2016년 시스템 프로그래밍

-HW 06-

제출일자	2016.10.23.
이 름	정윤수
학 번	201302482
분 반	00

따라하기 1

```
1 .section .data
2 message :
3     .string "%d + %d = %d \n"
4 val1 :
5     .int 100
6 val2 :
7     .int 200
8 .section .text
9 .globl main
10 main :
11     movq val1, %rsi
12     movq val2, %rdx
13     call add_func
14
15     movq %rax, %rcx
16     movq val1, %rsi
17     movq val2, %rdx
18     movq $message, %rdi
19     movq $0,%rax
20     call printf
21
22     ret
23 .type add_func, @function
24 add_func :
25     movq %rsi, %rax
26     addq %rdx, %rax
27     ret
```

```
a201302482@localhost:~/10.18$ ./practice01.out
100 + 200 = 300
a201302482@localhost:~/10.18$
```

이 어셈블리어는 두 개의 변수의 값을 레지스터에 저장을 하여 add_func함수를 호출을 하여 덧셈연산을 하여 값을 반환을 한 것을 출력을해주는 프로그램이다. val1과 val2에 값을 %rsi,%rdx에 저장을 한 후 add_func함수에서 %rax에 두 개의 값의 합을 저장을 한 후 main으로 돌아와서는 레지스터들에 값을 다시 초기화 시켜주고 출력을 시켜준다.

따라하기 2

```
1  .section .data
2  message :
3       .string "%d, %d \n"
4  val1 :
5       .int 100
6  val2 :
7       .int 200
8
9  .section .text
10 .globl main
11 main :
12      movq val1,%rsi
13      movq val2,%rdx
14
15      movq $message, %rdi
16      movq $0,%rax
17      call printf
18
19      movq val1,%rsi
20      movq val2,%rdx
21
22      call swap
23
24      movq val1,%rsi
25      movq val2,%rdx
26      movq $message,%rdi
27      movq $0,%rax
28      call printf
29
30      ret
31 .type swap, @function
32 swap :
33      movq val1, %rcx
34      movq %rdx, val1
35      movq %rcx, val2
36
37      ret
```

```
a201302482@localhost:~/10.18$ ./practice02.out
100, 200
200, 100
a201302482@localhost:~/10.18$
```

이 어셈블리어 프로그램은 두 개의 변수 val1,val2에 저장이 되어있는 값들을 서로 교환을 하는 프로그램이다. 두 개의 변수를 각각 %rsi,%rdx에 저장을 한 후 두 개의 값을 교환하기 전에값들을 printf함수를 call을 하여 출력을해준다. 그 후 val1과,val2의 값을 다시 레지스터에 저장을 해주고 swap함수를 호출하여 값을 서로 바꾼다 swap함수에서는 val1의 값을 임시로 저장을 하기 위해 레지스터 %rcx에 저장을 하여 val1에 val2의 값을 넣어준후 val2에 %rcx의 값을 저장을 한다. 다시 main으로 돌아와 %rsi,%rdx에 val1,val2의 값을 재정의 해주고 printf함수를 call하여 값이 서로 교환된 것을 보여준다.

따라하기 3

```
 1 .section .data
 2 printf_str :
 3       .string "result = %d \n"
 4 x :
 5       .int 1
 6
 7 JUMP_TABLE :
 8       .quad .L0
 9       .quad .L1
10 .section .text
11 .globl main
12 main :
13 movl x, %ecx
14 cmpl $2,%ecx
15 jg DEFALUT
16 jmp *JUMP_TABLE(,%rcx,8)
17 .L0 :
18       movl $65, %esi
19       jmp END
20 .L1 :
21       movl $66, %esi
22       jmp END
23
24       DEFALUT :
25       movl $0,%esi
26 END :
27       movq $printf_str,%rdi
28       movq $0,%rax
29       call printf
30
31       ret
~
```

```
a201302482@localhost:~/10.18$ ./practice03.out
result = 66
a201302482@localhost:~/10.18$
```

이 어셈블리어 프로그램은 switch문을 JUMP_TABLE로 구현을 한 것이다. x의 값을 %ecx에 저장을 한 후 cmpl연산을 이용하여 x의 값이 2보다 크거나 같으면 0을 출력을 하고 x의 값이 2보다 작으면 x의 값에 따라서 다른 Label로 jump를 하게 된다. x의 값이 0이면 65를 출력을하고 x의 값이 1이면 66을 출력을 한다.

과제 1

```
1  .section .data
2  i :
3       .int 1
4  result :
5       .int 1
6  msg :
7       .string "result : %d \n"
8  .section .text
9  .globl main
10 main :
11      movl i, %ecx
12      movq result, %rsi
13 loop :
14      call factorial
15      incl %ecx
16      cmpl $5, %ecx
17      jle loop
18
19      movq $msg, %rdi
20      movq $0, %rax
21      call printf
22      ret
23
24 .type factorial, @function
25 factorial :
26      imulq %rcx, %rsi
27      ret
~
~
```

```
a201302482@localhost:~/10.18$ ./factorial
result : 120
a201302482@localhost:~/10.18$
```

	%esi	%rcx	%rsp
loop 1	1	1	0x7fffffffe568
factorial 1	1	1	0x7fffffffe560
ret 1	1	1	0x7fffffffe560
loop 2	1	2	0x7fffffffe568
factorial 2	1	2	0x7fffffffe560
ret 2	2	2	0x7fffffffe560
loop 3	2	3	0x7fffffffe568
factorial 3	2	3	0x7fffffffe560
ret 3	6	3	0x7fffffffe560
loop 4	6	4	0x7fffffffe568
factorial 4	6	4	0x7fffffffe560
ret 4	24	4	0x7fffffffe560
loop 5	24	5	0x7fffffffe568
factorial 5	24	5	0x7fffffffe560
ret5	120	5	0x7fffffffe560

%esi의 값은 factorial에서 ret으로 가는 사이에 imul 연산으로 인해 변하게된다. %esi는 1부터 시작해서 120까지 올라간다. %rcx의 값은 factorial함수에서 빠져나오고 incl명령어에 의해서 값이 변경이 된다 break 포인트 ret과 loop사이에서 값이 변화는 것을 확인할수 있다. %rsp는 factorial함수의 호출과 반환때 마다 값이 변경이 된다. 함수를 호출을 하면 값이 8만큼 감소를 하게 되고 값을 반환을 하여 함수를 빠져나오면 다시 %rsp의 값이 8증가 하는 것을 볼 수 있다.

과제 2

```
1  .section .data
2  i :
3      .int 1
4  j :
5      .int 1
6  count :
7      .int 0
8  msg1 :
9      .string "%d %d "
10 msg2 :
11     .string "% d "
12 msg3 :
13     .string "\n"
14 .section .text
15 .globl main
16 main :
17     movl i, %esi
18     movl j, %edx
19     movl count,%ecx
20     movq $msg1, %rdi
21     movq $0, %rax
22     call printf
23     movl i, %esi
24     movl j, %edx
25 loop :
26     call fibonachi
27     movq $msg2,%rdi
28     movq $0,%rax
29     call printf
30     movl i,%esi
31     movl j,%edx
32     incl count
33     movl count, %ecx
34     cmpl $9, %ecx
35     jl loop
36     movq $msg3,%rdi
37     movq $0, %rax
38     call printf
39     ret
40 .type fibonachi, @function

    fibonachi :
        movl j, %r8d
        addl %esi, j
        movl %r8d, i
        movl j, %esi

    ret
```

```
a201302482@localhost:~/10.18$ ./hww02
1 1  2  3  5  8  13  21  34  55  89
a201302482@localhost:~/10.18$
```

이 어셈블리어 코드는 피보나치 수열을 구현을 한 것으로 먼저 피보나치 수열의 2개의 초기값을 위해서 I,j 선언, 횟수를 정해주기 위해서 count 및 출력을 위해서 string형 변수들을 선언을 해준다. 먼저 피보나치 수열의 초기값인 1로 초기화한 I와 j를 레지스터 %rsi,%rdx에 저장을 한 후 초기값 두 개를 msg1에 저장이 된 형식되로 출력을 해준다. printf함수를 호출을 하면 레지스터의 값들이 달라 짐으로 다시 I,j의 값을 초기화 시켜주고 loop문으로 들어가자마자 fibonachi함수를 호출을 한다. j의 값을 다른곳에 임시로 저장을 한 후 j+i의 값을 j에 저장을 하고 임시로 저장이 된 j의 값을 I에 저장을 한 후 %esi에 j의 값을 저장시킴으로서 j의 값이 출력이 되게 한다. 출력을 하게되면 count의 값을 하나 증가시켜주고 맨 처음에 출력 시켜 준 것을 고려하여 Loop문을 9번을 실행을 한다. 그러면 총 10번의 피보나치 수열 연산을 한 값들을 얻을수 있다.

	%rsi	%rdx	%rcx	%rsp
loop 1	1	1	0	0x7fffffffe578
fibonachi 1	1	1	0	0x7fffffffe570
return 1	2	1	0	0x7fffffffe570
loop 2	1	2	1	0x7fffffffe578
fibonachi 2	1	2	1	0x7fffffffe570
return 2	3	2	1	0x7fffffffe570
loop 3	2	3	2	0x7fffffffe578
fibonachi 3	2	3	2	0x7fffffffe570
return 3	5	3	2	0x7fffffffe570
loop 4	3	5	3	0x7fffffffe578
fibonachi 4	3	5	3	0x7fffffffe570
return 4	8	5	3	0x7fffffffe570
loop 5	5	8	4	0x7fffffffe578
fibonachi 5	5	8	4	0x7fffffffe570
return 5	13	8	4	0x7fffffffe570
loop 6	8	13	5	0x7fffffffe578
fibonachi 6	8	13	5	0x7fffffffe570
return 6	21	13	5	0x7fffffffe570
loop 7	13	21	6	0x7fffffffe578
fibonachi 7	13	21	6	0x7fffffffe570
return 7	34	21	6	0x7fffffffe570
loop 8	21	34	7	0x7fffffffe578
fibonachi 8	21	34	7	0x7fffffffe570
return 8	55	34	7	0x7fffffffe570
loop 9	34	55	8	0x7fffffffe578
fibonachi 9	34	55	8	0x7fffffffe570
return 9	89	55	8	0x7fffffffe570

fibonachi에서 return으로 이동하면서 %rsi 는 I+j의 값이, %rdx의 값은 j의 값으로 변화 는 것을 볼수 있고 return에서 loop로 갈 때 %rcx의 값이 1씩 증가 하는 것을 볼 수 있다. 이미 앞에서 한번 출력을 하여서 loop를 9번을 수행하였다. %rsp는 함수호출시 값이 감소하고 함수의 반환시 값이 다시 증가하는 것을 볼수있었다.

과제 3

```
1  .section .data
2  scanf_format :
3      .string "%d"
4  scanf_str :
5      .string "input number 0~4 : "
6  printf_str0 :
7      .string "linux \n"
8  printf_str1 :
9      .string "gcc \n"
10 printf_str2 :
11     .string "switch \n"
12 printf_str3 :
13     .string "asm \n"
14 printf_str4 :
15     .string "gdb \n"
16 printf_str5 :
17     .string "example \n"
18 x :
19     .int 0
20 JUMP_TABLE :
21     .quad .L0
22     .quad .L1
23     .quad .L2
24     .quad .L3
25     .quad .L4
26 .section .text
27 .globl main
28 main :
29     movq $scanf_str, %rdi
30     movq $0,%rax
31     call printf
32     movq $x, %rsi
33     movq $scanf_format, %rdi
34     movq $0,%rax
35     call scanf
36     movl x, %esi
37
38     cmpl $4, %esi
39     jg DEFAULT
40     jmp *JUMP_TABLE(,%rsi,8)
```

```
.L0 :
    movq $printf_str0, %rdi
    movq $0,%rax
    call printf
    jmp END
.L1 :
    movq $printf_str1, %rdi
    movq $0,%rax
    call printf
    jmp END
.L2 :
    movq $printf_str2, %rdi
    movq $0,%rax
    call printf
.L3 :
    movq $printf_str3, %rdi
    movq $0, %rax
    call printf
    jmp def
.L4 :
    movq $printf_str4, %rdi
    movq $0, %rax
    call printf
    jmp END
DEFAULT :
def :
    movq $printf_str5, %rdi
    movq $0, %rax
    call printf
END :
    ret
```

```
a201302482@localhost:~/10.18$ ./hw03
input number 0~4 : 0
linux
a201302482@localhost:~/10.18$ ./hw03
input number 0~4 : 1
gcc
a201302482@localhost:~/10.18$ ./hw03
input number 0~4 : 2
switch
asm
example
a201302482@localhost:~/10.18$ ./hw03
input number 0~4 : 3
asm
example
a201302482@localhost:~/10.18$ ./hw03
input number 0~4 : 4
gdb
a201302482@localhost:~/10.18$ ./hw03
input number 0~4 : 5
example
a201302482@localhost:~/10.18$
```

Switch문을 Jump_table로 구현을 한 어셈블리어 코드이다. 변수로는 값을 입력을 받기 위한 변수 x, 입력의 형식과 출력의 형식을 저장을 하는 string형 변수들 및 JUMP_TABLE을 선언을한다. x의 값을 scanf함수를 이용을 하여 입력을 받은 후 x의 값을 %esi에 저장을한다. 만약 x의 값이 4보다 크다면 default label로 jump하게 되어 example이라는 문장을 출력을 하게 될 것이다. x의 값이 4 이하이면 x의 값에 따라 JUMP_TABLE의 주소에서 8씩 떨어진 테이블 중에 하나에 도착을 하게 될 것이다. x의 값이 0이면 linux라는 문장을 출력을 하고 x의 값이 1이면 gcc, x의 값이 2이면 다른곳으로 빠질 수 있는 jmp명령어가 존재 하지 않음으로 x의 값이 3일때의 문장인 asm을 출력을 하고 default label로 이동을 하여 example 문장을 출력을 할 것이다. x의 값이 4라면 gdb라는 문장을 출력을 하게 될 것이다.