APPENDIX A

Quick Reference

INTEGER INSTRUCTION SET

Name	Syntax		Space/Time
Add	add	Rd, Rs, Rt	1/1
Add Immediate	addi	Rt, Rs, Imm	1/1
Add Immediate Unsigned	addiu	Rt, Rs, Imm	1/1
Add Unsigned	addu	Rd, Rs, Rt	1/1
And	and	Rd, Rs, Rt	1/1
And Immediate	andi	Rt, Rs, Imm	1/1
Branch if Equal	beq	Rs, Rt, Label	1/1
Branch if Greater Than or Equal to Zero	bgez	Rs, Label	1/1
Branch if Greater Than or Equal to Zero and Link	bgezal	Rs, Label	1/1
Branch if Greater Than Zero	bgtz	Rs, Label	1/1
Branch if Less Than or Equal to Zero	blez	Rs, Label	1/1
Branch if Less Than Zero and Link	bltzal	Rs, Label	1/1
Branch if Less Than Zero	bltz	Rs, Label	1/1
Branch if Not Equal	bne	Rs, Rt, Label	1/1
Divide	div	Rs, Rt	1/38
Divide Unsigned	divu	Rs, Rt	1/38
Jump	j	Label	1/1
Jump and Link	jal	Label	1/1
Jump and Link Register	jalr	Rd, Rs	1/1
Jump Register	jr	Rs	1/1
Load Byte	lb	Rt, offset(Rs)	1/1
Load Byte Unsigned	lbu	Rt, offset(Rs)	1/1
Load Halfword	lh	Rt, offset(Rs)	1/1
Load Halfword Unsigned	lhu	Rt, offset(Rs)	1/1
Load Upper Immediate	lui	Rt, Imm	1/1
Load Word	lw	Rt, offset(Rs)	1/1
Load Word Left	lwl	Rt, offset(Rs)	1/1
Load Word Right	lwr	Rt, offset(Rs)	1/1
Move From Coprocessor 0	mfc0	Rd, Cs	1/1
Move From High	mfhi	Rd	1/1
Move From Low	mflo	Rd	1/1

Move To Coprocessor 0	mtc0	Rt, Cd	1/1
Move to High	mthi	Rs	1/1
Move to Low	mtlo	Rs	1/1
Multiply	mult	Rs, Rt	1/32
Multiply Unsigned	multu	Rs, Rt	1/32
NOR	nor	Rd, Rs, Rt	1/1
OR	or	Rd, Rs, Rt	1/1
OR Immediate	ori	Rt, Rs, Imm	1/1
Return From Exception	rfe		1/1
Store Byte	sb	Rt, offset(Rs)	1/1
Store Halfword	sh	Rt, offset(Rs)	1/1
Shift Left Logical	sll	Rd, Rt, sa	1/1
Shift Left Logical Variable	sllv	Rd, Rt, Rs	1/1
Set on Less Than	slt	Rd, Rt, Rs	1/1
Set on Less Than Immediate	slti	Rt, Rs, Imm	1/1
Set on Less Than Immediate Unsigned	sltiu	Rt, Rs, Imm	1/1
Set on Less Than Unsigned	sltu	Rd, Rt, Rs	1/1
Shift Right Arithmetic	sra	Rd, Rt, sa	1/1
Shift Right Arithmetic Variable	srav	Rd, Rt, Rs	1/1
Shift Right Logical	srl	Rd, Rt, sa	1/1
Shift Right Logical Variable	srlv	Rd, Rt, Rs	1/1
Subtract	sub	Rd, Rs, Rt	1/1
Subtract Unsigned	subu	Rd, Rs, Rt	1/1
Store Word	sw	Rt, offset(Rs)	1/1
Store Word Left	swl	Rt, offset(Rs)	1/1
Store Word Right	swr	Rt, offset(Rs)	1/1
System Call	syscall		1/1
Exclusive OR	xor	Rd, Rs, Rt	1/1
Exclusive OR Immediate	xori	Rt, Rs, Imm	1/1

MACRO INSTRUCTIONS

Name	Syntax		Space/Time
Absolute Value	abs	Rd, Rs	3/3
Branch if Equal to Zero	beqz	Rs, Label	1/1
Branch if Greater Than or Equal	bge	Rs, Rt, Label	2/2
Branch if Greater Than or Equal Unsigned	bgeu	Rs, Rt, Label	2/2
Branch if Greater Than	bgt	Rs, Rt, Label	2/2
Branch if Greater Than Unsigned	bgtu	Rs, Rt, Label	2/2
Branch if Less Than or Equal	ble	Rs, Rt, Label	2/2
Branch if Less Than or Equal Unsigned	bleu	Rs, Rt, Label	2/2
Branch if Less Than	blt	Rs, Rt, Label	2/2
Branch if Less Than Unsigned	bltu	Rs, Rt, Label	2/2
Branch if Not Equal to Zero	bnez	Rs, Label	1/1
Branch Unconditional	b	Label	1/1
Divide	div	Rd, Rs, Rt	4/41
Divide Unsigned	divu	Rd, Rs, Rt	4/41
Load Address	la	Rd, Label	2/2
Load Immediate	li	Rd, value	2/2
Move	move	Rd, Rs	1/1
Multiply	mul	Rd, Rs, Rt	2/33
Multiply (with overflow exception)	mulo	Rd, Rs, Rt	7/37
Multiply Unsigned (with overflow exception)	mulou	Rd, Rs, Rt	5/35
Negate	neg	Rd, Rs	1/1

Negate Unsigned	negu	Rd, Rs	1/1
Nop	nop		1/1
Not	not	Rd, Rs	1/1
Remainder Unsigned	remu	Rd, Rs, Rt	4/40
Rotate Left Variable	rol	Rd, Rs, Rt	4/4
Rotate Right Variable	ror	Rd, Rs, Rt	4/4
Remainder	rem	Rd, Rs, Rt	4/40
Rotate Left Constant	rol	Rd, Rs, sa	3/3
Rotate Right Constant	ror	Rd, Rs, sa	3/3
Set if Equal	seq	Rd, Rs, Rt	4/4
Set if Greater Than or Equal	sge	Rd, Rs, Rt	4/4
Set if Greater Than or Equal Unsigned	sgeu	Rd, Rs, Rt	4/4
Set if Greater Than	sgt	Rd, Rs, Rt	1/1
Set if Greater Than Unsigned	sgtu	Rd, Rs, Rt	1/1
Set if Less Than or Equal	sle	Rd, Rs, Rt	4/4
Set if Less Than or Equal Unsigned	sleu	Rd, Rs, Rt	4/4
Set if Not Equal	sne	Rd, Rs, Rt	4/4
Unaligned Load Halfword Unsigned	ulh	Rd, n(Rs)	4/4
Unaligned Load Halfword	ulhu	Rd, n(Rs)	4/4
Unaligned Load Word	ulw	Rd, n(Rs)	2/2
Unaligned Store Halfword	ush	Rd, n(Rs)	3/3
Unaligned Store Word	usw	Rd, n(Rs)	2/2

SYSTEM I/O SERVICES

Service	Code in $v0	Argument(s)	Result(s)
Print Integer	1	$a0 = number to be printed	
Print Float	2	$f12 = number to be printed	
Print Double	3	$f12 = number to be printed	
Print String	4	$a0 = address of string in memory	
Read Integer	5		number returned in $v0
Read Float	6		number returned in $f0
Read Double	7		number returned in $f0
Read String	8	$a0 = address of input buffer in memory $a1 = length of buffer (n)	
Sbrk	9	$a0 = amount	address in $v0
Exit	10		

The system call Read Integer reads an entire line of input from the keyboard up to and including the newline. Characters following the last digit in the decimal number are ignored. Read String has the same semantics as the Unix library routine fgets. It reads up to $n - 1$ characters into a buffer and terminates the string with a null byte. If fewer than $n - 1$ characters are on the current line, Read String reads up to and including the newline and again null terminates the string. Print String will display on the terminal the string of characters found in memory starting with the location pointed to by the address stored in `$a0`. Printing will stop when a null character is located in the string. Sbrk returns a pointer to a block of memory containing n additional bytes. Exit terminates the user program execution and returns control to the operating system.

ASSEMBLER DIRECTIVES

.align n	Align the next datum on a 2^n byte boundary. For example, .align 2 aligns the next value on a word boundary. .align 0 turns off automatic alignment of .half, .word, .float, and .double directives until the next .data or .kdata directive.
.ascii string*	Store the string in memory, but do not null-terminate it.
.asciiz string*	Store the string in memory and null-terminate it.
.byte b1,..., bn	Store the *n* 8-bit values in successive bytes of memory.
.data <addr>	Subsequent items are stored in the data segment. If the optional argument *addr* is present, subsequent items are stored starting at address *addr*. For example: `.data 0x00008000`
.double d1, ..., dn	Store the n floating-point double-precision numbers in successive memory locations.
.extern Symb size	Declare that the datum stored at *Symb* is of size bytes large and is a global label. This directive enables the assembler to store the datum in a portion of the data segment that is efficiently accessed via register `$gp`.
.float f1, ..., fn	Store the *n* floating-point single-precision numbers in successive memory locations.
.globl Symb	Declare that label *Symb* is global so it can be referenced from other files.
.half h1,... hn	Store the *n* 16-bit quantities in successive memory half words.
.kdata <add>	Subsequent items are stored in the kernel data segment. If the optional argument *addr* is present, subsequent items are stored starting at address *addr*.
.ktext <addr>	Subsequent items are put in the kernal text segment. In SPIM, these items may only be instructions or words. If the optional argument *addr* is present, subsequent items are stored starting at address *addr* (e.g., `.ktext 0x80000080`).
.space n	Allocate *n* bytes of space in the current segment (which must be the data segment in PCSpim).
.text <addr>	Subsequent items are put in the user text segment. In SPIM, these items may only be instructions or words (see the .word directive below). If the optional argument *addr* is present, subsequent items are stored starting at address *addr* (e.g., `.data 0x00400000`).
.word w1,..., wn	Store the *n* 32-bit quantities in successive memory words.
.word w : n	Stores the 32-bit value *w* into *n* successive memory words.

*Strings are enclosed in double quotes ("). Special characters in strings follow the C convention: `newline: \n, tab: \t, quote: \"`. Instruction op-codes are reserved words and may not be used as labels. Labels must appear at the beginning of a line followed by a colon. The ASCII code "back space" is not supported by the SPIM simulator. Numbers are base 10 by default. If they are preceded by *0x*, they are interpreted as hexadecimal. Hence, 256 and 0x100 denote the same value.