MIPS reference sheet

General-purpose registers

Name	Number	Purpose
$zero	$0	constant zero
$at	$1	reserved for assembler
$v0-$v1	$2-$3	return values, system call code
$a0-$a3	$4-$7	function and system call arguments
$t0-$t7, $t8-$t9	$8-$15,$24-$25	temporary storage (caller-saved)
$s0-$s7	$16-$23	temporary storage (callee-saved)
$k0-$k1	$26-$27	reserved for kernel
$gp	$28	pointer to global area
$sp	$29	stack pointer
$fp or $s8	$30	frame pointer
$ra	$31	return address

SPIM System calls

Service	Call code ($v0)	Arguments	After return	Notes
Print integer	1	$a0 = value to print		value is signed
Print string	4	$a0 = address of string to print		string is terminated with '\0'
Read integer	5		$v0 = entered integer	value is signed
Read string	8	$a0 = address to store string at $a1 = maximum number of characters		returns if $a1−1 characters or newline typed; string is terminated with '\0'
Exit	10			ends simulation
Print char	11	$a0 = character to print		upper 24 bits ignored
Read char	12		$v0 = entered character	sign- or zero-extended, depending on underlying system

Function calling convention

Function call

Caller:	**Callee:**
- saves temporary registers on stack - passes arguments on stack - calls function with `jal` instruction	- saves value of `$ra` on stack - saves value of `$fp` on stack - copies `$sp` to `$fp` - allocates local variables on stack

Function return

Caller:	**Callee:**
- clears arguments off stack - restores temporary registers off stack - uses return value in $v0	- sets $v0 to return value - clears local variables off stack - restores saved `$fp` off stack - restores saved `$ra` off stack - returns with `jr` instruction

Assembler directives

`.data`	assemble into data segment
`.text`	assemble into text segment
`.byte b1[, b2, ...]`	allocate byte(s), with initial value(s)
`.half h1[, h2, ...]`	allocate halfword(s), with initial value(s)
`.word w1[, w2, ...]`	allocate word(s), with initial value(s)
`.space n`	allocate n bytes of uninitialized space
`.ascii "string"`	allocate ASCII string, do not terminate
`.asciiz "string"`	allocate ASCII string, terminate with `'\0'`

Instruction Set

A partial MIPS instruction set is on the following pages. The following conventions apply:

Instruction format:
`Rsrc`, `Rsrc1`, `Rsrc2`: source (must be register)
`Src2`: source (register or immediate)
`Rdest`: destination (must be register)
`Imm`: Immediate value
`Imm16`: Immediate value, 16 bits
`Addr`: Address in form `const(Rsrc)`
`label`: label of instruction
★: pseudoinstruction

Immediate form:
—: no immediate form, or this *is* the immediate form
★: immediate form synthesized as pseudoinstruction

Unsigned form (append u to opcode):
—: no unsigned form, or this *is* the unsigned form

Instruction format	Meaning	Operation	Immediate form	Unsigned form
add Rdest, Rsrc1, Src2	Add	Rdest = Rsrc1 + Src2	addi	no overflow
sub Rdest, Rsrc1, Src2	Subtract	Rdest = Rsrc1 - Src2	★	no overflow
mul Rdest, Rsrc1, Src2 ★	Multiply	Rdest = Rsrc1 * Src2	★	unsigned operands
mulo Rdest, Rsrc1, Src2 ★	Multiply (with overflow)	Rdest = Rsrc1 * Src2	★	unsigned operands
mult Rsrc1, Rsrc2	Multiply (machine instruction)	HI:LO = Rsrc1 * Src2	—	unsigned operands
div Rdest, Rsrc1, Src2 ★	Divide	Rdest = Rsrc1 / Src2	★	unsigned operands
rem Rdest, Rsrc1, Src2 ★	Remainder	Rdest = Rsrc1 % Src2	★	unsigned operands

div Rsrc1, Rsrc2	Divide (machine instruction)	LO = Rsrc1 / Rsrc2;
		HI = Rsrc1 % Rsrc2
			—	unsigned operands

neg Rdest, Rsrc ★	Negate	Rdest = - Rsrc	—	no overflow
and Rdest, Rsrc1, Src2	Bitwise AND	Rdest = Rsrc1 & Src2	andi	—
or Rdest, Rsrc1, Src2	Bitwise OR	Rdest = Rsrc1 | Src2	ori	—
xor Rdest, Rsrc1, Src2	Bitwise Exclusive OR	Rdest = Rsrc1 ^ Src2	xori	—
nor Rdest, Rsrc1, Src2	Bitwise NOR	Rdest = ~(Rsrc1 | Src2)	★	—
not Rdest, Rsrc ★	Bitwise NOT	Rdest = ~Rsrc	—	—
sll Rdest, Rsrc1, Src2	Logical shift left	Rdest = Rsrc1 << Src2	—	—
srl Rdest, Rsrc1, Src2	Logical shift right	Rdest = Rsrc1 >> Src2 (MSB = 0)	—	—
sra Rdest, Rsrc1, Src2	Arithmetic shift right	Rdest = Rsrc1 >> Src2 (MSB preserved)	—	—
move Rdest, Rsrc ★	Move	Rdest = Rsrc	—	—
mfhi Rdest	Move from HI	Rdest = HI	—	—
mflo Rdest	Move from LO	Rdest = LO	—	—

Instruction format	Meaning	Operation	Immediate form	Unsigned form
li Rdest, Imm ★	Load immediate	Rdest = Imm	—	—
lui Rdest, imm16	Load upper immediate	Rdest = Imm16 << 16	—	—
la Rdest, Addr ★	Load address	Rdest = Addr	—	—
lb Rdest, Addr	Load byte	Rdest = *((char *) Addr)	—	zero-extend
lh Rdest, Addr	Load halfword	Rdest = *((short *) Addr)	—	zero-extend
lw Rdest, Addr	Load word	Rdest = *((long *) Addr)	—	—
sb Rsrc, Addr	Store byte	*((char *) Addr) = Rsrc	—	—
sh Rsrc, Addr	Store halfword	*((short *) Addr) = Rsrc	—	—
sw Rsrc, Addr	Store word	*((long *) Addr) = Rsrc	—	—
beq Rsrc1, Src2, label	branch if equal	if (Rsrc1 == Src2) PC = label	★	—
bne Rsrc1, Src2, label	branch if not equal	if (Rsrc1 != Src2) PC = label	★	—
blt Rsrc1, Src2, label ★	branch if less than	if (Rsrc1 < Src2) PC = label	★	unsigned operands
ble Rsrc1, Src2, label ★	branch if less than or equal	if (Rsrc1 <= Src2) PC = label	★	unsigned operands
bgt Rsrc1, Src2, label ★	branch if greater than	if (Rsrc1 > Src2) PC = label	★	unsigned operands
bge Rsrc1, Src2, label ★	branch if greater than or equal	if (Rsrc1 >= Src2) PC = label	★	unsigned operands

slt Rdest, Rsrc1, Src2	set if less than	if (Rsrc1 < Src2) Rdest = 1
		else Rdest = 0
			slti	unsigned operands

j label	jump	PC = label	—	—
jal label	jump and link	$ra = PC + 4; PC = label	—	—
jr Rsrc	jump register	PC = Rsrc	—	—
jalr Rsrc	jump and link register	$ra = PC + 4; PC = Rsrc	—	—
syscall	System call	depends on call code ($v0)	—	—