Going From C to MIPS Assembly

Getting Started

Charles Gordon
(version 1.0, April 2000)

Contents

1 Downloading and Installing SPIM

We will be learning MIPS assembly using an emulator rather than real MIPS hardware. This allows you to write, compile and run programs on any standard Windows or Unix based computer. The following is a quick start guide to downloading, installing and using SPIM, the MIPS emulator.

1.1 Where to get SPIM

SPIM is a free MIPS emulator written by James Larus and available from his web page (http://www.cs.wisc.edu/~larus/spim.html). Executables are available for both Windows and DOS. The source code is also available for use with UNIX, including variants such as Linux. The program comes in two forms: command line and graphical. There is also a downloadable manual for the program which I suggest you read.

1.2 Installing SPIM

If you are using the Windows version of SPIM this is a very simple operation. The file downloaded from Larus's web site is an installation program that will create all the correct directories and shortcuts to run PC-SPIM. If you are using Linux or another UNIX variant the process will be slightly more involved. You will need to download and compile the source code for spim and xspim. As there are already compiled versions of these programs available on the computer science network, I don't recommend taking this approach. If you decide to take this approach and run into trouble, email your TA.

1.3 Running SPIM

Once again, if you are using the Windows version this is a simple operation. Simply find the "PCSPIM for Windows" option in the start menu and run PCSPIM. If you would rather use the X-Windows version of SPIM (which I find much easier to use) you will need to do a little more work. First you will need an X server for Windows such as Reflection-X (available on the lab machines in Sieg) or X-Win32. SPIM is available on the "tropical island" servers in the CSE department (fiji, ceylon, sumatra, tahiti). To run SPIM follow these steps:

a. Start up your X-Server (i.e. if you are in a CSE lab, just run Reflection X).
b. Start a telnet session to one of the tropical island servers by going to the Start Menu, choosing run and typing "telnet <server name>".
c. Log into the telnet session and set the display by typing "setenv DISPLAY <your ip address>:0.0". Obviously you need to fill in the blank with either the name of your computer or its IP address. One of these should be printed on the computer if you are in a lab. This will let SPIM know which X-Server to display itself on.
d. Now run xspim by typing the following at the command prompt, "/cse/courses/cse378/2000sp/lib/xspim". Wait a few seconds and you should see the SPIM window appear on your screen.

2 A First Program in SPIM

Writing assembly language programs is very similar to writing programs in a low level language like C. This section introduces you to the concepts you will need to know to write and compile a simple MIPS assembly program.

2.1 The Program Entry Point

Like C programs, MIPS assembly programs have an entry point called main. This is where the program will start executing when it is compiled and run. The syntax for declaring main is much different in MIPS however. A simple main function is shown in figure 2.1.

This main function does nothing except note the address of the program that loaded it and then return control to that program. However, there are a number of things that deserve pointing out about this simple program.

The first line of this program tells the compiler that this a text section. This means that everything after the ".text" is part of the program's executable code and needs to be compiled. In the next section I will introduce the ".data" section which is used for declaring global and static variables. For now just remember to put the ".text" line at the start of all your program code. After entering the code section of the file we need to declare the main function. This is done with a pair of lines, one of which tells the assembler that main is a "global" function, and the other which tells the assembler where main starts. Functions in MIPS assembly are declared just like labels in C or C++, as you can see by the declaration of main in figure 2.1.

```
     .text

# declare main as a global symbol
     .globl main
main:

# initialize the program stack and save
# the return address
     subu $sp, $sp, 4
     sw   $ra, 4($sp)

# put main function code here



# exit the program
     lw   $ra, 4($sp)
     addu $sp, $sp, 4

     j    $ra
```

Figure 2.1 – a simple main function

The rest of the code is doing some simple stack manipulation to save the return address of the loader and then return to it. This is mostly a formality, but it will be very important later when we start writing programs with more than one function. In between the initialization and exit of the function there is space to actually write a program. Note that I have left comments and labels unindented while I have indented everything else in the program. This is simply a stylistic convention of my own and it is not required by the assembler. You may feel free to indent however you find most readable and aesthetically pleasing.

2.2 The Global Data Section

The “.text” section of a file contains all of the executable program code, but it is still necessary to declare global data such as string and integer constants. This can be done in the “.data” section which allows you to create and name a number of different types of variables. An example of this section is shown in figure 2.2.

```
        .data

str_len:  .byte          12
str:      .asciiz   "Hello World!"
```

Figure 2.2 – a simple global data section

The section is, of course, prefixed by the ".data" command. This tells the assembler that all the lines following are a part of the global data section. Declaring variables here is identical to declaring global variables in a C program. These variables are available for the life of the program. To declare a variable, give it a name followed by a semicolon, such as str_len and str in fingure 2.2. Then you need to declare its type, which is always prefixed by a period (there will be a more comprehensive list of types given when we get to writing real programs). Following the variable type should be a value for the variable to start with.

2.3 A Simple Example Program

Figure 2.3 presents a small example program for your use to make sure that SPIM is running correctly. In the spirit of beginner texts everywhere, this program simply prints the words, "Hello World!" on the screen and then exits. You are encouraged to type this program into a file using your favorite text editor and run it using spim or xspim or PCSPIM.

To compile and run this program you will need to follow these steps:

a. start up your favorite text editor. If you are using Windows I recommend notepad or (my favorite) the Programmer's File Editor, which you can get on the internet. If you are using UNIX you will need to use vi or emacs.
b. Type in the program listing in figure 2.3 and save it as "hello.s".
c. Now start up SPIM (either the X version or the Windows version) and use the Open or Load command to load the file.
d. There will be a command to run the program (where it is depends on which version you are using, but it shouldn't be hard to find). Execute this command and a dialog should come up asking you where to being execution. The default is fine, so just hit OK.
e. Now a console window should pop up with the words "Hello World!" in it. On the Windows version this flashes by very quickly and you will need to go to the windows menu and select console to get it to show up.

If you have any trouble following these steps or some step fails to work please email your TA. You are probably experiencing a common problem that can be easily resolved.

```
#------------{ global data section } ------------#
# declare all global variables and string        #
# constants in this section.                 #
#------------------------------------------------#
    .data

hello_str:   .asciiz        "Hello World!"

#------------{ code section }--------------------#
# place all main code and procedure code in      #
# this section of the file            #
#------------------------------------------------#
    .text

# declare main as a global symbol
    .globl main
main:

    subu     $sp, $sp, 4 # create a word on the stack
    sw  $ra, 4($sp)    #store the return address

# put main function code here

    li  $v0, 4
    la  $a0, hello_str
    syscall

# exit the program
    lw  $ra, 4($sp)    #restore the return address
    addu     $sp, $sp, 4    #restore the stack

    j   $ra       #return
```

Figure 2.3 – SPIM does "Hello World!"