Chungnam National University
Department of Computer Science and Engineering

2009년 봄학기 **김 형신**

중간고사 답안

2009년 4월 30일
어셈블리어 프로그래밍

학번	
이름	

문제	배점	점수
1	20	
2	25	
3	30	
4	25	
5	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] C 프로그램이 컴파일러에 의해 기계어 코드가 만들어지기까지의 세부 단계를 쓰시오.

소스코드=> 프리프로세서 => 컴파일러 => 어셈블러 => 링커 의 과정을 설명해야함

2) [5점] 8비트를 이용하여 2의 보수를 구현하고자 한다. 아래 테이블의 빈 곳을 채우시오.

수	십진수	2의보수(2진수)
0	0	(a)0000 0000
	-17	(b)1110 1111
	(c)41	0010 1001
TMax	127	(d)0111 1111
TMin	-128	(e)1000 0000

3) [5점] 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 }
```

4) [5점] Moore's Law가 무엇인가?

반도체에 집적되는 트랜지스터의 수는 매년 두 배씩 증가한다. 대략 이런 맥락이면 맞게 채점

문제 2. 데이터 이동명령과 성능(25점)

1) [10점] 다음과 같이 CPU내의 레지스터 값이 들어 있을 때, 아래 명령어를 각각 실행한 후의 레지스터의 값을 쓰시오.

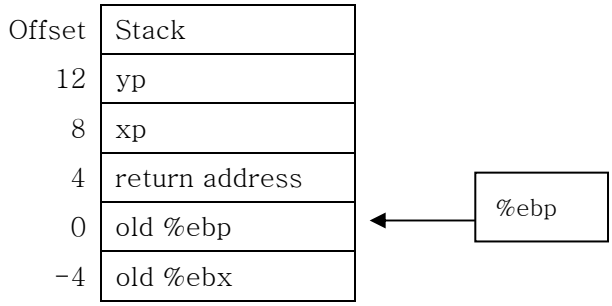
초기상태	
%eax	0x456
%edx	0
%esp	0x210

pushl %eax	
%eax	0x456
%edx	0x0
%esp	0x20c

popl %edx	
%eax	0x456
%edx	0x456
%esp	0x210

2) [10점] 두 개의 메모리에 저장되어 있는 값을 바꿔주는 void swap() 함수를 다음과 같이 작성하였다. Setup 부분 후, 스택에 저장된 xp와 yp의 위치가 다음 그림과 같을 때, 이 함수의 body 부분을 아래 빈 곳에 어셈블리로 작성하시오.

```
void swap(int *xp, int *yp)
{
  int t0 = *xp;
  int t1 = *yp;
  *xp = t1;
  *yp = t0;
}
```



단, 각 변수들과 레지스터는 다음과 같이 대응되어야 한다.

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

```
swap:
    pushl %ebp
    movl   %esp,%ebp
    pushl %ebx
    /* 이 곳에 body 를 채우시오 *

        mov     12(%ebp), %ecx          # ecx = yp
        mov     8(%ebp), %edx           # edx = xp
        movl    (%ecx), %eax            # eax = t1
        movl    (%edx), %ebx            # ebx = t0
        movl    %eax, (%edx)            # eax 저장
        movl    %ebx,(%ecx)

    movl %ebp,%esp
    popl %ebp
    ret
```

3) [5점] 아래와 같은 c 프로그램을 작성하여 컴파일 하는 경우 t4 = y * 48 을 mul 명령어를 사용하지 않고 leal 명령을 이용하면 계산속도가 매우 빨라진다. 아래 빈 곳에 채우시오.

```
leal (%edx, %edx, 2), %edx          # (a) edx = 3*y
sall $4, %edx               # (b) edx = 48*y (t4)
```

```
int arith
   (int x, int y, int z)
{
  int t1 = x+y;
  int t2 = z+t1;
  int t3 = x+4;
  int t4 = y * 48;
  int t5 = t3 + t4;
  int rval = t2 * t5;
  return rval;
}
```

문제 3. 조건문의 응용(30점)

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) [10점] 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:
        /* (c) 점프테이블을 작성하시오 */
.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
```

2) [5점] 위 코드를 어셈블한 뒤에 objdump를 이용하여 case 문들이 아래와 같은 컴파일 되었다는 것을 알 수 있었다면, .L49 의 값이 얼마로 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) [5점] 위의 objdump 결과를 이용하여 점프 테이블의 내용인 .L51~.L56 값이 얼마로 결정되었는지 각각 쓰시오.

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

4) [5점] 점프테이블 방식으로 번역하게 되면 무슨 장점이 있는지 쓰시오.
항상 일정하게 O(1)의 처리 속도를 갖게 되어 분기처리 성능이 뛰어나다

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

문제 4. 반복문의 구현 (20점)

이 문제에서는 어셈블리어로부터 C 언어를 재구성해 낼 수 있는 능력을 평가하고자 한다. 아래 C 코드를 컴파일하여 우측의 어셈블리어 프로그램을 얻어냈다.

```
static int bunny(int l, int r, int *A)
{
   int x = (a)___________;
   int i = l - 1;
   int j = r + 1;
   while((b)_____________ ) {
      do j--; while((c)_____________);
      do i++; while( (d)__________ );
      if ( (e)______ ) {
         int t = A[i];
         A[i] = A[j];
         A[j] = t;
      }
    }
    return __________ ;
}
```

```
bunny:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %edi
        pushl   %esi
        pushl   %ebx
        movl    8(%ebp), %eax
        movl    16(%ebp), %esi
        movl    (%esi,%eax,4), %edi
        leal    -1(%eax), %ecx
        movl    12(%ebp), %ebx
        incl    %ebx
        cmpl    %ebx, %ecx
        jge     .L3
.L16:
        decl    %ebx
        cmpl    %edi, (%esi,%ebx,4)
        jg      .L16
.L7:
        incl    %ecx
        cmpl    %edi, (%esi,%ecx,4)
        jl      .L7
        cmpl    %ebx, %ecx
        jge     .L3
        movl    (%esi,%ecx,4), %edx
        movl    (%esi,%ebx,4), %eax
        movl    %eax, (%esi,%ecx,4)
        movl    %edx, (%esi,%ebx,4)
        jmp     .L16
.L3:
        movl    %ebx, %eax
        popl    %ebx
        popl    %esi
        popl    %edi
        popl    %ebp
        ret
```

1) [5점] 좌측의 C 프로그램의 변수 A와 x 가 펜티엄 프로세서의 어떤 레지스터로 번역되고 있는지 각각 쓰시오.

A => %esi, x => edi

2) [5점] 우측의 어셈블리 프로그램에서 %edi와 %esi 를 스택에 push 하는 이유는 무엇인가?

이전 함수에서 이용하던 %edi,%esi 레지스터에 저장된 값들을 보호하기 위해

3) [10점] 좌측의 (a) ~ (e) 를 각각 채우시오.

(a) A[l] (b) i < j (c)A[j] > x (d) A[i] < x (e) i < j

문제 5. [10점]새로운 명령어

다음에 첨부된 REP MOVSB, CLD 명령을 사용하여 반복문을 간단히 작성할 수 있다. 메모리 주소 startup_32 부터 _end 까지에 저장된 데이터를 그대로 메모리 주소 new_start 로 복사하는 프로그램을 작성하시오.

---

movsb 를 이용하면 %esi 주소에 저장된 데이터를 %edi 주소로 복사할 수 있다.
rep movsb 를 이용하면 %ecx 바이크 갯수 만큼 데이터를 메모리에서 다른 메모리 위치로 복사할 수 있다.

```
leal    startup_32, %esi          # 대상 주소가져오기
leal    new_start, %edi           # 옮길 곳의 주소 가져오기
movl $(_end – startup_32), %ecx       # 몇 바이트를 복사할 지 계산하기
cld                               # %esi, %edi 레지스터 값이 자동으로 증가한다
rep movsb
```