Chungnam National University
Department of Computer Science and Engineering

2010년 가을학기 **김 형신/조현우**

중간고사

2010년 10월 20일
시스템 프로그래밍

분반/학번	반
이름	

문제	배점	점수
1	20	
2	20	
3	10	
4	20	
5	20	
6	20	
총계	110	

나는 이 답안을 부정행위 없이, 내 스스로의 힘으로 작성하였으며, 다른 학생의 것을 보거나, 다른 학생에게 보여주지 않았음을 맹세합니다. ( )

문제 1. 기초지식 (20점)

1) (5점) CPU의 인스트럭션 싸이클 4단계를 쓰시오.

fetch-decode-execution-store

선입-해독-실행-저장

2) (5점) 다음과 같은 C 프로그램에서 s 계산 시에 오버플로우가 발생하는 조건을 쓰시오.

```
int s, u, v;
s = u + v;
```

u, v < 0, s >= 0 (음의 오버플로우)

u, v > 0, s < 0 (양의 오버플로우)

3) (5점) **다음은 Little Endian 머신에서 실행코드를 어셈블리 코드로 디스어셈블한 것이다. 빈칸에 들어갈 코드를 채우시오. (단 본 머신은 32-bit워드 크기를 따른다.)**

Address	Instruction Code	Assembly Rendition
8048337:	81 ec ①b8 01 00 00	sub $0x1b8, %esp
804833d:	8b 85 58 fe ff ff ff	mov ②0xfffffe(%ebp), %eax

4) (5점) IA32 어셈블리 명령어인 call label 을 실행할 때 어떤 일들이 일어나는지 설명하시오.

스택에 리턴 주소를 저장하고 label 로 점프한다.(eip에 label의 주소를 써준다)

문제 2. (20점)[Floating point 표시] 3비트 exp 필드와 2비트 frac 필드로 구성된 (1비트 부호) IEEE 인코딩을 따르는 6비트 Floating point 시스템을 구현한다고 하자. 아래 표의 빈 칸을 모두 채우시오. 십진수를 floating 으로 변환하는 과정에서 근사(rounding)를 해야 한다. 근사를 한 경우에는 근사 전의 참값과 근사후의 값을 모두 표시해야 한다.

Description	Decimal	Binary Representation
Bias		-----
Smallest positive number		
Lowest finite		
Smallest positive normalized		
-----	$-\frac{7}{16}$	
-----	$\frac{5}{4}$	
-----		1 010 01
-----	13	

Description	Decimal	Binary	
Bias	3	------	
Smallest Positive number	1/16	000001	
Lowest finite	-14	111011	
Smallest positive normalized	1/4	000100	
------	-7/16	100111	-(7/4 * 1/4)
------	5/4	001101	5/4 * 1
------	-5/8	101001	-(5/4 * 1/2)
------	13(12)	011010	

13 => 1101 =>1.101 * 2^3, 그런데, frac 이 2 비트이므로, 근사가 필요
1.101 * 2^3 => 1.10 * 2^3 (인접 짝수 근사) => 12 가 됨
exp = 3 + bias = 110, frac = 10 (정규화 표현)

문제 3. (10점) [어셈 기초] 레지스터와 메모리에는 다음과 같은 값이 들어있다. 다음의 각 라인을 실행한 후의 스택과 레지스터값을 아래 표에 채우시오.

Register	Value
%eax	0x100
%ebx	
%edx	0x3
%esp	0x100

Address	value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x08

```
addl    (%eax, %edx, 4), %eax        # (1)
popl    %ebx                         # (2)
```

1) 변하는 값만 표시하였음

Register	Value
%eax	0x108
%ebx	
%edx	
%esp	

Address	Value
0x100	
0x104	
0x108	
0x10c	

2) 변하는 값만 표시하였음

Register	Value
%eax	
%ebx	0xFF
%edx	
%esp	0x104

Address	Value
0x100	
0x104	
0x108	
0x10c	

문제 4. (20점) [루프의 구현] 양수 x의 factorial을 구하는 프로그램을 어셈블리어로 작성하고자 한다.

(1) (5점) C 언어로 함수 int fact(int x)를 do while 구조로 작성하시오.

```
int fact_do(int x)
{
  int result = 1;
  do {
    result *= x;
    x = x-1;
  } while (x > 1);
  return result;
}
```

(2) (5점) (1)번의 C 프로그램을 goto 문으로 바꾸시오.

```
int fact_goto(int x)
{
  int result = 1;
loop:
  result *= x;
  x = x-1;
  if (x > 1)
    goto loop;
  return result;
}
```

(3) (10점) (2)번의 C 프로그램을 어셈블리어로 바꾸시오. (주. setup 부분과 finish 부분을 모두 포함해서 작성하시오)

```
_fact_goto:
    pushl %ebp                          # Setup
    movl %esp,%ebp                      # Setup
    movl $1,%eax                        # eax = 1
    movl 8(%ebp),%edx                   # edx = x
L11:
    imull %edx,%eax                     # result *= x
    decl %edx                           # x--
```

```
cmpl $1,%edx                    # Compare x - 1
jg L11                          # if > goto loop
movl %ebp,%esp                  # Finish
popl %ebp                       # Finish
ret                             # Finish
```

문제 5. (20점) [switch 문] 다음의 함수 switch_prob() 에서 switch 문의 본체 부분을 일부러 삭제하였다.

```
1   int switch_prob(int x, int n)
2   {
3       int result = x;
4
5       switch(n) {
6           /* Fill in code here */
7       }
8       return result;
9   }
```

위 함수를 컴파일 한 후에 디스어셈블 한 결과는 다음과 같다.

```
1   08048420 <switch_prob>:
2    8048420:  55                      push   %ebp
3    8048421:  89 e5                   mov    %esp,%ebp
4    8048423:  8b 45 0c                mov    0xc(%ebp),%eax
5    8048426:  83 e8 28                sub    $0x28,%eax
6    8048429:  83 f8 05                cmp    $0x5,%eax
7    804842c:  77 07                   ja     8048435 <switch_prob+0x15>
8    804842e:  ff 24 85 f0 85 04 08    jmp    *0x80485f0(,%eax,4)
9    8048435:  8b 45 08                mov    0x8(%ebp),%eax
10   8048438:  eb 24                   jmp    804845e <switch_prob+0x3e>
11   804843a:  8b 45 08                mov    0x8(%ebp),%eax
12   804843d:  8d 76 00                lea    0x0(%esi),%esi
13   8048440:  eb 19                   jmp    804845b <switch_prob+0x3b>
14   8048442:  8b 45 08                mov    0x8(%ebp),%eax
15   8048445:  c1 e0 03                shl    $0x3,%eax
16   8048448:  eb 17                   jmp    8048461 <switch_prob+0x41>
17   804844a:  8b 45 08                mov    0x8(%ebp),%eax
18   804844d:  c1 f8 03                sar    $0x3,%eax
19   8048450:  eb 0f                   jmp    8048461 <switch_prob+0x41>
20   8048452:  8b 45 08                mov    0x8(%ebp),%eax
21   8048455:  c1 e0 03                shl    $0x3,%eax
22   8048458:  2b 45 08                sub    0x8(%ebp),%eax
23   804845b:  0f af c0                imul   %eax,%eax
24   804845e:  83 c0 11                add    $0x11,%eax
25   8048461:  5d                      pop    %ebp
26   8048462:  c3                      ret
```

점프 테이블이 0x80485f0에 위치하고 있어서 gdb 실행후 다음과 같은 명령을 실행했더니, 그 출력이 아래와 같았다.

```
(gdb) x/6w 0x80485f0
0x80485f0: 0x08048442 0x08048435 0x08048442 0x0804844a
0x8048600: 0x08048452 0x0804843a
```

이 정보를 이용해서 위 C 함수의 switch 문의 본체를 작성하시오.

```
1  int switch_prob(int x, int n)
2  {
3      int result = x;
4
5      switch(n) {
6      case 40:
7      case 42:

8          result <<= 3;
9          break;
10     case 43:
11         result >>= 3;
12         break;
13     case 44:
14         result *= 7;
15         /* Fall through */
16     case 45:
17         result *= result;
18         /* Fall through */
19     default:
20         result += 17;
21     }
22
23     return result;
24 }
```

문제 6. (20점)[프로시져+ 최적화] 다음은 C 함수 mystery를 컴파일한 어셈블리 코드를 보여준다.

```
08048334 <mystery>:
 8048334: 55                    push   %ebp
 8048335: 89 e5                 mov    %esp,%ebp
 8048337: 83 ec 0c              sub    $0xc,%esp
 804833a: 8b 45 08              mov    0x8(%ebp),%eax
 804833d: c7 45 fc 00 00 00 00  movl   $0x0,0xfffffffc(%ebp)
 8048344: 3b 45 fc              cmp    0xfffffffc(%ebp),%eax
 8048347: 75 09                 jne    8048352 <mystery+0x1e>
 8048349: c7 45 f8 00 00 00 00  movl   $0x0,0xfffffff8(%ebp)
 8048350: eb 12                 jmp    8048364 <mystery+0x30>
 8048352: 8b 45 08              mov    0x8(%ebp),%eax
 8048355: 48                    dec    %eax
 8048356: 89 04 24              mov    %eax,(%esp)
 8048359: e8 d6 ff ff ff        call   8048334 <mystery>
 804835e: 03 45 08              add    0x8(%ebp),%eax
 8048361: 89 45 f8              mov    %eax,0xfffffff8(%ebp)
 8048364: 8b 45 f8              mov    0xfffffff8(%ebp),%eax
 8048367: c9                    leave
 8048368: c3                    ret
```

1) (10점) 다음의 mystery 함수의 빈 곳을 채우시오.

```
int  mystery(int  i)
{
  if (_______________) return _____________________;

  return _____________________;
}
```

```
int mystery(int i)
{
  if ( i!= 0 ) return i +  mystery(i-1);
     return i;
}
```

2) (10점) "Peephole 최적화" 기법은 몇 개의 인스트럭션들을 참조해서 코드를 최적화(인스트럭션의 갯수를 줄임)하는 방법이다. 최적화를 한 뒤에는 기존의 코드와 결과가 동일해야 한다. 0x804833d와 0x8048344에 저장된 두 개의 인스트럭션을 최적화하시오.

cmp $0x0, %eax