Project 9 - File Input, Simple Statistics, and Memory Usage in MIPS

Goals

In this project you will work with programming MIPS to read a file containing N floating point numbers and compute the mean and standard deviation of the values.

Reading Files in MIPS

As in other languages, reading a file involves interaction with the operating system. In MIPS, this is done through syscalls, similar to user I/O. To read data from a file, it first needs to be opened. Opening a file returns a file descriptor, essentially an index into an array of file information maintained by the operating system. Once the file is opened, one can read information. The bit patterns stored in the file are typically read sequentially and stored in memory. Because there is a difference between the ASCII representation of a number and its two's complement representation, one must be careful on how a file is read and interpreted. For example, the number 16383 requires 5 bytes to be represented in ASCII (31 36 33 38 33 [base 16]) and 4 bytes in a 32-bit two's complement representation (00 00 3f ff [base 16]). Once all information is read from the file, one needs to close the file to free the file descriptor.

The program file_io.s shown below illustrates opening a file, reading 4 bytes from the file, printing those bytes to the screen interpreted as a 4 byte two's complement integer, and closing the file. Download and run this program, stepping through the program and watch the memory and the register values change. Since the file /home/sleightlab/crap contains the bytes 63 72 61 70 0a [base 16], the program should produce a value of 1885434467 [base 10] = 63726170 [base 16] when run.

```
# Read and echo file on a character by character basis
# David A. Reimann
# April 2008

        .text
main:

# Open File

        li      $v0, 13                 # 13=open file
        la      $a0, file               # $a2 = name of file to read
        add     $a1, $0, $0             # $a1=flags=O_RDONLY=0
        add     $a2, $0, $0             # $a2=mode=0
        syscall                         # Open FIle, $v0<-fd
        add     $s0, $v0, $0    # store fd in $s0


# Read 4 bytes from file, storing in buffer

        li      $v0, 14                 # 14=read from  file
        add     $a0, $s0, $0    # $s0 contains fd
        la      $a1, buffer             # buffer to hold int
        li      $a2, 4                  # Read 4 bytes
        syscall

        li      $v0, 1                  # 1=print int
        lw      $a0, buffer             # buffer contains the int
        syscall                         # print int


# Close File

done:
        li      $v0, 16                 # 16=close file
        add     $a0, $s0, $0    # $s0 contains fd

        syscall                         # close file


# Exit Gracefully

        li      $v0, 10
        li      $v0, 10
        syscall



        .data

file:
        .asciiz "/home/sleightlab/crap" # File name
        .word   0
buffer:
        .space  4                       # Place to store character
```

Here are some important things to note about the above program. Syscall 13 opens a file, $a0 contains the address of a null terminated string corresponding to the file name, $a1 contains the read flags (use 0 for reading), and $a2 contains the mode (again use 0 for reading). The file descriptor is returned in $a0 and must be stored for future file operations. Syscall 14 reads N bytes from the file corresponding to the file descriptor in $a0 into the buffer specified by $a1. The number of bytes to read N is specified in $a2 and the number of bytes actually read from the file is returned in $a0 (normally this can be used to verify correct reading, but MARS will just crash your program if there is a problem). Syscall 16 closes the file corresponding to the file descriptor in $a0.

Memory Usage

The MIPS memory model follows the typical conventions found on most modern computers. There are several basic memory segments: the stack, the heap (or dynamic data), the static data, the text segment (also static), the reserved area (for operating system issues), and the free memory. As a program runs and needs additional memory, it is added to the stack or heap segments from the free area. See Figure B.5.1 and the discussion of memory usage in section B.5 of your text for more details.

Since most nontrivial programs need to have flexibility in the amount of memory used based on user input, it is essential to have a memory model that can adjust to the program demands. The stack segment is mostly used for function calls. The heap is used for additional memory blocks. While the heap is generally managed by the run-time environment, one can often just live dangerously and simply assume anything above a the last address in the .data segment is available for you exclusive use. So note that in the above program buffer was allocated to be 4 bytes, you can assume it is infinitely large (for small values of infinity). While this is extremely bad practice in general, you should be able to get away with it here.

Simple Statistics

One often uses simple statistics to describe large data sets. The most commonly used statistics are the mean (μ) and standard deviation (σ). Given a set of N values $\{x_1, x_2, . . ., x_N\}$, the mean is defined by

$$\mu = (x_1 + x_2 + . . . + x_N)/N$$

and the standard deviation is defined by

$$\sigma = \mathrm{sqrt}(((x_1 - \mu)^2 + (x_2 - \mu)^2 + . . . + (x_N - \mu)^2)/N).$$

Intuitively, the mean gives a midpoint of the data set and the standard deviation describes how far away the data is from the mean. Note this requires two passes through the data, once for the mean and the second for the standard deviation.

Task

Write a MIPS program that reads a file that contains the following format

N	x_1	x_2	. . .	x_N

where each element of the file is a 4 byte value. N will be a 32 bit two's complement value. Each x_i value will be a 32-bit single precision floating point value. Prompt the user for the file name, read the file (assuming the above format), compute and print the mean and standard deviation of the values in the file. You should have only two reads in your file: the first gets N, the sconds reads the data. You need to do passes (two independent loops) over your data: once to compute the average, the second to compute the standard deviation.

Some example files are given below:

file	N	mean	standard deviation
stats_10.dat	10	4.953309	0.847057
stats_100.dat	100	0.508110	0.292814
stats_1000.dat	1000	996.666232	144.084131

You can verify the contents of these files using the linux command od. For example

od -t d4 *file*

will print the contents of the file as 4 byte decimal integer values. You can use

```
od -t f4 file
```

will print the contents of the file as 4 byte single precision floating point values.

This program will require you to use some additional floating point instructions, in particular sqrt.s will compute a single precision floating point square root. Note that many operations involving moving data (lw, sw) do not distinguish between integer, character, or floating point values. See Appendix B for details on how to move floating point values from the register file to the floating point coprocessor using the mfc1 and mtc1 instructions.

Deliverables

Send me an email with the MIPS file containing your program. Comment each line and code segment explaining its function.

References

1. Appendix B
