CUADRO RESUMEN DEL LENGUAJE ENSAMBLADOR BÁSICO DEL MIPS-R2000

CARGA

lw rt,dirección *Carga palabra* **I**

Carga los 32 bits almacenados en la palabra de memoria especificada por dirección en el registro rt.

```
lw $s0, 12($a0)   #  $s0 ← Mem[ 12 + $a0 ]
```

lb rt, dirección *Carga byte y extiende signo* **I**

Carga los 8 bits almacenados en el byte de memoria especificado por dirección en el LSB del registro rt y extiende el signo

```
lb $s0, 12($a0)   #  $s0(7..0) ← Mem[ 12 + $a0 ](1byte)
                  #  $s0(31..8) ← $s0(7)
```

lbu rt, dirección *Carga byte y no extiende signo* **I**

Carga los 8 bits almacenados en el byte de memoria especificado por dirección en el LSB del registro rt sin extender el signo

```
lbu $s0, 12($a0)  #  $s0 ← 0x000000(Mem[ 12 + $a0 ])(1byte)
```

lh rt, dirección *Carga media palabra y ext. signo* **I**

Carga media palabra (16 bits) almacenada en la media palabra de memoria especificada por la dirección en la parte baja del registro rt y extiende el signo

```
lh $s0, 12($a0)   #  $s0 (15..0)← Mem[ 12 + $a0 ](2bytes)
                  #  $s0 (31..16)← $s0(15)
```

lhu rt, dirección *Carga media palabra y no ext. signo* **I**

Carga media palabra (16 bits) almacenada en la media palabra de memoria especificada por la dirección en la parte baja del registro rt y no extiende el signo

```
lhu $s0, 12($a0)  #  $s0← 0x0000Mem[ 12 + $a0 ](2bytes)
```

la reg, dirección *Carga dirección* **PS**

Carga la dirección calculada en reg

```
la $s0, VAR       #  $s0← dir. asociada a etiqueta VAR
```

lui rt, dato *Carga inmediata superior* **I**

Carga el dato inmediato en los 16 MSB del registro rt

```
lui $s0, 12       #  $s0(31..16) ← 12
                  #  $s0(15..0) ← 0x0000
```

li reg, dato *Carga inmediato* **PS**

Carga el dato inmediato en el registro reg.

```
li $s0, 12        #  $s0 ← 12
```

ARITMÉTICAS

add rd, rs, rt *Suma* **R**

Suma el contenido de los registros rs y rt, considerando el signo. El resultado se almacena en el registro rd

```
add $t0, $a0, $a1         # $t0 ←$a0 + $a1
```

addu rd, rs, rt *Suma sin signo* **R**

Suma el contenido de los registros rs y rt, sin considerar el signo. El resultado se almacena en el registro rd

```
addu $t0, $a0, $a1        # $t0 ←$a0 + $a1
```

sub rd, rs, rt *Resta* **R**

Resta el contenido de los registro rs y rt considerando el signo. El resultado se almacena en el registro rd

```
sub $t0, $a0, $a1         # $t0 ←$a0 - $a1
```

subu rd, rs, rt *Resta sin signo* **R**

Resta el contenido de los registros rs y rt, sin considerar el signo. El resultado se almacena en el registro r.

```
subu $t0, $a0, $a1        # $t0 ←$a0 - $a1
```

addi rt, rs, valor *Suma inmediata* **I**

Suma el contenido del registro rs con el valor inmediato, considerando el signo. El resultado se almacena en el registro rt.

```
addi $t0, $a0, -24        # $t0 ←$a0 + (-24)
```

addiu rt, rs, valor *Suma inmediata sin signo* **I**

Suma el contenido del registro rs con el valor inmediato, sin considerar el signo. El resultado se almacena en el registro rt.

```
addiu $t0, $a0, 24                # $t0 ←$a0 + 24
```

mult rs, rt *Multiplicación* **R**

Multiplica el contenido de los registros rs y rt. Los 32 MSB del resultado se almacenan en el registro HI y los 32 LSB en el registro LO

```
mult $s0, $s1     # $HI ←($s0 * $s1) (31…16)
                  # $LO←($s0 * $s1) (15…0)
```

div rs, rt *División* **R**

Divide el registro rs por el rt. El cociente se almacena en LO y el resto en HI.

```
div $s0, $s1      # $LO ← $s0 / $s1
                  # $HI ← $s0 % $s1
```

COMPARACIONES

slt rd, rs, rt *Activa si menor* **R**

Pone el registro rd a 1 si rs es menor que rt y a 0 en caso contrario

```
slt $t0, $a0, $a1         # if ( $a0 < $a1) $t0 ← 1
                          # else $t0 ← 0
```

slti rt, rs, inm *Activa si menor con inmediato* **I**

Pone el registro rt a 1 si rs es menor que el dato inmediato inm y a 0 en caso contrario

```
slti $t0, $a0, -15        # if ( $a0 < -15) $t0 ← 1
                          # else $t0 ← 0
```

seq rdest, rsrc1, rsrc2 *Activa si igual* **PS**

Pone el registro rdest a 1 si rsrc1 es igual que rsrc2 y a 0 en caso contrario

```
seq $t0, $a0, $a2         # if ( $a0 == $a2) $t0 ← 1
                          # else $t0 ← 0
```

sge rdest, rsrc1, rsrc2 *Activa si mayor o igual* **PS**

Pone el registro rdest a 1 si rsrc1 es mayor o igual que rsrc2 y a 0 en caso contrario

```
sge $t0, $a0, $a2         # if ( $a0 >= $a2) $t0 ← 1
                          # else $t0 ← 0
```

sgt rdest, rsrc1, rsrc2 *Activa si mayor* **PS**

Pone el registro rdest a 1 si rsrc1 es mayor que rsrc2 y a 0 en caso contrario

```
sgt $t0, $a0, $a2         # if ( $a0 > $a2) $t0 ← 1
                          # else $t0 ← 0
```

sle rdest, rsrc1, rsrc2 *Activa si menor o igual* **PS**

Pone el registro rdest a 1 si rsrc1 es menor o igual que rsrc2 y a 0 en caso contrario

```
sle $t0, $a0, $a2         # if ( $a0 <= $a2) $t0 ← 1
                          # else $t0 ← 0
```

sne rdest, rsrc1, rsrc2 *Activa si no igual* **PS**

Pone el registro rdest a 1 si rsrc1 es diferente de rsrc2 y a 0 en caso contrario

```
sne $t0, $a0, $a2         # if ( $a0 != $a2) $t0 ← 1
                          # else $t0 ← 0
```

CUADRO RESUMEN DEL LENGUAJE ENSAMBLADOR BÁSICO DEL MIPS-R2000

ALMACENAMIENTO

sw rt, dirección *Almacena palabra* **I**

Almacena el contenido del registro rt en la palabra de memoria indicada por dirección

```
sw $s0, 12($a0)  # Mem[ 12 + $a0 ] ←$s0
```

sb rt, dirección *Almacena byte* **I**

Almacena el LSB del registro en el byte de memoria indicado por dirección

```
sb $s0, 12($a0)  # Mem[ 12 + $a0 ] ←$s0(7..0)
```

sh rt, dirección *Almacena media palabra* **I**

Almacena en los 16 bits de menos peso del registro en la media palabra de memoria indicada por dirección.

```
sh $s0, 12($a0)  # Mem[ 12 + $a0 ] ←$s0(15..0)
```

LÓGICAS

and rd, rs, rt *AND entre registros* **R**

Operación AND bit a bit entre los registros rs y rt. El resultado se almacena en rd

```
and $t0, $a0, $a1        # $t0 ←$a0 & $a1
```

andi rt, rs, inm *AND con inmediato* **I**

Operación AND bit a bit entre el dato inmediato, extendiento ceros, y el registro rs. El resultado se almacena en rt.

```
andi $t0, $a0, 0xA1FF    # $t0 ←$a0 & (0x0000A1FF)
```

or rd, rs, rt *OR entre registros* **R**

Operación OR bit a bit entre los registros rs y rt. El resultado se almacena en rd

```
or $t0, $a0, $a1         # $t0 ←$a0 | $a1
```

ori rt, rs, inm *OR con inmediato* **I**

Operación OR bit a bit entre el dato inmediato, extendiendo ceros, y el registro rs. El resultado se almacena en rt.

```
ori $t0, $a0, 0xA1FF     # $t0 ←$a0 | (0x0000A1FF)
```

MOVIMIENTO ENTRE REGISTROS

mfhi rd *mueve desde HI* **R**

Transfiere el contenido del registro HI al registro rd.

```
mfhi $t0         # $t0 ← HI
```

mflo rd *mueve desde LO* **R**

Transfiere el contenido del registro LO al registro rd.

```
mflo $t1         # $t1 ← LO
```

FORMATO DE LAS INSTRUCCIONES

tipo **R**

inst(6 bits)	rs(5bits)	rt (5bits)	rd(5bits)	shamt(5bits)	co (6bits)

tipo **I**

inst(6bits)	rs(5bits)	rt(5bits)	inm(16 bits)

tipo **J**

inst(6bits)	objetivo (26 bits)

DESPLAZAMIENTO

sll rd, rt, shamt *Desplamiento logico a la izquierda* **R**

Desplaza el registro rt a la izquierda tantos bits como indica shamt

```
sll $t0, $t1, 16     # $t0 ← $t1 << 16
```

srl rd, rt, shamt *Desplazamiento lógico a la derecha* **R**

Desplaza el registro rt a la derecha tantos bits como indica shamt.

```
srl $s0,$t1, 4       # $s0 ← $t1 >> 4
```

sra rd, rt, shamt *Desplaz. aritmético a la derecha* **R**

Desplaza el registro rt a la derecha tantos bits como indica shamt. Los bits MSB toman el mismo valor que el bit de signo de rt. El resultado se almacena en rd

```
sra $s0,$t1, 4       # $s0 ← $t1 >> 4
                     # $s0(31..28) ← $t1(31)
```

SALTOS INCONDICIONALES

j dirección *Salto incondicional* **J**

Salta a la instrucción apuntada por la etiqueta dirección

```
j finbucle           # $pc ← dirección etiqueta finbucle
```

jal dirección *Saltar y enlazar* **J**

Salta a la instrucción apuntada por la etiqueta dirección y almacena la dirección de la instrucción siguiente en $ra

```
jal rutina           # $pc ← dirección etiqueta rutina
                     # $ra ← dirección siguiente instrucción
```

jr rs *Saltar a registro* **R**

Salta a la instrucción apuntada por el contenido del registro rs.

```
jr $ra     # $pc ← $ra
```

SALTOS CONDICIONALES

beq rs, rt, etiqueta *Salto si igual* **I**

Salta a etiqueta si rs es igual a rt

```
beq $t0, $t1, DIR        # if ( $t0=$t1) $pc ← DIR
```

bgez rs, etiqueta *Salto si mayor o igual que cero* **I**

Salta a etiqueta si rs es mayor o igual que 0

```
bgez $t0, SLT            # if ( $t0>=0) $pc ← SLT
```

bgtz rs, etiqueta *Salto si mayor que cero* **I**

Salta a etiqueta si rs es mayor que 0

```
bgtz $t0, SLT            # if ( $t0>0) $pc ← SLT
```

blez rs, etiqueta *Salto si menor o igual que cero* **I**

Salta a etiqueta si rs es menor o igual que 0

```
blez $t1, ETQ            # if ( $t1<=0) $pc ← ETQ
```

bltz rs, etiqueta *Salto si menor que cero* **I**

Salta a etiqueta si rs es menor que 0

```
bltz $t1, ETQ            # if ( $t1<0) $pc ← ETQ
```

bne rs, rt, etiqueta *Salto si distinto* **I**

Salta a etiqueta si rs es diferente de rt

```
bne $t0, $t1, DIR        # if ( $t0<>$t1) $pc ← DIR
```

bge reg1, reg2, etiq *Salto mayor o igual* **PS**

Salta a etiq si reg1 es mayor o igual que reg2

```
bge $t0, $t1, DIR        # if ( $t0>=$t1) $pc ← DIR
```

bgt reg1, reg2, etiq *Salto mayor* **PS**

Salta a etiq si reg1 es mayor que reg2

```
bgt $t0, $t1, DIR        # if ( $t0>$t1) $pc ← DIR
```

ble reg1, reg2, etiq *Salto menor o igual* **PS**

Salta a etiq si reg1 es menor o igual que reg2

```
ble $t0, $t1, DIR        # if ( $t0<=$t1) $pc ← DIR
```

blt reg1, reg2, etiq *Salto menor* **PS**

Salta a etiq si reg1 es menor que reg2

```
blt $t0, $t1, DIR        # if ( $t0<$t1) $pc ← DIR
```