Laboratory 1 Notes

Introduction to MIPS/MARS

- **MIPS Architecture** (Microprocessor without Interlocked Pipeline Stages)
  - **32-bit word = 4 bytes**
  - **32 registers**
  - **Basic Instructions**
  - **Assembly Directives**

- **System Calls for Input/Output**

- **Representation of numbers**
  - **Hexadecimal (base 16) notation**
  - **Little-Endian byte order**

- **Operation of MARS** (MIPS Development And Runtime Simulator)

MIPS Archictecture (32 registers)

$0 or $zero	- always contains 0.
$t0 - $t9	- use for temporary storage of data
$s0 - $s7	- use to hold address locations in memory
$a0 - $a3	- use as arguments to system calls
$v0 and $v1	- use as arguments to system calls

Basic MIPS instructions for lab today

Arithmetic (R-type) Instructions

```
add  $t3,$t1,$t2         #$t3 <- contents of $t1 + contents of $t2

addi  $t3, $t1,5         #$t3 <- contents of $t1 + 5
```

Memory Access Instructions

```
lw    $t1,label          # $t1 <- value of word stored at memory
                                 address/location specified by label

lw    $t1,3($s0)        # $t1 <= value of word stored at memory
                                address specified by base address
                               in $s0 + 3

sw    $t1,label          # store value of word in $t1 to address/
                              location in memory  specified by label
```

(**lw** and **sw** can also be byte or halfword, i.e. **lb**, **lh, sb,sh**)

Pseudo-instructions

```
li      $t1, 3              # $t1 <- 3

la     $s1, label          # $s1 <- address corresponding to label

move,$t1,$t2             #move contents of $t2 to $t1
```

Directives - tell assembler how to translate program, but are not instructions.

.text
.globl main

Precedes your **text segment** (program instructions), and specifies **main** as a global symbol (recognized by other files in a multi-file project

.data

Precedes your **data segment** (data declarations)

(text segment can come before data segment, or vice versa)

.ascii “string”

Defines a string of characters (each character is stored as a 1-byte ascii value)

.asciiz “string”

Defines a null-terminated string (ends with a null byte)

.byte b0,b1,b2

Defines and initializes subsequent bytes in memory

.half h0,h1,h2

Defines and initializes subsequent half-words (16-bit values – alignment forced to next even address

.word w0,w1,w2

Defines and initializes subsequent words (32-bit values) – alignment forced to next word address (multiple of 4)

.space n

allocates n bytes of space, usually initialized to 0

SYSCALL functions overview

System services used for input/output

How to use SYSCALL system services

1. Load the service number in register $v0.
2. Load argument values, if any, in $a0, $a1, or $a2
3. Issue the SYSCALL instruction.
4. Retrieve return values, if any, from result registers

Table of Commonly Used Services

Service	$v0	Arguments	Result
print integer	1	$a0 = integer to print	
print string	4	$a0 = address of null-terminated string to print	
read integer	5		$v0 contains integer read
read string	8	$a0=address of input buffer $a1=max. # of chars. to read	
exit (stop execution)	10		
print character	11	$a0=character to print	
read character	12		$v0 contains character read
open file	13	$a0=address of null-terminated string containing filename $a1=flags $a2=mode	$a0 contains file descriptor (- if error)
read from file	14	$a0 = file descriptor $a1=address of output buffer $a2=max. # of chars to read	$a0 contains # of chars. read (0=EOF,- if error)
write to file	15	$a0 = file descriptor $a1=address of output buffer $a2= # of chars to write	$a0 contains # chars. written (- if error)
close file	16	$a0 = file descriptor	

Examples of Simple I/O for lab today

```
    # print an integer
        li $v0,1        # load service number into $v0
        li $a0,5        # load value to be printed into $a0
        syscall

    #print a null-terminated string
        li $v0,4        #load service number in $v0
        la $a0,prompt_string
                        # load address of string to be printed into $a0
        syscall

        # the null-terminated string must be defined in data segment
        .data
      prompt_string:  .asciiz “Enter a value: “

    # read in an integer
        li $v0,5        #load service number in $v0
        syscall         #the value entered by the user is returned in $v0

        move $t0,$v0        #store value entered into another register

    # read in a string
        li $v0,8                #load service number in $v0
        la $a0,answer           #put address of answer string in $a0
        lw $a1,alength          #put length of string in $a1
        syscall          # string is stored in memory at the answer  location

        #answer and alength must be defined in data segment
            .data
        answer:  .space 50  # allocate space for string to be stored
        alength:  .word 50   #length of string to be entered

    # terminate execution of program
        li $v0,10       #load service number in $v0
        syscall         #should always be the final instructions in your program
```