Set de Instrucciones de la unidad de enteros

Todas las instrucciones tienen un tamaño fijo de 32 bits. Para simplificar la descripción del conjunto de instrucciones, usaremos la siguiente notación:

Notación	Significado
reg	Un registro de la unidad de números enteros (%r0 - %r31). El subíndice indica el campo que ocupa el registro en la instrucción. reg_{rd} = registro de destino. reg_{ro} = registro de origen
const	Una constante sin signo. El subíndice indica el tamaño de la constante en bits.
disp	Desplazamiento entero en notación complemente. El subíndice indica el tamño en bits.
siconst	Una constante con signo. El subíndice indica el tamaño en bits.
dir	Una dirección que se obtendrá de alguna de las siguientes formas: reg_{ro1} $reg_{ro1} + reg_{ro2}$ $reg_{ro1} + siconst_{13}$ $reg_{ro1} - siconst_{13}$ $siconst_{13}$ $siconst_{13} + reg_{ro1}$
regdir	Una dirección que se obtendrá de alguna de las siguientes formas: reg_{ro1} $reg_{ro1} + reg_{ro2}$

Instrucciones de Acceso a Memoria

Sintaxis del lenguaje ensamblador

operación reg_{rd} , $regr_{o2}$, reg_{ro1}	$dir = reg_{o1} + reg_{o2}$
operación reg_{rd} , $siconst_{13}$, reg_{ro1}	$dir = reg_{o1} + siconst_{13}$

Formato de la instrucción

31:30	29:25	24:19	18:14	13	12:5	4:0
11	rd	op3	ro1	0	cero	ro2
11	rd	op3	ro1	1	$siconst_{13}$	

Descripción de las operaciones

Las instrucciones de acceso a memoria permiten operandos de diferenets tamaños:

Nombre	Tamaño
byte	8 bits
media palabra	16 bits
palabra	32 bits

Las operaciones de lectura y escritura requieren que las medias palabras estén alineadas en una dirección par (alineación a media palabra) y que las palabras y palabras dobles estén alineadas en direcciones múltiplos de 4 (alineación a palabra). La operación de lectura es *ld* y la de escritura *st*. A las operaciones se les agregará un sufijo para indicar el tamaño del operando y si el operando es un operando con signo o sin signo.

Tamaño	Lectura	Escritura	Notas
byte	ldsb	stb	
media palabra	ldsh	sth	La dirección debe ser par.
palabra	ld	st	La dirección debe ser múltiplo de 4

En la ta anterior, la letra *s* en el sufijo indica que se realizará una extensión del signo al leer el dato desde la memoria. Para indicar una operación en la que no es extiende el signo se utilizará la letra *u*.

Operación	**op3**	Descripción
stb	000101	$mem[dir]_8 = reg[rd]_8$ Guarda un byte en memoria.
sth	000110	$mem[dir]_{16} = reg[rd]_{16}$ Guarda media palabra en memoria.
st	000100	$mem[dir]_{32} = reg[rd]_{32}$ Guarda una palabra entera en memoria.
ldsb	001001	reg[rd] = signextend ($mem[dir]_8$) Lee un byte de memoria. Extiende el signo del byte a los bits 8- 31 de reg(rd).

ldsh	001010	reg[rd] = signextend(mem[*dir*]$_{16}$) Lee media palabra de memoria. Extiende el signo del byte a los bits 15 – 31.
ldub	000001	reg[rd] = zerofill(mem[*dir*]$_{8}$) Lee un byte de memoria. Los bits 8- 31 de reg[rd] se ponen en 0.
lduh	000010	reg[rd] = zerofill(mem[*dir*]$_{16}$) Lee media palabra de memoria. Los bits 16- 31 de reg[rd] se ponen en 0.
ld	000000	reg[rd] = mem[*dir*]$_{32}$ Lee una palabra de memoria.

Ejemplos:

Instrucciones SETHI

sethi *const*$_{22}$, *reg*$_{rd}$

31:30	29:25	24:22	21:0
00	rd	100	const$_{22}$

Descripción de la operación

rd$_{[0..9]}$ = 0 rd$_{[10..31]}$ = const$_{22}$

Guarda la constante const$_{22}$ en los 22 bits más altos del registro de destino. Los 10 bits menos significativos son puestos en 0.

Ejemplos:

Instrucciones Lógicas y Aritméticas

operación reg$_{rd}$, *reg*$_{ro2,}$ *regr*$_{o1}$
operación reg$_{rd}$, *siconst*$_{13}$, *reg*$_{ro1}$

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	op3	ro1	0	Cero	ro2
10	rd	op3	ro1	1	siconst$_{13}$	

Descripción de las operaciones

Las operaciones lógicas y aritméticas pueden o no modificar la palabra de estado. Aquellas con el sufijo *cc* modifican los la palabra de estado; las que carecen del sufijo *cc* no la modifcarán. Las operaciones de multiplicación entera multiplican dos valores de 32 bits y producen un resultado de 64 bits. Los 32 bits más significativos del resultado se guardarán en el registro Y (%y). Las operaciones de división entera dividen un valor de 64 bits por otro de 32 bits y producen un resultado de 32 bits. El registro Y (%y) provee los 32 bits más significativos del dividendo. Uno de los registros de provee los 32 bits menos significativos del dividendo y el otro los 32 bits del divisor. El resto de la división se guarda en el registro Y.

Códigos de condición:
Las instrucciones andcc, orcc, xorcc, andncc, orncc y xnorcc, establecen los códigos de condición V y C en cero. Si el resultado es cero, el flag Z se prende; en caso contrario, se apaga. El flag N toma el valor del bit más significativo del resultado.

Las instrucciones de multiplicación (smulcc y umulcc) ponen los flag V y C en cero. Modifican los flag Z y N, como se describió en el párrafo anterior, pero únicamente tienen en cuenta los 32 bits menos significativos del resultado (se ignora el contenido del registro Y).

Las instrucciones de suma y resta, modifican los flags N y Z de la misma manera que las instrucción lógicas, y además prenden el flag C si se produce un acarreo desde el bit más significativo (caso contrario apagan el flag) y prenden el flag V si el resultado no se puede expresar como una constante con signo de 32 bits.

Notas: En las operaciones de multiplicación y división, *iconst* representa una constante de 13 bits que se interpretará con signo cuando se trabaja con operaciones con signo (smul y sdiv) y como una constante sin signo cuando se trabaja con operaciones sin signo (udiv y umul).

Operación	**op3**	Descripción
add addcc	000000 010000	reg[rd] = reg[ro1] + reg[ro2] reg[rd] = reg[ro1] + uiconst$_{13}$.
and andcc	000001 010001	reg[rd] = reg[ro1] AND reg[ro2] reg[rd] = reg[ro1] AND uiconst13
andn andncc	000101 010101	reg[rd] = reg[ro1] AND NOT reg[ro2] reg[rd] = reg[ro1] AND NOT uiconst13

or orcc	000010 010010	reg[rd] = reg[ro1] OR reg[ro2] reg[rd] = reg[ro1] OR uiconst13
orn orncc	000110 010110	reg[rd] = reg[ro1] OR NOT reg[ro2] reg[rd] = reg[ro1] OR NOT uiconst13
udiv udivcc	001110 011110	{reg [rd] , %y} = { %y, reg[ro1]} / reg[ro2] {reg [rd] , %y} = { %y, reg[ro1]} / u$iconst_{13}$
umul umulcc	001010 011010	{%y, reg[rd]} = reg[ro1] x reg[ro2] {%y, reg[rd]} = reg[ro1] x u$iconst_{13}$
sdiv sdivcc	001111 011111	{reg [rd] , %y} = { %y, reg[ro1]} / reg[ro2] {reg [rd] , %y} = { %y, reg[ro1]} / u$iconst_{13}$
smul smulcc	001011 011011	{%y, reg[rd]} = reg[ro1] x reg[ro2] {%y, reg[rd]} = reg[ro1] x u$iconst_{13}$
sub subcc	000100 010100	reg[rd] = reg[ro1] – reg[ro2] reg[rd] = reg[ro1] – uiconst$_{13}$).
xor xorcc	000011 010011	reg[rd] = reg[ro1] XOR reg[ro2] reg[rd] = reg[ro1] XOR uiconst13
xnor xnorcc	000111 010111	reg[rd] = reg[ro1] XOR NOT reg[ro2] reg[rd] = reg[ro1] XOR NOT uiconst13

Instrucciones de decalaje

operación reg_{rd} , $regr_{o2}$, reg_{ro1}
operación reg_{rd} , $siconst_5$, reg_{ro1}

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	op3	ro1	0	Cero	ro2
10	rd	op3	ro1	1	se ignoran	const$_5$

Descripción de las operaciones

Las instrucciones de decalaje no modifican los código de condición.

Operación	**op3**	Descripción
sll	100101	reg[rd] = reg[ro1] << reg[ro2] reg[rd] = reg[ro1] << const$_5$ decalaje hacia la izquierda lógico.
sra	100111	reg[rd] = reg[ro1]$_{31}$ OR reg[ro1] >> reg[ro2] reg[rd] = reg[ro1]$_{31}$ OR reg[ro1] >> const$_5$
srl	100110	reg[rd] = reg[ro1] >> reg[ro2] reg[rd] = reg[ro1] >> const$_5$ decalaje hacia la derecha lógico.

Instrucciones de Salto Relativo

operación $disp_{22}$
operación,a $disp_{22}$

31:30	29	28:25	24::22	21:0
00	a	cond	010	disp$_{22}$

Descripción de las operaciones

Los saltos son relativos y con signo. Al terminar un ciclo de instrucción la asignación la actualización del nPC se lleva a cabo de la siguiente manera:

PC = nPC
Si se toma el salto: nPC = nPC + disp$_{22}$ * 4
Si no se toma el salto: nPC = nPC + 4

La arquitectura implementa saltos retardados. Por omisión, la instrucción que sigue a la instrucción del salto es ejecutada cuando se ejecuta el salto.

Cada instrucción de salto puede especificar que el efecto de la instrucción siguiente sea anulado si el salto condicional no es tomado. El bit *a* indica si la instrucción "retardada" debe ser o no anulada.

Operación	Cond	Descripción	Condición de salto.
bn	0000	Nunca.	1
ba	1000	Siempre.	0
be	0001	Igual	Z
bne	1001	No Igual	not Z
ble	0010	Menor o Igual	Z or (N xor V)
bg	1010	Mayor	not (Z or (N xor V))
bl	0011	Menor	N xor V
bge	1011	Mayor o Igual	not (N xor V)
bleu	0100	Menor o Igual sin signo	C or Z
bgu	1100	Mayor sin signo	not (C or Z)
bcs	0101	Carry Prendido	C
bcc	1101	Carry Apagado	not C
bneg	0110	Negativo Prendido	N
bpos	1110	Negativo Apagado	not N
bvs	0111	Overflow Prendido	V
bvc	1111	Overflow Apagado	not V

Instrucciones de Salto Absoluto

jmp	reg_{rd}, reg_{ro2}, reg_{ro1}	$dir = reg_{ro1} + reg_{ro2}$
jmp	reg_{rd}, $siconst_{13}$, reg_{ro1}	$dir = reg_{ro1} + siconst_{13}$

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	111000	ro1	0	Cero	ro2
10	rd	111000	ro1	1	$siconst_{13}$	

Descripción de las operaciones

Esta instrucción actualiza los contadores de programa de la siguiente manera:

PC = nPC
nPC = *dir*
reg[rd] = *dir*

Instrucción regreso de un trap

rett	reg_{rd}, reg_{ro2}, reg_{ro1}	$dir = reg_{ro1} + reg_{ro2}$
rett	reg_{rd}, $siconst_{13}$, reg_{ro1}	$dir = reg_{ro1} + siconst_{13}$

31:30	29:25	24:19	18:14	13	12:5	4:0
10	00000	111001	ro1	0	Cero	ro2
10	00000	111001	ro1	1	$siconst_{13}$	

Descripción de las operaciones

Esta instrucción actualiza los contadores de programa de la siguiente manera:

INC (CWP)	' Se regresa a la ventana de registros utilizada por el programa anterior.
ET=1	' Se habilitan los traps.
nPC=Address	' Se setea como Next Program Counter la dirección pasada como parámetro a la instrucción RET.
S=PS	' Se regresa al modo anterior al trap.

Instrucción de llamada a un procedimiento

call $disp_{30}$

31:30	29:0
01	$disp_{30}$

Descripción de las operaciones

La instrucción call realiza un salto relativo y guarda la dirección actual en %o7:

%o7 = PC
PC = nPC
nPC = nPC + $disp_{30}$ * 4

La instrucción ret no existe como tal. Se implementa ejecutando:

jmp %g0, 8, %i7

Instrucciones de desplazamiento de la ventana de registros

operación reg_{rd} , $regr_{o1}$, reg_{ro2}
operación reg_{rd} , $siconst_{13}$, reg_{ro1}

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	op3	ro1	0	Cero	ro2
10	rd	op3	ro1	1	$siconst_{13}$	

Descripción de las operaciones

Operación	**Op3**	Descripción
restore	111101	res = reg[ro1] + reg[ro2] *o bien* res = reg[ro1] + $siconst_{13}$ CWP = (CWP + 1) mod 32 reg[rd] = res
save	111100	res = reg[ro1] + reg[ro2] *o bien* res = reg[ro1] + $siconst_{13}$ CWP = (CWP – 1) mod 32 reg[rd] = res

Notas: El registro de destino se calcula una vez que se ha desplazado la ventana de registros.

Instrucciones de requermiento al supervisor (trap)

operación $regr_{o1}$, reg_{ro2}
operación $siconst_{13}$, reg_{ro1}

31:30	29	28:25	24:19	18:14	13	12:5	4:0
10	r	cond	111010	ro1	0	Cero	ro2
10	r	cond	111010	ro1	1	$siconst_{13}$	

Descripción de las operaciones

Operación	**Cond**	Descripción	Condición de salto.
tn	0000	Nunca.	1
ta	1000	Siempre.	0
te	0001	Igual	Z
tne	1001	No Igual	not Z
tle	0010	Menor o Igual	Z or (N xor V)
tg	1010	Mayor	not (Z or (N xor V))
tl	0011	Menor	N xor V
tge	1011	Mayor o Igual	not (N xor V)
tleu	0100	Menor o Igual sin signo	C or Z
tgu	1100	Mayor sin signo	not (C or Z)
tcs	0101	Carry Prendido	C
tcc	1101	Carry Apagado	not C
tneg	0110	Negativo Prendido	N
tpos	1110	Negativo Apagado	not N
tvs	0111	Overflow Prendido	V
tvc	1111	Overflow Apagado	not V

Instrucción NOP

nop

31:30	29:25	24::22	21:0
00	00000	100	0000000000000000000000

Instrucciones de Lectura de Registros de Estado

rd reg_{rd}, *statereg*

31:30	29:25	24:19	18:14	13:0
10	rd	op3	ro1	cero

Codificación de los registros

Operación	**op3**	ro1
%y	101000	0
%psr	101001	reservados
%wim	101010	reservados
%tbr	101011	reservados
%base	101000	1
%limite	101000	2

Instrucciones de Escritura de Registros de Estado

wr *statereg*, reg_{ro1} , reg_{ro2}
wr *statereg*, $siconst_{13}$, reg_{ro1}

31:30	29:25	24:19	18:14	13	12:5	4:0
10	rd	Op3	ro1	0	Cero	ro2
10	rd	Op3	ro1	1	$siconst_{13}$	

Codificación de los registros

Operación	**Op3**	ro1
%y	111000	0
%psr	110001	reservados
%wim	110010	reservados
%tbr	110011	reservados
%base	101000	1
%limite	101000	2

El valor que se escribirá a memoria es: reg_{ro1} *XOR* reg_{ro2}