The following pages contain a list of MIPS instructions for use during the exam. You do not need to submit these pages when you finish your exam. Note that:

- Constants are in decimal unless explicitly written as hexadecimal using leading 0x (i.e., 0x1234)
- The notation "`mem[<addr>]`" means the contents of memory at the address in brackets.
    - `<addr>` is typically specified as the sum of a base register and constant offset. For example, the instruction `lw $t0, 0($s0)` accesses address `0+$s0`.

Category	Instruction	Example	Meaning
Data transfer	Load byte (signed)	`lb $t0, 0($t1)`	`$t0 = sign-extended byte at mem[0+$t1]`
	Load halfword (signed)	`lh $t3, 6($s0)`	`$t3 = sign-extended halfword at mem[6+$s0]`

	Load word	`lw $s3, 12($t4)`
		`lwu $s3, 12($t4)`
			`$s3 = word at mem[12+$t4]`

	Load byte (unsigned)	`lbu $t0, 0($t1)`	`$t0 = zero-extended byte at mem[0+$t1]`
	Load halfword (unsigned)	`lhu $t3, 6($s0)`	`$t3 = zero-extended halfword at mem[6+$s0]`
	Store byte	`sb $t0, 3($s4)`	`Byte at mem[3+$s4] = lowest byte of $t0`
	Store halfword	`sh $t1, 10($t0)`	`Halfword at mem[10+$t0] = lowest halfword of $t1`
	Store word	`sw $t2, 4($s0)`	`Word at mem[4+$s0] = full contents of $t2`

	Move from Hi/Lo (special regs used for multiplication/ division)	`mfhi $t0`
		`mflo $t1`
			`$t0 = Hi`
			`$t1 = Lo`

Arithmetic	Add	`add $t0, $t1, $t2`
		`addu $t0, $t1, $t2`
			`$t0 = $t1 + $t2`
			`(addu ignores overflow)`

	Add immediate	`addi $t0, $t1, 16`
		`addiu $t0, $t1, 16`
			`$t0 = $t1 + 16`
			`(addiu ignores overflow)`

	Subtract	`sub $t3, $t4, $t5`
		`subu $t3, $t4, $t5`
			`$t3 = $t4 - $t5`
			`(subu ignores overflow)`

	Multiply	`mult $s0, $s1`
		`multu $s0, $s1`
		
		`mul $s2, $s0, $s1`
			`(Hi,Lo) = $s0 * $s1`
			`Hi = upper 32 bits of result, Lo = lower 32 bits of result`
			
			`$s2 = lowest 32 bits of $s0 * $s1`

Logical	Logical AND	`and $t0, $t1, $t2`	`$t0 = $t1 AND $t2`
	AND immediate	`andi $t0, $t1, 0xFFFF`	`$t0 = $t1 AND 0x0000FFFF`
	Logical inclusive OR	`or $t3, $t4, $t5`	`$t3 = $t4 OR $t5`
	OR immediate	`ori $t3, $t4, 0x1001`	`$t3 = $t4 OR 0x00001001`
	Logical exclusive OR	`xor $t6, $t7, $t8`	`$t6 = $t7 XOR $t8`
	XOR immediate	`xori $t6, $t7, 0xABCD`	`$t6 = $t7 XOR 0x0000ABCD`
	Logical NOR	`nor $s0, $s1, $s2`	`$s0 = $s1 NOR $s2`

Category	Instruction	Example	Meaning
Shift	Shift left	`sll $t0, $t1, 5`	`$t0 = $t1 << 5`

	Logical shift right	`srl $s5, $s6, 4`	`$s5 = $s6 >> 4`
			`(upper 4 bits = 0)`

	Arithmetic shift right (treat value as signed; maintain sign)	`srl $s5, $s6, 4`	`$s5 = $s6 >> 4`
			`(upper 4 bits = MSB of original value)`

Misc. computation	Set less than	`slt $t5, $t0, $t1`	`$t5 = 1 if $t0 < $t1`
			`(signed comparison)`
			`$t5 = 0 otherwise`

	Set less than unsigned	`sltu $t5, $t0, $t1`	`$t5 = 1 if $t0 < $t1`
			`(unsigned comparison)`
			`$t5 = 0 otherwise`

	Set less than immediate	`slti $s0, $t0, 14`	`$s0 = 1 if $t0 < 14`
			`(signed comparison)`
			`$s0 = 0 otherwise`

	Set less than immediate unsigned	`sltiu $s0, $t0, 14`	`$s0 = 1 if $t0 < 14`
			`(unsigned comparison)`
			`$s0 = 0 otherwise`

	Load upper immediate	`lui $t4, 0x1234`	`$t4 = 0x12340000`

Control flow	Branch on equal	`beq $t0, $t1, label`	`Jump to "label" if $t0 == $t1`
			`Otherwise, go to next sequential instruction`

	Branch on not equal	`bne $t0, $t1, label`	`Jump to "label" if $t0 != $t1`
			`Otherwise, go to next sequential instruction`

	Unconditional jump	`j label`	`Jump to "label"`
	Register jump	`jr $ra`	`Jump to address stored in register $ra`
	Jump and link	`jal f`	`Store PC+4 (return address) on stack, then jump to "f"`
Floating point (Instructions ending in .s are single precision; .d are double precision. Registers are paired in double-precision ops.)	FP add	`add.d F0, F2, F4`	`(F1/F0) = (F3/F2)+(F5/F4)`
	FP subtract	`sub.d F6, F8, F0`	`(F7/F6) = (F9/F8)-(F1/F0)`
	FP multiply	`mult.s F0, F1, F2`	`F0 = F1 * F2`
	FP divide	`div.s F4, F5, F6`	`F4 = F5 / F6`

	FP load	`l.d F0, 0(R2)`	`F0 = lower 32 bits of double-precision value at mem[0+R2]`
			`F1 = upper 32 bits of double-precision value at mem[0+R2]`

	FP store	`s.s F5, 10(R1)`	`mem[10+R1] = single-precision value stored in F5`