SPIM Instruction Set

This document gives an overview of the more common instructions used in the SPIM simulator.

Overview

The SPIM simulator implements the full MIPS instruction set, as well as a large number of *pseudoinstructions* that correspond to one or more equivalent MIPS instructions. There are also a small number of system call commands used to interface with the console window of the SPIM simulator. Finally, SPIM renames registers according to commonly used conventions in order to facilitate the readability of programs.

Instructions and PseudoInstructions

The following is an abbreviated list of MIPS instructions and SPIM pseudoinstructions. This list is not complete. Notably missing are all Floating Point and coprocessor instructions.

• - Indicates an actual MIPS instruction. Others are SPIM pseudoinstructions.

Instruction		Function
• add	Rd, Rs, Rt	Rd = Rs + Rt (signed)
• addu	Rd, Rs, Rt	Rd = Rs + Rt (unsigned)
• addi	Rd, Rs, Imm	Rd = Rs + Imm (signed)
• sub	Rd, Rs, Rt	Rd = Rs - Rt (signed)
• subu	Rd, Rs, Rt	Rd = Rs - Rt (unsigned)
• div	Rs, Rt	lo = Rs/Rt, hi = Rs **mod** Rt (integer division, signed)
• divu	Rs, Rt	lo = Rs/Rt, hi = Rs **mod** Rt (integer division, unsigned)
div	Rd, Rs, Rt	Rd = Rs/Rt (integer division, signed)
divu	Rd, Rs, Rt	Rd = Rs/Rt (integer division, unsigned)
rem	Rd, Rs, Rt	Rd = Rs **mod** Rt (signed)
remu	Rd, Rs, Rt	Rd = Rs **mod** Rt (unsigned)
mul	Rd, Rs, Rt	Rd = Rs * Rt (signed)
• mult	Rs, Rt	hi, lo = Rs * Rt (signed, hi = high 32 bits, lo = low 32 bits)
• multu	Rd, Rs	hi, lo = Rs * Rt (unsigned, hi = high 32 bits, lo = low 32 bits)
• and	Rd, Rs, Rt	Rd = Rs • Rt
• andi	Rd, Rs, Imm	Rd = Rs • Imm
neg	Rd, Rs	Rd = -(Rs)
• nor	Rd, Rs, Rt	Rd = (Rs + Rt)’
not	Rd, Rs	Rd = (Rs)’
• or	Rd, Rs, Rt	Rd = Rs + Rt
• ori	Rd, Rs, Imm	Rd = Rs + Imm
• xor	Rd, Rs, Rt	Rd = Rs ⊕ Rt
• xori	Rd, Rs, Imm	Rd = Rs ⊕ Imm

•	sll	Rd, Rt, Sa	Rd = Rt left shifted by Sa bits
•	sllv	Rd, Rs, Rt	Rd = Rt left shifted by Rs bits
•	srl	Rd, Rs, Sa	Rd = Rt right shifted by Sa bits
•	srlv	Rd, Rs, Rt	Rd = Rt right shifted by Rs bits

	move	Rd, Rs	Rd = Rs
•	mfhi	Rd	Rd = hi
•	mflo	Rd	Rd = lo
	li	Rd, Imm	Rd = Imm
•	lui	Rt, Imm	Rt[31:16] = Imm, Rt[15:0] = 0

•	lb	Rt, Address(Rs)	Rt = byte at M[Address + Rs] (sign extended)
•	sb	Rt, Address(Rs)	Byte at M[Address + Rs] = Rt (sign extended)
•	lw	Rt, Address(Rs)	Rt = word at M[Address + Rs]
•	sw	Rt, Address(Rs)	Word at M[Address + Rs] = Rt

•	slt	Rd, Rs, Rt	Rd = 1 if Rs < Rt, Rd = 0 if Rs ≥ Rt (signed)
•	slti	Rd, Rs, Imm	Rd = 1 if Rs < Imm, Rd = 0 if Rs ≥ Imm (signed)
•	sltu	Rd, Rs, Rt	Rd = 1 if Rs < Rt, Rd = 0 if Rs ≥ Rt (unsigned)

•	beq	Rs, Rt, Label	Branch to Label if Rs == Rt
	beqz	Rs, Label	Branch to Label if Rs == 0
	bge	Rs, Rt, Label	Branch to Label if Rs ≥ Rt (signed)
•	bgez	Rs, Label	Branch to Label if Rs ≥ 0 (signed)
•	bgezal	Rs, Label	Branch to Label and Link if Rs ≥ Rt (signed)
	bgt	Rs, Rt, Label	Branch to Label if Rs > Rt (signed)
	bgtu	Rs, Rt, Label	Branch to Label if Rs > Rt (unsigned)
•	bgtz	Rs, Label	Branch to Label if Rs > 0 (signed)
	ble	Rs, Rt, Label	Branch to Label if Rs ≤ Rt (signed)
	bleu	Rs, Rt, Label	Branch to Label if Rs ≤ Rt (unsigned)
•	blez	Rs, Label	Branch to Label if Rs ≤ 0 (signed)
•	bgezal	Rs, Label	Branch to Label and Link if Rs ≥ 0 (signed)
•	bltzal	Rs, Label	Branch to Label and Link if Rs < 0 (signed)
	blt	Rs, Rt, Label	Branch to Label if Rs < Rt (signed)
	bltu	Rs, Rt, Label	Branch to Label if Rs < Rt (unsigned)
•	bltz	Rs, Label	Branch to Label if Rs < 0 (signed)
•	bne	Rs, Rt, Label	Branch to Label if Rs ≠ Rt
	bnez	Rs, Label	Branch to Label if Rs ≠ 0

•	j	Label	Jump to Label unconditionally
•	jal	Label	Jump to Label and link unconditionally
•	jr	Rs	Jump to location in Rs unconditionally
•	jalr	Label	Jump to location in Rs and link unconditionally

Registers

By convention, many MIPS registers have special purpose uses. To help clarify this, SPIM defines aliases for each register that represent its purpose. The following table lists these aliases and the commonly accepted uses for the registers.

Register	Number	Usage
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Used for return values from function calls.
v1	3	
a0	4	Used to pass arguments to procedures and functions.
a1	5	
a2	6	
a3	7	
t0	8	Temporary (Caller-saved, need not be saved by called procedure)
t1	9	
t2	10	
t3	11	
t4	12	
t5	13	
t6	14	
t7	15	
s0	16	Saved temporary (Callee-saved, called procedure must save and restore)
s1	17	
s2	18	
s3	19	
s4	20	
s5	21	
s6	22	
s7	23	
t8	24	Temporary (Caller-saved, need not be saved by called procedure)
t9	25	
k0	26	Reserved for OS kernel
k1	27	
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address for function calls.

System Calls

In order to perform I/O with the console, SPIM provides a small library of system calls. In general, system calls are set up by placing a system call in register $v0, and any arguments in register $a0 and $a1. Returned values are placed in register $v0.   See the table and the example program below for usage.

Example Program

```
# This program takes input from the user and echoes it back
```

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		

```
         .data
# Constant strings to be output to the terminal
promptInt:          .asciiz "Please input an integer: "
resultInt:          .asciiz "Next integer is: "
linefeed:.asciiz    "\n"
enterkey:.asciiz "Press any key to end program."

    .text
main:
# prompt for an integer
         li         $v0,4                # code for print_string
         la         $a0,promptInt        # point $a0 to prompt string
         syscall                         # print the prompt

# get an integer from the user
         li         $v0,5                # code for read_int
         syscall                         #get int from user --> returned in $v0
         move       $t0,$v0              # move the resulting int to $t0

# compute the next integer
         addi       $t0, $t0, 1          # t0 <-- t0 + 1

# print out text for the result
         li         $v0,4                #code for print_string
         la         $a0,resultInt        # point $a0 to result string
         syscall                         # print the result string

# print out the result
         li         $v0,1                # code for print_int
         move       $a0,$t0              # put result in $a0
         syscall                         # print out the result

# print out a line feed
         li         $v0,4                # code for print_string
         la         $a0,linefeed         # point $a0 to linefeed string
         syscall                         # print linefeed
```

```
# wait for the enter key to be pressed to end program
        li          $v0,4                   # code for print_string
        la          $a0,enterkey            # point $a0 to enterkey string
        syscall                             # print enterkey

# wait for input by getting an integer from the user (integer is ignored)
        li          $v0,5                   # code for read_int
        syscall                             #get int from user --> returned in $v0

# All done, thank you!
        li          $v0,10                  # code for exit
        syscall                             # exit program
```