System Programming

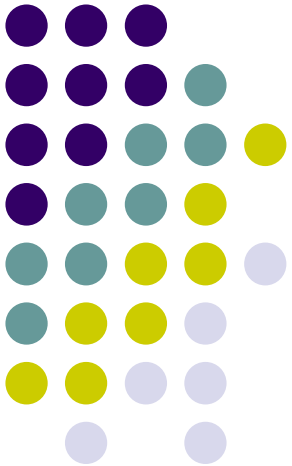
HW2. Compile and Assembly Program

2019. Fall

Instructor: Joonho Kwon

jhkwon@pusan.ac.kr

Data Science Lab @ PNU



datalab

data science laboratory

GNU Binutils

- The GNU binutils are a collection of binary tools
  - **ld** - the GNU linker
  - **as** - the GNU assembler
  - Other binary tools
    - ar - A utility for creating, modifying and extracting from archives
    - gprof - Displays profiling information
    - objcopy - Copys and translates object files
    - **objdump - Displays information from object files**
    - …
- Easy to port **binutils** to other platforms
- Suitable for embedded-system development

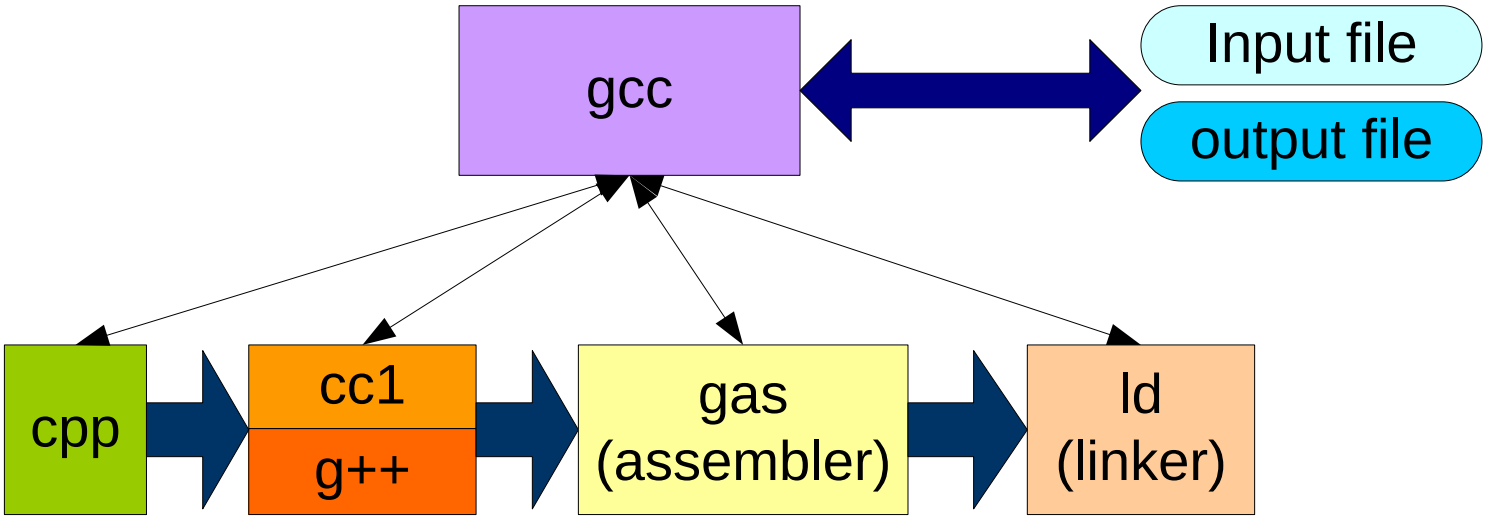
GNU C/C++ Compiler

- Retargettable
- Available from the Free Software Foundation
  - GPL Licensed
  - Free
- Written by *Richard Stallman* originally (FSF)
- Front-end, back-end system with support for many of each
- Code quality is better than some known compiler, very close t o DEC's compiler on Alpha
- Supplied as vendor compiler by NeXT, BeOS, Linux, BSD

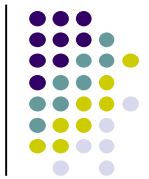
Front Ends

- C, C++,Objective C
- Ada 95 (GNAT)
- Fortran77, Fortran95
- Pascal
- Modula-2, Modula-3
- Java (supported from gcc 3.0)
  - Java->Bytecode, Bytecode->native code
  - Java->Native code
- Cobol
- Chill (Cygnus, a language in the Modula II family used in Europ ean telecommunications)

GCC Execution



Example: Compiling multiple source files (1/3)



- Main.c

```
#include "hello.h"

int main (void)
{
  hello ("world");
  return 0;
}
```

Example: Compiling multiple source files (2/3)

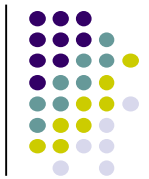
- hello.h

```
void hello (const char * name);
```

- hello_fn.c

```
#include <stdio.h>
#include "hello.h"

void hello (const char * name)
{
  printf ("Hello, %s!□n", name);
}
```

Example: Compiling multiple source files (3/3)



- Compile

```
$ gcc -Wall main.c hello_fn.c -o newhello
```

- Execute

```
$ ./newhello
Hello World!
```

How to Compile a C program

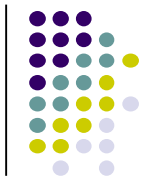
- Method1
  - Wide meaning of a compile is done

```
$ gcc hello.c
```

- Method2 ()
  - Narrow meaning of a compile: .c → .o
  - Linking: combining .o files as one executable file

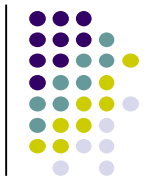
```
$ gcc -c main.c hello_fn.c
$ ls *.o                                  (generat main.o hello_fn.o )
$ gcc –o hello main.o hello_fn.o          (hello is an executable file)
$ ./hello
```

Assembler Example 1 : exit.s



```
#PURPOSE:  Simple program that exits and returns a
#          status code back to the Linux kernel

#INPUT:    none

#OUTPUT:   returns a status code. This can be viewed
#          by typing echo $? after running the program
#

#VARIABLES:%eax holds the system call number
#          %ebx holds the return status

.section .data

.section .text

.globl _start
_start:
    movl $1, %eax       # this is the linux kernel command number (system call) for

                        # exiting a program
    movl $0, %ebx       # this is the status number we will return to the operating
                        # system. Change this around and it will return different
                        # things to echo $?
    int $0x80           # this wakes up the kernel to run the exit command
```

Assembler Example 1: Assemble, link and execute



- Assemble
  - exit.s

```
$ as exit.s -o exit.o
```

- Linking

```
$ ld exit.o -o exit
```

- Executing

```
$ ./exit
$ echo $?
```

Disassemble Example1

- objdump command
  - a command-line program for displaying various information about object files on Unix-like OSs
  - For instance, it can be used as a disassembler to view an executable in assembly form

```
$ objdump -d ./exit

./exit:     file format elf64-x86-64

Disassembly of section .text:

0000000000400078 <_start>:
  400078:       b8 01 00 00 00          mov    $0x1,%eax
  40007d:       bb 00 00 00 00          mov    $0x0,%ebx
  400082:       cd 80                   int    $0x80
```

Assembler Example 2: maximum.s (1/2)

```
#PURPOSE:  This program finds the maximum number of a
#          set of data items.
#
#VARIABLES: The registers have the following uses:
#
# %edi - Holds the index of the data item being examined
# %ebx - Largest data item found
# %eax - Current data item
#
# The following memory locations are used:
#
# data_items - contains the item data.  A 0 is used
#              to terminate the data
#
.section .data

data_items:                         #These are the data items
    .long 3,67,34,222,45,75,54,34,44,33,22,11,66,0

.section .text

.globl _start
```

Assembler Example 2: maximum.s (1/2)

```
movl $0, %edi                  # move 0 into the index register
movl data_items(,%edi,4), %eax      # load the first byte of data
movl %eax, %ebx                # since this is the first item, %eax is
                               # the biggest

start_loop:                        # start loop
cmpl $0, %eax                  # check to see if we've hit the end
je loop_exit
incl %edi                          # load next value
movl data_items(,%edi,4), %eax
cmpl %ebx, %eax                # compare values
jle start_loop                 # jump to loop beginning if the new
                               # one isn't bigger
movl %eax, %ebx                # move the value as the largest
jmp start_loop                 # jump to loop beginning

loop_exit:
# %ebx is the status code for the exit system call
# and it already has the maximum number
    movl $1, %eax                  #1 is the exit() syscall
    int  $0x80
```

Assembler Example 2: Assemble, link and execute

- Assemble
  - maximum.s

```
$ as maximum.s -o maximum.o
```

- Linking

```
$ ld maximum.o -o maximum
```

- Executing

```
$ ./maximum
$ echo $?
222
```

Disassemble Example2

$ $ objdump -d ./maximum
./maximum: file format elf64-x86-64

Disassembly of section .text:

```
00000000004000b0 <_start>:
  4000b0:   bf 00 00 00 00          mov    $0x0,%edi
  4000b5:   67 8b 04 bd dd 00 60    mov    0x6000dd(,%edi,4),%eax
  4000bc:   00
  4000bd:   89 c3                   mov    %eax,%ebx

00000000004000bf <start_loop>:
  4000bf:   83 f8 00                cmp    $0x0,%eax
  4000c2:   74 12                   je     4000d6 <loop_exit>
  4000c4:   ff c7                   inc    %edi
  4000c6:   67 8b 04 bd dd 00 60    mov    0x6000dd(,%edi,4),%eax
  4000cd:   00
  4000ce:   39 d8                   cmp    %ebx,%eax
  4000d0:   7e ed                   jle    4000bf <start_loop>
  4000d2:   89 c3                   mov    %eax,%ebx
  4000d4:   eb e9                   jmp    4000bf <start_loop>

00000000004000d6 <loop_exit>:
  4000d6:   b8 01 00 00 00          mov    $0x1,%eax
  4000db:   cd 80                   int    $0x80
```

References

- Programming from the Ground Up
  - An introduction to programming using linux assembly language
    - Eng:
      - http://programminggroundup.blogspot.kr/
    - Kor:
      - http://www.joinc.co.kr/w/Site/Assembly/Documents/ProgrammingGroundUp/index.html

Q&A