CPSC-240 Computer Organization and Assembly Language

Chapter 14

Multiple Source Files

Instructor: Yitsen Ku, Ph.D.
Department of Computer Science,
California State University, Fullerton, USA

CALIFORNIA STATE UNIVERSITY
FULLERTON™

Outline

- Extern Statement
- Example, Sum and Average
  - Assembly Main
  - Function Source
  - Assemble and Link
- Interfacing with a High-Level Language
  - Example, C++ Main/Assembly Function
  - Compile, Assemble, and Link

Extern Statement

Extern Statement

- In order to inform the assembler that the function code or variables are in another file, the **extern** statement is used. The syntax is as follows:

  extern <symbolName>

- The symbol name would be the name of the function or a variable that is located in a different source file.
- In general, using global variables accessed across multiple files is considered poor programming practice and should be used sparingly (if at all). Data is typically passed between functions as arguments for the function call.

Example, Sum and Average

Example, Sum and Average

- The following is a simple example of a main that calls an assembly language function, *stats()*, to compute the integer sum and integer average for a list of signed integers.
- The main and the function are in different source files and are presented as an example of how multiple source files are used.

Assembly Main

Assembly Main

- The main is as follows:

```
; Simple example to call an external function.
; ----------------------------------------------------
; Data section
section .data
; -----  Define standard constants
LF                      equ     10              ; line feed
NULL                    equ     0               ; end of string
TRUE                    equ     1
FALSE                   equ     0
EXIT_SUCCESS            equ     0               ; success code
SYS_exit                equ     60              ; terminate
; ----- Declare the data
lst1                    dd      1, -2, 3, -4, 5
                        dd      7, 9, 11
len1                    dd      8
lst2                    dd      2, -3, 4, -5, 6
                        dd      -7, 10, 12, 14, 16
len2                    dd      10
```

Assembly Main

```
section .bss
sum1      resd      1
ave1      resd      1
sum2      resd      1
ave2      resd      1
; ------------------------------------------------------
extern stats
section .text
global _start
_start:
; ----
; Call the function
; HLL Call: stats(lst, len, &sum, &ave);
          mov       rdi, lst1                ; data set 1
          mov       esi, dword [len1]
          mov       rdx, sum1
          mov       rcx, ave1
          call      stats
```

Assembly Main

```
                mov     rdi, lst2               ; data set 2
                mov     esi, dword [len2]
                mov     rdx, sum2
                mov     rcx, ave2
                call    stats
; -----
; Example program done
exampleDone:
                mov     rax, SYS_exit
                mov     rdi, EXIT_SUCCESS
                syscall
```

Function Source

Function Source

```
; Simple example void function.
;*****************************************************************
; Data declarations
; Note, none needed for this example.
; If needed, they would be declared here as always.
section .data
;*****************************************************************
section .text
; ---------------------------------------------
; Function to find integer sum and integer average
; for a passed list of signed integers.
```

Function Source

```
; Call:
; stats(lst, len, &sum, &ave);
; Arguments Passed:
; 1) rdi - address of array
; 2) rsi - length of passed array
; 3) rdx - address of variable for sum
; 4) rcx - address of variable for average
; Returns:
; sum of integers (via reference)
; average of integers (via reference)
global stats
stats:
            push    r12
; -----
; Find and return sum.
            mov     r11, 0                  ; i=0
            mov     r12d, 0                 ; sum=0
```

Function Source

```
sumLoop:
            mov     eax, dword [rdi+r11*4]          ; get lst[i]
            add     r12d, eax ; update sum
            inc     r11                             ; i++
            cmp     r11, rsi
            jb      sumLoop
            mov     dword [rdx], r12d       ; return sum
; -----
; Find and return average.
            mov     eax, r12d
            cdq
            idiv    esi
            mov     dword [rcx], eax        ; return average
; -----
; Done, return to calling function.
            pop     r12
            ret
```

Assemble and Link

Assemble and Link

- Assuming the main source file is named *main.asm* and the functions source file is named *stats.asm*, the following command will perform the assemble and link.

```
yasm -g dwarf2 -f elf64 main.asm -l main.lst
yasm -g dwarf2 -f elf64 stats.asm -l stats.lst
ld -g -o main main.o stats.o
```

- The files names can be changed as desired.

Interfacing with a High-Level Language

Interfacing with a High-Level Language

- This section provides information on how a high-level language can call an assembly language function and how an assembly language function can call a high-level language function.
- This chapter presents examples for both.

Example, C++ Main / Assembly Function

Example, C++ Main / Assembly Function

- When calling any functions that are in a separate source file, the compiler must be informed that the function or functions source code are external to the current source file.
- This is performed with an **extern** statement in C or C++. Other languages will have a similar syntax.
- For a high-level language, the **extern** statement will include the function prototype which will allow the compiler to verify the function parameters and associated types.

C++ Main

C++ Main

```
#include <iostream>
using namespace std;
extern "C" void stats(int[], int, int *, int *);
int main()
{
	int lst[] = {1, -2, 3, -4, 5, 7, 9, 11};
	int len = 8;
	int sum, ave;
	stats(lst, len, &sum, &ave);
	cout << "Stats:" << endl;
	cout << " Sum = " << sum << endl;
	cout << " Ave = " << ave << endl;
	return 0;
}
```

Compile, Assemble, and Link

C++ Compile, Assemble, and Link

- Assuming that the C++ main is named *main.cpp*, and the assembly source file is named *stats.asm*, the commands to compile, assemble, link, and execute as follows:

```
g++ -g -Wall -c main.cpp
yasm -g dwarf2 -f elf64 stats.asm -l stats.lst
g++ -g -o main main.o stats.o
```

- Ubuntu 18 will require the **no-pie** option on the g++ command as shown:

```
g++ -g –no-pie -o main main.o stats.o
```

Execution

- The file names can be changed as desired. Upon execution, the output would be as follows:

```
./main
Stats:
    Sum = 30
    Ave = 3
```

C++ Main

C Main

```
#include<stdio.h>
extern void stats(int[], int, int *, int *);
int main()
{
	int lst[] = {1, -2, 3, -4, 5, 7, 9, 11};
	int len = 8;
	int sum, ave;
	stats(lst, len, &sum, &ave);
	printf ("Stats:\n");
	printf (" Sum = %d \n", sum);
	printf (" Ave = %d \n", ave);
	return 0;
}
```

C Compile, Assemble, and Link

- If a C main is used, and assuming that the C main is named *main.c*, and the assembly source file is named *stats.asm*, the commands to compile, assemble, link, and execute as follows:

```
gcc -g -Wall -c main.c
yasm -g dwarf2 -f elf64 stats.asm -l stats.lst
gcc -g -o main main.o stats.o
```

- Ubuntu 18 will require the **no-pie** option on the g++ command as shown:

```
gcc -g –no-pie -o main main.o stats.o
```

End of Chapter 14