Chungnam National University
Department of Computer Science and Engineering

2014년 가을학기

중간고사

2014년 10월 31일
시스템 프로그래밍

분반/학번	반/
이름	

문제	배점	점수
1	20	
2	24	
3	18	
4	20	
5	28	
총계	110	

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

문제 1. 기초지식 (20점)

1) (4점) CPU의 인스트럭션 사이클 4단계를 쓰시오.

Fetch-decode-execution-store 또는 선입-해독-실행-저장

2) (4점) 다음과 같은 C 프로그램에서 s 계산 시에 오버플로우가 발생하는 조건을 쓰시오.

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

u>0, v>0, s<0 그리고 u<0, v<0, s>0 각 2점씩

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

스택에서 리턴주소를 pop해서 eip에 써주고, esp를 4 증가시킨다

4) (4점) 다중 비교를 해야할 필요가 있을 때, if ~ else 문 대신에 switch 문을 사용하면 어떤 좋은 점이 있는지 설명하시오.

O(1) 실행이 가능해진다

5) (4점) is_little_endian 이라는 함수를 아래와 같이 작성하려고 한다. 이 함수에서는 little_endian 컴퓨터에서 실행하는 경우 1을 리턴하고, big_endian 컴퓨터에서는 0을 리턴해야 한다. 단, 이 프로그램은 컴퓨터의 워드크기와 무관하게 동작해야 한다. 빈 곳을 채워서 프로그램이 원하는 결과를 얻도록 하시오.

```
1 int is_little_endian(void)
2 {
3       /* MSB = 0, LSB = 1 */
4       int x = 1;
5
6       /* Return MSB when big-endian, LSB when little-endian */
7       return (int) (* (char *) &x);
8 }
```

문제 2. 패트리엇 미사일 (24점)

패트리엇 미사일 방어 시스템은 컴퓨터를 내장하고 있으며, 이 컴퓨터에는 0.1초마다 증가하는 카운터를 사용한다. 이 카운터로부터 1초를 계산하기 위해 이 카운터 값을 24비트로 표시한 0.1의 근사값과 곱해준다. 0.1 은 $0.000110011[0011]\cdots_2$ 형태의 무한 반복되는 이진수로 표시된다. 이 컴퓨터는 0.1의 무한반복 패턴 중에 소수점 우측의 23비트만을 이용하여 0.1을 $x=0.00011001100110011001100_2$ 로 근사하여 사용한다. 다음 문제에 답하시오. 반드시 계산과정을 보여야 한다.

1) (4점) 이 패트리엇의 제어 프로그램에서 사용하는 0.1 값이 포함하는 오차는 얼마인지 이진수로 표시하라.

오차는 참값 – 근사값 x =

$0.000110011[0011]\cdots_2 - x=0.00011001100110011001100_2$ =

$$0.0000000000000000000000001100[1100]\cdots_2$$

2) (4점) 이 패트리엇 시스템에 전원을 공급한 뒤 200시간동안 동작을 시키게 되면, 위의 0.1초 카운터 값도 200시간 동안 증가한다. 200시간 동작 후에 컴퓨터 프로그램에 의해 누적되는 오차는 몇 초가 되는가? (1)번 문제에서 계산한 오차는 십진수로 약 9.54×10^{-8} 초 라는 값을 이용할 것)

9.54×10^{-8} 초 x 200 시간 x 60분 x 60초 x 10 = 약 0.686초

3) (4점) 이 패트리엇 시스템이 6km/초 의 속도로 비행하는 스커드 미사일을 전원을 공급한 뒤 200시간 뒤에 격추하려 한다면, 이 컴퓨터의 프로그램이 계산해낸 스커드 미사일위치는 몇 m의 오차를 갖게 되는가?

0.686초 x 6km/sec = 약 4116미터

4) (4점) 이 패트리엇 시스템의 프로그램이 0.1의 근사값으로, 이진 소수의 우측 23비트를 이용하는 대신에, IEEE 표준의 인접짝수 근사를 24번째 비트에 적용하는 방법으로 23비트를 사용한다면, 이 프로그램에서 사용하는 0.1의 근사값은 이진 소수로 어떤 값을 이용하는가?

$x = 0.00011001100110011001101_2$

5) (4점) 문제 4)의 근사값을 참값 0.1 대신 이 프로그램이 이용한다면, 패트리엇 시스템의 전원을 공급한 후 200시간 후에 누적된 오차는 몇 초가 되는가? (4)번 문제에서 계산한 오차는 십진수로 약 2.38×10^{-8} 초 라는 값을 이용할 것)

2.38×10^{-8} 초 x 200 시간 x 60분 x 60초 x 10 = 약 0.172초

6) (4점) 문제 5)의 오차가 발생하는 경우, 이 패트리엇 시스템이 문제 3)과 동일하게 6km/초의 속도로 비행하는 스커드 미사일을 전원을 공급한 뒤 200시간 뒤에 격추하려 한다면, 이 컴퓨터의 프로그램이 계산해낸 스커드 미사일위치는 몇 m의 오차를 갖게 되는가?

0.172 초 x 6000 = 1032 미터

문제 3. 프로시져 (18점) 수업시간에 배운 swap(int *xp, int *yp) 함수는 두 개의 포인터를 이용하여 두 변수의 값을 치환하는 함수이다.

1) (8점) 이 함수를 아래와 같이 작성하려고 할 때 빈 곳 (1)~(4)에 들어갈 코드를 쓰시오.

```
void swap(int *xp, int *yp)
{
  int t0 = (1)_*xp_____ ;
  int t1 = (2)_*yp______;
  (3)_*xp____ = t1;
  (4)_*yp____ = t0;
}
```

2) (10점) 위 함수를 아래와 같이 어셈블리어로 번역할 때, (1)~(5)의 빈 곳을 주어진 주석문에 맞추어 채우시오. 단, 각 변수들과 레지스터는 다음과 같이 대응되어야 하며, xp 값은 %ebp에서 +8바이트, yp값은 %ebp에서 +12바이트 떨어진 곳에 저장되어 있는 상태에서 프로그래밍을 해야 함.

Register	Variable
%ecx	yp
%edx	xp
%eax	t1
%ebx	t0

```
        pushl %ebp
        movl  %esp,%ebp
        pushl %ebx

        (1)                     # ecx = yp
        (2)                     # edx = xp
        (3)                     # eax = t1
        (4)                     # ebx = t0
        (5)                     # eax 저장
        movl %ebx,(%ecx)

        movl -4(%ebp),%ebx
        movl %ebp,%esp
```

```
popl %ebp
ret
```

```
(1) mov 12(%ebp), %ecx          # ecx = yp
(2) mov 8(%ebp), %edx           # edx = xp
(3) movl (%ecx), %eax           # eax = t1
(4) movl (%edx), %ebx           # ebx = t0
(5) movl  %eax, (%edx)          # eax 저장
```

문제 4. [루프의 구현] (20점) 다음의 어셈블리어와 관련된 문제에 답하시오.

```
; 초기에 x, y, n 은 %ebp에 대해서 8, 12, 16 오프셋 떨어진 곳에 저장되어 있다.
;
        movl 8(%ebp), %ebx
        movl 16(%ebp), %edx
        xorl   %eax, %eax
        decl   %edx
        js      .L4
        movl %ebx, %ecx
        imull 12(%ebp), %ecx
        .p2align 4, , 7              ; 캐쉬 성능 최적화를 위해 삽입함
.L6 :
        addl %ecx, %eax
        subl %ebx, %edx
        jns   .L6
.L4 :
```

위 코드는 다음의 C 코드를 번역하여 만든 것이다. 아래의 빈 칸을 채우시오. 주의할 점은 리턴 값은 %eax 레지스터를 이용한다는 것이다.

```
int loop(int x, int y, int n)
{
        int result = 0 ;
        int i ;
        for (i= (1) n-1 ; i (2) >= 0 ; i = (3) i- x) {
                result + = (4) y * x ;
        }
        return result ;
}
```

문제 5. 버퍼 오버플로우(28점) 다음 C 프로그램과 어셈블리 프로그램을 이용하여 다음의 문제에 답하시오.

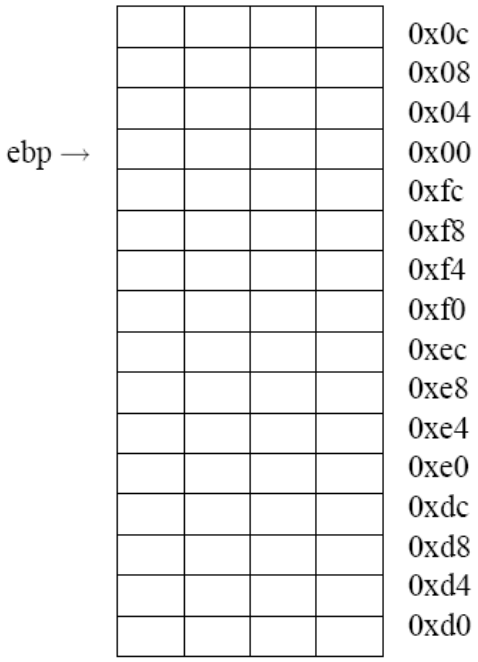
```
void top_secret(int len)
{
       char buf[8];
       scanf("%s", buf);
       if(strlen(buf) != len)
       exit(1);
}
int main()
{
       printf("Enter a passphrase: ");
       top_secret(8);
       printf("The chicken flies at midnight!\n");
       return 0;
}
```

```
08048530 <top_secret>:
 8048530:       55                      push   %ebp
 8048531:       89 e5                   mov    %esp,%ebp
 8048533:       83 ec 08                sub    $0x8,%esp
 8048536:       8d 45 f8                lea    0xfffffff8(%ebp),%eax
 8048539:       50                      push   %eax
 804853a:       68 40 86 04 08          push   $0x8048640
 804853f:       e8 44 fe ff ff          call   8048388 <scanf>
 8048544:       8d 45 f8                lea    0xfffffff8(%ebp),%eax
 8048547:       50                      push   %eax
 8048548:       e8 5b fe ff ff          call   80483a8 <strlen>
 804854d:       83 c4 0c                add    $0xc,%esp
 8048550:       3b 45 08                cmp    0x8(%ebp),%eax
 8048553:       74 0b                   je     8048560 <top_secret+0x30>
 8048555:       6a 01                   push   $0x1
 8048557:       e8 8c fe ff ff          call   80483e8 <exit>
 804855c:       8d 74 26 00             lea    0x0(%esi,1),%esi
 8048560:       89 ec                   mov    %ebp,%esp
 8048562:       5d                      pop    %ebp
 8048563:       c3                      ret
```

다음의 사항을 참고하시오.

- scanf(“%s”, buf) 는 입력문자들을 NULL 문자(‘\0’)를 포함해서 buf 주소에 저장한다.
- strlen(s) 는 스트링 s 에 포함된 문자의 갯수를 리턴해 준다.(NULL 문자 포함)
- exit(1) 은 현재 프로시져에서 리턴하지 않고 바로 종료시킨다.
- 리눅스/x86 에서는 Little Endian 이다.

문제를 푸는 과정에서 다음의 스택 그림에 관련 자료를 채워가며 생각하면 유용하다.



1) (10점) 프로그램이 scanf 를 호출하기 직전까지 실행되었을 때, 다음 값들이 스택의 어디에 위치하는지 위의 스택 그림에서 우측에 표시한 ebp 레지스터와의 오프셋 값으로 답하시오.

- 리턴 주소 (0x04)
- 이전 ebp 값 (0x00)
- len (0x08)
- &buf (0xf8)
- "%s" 의 주소 (0xf0)

2) (4점) 여러분이 "chickenstonight" 라는 스트링을 위 프로그램을 실행시키면서 입력했다면(따옴표는 빼고) ebp 레지스터가 가리키는 곳에 저장된 값을 아래의 아스키 코드 값을 이용해서 네 바이트의 16진수로 쓰시오. (순서에 주의)

(%ebp) = 0x _696e6f74

Character	Hex value	Character	Hex value
'c'	0x63	'h'	0x68
'i'	0x69	'k'	0x6b
'e'	0x65	'n'	0x6e
's'	0x73	't'	0x74
'o'	0x6f	'g'	0x67
'h'	0x68	\0	0x00

3) (14점) 함수 top_secret이 main( ) 함수 실행 중에 주소 0x804857f 에 저장되어 있는 5 바이트 call 명령어로부터 호출된다고 하자. top_secret 의 첫번째 명령어가 실행되기 이전에 레지스터에 다음과 같은 값들이 저장되어 있다.

이후 top_secret 함수 내부의 call scanf 까지 실행되어 0x8048544 에 저장되어 있는 lea 명령을 실행하기 전까지 프로그램이 진행되었다. 만일 사용자가 "chickens"라고 입력을 한 경우에 아래 메모리 주소에 저장된 값을 쓰시오. 값을 알 수 없는 경우에는 unknown 이라고 쓰시오.

Register	*Hex Value*
eax	0x14
ecx	0x0
edx	0x0
ebx	0x40157770
esp	0xbffff98c
ebp	0xbffff998
esi	0x40015e8c
edi	0xbffffa04
eip	0x8048530

(gdb) x 0xbffff990
0xbffff990: 0x00000008
(gdb) x 0xbffff98c
0xbffff98c: 0x08048584
(gdb) x 0xbffff988
0xbffff988: 0xbffff900 {* buffer overwrite from null character!}
(gdb) x 0xbffff984
0xbffff984: 0x736e656b
(gdb) x 0xbffff980
0xbffff980: 0x63696863
(gdb) x 0xbffff97c
0xbffff97c: 0xbffff980
(gdb) x 0xbffff978
0xbffff978: 0x08048640