상명대학교 컴퓨터과학과 "EA0003: 시스템소프트웨어" 과제

- 과제 번호: lab01

- 과제 디렉터리: ~/sys\_sw/lab01 이름: 임 현

- 과제 날짜: 2016년 4월 11일

- 과제 제목: NASM 사용 학번: 201511054

- 과제 내용:

1. (과제준비) 자신의 홈 디렉터리 아래에 과제 디렉터리 (~/sys\_sw/lab01)를 만든다. 이번 과제에서 만들어지는 모든 파일들은 이 디렉터리에 두도록 한다.
2. (nasm 명령) nasm 명령의 사용법에 대해 알아보자. “man nasm” 명령을 입력하면 nasm 사용법에 대한 매뉴얼을 볼 수 있다.
   1. nasm 명령줄의 형식을 적으시오.

nasm [ -@ response file ] [ -f format ] [ -o outfile ] [ -l listfile ] [ options... ] filename

nasm -h

nasm –v

* 1. nasm 명령 중 자주 사용하게 될 –f, –o, –l 옵션에 대해 조사하여 아래에 적으시오.

-f format : Specifies the output file format. To see a list of valid output formats, use the -hf option.

-o outfile : Specifies a precise name for the output file, overriding nasm's default means of determining it.

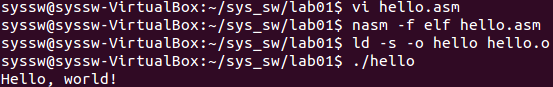
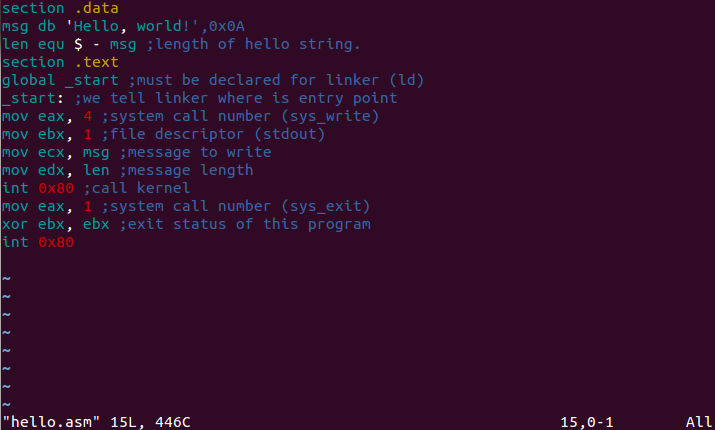
-l listfile : Causes an assembly listing to be directed to the given file, in which the original source is displayed on the right hand side (plus the source for included files and the expansions of multiline macros) and the generated code is shown in hex on the left.

* 1. “nasm –hf” 를 입력하면 nasm 이 지원하는 오브젝트 파일 형식을 알 수 있다. 이를 실행해 보고, 디폴트 오브젝트 파일 형식과 우리가 앞으로 사용해야 할 오브젝트 파일 형식이 무엇인지 적으시오.

디폴트 오브젝트 파일 형식 : bin flat-form binary files (e.g. DOS .COM, .SYS)

우리가 앞으로 사용 할 오브젝트 파일 형식 : elf ELF (e. g. Linux)

1. (nasm 실행) nasm 을 사용하여 간단한 어셈블리 프로그램을 실행시켜 보자.
   1. (간단한 어셈블리 프로그램의 실행) vi 에디터 등을 사용하여 강의 자료에 있는 hello.asm 파일을 작성하고, 이를 어셈블, 링크하여 실행 파일 hello 를 생성한 후, 제대로 실행되는지 확인하시오 (이 과정은 수업 시간에 다루었음). 이를 확인하기 위해 어셈블, 링크, 실행하는 과정을 수행한 터미널 창을 캡쳐하여 아래에 첨부하시오.



* 1. (어셈블리 리스트 파일) Nasm은 어셈블 과정에서 오브젝트 파일 외에 리스트 파일도 만들 수 있다. 리스트 파일을 만들면, 어셈블러가 생성하는 16진수 형태의 코드를 확인할 수 있다. 3.1의 컴파일 과정에 –l 옵션을 사용하여 hello.lst 라는 이름으로 리스트 파일을 만들어 보자 (이 파일의 내용은 수업 시간에 다루었음). 확인을 위해 이 과정을 수행한 터미널 창을 캡쳐하여 아래에 첨부하시오.



* 1. (오브젝트 파일과 심볼) 리눅스의 nm 명령을 사용하면 어셈블리 프로그램에서 정의된 심볼들에 대한 정보를 얻을 수 있다. 위 3.1 에서 생성된 오브젝트 파일 hello.o 에 대해 명령 “nm hello.o” 를 수행해 보자. 이 결과 출력된 내용을 아래에 적고 출력된 내용에 대해 설명하시오 (nm 명령의 출력에 대한 설명은 “man nm” 명령을 참조하시오).

00000000 T \_start

0000000e a len

00000000 d msg

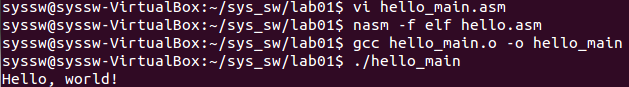
T = The symbol is in the text (code) section.

a = The symbol's value is absolute, and will not be changed by further linking.

d = The symbol is in the initialized data section.

* 1. (프로그램의 entry point) 위의 hello.asm 에 사용된 심볼 중 \_start 라는 심볼을 \_mystart 로 변경한 후, 이를 hello\_mystart.asm으로 저장하시오. 3.1과 동일한 방법으로 hello\_mystart.asm을 어셈블, 링크하여 실행 파일 hello\_mystart 를 생성한 후, (1) hello\_mystart 가 링크되는 과정에서 어떤 문제가 발생하는지 아래에 적으시오. (2) 왜 이와 같은 문제가 발생하는지 이유를 적고, 이 문제를 해결하기 위해서는 ld 명령을 사용할 때 어떤 옵션이 필요한지 적으시오 (힌트: “man ld” 명령을 통해 ld 에 대해 조사해 볼 것). (3) 확인을 위해 문제를 해결한 과정을 수행한 터미널 창을 캡쳐하여 아래에 첨부하시오.

1. ld: warning: cannot find entry symbol \_start; defaulting to 0000000008048080
2. 시작 상징인 \_start를 찾을 수 없기 때문이다. -e 명령어를 사용하여 \_mystart를 시작 상징으로 해주어야한다.
3. 
   1. (gcc를 이용한 링크) 위의 hello.asm 에 사용된 심볼 중 \_start 라는 심볼을 main 으로 변경한 후, 이를 hello\_main.asm으로 저장하시오. 위와 동일한 방법으로 hello\_main.asm을 어셈블 하되, 링크 시에는 ld 대신 gcc 를 사용하시오. 명령 “gcc hello\_main.o –o hello\_main” 과 같이 실행 파일을 만든 후 제대로 실행되는지 확인하는 과정을 수행한 터미널 창을 캡쳐하여 아래에 첨부하시오.



1. (nasm 의사명령/지시자) ECLASS 게시판의 nasm 어셈블러 매뉴얼(nasmdoc.pdf)을 읽고, 다음 내용들을 조사/요약하시오.
   1. nasm의 의사명령어들 (pseudo instructions) (매뉴얼 3.2절 참고)

DB, DW, DD, DQ and DT are used, much as in MASM, to declare initialised data in the output file. RESB, RESW, RESD, RESQ and REST are designed to be used in the BSS section of a module: they declare uninitialised storage space. INCBIN is borrowed from the old Amiga assembler DevPac: it includes a binary file verbatim into the output file. EQU defines a symbol to a given constant value: when EQU is used, the source line must contain a label. The TIMES prefix causes the instruction to be assembled multiple times.

* 1. nasm에서의 상수 (매뉴얼 3.4절 참고)]

A numeric constant is simply a number. A character constant consists of up to four characters enclosed in either single or double quotes. String constants are only acceptable to some pseudo−instructions, namely the DB family and INCBIN. Floating−point constants are acceptable only as arguments to DD, DQ and DT.

* 1. 어셈블러 지시자 section (매뉴얼 5.2절 참고)

The SECTION directive changes which section of the output file the code you write will be assembled into. In some object file formats, the number and names of sections are fixed; in others, the user may make up as many as they wish.

* 1. 어셈블러 지시자 global (매뉴얼 5.5절 참고)

GLOBAL is the other end of EXTERN: if one module declares a symbol as EXTERN and refers to it, then in order to prevent linker errors, some other module must actually define the symbol and declare it as GLOBAL. Some assemblers use the name PUBLIC for this purpose.

끝.