시스템 프로그래밍

- 어셈블리어의 데이터 이동 및 연산명령, 제어문 -

2014. 10. 02.

박시형

sihyeong@cnu.ac.kr

Embedded System Lab. Computer Engineering Dept.

Chungnam National University

개요

실습 명

- 어셈블리어의 데이터 이동 및 연산명령, 제어문

목표

- 어셈블리 프로그램의 구조 이해
- 데이터 이동 및 연산명령의 사용
- 어셈블리 함수의 이해와 사용

내용

- 실습 1. Hello, CNU!
- 실습 2. 데이터의 이동명령어 이해
- 실습 3. 출력 함수의 이용
- 실습 4. 연산명령어
- 실습 5. 함수의 이용
- 실습 6. Loop의 이용
- 실습 7. Switch 문의 구현

어셈블리어 ?

- 어셈블리어(Assembly Language)는 0과 1의 이진수 프로그래밍을 좀더 편하게 하기 위해 비트 패턴을 명령어로 만든 언어
  - 하드웨어 디바이스 드라이버, 일반 프로그램의 특정기능 최적화 등에 사용
- 어셈블리어는 시스템의 구조에 따라 문법과 명령어 셋 등이 다르다
  - masm(MS), TASM(Borland), NASM(open source), GAS(GNU)
- 실습시간에는 GNU 소프트웨어인 GAS 어셈블리어를 사용한다.
  - 자세한 내용은 24p '참고 – 어셈블리어' 를 참고

어셈블리어의 장/단점

- 어셈블리어의 **장점**
  - **기계와 바로 통신**이 가능함
  - 명령 실행 **속도가 빠름**
  - 프로그램 **크기가 작음**

- 어셈블리어의 **단점**
  - **배우기 어려움**
  - 큰 프로그램을 만들기 힘듦
  - 프로그램 작성시간과 버그를 잡기 힘듦
  - **하드웨어 별로 특성이 다름**

기본 레지스터

- **ESP**
  - Extend Stack Pointer : Stack의 상위 주소를 가리키는 레지스터
- **EBP**
  - Extend Base Pointer : Stack의 Base 주소를 가리키는 레지스터
- **EIP**
  - Extend Instruction Pointer : 실행 할 명령의 주소를 가리키는 레지스터.
  - 각각 명령이 실행될 때, EIP에 CPU가 현재 실행하고 있는 주소가 저장됨

어셈블리어 기본 구조

.section

- data, text 등의 메모리 영역을 지정
- 일반적으로 코드는 text 영역
- 전역 변수, 정적 변수, 배열, 구조체 등은 data 영역에 저장

label

- c언어의 goto 구문과 같이 주소를 저장해 주는 포인터 같은 역할

.global main

- 프로그램의 시작점

```
.section .data
label :

.section .text

.global main
main :


ret
```

실습 1. Hello, CNU!

❖ 따라 하기

- 어셈블리어를 이용해서 "Hello, CNU!" 라는 메시지를 출력하는 프로그램을 아래의 과정을 따라 작성

소스코드

```
.section .data
message :
        .string "Hello, CNU! \n"

.section .text
.global main
main :
        push    %ebp
        movl    %esp, %ebp
        pushl   $message
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

소스파일명 : ex01.s
실행파일명 : ex01.out

컴파일 및 실행

```
[c000000000@eslab ~]$ vi ex01.s
[c000000000@eslab ~]$ gcc -o ex01.out ex01.s
[c000000000@eslab ~]$ ./ex01.out
Hello, CNU!
[c000000000@eslab ~]$
```

실습 1. Hello, CNU!

과제

- 따라 하기를 참고하여 자신의 학번과 영문이름을 출력하는 프로그램을 작성

- 컴파일 및 실행 결과

```
[c0000000000@eslab ~]$ gcc -o hw01.out hw01.s
[c0000000000@eslab ~]$ ./hw01.out
201300000 Han Dong Geon
```

실습 2. 출력 함수의 사용법

❖ 따라 하기

- 어셈블리어로 아래의 C언어 스타일과 같은 형식의 printf 결과를 보이는 실습을 따라 해본다.
  - printf("val1=%d, val2=%d, val3=%d₩n", val1, val2, val3);

소스코드

```
.section .data
message:
        .string "val1=%d, val2=%d, val3=%d\n"
val1:
        .int 100
val2:
        .int 200
val3:
        .int 300

.section .text
.global main
main:
        push    %ebp
        movl    %esp, %ebp
        pushl   val3
        pushl   val2
        pushl   val1
        pushl   $message
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

실행결과

```
[c000000000@eslab ~]$ ./ex02.out
val1=100, val2=200, val3=300
```

소스파일명 : ex02.s
실행파일명 : ex02.out

실습 2. 출력 함수의 사용법

과제

- 앞의 따라 하기를 바탕으로 자신의 키와 나이를 출력하는 어셈블리어 프로그램을 작성하세요. 단, 함수호출을 이용하여 아래와 같이 printf 함수를 호출해서 구현

  - printf("My height is %d cm and my age is %d years old₩n", my_height, my_age);

실습 3. 연산 명령어

❖ 따라 하기

- 더하기 연산 명령 addl을 이용하여 100과 200을 더하는 프로그램을 아래와 같이 작성하고 printf를 이용하여 결과를 출력

소스코드

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

.section .text
.global main
main:
        push    %ebp
        movl    %esp, %ebp
        movl    val1, %eax      # val1 -> eax
        movl    val2, %ebx      # val2 -> ebx
        addl    %ebx, %eax      # val1 + val2
        pushl   %eax
        pushl   val2
        pushl   val1
        pushl   $message
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

실행결과

```
[c000000000@eslab ~]$ ./ex03.out
100 + 200 = 300
```

소스파일명 : ex03.s
실행파일명 : ex03.out

실습 3. 연산 명령어

❖ 과제 1

- 앞서 따라 하기에서 작성했던 addl 연산을 subl, imull, incl, decl, xorl, andl 로 변경하여 작성 후, 결과를 출력

Instruction	Effect	Description
subl S, D	D <- D - S	뺄셈
imul S, D	D <- D * S	곱셈
incl 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

- S : Source
- D : Destination

실습 3. 연산 명령어

❖ 과제 2

- 다음 두 개의 소스를 작성 후 결과를 각각 출력하고 sarl 과 sall 의 역할과 연산과정을 보고서에 작성하세요.
  - sarl, sall : Arithmetic shift. 자세한 부분은 교과서 참고

소스코드1

```
.section .data
message:
        .string "%d 를 %d 만큼 Shift = %d\n"
val1:
        .int 27

.section .text
.global main
main:
        push    %ebp
        movl    %esp, %ebp
        movl    val1, %eax      # val1 -> eax
        sarl    $4, %eax
        pushl   %eax            # 연산결과 저장
        pushl   $4
        pushl   val1
        pushl   $message
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

소스코드2

```
.section .data
message:
        .string "%d 를 %d 만큼 shift = %d\n"
val1:
        .int 27

.section .text
.global main
main:
        push    %ebp
        movl    %esp, %ebp
        movl    val1, %eax      # val1 -> eax
        sall    $4, %eax
        pushl   %eax            # 연산결과 저장
        pushl   $4
        pushl   val1
        pushl   $message
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

실습 3. 연산 명령어

❖ 과제 3

- 다음 소스코드를 작성하여 결과값을 출력하고 왜 이런 결과값이 나왔는지를 보고서에 설명하세요.
  - leal 명령어는 교과서를 참조.

소스코드

```
.section .data
message:
        .string "%d\n"
val1:
        .int 100

.section .text
.global main
main:
        push    %ebp
        movl    %esp, %ebp
        leal    val1, %eax
        pushl   %eax
        pushl   $message
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

컴파일 및 실행 결과

```
[c000000000@eslab week4]$ gcc -o leal.out leal.s
[c000000000@eslab week4]$ ./leal.out
134520864
```

※ 결과값이 틀릴 수도 있음

실습 4. scanf 와 printf

❖ 따라 하기

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

.section .text
.global main
main:
        pushl   %ebp
        movl    %esp, %ebp

        pushl   $input
        pushl   $scanf_format
        call    scanf
        pushl   input
        pushl   $printf_format
        call    printf

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

❖ scanf_format은 scanf로 입력을 받을 형식을 저장
❖ printf_forma은 printf로 출력할 형식을 저장
❖ Input은 입력 값을 저장한 변수

❖ setup 부분

❖ scanf로 입력을 받는다
❖ 입력 받은 값을 printf로 출력

❖ setup 부분

실습 4. scanf 와 printf

❖ 따라 하기

- 앞의 코드를 작성하고 컴파일 하세요. 실행 후, scanf의 사용법과 printf 사용법을 익히세요.

- 컴파일 및 실행 결과

```
[c0000000000@eslab week4]$ gcc -o ex04.out ex04.s
[c0000000000@eslab week4]$ ./ex04.out
3
your input number : 3
```

Setting Condition Codes - CMP 명령 이용

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

실습 5. Jump 명령의 이용 - 분기

- 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

❖ 따라 하기

- 두 수를 입력 받아 둘 중에 더 큰 수를 출력한다.

```
.section .data
scanf_str:
        .string "%d %d"
printf_str:
        .string "%d is greater.\n"

val1:
        .int 0
val2:
        .int 0

.section .text
.global main
main:
        #setup 부분
        pushl   %ebp
        movl    %esp, %ebp

        #비교할 두 수를 입력 받음
        pushl   $val2
        pushl   $val1
        pushl   $scanf_str
        call    scanf

        #입력받은 두 수를 비교하기
        #위해서 레지스터로 이동
        movl    val1, %eax
        movl    val2, %ebx
```

```
        #두 수를 비교한다.
        cmpl    %ebx, %eax

        #앞의 수가 크면
        #greater로 점프한다.
        jg      greater
        movl    %ebx, %eax

greater:
        #eax에 들어있는
        #큰 수를 출력한다.
        pushl   %eax
        pushl   $printf_str
        call    printf

        #finish 부분
        movl    %ebp, %esp
        popl    %ebp
        ret
```

```
[c000000000@eslab week4]$ gcc -o ex05.out ex05.s
[c000000000@eslab week4]$ ./ex05.out
10 100
100 is greater.
[c000000000@eslab week4]$ ./ex05.out
100 20
100 is greater.
```

실습 5. Jump 명령의 이용

과제 1

- 두 숫자 중 더 작은 수를 판별하는 어셈블리어 코드를 작성하세요.
  - 사용자로부터 두 수를 입력 받고, 작은 수를 출력

```
[c000000000@eslab week4]$ gcc -o less.out less.s
[c000000000@eslab week4]$ ./less.out
10 100
10 is lesser.
[c000000000@eslab week4]$ ./less.out
20 10
10 is lesser.
```

과제 2

- 두 수가 같은지 판별하는 어셈블리어 코드를 작성하세요.
  - 사용자로부터 두 수를 입력 받고, 두 수가 같은지 판별하고 출력

```
[c000000000@eslab week4]$ gcc -o equal.out equal.s
[c000000000@eslab week4]$ ./equal.out
10 200
10 and 200 are different.
[c000000000@eslab week4]$ ./equal.out
10 10
10 and 10 both are equal.
```

실습 6. Loop의 이용

❖ 따라 하기

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

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

.section .text
.global main
main
        pushl   %ebp
        movl    %esp, %ebp

        pushl   $n
        pushl   $scanf_str
        call    scanf
        movl    sum, %eax
        movl    i, %ebx
```

```
loop:
        addl    %ebx, %eax      # sum += i
        incl    %ebx            # i++
        cmpl    n, %ebx         # n과 i 비교
        jle     loop            # i < n 이면 점프

        pushl   %eax
        pushl   $printf_str
        call    printf

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

```
[c000000000@eslab week4]$ gcc -o ex06.out ex06.s
[c000000000@eslab week4]$ ./ex06.out
10
result : 55
```

실습 6. Loop의 이용

과제

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

실습 7. Switch 문의 구현

❖ 따라 하기

- switch 문은 2가지 형태로 구현가능
  - if ... else 형태
  - jump table 형태

```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);
}
```

C언어에서의 switch 문

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

.section .text
.global main
main:
        pushl   %ebp
        movl    %esp, %ebp

        movl    x, %ebx

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

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

        movl    $0, %eax

END:
        pushl   %eax
        pushl   $printf_str
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

if ... else 형태로 표현

실습 7. Switch 문의 구현

❖ 따라 하기

- switch 문은 2가지 형태로 구현가능
  - if ... else 형태
  - jump table 형태

```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);
}
```

C언어에서의 switch 문

```
.section .data
printf_str:
        .string "result: %c\n"
x:
        .int 1
JUMP_TABLE:
        .long .L0
        .long .L1

.section .text
.global main
main:
        pushl   %ebp
        movl    %esp, %ebp

        movl    x, %ebx
        cmpl    $1, %ebx
        jg      DEFAULT
        jmp     *JUMP_TABLE(, %ebx, 4)
.L0:
        movl    $65, %eax
        jmp     END
.L1:
        movl    $66, %eax
        jmp     END
DEFAULT:
        movl    $0, %eax
END:
        pushl   %eax
        pushl   $printf_str
        call    printf
        movl    %ebp, %esp
        popl    %ebp
        ret
```

jump table 형태로 표현

실습 7. Switch 문의 구현

❖ 과제

- 왼쪽의 C 코드의 기능과 같은 어셈블리어 코드를 작성하세요.

```c
#include <stdio.h>

void main(){
        int x=0;

        printf("input number 0~4 : ");
        scanf("%d", &x);

        switch(x){
                case 0:
                        printf("linux\n");
                        break;
                case 1:
                        printf("gcc\n");
                        break;
                case 2:
                        printf("switch\n");
                case 3:
                        printf("asm\n");
                        goto def;
                case 4:
                        printf("gdb\n");
                        break;
                default:
                        def: printf("example\n");
        }
}
```

어셈블리어 코드로 작성할 때,
if...else 와 **jump table** 형태 모두 작성
(단, break가 없는 부분과 goto 문에 유의하여 코드를 작성)

실행 결과

```
[cOOOOOOOOO@eslab week4]$ ./switchEX
input number 0~4 : 0
linux
[cOOOOOOOOO@eslab week4]$ ./switchEX
input number 0~4 : 2
switch
asm
example
[cOOOOOOOOO@eslab week4]$ ./switchEX
input number 0~4 : 3
asm
example
```

참고 - 어셈블리어

- x86 인스트럭션 셋 참고 사이트
  - https://wiki.kldp.org/HOWTO//html/Assembly-HOWTO/index.html
  - 주의사항
    - 우리는 GAS 어셈블러를 사용하기 때문에 인텔 어셈블리어의 문법규칙이 아니라 AT&T의 문법규칙을 따른다. 그렇기 때문에 위의 사이트에서 알아낸 인스트럭션의 문법을 그래도 써서는 안 된다.
    - AT&T와 인텔의 구문은 source와 destination을 반대로 사용한다. 예를 들면 다음과 같다.
      - 인텔 : mov eax, 4
      - AT&T : movl $4, %eax
    - AT&T 구문에서는 직접 피연산자는 $로 시작한다. 인텔 구문에서는 그렇지 않다. 예를 들면 다음과 같다.
      - 인텔 : push 4
      - AT&T : pushl $4
    - AT&T 구문에서 메모리 피연산자의 크기는 opcode 이름의 마지막 글자로 결정된다. opcode 는 b(8bits), w(16bits), l(32bits)로 각각 정해진 메모리 참조를 나타낸다. 인텔 구문은 메모리 피연산자 접두어(byte ptr, word ptr, dword ptr)로 결정된다. 다음 예를 살펴보자.
      - 인텔 : mov al, byte ptr foo
      - AT&T : movb foo, %al

제출사항

- 따라 하기와 과제를 진행한 내용을 모두 보고서로 작성하여 **이메일과 서면으로 제출**
  - sihyeong@cnu.ac.kr
  - 제목 양식 : [sys02]HW04_학번_이름
  - 파일 제목 : [sys02]HW04_학번_이름
  - 반드시 메일 제목과 파일 양식을 지켜야 함. **(위반 시 감점)**
  - **보고서는 제공된 양식 사용**

- 자신이 실습한 내용을 증명할 것 (자신의 학번이 항상 보이도록)

- **제출 일자**
  - **이메일 :** 2014년 10월 8일 23시 59분 59초
  - **서면 :** 2014년 10월 16일 수업시간
    - **2014년 10월 9일 한글날 휴강**