Manual de Programación MIPS.

Las tablas siguientes permiten ensamblar y desensamblar. La tabla assembler es la especificación del repertorio, y se usa para programar.

Códigos Binarios | **Lenguaje Simbólico. Assembler.**

OP	RS	RT	RD	Shamt	Funct		Nemo.	Cmp1	Campo2	Cmp3	Descripción
000000	00000	Fte1	Dst	shitf5	000000	RI	**sll**	Dst,	Fte1,	shift5	#Shift Left Logical
000000	00000	Fte1	Dst	shitf5	000010	RI	**srl**	Dst,	Fte1,	shift5	#Shift Right Logical
000000	00000	Fte1	Dst	shitf5	000011	RI	**sra**	Dst,	Fte1,	shift5	#Shift Right Arithmetic
000000	Fte2	Fte1	Dst	00000	000100	R	**sllv**	Dst,	Fte1,	Fte2	#Shift Left Logical Variable
000000	Fte2	Fte1	Dst	00000	000110	R	**srlv**	Dst,	Fte1,	Fte2	#Shift Right Logical Variable
000000	Fte2	Fte1	Dst	00000	000111	R	**srav**	Dst,	Fte1,	Fte2	#Shift Right Arithmetic Variable
000000	Dst1	00000	00000	00000	001000	R	**jr**	Dst1			#Jump Register
000000	Dst1	00000	11111	00000	001001	R	**jalr**	Dst1			#Jump and Link Register
000000	00000	00000	00000	00000	001100	R	**syscall**				#System Call. (see Table 1) provided by SPIM.
000000	n1	n2	n3	n4	001101	R	**break**	n20			#Break Cause exception n .Exception 1 is reserved for the debugger.
000000	00000	00000	Dst	00000	010000	R	**mfhi**	Dst			#Move From hi
000000	Dst1	00000	00000	00000	010001	R	**mthi**	Dst1			#Move To hi
000000	00000	00000	Dst	00000	010010	R	**mflo**	Dst			#Move From lo
000000	Dst1	00000	00000	00000	010011	R	**mtlo**	Dst1			#Move To lo
000000	Fte1	Fte2	00000	00000	011000	R	**mult**	Fte1,	Fte2		#Multiply
000000	Fte1	Fte2	00000	00000	011001	R	**multu**	Fte1,	Fte2		#Unsigned Multiply
000000	Fte1	Fte2	00000	00000	011010	R	**div**	Fte1,	Fte2		#Divide (signed)
000000	Fte1	Fte2	00000	00000	011011	R	**divu**	Fte1,	Fte2		#Divide (unsigned)
000000	Fte1	Fte2	Dst	00000	100000	R	**add**	Dst,	Fte1,	Fte2	#Addition (with over ow)
000000	Fte1	Fte2	Dst	00000	100001	R	**addu**	Dst,	Fte1,	Fte2	#Addition (without over ow)
000000	Fte1	Fte2	Dst	00000	100010	R	**sub**	Dst,	Fte1,	Fte2	#Subtract (with over ow)
000000	Fte1	Fte2	Dst	00000	100011	R	**subu**	Dst,	Fte1,	Fte2	#Subtract (without over ow)
000000	Fte1	Fte2	Dst	00000	100100	R	**and**	Dst,	Fte1,	Fte2	#AND
000000	00000	00000	00000	00000	100101	R	**nop**				#No operation Do nothing. Implementada con or en SPIM.
000000	Fte1	Fte2	Dst	00000	100101	R	**or**	Dst,	Fte1,	Fte2	#OR
000000	Fte1	Fte2	Dst	00000	100110	R	**xor**	Dst,	Fte1,	Fte2	#XOR
000000	Fte1	Fte2	Dst	00000	100111	R	**nor**	Dst,	Fte1,	Fte2	#NOR
000000	Fte1	Fte2	Dst	00000	101010	R	**slt**	Dst,	Fte1,	Fte2	#Set Less Than
000000	Fte1	Fte2	Dst	00000	101011	R	**sltu**	Dst,	Fte1,	Fte2	#Set Less Than Unsigned

000001	Dst1	00000	label16			I	**bltz**	Dst,	label16		#Branch on Less Than Zero
000001	Dst1	00001	label16			I	**bgez**	Dst,	label16		#Branch on Greater Than Equal Zero
000001	Dst1	10000	label16			I	**bltzal**	Dst,	label16		#Branch on Less Than And Link
000001	Dst1	10001	label16			I	**bgezal**	Dst,	label16		#Branch on Greater Than Equal Zero And Link
000001	Dst1	10001	label16			I	**bgezal**	Dst,	label16		#Branch on Greater Than Equal Zero And Link
000010	jmp26					J	**j**	jmp26			#Jump
000011	jmp26					J	**jal**	jmp26			#Jump and Link
000100	Fte1	Fte2	label16			I	**beq**	Fte1,	Fte2,	label16	#Branch on Equal
000101	Fte1	Fte2	label16			I	**bne**	Fte1,	Fte2,	label16	#Branch on Not Equal
000110	Dst1	00000	label16			I	**blez**	Dst,	label16		#Branch on Less Than Equal Zero
000111	Dst1	00000	label16			I	**bgtz**	Dst,	label16		#Branch on Greater Than Zero
001000	Fte1	Dst	inm16			I	**addi**	Dst,	Fte1,	inm16	#Addition Immediate (with over ow)
001001	Fte1	Dst	inm16			I	**addiu**	Dst,	Fte1,	inm16	#Addition Immediate (without over ow)
001010	Fte1	Dst	inm16			I	**slti**	Dst,	Fte1,	inm16	#Set Less Than Immediate
001011	Fte1	Dst	inm16			I	**sltiu**	Dst,	Fte1,	inm16	#Set Less Than Unsigned Immediate
001100	Fte1	Dst	inm16			I	**andi**	Dst,	Fte1,	inm16	#AND Immediate
001101	Fte1	Dst	inm16			I	**ori**	Dst,	Fte1,	inm16	#OR Immediate
001110	Fte1	Dst	inm16			I	**xori**	Dst,	Fte1,	inm16	#XOR Immediate
001111	00000	Dst	addr16			I	**lui**	Dst,	addr16		#Load Upper Immediate
010000	10000	00000	00000	00000	010000	R	**rfe**				#Return From Exception. Restore the Status register.
100000	Rbase	Dst	offset16			I	**lb**	Dst,	Offset16(RBase)		#Load Byte
100001	Rbase	Dst	offset16			I	**lh**	Dst,	Offset16(RBase)		#Load Halfword
100010	Rbase	Dst	offset16			I	**lwl**	Dst,	Offset16(RBase)		#Load Word Left
100011	Rbase	Dst	offset16			I	**lw**	Dst,	Offset16(RBase)		#Load Word
100100	Rbase	Dst	offset16			I	**lbu**	Dst,	Offset16(RBase)		#Load Unsigned Byte
100101	Rbase	Dst	offset16			I	**lhu**	Dst,	Offset16(RBase)		#Load Unsigned Halfword
100110	Rbase	Dst	offset16			I	**lwr**	Dst,	Offset16(RBase)		#Load Word Right
101000	Rbase	Fte1	offset16			I	**sb**	Fte1,	Offset16(RBase)		#Store Byte
101001	Rbase	Fte1	offset16			I	**sh**	Fte1,	Offset16(RBase)		#Store the low halfword.
101010	Rbase	Fte1	offset16			I	**swl**	Fte1,	Offset16(RBase)		#Store Word Left
101011	Rbase	Fte1	offset16			I	**sw**	Fte1,	Offset16(RBase)		#Store Word
101110	Rbase	Fte1	offset16			I	**swr**	Fte1,	Offset16(RBase)		#Store Word Right

Tipos de Campos	
inm16	Valor inmediato (con signo, en caso aritmético) de 16 bits.
addr16	Dirección. Valor de 16 bits sin signo.
label16	Direccionamiento relativo a PC. Especifica número (con signo) de instrucciones.
jmp26	Campo de 26 bits. Para formar dirección de salto.
offset16	Número con signo, en complemento dos.
shitf5	Campo de 5 bits. Especifica el número de corriemientos en bits.
Dst	Especificación de registro destino. En campo 15..11
Dst1	Especificación de registro destino. En campo 25..21
Fte1	Especificación de registro.
Fte2	Especificación de registro.
n20	Campo de 20 bits.

Registros. Todos de 32 bits. Se requieren 5 bits para especificar un registro.

Register Name.	**Number**	**Usage**
zero	0	Constant 0
at	1	Reserved for assembler
v0	2	Expression evaluation and
v1	3	results of a function
a0	4	Argument 1
a1	5	Argument 2
a2	6	Argument 3
a3	7	Argument 4
t0	8	Temporary (not preserved across call)
t1	9	Temporary (not preserved across call)
t2	10	Temporary (not preserved across call)
t3	11	Temporary (not preserved across call)
t4	12	Temporary (not preserved across call)
t5	13	Temporary (not preserved across call)
t6	14	Temporary (not preserved across call)
t7	15	Temporary (not preserved across call)
s0	16	Saved temporary (preserved across call)
s1	17	Saved temporary (preserved across call)
s2	18	Saved temporary (preserved across call)
s3	19	Saved temporary (preserved across call)
s4	20	Saved temporary (preserved across call)
s5	21	Saved temporary (preserved across call)
s6	22	Saved temporary (preserved across call)
s7	23	Saved temporary (preserved across call)
t8	24	Temporary (not preserved across call)
t9	25	Temporary (not preserved across call)
k0	26	Reserved for OS kernel
k1	27	Reserved for OS kernel
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address (used by function call)

Table 2: MIPS registers and the convention governing their use.

1. Instrucciones del Repertorio del procesador MIPS.

a) Aritmético Lógicas.

```
.text
add     $t1, $t2, $t3          #Addition (with over ow)
addi    $t1, $t2, 0x1000       #Addition Immediate (with over ow)
addu    $t1, $t2, $t3          #Addition (without over ow)
addiu   $t1, $t2, 0x1000       #Addition Immediate (without over ow)
and     $t1, $t2, $t3          #AND
andi    $t1, $t2, 0x1000       #AND Immediate
div     $t2, $t3               #Divide (signed)
divu    $t2, $t3               #Divide (unsigned)
mult    $t2, $t3               #Multiply
multu   $t2, $t3               #Unsigned Multiply
nor     $t1, $t2, $t3          #NOR
or      $t1, $t2, $t3          #OR
ori     $t1, $t2, 0x1000       #OR Immediate
sll     $t1, $t2, 31           #Shift Left Logical
sllv    $t1, $t2, $t3          #Shift Left Logical Variable
sra     $t1, $t2, 31           #Shift Right Arithmetic
srav    $t1, $t2, $t3          #Shift Right Arithmetic Variable
srl     $t1, $t2, 31           #Shift Right Logical
srlv    $t1, $t2, $t3          #Shift Right Logical Variable
sub     $t1, $t2, $t3          #Subtract (with over ow)
subu    $t1, $t2, $t3          #Subtract (without over ow)
xor     $t1, $t2, $t3          #XOR
xori    $t1, $t2, 0x1000       #XOR Immediate
lui     $t1, 0x1000            #Load Upper Immediate
slt     $t1, $t2, $t3          #Set Less Than
slti    $t1, $t2, 0x1000       #Set Less Than Immediate
sltu    $t1, $t2, $t3          #Set Less Than Unsigned
sltiu   $t1, $t2, 0x1000       #Set Less Than Unsigned Immediate
```

b) Bifurcaciones y Saltos

```
label:  beq     $t2, $t3, label         #Branch on Equal
        bgez    $t1, label              #Branch on Greater Than Equal Zero
        bgezal  $t1, label              #Branch on Greater Than Equal Zero And Link
        bgtz    $t1, label              #Branch on Greater Than Zero
        blez    $t1, label              #Branch on Less Than Equal Zero
        bgezal  $t1, label              #Branch on Greater Than Equal Zero And Link
        bltzal  $t1, label              #Branch on Less Than And Link
        bltz    $t1, label              #Branch on Less Than Zero
        bne     $t2, $t3, label         #Branch on Not Equal

        j       label                   #Jump
        jal     label                   #Jump and Link
        jalr    $t1                     #Jump and Link Register
        jr      $t1                     #Jump Register
```

c) Cargas y transferencias.

```
        lb      $t1, 0($t2)             #Load Byte
        lbu     $t1, 0($t2)             #Load Unsigned Byte
        lh      $t1, 0($t2)             #Load Halfword
        lhu     $t1, 0($t2)             #Load Unsigned Halfword
        lw      $t1, 0($t2)             #Load Word
        lwl     $t1, 0($t2)             #Load Word Left
        lwr     $t1, 0($t2)             #Load Word Right
        sb      $t1, 0($t2)             #Store Byte
        sh      $t1, 0($t2)             #Store Halfword
#Store the low halfword from register Rsrc at address.
        sw      $t1, 0($t2)             #Store Word
        swl     $t1, 0($t2)             #Store Word Left
        swr     $t1, 0($t2)             #Store Word Right
        mfhi    $t1                     #Move From hi
        mflo    $t1                     #Move From lo
        mthi    $t1                     #Move To hi
        mtlo    $t1                     #Move To lo
```

d) Control del procesador y coprocesadores.

```
        rfe                             #Return From Exception.  Restore the Status register.
        syscall                         #System Call. Register $v0 contains the number of
                                        #the system call (see Table 1) provided by SPIM.
        break   0                       #Break Cause exception n .
                                        #Exception 1 is reserved for the debugger.
        nop                             #No operation Do nothing.
El coprocesador de punto flotante es el número 1.
        lwcz    Reg, 0($t2)             #Load Word Coprocessor
#Load the word at address into register Reg of coprocessor z (0..3).
        swcz    Reg, 0($t2)             #Store Word Coprocessor
#Store the word from register Reg of coprocessor z at address.
        mfcz    CpuSrc, CPdest          #Move CPdest From Coprocessor z
#Move the contents of coprocessor z 's register CPdest to CPU register CpuSrc.
        mtcz    CpuSrc, CPdest          #Move To Coprocessor z
#Move the contents of CPU register CpuSrc to coprocessor z 's register CPdest.
        bczt    label                   #Branch to label if Coprocessor z flag is set (True)
        bczf    label                   #Branch to label if Coprocessor z flag not set (False)
```

e) Instrucciones de punto flotante.

```
abs.d    $f0, $f2          #Floating Point Absolute Value Double
abs.s    $f0, $f2          #Floating Point Absolute Value Single
#Compute the absolute value of the floating float double (single) in register $f2 and put it in register $f0.

add.d    $f0, $f2, $f4     #Floating Point Addition Double
add.s    $f0, $f2, $f4     #Floating Point Addition Single
#Compute the sum of the floating float doubles (singles) in registers $f2 and $f4 and put it in register $f0.

c.eq.d   $f0, $f2          #Compare Equal Double
c.eq.s   $f0, $f2          #Compare Equal Single
#Compare the floating point double (single) in register $f0 against the one in $f2 and set the floating point
#condition flag true if they are equal.

c.le.d   $f0, $f2          #Compare Less Than Equal Double
c.le.s   $f0, $f2          #Compare Less Than Equal Single
c.lt.d   $f0, $f2          #Compare Less Than Double
c.lt.s   $f0, $f2          #Compare Less Than Single
#Compare the floating point double in register $f0 against the one in $f2 and set the condition flag true
#if the first is less than the second.

cvt.d.s  $f0, $f4          #Convert Single  to Double
cvt.d.w  $f0, $f4          #Convert Integer to Double
#Convert the single precision floating point number or integer in register $f4 to a
#double precision number and put it in register $f0.

cvt.s.d  $f0, $f4          #Convert Double to Single
cvt.s.w  $f0, $f4          #Convert Integer to Single
#Convert the double precision floating point number or integer in register $f4 to a
#single precision number and put it in register $f0.

cvt.w.d  $f0, $f4          #Convert Double to Integer
cvt.w.s  $f0, $f4          #Convert Single to Integer
#Convert the double or single precision floating point number in register $f4 to
#an integer and put it in register $f0.

div.d    $f0, $f0, $f2     #Floating Point Divide Double
div.s    $f0, $f0, $f2     #Floating Point Divide Single

mov.d    $f0, $f4          #Move Floating Point Double
mov.s    $f0, $f4          #Move Floating Point Single
#Move the floating float double (single) from register $f4 to register $f0.

mul.d    $f0, $f2, $f4     #Floating Point Multiply Double
mul.s    $f0, $f2, $f4     #Floating Point Multiply Single
#Compute the product of the floating float doubles (singles)
#in registers $f2 and $f4 and put it in register $f0.

neg.d     $f0, $f4         #Negate Double
neg.s    $f0, $f4          #Negate Single
#Negate the floating point double (single) in register $f4 and put it in register $f0.

sub.d    $f0, $f2, $f4     #Floating Point Subtract Double
sub.s    $f0, $f2, $f4     #Floating Point Subtract Single
#Compute the difference of the floating float doubles (singles)
#in registers $f2 and $f4 and put it in register $f0.
```

2. Macros definidos en SPIM.

```
#En el breve comentario, se explica la función de la macro.

#En general RegDst=$t1   Src1=$t2  Src2=$t3

abs $t1, $t2                #Absolute Value

div $t1, $t2, $t3           #Divide (signed, with over ow)
divu $t1, $t2, $t3          #Divide (unsigned, without over ow)
#Put the quotient of the integers from register $t2 and $t3 into register $t1.
#divu treats is operands as unsigned values.

mul $t1, $t2, $t3           #Multiply (without over ow)

mulo $t1, $t2, $t3          #Multiply (with over ow)

mulou $t1, $t2, $t3         #Unsigned Multiply (with over ow)

neg $t1, $t2                #Negate Value (with over ow)

negu $t1, $t2               #Negate Value (without over ow)

not $t1, $t2                #NOT

rem $t1, $t2, $t3           #Remainder
remu $t1, $t2, $t3          #Unsigned Remainder
#Put the remainder from dividing the integer in register $t2 by the
#integer in $t3 into register $t1. Note that if an operand is negative,
#the remainder is unspecied by the MIPS architecture
#and depends on the conventions of the machine on which SPIM is run.

rol $t1, $t2, $t3           #Rotate Left
ror $t1, $t2, $t3           #Rotate Right
#Rotate the contents of register $t2 left (right) by the distance
#indicated by $t3 and put the result in register $t1.

li $t1, 0x10002000          #Load Immediate


#In all instructions below, Src2 can either be a register or an
# immediate value (a 16 bit integer).

seq $t1, $t2, $t3           #Set Equal reg
seq $t1, $t2, 0x1000        #Set Equal inmm

sge $t1, $t2, $t3
sge $t1, $t2, 0x1000        #Set Greater Than Equal

sgeu $t1, $t2, $t3
sgeu $t1, $t2, 0x1000       #Set Greater Than Equal Unsigned
#Set register $t1 to 1 if register $t2 is greater than or equal to Src2 and to 0 otherwise.

sgt $t1, $t2, $t3
sgt $t1, $t2, 0x1000        #Set Greater Than

sgtu $t1, $t2, $t3
```

```
sgtu $t1, $t2, 0x1000        #Set Greater Than Unsigned
#Set register $t1 to 1 if register $t2 is greater than Src2 and to 0 otherwise.

sle $t1, $t2, $t3
sle $t1, $t2, 0x1000         #Set Less Than Equal

sleu $t1, $t2, $t3
sleu $t1, $t2, 0x1000        #Set Less Than Equal Unsigned


sne $t1, $t2, $t3            #Set Not Equal
#Set register $t1 to 1 if register $t2 is not equal to $t3 and to 0 otherwise.


#In all instructions below, $t2 can either be a register or an immediate value (integer).
#Branch instructions use a signed 16-bit offset field; hence they can jump 2 elevado a 15
#menos 1 instructions (not bytes) forward or 2 elevado a 15 instructions backwards.
#The jump instruction contains a 26 bit address field.

label:    b label                    #Branch instruction

beqz $t1, label                      #Branch on Equal Zero
#Conditionally branch to the instruction at the label if the contents of $t1 equals 0.

bge $t1, $t2, label
bge $t1, 0x10002000, label           #Branch on Greater Than Equal

bgeu $t1, $t2, label
bgeu $t1, 0x10002000, label          #Branch on GTE Unsigned

bgt $t1, $t2, label                  #Branch on Greater Than

bgtu $t1, $t2, label                 #Branch on Greater Than Unsigned

ble $t1, $t2, label                  #Branch on Less Than Equal

bleu $t1, $t2, label                 #Branch on LTE Unsigned

blt $t1, $t2, label                  #Branch on Less Than

bltu $t1, $t2, label                 #Branch on Less Than Unsigned

bnez $t1, label                      #Branch on Not Equal Zero

la $t1, 0x10002000                   #Load Address

ld $t1, 0x10002000300040000          #Load Double-Word
#Load the 64-bit quantity ataddress into registers $t1 and $t1 + 1.

ulh $t1, 0x10002000                  #alineada
ulh $t1, 0x10002001                  #dirección no alineada
# Unaligned Load Halfword

ulhu $t1, 0x10002000                 #Unaligned Load Halfword Unsigned
#Load the 16-bit quantity (halfword) at the possibly-unaligned address into register $t1.
# The halfword is sign-extended by the ulh, but not the ulhu, instruction
```

```
ulw $t1, 0x10002000             # Unaligned Load Word
#Load the 32-bit quantity (word) at the possibly-unaligned address into register $t1.


sd $t1, 0x1000200030004000      # Store Double-Word
#Store the 64-bit quantity in registers $t1 and $t2 + 1 at address.

ush $t1, 0x10002000             # Unaligned Store Halfword
#Store the low halfword from register $t1 at the possibly-unaligned address.

usw $t1, 0x10002000             #Unaligned Store Word
#Store the word from register $t1 at the possibly-unaligned address.

move $t1, $t2                   #Move
```

#macros inversos.

```
#Se muestra el código que expande automáticamente el ensamblador.
#Como comentario se escribe el llamado a la macro..

#set noat permite usar el registro  at.
        .set noat

        addu $t1, $zero, $t2    #abs $t1, $t2
        bgez $t2, siga0
        sub $t1, $zero, $t2
siga0:

        bne $t3,$zero, divida   #div $t1, $t2, $t3
        break 0
divida: div $t2, $t3
        mflo $t1

        bne $t3, $zero,divida2  #divu $t1, $t2, $t3
        break 0
divida2:divu $t2, $t3
        mflo $t1

        mult $t2, $t3           #mul $t1, $t2, $t3
        mflo $t1

        mult $t2, $t3           #mulo $t1, $t2, $t3
        mfhi $at                #$at $at
        mflo $t1
        sra $t1, $t1, 31
        beq $at, $t1, mult3
        break 0
mult3:  mflo $t1

        multu $t2, $t3          #mulou $t1, $t2, $t3
        mfhi $at
        beq $at, $zero, mult4
        break 0
```

```
mult4:  mflo $t1

        sub $t1, $zero, $t2         #neg $t1, $t2

        subu $t1, $zero, $t2        #negu $t1, $t2

        nor $t1, $t2, $zero         #not $t1, $t2

        bne $t3, $zero, div3        #rem $t1, $t2, $t3
        break 0
div3:   div $t2, $t3
        mfhi $t1

        bne $t3, $zero, div4        #remu $t1, $t2, $t3
        break 0
div4:   divu $t2, $t3
        mfhi $t1

        subu $at, $zero, $t3        #rol $t1, $t2, $t3
        srlv $at, $t2, $at
        sllv $t1, $t2, $t3
        or $t1, $t1, $at

        subu $at, $zero, $t3        #ror $t1, $t2, $t3
        sllv $at, $t2, $at
        srlv $t1, $t2, $t3
        or $t1, $t1, $at

        lui $at, 0x1000             #li $t1, 0x10002000
        ori $t1, $at, 0x2000

        beq $t3, $t2, siga21        #seq $t1, $t2, $t3
        ori $t1, $zero, 0
siga21: beq $zero, $zero, siga2
        ori $t1, $zero, 1
siga2:
        ori $at, $zero, 0x1000      #seq $t1, $t2, 0x1000
        beq $at, $t2, siga1
        ori $t1, $zero, 0
siga1:  beq $zero, $zero, siga3
        ori $t1, $zero, 1
siga3:

        bne $t3, $t2, siga41        #sge $t1, $t2, $t3
        ori $t1, $zero, 1
siga41: beq $zero, $zero, siga4
        slt $t1, $t3, $t2
siga4:

        ori $at, $zero, 0x1000      #sge $t1, $t2, 0x1000
        bne $at, $t2, siga51
        ori $t1, $zero, 1
siga51: beq $zero, $zero, siga5
        slt $t1, $at, $t2
siga5:
```

```
        bne $t3, $t2, siga61        #sgeu $t1, $t2, $t3
        ori $t1, $zero, 1
siga61: beq $zero, $zero, siga6
        sltu $t1, $t3, $t2
siga6:

        ori $at, $zero, 0x1000      #sgeu $t1, $t2, 0x1000
        bne $at, $t2, siga71
        ori $t1, $zero, 1
siga71: beq $zero, $zero, siga7
        sltu $t1, $at, $t2
siga7:

        slt $t1, $t3, $t2           #sgt $t1, $t2, $t3

        ori $at, $zero, 0x1000      #sgt $t1, $t2, 0x1000
        slt $t1, $at, $t2

        sltu $t1, $t3, $t2          #sgtu $t1, $t2, $t3

        ori $at, $zero, 0x1000      #sgtu $t1, $t2, 0x1000
        sltu $t1, $at, $t2

        bne $t3, $t2, siga81        #sle $t1, $t2, $t3
        ori $t1, $zero, 1
siga81: beq $zero, $zero, siga8
        slt $t1, $t2, $t3
siga8:

        ori $at, $zero, 0x1000      #sle $t1, $t2, 0x1000
        bne $at, $t2, siga10
        ori $t1, $zero, 1
siga10: beq $zero, $zero, siga9
        slt $t1, $t2, $at
siga9:

        bne $t3, $t2, siga11a       #sleu $t1, $t2, $t3
        ori $t1, $zero, 1
siga11a:beq $zero, $zero, siga11
        sltu $t1, $t2, $t3
siga11:

        ori $at, $zero, 0x1000      #sleu $t1, $t2, 0x1000
        bne $at, $t2, siga13
        ori $t1, $zero, 1
siga13: beq $zero, $zero,siga12
        sltu $t1, $t2, $at
siga12:

        beq $t3, $t2, siga15        #sne $t1, $t2, $t3
        ori $t1, $zero, 1
siga15: beq $zero, $zero, siga14
        ori $t1, $zero, 0
siga14:

label:  bgez $zero,label            #b label
```

```
beq $t1, $zero, label       #beqz $t1, label

slt $at, $t1, $t2           #bge $t1, $t2, label
beq $at, $zero, label

lui $at, 0x1000             #bge $t1, 0x10002000, label
ori $at, $at, 0x2000
slt $at, $t1, $at
beq $at, $zero, label

sltu $at, $t1, $t2          #bgeu $t1, $t2, label
beq $at, $zero,label

lui $at, 0x1000             #bgeu $t1, 0x10002000, label
ori $at, $at, 0x2000
sltu $at, $t1, $at
beq $at, $zero, label

slt $at, $t2, $t1           #bgt $t1, $t2, label
bne $at, $zero,label

sltu $at, $t2, $t1          #bgtu $t1, $t2, label
bne $at, $zero, label

slt $at, $t2, $t1           #ble $t1, $t2, label
beq $at, $zero, label

sltu $at, $t2, $t1          #bleu $t1, $t2, label
beq $at, $zero, label

slt $at, $t1, $t2           #blt $t1, $t2, label
bne $at, $zero, label

sltu $at, $t1, $t2          #bltu $t1, $t2, label
bne $at, $zero, label

bne $t1, $zero, label       #bnez $t1, label

lui $at, 0x1000             #la $t1, 0x10002000
ori $t1, $at, 0x2000

lui $at, 4                  #ld $t1, 0x10002000300040000
lw $t1, 0($at)
lui $at, 4
lw $t2, 4($at)

lui $at, 0x1000             #ulh $t1, 0x10002000
lb $t1,  0x2001($at)
lui $at, 0x1000
lbu $at, 0x2000($at)
sll $t1, $t1, 8
or $t1, $t1, $at

lui $at, 0x1000             #ulh $t1, 0x10002001
lb $t1,  0x2002($at)
```

```
lui $at, 0x1000
lbu $at, 0x2001($at)
sll $t1, $t1, 8
or $t1, $t1, $at

lui $at, 0x1000          #ulhu $t1, 0x10002000
lbu $t1, 0x2001($at)
lui $at, 0x1000
lbu $at 0x2000($at)
sll $t1, $t1, 8
or $t1, $t1, $at

lui $at, 0x1000          #ulw $t1, 0x10002000
lwl $t1, 0x20003($at)
lui $at, 0x1000
lwr $t1, 0x2000($at)

lui $at, 12288           #sd $t1, 0x1000200030004000
sw $t1, 16384($at)
lui $at, 12288
sw $t2, 16388($at)

lui $at, 0x1000          #ush $t1, 0x10002000
sb $t1, 0x2000($at)
srl $at, $t1, 8
lui $at, 0x1000
sb $at, 0x20001($at)

lui $at, 0x1000          #usw $t1, 0x10002000
swl $t1, 0x2003($at)
lui $at, 0x1000
swr $t1, 0x2000($at)

addu $t1, $zero, $t2     #move $t1, $t2
```