정

1) 디렉토리 만들기 및 사용할 저장소 다운(github사용)

mkdir my_proj

cd my_proj

git clone https://github.com/SHL-Education/Homework.git

2) pwd를 입력해 현재 위치를 확인하고 cd ~/my_proj/Homework/sanghoonlee로 이동

3) 최적화를 못하게 하는 디버깅 실행

'gcc -g -O0 -o debug func1.c'

4) ls를 입력하여 debug가 존재하는지 확인

5) 'gdb debug'를 입력하여 디버거를 킴

6) 'b main'을 입력하여 main 함수에 breakpoint(정지)를 걸어준다.

7) 'r'을 눌러 실행하고 disas를 입력하여 어셈블리어 코드를 확인한다.

8) 'p/x $rip'를 입력해 실행할 명령어의 주소를 알고, 'b *복사한주소'를 적은 후 r을 눌러 실행하여 =>이 첫번째로 오게 한다.

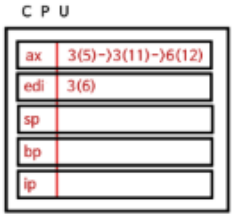
-소스코드

```
#include<stdio.h>
intmyfunc(int num)
{
return num + 3;
}
intmain(void)
{
int num = 3, res;
res = myfunc(num);
printf("res = %d\n", res);
return0;
}
```

-기계어 분석

```
=> 0x0000555555554659 <+0>:   push  %rbp  (1)
   0x000055555555465a <+1>:   mov   %rsp,%rbp  (2)
   0x000055555555465d <+4>:   sub   $0x10,%rsp  (3)
   0x0000555555554661 <+8>:   movl  $0x3,-0x8(%rbp)  (4)
   0x0000555555554668 <+15>:  mov   -0x8(%rbp),%eax  (5)
   0x000055555555466b <+18>:  mov   %eax,%edi  (6)
복귀주소 0x000055555555466d <+20>:  callq 0x55555554464a <myfunc>  (7)
   0x0000555555554672 <+25>:  mov   %eax,-0x4(%rbp) (15)
   0x0000555555554675 <+28>:  mov   -0x4(%rbp),%eax
   0x0000555555554678 <+31>:  mov   %eax,%esi
   0x000055555555467a <+33>:  mov   $0x4005f4,%edi
   0x0000555555554681 <+40>:  mov   $0x0,%eax
   0x0000555555554686 <+45>:  callq 0x555555554520 <printf@plt>
   0x000055555555468b <+50>:  mov   $0x0,%eax
   0x0000555555554690 <+55>:  leaveq
   0x0000555555554691 <+56>:  retq
```

```
=> 0x000055555555464a <+0>:   push  %rbp  (8)
   0x000055555555464b <+1>:   mov   %rsp,%rbp  (9)
   0x000055555555464e <+4>:   mov   %edi,-0x4(%rbp)  (10)
   0x0000555555554651 <+7>:   mov   -0x4(%rbp),%eax  (11)
   0x0000555555554654 <+10>:  add   $0x3,%eax  (12)
   0x0000555555554657 <+13>:  pop   %rbp  (13)
   0x0000555555554658 <+14>:  retq  (14)
```



2. 포인터 크기

8 비트 시스템의 경우 1 byte
16 비트는 2 byte
32 비트는 4 byte
64 비트는 8 byte

컴퓨터의 중앙처리장치에 있는 ALU의 연산은 범용 레지스터에 종속적이다. 따라서 컴퓨터가 64비트일 때 이들은 64 비트로 구성되어 있음을 의미한다.
변수와 포인터는 메모리에 정보나 주소를 저장하는 공간인데 만약 메모리의 크기가 작다면 주소를 표현할 방법이 없어 ALU의 값이 나올 수 있는 최대치인 64비트(8byte)가 포인터의 크기가 된 것이다.

실험 1
1) 'vi pointer_size.c'입력

2) 아래의 소스코드 입력

```
#include <stdio.h>

int main(void) {
        printf("sizeof(int *) = %lu₩n", sizeof(int *));
        printf("sizeof(double *) = %lu₩n", sizeof(double *));
        printf("sizeof(float *) = %lu₩n", sizeof(float *));
        return 0;
}
```

3) 결과 값 분석
결과가 전부 8이 되는 것을 확인했는데, 이는 포인터의 크기가 8byte이면 자료형에 관계없이 8byte라는 것을 알 수 있다.

3. 2진수, 16진수 변환

16진수와 2진수는 컴퓨터가 사용하는 진수 시스템이다. 16진수 시스템은 컴퓨터가 기계어를 사용하면서도 인간이 상대적으로 쉽게 볼 수 있는 진수이기 때문에 채택하여 사용한다.

2진수 - 0, 1
10진수 - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
16진수 - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f

-진수 변환법

144의 각 진수 변환

10진수	분해	$144 = 1\times10^2 + 4\times10^1 + 4\times10^0$
	10진수	144
2진수	분해	$144 = 2^7 + 2^4$
	2진수	$1001\,0000$
16진수	분해	$144 = 9\times16^1$
	16진수	$0X90$
	2진수 표현	$1001\,0000$

예제 1)
$33(10)$
$\rightarrow 33 = 2^5 + 1 \equiv 10\,0001$
$\rightarrow 33 = 2\times16 + 1 = 0X21$

예제 2)
$2568(10)$
2진수 $\rightarrow 2568 = 2^{11} + 2^9 + 2^3 = 1010\ 0000\ 1000$
16진수 $\rightarrow\ 2568 = 1010\,0000\,1000 = 0XA08$

예제 3)
$0X48932110(16)$
2진수 $\rightarrow$ $0100\ 1000\ 1001\ 0011\ 0010\ 0001\ 0001\ 0000$