F. Appendix 6 – MIPS Instruction Reference

Note: ALL immediate values should be sign extended.
Exception: For logical operations immediate values should be zero extended.
After extensions, you treat them as signed or unsigned 32-bit numbers.

For the non-immediate instructions, the only difference between signed and unsigned instructions (ex ADD vs. ADDU) is that signed instructions can generate an overflow.

The instruction formats are given, you can figure out the binary instruction codes. The instruction descriptions are given below. Additional details can be found here: "MIPS Single Cycle Processor", John Alexander, Barret Schloerke, Daniel Sedam, Iowa State University

ADD – Add

Description:	Adds two registers and stores the result in a register
Operation:	$d ← $s + $t; advance_pc (4);
Syntax:	add $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0000

ADDI – Add immediate

Description:	Adds a register and a signed immediate value and stores the result in a register
Operation:	$t ← $s + imm; advance_pc (4);
Syntax:	addi $t, $s, imm
Encoding:	0010 00ss ssst tttt iiii iiii iiii iiii

ADDIU – Add immediate unsigned

Description:	Adds a register and an unsigned immediate value and stores the result in a register
Operation:	$t ← $s + imm; advance_pc (4);
Syntax:	addiu $t, $s, imm
Encoding:	0010 01ss ssst tttt iiii iiii iiii iiii

ADDU – Add unsigned

Description:	Adds two registers and stores the result in a register
Operation:	$d ← $s + $t; advance_pc (4);
Syntax:	addu $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0001

AND – Bitwise and

Description:	Bitwise ands two registers and stores the result in a register
Operation:	$d ← $s & $t; advance_pc (4);
Syntax:	and $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0100

ANDI – Bitwise and immediate

Description:	Bitwise ands a register and an immediate value and stores the result in a register
Operation:	$t ← $s & imm; advance_pc (4);
Syntax:	andi $t, $s, imm
Encoding:	0011 00ss ssst tttt iiii iiii iiii iiii

BEQ – Branch on equal

Description:	Branches if the two registers are equal
Operation:	if $s == $t advance_pc (offset << 2); else advance_pc (4);
Syntax:	beq $s, $t, offset
Encoding:	0001 00ss ssst tttt iiii iiii iiii iiii

BGEZ – Branch on greater than or equal to zero

Description:	Branches if the register is greater than or equal to zero
Operation:	if $s >= 0 advance_pc (offset << 2); else advance_pc (4);
Syntax:	bgez $s, offset
Encoding:	0000 01ss sss0 0001 iiii iiii iiii iiii

BGEZAL – Branch on greater than or equal to zero and link

Description:	Branches if the register is greater than or equal to zero and saves the return address in $31
Operation:	if $s >= 0 $31 = PC + 8 (or nPC + 4); advance_pc (offset << 2); else advance_pc (4);
Syntax:	bgezal $s, offset
Encoding:	0000 01ss sss1 0001 iiii iiii iiii iiii

BGTZ – Branch on greater than zero

Description:	Branches if the register is greater than zero
Operation:	if $s > 0 advance_pc (offset << 2); else advance_pc (4);
Syntax:	bgtz $s, offset
Encoding:	0001 11ss sss0 0000 iiii iiii iiii iiii

BLEZ – Branch on less than or equal to zero

Description:	Branches if the register is less than or equal to zero
Operation:	if $s <= 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	blez $s, offset
Encoding:	0001 10ss sss0 0000 iiii iiii iiii iiii

BLTZ – Branch on less than zero

Description:	Branches if the register is less than zero
Operation:	if $s < 0 advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bltz $s, offset
Encoding:	0000 01ss sss0 0000 iiii iiii iiii iiii

BLTZAL – Branch on less than zero and link

Description:	Branches if the register is less than zero and saves the return address in $31
Operation:	if $s < 0 $31 = PC + 8 (or nPC + 4); advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bltzal $s, offset
Encoding:	0000 01ss sss1 0000 iiii iiii iiii iiii

BNE – Branch on not equal

Description:	Branches if the two registers are not equal
Operation:	if $s != $t advance_pc (offset << 2)); else advance_pc (4);
Syntax:	bne $s, $t, offset
Encoding:	0001 01ss ssst tttt iiii iiii iiii iiii

DIV – Divide

Description:	Divides $s by $t and stores the quotient in $LO and the remainder in $HI
Operation:	$LO ← $s / $t; $HI ← $s % $t; advance_pc (4);
Syntax:	div $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1010

DIVU – Divide unsigned

Description:	Divides $s by $t and stores the quotient in $LO and the remainder in $HI
Operation:	$LO ← $s / $t; $HI ← $s % $t; advance_pc (4);
Syntax:	divu $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1011

J – Jump

Description:	Jumps to the calculated address
Operation:	PC ← nPC; nPC = (PC & 0xf0000000) | (target << 2);
Syntax:	j target
Encoding:	0000 10ii iiii iiii iiii iiii iiii iiii

JAL – Jump and link

Description:	Jumps to the calculated address and stores the return address in $31
Operation:	$31 ← PC + 8 (or nPC + 4); PC = nPC; nPC = (PC & 0xf0000000) | (target << 2);
Syntax:	jal target
Encoding:	0000 11ii iiii iiii iiii iiii iiii iiii

JR – Jump register

Description:	Jump to the address contained in register $s
Operation:	PC ← nPC; nPC = $s;
Syntax:	jr $s
Encoding:	0000 00ss sss0 0000 0000 0000 0000 1000

LB – Load byte

Description:	A byte is loaded into a register from the specified address.
Operation:	$t ← MEM[$s + offset]; advance_pc (4);
Syntax:	lb $t, offset($s)
Encoding:	1000 00ss ssst tttt iiii iiii iiii iiii

LUI – Load upper immediate

Description:	The immediate value is shifted left 16 bits and stored in the register. The lower 16 bits are zeroes.
Operation:	$t ← (imm << 16); advance_pc (4);
Syntax:	lui $t, imm
Encoding:	0011 11-- ---t tttt iiii iiii iiii iiii

LW – Load word

Description:	A word is loaded into a register from the specified address.
Operation:	$t ← MEM[$s + offset]; advance_pc (4);
Syntax:	lw $t, offset($s)
Encoding:	1000 11ss ssst tttt iiii iiii iiii iiii

MFHI – Move from HI

Description:	The contents of register HI are moved to the specified register.
Operation:	$d ← $HI; advance_pc (4);
Syntax:	mfhi $d
Encoding:	0000 0000 0000 0000 dddd d000 0001 0000

MFLO – Move from LO

Description:	The contents of register LO are moved to the specified register.
Operation:	$d ← $LO; advance_pc (4);
Syntax:	mflo $d
Encoding:	0000 0000 0000 0000 dddd d000 0001 0010

MULT – Multiply

Description:	Multiplies $s by $t and stores the result in $Hi and $LO.
Operation:	$Hi, $LO ← $s * $t; advance_pc (4);
Syntax:	mult $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1000

MULTU – Multiply unsigned

Description:	Multiplies $s by $t and stores the result in $Hi and $LO.
Operation:	$Hi, $LO ← $s * $t; advance_pc (4);
Syntax:	multu $s, $t
Encoding:	0000 00ss ssst tttt 0000 0000 0001 1001

NOOP – no operation

Description:	Performs no operation.
Operation:	advance_pc (4);
Syntax:	noop
Encoding:	0000 0000 0000 0000 0000 0000 0000 0000

Note: The encoding for a NOOP represents the instruction SLL $0, $0, 0 which has no side effects. In fact, nearly every instruction that has $0 as its destination register will have no side effect and can thus be considered a NoOP instruction.

OR – Bitwise or

Description:	Bitwise logical ors two registers and stores the result in a register
Operation:	$d ← $s | $t; advance_pc (4);
Syntax:	or $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0101

ORI – Bitwise or immediate

Description:	Bitwise ors a register and an immediate value and stores the result in a register
Operation:	$t ← $s | imm; advance_pc (4);
Syntax:	ori $t, $s, imm
Encoding:	0011 01ss ssst tttt iiii iiii iiii iiii

SB – Store byte

Description:	The least significant byte of $t is stored at the specified address.
Operation:	MEM[$s + offset] ← (0xff & $t); advance_pc (4);
Syntax:	sb $t, offset($s)
Encoding:	1010 00ss ssst tttt iiii iiii iiii iiii

SLL – Shift left logical

Description:	Shifts a register value left by the shift amount listed in the instruction and places the result in a third register. Zeroes are shifted in.
Operation:	$d ← $t << h; advance_pc (4);
Syntax:	sll $d, $t, h
Encoding:	0000 00ss ssst tttt dddd dhhh hh00 0000

SLLV – Shift left logical variable

Description:	Shifts a register value left by the value in a second register and places the result in a third register. Zeroes are shifted in.
Operation:	$d ← $t << $s; advance_pc (4);
Syntax:	sllv $d, $t, $s
Encoding:	0000 00ss ssst tttt dddd d--- --00 0100

SLT – Set on less than (signed)

Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d ← 1; advance_pc (4); else $d ← 0; advance_pc (4);
Syntax:	slt $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 1010

SLTI – Set on less than immediate (signed)

Description:	If $s is less than immediate, $t is set to one. It gets zero otherwise.
Operation:	if $s < imm $t ← 1; advance_pc (4); else $t ← 0; advance_pc (4);
Syntax:	slti $t, $s, imm
Encoding:	0010 10ss ssst tttt iiii iiii iiii iiii

SLTIU – Set on less than immediate unsigned

Description:	If $s is less than the unsigned immediate, $t is set to one. It gets zero otherwise.
Operation:	if $s < imm $t ← 1; advance_pc (4); else $t ← 0; advance_pc (4);
Syntax:	sltiu $t, $s, imm
Encoding:	0010 11ss ssst tttt iiii iiii iiii iiii

SLTU – Set on less than unsigned

Description:	If $s is less than $t, $d is set to one. It gets zero otherwise.
Operation:	if $s < $t $d ← 1; advance_pc (4); else $d ← 0; advance_pc (4);
Syntax:	sltu $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 1011

SRA – Shift right arithmetic

Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. The sign bit is shifted in.
Operation:	$d ← $t >> h; advance_pc (4);
Syntax:	sra $d, $t, h
Encoding:	0000 00-- ---t tttt dddd dhhh hh00 0011

SRL – Shift right logical

Description:	Shifts a register value right by the shift amount (shamt) and places the value in the destination register. Zeroes are shifted in.
Operation:	$d ← $t >> h; advance_pc (4);
Syntax:	srl $d, $t, h
Encoding:	0000 00-- ---t tttt dddd dhhh hh00 0010

SRLV – Shift right logical variable

Description:	Shifts a register value right by the amount specified in $s and places the value in the destination register. Zeroes are shifted in.
Operation:	$d ← $t >> $s; advance_pc (4);
Syntax:	srlv $d, $t, $s
Encoding:	0000 00ss ssst tttt dddd d000 0000 0110

SUB – Subtract

Description:	Subtracts two registers and stores the result in a register
Operation:	$d ← $s - $t; advance_pc (4);
Syntax:	sub $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0010

SUBU – Subtract unsigned

Description:	Subtracts two registers and stores the result in a register
Operation:	$d ← $s - $t; advance_pc (4);
Syntax:	subu $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d000 0010 0011

SW – Store word

Description:	The contents of $t is stored at the specified address.
Operation:	MEM[$s + offset] ← $t; advance_pc (4);
Syntax:	sw $t, offset($s)
Encoding:	1010 11ss ssst tttt iiii iiii iiii iiii

SYSCALL – System call

Description:	Generates a software interrupt.
Operation:	advance_pc (4);
Syntax:	syscall
Encoding:	0000 00-- ---- ---- ---- ---- --00 1100

XOR – Bitwise exclusive or

Description:	Exclusive ors two registers and stores the result in a register
Operation:	$d ← $s ^ $t; advance_pc (4);
Syntax:	xor $d, $s, $t
Encoding:	0000 00ss ssst tttt dddd d--- --10 0110

XORI – Bitwise exclusive or immediate

Description:	Bitwise exclusive ors a register and an immediate value and stores the result in a register
Operation:	$t ← $s ^ imm; advance_pc (4);
Syntax:	xori $t, $s, imm
Encoding:	0011 10ss ssst tttt iiii iiii iiii iiii