Instruction – Machine Code – Description
(Operand substitution: f=first s=second t=third)

R-Type

op	rs	rt	rd	shamt	funct
6-bit	5-bit	5-bit	5-bit	5-bit	6-bit

abs.d $f2, $f4
010001 10001 00000 sssss fffff 000101
Floating point absolute value double precision : Set $f2 to absolute value of $f4, double precision

abs.s $f0, $f1
010001 10000 00000 sssss fffff 000101
Floating point absolute value single precision : Set $f0 to absolute value of $f1, single precision

add $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100000
Addition with overflow : set $t1 to ($t2 plus $t3)

add.d $f2, $f4, $f6
010001 10001 ttttt sssss fffff 000000
Floating point addition double precision : Set $f2 to double-precision floating point value of $f4 plus $f6

add.s $f0, $f1, $f3
010001 10000 ttttt sssss fffff 000000
Floating point addition single precision : Set $f0 to single-precision floating point value of $f1 plus $f3

addu $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100001
Addition unsigned without overflow : set $t1 to ($t2 plus $t3), no overflow

and $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100100
Bitwise AND : Set $t1 to bitwise AND of $t2 and $t3

break 100
000000 ffffffffffffffffffff 001101
Break execution with code : Terminate program execution with specified exception code

break
000000 00000 00000 00000 00000 001101
Break execution : Terminate program execution with exception

c.eq.d $f2, $f4
010001 10001 sssss fffff 00000 110010
Compare equal double precision : If $f2 is equal to $f4 (double-precision), set Coprocessor 1 condition flag 0 true else set it false

c.eq.d 1, $f2, $f4
010001 10001 ttttt sssss fff 00 110010
Compare equal double precision : If $f2 is equal to $f4 (double-precision), set Coprocessor 1 condition flag specified by immediate to true else set it to false

c.eq.s $f0, $f1
010001 10000 sssss fffff 00000 110010
Compare equal single precision : If $f0 is equal to $f1, set Coprocessor 1 condition flag 0 true else set it false

c.eq.s 1, $f0, $f1
010001 10000 ttttt sssss fff 00 11 0010
Compare equal single precision : If $f0 is equal to $f1, set Coprocessor 1 condition flag specied by immediate to true else set it to false

c.le.d $f2, $f4
010001 10001 sssss fffff 00000 111110
Compare less or equal double precision : If $f2 is less than or equal to $f4 (double-precision), set Coprocessor 1 condition flag 0 true else set it false

c.le.d 1, $f2, $f4
010001 10001 ttttt sssss fff 00 111110
Compare less or equal double precision : If $f2 is less than or equal to $f4 (double-precision), set Coprocessor 1 condition flag specfied by immediate true else set it false

c.le.s $f0, $f1
010001 10000 sssss fffff 00000 111110
Compare less or equal single precision : If $f0 is less than or equal to $f1, set Coprocessor 1 condition flag 0 true else set it false

c.le.s 1, $f0, $f1
010001 10000 ttttt sssss fff 00 111110
Compare less or equal single precision : If $f0 is less than or equal to $f1, set Coprocessor 1 condition flag specified by immediate to true else set it to false

c.lt.d $f2, $f4
010001 10001 sssss fffff 00000 111100
Compare less than double precision : If $f2 is less than $f4 (double-precision), set Coprocessor 1 condition flag 0 true else set it false

c.lt.d 1, $f2, $f4
010001 10001 ttttt sssss fff 00 111100
Compare less than double precision : If $f2 is less than $f4 (double-precision), set Coprocessor 1 condition flag specified by immediate to true else set it to false

c.lt.s $f0, $f1
010001 10000 sssss fffff 00000 111100
Compare less than single precision : If $f0 is less than $f1, set Coprocessor 1 condition flag 0 true else set it false

c.lt.s 1, $f0, $f1
010001 10000 ttttt sssss fff 00 111100
Compare less than single precision : If $f0 is less than $f1, set Coprocessor 1 condition flag specified by immediate to true else set it to false

ceil.w.d $f1, $f2
010001 10001 00000 sssss fffff 001110
Ceiling double precision to word : Set $f1 to 32-bit integer ceiling of double-precision float in $f2

ceil.w.s $f0, $f1
010001 10000 00000 sssss fffff 001110
Ceiling single precision to word : Set $f0 to 32-bit integer ceiling of single-precision float in $f1

clo $t1, $t2
011100 sssss 00000 fffff 00000 100001
Count number of leading ones : Set $t1 to the count of leading one bits in $t2 starting at most significant bit position. rt bits should equal rd.

clz $t1, $t2
011100 sssss 00000 fffff 00000 100000
Count number of leading zeroes : Set $t1 to the count of leading zero bits in $t2 starting at most significant bit position. rt bits should equal rd.

cvt.d.s $f2, $f1
010001 10000 00000 sssss fffff 100001
Convert from single precision to double precision : Set $f2 to double precision equivalent of single precision value in $f1

cvt.d.w $f2, $f1
010001 10100 00000 sssss fffff 100001
Convert from word to double precision : Set $f2 to double precision equivalent of 32-bit integer value in $f1

cvt.s.d $f1, $f2
010001 10001 00000 sssss fffff 100000
Convert from double precision to single precision : Set $f1 to single precision equivalent of double precision value in $f2

cvt.s.w $f0, $f1
010001 10100 00000 sssss fffff 100000
Convert from word to single precision : Set $f0 to single precision equivalent of 32-bit integer value in $f2

cvt.w.d $f1, $f2
010001 10001 00000 sssss fffff 100100
Convert from double precision to word : Set $f1 to 32-bit integer equivalent of double precision value in $f2

cvt.w.s $f0, $f1
010001 10000 00000 sssss fffff 100100
Convert from single precision to word : Set $f0 to 32-bit integer equivalent of single precision value in $f1

div $t1, $t2
000000 fffff sssss 00000 00000 011010
Division with overflow : Divide $t1 by $t2 then set LO to quotient and HI to remainder (use mfhi to access HI, mflo to access LO)

div.d $f2, $f4, $f6
010001 10001 ttttt sssss fffff 000011
Floating point division double precision : Set $f2 to double-precision floating point value of $f4 divided by $f6

div.s $f0, $f1, $f3
010001 10000 ttttt sssss fffff 000011
Floating point division single precision : Set $f0 to single-precision floating point value of $f1 divided by $f3

divu $t1, $t2
000000 fffff sssss 00000 00000 011011
Division unsigned without overflow : Divide unsigned $t1 by $t2 then set LO to quotient and HI to remainder (use mfhi to access HI, mflo to access LO)

eret
010000 1 00000000000000000000 011000
Exception return : Set Program Counter to Coprocessor 0 EPC register value, set Coprocessor Status register bit 1 (exception level) to zero

floor.w.d $f1, $f2
010001 10001 00000 sssss fffff 001111
Floor double precision to word : Set $f1 to 32-bit integer floor of double-precision float in $f2

floor.w.s $f0, $f1
010001 10000 00000 sssss fffff 001111
Floor single precision to word : Set $f0 to 32-bit integer floor of single-precision float in $f1

jalr $t1, $t2
000000 sssss 00000 fffff 00000 001001
Jump and link register : Set $t1 to Program Counter (return address) then jump to statement whose address is in $t2

jalr $t1
000000 fffff 00000 11111 00000 001001
Jump and link register : Set $ra to Program Counter (return address) then jump to statement whose address is in $t1

jr $t1
000000 fffff 00000 00000 00000 001000
Jump register unconditionally : Jump to statement whose address is in $t1

madd $t1, $t2
011100 fffff sssss 00000 00000 000000
Multiply add : Multiply $t1 by $t2 then increment HI by high-order 32 bits of product, increment LO by low-order 32 bits of product (use mfhi to access HI, mflo to access LO)

maddu $t1, $t2
011100 fffff sssss 00000 00000 000001
Multiply add unsigned : Multiply $t1 by $t2 then increment HI by high-order 32 bits of product, increment LO by low-order 32 bits of product, unsigned (use mfhi to access HI, mflo to access LO)

mfc0 $t1, $8
010000 00000 fffff sssss 00000 000000
Move from Coprocessor 0 : Set $t1 to the value stored in Coprocessor 0 register $8

mfc1 $t1, $f1
010001 00000 fffff sssss 00000 000000
Move from Coprocessor 1 (FPU) : Set $t1 to value in Coprocessor 1 register $f1

mfhi $t1
000000 00000 00000 fffff 00000 010000
Move from HI register : Set $t1 to contents of HI (see multiply and divide operations)

mflo $t1
000000 00000 00000 fffff 00000 010010
Move from LO register : Set $t1 to contents of LO (see multiply and divide operations)

mov.d $f2, $f4
010001 10001 00000 sssss fffff 000110
Move floating point double precision : Set double precision $f2 to double precision value in $f4

mov.s $f0, $f1
010001 10000 00000 sssss fffff 000110
Move floating point single precision : Set single precision $f0 to single precision value in $f1

movf $t1, $t2, 1
000000 sssss ttt 00 fffff 00000 000001
Move if specified FP condition flag false : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag specified by the immediate is false (zero)

movf $t1, $t2
000000 sssss 000 00 fffff 00000 000001
Move if FP condition flag 0 false : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag 0 is false (zero)

movf.d $f2, $f4, 1
010001 10001 ttt 00 sssss fffff 010001
Move floating point double precision : If condition flag specified by immediate is false, set double precision $f2 to double precision value in $f4

movf.d $f2, $f4
010001 10001 000 00 sssss fffff 010001
Move floating point double precision : If condition flag 0 false, set double precision $f2 to double precision value in $f4

movf.s $f0, $f1, 1
010001 10000 ttt 00 sssss fffff 010001
Move floating point single precision : If condition flag specified by immediate is false, set single precision $f0 to single precision value in $f1e

movf.s $f0, $f1
010001 10000 000 00 sssss fffff 010001
Move floating point single precision : If condition flag 0 is false, set single precision $f0 to single precision value in $f1

movn $t1, $t2, $t3
000000 sssss ttttt fffff 00000 001011
Move conditional not zero : Set $t1 to $t2 if $t3 is not zero

movn.d $f2, $f4, $t3
010001 10001 ttttt sssss fffff 010011
Move floating point double precision : If $t3 is not zero, set double precision $f2 to double precision value in $f4

movn.s $f0, $f1, $t3
010001 10000 ttttt sssss fffff 010011
Move floating point single precision : If $t3 is not zero, set single precision $f0 to single precision value in $f1

movt $t1, $t2, 1
000000 sssss ttt 01 fffff 00000 000001
Move if specfied FP condition flag true : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag specified by the immediate is true (one)

movt $t1, $t2
000000 sssss 000 01 fffff 00000 000001
Move if FP condition flag 0 true : Set $t1 to $t2 if FPU (Coprocessor 1) condition flag 0 is true (one)

movt.d $f2, $f4, 1
010001 10001 ttt 01 sssss fffff 010001
Move floating point double precision : If condition flag specified by immediate is true, set double precision $f2 to double precision value in $f4e

movt.d $f2, $f4
010001 10001 000 01 sssss fffff 010001
Move floating point double precision : If condition flag 0 true, set double precision $f2 to double precision value in $f4

movt.s $f0, $f1, 1
010001 10000 ttt 01 sssss fffff 010001
Move floating point single precision : If condition flag specified by immediate is true, set single precision $f0 to single precision value in $f1e

movt.s $f0, $f1
010001 10000 000 01 sssss fffff 010001
Move floating point single precision : If condition flag 0 is true, set single precision $f0 to single precision value in $f1e

movz $t1, $t2, $t3
000000 sssss ttttt fffff 00000 001010
Move conditional zero : Set $t1 to $t2 if $t3 is zero

movz.d $f2, $f4, $t3
010001 10001 ttttt sssss fffff 010010
Move floating point double precision : If $t3 is zero, set double precision $f2 to double precision value in $f4

movz.s $f0, $f1, $t3
010001 10000 ttttt sssss fffff 010010
Move floating point single precision : If $t3 is zero, set single precision $f0 to single precision value in $f1

msub $t1, $t2
011100 fffff sssss 00000 00000 000100
Multiply subtract : Multiply $t1 by $t2 then decrement HI by high-order 32 bits of product, decrement LO by low-order 32 bits of product (use mfhi to access HI, mflo to access LO)

msubu $t1, $t2
011100 fffff sssss 00000 00000 000101
Multiply subtract unsigned : Multiply $t1 by $t2 then decrement HI by high-order 32 bits of product, decement LO by low-order 32 bits of product, unsigned (use mfhi to access HI, mflo to access LO)

mtc0 $t1, $8
010000 00100 fffff sssss 00000 000000
Move to Coprocessor 0 : Set Coprocessor 0 register $8 to value stored in $t1

mtc1 $t1, $f1
010001 00100 fffff sssss 00000 000000
Move to Coprocessor 1 (FPU) : Set Coprocessor 1 register $f1 to value in $t1

mthi $t1
000000 fffff 00000 00000 00000 010001
Move to HI registerr : Set HI to contents of $t1 (see multiply and divide operations)

mtlo $t1
000000 fffff 00000 00000 00000 010011
Move to LO register : Set LO to contents of $t1 (see multiply and divide operations)

mul $t1, $t2, $t3
011100 sssss ttttt fffff 00000 000010
Multiplication without overflow : Set HI to high-order 32 bits, LO and $t1 to low-order 32 bits of the product of $t2 and $t3 (use mfhi to access HI, mflo to access LO)

mul.d $f2, $f4, $f6
010001 10001 ttttt sssss fffff 000010
Floating point multiplication double precision : Set $f2 to double-precision floating point value of $f4 times $f6

mul.s $f0, $f1, $f3
010001 10000 ttttt sssss fffff 000010
Floating point multiplication single precision : Set $f0 to single-precision floating point value of $f1 times $f3

mult $t1, $t2
000000 fffff sssss 00000 00000 011000
Multiplication : Set hi to high-order 32 bits, lo to low-order 32 bits of the product of $t1 and $t2 (use mfhi to access hi, mflo to access lo)

multu $t1, $t2
000000 fffff sssss 00000 00000 011001
Multiplication unsigned : Set HI to high-order 32 bits, LO to low-order 32 bits of the product of unsigned $t1 and $t2 (use mfhi to access HI, mflo to access LO)

neg.d $f2, $f4
010001 10001 00000 sssss fffff 000111
Floating point negate double precision : Set double precision $f2 to negation of double precision value in $f4

neg.s $f0, $f1
010001 10000 00000 sssss fffff 000111
Floating point negate single precision : Set single precision $f0 to negation of single precision value in $f1

nop
000000 00000 00000 00000 00000 000000
Null operation : machine code is all zeroes

nor $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100111
Bitwise NOR : Set $t1 to bitwise NOR of $t2 and $t3

or $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100101
Bitwise OR : Set $t1 to bitwise OR of $t2 and $t3

round.w.d $f1, $f2
010001 10001 00000 sssss fffff 001100
Round double precision to word : Set $f1 to 32-bit integer round of double-precision float in $f2

round.w.s $f0, $f1
010001 10000 00000 sssss fffff 001100
Round single precision to word : Set $f0 to 32-bit integer round of single-precision float in $f1

sll $t1, $t2, 10
000000 00000 sssss fffff ttttt 000000
Shift left logical : Set $t1 to result of shifting $t2 left by number of bits specified by immediate

sllv $t1, $t2, $t3
000000 ttttt sssss fffff 00000 000100
Shift left logical variable : Set $t1 to result of shifting $t2 left by number of bits specified by value in low-order 5 bits of $t3

slt $t1, $t2, $t3
000000 sssss ttttt fffff 00000 101010
Set less than : If $t2 is less than $t3, then set $t1 to 1 else set $t1 to 0

sltu $t1, $t2, $t3
000000 sssss ttttt fffff 00000 101011
Set less than unsigned : If $t2 is less than $t3 using unsigned comparision, then set $t1 to 1 else set $t1 to 0

sqrt.d $f2, $f4
010001 10001 00000 sssss fffff 000100
Square root double precision : Set $f2 to double-precision floating point square root of $f4

sqrt.s $f0, $f1
010001 10000 00000 sssss fffff 000100
Square root single precision : Set $f0 to single-precision floating point square root of $f1

sra $t1, $t2, 10
000000 00000 sssss fffff ttttt 000011
Shift right arithmetic : Set $t1 to result of sign-extended shifting $t2 right by number of bits specified by immediate

srav $t1, $t2, $t3
000000 ttttt sssss fffff 00000 000111
Shift right arithmetic variable : Set $t1 to result of sign-extended shifting $t2 right by number of bits specified by value in low-order 5 bits of $t3

srl $t1, $t2, 10
000000 00000 sssss fffff ttttt 000010
Shift right logical : Set $t1 to result of shifting $t2 right by number of bits specified by immediate

srlv $t1, $t2, $t3
000000 ttttt sssss fffff 00000 000110
Shift right logical variable : Set $t1 to result of shifting $t2 right by number of bits specified by value in low-order 5 bits of $t3

sub $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100010
Subtraction with overflow : set $t1 to ($t2 minus $t3)

sub.d $f2, $f4, $f6
010001 10001 ttttt sssss fffff 000001
Floating point subtraction double precision : Set $f2 to double-precision floating point value of $f4 minus $f6

sub.s $f0, $f1, $f3
010001 10000 ttttt sssss fffff 000001
Floating point subtraction single precision : Set $f0 to single-precision floating point value of $f1 minus $f3

subu $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100011
Subtraction unsigned without overflow : set $t1 to ($t2 minus $t3), no overflow

syscall
000000 00000 00000 00000 00000 001100
Issue a system call : Execute the system call specified by value in $v0

teq $t1, $t2
000000 fffff sssss 00000 00000 110100
Trap if equal : Trap if $t1 is equal to $t2

tge $t1, $t2
000000 fffff sssss 00000 00000 110000
Trap if greater or equal : Trap if $t1 is greater than or equal to $t2

tgeu $t1, $t2
000000 fffff sssss 00000 00000 110001
Trap if greater or equal unsigned : Trap if $t1 is greater than or equal to $t2 using unsigned comparision

tlt $t1, $t2
000000 fffff sssss 00000 00000 110010
Trap if less than: Trap if $t1 less than $t2

tltu $t1, $t2
000000 fffff sssss 00000 00000 110011
Trap if less than unsigned : Trap if $t1 less than $t2, unsigned comparison

tne $t1, $t2
000000 fffff sssss 00000 00000 110110
Trap if not equal : Trap if $t1 is not equal to $t2

trunc.w.d $f1, $f2
010001 10001 00000 sssss fffff 001101
Truncate double precision to word : Set $f1 to 32-bit integer truncation of double-precision float in $f2

trunc.w.s $f0, $f1
010001 10000 00000 sssss fffff 001101
Truncate single precision to word : Set $f0 to 32-bit integer truncation of single-precision float in $f1

xor $t1, $t2, $t3
000000 sssss ttttt fffff 00000 100110
Bitwise XOR (exclusive OR) : Set $t1 to bitwise XOR of $t2 and $t3

I-Type

op	rs	rt	immediate
6-bit	5-bit	5-bit	16-bit

addi $t1, $t2, -100
001000 sssss fffff tttttttttttttttt
Addition immediate with overflow : set $t1 to ($t2 plus signed 16-bit immediate)

addiu $t1, $t2, -100
001001 sssss fffff tttttttttttttttt
Addition immediate unsigned without overflow : set $t1 to ($t2 plus signed 16-bit immediate), no overflow

andi $t1, $t2, 100
001100 sssss fffff tttttttttttttttt
Bitwise AND immediate : Set $t1 to bitwise AND of $t2 and zero-extended 16-bit immediate

bc1f 1, label
010001 01000 fff 00 ssssssssssssssss
Branch if specified FP condition flag false (BC1F, not BCLF) : If Coprocessor 1 condition flag specified by immediate is false (zero) then branch to statement at label's address

bc1f label
010001 01000 00000 ffffffffffffffff
Branch if FP condition flag 0 false (BC1F, not BCLF) : If Coprocessor 1 condition flag 0 is false (zero) then branch to statement at label's address

bc1t 1, label
010001 01000 fff 01 ssssssssssssssss
Branch if specified FP condition flag true (BC1T, not BCLT) : If Coprocessor 1 condition flag specified by immediate is true (one) then branch to statement at label's address

bc1t label
010001 01000 00001 ffffffffffffffff
Branch if FP condition flag 0 true (BC1T, not BCLT) : If Coprocessor 1 condition flag 0 is true (one) then branch to statement at label's address

beq $t1, $t2, label
000100 fffff sssss tttttttttttttttt
Branch if equal : Branch to statement at label's address if $t1 and $t2 are equal

bgez $t1, label
000001 fffff 00001 ssssssssssssssss
Branch if greater than or equal to zero : Branch to statement at label's address if $t1 is greater than or equal to zero

bgezal $t1, label
000001 fffff 10001 ssssssssssssssss
Branch if greater then or equal to zero and link : If $t1 is greater than or equal to zero, then set $ra to the Program Counter and branch to statement at label's address

bgtz $t1, label
000111 fffff 00000 ssssssssssssssss
Branch if greater than zero : Branch to statement at label's address if $t1 is greater than zero

blez $t1, label
000110 fffff 00000 ssssssssssssssss
Branch if less than or equal to zero : Branch to statement at label's address if $t1 is less than or equal to zero

bltz $t1, label
000001 fffff 00000 ssssssssssssssss
Branch if less than zero : Branch to statement at label's address if $t1 is less than zero

bltzal $t1, label
000001 fffff 10000 ssssssssssssssss
Branch if less than zero and link : If $t1 is less than or equal to zero, then set $ra to the Program Counter and branch to statement at label's address

bne $t1, $t2, label
000101 fffff sssss tttttttttttttttt
Branch if not equal : Branch to statement at label's address if $t1 and $t2 are not equal

lb $t1, -100($t2)
100000 ttttt fffff ssssssssssssssss
Load byte : Set $t1 to sign-extended 8-bit value from effective memory byte address

lbu $t1, -100($t2)
100100 ttttt fffff ssssssssssssssss
Load byte unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address

ldc1 $f2, -100($t2)
110101 ttttt fffff ssssssssssssssss
Load double word Coprocessor 1 (FPU)) : Set $f2 to 64-bit value from effective memory doubleword address

lh $t1, -100($t2)
100001 ttttt fffff ssssssssssssssss
Load halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address

lhu $t1, -100($t2)
100101 ttttt fffff ssssssssssssssss
Load halfword unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address

ll $t1, -100($t2)
110000 ttttt fffff ssssssssssssssss
Load linked : Paired with Store Conditional (sc) to perform atomic read-modify-write. Treated as equivalent to Load Word (lw) because MARS does not simulate multiple processors.

lui $t1, 100
001111 00000 fffff ssssssssssssssss
Load upper immediate : Set high-order 16 bits of $t1 to 16-bit immediate and low-order 16 bits to 0

lw $t1, -100($t2)
100011 ttttt fffff ssssssssssssssss
Load word : Set $t1 to contents of effective memory word address

lwc1 $f1, -100($t2)
110001 ttttt fffff ssssssssssssssss
Load word into Coprocessor 1 (FPU) : Set $f1 to 32-bit value from effective memory word address

lwl $t1, -100($t2)
100010 ttttt fffff ssssssssssssssss
Load word left : Load from 1 to 4 bytes left-justified into $t1, starting with effective memory byte address and continuing through the low-order byte of its word

lwr $t1, -100($t2)
100110 ttttt fffff ssssssssssssssss
Load word right : Load from 1 to 4 bytes right-justified into $t1, starting with effective memory byte address and continuing through the high-order byte of its word

ori $t1, $t2, 100
001101 sssss fffff tttttttttttttttt
Bitwise OR immediate : Set $t1 to bitwise OR of $t2 and zero-extended 16-bit immediate

sb $t1, -100($t2)
101000 ttttt fffff ssssssssssssssss
Store byte : Store the low-order 8 bits of $t1 into the effective memory byte address

sc $t1, -100($t2)
111000 ttttt fffff ssssssssssssssss
Store conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Stores $t1 value into effective address, then sets $t1 to 1 for success. Always succeeds because MARS does not simulate multiple processors.

sdc1 $f2, -100($t2)
111101 ttttt fffff ssssssssssssssss
Store double word from Coprocessor 1 (FPU)) : Store 64 bit value in $f2 to effective memory doubleword address

sh $t1, -100($t2)
101001 ttttt fffff ssssssssssssssss
Store halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

slti $t1, $t2, -100
001010 sssss fffff tttttttttttttttt
Set less than immediate : If $t2 is less than sign-extended 16-bit immediate, then set $t1 to 1 else set $t1 to 0

sltiu $t1, $t2, -100
001011 sssss fffff tttttttttttttttt
Set less than immediate unsigned : If $t2 is less than sign-extended 16-bit immediate using unsigned comparison, then set $t1 to 1 else set $t1 to 0

sw $t1, -100($t2)
101011 ttttt fffff ssssssssssssssss
Store word : Store contents of $t1 into effective memory word address

swc1 $f1, -100($t2)
111001 ttttt fffff ssssssssssssssss
Store word from Coprocesor 1 (FPU) : Store 32 bit value in $f1 to effective memory word address

swl $t1, -100($t2)
101010 ttttt fffff ssssssssssssssss
Store word left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective byte address and continuing through the low-order byte of its word

swr $t1, -100($t2)
101110 ttttt fffff ssssssssssssssss
Store word right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective byte address and continuing through that byte address

teqi $t1, -100
000001 fffff 01100 ssssssssssssssss
Trap if equal to immediate : Trap if $t1 is equal to sign-extended 16 bit immediate

tgei $t1, -100
000001 fffff 01000 ssssssssssssssss
Trap if greater than or equal to immediate : Trap if $t1 greater than or equal to sign-extended 16 bit immediate

tgeiu $t1, -100
000001 fffff 01001 ssssssssssssssss
Trap if greater or equal to immediate unsigned : Trap if $t1 greater than or equal to sign-extended 16 bit immediate, unsigned comparison

tlti $t1, -100
000001 fffff 01010 ssssssssssssssss
Trap if less than immediate : Trap if $t1 less than sign-extended 16-bit immediate

tltiu $t1, -100
000001 fffff 01011 ssssssssssssssss
Trap if less than immediate unsigned : Trap if $t1 less than sign-extended 16-bit immediate, unsigned comparison

tnei $t1, -100
000001 fffff 01110 ssssssssssssssss
Trap if not equal to immediate : Trap if $t1 is not equal to sign-extended 16 bit immediate

xori $t1, $t2, 100
001110 sssss fffff tttttttttttttttt
Bitwise XOR immediate : Set $t1 to bitwise XOR of $t2 and zero-extended 16-bit immediate

J-Type

op	address
6-bit	26-bit

j target
000010 ffffffffffffffffffffffff
Jump unconditionally : Jump to statement at target address

jal target
000011 ffffffffffffffffffffffff
Jump and link : Set $ra to Program Counter (return address) then jump to statement at target address

The information in this document has been extracted from the source code of the MARS 4.5 simulator.