Chungnam National University
Department of Computer Science and Engineering

2012년 가을학기

중간고사

2012년 10월 18일
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) (4점) CPU의 인스트럭션 사이클을 쓰시오.

fetch-decode-execution-store
인출(선입)-해독-실행-저장

채점기준 : 모두 맞아야 4점. 단어가 조금 틀려도 알고 있다고 생각하면 4점 준다

2) (4점) 0x43AB120A 을 Little Endian 과 Big Endian 컴퓨터에서 0x800100 번지부터 저장한다면, 0x800101 번지에 저장되는 값은 얼마인가?

Little Endian : 0x12 Big Endian : 0xAB

채점 기준 : 각각 2점씩. 정확히 맞아야 함

3) (4점) IA32 어셈블리 명령어인 ret 를 실행할 때 어떤 일들이 일어나는지 eip와 esp 레지스터를 사용하여 설명하시오.

스택에서 리턴주소를 가져와서 eip 레지스터에 쓴다.
esp 값이 4 증가한다.

채점 기준 : eip와 esp에 대해 비슷한 설명을 하면 점수를 각각 2점씩 줌

4) (4점) 스택의 버퍼 오버플로우 취약점을 막기 위한 방법 두 가지를 쓰시오.

스택 랜덤화, 스택손상 감지, 실행가능 코드 영역제한 중에서 두 가지를 써야함

채점기준 : 읽기 예습 숙제를 한 사람이라면 답을 쓸 수 있을 것임. 개당 2점씩

5) (4점) 16비트의 2의 보수로 표현할 수 있는 가장 작은 정수와 가장 큰 정수를 각각 16진수 형태로 나태내시오.

가장큰수 : 0111 1111 1111 1111 => 0x7FFF
가장작은수 : 1000 0000 0000 0000 => 0x8000
채점기준 : 각 2점씩

문제 2. 실습 (20점)

1) (10점) 다음과 같은 소스코드를 사용하여 실행파일 diary를 생성하기 위한 make 파일을 완성하시오.

```
diary.h:
#include <stdio.h>
int memo();
int calendar();

main.c:
#include "diary.h"
int main() {
    memo();
    calendar();
    return 0;
}
```

```
memo.c :
#include "diary.h"
int memo() {
return 0;
}

calendar.c :
#include "diary.h"
int calendar() {
      printf("calendar₩n");
      return 0;
}
```

```
CC      = gcc
CFLAGS  = -W -Wall
TARGET  = diary
all : $(TARGET)

$(TARGET): main.o memo.o calendar.o
    $(CC) $(CFLAGS) –o diary main.o memo.o calendar.o
(또는 $(CC) $(CFLAGS) –o $@ $^)
main.o: main.c
    $(CC) $(CFLAGS) –c –o main.o main.c
(또는 $(CC) $(CFLAGS) –c –o $@ $^)
memo.o: memo.c
    $(CC) $(CFLAGS) –c –o memo.o memo.c
(또는 $(CC) $(CFLAGS) –c –o $@ $^)
calendar.o: calendar.c
    $(CC) $(CFLAGS) –c –o calendar.o calendar.c
(또는$(CC) $(CFLAGS) –c –o $@ $^)
```

2) (10점) int getByte(x,n) 함수를 아래 제한조건에 유의해서 C 언어로 구현하시오. 이 함수는 주어진 정수 x 의 오른쪽에서 n번째 바이트를 반환해야 한다.

- "! ~ & ^ | + « »" 연산자만을 사용
- 제어문 (if,do,while,for,switch 문) 은 사용금지

```
int getByte(x,n) {
      return (x>>(n<<3))&0xff;
 }
```

문제 3. (20점) 부동소숫점 수의 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 사용한다. 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. 그 외의 경우 부분점수 가능 2점

2) (4점) 위 IEEE 부동소수 표시 방법을 사용해서 비정규화 방식으로 표시되는 양의 최대값을 표시하시오. (위의 8비트 형식으로 표시)

비정규화 방식은 exp가 0이어야 한다. 따라서 0이 아닌 양의 최대값은

0 0000 111

채점기준 : 정확히 맞으면 4점. 그 이외의 경우는 0점

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

E = 1 - 7 = -6, M = 0.111, S = 0
7/8*1/64 = 7/512

채점기준 : 답이 맞으면 4점. 그 이외의 경우는 0점

4)(4점) 십진수 19.0을 위 8비트 IEEE 부동소숫점 표시로 나타내려고 한다. 19.0을 $(-1)^S*M*2^E$ 형태로 표시할 때, M과 E를 각각 나타내시오. 단, M은 1.xxxxxx 형태로 표시해야 함.

19.0 = 10011.0 = 1.0011 x 2^4
M = 1.0011, E = 4

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.

frac 부분의 길이가 3비트이므로, 1.001 인지 1.010인지, 결정해야 한다.
소숫점 이하 값 중에서 GRS가 110 이므로, 짝수 근사를 하면 올림을 해야 한다.
1.010 x 2^4

frac = 010, E = 6, exp = E + 7 = 11 => 1011

채점기준 : 답이 모두 맞으면 4점. 그 외에는 0점

문제 4. 어셈블리어 I (20점)

다음과 같은 C 함수를 점프테이블 방식을 이용한 어셈블리어로 작성하고자 한다.

```
typedef enum {ADD, MULT, MINUS, DIV, MOD, BAD}   op_type;

char unparsed_symbol(optype op)
{
  switch (op) {
    case ADD :
       return '+';
    case MULT:
       return '*';
    case MINUS:
       return '-';
    case DIV:
       return '/';
    case MOD:
       return '%';
    case BAD:
      return '?';
    }
}
```

1) [4점] 위 프로그램을 어셈블리어로 아래와 같이 번역하려고 한다. 빈 곳을 채우시오.

```
unparse_symbol:
        pushl %ebp              # Setup
        movl %esp,%ebp          # Setup
        movl 8(%ebp),%eax       # eax = op
        cmpl $5,%eax            # Compare op - 5
        (a)ja    .L49           # If > goto done
        (b)jmp *.L57(, %eax, 4)         # goto Table[op]
.L51:
        movl $43,%eax           # '+'
        jmp .L49
.L52:
        movl $42,%eax           # '*'
        jmp .L49
.L53:
        movl $45,%eax           # '-'
        jmp .L49
.L54:
        movl $47,%eax           # '/'
        jmp .L49
.L55:
        movl $37,%eax           # '%'
        jmp .L49
.L56:
        movl $63,%eax           # '?'
.L49:   # Done:
        movl %ebp,%esp          # Finish
```

```
        popl %ebp       # Finish
        ret     # Finish

.section .rodata
    .align 4
.L57:
```

2)(4점) 위와 같은 스위치문이 실행되기 위한 점프테이블을 완성하시오.

```
.section .rodata
    .align 4
.L57:
        .long .L51      #Op = 0
        .long .L52      #Op = 1
        .long .L53      #Op = 2
        .long .L54      #Op = 3
        .long .L55      #Op = 4
        .long .L56      #Op = 5
```

3) [4점] 위 코드를 어셈블한 뒤에 objdump를 이용하여 case 문들이 아래와 같은 컴파일 되었다는 것을 알 수 있었다면, .L49 의 값이 실행시에 어떤 주소값으로 결정되었는지 쓰시오. 0x804875c

```
8048730:   b8 2b 00 00 00    movl   $0x2b,%eax
8048735:   eb 25             jmp    804875c <unparse_symbol+0x44>
8048737:   b8 2a 00 00 00    movl   $0x2a,%eax
804873c:   eb 1e             jmp    804875c <unparse_symbol+0x44>
804873e:   89 f6             movl   %esi,%esi
8048740:   b8 2d 00 00 00    movl   $0x2d,%eax
8048745:   eb 15             jmp    804875c <unparse_symbol+0x44>
8048747:   b8 2f 00 00 00    movl   $0x2f,%eax
804874c:   eb 0e             jmp    804875c <unparse_symbol+0x44>
804874e:   89 f6             movl   %esi,%esi
8048750:   b8 25 00 00 00    movl   $0x25,%eax
8048755:   eb 05             jmp    804875c <unparse_symbol+0x44>
8048757:   b8 3f 00 00 00    movl   $0x3f,%eax
```

3) [4점] 위의 objdump 결과를 이용하여 점프 테이블의 내용인 .L51~.L56 값이 얼마로 결정되었는지 각각 쓰시오.

0x8048730, 0x8048737, 0x8048740, 0x8048747, 0x8048750, 0x8048757

4) [4점] 점프테이블 방식으로 스위치문을 번역하게 되면 무슨 장점이 있는지 쓰시오.
항상 일정하게 O(1)의 처리 속도를 갖게 되어 분기처리 성능이 뛰어나다

5) [4점] 어떤 경우에 컴파일러가 switch 문을 점프테이블이 아닌 if else 구조로 컴파일 하게 되는지 쓰시오.
Sparse 한 분기조건의 경우와 분기의 가지수가 작은 경우

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 진입해서 이 함수를 2회 재귀적으로 호출한 직후의 스택을 아래와 같이 나타내었을 때, 다음 질문에 답하시오.

참고. leave 명령은

movl %ebp, %esp

popl %ebp

과 동일한 효과를 낸다.

스택내용	주소	
c	0x8100	
b	0x80FC	
a	0x80F8	
ret Addr to caller	0x80F4	
(1)	0x80F0	doSomething 최초 진입시 스택 프레임
(2)	0x80EC	
(3)	0x80E8	
(4)	0x80E4	
(5)	0x80E0	
(6)	0x80DC	doSomething 첫번째 재귀호출 스택프레임

(7)	0x80D8	
(8)	0x80D4	
(9)	0x80D0	
(10)	0x80CC	
(11)	0x80C8	두번째 doSomething 재귀호출시 스택프레임
(12)	0x80C4	
(13)	0x80C0	
(14)	0x80BC	

1) (4점) 위 프로그램의 주소 0xb5에 저장되어 있는 movl 0x08(%ebp), %ecx 를 실행하면 %ecx 레지스터에 저장되는 값은 매개변수 a, b, c, 중에 어느 값이 저장되는가?

답 : a

2) (4점) 함수의 리턴값이 저장되는 레지스터의 이름을 쓰시오.

답 : %eax

3) (6점) 위 스택에서 (11)번 위치에 저장되어 있을 값을 스택의 주소를 참고하여 쓰시오. 반드시 실제 저장되어 있는 값을 16진수로 표시해야 함

old ebp 이므로, 이전 스택 프레임의 시작 주소인 0x80DC 가 들어 있다.

채점기준 : 주소가 정확히 일치해야 6점

4) (6점) (10)번 위치에는 doSomething 호출시 리턴주소가 저장되어 있다. 이 주소의 값을 16진수로 쓰시오.

doSomething의 리턴 주소는 바로 다음명령어가 저장된 주소인 0xde 가 들어 있다.

채점기준 : 주소가 정확히 일치해야 6점

문제 6. 기타 (10점) 아래의 C 코드의 일부가 지워진 상태다. 번역된 어셈블리 코드로 부터 C 코드의 빈곳 두 곳을 채우시오.

```
unsigned mystery1(unsigned n) {
     if(_________)
          return 1;
     else
          return 1 + mystery1(_________);
}

mystery1:
    pushl %ebp
    movl %esp, %ebp
    subl $8, %esp
    cmpl $0, 8(%ebp)
    jne .L2
    movl $1, -4(%ebp)
    jmp .L3
.L2:
    movl 8(%ebp), %eax
    shrl %eax
    movl %eax, (%esp)
    call mystery1
    addl $1, %eax
    movl %eax, -4(%ebp)
.L3:
    movl -4(%ebp), %eax
    leave
    ret
```

답 :

1st blank: n == 0 or n < 1

2nd blank: n >> 1 or n / 2

채점 기준 : 각 5점씩