시스템 프로그래밍

어셈블리어의 산술 명령, 제어문

2016.10.11

황슬아

seula.hwang@cnu.ac.kr

개요

1. 실습명
   - ✓ 어셈블리어의 산술 명령, 제어문
2. 목표
   - ✓ 어셈블리어 프로그램의 구조 이해
   - ✓ 어셈블리어의 산술 명령의 사용
   - ✓ 어셈블리 함수의 이해와 사용
3. 내용
   - ✓ 실습 1. 연산 명령어
   - ✓ 연산 명령어 – 비교 (Flag)
   - ✓ 제어문 – Jump 명령어
   - ✓ 제어문 - Loop
   - ✓ 제어문 – Switch

어셈블리어 코드 확인하기

1. mstore.c 파일을 생성하여 다음과 같이 입력한다.

```
1 long mult2(long, long);
2
3 void multstore(long x, long y, long *dest)
4 {
5     long t = mult2(x, y);
6     *dest = t;
7 }
```

파일명 : mstore.c

2. -S 옵션을 이용하여 컴파일 후, 생성된 어셈블리 코드를 확인한다. (대소문자 주의)

   1) gcc –Og –S mstore.c

   2) 생성된 파일명 : mstore.s

```
    .file   "mstore.c"
    .text
    .globl  multstore
    .type   multstore, @function
multstore:
.LFB0:
    .cfi_startproc
    pushq   %rbx
    .cfi_def_cfa_offset 16
    .cfi_offset 3, -16
    movq    %rdx, %rbx
    call    mult2
    movq    %rax, (%rbx)
    popq    %rbx
    .cfi_def_cfa_offset 8
    ret
    .cfi_endproc
.LFE0:
    .size   multstore, .-multstore
    .ident  "GCC: (Ubuntu 4.8.4-2ubuntu1~14.04.3) 4.8.4"
    .section    .note.GNU-stack,"",@progbits
```

파일명 : mstore.s

어셈블리어 코드 확인하기

3. 이번엔 -c 옵션을 이용해 컴파일한다.
   - ✓ gcc –Og -c mstore.c
4. GDB 를 이용해 mstore.o 를 디버깅한다.
   - ✓ gdb mstore.o
   - ✓ x/14xb multstore
   - ✓ 14-Byte 의 16진수를 확인할 수 있다.

```
(gdb) x/14xb multstore
0x0 <multstore>:        0x53    0x48    0x89    0xd3    0xe8    0x00    0x00    0x00
0x8 <multstore+8>:      0x00    0x48    0x89    0x03    0x5b    0xc3
(gdb)
```

5. GDB 종료 후, 다음 명령어를 이용해 수행해본다.
   - ✓ objdump –d msotre.o
   - ✓ 앞서 수행한 결과인 14B가 어셈블리어와 대응되는 것을 확인할 수 있다.

```
student@localhost:~/test/m_book$ objdump -d mstore.o

mstore.o:     file format elf64-x86-64


Disassembly of section .text:

0000000000000000 <multstore>:
   0:   53                      push   %rbx
   1:   48 89 d3                mov    %rdx,%rbx
   4:   e8 00 00 00 00          callq  9 <multstore+0x9>
   9:   48 89 03                mov    %rax,(%rbx)
   c:   5b                      pop    %rbx
   d:   c3                      retq
```

실습1. 연산 명령어

1. 연산 명령 addq 를 이용하여 아래 프로그램과 같은 연산을 수행하고, printf 를 이용하여 출력하여라.

실행결과

```
sysOO@localhost:~/lab05/student$ gcc -o ex01 ex01.s
sysOO@localhost:~/lab05/student$ ./ex01
val1= 100 val2 =  200 result = 300
```

```
.section .data
message :
    .string "val1= %d val2 =  %d result = %d ₩n"
val1:
    .int 100
val2:
    .int 200


.section .text
.globl main

main:
    # move a message to a register(=reg) rdi
    movq    $message, %rdi

    # move variables val1 & val2 to reg rsi & rdx
    movq    val1, %rsi
    movq    val2, %rdx

    # add  rsi(val1) & rdx(val2), rdx = rdx + rsi
    addq    %rsi, %rdx

    # move the result to reg rcx
    # since val2 was overlapped by the result, move val2 to rdx
    movq    %rdx, %rcx
    movq    val2, %rdx

    # call function printf
    movq    $0, %rax
    call    printf

    #return
    ret
```

소스 파일명 : ex01.s
실행 파일명 : ex01

과제 1

- 앞서 따라하기에서 작성했던 addq 연산을 subq, imulq, incq, decq, xorq, andq 로 변경하여 작성 후, 각각의 결과를 출력하라.
- 소스파일 명 : hw01.s
- 실행파일 명 : hw01

Instruction	Effect	Description
subq S, D	D <- D - S	뺄셈
mulq S, D	D <- D * S	곱셈
incq D	D <- D + 1	증가
decl D	D <- D - 1	감소
andl S, D	D <- D & S	AND
xorl S, D	D <- D ^ S	Exclusive OR

실습1. 연산 명령어

2. /home/ubuntu/lab05/ex03.s 를 자신의 홈 디렉토리에 복사하여 다음을 수행하여라.
   - ✓ 명령어 leaq 에 대한 과제이다. 완성되어 있는 1, 2) 를 참고하여 (3), (4)의 결과가 어떠할지 예측하고 빈 칸을 완성하여 결과를 비교한다.

```
.section .data
message :
    .string "Register rsi has... 0x%x #n#n"


.section .text
.globl main

main:
    # set values
    movq    $0x1234, %rdx
    movq    $message, %rdi

    # (1)
    leaq    (%rdx), %rsi
    movq    $0, %rax
    call    printf

    # set values
    movq    $0x1234, %rdx
    movq    $message, %rdi

    # (2)
    leaq    6(%rdx), %rsi
    movq    $0, %rax
    call    printf

    # set values
    movq    $0x1234, %rdx
    movq    $message, %rdi
```

```
    # (3) (rdx + rdx)
    leaq          , %rsi  #fill in the blank
    movq    $0, %rax
    call    printf

    # set values
    movq    $0x1234, %rdx
    movq    $0x4321, %rcx
    movq    $message, %rdi

    # (4)   (rdx + rcx * 4) + 6
    leaq        , %rsi      #fill in the blank
    movq    $0, %rax
    call    printf

    ret
```

소스 파일명 : ex03.s
실행 파일명 : ex03

과제 2

1) 아래에 제시된 sarq 를 수행하는 코드를 작성하고 결과를 출력하라.

2) shrq 를 수행하는 코드도 작성하여 출력을 보이고 sarq 과 shrq 의 차이를 설명하라.

```
.section .data
message :
    .string "val= %d ->> result = %d \n"
val:
    .int 100

.section .text
.globl main

main:
    # move a message to a register(=reg) rdi
    movq    $message, %rdi

    # move variables val to reg rsi
    movq    val, %rsi

    # sarq
    sarq    $2, %rsi

    #save the result
    movq    %rsi, %rdx
    movq    val, %rsi

    # call function printf
    movq    $0, %rax
    call    printf
    movq    $0, %rax

    #return
    ret
```

소스 파일명 : hw02.s
실행 파일명 : hw02

실습2. scanf와 printf

```
.section .data
scanf_format:
    .string "%d"
printf_format:
    .string "your input number : %d \n"
input:
    .int 0

.section .text
.globl main

main:

    movq    $input, %rsi
    movq    $scanf_format, %rdi

    # call function scanf
    movq    $0, %rax
    call    scanf
    movq    input, %rsi

    # call function printf
    movq    $printf_format, %rdi
    movq    $0, %rax
    call    printf

    #return
    ret
```

- ✓ scanf_format 은 scanf로 입력을 받을 형식을 저장
- ✓ prinft_format 은 printf 로 출력할 형식을 저장
- ✓ input은 입력 값을 저장한 변수

- ✓ scanf 로 입력을 받는다
- ✓ 이때 입력 받은 값은 변수 input에 저장되기 때문에 레지스터 rsi 로 옮겨 줘야한다.

- ✓ 입력 받은 값을 printf 로 출력!

실습3. scanf와 printf

1. 앞의 코드를 작성하고 컴파일하여라.
2. 실행 후, scanf 와 printf 사용법을 숙지하여라.

✓ **컴파일 및 실행 결과**

```
sys00@localhost:~/lab05/TA$ vi ex03.s
sys00@localhost:~/lab05/TA$ gcc -o ex03 ex03.s
sys00@localhost:~/lab05/TA$ ./ex03
1234
your input number : 1234
```

Setting Condition Codes – CMP 명령 이용

1. 비교 연산자를 통한 조건(상태) 플래그 설정
   - 오버 플로우, 제로, 부호, 캐리 플래그 등이 있다.
2. cmpq Dest, Src
   - cmpq Dest, Src는 Src – Dest 연산을 통해 값을 비교함
3. 조건 플래그 (Condition Flag)
   1) CF(Carry Flag) : MSB(Most Significant Bit)로 부터의 자리 올림(carry) 혹은 빌림(borrow)이 발생할 경우에 1로 설정
      - 부호 없는 산술 연산에서 오버 플로우를 검출할 때 사용
   2) ZF(Zero Flag) : dest 와 src 의 값이 같은 경우(Src– Dest == 0) 1로 설정
   3) SF(Sign Flag) : Src – Dest 의 부호를 나타낸다. 0은 양수, 1은 음수.
   4) OF(Overflow Flag) : 부호 있는 연산 결과가 오버 플로우가 발생된 경우. 오버 플로우가 발생한 경우 1로 설정.
      - 양수/음수의 2의 보수 오버 플로우를 발생시킨 것을 표시

실습 5. Jump 명령의 이용 - 분기

1. jx Label – if-else, while등 조건 문으로 사용 가능
   - 컨디션 코드들에 따라서 조건 분기 및 무조건 분기

jx	Condition	Description
jmp	1	무조건 분기
je	ZF (ZF = 1)	ZF가 1인 경우(Dest == Src)
jne	~ZF	ZF가 0인 경우(Dest != Src)
js	SF	SF가 1인 경우(음수)
jns	~SF	SF가 0인 경우(양수)
jg	~(SF^OF)&~ZF (ZF = 0 and SF == OF)	큰 경우 > (Signed)
jge	~(SF^OF)	크거나 같은 경우 ≥ (Signed)
jl	(SF^OF)	작은 경우 < Signed)
jle	(SF^OF)|ZF	작거나 같은 경우 ≤ (Signed)
ja	~CF&~ZF (CF = 0 and ZF = 0)	앞의 숫자가 큰 경우 > (Unsigned)
jb	CF	뒤의 숫자가 큰 경우 < (Unsigned)

실습 5. Jump 명령의 이용 - 분기

1. 다음은 두 수를 입력 받아 둘 중 더 큰 수를 출력하는 프로그램이다.

```
.section .data
scanf_str:
    .string "%d %d"

printf_str:
    .string "%d is greater ₩n"

val1:
    .int 0
val2:
    .int 0

.section .text
.globl  main
main:
    movl    $val1, %esi
    movl    $val2, %edx
    movq    $scanf_str, %rdi

    movq    $0, %rax
    call    scanf

    # move results to register to compare
    movl    val1, %esi
    movl    val2, %edx

    #compare
    cmpl    %edx, %esi

    #choose the greater one to print
    jg  greater
    movl    %edx, %esi

greater:
    movq    $printf_str, %rdi
    movq    $0, %rax
    call    printf

    ret
```

실행 결과

```
sys00@localhost:~/lab05/TA$ ./ex04
10 15
15 is greater
```

과제 4, 과제 5

1. **과제 4**

   1) 두 숫자 중 더 작은 수를 판별하는 어셈블리어 코드를 작성하세요.

      - 사용자로부터 두 수를 입력 받고, 작은 수를 출력

```
sysOO@localhost:~/labO5/TA$ gcc -o less less.s
sysOO@localhost:~/labO5/TA$ ./less
3 5
3 is less
```

2. **과제 5**

   1) 두 수가 같은지 판별하는 어셈블리어 코드를 작성하세요.

      - 사용자로부터 두 수를 입력 받고, 두 수가 같은지 판별하고 출력

```
sysOO@localhost:~/labO5/TA$ gcc -o exO4 exO4.s
sysOO@localhost:~/labO5/TA$ ./exO4
1O 1O
1O and 1O are equal
sysOO@localhost:~/labO5/TA$ ./exO4
1O 15
1O and 15 are not equal
sysOO@localhost:~/labO5/TA$
```

실습 6. Loop의 이용

1. 사용자로부터 n을 입력 받아 0에서부터 n까지 합을 출력하는 프로그램이다.
   - 다음을 입력하여 아래와 같이 결과를 출력.

실행 결과

```
sys00@localhost:~/lab05/TA$ ./ex05
10
result : 55
```

```
.section .data
scanf_str:
    .string "%d"
printf_str:
    .string "result : %d #n"
i:
    .int 0
sum:
    .int 0
n:
    .int 0

.section .text
.globl  main

main:

    movl    $n, %esi
    movq    $scanf_str, %rdi

    movq    $0, %rax
    call    scanf

    movl    sum, %edx
    movl    i, %ecx

loop:
    addl    %ecx, %edx      #sum += i
    incl    %ecx            #i++
    cmpl    n,  %ecx        #
    jle     loop            #if ecx(i) <= n , jump


    movl    %edx, %esi      #print sum

    movq    $printf_str, %rdi
    movq    $0, %rax
    call    printf
    ret
```

과제 6

1. m과 n을 입력 받은 후, m의 n제곱을 계산하는 어셈블리 코드 작성
   1) 사용자로부터 두 수를 입력 받고,
   2) Loop 문을 이용하여 제곱 연산을 수행한다.
   3) 0승이면 1을 출력하게 한다.

실습 7. Switch 문의 구현

- switch 문은 if … else 형태 로 구현이 가능하다.

```c
#include <stdio.h>

int main(){

    int x = 1;
    int ch = 0;

    switch(x){
        case 0:
            ch = 65;
            break;
        case 1:
            ch = 66;
            break;
        default:
            ch = 0;
    }

    printf("result: %d \n",  ch);
}
```

C 언어의 switch 문

```
.section .data
printf_str:
    .string "result : %d \n"
x:
    .int 1

.section .text
.globl  main

main:

    movl    x, %ebx

    cmpl    $0, %ebx
    movl    $65,%esi
    je      END

    cmpl    $1, %ebx
    movl    $66, %esi
    je      END

    movl    $0, %esi

END:
    #movl   %eax, %rsi
    movq    $printf_str, %rdi
    movq    $0, %rax
    call    printf
    ret
```

if…else 형태

과제 7

1. 다음은 c 언어로 짜여진 프로그램을 역어셈블 한 결과이다. 원래 어떤 프로그램인지 추측하여 c 언어로 복구하여라.

```
.LC0:
    .string "%d %d"
.LC1:
    .string "b is bigger than b "
.LC2:
    .string "a is bigger than a "
.LC3:
    .string "a and b are equal "
    .text
    .globl  main
    .type   main, @function
main:
.LFB24:
    .cfi_startproc
    subq    $24, %rsp
    .cfi_def_cfa_offset 32
    movl    $0, 8(%rsp)
    movl    $0, 12(%rsp)
    leaq    12(%rsp), %rdx
    leaq    8(%rsp), %rsi
    movl    $.LC0, %edi
    movl    $0, %eax
    call    __isoc99_scanf
    movl    12(%rsp), %eax
    addl    $1, %eax
    movl    %eax, 12(%rsp)
    movl    8(%rsp), %edx
    cmpl    %edx, %eax
    jle .L2
    movl    $.LC1, %edi
    call    puts
    jmp .L3
.L2:
    cmpl    %edx, %eax
    jge .L4
    movl    $.LC2, %edi
    call    puts
    jmp .L3
.L4:
    movl    $.LC3, %edi
    call    puts
.L3:
    movl    $0, %eax
    addq    $24, %rsp
    .cfi_def_cfa_offset 8
    ret
    .cfi_endproc
```

✓ **힌트 : 예상되는 c 프로그램을 코딩하여 역어셈블하여 비교해본다.**

과제

1. 과제를 진행한 내용을 모두 보고서로 작성하여 **사이버캠퍼스와 서면**으로 제출 **(코드도 포함)**
   1) 파일 제목 : [sys00]HW05_학번_이름
   2) 반드시 파일 제목과 파일 양식을 지켜야 함. **(위반 시 감점)**
   3) **보고서는 제공된 양식 사용**
   4) **설명(해결방법) 필수!!**

2. 자신이 실습한 내용을 증명할 것 (결과 화면 – 자신의 학번이 보이도록)

3. **제출 일자**
   1) 사이버캠퍼스 : 2016년 10월 11일 08시 59분 59초
   2) 서면 : 2016년 10월 11일 수업시간