SSE2030: Introduction to Computer Systems (Fall 2010)

Programming Assignment #2:

Simple assembly program in x86 assembly language

Due: October 27, 11:59PM.

1. Introduction

이 과제의 목적은 리눅스 환경에서 어셈블리 프로그램에 익숙해지도록 하는데 있다. 이번 과제는 주어진 입력 값 N에 따라 0 또는 $2^{|N|}$-1을 리턴 하는 프로그램을 x86 어셈블리 언어로 구현하는 것이다.

2. Problem Specification

2.1. Overview

인자로 넘어온 값에 따라 0 또는 $2^{|N|}$-1을 리턴 하는 함수 simpleASM()을 구현한다. simpleASM() 함수는 1개의 정수를 인자로 받고 결과값을 반환한다. 여기서 결과값은 인자로 주어진 N 값에 따라 아래의 조건으로 생성한다.

$$\text{Return Value} = \begin{cases} 0 & (N \geq 0) \\ 2^{|N|} - 1 & (N < 0) \end{cases}$$

과제로 구현해야 할 simpleASM() 함수의 원형은 아래와 같다.

```
int simpleASM (int N);
```

이번 과제는 위에서 언급한 역할을 수행하는 simpleASM() 코드를 x86 어셈블리 언어로 구현하는 것이며, simpleASM() 는 함께 제공된 C 코드로 구현된 main() 함수에서 호출된다.
main() 함수는 제공된 main.c 파일에 모두 구현되어 있으며, simpleASM() 의 인자로 주어지는 정수는 main()함수에서 사용자로부터 직접 입력 받고, 입력 받는 정수는 (-30 ≤ N ≤ 30)으로 제한한다. 이 때, 입력 받는 정수가 지정된 범위(-30 ≤ N ≤ 30)를 벗어날 경우 simpleASM() 함수가 호출되지 않으므로 simpleASM() 함수 내에서 별도의 에러 처리를 할 필요는 없다.

2.2. Skeleton Codes

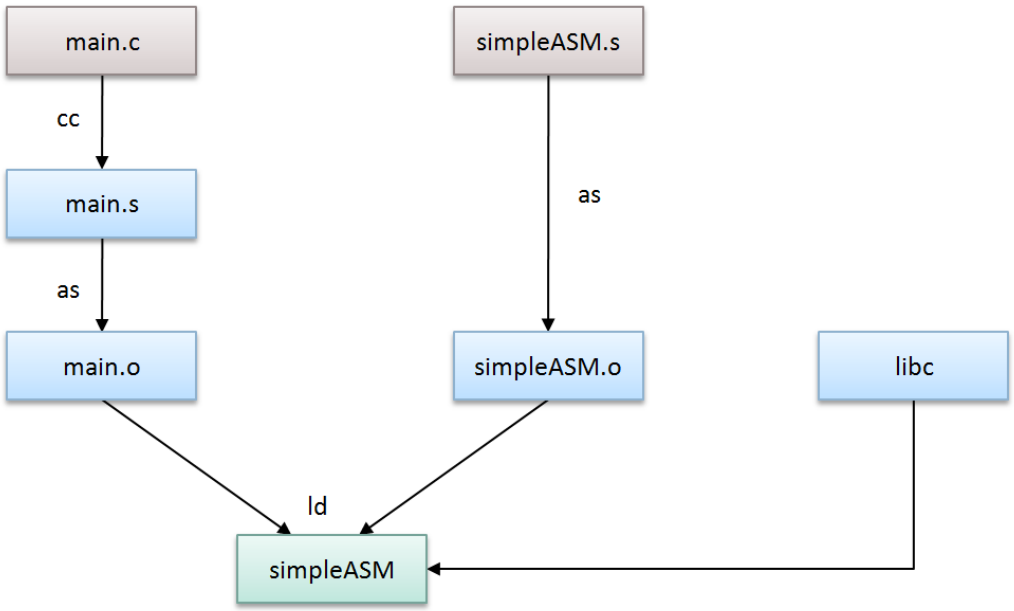
본 과제 수행을 위하여 아래와 같이 3개의 파일이 주어진다.

Makefile:	GNU make utility를 위해 사용되는 파일.
main.c:	simpleASM() 함수를 호출하는 main() 함수가 구현되어있는 파일.
simpleASM.s:	simpleASM() 함수 구현을 위한 기초 어셈블리 코드가 구현된 파일. 기본적으로 주어지는 어셈블리 코드 파일 내에서 처음 두 명령어(pushl, movl)과 마지막 세 개의 명령어(movl, popl, ret)는 수정하지 않고, simpleASM() 함수가 역할을 수행하는 메인 부분만 구현한다.

simpleASM() 함수 구현을 마친 후, 최종 실행파일 생성은 GNU make utility를 이용한다. make utility의 사용법은 아래와 같다.

```
$  make
```

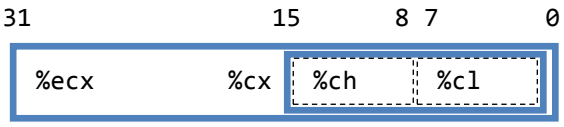
GNU make utility는 한 개의 프로그램을 구성하는 1개 이상의 파일에서 각 파일간의 종속관계를 파악하여 미리 규칙이 기술된 파일(Makefile)에 따라 컴파일 혹은 쉘 명령을 순차적으로 수행시키는 파일 관리 유틸리티이다. 본 과제 수행을 위해 제공된 코드를 통해 살펴보면, main.c 파일은 C 컴파일러에 의해 main.o로 변환되고, simpleASM.s 파일은 어셈블러에 의해 simpleASM.o파일로 변환된다. 이렇게 생성된 두 개의 오브젝트 파일 main.o와 simpleASM.o는 링커에 의해 필요한 라이브러리들을 추가하여 최종적인 실행파일인 simpleASM으로 만들어진다.



과제 수행을 위한 skeleton code는 본 수업의 홈페이지인 http://csl.skku.edu/SSE2030F10에서 다운받을 수 있다.

2.3. Background (shift operation)

Shift operation은 크게 Logical operation과 Arithmetic operation으로 구분할 수 있으며, 두 개의 operand를 갖는다. 첫 번째 operand는 얼마나 shift를 시킬지 그 값을 1바이트 이내(0~31)의 값으로 적고, 두 번째 operand는 shift 시킬 레지스터를 기록한다.
여기서 첫 번째 operand는 1바이트 값으로 기록하기 때문에 0~31의 값을 직접 기록하거나, single-byte register인 %cl 레지스터를 이용한다.



%cl 레지스터는 %ecx레지스터의 하위 1바이트로 아래와 같은 형태를 갖는다.
하위 2바이트에 독립적으로 읽기, 쓰기 연산을 할 때는 %cx 레지스터를 이용하고, 하위 1바이트에 독립적인 연산이 필요할 경우 %ch 혹은 %cl을 사용한다.
shift연산에서는 0~31 사이의 값 표현을 위해 가장 하위 1바이트 레지스터인 %cl을 사용하는 것이다.

예를 들어

1. %eax를 Immediate value를 이용하여 오른쪽으로 2비트 Arithmetic shift 시키는 경우

```
sarl    $2, %eax
```

2. %eax를 Register %cl을 이용하여 오른쪽으로 2비트 Arithmetic shift 시키는 경우

```
movl    $2,  %ecx
sarl    %cl, %eax
```

와 같은 형태로 사용할 수 있다.

2.4. Restrictions

- simpleASM() 구현을 위해 사용되는 x86 Assembly Operation은
  몇 개의 Moving data Operation, Arithmetic/Logical Operation으로 제한한다.
  사용 가능한 Operation 목록은 아래와 같다.

Data transfer Operation
movl

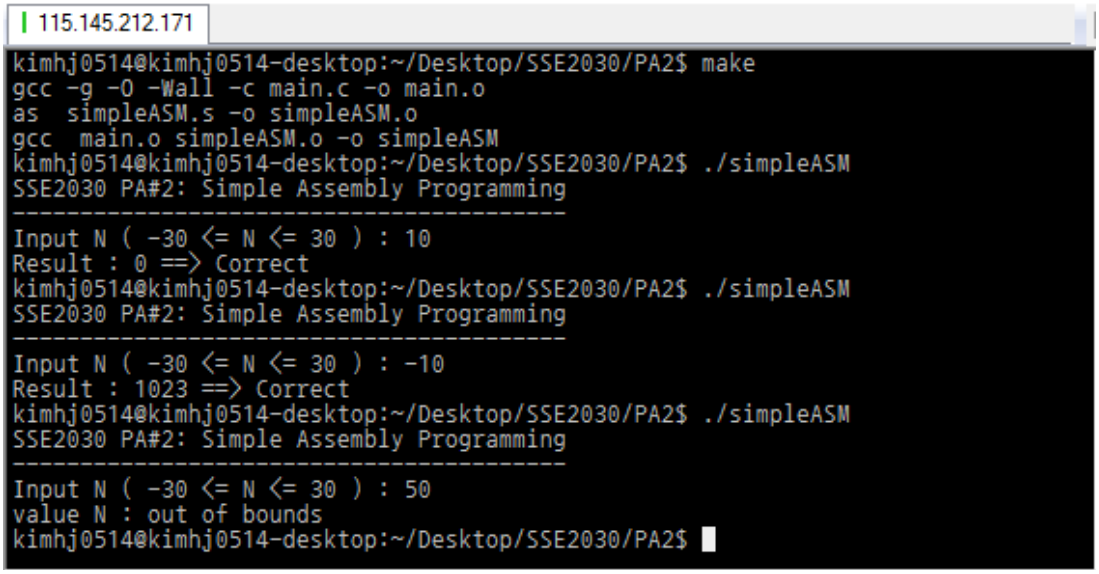
Arighmetic/Logical Operation
addl, subl, imull, sall, sarl, shll, shrl, xorl, andl, orl
incl, decl, negl, notl

- simpleASM() 함수의 setup, finish 코드를 제외한 메인 부분 코드 작성시, 사용 가능한 레지스터는 %eax, %ecx, %edx 총 3개로 제한한다.

- 과제는 본인이 직접 설치한 리눅스 환경에서 수행하고, 과제 보고서에 본인의 리눅스 환경에서 컴파일하고, 실행한 화면을 캡쳐하여 추가한다.

- `simpleASM()` 함수 내에 `setup, finish` 코드를 제외하고 8줄 이하로 작성할 경우 추가 점수 10점을 부여한다.

2.5. Sample screen shot



3. Hand in instructions

- 작성한 프로그램 코드 상단에 이름과 학번을 주석으로 넣는다.
- 과제 제출시 `simpleASM.s` 파일을 "학번.s" 파일로 이름을 변경하여 제출한다 (예: `2009310123.s`)
- 보고서는 워드, 한글 등의 형식도 상관 없지만 가능하면 PDF format으로 제출하고, 제출시 "학번.pdf" 형태로 파일을 저장하여 제출한다.
- 보고서에는 과제 수행을 위해 구축한 리눅스 환경의 스크린샷, 해당 환경에서 컴파일 하고 실행한 결과의 스크린샷, 프로그램의 설계, 구현에 관한 내용을 담는다.
- 과제 제출은 프로그램 소스 코드인 "학번.s"와 보고서인 "학번.pdf"를 메일로 첨부하여 sse2030@csl.skku.edu 로 보내고, 메일의 제목은 아래와 같은 형식을 따른다.

  [SSE2030] PA#2, 학번, 이름

4. Logistics

- 과제 제출 결과는 본 수업의 홈페이지인 http://csl.skku.edu/SSE2030F10 에서 확인할 수 있다.
- 과제 제출 시간은 메일 도착시간을 기준으로 하며, 기한을 넘겨 제출할 경우 하루에 25% 씩 감점한다.
- 과제에 대한 의논은 함께 할 수 있으나, 프로그램 소스코드 작성은 스스로 해야 한다.
- 다른 사람의 과제를 copy한 경우, 두 사람 모두 0점 처리한다. 인터넷 등에서 찾은 소스 코드를 그대로 copy한 경우에도 0점 처리한다. 두 번 이상 이와 같은 이유로 0점 처리된 경우 F 학점을 받을 수 있다.

Good luck!

담당조교: 김형준, 031-299-4970, #400629 (hjkim at csl.skku.edu)