MIPS Programming Handout

MIPS (Microprocessor without Interlocked Pipeline Stages) is a RISC microprocessor architecture.

Spim is a simulator that can run assembly language programs written for the MIPS microprocessor. The CS Department lab machines (in 301MLH and B5) have Spim already installed. If you would like to install Spim on your PC, please refer to the section at the end of this handout.

Writing Assembly Programs Using Spim:

Program Structure:

- ✔ The programs are plain text files. Therefore to write assembly programs for Spim, you need a text editor that can create text (.txt in Windows) and not binary (Word) files.
  Tip: Save your program with a ".s" extension.
- ✔ Programs have data declaration section which is followed by the program code section.

Data declaration section:

- ✔ This is placed in the section of the program identified with the assembler directive **.data**. An assembler directive is just an indication of what the programmer wants. It does not result in any machine instructions when compiled. Assembler directives are preceded by a **dot(.)** as in **.data**
- ✔ **.data** directs the assembler that whatever follows is the data (variables) to be used in the program. These are stored in the data section of the program.

Code Section:

- ✔ The assembler directive **.text** identifies the code section.
- ✔ The code section contains the program code. This code translates into the machine instructions when compiled.
- ✔ The starting point of the code execution is identified by the label **main**:

How to include comments in your source code:

- ✔ Comments are written to the right of the **#** symbol.
  Eg:
  ```
  .data        # variable declarations follow this line
  .text        # instructions follow this line
  ```
- ✔ Comments always end a line

Note: The programs written for Spim must end with a blank line

Here is a simple program that reads a number and prints the square of the number. The numbers at the start of each line are for the sake of identifying the lines. The actual program should not have these numbers.

```
1.     #Program to read a number and print its square
2.     .data                                  #variable used follow this line
3.  prompt1:        .asciiz  "Enter a number:"
4.  prompt2:        .asciiz  "\n The square of the number you have entered is:"
5.                  .globl main
6.     .text                                  #program's code after this line
7.
8.  main:
9.     li           $v0,4                     #System call code for print string
10.    la           $a0,prompt1               #Load address of the prompt1 string
11.    syscall                                #call OS to Print prompt1
12.    li           $v0,5                     #System call code for read integer
13.    syscall                                #call OS to Read integer into $v0
14.    move         $t1,$v0                   #Move the integer into $t1
15.    mul          $t1,$t1,$t1               #Multiply the contents of $t1 with itself
16.    li           $v0,4                     #System call code for print string
17.    la           $a0,prompt2               #Load address of the prompt2 string
18.    syscall                                #call OS to Print prompt2
19.    move         $a0,$t1                   #Load $a0 with the value in $t1
20.    li           $v0,1                     #System code to print integer
21.    syscall                                #call OS to print the value in $v0
22. end:
23.    li           $v0,10                    #System call code to Exit
24.    syscall                                #call OS to Exit the program
25.
```

Anatomy of the above program: (the step number in the following indicates the line number in the above program)

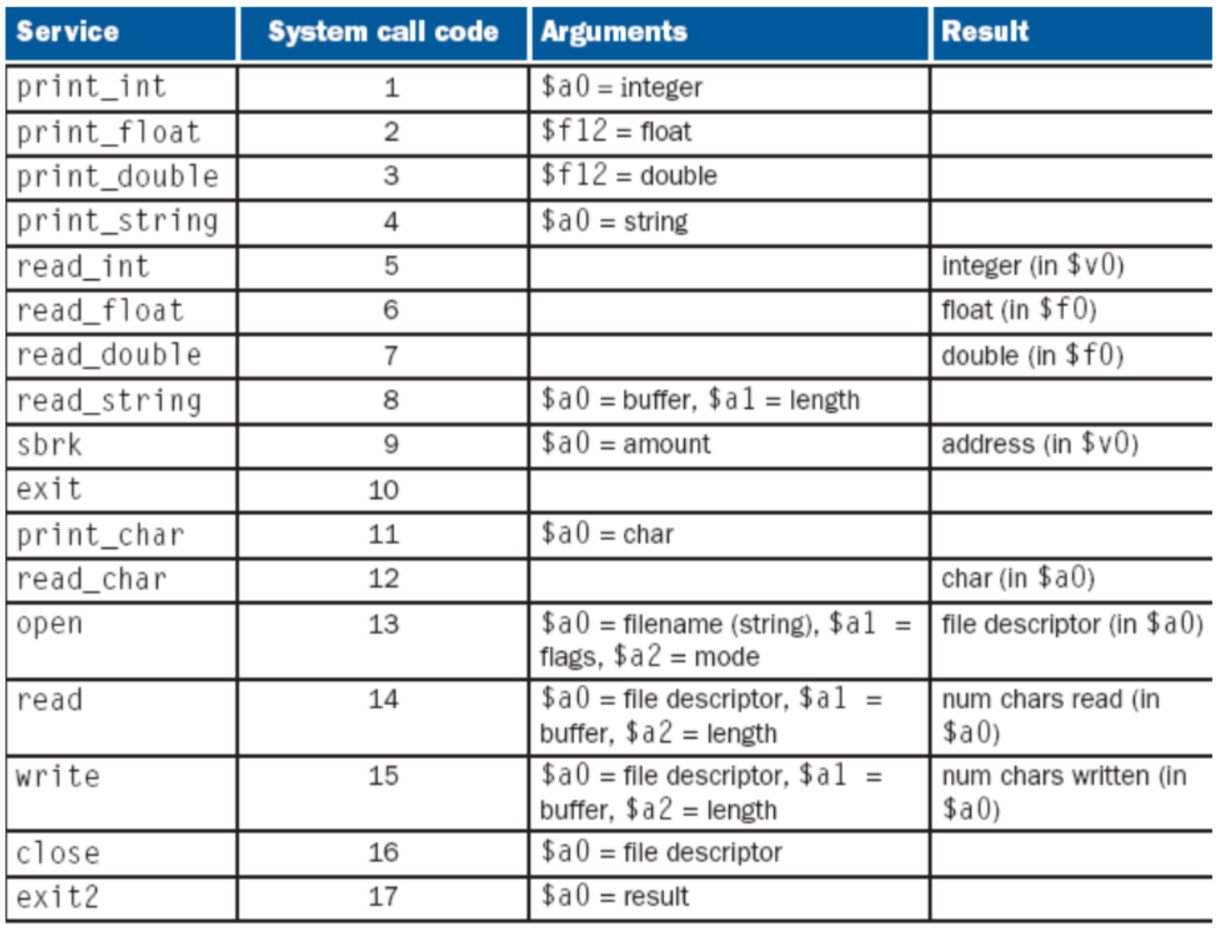
1. The first line contains a comment stating the purpose of the program.
2. Declares the **.data** directive indicating that the variables that are used in the program are declared after this line.
3. Declares **prompt1** using the asciiz directive. Strings declared using asciiz directive are NULL terminated and are of fixed length. In this case prompt1 contains "Enter a number:" plus the NULL character appended at the end.
4. Declares **prompt2** as an asciiz string which contains "\n The square of the number you have entered is:" plus the NULL character at the end.
   Note: The directive .ascii does not NULL terminate the string unlike .asciiz.
5. Declares the **.globl** directive. This indicates that the symbol **main** is accessible to other modules.
6. Declares the **.text** directive indicating that the program's code follows this line.
7. Is intentionally left blank.
8. **main:** is the label given to the first instruction that executes when the program is run.

The next 3 lines(9,10,11) demonstrate how use to perform I/O using **syscall**. The code for the correponding syscall operation must be loaded in **$v0.** These lines print the string in prompt1 on the screen.

The lines 12,13 cause an integer to be read into $v0.

14. Stores the value of the register $v0 in $t1.
15. Multiply the value of the number entered, which is in $t1, by itself.
16-18. Perform the system call to print the string prompt2.
19. Load the register $a0 with the result, which is in $t1.
20-21. Perform the system call to print the integer in $a0.
22. Labels the next instruction with the symbol **end:** Such labels are useful when we would like to branch to another instruction.
23-24. Perform the system call to stop the execution of the program.
25. Is a blank line at the end of the program.

The following table lists the various system call codes and the required arguments.



Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $a0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	

Please refer to Appendix A of the textbook for further information about the MIPS architecture.

Installing SPIM:

Microsoft Windows

1. Download the file http://www.cs.wisc.edu/~larus/SPIM/pcspim.zip and save it on your machine.
2. Unzip the file.
3. Click on the *setup.exe* program.

Unix, Linux, or Mac OS X

Installation is a bit more complex for a Unix or Linux system, as you need to compile the program for your particular computer and operating system.

1. Download either the file

   http://www.cs.wisc.edu/~larus/SPIM/spim.tar.Z **or** http://www.cs.wisc.edu/~larus/SPIM/spim.tar.gz.

2. Decompress the file, using either the program *uncompress* for the first file or *gzip* for the second file:
   %uncompress spim.tar.Z
   or
   % gzip -d spim.tar.gz

3. Move the file *spim.tar* to the directory in which you want to build spim and untar it:
   %tar xf spim.tar
   It will create a directory named *spim7.2* (or the most recent version number).

4. The simple terminal interface is contained in the *spim-7.2/spim* directory and the X-windows interfaces is in the *spim-7.2/xspim* directory. The other directories are described in the README file.

5. Next, you must set the directories in which spim will be installed by editing the Makefile (the file that contains instructions on building spim). In general, if you are installing spim and want the windowing version (*xspim*), edit the file *xspim/Imakefile*. If you don't want *xspim* or are running on a system without X-windows installed, you use the file *spim/Makefile*. Set these pathnames to the appropriate locations for your system:

   EXCEPTION_DIR -- The full pathname of the directory in which to install the **spimi** exception handler (*exceptions.s*). Use **/usr/local/lib**.

   BIN_DIR -- The full pathname of the directory in which *spim* and *xspim* should be installed. Use **/use/local/bin.**

   MAN_DIR -- The full pathname of the directory in which the manual pages for *spim* and *xspim* should be installed. Use **/usr/local/man.**
   In general, the remaining parameters in a Makefile need not be changed.

6. Then, if you are using *Imakefile* file, change to the *spim7.2/xspim* directory and type:

```
% xmkmf
% make
```

If you do not have a copy of *xmkmf*, you can use the Makefile in the *xspim* directory, but beware that it may not work on your system because the paths to the X windows libraries could be different.

7. If you do not have X-windows, change to the *spim-7.2/spim* directory, edit *Makefile,* and type:

   ```
   % make
   ```

8. To run *spim* or *xspim*, the exception handler must be installed in the directory specified by the variable EXCEPTION_DIR in the Makefile. If the file *exception.s* is not installed, *spim* and *xspim* fail before they start running. You can either install this file by hand or by typing

   ```
   % make install
   ```

   which also installs *spim* or *xspim*, and the manual pages in the directories that you set (above).

9. To test that spim is correctly built, change to the *spim-7.2/spim* directory and type:

   ```
   % make test
   ```

   and examine the output of the test. (Note: the exception handler must be installed before running the test.)

Sources:

1. http://www.cs.wisc.edu/~larus/spim.html
2. Computer Organization and Design , David A. Patterson and John L. Hennessy 3rd edition.
3. http://logos.cs.uic.edu/366/notes/MIPS%20Quick%20Tutorial.htm