CONJUNTO DE INSTRUCCIONES DE MIPS I (versión del 2-11-2010)

OPERACIONES DE CARGA Y DE ALMACENAMIENTO DESDE REGISTROS DE LA UCP

Sintaxis		Descripción	Codificación binaria
la	rdest,direc	rdest=direc	Pseudoinstrucción
lb	rt,direc	rt=ext_signo(Mem8[direc],32)	0x20(6),rs(5),rt(5),desp(16)
lbu	rt,direc	rt=ext_ceros(Mem8[direc],32)	0x24(6),rs(5),rt(5),desp(16)
ld	rdest,direc	rdest | r(dest+1)=Mem64[direc]	Pseudoinstrucción
lh	rt,direc	rt=ext_signo(Mem16[direc],32)	0x21(6),rs(5),rt(5),desp(16)
lhu	rt,direc	rt=ext_ceros(Mem16[direc],32)	0x25(6),rs(5),rt(5),desp(16)
li	rdest,inm32	rdest=inm32	Pseudoinstrucción
lui	rt,inm16	rt[31..16]=inm16; rt[15..0]=0	0xF(6),0(5),rt(5),inm(16)
lw	rt,direc	rt=Mem32[direc]	0x23(6),rs(5),rt(5),desp(16)
lwcZ	rt,direc	coprocesadorZ(rt)=Mem32[direc]	0xC(4),0xZ(2),rs(5),rt(5),desp(16)
lwl	rt,direc	rt[31..16]=Mem16[direc]	0x22(6),rs(5),rt(5),desp(16)
lwr	rt,direc	rt[15..0]=Mem16[direc]	0x26(6),rs(5),rt(5),desp(16)
sb	rt,direc	Mem8[direc]=rt[7..0]	0x28(6),rs(5),rt(5),desp(16)
sd	rt,direc	Mem64[direc]=rt | r(t+1);	Pseudoinstrucción
sh	rt,direc	Mem16[direc]=rt[15..0]	0x29(6),rs(5),rt(5),desp(16)
sw	rt,direc	Mem32[direc]=rt	0x2B(6),rs(5),rt(5),desp(16)
swcZ	rt,direc	Mem32[direc]=coprocesadorZ(rt)	0xE(4),0xZ(2),rs(5),rt(5),desp(16)
swl	rt,direc	Mem16[direc]=rt[31..16]	0x2A(6),rs(5),rt(5),desp(16)
swr	rt,direc	Mem16[direc]=rt[15..0]	0x2E(6),rs(5),rt(5),desp(16)
ulh	rdest,direc	rdest=ext_signo(Mem16[direc],32)	Pseudoinstrucción
ulhu	rdest,direc	rdest=ext_ceros(Mem16[direc],32)	Pseudoinstrucción
ulw	rdest,direc	rdest=Mem32[direc]	Pseudoinstrucción
ush	rs1,direc	Mem16[direc]=rs1[15..0]	Pseudoinstrucción
usw	rs1,direccion	Mem32[direc]=rs1	Pseudoinstrucción

- En todas las operaciones, **direc** puede ser una etiqueta, una dirección absoluta o **desp(rs)**.
- Todas las operaciones que tienen como segundo operando **direc** son instrucciones sólo si dicho operando se expresa como **desp(rs),**y pseudoinstrucciones si se expresa mediante una etiqueta o dirección absoluta.
- Todas estas operaciones generan excepción si la transferencia es de tamaño palabra o media palabra en caso de que esté desalineada, a excepción de las pseudoinstrucciones ulh, ulhu, ulw, ush y usw.

OPERACIONES DE TRANSFERENCIA ENTRE REGISTROS

Sintaxis		Descripción	Codificación binaria
mfcZ	rt,rd	rt=coprocesadorZ(rd)	0x1Z(6),0(5),rt(5),rd(5),0(11)
mfc1.d	rdest,fs	rdest=fs; r(dest+1)=f(s+1)	Pseudoinstrucción
mfhi	rd	rd=hi	0x0(6),0(10),rd(5),0(5),0x2A(6)
mflo	rd	rd=lo	0x0(6),0(10),rd(5),0(5),0x12(6)
move	rdest,rs1	rdest=rs1	Pseudoinstrucción
mtcZ	rt,rd	coprocesadorZ(rd)=rt	0x1Z(6),4(5),rt(5),rd(5),0(11)
mthi	rs	hi=rs	0x0(6),rs(5),0(15),0x11(6)
mtlo	rs	lo=rs	0x0(6),rs(5),0(15),0x13(6)

- En mfcZ y mtcZ: Z∈{0,1,2,3}, **rt** es un registro UCP, y **rd** es un registro del coprocesador Z
- En mfc1: **fs** y **f(s+1)** son registros de coma flotante.

OPERACIONES LÓGICAS

Sintaxis		Descripción	Codificación binaria
and	rd,rs,rt	rd=rs AND rt	0(6),rs(5),rt(5),rd(5),0(5),0x24(6)
andi	rt,rs,inm16	rt=rs AND ext_ceros(inm16,32)	0xC(6),rs(5),rt(5),inm(16)
nor	rd,rs,rt	rd=rs NOR rt	0(6),rs(5),rt(5),rd(5),0(5),0x27(6)
not	rdest,rs1	rdest=NOT rs1	Pseudoinstrucción
or	rd,rs,rt	rd=rs OR rt	0(6),rs(5),rt(5),rd(5),0(5),0x25(6)
ori	rt,rs,inm16	rt=rs OR ext_ceros(inm16,32)	0xD(6),rs(5),rt(5),inm(16)
xor	rd,rs,rt	rd=rs XOR rt	0(6),rs(5),rt(5),rd(5),0(5),0x26(6)
xori	rt,rs,inm16	rt=rs XOR ext_ceros(inm16,32)	0xE(6),rs(5),rt(5),inm(16)

OPERACIONES ARITMÉTICAS PARA ENTEROS

Sintaxis		Descripción	Codificación binaria
abs	rdest,rs1	rdest=abs(rs1)	Pseudoinstrucción
add	rd,rs,rt	rd=rs+rt	0(6),rs(5),rt(5),rd(5),0(5),0x20(6)
addi	rt,rs,inm16	rt=rs+ext_signo(inm16,32)	0x8(6),rs(5),rt(5),inm(16)
addu	rd,rs,rt	rd=rs+rt	0(6),rs(5),rt(5),rd(5),0(5),0x21(6)
addiu	rt,rs,inm	rt=rs+ext_signo(inm16,32)	0x9(6),rs(5),rt(5),inm(16)
div	rs,rt	lo=rs/rt; hi=rem(rs/rt)	0(6),rs(5),rt(5),0(10),0x1A(6)
div	rdest,rs1,s2	rdest=lo=rs1/s2; hi=rem(rs1,s2)	Pseudoinstrucción
divu	rs,rt	lo=rs/rt; hi=rem(rs/rt)	0(6),rs(5),rt(5),0(10),0x1B(6)
divu	rdest,rs1,s2	rdest=lo=rs1/s2; hi=rem(rs1,s2)	Pseudoinstrucción
mul	rdest,rs1,s2	hi-lo=rs1*s2; rdest=lo	Pseudoinstrucción
mulo	rdest,rs1,s2	hi-lo=rs1*s2; rdest=lo	Pseudoinstrucción
mulou	rdest,rs1,s2	hi-lo=rs1*s2; rdest=lo	Pseudoinstrucción
mult	rs,rt	hi-lo=rs*rt	0(6),rs(5),rt(5),0(10),0x18(6)
multu	rs,rt	hi-lo=rs*rt	0(6),rs(5),rt(5),0(10),0x19(6)
neg	rdest,rs1	rdest=-rs1	Pseudoinstrucción
negu	rdest,rs1	rdest=-rs1	Pseudoinstrucción
rem	rdest,rs1,rs2	lo=rs1/rs2; rdest=hi=rem(rs1,rs2)	Pseudoinstrucción
remu	rdest,rs1,rs2	lo=rs1/rs2; rdest=hi=rem(rs1,rs2)	Pseudoinstrucción
sub	rd,rs,rt	rd=rs-rt	0(6),rs(5),rt(5),rd(5),0(5),0x22(6)
subu	rd,rs,rt	rd=rs-rt	0(6),rs(5),rt(5),rd(5),0(5),0x23(6)

- Excepto **mulou**, los nemotécnicos que terminan con “u” corresponden a operaciones que no generan excepción por desbordamiento. Tampoco la generan **mult** ni **div**.
- Las operaciones **divu**, **mulou**, **multu** y **remu** consideran operandos en binario puro.
- Las restantes operaciones consideran operandos en complemento a 2.

OPERACIONES DE ACTIVACIÓN CONDICIONAL

Sintaxis		Descripción	Codificación binaria
seq	rdest,rs1,s2	Si rs1=s2, rdest=1; si no, rdest=0	Pseudoinstrucción
sge	rdest,rs1,s2	Si rs1>=s2, rdest=1; si no, rdest=0	Pseudoinstrucción
sgeu	rdest,rs1,s2	Si rs1>=s2, rdest=1; si no, rdest=0	Pseudoinstrucción
sgt	rdest,rs1,s2	Si rs1>s2, rdest=1; si no, rdest=0	Pseudoinstrucción
sgtu	rdest,rs1,s2	Si rs1>s2, rdest=1; si no, rdest=0	Pseudoinstrucción
sle	rdest,rs1,s2	Si rs1<=s2, rdest=1; si no, rdest=0	Pseudoinstrucción
sleu	rdest,rs1,s2	Si rs1<=s2, rdest=1; si no, rdest=0	Pseudoinstrucción
slt	rd,rs,rt	Si rs<rt, rd=1; si no, rd=0	0(6),rs(5),rt(5),rd(5),0(5),0x2A(6)
slti	rt,rs,inm16	Si rs<ext_signo(inm16,32),rt=1; si no, rt=0	0xA(6),rs(5),rt(5),inm(16)
sltu	rd,rs,rt	Si rs<rt, rd=1; si no, rd=0	0(6),rs(5),rt(5),rd(5),0(5),0x2B(6)
sltiu	rt,rs,inm16	Si rs<ext_signo(inm16,32),rt=1; si no, rt=0	0xB(6),rs(5),rt(5),inm(16)
sne	rdest,rs1,s2	Si rs1<>s2, rdest=1; si no, rdest=0	Pseudoinstrucción

OPERACIONES DE DESPLAZAMIENTO Y ROTACIÓN

Sintaxis		Descripción	Codificación binaria
rol	rdest,rs1,s2	rdest=rotacion(rs1,s2,izqda)	Pseudoinstrucción
ror	rdest,rs1,s2	rdest=rotacion(rs1,s2,drcha)	Pseudoinstrucción
sll	rd,rt,shamt5	rd=desp_log(rt,shamt5,izqda)	0(6),0(5),rt(5),rd(5),shamt(5),0(6)
sllv	rd,rt,rs	rd=desp_log(rt,rs[4..0],izqda)	0(6),rs(5),rt(5),rd(5),0x4(6)
sra	rd,rt,shamt5	rd=desp_arit(rt,shamt5,drcha)	0(6),0(5),rt(5),rd(5),shamt(5),0x3(6)
srav	rd,rt,rs	rd=desp_arit(rt,rs[4..0],drcha)	0(6),rs(5),rt(5),rd(5),0x7(6)
srl	rd,rt,shamt5	rd=desp_log(rt,shamt5,drcha)	0(6),0(5),rt(5),rd(5),shamt(5),0x2(6)
srlv	rd,rt,rs	rd=desp_log(rt,rs[4..0],drcha)	0(6),rs(5),rt(5),rd(5),0x6(6)

OPERACIONES DE RAMIFICACIÓN

Sintaxis		Descripción	Codificación binaria
b	etiqueta	Ramificar a etiqueta	Pseudoinstrucción
bcZf	etiqueta	Si flag(coprocesadorZ)=0, ramificar a etiqueta	0x1Z(6),0x08(5),0(5),desp(16)
bcZt	etiqueta	Si flag(coprocesadorZ)=1, ramificar a etiqueta	0x1Z(6),0x08(5),0x01(5),desp(16)
beq	rs,rt,etiq	Si rs=rt, ramificar a etiq	0x04(6),rs(5),rt(5),desp(16)
beqz	rs1,etiq	Si rs1=0, ramificar a etiq	Pseudoinstrucción
bge	rs1,s2,etiq	Si rs1>=s2, ramificar a etiq	Pseudoinstrucción
bgeu	rs1,s2,etiq	Si rs1>=s2, ramificar a etiq	Pseudoinstrucción
bgez	rs,etiqueta	Si rd>=0, ramificar a etiqueta	0x01(6),rs(5),0x01(5),desp(16)
bgezal	rs,etiqueta	Si rd>=0, ramificar a etiqueta y enlazar ($ra=PC)	0x01(6),rs(5),0x11(5),desp(16)
bgt	rs1,s2,etiq	Si rs1>s2, ramificar a etiq	Pseudoinstrucción
bgtu	rs1,s2,etiq	Si rs1>s2, ramificar a etiq	Pseudoinstrucción
bgtz	rs,etiqueta	Si rd>0, ramificar a etiq	0x07(6),rs(5),0(5),desp(16)
ble	rs1,s2,etiq	Si rs1<=s2, ramificar a etiq	Pseudoinstrucción
bleu	rs1,s2,etiq	Si rs1<=s2, ramificar a etiq	Pseudoinstrucción
blez	rs,etiqueta	Si rd<=0, ramificar a etiq	0x06(6),rs(5),0(5),desp(16)
blt	rs1,s2,etiq	Si rs1<s2, ramificar a etiq	Pseudoinstrucción
bltu	rs1,s2,etiq	Si rs1<s2, ramificar a etiq	Pseudoinstrucción
bltz	rs,etiq	Si rs<0, ramificar a etiq	0x01(6),rs(5),0(5),desp(16)
bltzal	rs,etiq	Si rs<0, ramificar a etiq y enlazar ($ra=PC)	0x01(6),rs(5),0x10(5),desp(16)
bne	rs,rt,etiq	Si rs<>rt, ramificar a etiq	0x05(6),rs(5),rt(5),desp(16)
bnez	rs1,etiq	Si rs1<>0, ramificar a etiq	Pseudoinstrucción

La etiqueta indica el número de instrucciones que hay que saltarse ($-2^{15} \leq$ **offset** $\leq 2^{15}-1$).

OPERACIONES DE SALTO INCONDICIONAL

Sintaxis		Descripción	Codificación binaria
j	objetivo	PC=PC[31..28] | (objetivo << 2)	0x02(6),target(26)
jal	objetivo	ra=PC; PC=PC[31..28] | (objetivo << 2)	0x03(6),target(26)
jalr	rs,rd	rd=PC; PC=rs	0x0(6),rs(5),0(5),rd(5),0(5),0x09(6)
jr	rs	PC=rs	0x0(6),rs(5),0(15),0x08(6)

OTRAS OPERACIONES

Sintaxis		Descripción	Codificación binaria
rfe		Restaurar desde excepción	0x10(6),1(1),0(19),0x20(6)
syscall		Invocar un servicio del sistema	0x0(6),0(20),0xC(6)
break	codigo20	Provoca una excepción	0x0(6),código(20),0xD(6)
nop		No operación	0(32)

OPERACIONES DE CARGA Y DE ALMACENAMIENTO CON REGISTROS DE COMA FLOTANTE

Sintaxis		Descripción	Codificación binaria
l.d	fdest,direc	fdest=Mem64[direc]	Pseudoinstrucción
l.s	fdest,direc	fdest=Mem32[direc]	Pseudoinstrucción
s.d	fs1,direc	Mem64[direccion]=fs1	Pseudoinstrucción
s.s	fs1,direc	Mem32[direccion]=fs1	Pseudoinstrucción

- En todos los casos, **direc** puede ser una etiqueta, una dirección absoluta o **desp(rs)**.
- Sufijo **.d**: operandos en coma flotante precisión doble
- Sufijo **.s**: operandos en coma flotante de precisión simple

Significado de los símbolos utilizados

rd, rs, rt, rs1, rs2, r(t+1), r(dest+1): registros de la UCP
fd, fs, ft, f(s+1), fdest, fs1: registros de coma flotante
direc: puede ser una etiqueta, una dirección completa o desp(rs)
s2: puede ser un registro o un dato inmediato
shamt5: longitud del desplazamiento (5 bits)
[i..j]: subrango de bits desde el i hasta el j, ambos inclusive

OPERACIONES DE TRANSFERENCIA ENTRE REGISTROS DE COMA FLOTANTE

Sintaxis		Descripción	Codificación binaria
mov.d	fd,fs	fd=fs	0x11(6),0x01(5),0(5),fs(5),fd(5),0x06
mov.s	fd,fs	fd=fs	0x11(6),0x00(5),0(5),fs(5),fd(5),0x06

- Sufijo **.d**: operandos en coma flotante precisión doble
- Sufijo **.s**: operandos en coma flotante de precisión simple

OPERACIONES ARITMÉTICAS PARA COMA FLOTANTE

Sintaxis		Descripción	Codificación binaria
abs.d	fd,fs	fd=abs(fs)	0x11(6),0x01(5),0(5),fs(5),fd(5),0x05(6)
abs.s	fd,fs	fd=abs(fs)	0x11(6),0x00(5),0(5),fs(5),fd(5),0x05(6)
add.d	fd,fs,ft	fd=fs+ft	0x11(6),0x01(5),ft(5),fs(5),fd(5),0x00(6)
add.s	fd,fs,ft	fd=fs+ft	0x11(6),0x00(5),ft(5),fs(5),fd(5),0x00(6)
div.d	fd,fs,ft	fd=fs/ft	0x11(6),0x01(5),ft(5),fs(5),fd(5),0x03(6)
div.s	fd,fs,ft	fd=fs/ft	0x11(6),0x00(5),ft(5),fs(5),fd(5),0x03(6)
mul.d	fd,fs,ft	fd=fs*ft	0x11(6),0x01(5),ft(5),fs(5),fd(5),0x02(6)
mul.s	fd,fs,ft	fd=fs*ft	0x11(6),0x00(5),ft(5),fs(5),fd(5),0x02(6)
neg.d	fd,fs	fd=-fs	0x11(6),0x01(5),0(5),fs(5),fd(5),0x07(6)
neg.s	fd,fs	fd=-fs	0x11(6),0x00(5),0(5),fs(5),fd(5),0x07(6)
sub.d	fd,fs,ft	fd=fs-ft	0x11(6),0x01(5),ft(5),fs(5),fd(5),0x01(6)
sub.s	fd,fs,ft	fd=fs-ft	0x11(6),0x00(5),ft(5),fs(5),fd(5),0x01(6)

- Sufijo **.d**: operandos en coma flotante precisión doble
- Sufijo **.s**: operandos en coma flotante de precisión simple

OPERACIONES DE CONVERSIÓN ENTRE ENTEROS Y COMA FLOTANTE

Sintaxis		Descripción	Codificación binaria
cvt.d.s	fd,fs	fd=convertir(fs)	0x11(6),0x01(5),0(5),fs(5),fd(5),0x21(6)
cvt.d.w	fd,fs	fd=convertir(fs)	0x11(6),0x00(5),0(5),fs(5),fd(5),0x21(6)
cvt.s.d	fd,fs	fd=convertir(fs)	0x11(6),0x01(5),0(5),fs(5),fd(5),0x20(6)
cvt.s.w	fd,fs	fd=convertir(fs)	0x11(6),0x00(5),0(5),fs(5),fd(5),0x20(6)
cvt.w.d	fd,fs	fd=convertir(fs)	0x11(6),0x01(5),0(5),fs(5),fd(5),0x24(6)
cvt.w.s	fd,fs	fd=convertir(fs)	0x11(6),0x00(5),0(5),fs(5),fd(5),0x24(6)

- Sufijo **.d**: operando en coma flotante precisión doble
- Sufijo **.s**: operando en coma flotante de precisión simple
- Sufijo **.w**: entero.
- Primero va el sufijo del formato del destino, y después el del origen.

OPERACIONES DE COMPARACIÓN PARA COMA FLOTANTE

Sintaxis		Descripción	Codificación binaria
c.eq.d	fs,ft	Si fs=ft, flag=1; si no, flag=0	0x11(6),0x01(5),ft(5),fs(5),fd(5),0x32(6)
c.eq.s	fs,ft	Si fs=ft, flag=1; si no, flag=0	0x11(6),0x00(5),ft(5),fs(5),fd(5),0x32(6)
c.le.d	fs,ft	Si fs<=ft, flag=1; si no, flag=0	0x11(6),0x01(5),ft(5),fs(5),fd(5),0x3E(6)
c.le.s	fs,ft	Si fs<=ft, flag=1; si no, flag=0	0x11(6),0x00(5),ft(5),fs(5),fd(5),0x3E(6)
c.lt.d	fs,ft	Si fs<ft, flag=1; si no, flag=0	0x11(6),0x01(5),ft(5),fs(5),fd(5),0x3C(6)
c.lt.s	fs,ft	Si fs<ft, flag=1; si no, flag=0	0x11(6),0x00(5),ft(5),fs(5),fd(5),0x3C(6)

- Sufijo **.d**: operandos en coma flotante precisión doble
- Sufijo **.s**: operandos en coma flotante de precisión simple
- Estas instrucciones actualizan el flag del coprocesador 1 de coma flotante. Dicho flag puede usarse en las instrucciones de ramificación **bc1f** y **bc1t**.

RESUMEN DE LOS SYSCALLS DEL SIMULADOR MARS (VER LA AYUDA DEL PROGRAMA PARA UN LISTADO COMPLETO)

Paso 1: Cargar el número de servicio en el registro $v0.
Paso 2: Cargar los valores de los parámetros, si los hay, en $a0, $a1, etc.
Paso 3: Ejecutar la instrucción syscall.
Paso 4: Recoger los valores de retorno, si lo hay, de los registros de resultado.

Servicio	$v0	Parámetros	Resultado
imprimir número entero	1	$a0 = número entero a imprimir	
imprimir número flotante	2	$f12 = número flotante a imprimir	
imprimir número doble	3	$f12 = número doble a imprimir	
imprimir cadena	4	$a0 = dirección de la cadena terminada en el byte nulo a imprimir	
leer número entero	5		$v0 contiene el número entero leído
leer número flotante	6		$f0 contiene el número flotante leído
leer número doble	7		$f0 contiene el número doble leído
leer cadena	8	$a0 = dirección del buffer de entrada (variable que almacenará la cadena) $a1 = número máximo de caracteres que se leerán. Para un tamaño n, la cadena sólo podrá tener n-1 caracteres. Si es menos que eso, la cadena termina en el byte 0x10 (nueva línea) y se rellena luego con el byte nulo	
terminar la ejecución del programa	10		
imprimir carácter	11	$a0 = carácter a imprimir (byte menos significativo)	
leer carácter	12		$v0 contiene el carácter leído
abrir fichero	13	$a0 = dirección de la cadena terminada en el byte nulo que contiene el nombre del fichero $a1 = indicadores (0: solo lectura, 1: solo escritura con creación de nuevo fichero) 9: escritura para añadir nuevo contenido al final $a2 = modo (se ignora su valor)	$v0 contiene el identificador de fichero (negativo si hubo error)
leer fichero	14	$a0 = identificador de fichero $a1 = dirección del buffer de entrada (variable que almacenará el contenido del fichero) $a2 = número máximo de caracteres a leer	$v0 contiene el número de caracteres leídos (0 si era final-de-fichero, negativo si hubo error)
escribir fichero	15	$a0 = identificador de fichero $a1 = dirección del buffer de salida (variable que contiene lo que se quiere escribir en el fichero) $a2 = número de caracteres a escribir	$v0 contiene el número de caracteres escritos (negativo si hubo error)
cerrar fichero	16	$a0 = identificador de fichero	
imprimir entero en hexadecimal	34	$a0 = número entero a imprimir	
imprimir entero en binario	35	$a0 = número entero a imprimir	
imprimir entero sin signo	36	$a0 = número entero a imprimir	