Description – **Pseudo** – Real

A

ABSolute value : Set $t1 to absolute value of $t2 (algorithm from Hacker's Delight)
```
abs $t1, $t2
sra $1, RG2, 31
xor RG1, $1, RG2
subu RG1, RG1, $1
```

ADDition : set $t1 to ($t2 plus 16-bit immediate)
```
add $t1, $t2, -100
addi RG1, RG2, VL3
```

ADDition : set $t1 to ($t2 plus 32-bit immediate)
```
add $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
add RG1, RG2, $1
```

ADDition Immediate : set $t1 to ($t2 plus 32-bit immediate)
```
addi $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
add RG1, RG2, $1
```

ADDition Immediate Unsigned: set $t1 to ($t2 plus 32-bit immediate), no overflow
```
addiu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
addu RG1, RG2, $1
```

ADDition Unsigned : set $t1 to ($t2 plus 32-bit immediate), no overflow
```
addu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
addu RG1, RG2, $1
```

AND : set $t1 to ($t2 bitwise-AND 16-bit unsigned immediate)
```
and $t1, $t2, 100
andi RG1, RG2, VL3U
```

AND : set $t1 to ($t1 bitwise-AND 16-bit unsigned immediate)
```
and $t1, 100
andi RG1, RG1, VL2U
```

AND Immediate : set $t1 to ($t2 bitwise-AND 32-bit immediate)
```
andi $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
and RG1, RG2, $1
```

AND Immediate : set $t1 to ($t1 bitwise-AND 32-bit immediate)
```
andi $t1, 100000
lui $1, VHL2
ori $1, $1, VL2U
and RG1, RG1, $1
```

AND Immediate : set $t1 to ($t1 bitwise-AND 16-bit unsigned immediate)
```
andi $t1, 100
andi RG1, RG1, VL2U
```

B

Branch : Branch to statement at label unconditionally
```
b label
bgez $0, LAB
```

Branch if EQual : Branch to statement at label if $t1 is equal to 16-bit immediate
```
beq $t1, -100, label
addi $1, $0, VL2
beq $1, RG1, LAB
```

Branch if EQual : Branch to statement at label if $t1 is equal to 32-bit immediate
```
beq $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
beq $1, RG1, LAB
```

Branch if EQual Zero : Branch to statement at label if $t1 is equal to zero
```
beqz $t1, label
beq RG1, $0, LAB
```

Branch if Greater or Equal : Branch to statement at label if $t1 is greater or equal to $t2
```
bge $t1, $t2, label
slt $1, RG1, RG2
beq $1, $0, LAB
```

Branch if Greater or Equal : Branch to statement at label if $t1 is greater or equal to 16-bit immediate
```
bge $t1, -100, label
slti $1, RG1, VL2
beq $1, $0, LAB
```

Branch if Greater or Equal : Branch to statement at label if $t1 is greater or equal to 32-bit immediate
```
bge $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
slt $1, RG1, $1
beq $1, $0, LAB
```

Branch if Greater or Equal Unsigned : Branch to statement at label if $t1 is greater or equal to $t2 (unsigned compare)
```
bgeu $t1, $t2, label
sltu $1, RG1, RG2
beq $1, $0, LAB
```

Branch if Greater or Equal Unsigned : Branch to statement at label if $t1 is greater or equal to 16-bit immediate (unsigned compare)
```
bgeu $t1, -100, label
sltiu $1, RG1, VL2
beq $1, $0, LAB
```

Branch if Greater or Equal Unsigned : Branch to statement at label if $t1 is greater or equal to 32-bit immediate (unsigned compare)
```
bgeu $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
sltu $1, RG1, $1
beq $1, $0, LAB
```

Branch if Greater Than : Branch to statement at label if $t1 is greater than $t2
```
bgt $t1, $t2, label
slt $1, RG2, RG1
bne $1, $0, LAB
```

Branch if Greater Than : Branch to statement at label if $t1 is greater than 16-bit immediate
```
bgt $t1, -100, label
addi $1, $0, VL2
slt $1, $1, RG1
bne $1, $0, LAB
```

Branch if Greater Than : Branch to statement at label if $t1 is greater than 32-bit immediate
```
bgt $t1, 100000, label
lui $1, VHL2P1
ori $1, $1, VL2P1U
slt $1, RG1, $1
beq $1, $0, LAB
```

Branch if Greater Than Unsigned: Branch to statement at label if $t1 is greater than $t2 (unsigned compare)
```
bgtu $t1, $t2, label
sltu $1, RG2, RG1
bne $1, $0, LAB
```

Branch if Greater Than Unsigned: Branch to statement at label if $t1 is greater than 16-bit immediate (unsigned compare)
```
bgtu $t1, -100, label
addi $1, $0, VL2
sltu $1, $1, RG1
bne $1, $0, LAB
```

Branch if Greater Than Unsigned: Branch to statement at label if $t1 is greater than 16-bit immediate (unsigned compare)
```
bgtu $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
sltu $1, $1, RG1
bne $1, $0, LAB
```

Branch if Less or Equal : Branch to statement at label if $t1 is less than or equal to $t2
```
ble $t1, $t2, label
slt $1, RG2, RG1
beq $1, $0, LAB
```

Branch if Less or Equal : Branch to statement at label if $t1 is less than or equal to 16-bit immediate
```
ble $t1, -100, label
addi $1, RG1, -1
slti $1, $1, VL2
bne $1, $0, LAB
```

Branch if Less or Equal : Branch to statement at label if $t1 is less than or equal to 32-bit immediate
```
ble $t1, 100000, label
lui $1, VHL2P1
ori $1, $1, VL2P1U
slt $1, RG1, $1
bne $1, $0, LAB
```

Branch if Less or Equal Unsigned : Branch to statement at label if $t1 is less than or equal to $t2 (unsigned compare)
```
bleu $t1, $t2, label
sltu $1, RG2, RG1
beq $1, $0, LAB
```

Branch if Less or Equal Unsigned : Branch to statement at label if $t1 is less than or equal to 16-bit immediate (unsigned compare)
```
bleu $t1, -100, label
addi $1, $0, VL2
sltu $1, $1, RG1
beq $1, $0, LAB
```

Branch if Less or Equal Unsigned : Branch to statement at label if $t1 is less than or equal to 32-bit immediate (unsigned compare)
```
bleu $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
sltu $1, $1, RG1
beq $1, $0, LAB
```

Branch if Less Than : Branch to statement at label if $t1 is less than $t2
```
blt $t1, $t2, label
slt $1, RG1, RG2
bne $1, $0, LAB
```

Branch if Less Than : Branch to statement at label if $t1 is less than 16-bit immediate
```
blt $t1, -100, label
slti $1, RG1, VL2
bne $1, $0, LAB
```

Branch if Less Than : Branch to statement at label if $t1 is less than 32-bit immediate
```
blt $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
slt $1, RG1, $1
bne $1, $0, LAB
```

Branch if Less Than Unsigned : Branch to statement at label if $t1 is less than $t2
```
bltu $t1, $t2, label
sltu $1, RG1, RG2
bne $1, $0, LAB
```

Branch if Less Than Unsigned : Branch to statement at label if $t1 is less than 16-bit immediate
```
bltu $t1, -100, label
sltiu $1, RG1, VL2
bne $1, $0, LAB
```

Branch if Less Than Unsigned : Branch to statement at label if $t1 is less than 32-bit immediate
```
bltu $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
sltu $1, RG1, $1
bne $1, $0, LAB
```

Branch if Not Equal : Branch to statement at label if $t1 is not equal to 16-bit immediate
```
bne $t1, -100, label
addi $1, $0, VL2
bne $1, RG1, LAB
```

Branch if Not Equal : Branch to statement at label if $t1 is not equal to 32-bit immediate
```
bne $t1, 100000, label
lui $1, VHL2
ori $1, $1, VL2U
bne $1, RG1, LAB
```

Branch if Not Equal Zero : Branch to statement at label if $t1 is not equal to zero
```
bnez $t1, label
bne RG1, $0, LAB
```

D

DIVision : Set $t1 to ($t2 divided by $t3, integer division)
```
div $t1, $t2, $t3
bne RG3, $0, BROFF12
DBNOP
break
div RG2, RG3
mflo RG1
```

DIVision : Set $t1 to ($t2 divided by 16-bit immediate, integer division)
```
div $t1, $t2, -100
addi $1, $0, VL3
div RG2, $1
mflo RG1
```

DIVision : Set $t1 to ($t2 divided by 32-bit immediate, integer division)
```
div $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
div RG2, $1
mflo RG1
```

DIVision Unsigned : Set $t1 to ($t2 divided by $t3, unsigned integer division)
```
divu $t1, $t2, $t3
bne RG3, $0, BROFF12
DBNOP
break
divu RG2, RG3
mflo RG1
```

DIVision Unsigned : Set $t1 to ($t2 divided by 16-bit immediate, unsigned integer division)
```
divu $t1, $t2, -100
addi $1, $0, VL3
divu RG2, $1
mflo RG1
```

DIVision Unsigned : Set $t1 to ($t2 divided by 32-bit immediate, unsigned integer division)
```
divu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
divu RG2, $1
mflo RG1
```

L

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, ($t2)
ldc1 RG1,0(RG3)
```

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, -100
ldc1 RG1,VL2($0)
```

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
ldc1 RG1, VL2($1)
```

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, 100000
lui $1, VH2
ldc1 RG1,VL2($1)
```

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, label($t2)
lui $1, LH2
addu $1, $1, RG4
ldc1 RG1, LL2($1)
COMPACT
ldc1 RG1, LL2(RG4)
```

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
ldc1 RG1, LLP($1)
```

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, label+100000
lui $1, LHPA
ldc1 RG1, LLP($1)
```

Load floating point Double precision : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
l.d $f2, label
lui $1, LH2
ldc1 RG1, LL2($1)
COMPACT
ldc1 RG1, LL2($0)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, ($t2)
lwc1 RG1,0(RG3)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, -100
lwc1 RG1,VL2($0)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lwc1 RG1, VL2($1)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, 100000
lui $1, VH2
lwc1 RG1,VL2($1)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lwc1 RG1, LL2($1)
COMPACT
lwc1 RG1, LL2(RG4)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lwc1 RG1, LLP($1)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, label+100000
lui $1, LHPA
lwc1 RG1, LLP($1)
```

Load floating point Single precision : Set $f1 to 32-bit value at effective memory word address
```
l.s $f1, label
lui $1, LH2
lwc1 RG1, LL2($1)
COMPACT
lwc1 RG1, LL2($0)
```

Load Address : Set $t1 to contents of $t2
```
la $t1, ($t2)
addi RG1, RG3, 0
```

Load Address : Set $t1 to 16-bit immediate (sign-extended)
```
la $t1, -100
addiu RG1, $0, VL2
```

Load Address : Set $t1 to sum (of $t2 and 16-bit immediate)
```
la $t1, 100($t2)
ori $1, $0, VL2U
add RG1, RG4, $1
```

Load Address : Set $t1 to sum (of $t2 and 32-bit immediate)
```
la $t1, 100000($t2)
lui $1, VHL2
ori $1, $1, VL2U
add RG1, RG4, $1
```

Load Address : Set $t1 to 32-bit immediate
```
la $t1, 100000
lui $1, VHL2
ori RG1, $1, VL2U
```

Load Address : Set $t1 to 16-bit immediate (zero-extended)
```
la $t1, 100
ori RG1, $0, VL2U
```

Load Address : Set $t1 to sum (of $t2 and label's address)
```
la $t1, label($t2)
lui $1, LHL
ori $1, $1, LL2U
add RG1, RG4, $1
COMPACT
addi RG1, RG4, LL2
```

Load Address : Set $t1 to sum (of label's address, 32-bit immediate, and $t2)
```
la $t1, label+100000($t2)
lui $1, LHPN
ori $1, $1, LLPU
add RG1, RG6, $1
```

Load Address : Set $t1 to sum (of label's address and 32-bit immediate)
```
la $t1, label+100000
lui $1, LHPN
ori RG1, $1, LLPU
```

Load Address : Set $t1 to label's address
```
la $t1, label
lui $1, LHL
ori RG1, $1, LL2U
COMPACT
addi RG1, $0, LL2
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, ($t2)
lb RG1,0(RG3)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, -100
lb RG1, VL2($0)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
lb RG1, 0($1)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lb RG1, VL2($1)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, 100000
lui $1, VH2
lb RG1,VL2($1)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, 100
ori $1, $0, VL2U
lb RG1, 0($1)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lb RG1, LL2($1)
COMPACT
lb RG1, LL2(RG4)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lb RG1, LLP($1)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, label+100000
lui $1, LHPA
lb RG1, LLP($1)
```

Load Byte : Set $t1 to sign-extended 8-bit value from effective memory byte address
```
lb $t1, label
lui $1, LH2
lb RG1, LL2($1)
COMPACT
lb RG1, LL2($0)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, ($t2)
lbu RG1,0(RG3)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, -100
lbu RG1,VL2($0)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
lbu RG1, 0($1)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lbu RG1, VL2($1)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, 100000
lui $1, VH2
lbu RG1,VL2($1)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, 100
ori $1, $0, VL2U
lbu RG1, 0($1)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lbu RG1, LL2($1)
COMPACT
lbu RG1, LL2(RG4)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lbu RG1, LLP($1)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, label+100000
lui $1, LHPA
lbu RG1, LLP($1)
```

Load Byte Unsigned : Set $t1 to zero-extended 8-bit value from effective memory byte address
```
lbu $t1, label
lui $1, LH2
lbu RG1, LL2($1)
COMPACT
lbu RG1, LL2($0)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory word address
```
ld $t1, ($t2)
lw RG1, 0(RG3)
lw NR1, 4(RG3)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory byte address
```
ld $t1, -100($t2)
lw RG1, VL2(RG4)
lui $1, VH2P4
addu $1, $1, RG4
lw NR1, VL2P4($1)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory word address
```
ld $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lw RG1, VL2($1)
lui $1, VH2P4
addu $1, $1, RG4
lw NR1, VL2P4($1)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory word address
```
ld $t1, 100000
lui $1, VH2
lw RG1, VL2($1)
lui $1, VH2P4
lw NR1, VL2P4($1)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory word address
```
ld $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lw RG1, LL2($1)
lui $1, LH2P4
addu $1, $1, RG4
lw NR1, LL2P4($1)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory word address
```
ld $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lw RG1, LLP($1)
lui $1, LHPAP4
addu $1, $1, RG6
lw NR1, LLPP4($1)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory word address
```
ld $t1, label+100000
lui $1, LHPA
lw RG1, LLP($1)
lui $1, LHPAP4
lw NR1, LLPP4($1)
```

Load Doubleword : Set $t1 and the next register to the 64 bits starting at effective memory word address
```
ld $t1, label
lui $1, LH2
lw RG1, LL2($1)
lui $1, LH2P4
lw NR1, LL2P4($1)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2, ($t2)
ldc1 RG1,0(RG3)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2, -100
ldc1 RG1,VL2($0)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
ldc1 RG1, VL2($1)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2, 100000
lui $1, VH2
ldc1 RG1,VL2($1)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2, label($t2)
lui $1, LH2
addu $1, $1, RG4
ldc1 RG1, LL2($1)
COMPACT
ldc1 RG1, LL2(RG4)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2,
label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
ldc1 RG1, LLP($1)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2, label+100000
lui $1, LHPA
ldc1 RG1, LLP($1)
```

Load Doubleword Coprocessor 1 : Set $f2 and $f3 register pair to 64-bit value at effective memory doubleword address
```
ldc1 $f2, label
lui $1, LH2
ldc1 RG1, LL2($1)
COMPACT
ldc1 RG1, LL2($0)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, ($t2)
lh RG1,0(RG3)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, -100
lh RG1, VL2($0)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
lh RG1, 0($1)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lh RG1, VL2($1)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, 100000
lui $1, VH2
lh RG1,VL2($1)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, 100
ori $1, $0, VL2U
lh RG1, 0($1)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lh RG1, LL2($1)
COMPACT
lh RG1, LL2(RG4)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lh RG1, LLP($1)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, label+100000
lui $1, LHPA
lh RG1, LLP($1)
```

Load Halfword : Set $t1 to sign-extended 16-bit value from effective memory halfword address
```
lh $t1, label
lui $1, LH2
lh RG1, LL2($1)
COMPACT
lh RG1, LL2($0)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, ($t2)
lhu RG1,0(RG3)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, -100
lhu RG1,VL2($0)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
lhu RG1, 0($1)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lhu RG1, VL2($1)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, 100000
lui $1, VH2
lhu RG1,VL2($1)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, 100
ori $1, $0, VL2U
lhu RG1, 0($1)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lhu RG1, LL2($1)
COMPACT
lhu RG1, LL2(RG4)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lhu RG1, LLP($1)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, label+100000
lui $1, LHPA
lhu RG1, LLP($1)
```

Load Halfword Unsigned : Set $t1 to zero-extended 16-bit value from effective memory halfword address
```
lhu $t1, label
lui $1, LH2
lhu RG1, LL2($1)
COMPACT
lhu RG1, LL2($0)
```

Load Immediate : Set $t1 to 16-bit immediate (sign-extended)
```
li $t1, -100
addiu RG1, $0, VL2
```

Load Immediate : Set $t1 to 32-bit immediate
```
li $t1, 100000
lui $1, VHL2
ori RG1, $1, VL2U
```

Load Immediate : Set $t1 to unsigned 16-bit immediate (zero-extended)
```
li $t1, 100
ori RG1, $0, VL2U
```

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.
```
ll $t1, ($t2)
ll RG1,0(RG3)
```

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.

```
ll $t1, -100
ll RG1,VL2($0)
```

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.

```
ll $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
ll RG1, 0($1)
```

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.

```
ll $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
ll RG1, VL2($1)
```

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.

```
ll $t1, 100000
lui $1, VH2
ll RG1,VL2($1)
```

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.

```
ll $t1, 100
ori $1, $0, VL2U
ll RG1, 0($1)
```

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.

```
ll $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
ll RG1, LL2($1)
COMPACT
ll RG1, LL2(RG4)
```

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.

```
ll $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
ll RG1, LLP($1)
```

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.

```
ll $t1, label+100000
lui $1, LHPA
ll RG1, LLP($1)
```

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.

```
ll $t1, label
lui $1, LH2
ll RG1, LL2($1)
COMPACT
ll RG1, LL2($0)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, ($t2)
lw RG1,0(RG3)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, -100
lw RG1, VL2($0)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
lw RG1, 0($1)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lw RG1, VL2($1)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, 100000
lui $1, VH2
lw RG1,VL2($1)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, 100
ori $1, $0, VL2U
lw RG1, 0($1)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lw RG1, LL2($1)
COMPACT
lw RG1, LL2(RG4)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lw RG1, LLP($1)
```

Load Word : Set $t1 to contents of effective memory word address

```
lw $t1, label+100000
lui $1, LHPA
lw RG1, LLP($1)
```

Load Word : Set $t1 to contents of memory word at label's address

```
lw $t1, label
lui $1, LH2
lw RG1, LL2($1)
COMPACT
lw RG1, LL2($0)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1, ($t2)
lwc1 RG1,0(RG3)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1, -100
lwc1 RG1,VL2($0)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lwc1 RG1, VL2($1)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1, 100000
lui $1, VH2
lwc1 RG1,VL2($1)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lwc1 RG1, LL2($1)
COMPACT
lwc1 RG1, LL2(RG4)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1,label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lwc1 RG1, LLP($1)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1, label+100000
lui $1, LHPA
lwc1 RG1, LLP($1)
```

Load Word Coprocessor 1 : Set $f1 to 32-bit value from effective memory word address

```
lwc1 $f1, label
lui $1, LH2
lwc1 RG1, LL2($1)
COMPACT
lwc1 RG1, LL2($0)
```

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

```
lwl $t1, ($t2)
lwl RG1,0(RG3)
```

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

```
lwl $t1, -100
lwl RG1,VL2($0)
```

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

```
lwl $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
lwl RG1, 0($1)
```

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

```
lwl $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lwl RG1, VL2($1)
```

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

```
lwl $t1, 100000
lui $1, VH2
lwl RG1,VL2($1)
```

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

```
lwl $t1, 100
ori $1, $0, VL2U
lwl RG1, 0($1)
```

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

```
lwl $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lwl RG1, LL2($1)
COMPACT
lwl RG1, LL2(RG4)
```

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

```
lwl $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lwl RG1, LLP($1)
```

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

```
lwl $t1, label+100000
lui $1, LHPA
lwl RG1, LLP($1)
```

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

```
lwl $t1, label
lui $1, LH2
lwl RG1, LL2($1)
COMPACT
lwl RG1, LL2($0)
```

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

```
lwr $t1, ($t2)
lwr RG1,0(RG3)
```

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

```
lwr $t1, -100
lwr RG1,VL2($0)
```

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

```
lwr $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
lwr RG1, 0($1)
```

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

```
lwr $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
lwr RG1, VL2($1)
```

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

```
lwr $t1, 100000
lui $1, VH2
lwr RG1,VL2($1)
```

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

```
lwr $t1, 100
ori $1, $0, VL2U
lwr RG1, 0($1)
```

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

```
lwr $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
lwr RG1, LL2($1)
COMPACT
lwr RG1, LL2(RG4)
```

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

```
lwr $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
lwr RG1, LLP($1)
```

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

```
lwr $t1, label+100000
lui $1, LHPA
lwr RG1, LLP($1)
```

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

```
lwr $t1, label
lui $1, LH2
lwr RG1, LL2($1)
COMPACT
lwr RG1, LL2($0)
```

M

Move From Coprocessor 1 Double : Set $t1 to contents of $f2, set next higher register from $t1 to contents of next higher register from $f2

```
mfc1.d $t1, $f2
mfc1 RG1, RG2
mfc1 NR1, NR2
```

MOVE : Set $t1 to contents of $t2

```
move $t1, $t2
addu RG1, $0, RG2
```

Move To Coprocessor 1 Double : Set $f2 to contents of $t1, set next higher register from $f2 to contents of next higher register from $t1

```
mtc1.d $t1, $f2
mtc1 RG1, RG2
mtc1 NR1, NR2
```

MULtiplication : Set HI to high-order 32 bits, LO and $t1 to low-order 32 bits of the product of $t2 and 16-bit signed immediate (use mfhi to access HI, mflo to access LO)

```
mul $t1, $t2, -100
addi $1, $0, VL3
mul RG1, RG2, $1
```

MULtiplication : Set HI to high-order 32 bits, LO and $t1 to low-order 32 bits of the product of $t2 and 32-bit immediate (use mfhi to access HI, mflo to access LO)

```
mul $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
mul RG1, RG2, $1
```

MULtiplication with Overflow : Set $t1 to low-order 32 bits of the product of $t2 and $t3

```
mulo $t1, $t2, $t3
mult RG2, RG3
mfhi $1
mflo RG1
sra RG1, RG1, 31
beq $1, RG1, BROFF12
DBNOP
break
mflo RG1
```

MULtiplication with Overflow : Set $t1 to low-order 32 bits of the product of $t2 and signed 16-bit immediate

```
mulo $t1, $t2, -100
addi $1, $0, VL3
mult RG2, $1
mfhi $1
mflo RG1
sra RG1, RG1, 31
beq $1, RG1, BROFF12
DBNOP
break
mflo RG1
```

MULtiplication with Overflow : Set $t1 to low-order 32 bits of the product of $t2 and 32-bit immediate

```
mulo $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
mult RG2, $1
mfhi $1
mflo RG1
sra RG1, RG1, 31
beq $1, RG1, BROFF12
DBNOP
break
mflo RG1
```

MULtiplication with Overflow Unsigned : Set $t1 to low-order 32 bits of the product of $t2 and $t3

```
mulou $t1, $t2, $t3
multu RG2, RG3
mfhi $1
beq $1,$0, BROFF12
DBNOP
break
mflo RG1
```

MULtiplication with Overflow Unsigned : Set $t1 to low-order 32 bits of the product of $t2 and signed 16-bit immediate

```
mulou $t1, $t2, -100
addi $1, $0, VL3
multu RG2, $1
mfhi $1
beq $1,$0, BROFF12
DBNOP
break
mflo RG1
```

MULtiplication with Overflow Unsigned : Set $t1 to low-order 32 bits of the product of $t2 and 32-bit immediate

```
mulou $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
multu RG2, $1
mfhi $1
beq $1,$0, BROFF12
DBNOP
break
mflo RG1
```

MULtiplication Unsigned : Set HI to high-order 32 bits, LO and $t1 to low-order 32 bits of ($t2 multiplied by $t3, unsigned multiplication)

```
mulu $t1, $t2, $t3
multu RG2, RG3
mflo RG1
```

MULtiplication Unsigned : Set HI to high-order 32 bits, LO and $t1 to low-order 32 bits of ($t2 multiplied by 16-bit immediate, unsigned multiplication)

```
mulu $t1, $t2, -100
addi $1, $0, VL3
multu RG2, $1
mflo RG1
```

MULtiplication Unsigned : Set HI to high-order 32 bits, LO and $t1 to low-order 32 bits of ($t2 multiplied by 32-bit immediate, unsigned multiplication)

```
mulu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
multu RG2, $1
mflo RG1
```

N

NEGate : Set $t1 to negation of $t2

```
neg $t1, $t2
sub RG1, $0, RG2
```

NEGate Unsigned : Set $t1 to negation of $t2, no overflow

```
negu $t1, $t2
subu RG1, $0, RG2
```

Bitwise NOT (bit inversion)

```
not $t1, $t2
nor RG1, RG2, $0
```

O

OR : set $t1 to ($t2 bitwise-OR 16-bit unsigned immediate)

```
or $t1, $t2, 100
ori RG1, RG2, VL3U
```

OR : set $t1 to ($t1 bitwise-OR 16-bit unsigned immediate)

```
or $t1, 100
ori RG1, RG1, VL2U
```

OR Immediate : set $t1 to ($t2 bitwise-OR 32-bit immediate)

```
ori $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
or RG1, RG2, $1
```

OR Immediate : set $t1 to ($t1 bitwise-OR 32-bit immediate)

```
ori $t1, 100000
lui $1, VHL2
ori $1, $1, VL2U
or RG1, RG1, $1
```

OR Immediate : set $t1 to ($t1 bitwise-OR 16-bit unsigned immediate)

```
ori $t1, 100
ori RG1, RG1, VL2U
```

R

REMainder : Set $t1 to (remainder of $t2 divided by $t3)

```
rem $t1, $t2, $t3
bne RG3, $0, BROFF12
DBNOP
break
div RG2, RG3
mfhi RG1
```

REMainder : Set $t1 to (remainder of $t2 divided by 16-bit immediate)

```
rem $t1, $t2, -100
addi $1, $0, VL3
div RG2, $1
mfhi RG1
```

REMainder : Set $t1 to (remainder of $t2 divided by 32-bit immediate)

```
rem $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
div RG2, $1
mfhi RG1
```

REMainder : Set $t1 to (remainder of $t2 divided by $t3, unsigned division)

```
remu $t1, $t2, $t3
bne RG3, $0, BROFF12
DBNOP
break
divu RG2, RG3
mfhi RG1
```

REMainder : Set $t1 to (remainder of $t2 divided by 16-bit immediate, unsigned division)

```
remu $t1, $t2, -100
addi $1, $0, VL3
divu RG2, $1
mfhi RG1
```

REMainder : Set $t1 to (remainder of $t2 divided by 32-bit immediate, unsigned division)

```
remu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
divu RG2, $1
mfhi RG1
```

ROtate Left : Set $t1 to ($t2 rotated left by number of bit positions specified in $t3)

```
rol $t1, $t2, $t3
subu $1, $0, RG3
srlv $1, RG2, $1
sllv RG1, RG2, RG3
or RG1, RG1, $1
```

ROtate Left : Set $t1 to ($t2 rotated left by number of bit positions specified in 5-bit immediate)

```
rol $t1, $t2, 10
srl $1, RG2, S32
sll RG1, RG2, OP3
or RG1, RG1, $1
```

ROtate Right : Set $t1 to ($t2 rotated right by number of bit positions specified in $t3)

```
ror $t1, $t2, $t3
subu $1, $0, RG3
sllv $1, RG2, $1
srlv RG1, RG2, RG3
or RG1, RG1, $1
```

ROtate Right : Set $t1 to ($t2 rotated right by number of bit positions specified in 5-bit immediate)

```
ror $t1, $t2, 10
sll $1, RG2, S32
srl RG1, RG2, OP3
or RG1, RG1, $1
```

S

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, ($t2)
sdc1 RG1,0(RG3)
```

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, -100
sdc1 RG1,VL2($0)
```

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sdc1 RG1, VL2($1)
```

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, 100000
lui $1, VH2
sdc1 RG1,VL2($1)
```

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, label($t2)
lui $1, LH2
addu $1, $1, RG4
sdc1 RG1, LL2($1)
COMPACT
sdc1 RG1, LL2(RG4)
```

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, label+100000
lui $1, LHPA
sdc1 RG1, LLP($1)
```

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sdc1 RG1, LLP($1)
```

Store floating point Double precision : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address
```
s.d $f2, label
lui $1, LH2
sdc1 RG1, LL2($1)
COMPACT
sdc1 RG1, LL2($0)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, ($t2)
swc1 RG1,0(RG3)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, -100
swc1 RG1,VL2($0)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
swc1 RG1, VL2($1)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, 100000
lui $1, VH2
swc1 RG1,VL2($1)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, label($t2)
lui $1, LH2
addu $1, $1, RG4
swc1 RG1, LL2($1)
COMPACT
swc1 RG1, LL2(RG4)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
swc1 RG1, LLP($1)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, label+100000
lui $1, LHPA
swc1 RG1, LLP($1)
```

Store floating point Single precision : Store 32-bit value from $f1 to effective memory word address
```
s.s $f1, label
lui $1, LH2
swc1 RG1, LL2($1)
COMPACT
swc1 RG1, LL2($0)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, ($t2)
sb RG1,0(RG3)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, -100
sb RG1, VL2($0)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
sb RG1, 0($1)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sb RG1, VL2($1)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, 100000
lui $1, VH2
sb RG1,VL2($1)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, 100
ori $1, $0, VL2U
sb RG1, 0($1)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
sb RG1, LL2($1)
COMPACT
sb RG1, LL2(RG4)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sb RG1, LLP($1)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, label+100000
lui $1, LHPA
sb RG1, LLP($1)
```

Store Byte : Store the low-order 8 bits of $t1 into the effective memory byte address
```
sb $t1, label
lui $1, LH2
sb RG1, LL2($1)
COMPACT
sb RG1, LL2($0)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, ($t2)
sc RG1,0(RG3)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, -100
sc RG1,VL2($0)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
sc RG1, 0($1)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sc RG1, VL2($1)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, 100000
lui $1, VH2
sc RG1,VL2($1)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, 100
ori $1, $0, VL2U
sc RG1, 0($1)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
sc RG1, LL2($1)
COMPACT
sc RG1, LL2(RG4)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sc RG1, LLP($1)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, label+100000
lui $1, LHPA
sc RG1, LLP($1)
```

Store Conditional : Paired with Load Linked (ll) to perform atomic read-modify-write. Treated as equivalent to Store Word (sw) because MARS does not simulate multiple processors.
```
sc $t1, label
lui $1, LH2
sc RG1, LL2($1)
COMPACT
sc RG1, LL2($0)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory word address
```
sd $t1, ($t2)
sw RG1, 0(RG3)
sw NR1, 4(RG3)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory byte address
```
sd $t1, -100($t2)
sw RG1, VL2(RG4)
lui $1, VH2P4
addu $1, $1, RG4
sw NR1, VL2P4($1)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory word address
```
sd $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sw RG1, VL2($1)
lui $1, VH2P4
addu $1, $1, RG4
sw NR1, VL2P4($1)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory word address
```
sd $t1, 100000
lui $1, VH2
sw RG1, VL2($1)
lui $1, VH2P4
sw NR1, VL2P4($1)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory word address
```
sd $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
sw RG1, LL2($1)
lui $1, LH2P4
addu $1, $1, RG4
sw NR1, LL2P4($1)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory word address

```
sd $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sw RG1, LLP($1)
lui $1, LHPAP4
addu $1, $1, RG6
sw NR1, LLPP4($1)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory word address

```
sd $t1, label+100000
lui $1, LHPA
sw RG1, LLP($1)
lui $1, LHPAP4
sw NR1, LLPP4($1)
```

Store Doubleword : Store contents of $t1 and the next register to the 64 bits starting at effective memory word address

```
sd $t1, label
lui $1, LH2
sw RG1, LL2($1)
lui $1, LH2P4
sw NR1, LL2P4($1)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2, ($t2)
sdc1 RG1,0(RG3)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2, -100
sdc1 RG1,VL2($0)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sdc1 RG1, VL2($1)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2, 100000
lui $1, VH2
sdc1 RG1,VL2($1)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2, label($t2)
lui $1, LH2
addu $1, $1, RG4
sdc1 RG1, LL2($1)
COMPACT
sdc1 RG1, LL2(RG4)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2,
label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sdc1 RG1, LLP($1)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2, label+100000
lui $1, LHPA
sdc1 RG1, LLP($1)
```

Store Doubleword Coprocessor 1 : Store 64 bits from $f2 and $f3 register pair to effective memory doubleword address

```
sdc1 $f2, label
lui $1, LH2
sdc1 RG1, LL2($1)
COMPACT
sdc1 RG1, LL2($0)
```

Set EQual : if $t2 equal to $t3 then set $t1 to 1 else 0

```
seq $t1, $t2, $t3
subu RG1, RG2, RG3
ori $1, $0, 1
sltu RG1, RG1, $1
```

Set EQual : if $t2 equal to 16-bit immediate then set $t1 to 1 else 0

```
seq $t1, $t2, -100
addi $1, $0, VL3
subu RG1, RG2, $1
ori $1, $0, 1
sltu RG1, RG1, $1
```

Set EQual : if $t2 equal to 32-bit immediate then set $t1 to 1 else 0

```
seq $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
subu RG1, RG2, $1
ori $1, $0, 1
sltu RG1, RG1, $1
```

Set Greater or Equal : if $t2 greater or equal to $t3 then set $t1 to 1 else 0

```
sge $t1, $t2, $t3
slt RG1, RG2, RG3
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Greater or Equal : if $t2 greater or equal to 16-bit immediate then set $t1 to 1 else 0

```
sge $t1, $t2, -100
addi $1, $0, VL3
slt RG1, RG2, $1
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Greater or Equal : if $t2 greater or equal to 32-bit immediate then set $t1 to 1 else 0

```
sge $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
slt RG1, RG2, $1
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Greater or Equal Unsigned : if $t2 greater or equal to $t3 (unsigned compare) then set $t1 to 1 else 0

```
sgeu $t1, $t2, $t3
sltu RG1, RG2, RG3
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Greater or Equal Unsigned : if $t2 greater or equal to 16-bit immediate (unsigned compare) then set $t1 to 1 else 0

```
sgeu $t1, $t2, -100
addi $1, $0, VL3
sltu RG1, RG2, $1
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Greater or Equal Unsigned : if $t2 greater or equal to 32-bit immediate (unsigned compare) then set $t1 to 1 else 0

```
sgeu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
sltu RG1, RG2, $1
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Greater Than : if $t2 greater than $t3 then set $t1 to 1 else 0

```
sgt $t1, $t2, $t3
slt RG1, RG3, RG2
```

Set Greater Than : if $t2 greater than 16-bit immediate then set $t1 to 1 else 0

```
sgt $t1, $t2, -100
addi $1, $0, VL3
slt RG1, $1, RG2
```

Set Greater Than : if $t2 greater than 32-bit immediate then set $t1 to 1 else 0

```
sgt $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
slt RG1, $1, RG2
```

Set Greater Than Unsigned : if $t2 greater than $t3 (unsigned compare) then set $t1 to 1 else 0

```
sgtu $t1, $t2, $t3
sltu RG1, RG3, RG2
```

Set Greater Than Unsigned : if $t2 greater than 16-bit immediate (unsigned compare) then set $t1 to 1 else 0

```
sgtu $t1, $t2, -100
addi $1, $0, VL3
sltu RG1, $1, RG2
```

Set Greater Than Unsigned : if $t2 greater than 32-bit immediate (unsigned compare) then set $t1 to 1 else 0

```
sgtu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
sltu RG1, $1, RG2
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, ($t2)
sh RG1,0(RG3)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, -100
sh RG1, VL2($0)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
sh RG1, 0($1)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sh RG1, VL2($1)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, 100000
lui $1, VH2
sh RG1,VL2($1)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, 100
ori $1, $0, VL2U
sh RG1, 0($1)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
sh RG1, LL2($1)
COMPACT
sh RG1, LL2(RG4)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sh RG1, LLP($1)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, label+100000
lui $1, LHPA
sh RG1, LLP($1)
```

Store Halfword : Store the low-order 16 bits of $t1 into the effective memory halfword address

```
sh $t1, label
lui $1, LH2
sh RG1, LL2($1)
COMPACT
sh RG1, LL2($0)
```

Set Less or Equal : if $t2 less or equal to $t3 then set $t1 to 1 else 0

```
sle $t1, $t2, $t3
slt RG1, RG3, RG2
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Less or Equal : if $t2 less or equal to 16-bit immediate then set $t1 to 1 else 0

```
sle $t1, $t2, -100
addi $1, $0, VL3
slt RG1, $1, RG2
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Less or Equal : if $t2 less or equal to 32-bit immediate then set $t1 to 1 else 0

```
sle $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
slt RG1, $1, RG2
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Less or Equal Unsigned: if $t2 less or equal to $t3 (unsigned compare) then set $t1 to 1 else 0

```
sleu $t1, $t2, $t3
sltu RG1, RG3, RG2
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Less or Equal Unsigned: if $t2 less or equal to 16-bit immediate (unsigned compare) then set $t1 to 1 else 0

```
sleu $t1, $t2, -100
addi $1, $0, VL3
sltu RG1, $1, RG2
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Less or Equal Unsigned: if $t2 less or equal to 32-bit immediate (unsigned compare) then set $t1 to 1 else 0

```
sleu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
sltu RG1, $1, RG2
ori $1, $0, 1
subu RG1, $1, RG1
```

Set Not Equal : if $t2 not equal to $t3 then set $t1 to 1 else 0

```
sne $t1, $t2, $t3
subu RG1, RG2, RG3
sltu RG1, $0, RG1
```

Set Not Equal : if $t2 not equal to 16-bit immediate then set $t1 to 1 else 0

```
sne $t1, $t2, -100
addi $1, $0, VL3
subu RG1, RG2, $1
sltu RG1, $0, RG1
```

Set Not Equal : if $t2 not equal to 32-bit immediate then set $t1 to 1 else 0

```
sne $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
subu RG1, RG2, $1
sltu RG1, $0, RG1
```

SUBtraction : set $t1 to ($t2 minus 16-bit immediate)
```
sub $t1, $t2, -100
addi $1, $0, VL3U
sub RG1, RG2, $1
```

SUBtraction : set $t1 to ($t2 minus 32-bit immediate)
```
sub $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
sub RG1, RG2, $1
```

SUBtraction Immediate : set $t1 to ($t2 minus 16-bit immediate)
```
subi $t1, $t2, -100
addi $1, $0, VL3
sub RG1, RG2, $1
```

SUBtraction Immediate : set $t1 to ($t2 minus 32-bit immediate)
```
subi $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
sub RG1, RG2, $1
```

SUBtraction Immediate Unsigned : set $t1 to ($t2 minus 32-bit immediate), no overflow
```
subiu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
subu RG1, RG2, $1
```

SUBtraction Unsigned : set $t1 to ($t2 minus 32-bit immediate), no overflow
```
subu $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
subu RG1, RG2, $1
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, ($t2)
sw RG1,0(RG3)
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, -100
sw RG1, VL2($0)
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
sw RG1, 0($1)
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sw RG1, VL2($1)
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, 100000
lui $1, VH2
sw RG1,VL2($1)
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, 100
ori $1, $0, VL2U
sw RG1, 0($1)
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
sw RG1, LL2($1)
COMPACT
sw RG1, LL2(RG4)
```

Store Word : Store $t1 contents into memory word at label's address
```
sw $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sw RG1, LLP($1)
```

Store Word : Store $t1 contents into effective memory word address
```
sw $t1, label+100000
lui $1, LHPA
sw RG1, LLP($1)
```

Store Word : Store $t1 contents into memory word at label's address
```
sw $t1, label
lui $1, LH2
sw RG1, LL2($1)
COMPACT
sw RG1, LL2($0)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1, ($t2)
swc1 RG1,0(RG3)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1, -100
swc1 RG1,VL2($0)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
swc1 RG1, VL2($1)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1, 100000
lui $1, VH2
swc1 RG1,VL2($1)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1, label($t2)
lui $1, LH2
addu $1, $1, RG4
swc1 RG1, LL2($1)
COMPACT
swc1 RG1, LL2(RG4)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1,label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
swc1 RG1, LLP($1)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1, label+100000
lui $1, LHPA
swc1 RG1, LLP($1)
```

Store Word Coprocessor 1 : Store 32-bit value from $f1 to effective memory word address
```
swc1 $f1, label
lui $1, LH2
swc1 RG1, LL2($1)
COMPACT
swc1 RG1, LL2($0)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl ($t2)
swl RG1,0(RG3)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl -100
swl RG1,VL2($0)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
swl RG1, 0($1)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl 100000($t2)
lui $1, VH2
addu $1, $1, RG4
swl RG1, VL2($1)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl 100000
lui $1, VH2
swl RG1,VL2($1)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl 100
ori $1, $0, VL2U
swl RG1, 0($1)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl label($t2)
lui $1, LH2
addu $1, $1, RG4
swl RG1, LL2($1)
COMPACT
swl RG1, LL2(RG4)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
swl RG1, LLP($1)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl label+100000
lui $1, LHPA
swl RG1, LLP($1)
```

Store Word Left : Store high-order 1 to 4 bytes of $t1 into memory, starting with effective memory byte address and continuing through the low-order byte of its word
```
swl label
lui $1, LH2
swl RG1, LL2($1)
COMPACT
swl RG1, LL2($0)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr ($t2)
swr RG1,0(RG3)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, -100
swr RG1,VL2($0)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, 100($t2)
ori $1, $0, VL2U
addu $1, $1, RG4
swr RG1, 0($1)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
swr RG1, VL2($1)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, 100000
lui $1, VH2
swr RG1,VL2($1)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, 100
ori $1, $0, VL2U
swr RG1, 0
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
swr RG1, LL2($1)
COMPACT
swr RG1, LL2(RG4)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
swr RG1, LLP($1)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address
```
swr $t1, label+100000
lui $1, LHPA
swr RG1, LLP($1)
```

Store Word Right : Store low-order 1 to 4 bytes of $t1 into memory, starting with high-order byte of word containing effective memory byte address and continuing through that byte address

```
swr $t1, label
lui $1, LH2
swr RG1, LL2($1)
COMPACT
swr RG1, LL2($0)
```

U

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, ($t2)
lb RG1, 1(RG3)
lbu $1, 0(RG3)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, -100($t2)
lui $1, VH2P1
addu $1, $1, RG4
lb RG1, VL2P1($1)
lbu $1, VL2(RG4)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, 100000($t2)
lui $1, VH2P1
addu $1, $1, RG4
lb RG1, VL2P1($1)
lui $1, VH2
addu $1, $1, RG4
lbu $1, VL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, 100000
lui $1, VH2P1
lb RG1, VL2P1($1)
lui $1, VH2
lbu $1, VL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, label($t2)
lui $1, LH2P1
addu $1, $1, RG4
lb RG1, LL2P1($1)
lui $1, LH2
addu $1, $1, RG4
lbu $1, LL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, label+100000($t2)
lui $1, LHPAP1
addu $1, $1, RG6
lb RG1, LLPP1($1)
lui $1, LHPA
addu $1, $1, RG6
lbu $1, LLP($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, label+100000
lui $1, LHPAP1
lb RG1, LLPP1($1)
lui $1, LHPA
lbu $1, LLP($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, sign-extended, starting at effective memory byte address

```
ulh $t1, label
lui $1, LH2
lb RG1, LL2P1($1)
lui $1, LH2
lbu $1, LL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1, ($t2)
lbu RG1, 1(RG3)
lbu $1, 0(RG3)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1, -100($t2)
lui $1, VH2P1
addu $1, $1, RG4
lbu RG1, VL2P1($1)
lbu $1, VL2(RG4)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1, 100000($t2)
lui $1, VH2P1
addu $1, $1, RG4
lbu RG1, VL2P1($1)
lui $1, VH2
addu $1, $1, RG4
lbu $1, VL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1, 100000
lui $1, VH2P1
lbu RG1, VL2P1($1)
lui $1, VH2
lbu $1, VL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1, label($t2)
lui $1, LH2P1
addu $1, $1, RG4
lbu RG1, LL2P1($1)
lui $1, LH2
addu $1, $1, RG4
lbu $1, LL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1,label+100000($t2)
lui $1, LHPAP1
addu $1, $1, RG6
lbu RG1, LLPP1($1)
lui $1, LHPA
addu $1, $1, RG6
lbu $1, LLP($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1, label+100000
lui $1, LHPAP1
lbu RG1, LLPP1($1)
lui $1, LHPA
lbu $1, LLP($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Halfword : Set $t1 to the 16 bits, zero-extended, starting at effective memory byte address

```
ulhu $t1, label
lui $1, LH2P1
lbu RG1, LL2P1($1)
lui $1, LH2
lbu $1, LL2($1)
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, ($t2)
lwl RG1, 3(RG3)
lwr RG1, 0(RG3)
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, -100($t2)
lui $1, VH2P3
addu $1, $1, RG4
lwl RG1, VL2P3($1)
lwr RG1, VL2(RG4)
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, 100000($t2)
lui $1, VH2P3
addu $1, $1, RG4
lwl RG1, VL2P3($1)
lui $1, VH2
addu $1, $1, RG4
lwr RG1, VL2($1)
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, 100000
lui $1, VH2P3
lwl RG1, VL2P3($1)
lui $1, VH2
lwr RG1, VL2($1)
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, label($t2)
lui $1, LH2P3
addu $1, $1, RG4
lwl RG1, LL2P3($1)
lui $1, LH2
addu $1, $1, RG4
lwr RG1, LL2($1)
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, label+100000($t2)
lui $1, LHPAP3
addu $1, $1, RG6
lwl RG1, LLPP3($1)
lui $1, LHPA
addu $1, $1, RG6
lwr RG1, LLP($1)
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, label+100000
lui $1, LHPAP3
lwl RG1, LLPP3($1)
lui $1, LHPA
lwr RG1, LLP($1)
```

Unaligned Load Word : Set $t1 to the 32 bits starting at effective memory byte address

```
ulw $t1, label
lui $1, LH2P3
lwl RG1, LL2P3($1)
lui $1, LH2
lwr RG1, LL2($1)
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, ($t2)
sb RG1, 0(RG3)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
sb RG1, 1(RG3)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, -100($t2)
sb RG1, VL2(RG4)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
lui $1, VH2P1
addu $1, $1, RG4
sb RG1, VL2P1($1)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, 100000($t2)
lui $1, VH2
addu $1, $1, RG4
sb RG1, VL2($1)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
lui $1, VH2P1
addu $1, $1, RG4
sb RG1, VL2P1($1)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, 100000
lui $1, VH2
sb RG1, VL2($1)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
lui $1, VH2P1
sb RG1, VL2P1($1)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, label($t2)
lui $1, LH2
addu $1, $1, RG4
sb RG1, LL2($1)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
lui $1, LH2P1
addu $1, $1, RG4
sb RG1, LL2P1($1)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, label+100000($t2)
lui $1, LHPA
addu $1, $1, RG6
sb RG1, LLP($1)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
lui $1, LHPAP1
addu $1, $1, RG6
sb RG1, LLPP1($1)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, label+100000
lui $1, LHPA
sb RG1, LLP($1)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
lui $1, LHPAP1
sb RG1, LLPP1($1)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Halfword: Store low-order halfword $t1 contents into the 16 bits starting at effective memory byte address

```
ush $t1, label
lui $1, LH2
sb RG1, LL2($1)
sll $1, RG1, 24
srl RG1, RG1, 8
or RG1, RG1, $1
lui $1, LH2P1
sb RG1, LL2P1($1)
srl $1, RG1, 24
sll RG1, RG1, 8
or RG1, RG1, $1
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, ($t2)
swl RG1, 3(RG3)
swr RG1, 0(RG3)
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, -100($t2)
lui $1, VH2P3
addu $1, $1, RG4
swl RG1, VL2P3($1)
swr RG1, VL2(RG4)
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, 100000($t2)
lui $1, VH2P3
addu $1, $1, RG4
swl RG1, VL2P3($1)
lui $1, VH2
addu $1, $1, RG4
swr RG1, VL2($1)
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, 100000
lui $1, VH2P3
swl RG1, VL2P3($1)
lui $1, VH2
swr RG1, VL2($1)
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, label($t2)
lui $1, LH2P3
addu $1, $1, RG4
swl RG1, LL2P3($1)
lui $1, LH2
addu $1, $1, RG4
swr RG1, LL2($1)
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, label+100000($t2)
lui $1, LHPAP3
addu $1, $1, RG6
swl RG1, LLPP3($1)
lui $1, LHPA
addu $1, $1, RG6
swr RG1, LLP($1)
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, label+100000
lui $1, LHPAP3
swl RG1, LLPP3($1)
lui $1, LHPA
swr RG1, LLP($1)
```

Unaligned Store Word : Store $t1 contents into the 32 bits starting at effective memory byte address

```
usw $t1, label
lui $1, LH2P3
swl RG1, LL2P3($1)
lui $1, LH2
swr RG1, LL2($1)
```

X

XOR : set $t1 to ($t2 bitwise-exclusive-OR 16-bit unsigned immediate)

```
xor $t1, $t2, 100
xori RG1, RG2, VL3U
```

XOR : set $t1 to ($t1 bitwise-exclusive-OR 16-bit unsigned immediate)

```
xor $t1, 100
xori RG1, RG1, VL2U
```

XOR Immediate : set $t1 to ($t2 bitwise-exclusive-OR 32-bit immediate)

```
xori $t1, $t2, 100000
lui $1, VHL3
ori $1, $1, VL3U
xor RG1, RG2, $1
```

XOR Immediate : set $t1 to ($t1 bitwise-exclusive-OR 32-bit immediate)

```
xori $t1, 100000
lui $1, VHL2
ori $1, $1, VL2U
xor RG1, RG1, $1
```

XOR Immediate : set $t1 to ($t1 bitwise-exclusive-OR 16-bit unsigned immediate)

```
xori $t1, 100
xori RG1, RG1, VL2U
```

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