Chungnam National University
Department of Computer Science and Engineering

2011년 가을학기

중간고사 해답

2011년 10월 19일
시스템 프로그래밍

분반/학번	반
이름	

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

채점 기준 : 의미만 맞으면 5점. 일부 틀리면 3점

2) (5점) 0x12345678 을 Little Endian 컴퓨터에서 0x800100 번지부터 저장한다면, 0x800101 번지에 저장되는 값은 얼마인가?

0x56

채점 기준 : 정확히 맞아야 5점. 0x와 같이 16진수 표시 없이 그냥 78 쓰면 1점 감점

3) (5점) 왜 GCC 컴파일러는 함수 내에서 필요한 스택 공간 보다 더 큰 크기의 스택 공간을 할당하는지 그 이유를 설명하시오.

alignment 요건으로 인해서, 또는 스택 접근의 효율성을 높이기 위해

채점 기준 : 다른 이유는 0점. 비슷하면 3점

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

eip 레지스터에 lable의 주소를 기록하고, esp레지스터가 가리키는 스택 탑에 리턴 주소를 push한다.

채점기준 : eip와 esp에 대한 두 가지 설명 모두 맞으면 5점. 하나만 맞으면 3점

문제 2. (20점)[Floating point 표시] IEEE 부동소숫점 표시방법을 이용하여 8비트로 아래와 같이 부동소숫점을 표시하는 컴퓨터가 있다고 가정하자.

s	exp	frac

s : 부호 비트, 1비트
exp : 지수, 4비트
frac : 유효숫자, 3비트

위의 구조를 갖는 경우에, 어떤 부동소수 f는 $(-1)^s*M*2^E$로 계산할 수 있으며, E = exp - bias, M = $1.xxxx_2$로 표시한 경우 xxxx 를 frac으로 갖는다. 위와 같은 시스템에서 bias = 7이 된다. exp가 0이 아니고, 0x1111 도 아닌 경우에 정규화 표시방식을 따르게 된다. 이 경우 다음 질문에 답하라. exp 가 0인 경우는 비정규화 방식을 따르며, 이 경우 E = 1-bias, M=0.xxxxx 에서 xxxx 가 frac 이 된다.

1) (4점) 이 형식을 따르는 정규화 방식으로 표시한 양수 최소값은 8비트로 00001000 이다. 이 값이 십진수로 얼마인지 계산하시오(분수로 표시). 계산 과정이 드러나야 함.

E = 1 - 7 = -6, M = 1.0000, S = 0
$f = (-1)^0*1.0*2^{-6} = 8/8*1/64 = 8/512 = 1/64$

채점기준 : 수식이 제시되고 맞으면 4점. 수식이 틀리거나 답만 맞으면 2점

2) (4점) 위 IEEE 부동소수 표시 방법을 사용해서 비정규화 방식으로 표시되는 양의 최소값을(0에 가장 가까운 값) 표시하시오. (위의 8비트 형식으로 표시)

비정규화 방식은 exp가 0이어야 한다. 따라서 0이 아닌 양의 최소값은
0 0000 001

채점기준 : 설명 없이 값이 정확히 맞으면 4점. 그 이외의 경우는 0점

3) (4점) 위 2)번의 이진수를 십진수로 변환하여 분수로 나타내시오.

E = 1 - 7 = -6, M = 0.001, S = 0
1/8*1/64 = 1/512

채점기준 : 답이 맞으면 4점. 그 이외의 경우는 0점

4)(4점) 십진수 63.0을 위 8비트 IEEE 부동소숫점 표시로 나타내려고 할 때, 63.0을 $(-1)^{s}*M*2^{E}$ 형태로 표시할 때, M과 E를 각각 나타내시오. 단, M은 1.xxxxxx 형태로 표시해야 함.

63 = 00111111 = 1.11111×2^{5}
M = 1.11111, E = 5

채점기준 : M, E 모두 맞으면 4점. 나머지 경우는 모두 0점

5)(4점) 4)번 문제의 답을 3비트 frac 에 맞출 수 있도록 짝수 근사(round to even)한 후에 최종 정규화한 결과의 frac 필드와 exp 필드 값을 나타내시오. 과정 표시할 것

frac 부분의 길이가 3비트이므로, 1.111 인지 1.110인지, 10.000 인지 결정해야 한다.
소숫점 이하 값 중에서 GRS가 111 이므로, 짝수 근사를 하면 올림을 해야 한다.
$10.000 \times 2^{5} = 1.000 \times 2^{6}$

다시 정규화를 해야 하므로, 후처리를 하면
frac = 000, E = 6, exp = E + 7 = 13 => 1101

채점기준 : 과정과 답이 모두 맞으면 4점. 답만 맞으면 2점.

문제 3. (20점) [어셈 기초] 레지스터와 메모리에는 다음과 같은 값이 들어있다. 아래 문제들은 독립적으로 실행한다고 가정하고 풀어야 한다. (연속적으로 실행하는 것이 아님)

Register	Value
%eax	0x100
%ebx	0x00
%edx	0x3
%esp	0x108
%eip	0x8000

Address	value
0x100	0xFF
0x104	0xAB
0x108	0x13
0x10C	0x08
0x110	0x00

1) movl 0x4(%eax, %edx, 4), %edx
위 명령을 실행했을 때, %edx 레지스터에 저장되는 값을 쓰시오.
edx * 4 + eax + 4 = 0x110 0x110번지에 저장된 값 0x00 이 정답
채점기준 : 답이 정확히 맞아야 함.

2) call 0x8300
위 명령을 실행했을 때, %eip 레지스터와 %esp 의 값을 각각 쓰시오.
eip : 0x8300, %esp : 0x104
채점기준 : 하나만 맞으면 3점

3) cmpl %eax, %edx
위 명령을 실행했을 때 값이 달라지는 레지스터의 이름을 쓰시오
플래그 레지스터 또는 상태레지스터
채점기준 : eax, edx 등을 쓰면 틀림

4) leal 0x4(%eax), %ebx
위 명령을 실행했을 때, %ebx 레지스터에 저장되는 값을 쓰시오.
0x100 + 0x4 = 0x104
채점기준 : 정확히 맞아야 함.

문제 4. (20점) [역어셈블] 다음은 사용자로부터 input = read_line() 형태로 입력받은 스트링을 매개변수로 넘겨받아 실행된 phase_3() 함수를 gdb 에서 역어셈블한 것이다.

```
Dump of assembler code for function phase_3:
   0x08048bb1 <+0>:     sub    $0x2c,%esp
   0x08048bb4 <+3>:     lea    0x18(%esp),%eax
   0x08048bb8 <+7>:     mov    %eax,0xc(%esp)
   0x08048bbc <+11>:    lea    0x1c(%esp),%eax
   0x08048bc0 <+15>:    mov    %eax,0x8(%esp)
   0x08048bc4 <+19>:    movl   $0x8049be0,0x4(%esp)
   0x08048bcc <+27>:    mov    0x30(%esp),%eax
   0x08048bd0 <+31>:    mov    %eax,(%esp)
   0x08048bd3 <+34>:    call   0x8048788 <__isoc99_sscanf@plt> ----------A)
   0x08048bd8 <+39>:    cmp    $0x1,%eax
   0x08048bdb <+42>:    jg     0x8048be2 <phase_3+49>     // %eax > 1
   0x08048bdd <+44>:    call   0x8049417 <explode_bomb>
   0x08048be2 <+49>:    cmpl   $0x7,0x1c(%esp)
   0x08048be7 <+54>:    ja     0x8048c25 <phase_3+116>    // %eax 는 0~7
   0x08048be9 <+56>:    mov    0x1c(%esp),%eax
   0x08048bed <+60>:    jmp    *0x8049874(,%eax,4)
   0x08048bf4 <+67>:    mov    $0x1af,%eax
   0x08048bf9 <+72>:    jmp    0x8048c36 <phase_3+133>
   0x08048bfb <+74>:    mov    $0x342,%eax
   0x08048c00 <+79>:    jmp    0x8048c36 <phase_3+133>
   0x08048c02 <+81>:    mov    $0x120,%eax
   0x08048c07 <+86>:    jmp    0x8048c36 <phase_3+133>
   0x08048c09 <+88>:    mov    $0x389,%eax
   0x08048c0e <+93>:    jmp    0x8048c36 <phase_3+133>
   0x08048c10 <+95>:    mov    $0x1a7,%eax
   0x08048c15 <+100>:   jmp    0x8048c36 <phase_3+133>
   0x08048c17 <+102>:   mov    $0xa3,%eax
   0x08048c1c <+107>:   jmp    0x8048c36 <phase_3+133>
   0x08048c1e <+109>:   mov    $0x2fc,%eax
   0x08048c23 <+114>:   jmp    0x8048c36 <phase_3+133>
   0x08048c25 <+116>:   call   0x8049417 <explode_bomb>
   0x08048c2a <+121>:   mov    $0x0,%eax
   0x08048c2f <+126>:   jmp    0x8048c36 <phase_3+133>
   0x08048c31 <+128>:   mov    $0x3d,%eax
   0x08048c36 <+133>:   cmp    0x18(%esp),%eax
   0x08048c3a <+137>:   je     0x8048c41 <phase_3+144>
   0x08048c3c <+139>:   call   0x8049417 <explode_bomb>
   0x08048c41 <+144>:   add    $0x2c,%esp
   0x08048c44 <+147>:   ret
```

1)(5점) 위 프로그램이 실행될 때 A)위치의 sscanf 를 호출하기 직전의 스택을 그리고자 한다. 아래 표의 (1)~(4) 에 들어갈 내용을 오른편의 주소를 참고하여 표에 채우시오. 단, 스택의 한 칸은 4바이트씩 저장된 것으로 가정한다. 스택에 들어가는 내용은 정확한 값을 쓰거나 들어가는 값의 의미를 기록해야 한다. %eax 와 같이 레지스터 값을 무의미하게 채우면 안된다.

Stack memory	address
(4) input	%esp + 0x30
return addr to main	
....	
(1) %esp + 0x1c	
(2) 0x8049be0	
(3) input	%esp

채점기준 : (1)~(4) 각 1점. 모두 맞으면 1점 추가. 다른 형태로 답을 적은 경우에 스택에 들어가는 내용이 무엇인지 설명이 된다고 생각되면 점수. eax값과 같이 단순 레지스터 이름을 쓴 경우 틀림

2) (5점) 이 phase_3 함수는 sscanf에서 돌아온 후에 switch 문 구조를 갖는다. 위 역어셈블한 코드로부터 점프테이블이 메모리 몇 번지에 위치하고 있는지 쓰시오.

0x8049874

채점기준 : 정확히 맞은 경우만 5점. 나머지는 모두 0점

3) (10점) 점프테이블의 주소에 저장된 값을 다음과 같이 출력하였다.
(gdb) x/8xw 0x80xxxxx (주. 실제로는 주소로 2)번 답을 써줘야 함)
0x80xxxx : 0x08048bf4 0x08048c31 0x08048bfb 0x08048c02
0x08048c09 0x08048c10 0x8048c17 0x8048c1e

이 정보를 이용하여 위 역어셈블한 코드에 대응되는 C언어의 switch문을 완성하시오. switch 문의 내부만 작성하면 됨.

```
switch(op)
{
        case 0 :
                /* code at 0x8048bf4 */
                value = 0x1af;
                break;
        case 1 :
                /* code at 0x8048c31 */
                value = 0x3d;
                break;
        case 2 :
                /* code at 0x8048bfb */
                value = 0x342
                break;
        case 3 :
                /* code at 0x8048c02 */
                value = 0x120;
                break;
        case 4 :
                /* code at 0x8048c09 */
                value = 0x389;
                break;
        case 5 :
                /* code at 0x8048c10 */
                value = 0x1a7
                break;
        case 6 :
                /* code at 0x8048c17 */
                value = 0xa3;
                break;
        case 7 :
                /* code at 0x8048c1e */
                value = 0x2fc;
                break;
}
```

채점기준 : 정확히 맞으면 10점. case 하나씩 틀리면 1점씩 감점.

문제 5. (20점) [Recursive 함수] 다음은 함수 doSomething() 을 컴파일한 코드를 역어셈블한 결과를 보여준다.

```
000000af <doSomething>:
af:   push   %ebp
b0:   mov    %esp,%ebp
b2:   sub    $0xc,%esp
b5:   mov    0x8(%ebp),%ecx
b8:   mov    $0x1,%eax
bd:   test   %ecx,%ecx
bf:   je     de <doSomething+0x2f>
c1:   mov    %ecx,%edx
c3:   shr    $0x1f,%edx
c6:   lea    (%ecx,%edx,1),%edx
c9:   sar    %edx
cb:   mov    0x10(%ebp),%eax
ce:   mov    %eax,0x8(%esp)
d2:   mov    %ecx,0x4(%esp)
d6:   mov    %edx,(%esp)
d9:   call   da <doSomething+0x2b>
de:   leave
df:   ret
```

```
int doSomething(int a, int b, int c){
    int d;
    if (a == 0){ return 1;}
    d = a/2;
    c = doSomething(d,a,c);
    return c;
}
```

이 함수를 처음 호출하는 함수에서의 스택부터 시작해서 이 함수를 2회 재귀적으로 호출한 직후의 스택의 모습을 그리시오. (즉, 최소한 이 함수의 스택 프레임이 두 번은 표시되어야 함). 표시할 수 있는 스택 내부의 값은 최대한 표시하시오.

c
b
a
ret Addr to caller
old ebp
eax (c)
ecx (b)
edx (a)
0xde
old ebp
eax (c)
ecx (b)
edx (a)
0xde
old ebp <- %ebp

blank
blank
blank <- %esp

채점기준 : 위 색칠한 것과 같이 doSomething의 스택 프레임의 각 항목당 2점씩.

문제 6. (10점)[보너스] 아래 C 함수를 컴파일한 결과는 다음과 같다. 아래 역어셈블한 코드에서 잘못된 인스트럭션을 찾아서 동그라미로 표시하고, 그 이유를 설명하시오.

1)

```
int squareNumber(int x) {
  return (x * x);
}
```

```
08048344 <squareNumber>:
 8048344:       55                      push   %ebp
 8048345:       89 e5                   mov    %esp,%ebp
 8048347:       8b 45 04                mov    0x4(%ebp),%eax
 804834a:       0f af c0                imul   %eax,%eax
 804834d:       5d                      pop    %ebp
 804834e:       c3                      ret
```

0x4(%ebp)는 리턴 주소를 가져온다.

채점기준 : 틀린 곳과 이유를 모두 맞추면 5점. 다소 이상하면 3점.

2)

```
int unrandomNumber() {
  return 4;
}
```

```
0804835e <unrandomNumber>:
 804835e: 55                    push   %ebp
 804835f: 89 e5                 mov    %esp,%ebp
 8048361: a1 04 00 00 00        mov    0x4,%eax
 8048366: 5d                    pop    %ebp
 8048367: c3                    ret
```

0x4 를 상수로 바로 쓰면, 주소 접근을 한다. $0x4로 써야한다.

채점기준 : 틀린 곳과 이유를 모두 맞추면 5점. 다소 이상하면 3점