Introduction to Lab #1

Karim Ali

General Intro to 229 Labs

- In 229, a “lab” is a programming assignment:
  - A lab requires many more hours of work than the time allocated for lab sessions.
  - Lab sessions are “consulting hours” when TAs are available to answer questions and to help.
  - Reading/work prior to the lab date/time is essential.
  - The lab assignments will be progressively more difficult, and will require more time as the term advances.
- A CMPUT 229 lab is not a “lab” in the sense of a chemistry lab.

Part #1

- Read Appendix titled "Assemblers, Linkers, and the SPIM Simulator" (specially Section 9):
  - In the 4th edition of the book, this is Appendix B
  - In the 5th edition of the book, this is Appendix A
- Go through the SPIM tutorial in the Tutorials section of the CD that comes with the book.

Part #2

- Self-guided tutorial-style introduction to usage of XSPIM.

Part #3

- Simple exercise to illustrate use of SPIM.

Part #4

- Understand data storage in memory.
- Question #7: Understand little/big endianness and conversion to/from ASCII.
- Question #10: Understand 2's complement
- Question #11: Assembly directives

Part #5: Find bugs in `lab1-broken.s`

- The program `lab1-broken.s` was written to replace characters in a string.
- It should convert
  “Cmput 229 is the absolute bomb.” into
  “Cmput-229-is-the-absolute-bomb.”
- But it is not working as it should.
- Your job is to read and understand the program and report the errors in it.

Part #5 Submission

- You will describe the bugs in a text file called `bugs.txt` and submit this file.
- The solution for Parts 1-5 are answers to the questions in the lab assignment.
  - There is no specified format for these answers. Just use a reasonable formatting and provide clear and concise answers.

System call table

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	

Patterson and Hennessy pp. A-44

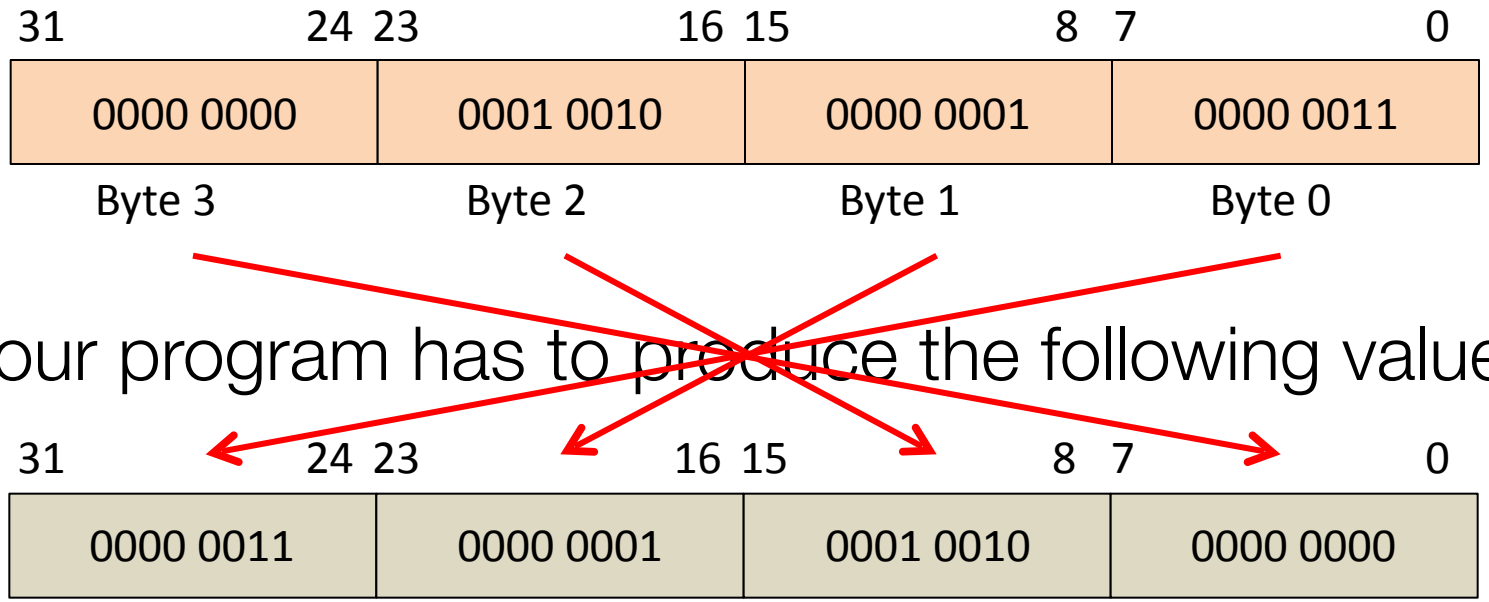
Part #6: Write a Simple Program

Write a MIPS assembly language program to:

- read an integer from the terminal
- invert the byte order of the integer
- print out the new value of the integer

Example #2

Integer: 1179907



Your program has to produce the following value:

Formatting and Style

- Check the grading lab mark-sheet

Assembler Syntax

comments begin with a sharp sign (#) and run to the end of the line.

identifiers are alphanumeric sequences, underbars (_), and dots (.) that do not begin with a number.

labels are identifiers placed at the beginning of a line, and followed by a colon.

```
            .data
item:       .word 1
            .text
            .globl      main
main: lw    $s3, item
Loop: add   $t1, $s3, $s3      #  $t1 ← 2 * i
      add   $t1, $t1, $t1      #  $t1 ← 4 * i
      add   $t1, $t1, $s6      #  $t1 ← Addr(save[i])
      lw    $t0, 0($t1)        #  $t0 ← MEM[save[i]]
      bne   $t0, $s5, Exit     #  if save[I] ≠ k goto Exit
      add   $s3, $s3, $s4      #  i ← i + j
      j     Loop               # goto Loop
Exit:
```


Patterson and Hennessy pp. B-10

Assembler Directives

.data identifies the beginning of the data segment
(in this example this segment contains a single word).

.word 1 stores the decimal number 1 in 32-bits (4 bytes)

.text identifies the beginning of the text segment
(where the instructions of the program are stored).

.globl main declares the label main global
(so that it can be accessed from other files).

```
        .data
item:   .word 1
        .text
        .globl  main
main: lw   $s3, item
Loop: add  $t1, $s3, $s3      #  $t1 ← 2 * i
      add  $t1, $t1, $t1      #  $t1 ← 4 * i
      add  $t1, $t1, $s6      #  $t1 ← Addr(save[i])
      lw   $t0, 0($t1)        #  $t0 ← MEM[save[i]]
      bne  $t0, $s5, Exit     #  if save[I] ≠ k goto Exit
      add  $s3, $s3, $s4      #  i ← i + j
      j    Loop               # goto Loop
Exit:
```

File `lab1-p1.s`

pseudo instruction that loads the immediate value in the register

```
# What's going on here ?
        .text
main:
        li $a1, 5
        la $t0, val
        xor $t1, $t1, $t1
        xor $t2, $t2, $t2

loop:   sub $t3, $a1, $t2
        blez $t3, exit
        lw $t4, 0($t0)
        add $t1, $t1, $t4
        add $t2, $t2, 1
        addu $t0, $t0, 4
        j loop
```

pseudo instruction that loads the address of specified label into register

```
exit:   div $t5, $t1, $a1
        li $v0, 4
        la $a0, outputMsg
        syscall
        li $v0, 1
        add $a0, $0, $t5
        syscall
        li $v0, 4
        la $a0, newln
        syscall

        jr $ra

        .data

val:    .word 12, 34, 56, 78, 90

outputMsg:
        .asciiz "\n Result = "
newln:
        .asciiz "\n\n"
```

OS-style call to obtain services from SPIM:
$a0-$a3: arguments
$v0: system call code
before the call;
return value
after the call.
(see Patterson and Hennessy pp. A-43).

Files to Submit

- There are three files to submit:
  - `lab1.txt`
  - `lab1.s`
  - `bugs.txt`