Arquitectura del modelo

Introducción

El diseño de la arquitectura del modelo se basa fuertemente en la especificación del *Integer Unit* del procesador SPARC de Sun Microsystems. Se trata de la misma arquitectura RISC, a la que se le han simplificado principalmente el set de instrucciones y el manejo de la memoria.

Para una descripción general de la arquitectura RISC y SPARC, se recomienda leer el capítulo 8 de *Structured Computer Organization* de Tanenbaum. También se recomienda leer una descripción detallada de las componentes de la arquitectura en el capítulo 4 del mismo libro.

Memoria

La memoria del modelo es plana, al no proveer mecanismos de paginación o segmentación. No soporta multiprogramación. La implementación incluye dos registros, BASE y LIMITE, que determinan el espacio de direcciones disponible para el programa.

Una dirección de memoria absoulta se determina sumando, a la dirección relativa utilizada en el programa, el valor del registro BASE.

Registros de uso general

La arquitectura tiene 520 registros de uso de enteros, organizados en ventanas que se superponen. En un instante dado, sólo 32 registros están disponibles. Estos 32 registros serán los que el usuario podrá utilizar y se notarán de dos formas diferentes:

Nombre	Nombre Alternativo	Descripción del grupo
%r0 - % r7	%g0 - % g7	Registros globales
%r8 - % r15	%o0 - % o7	Registros de salida

% r16 - %r 23	%l0 - %l7	Registros locales
%r24 - % r31	% i0 - % i7	Registros de entrada

El nombre alternativo refleja el uso de los registros al usar el esquema de ventanas para invocar un procedimiento. Los registros globales (%go - %g7) son compartidos por todos los procedimientos. Los registros de salida (%o0 - %o7) se utilizan para pasar parámetros al llamar a un procedimiento, del procedimiento llamador al procedimiento llamado. Los registros locales se utilizan para guardar variables locales, y los registros de entrada (%i0 - %i7) se utilizan para recibir los parámetros del procedimiento llamador.

Este esquema se implementa con un arreglo de 8 registros que guardará los registros globales, y otro de 512 registros, donde una ventana de 24 registros alojará los registros de salida, los locales y los de entrada. Cuando un procedimiento inicia su ejecución reservará 16 nuevos registros (utilizando la instrucción *save* que se describe más adelante) para sus registros de salida y locales, y utilizará como registros de entrada los registros de salida del procedimiento anterior, como se muestra en la **Figura 1**.

En la figura 1 se introduce una variable llamada CWP. En la arquitectura esta variable es de 5 bits y se utiliza para indicar, dentro del arreglo de 512 registros, cual es la ventana activa. Cada vez que se desplaza la una ventana mediante la instrucción *save*, la variable CWP se decrementa en 1. La organización de las ventanas de registros es circular. Esto quiere decir que, si al ejecutar la instrucción *save* CWP = 0, entonces, luego de la instrucción *save* CWP = 31. Para evitar sobreescribir una ventana de registros que ya está siendo utilizada por un procedimiento (esto podría ocurrir, por ejemplo, en una recurrencia de profundidad mayor a 32), la arquitectura tiene un registro de 32 bits llamado WIM (Window Invalid Mask). El WIM tiene un bit por ventana. Cuando el CWP se decrementa para establecer una nueva ventana, el hardware verifica si el WIM de la nueva ventana esta prendido. De estarlo, se produce un trap. La rutina de atención del trap deberá salvar el contenido de la ventana que se va a sobreescribir en memoria antes de seguir. Normalmente, el WIM contiene todos sus bits en 0 excepto por un bit de valor 1 que marcando la ventana más antigua. Cuando se alcanza esa ventana, se invocará un trap, y el WIM deberá rotarse una unidad para marcar la nueva ventana más vieja.

En la figura 1 también se introducen dos nuevos nombres para los registros %o6 y %i6. El registro %o6 se puede conocer también como %sp y se utiliza (como convención, no es forzado por el hardware) para guardar el stack pointer. El registro %i6 se puede llamar también %fp, y se utiliza para guardar el frame pointer.

Otros registros de uso especial durante la llamada a un procedimiento son el %o7 y el %i7. El registros %o7 es utilizado por el hardware para guardar la dirección a donde deberá volver el procedimiento llamado al terminar su ejecución. El registro %i7 se utilizará por el procedimiento llamado para obtener la dirección de retorno (ver la descripción detallada de las instrucciones *call* y *ret*)

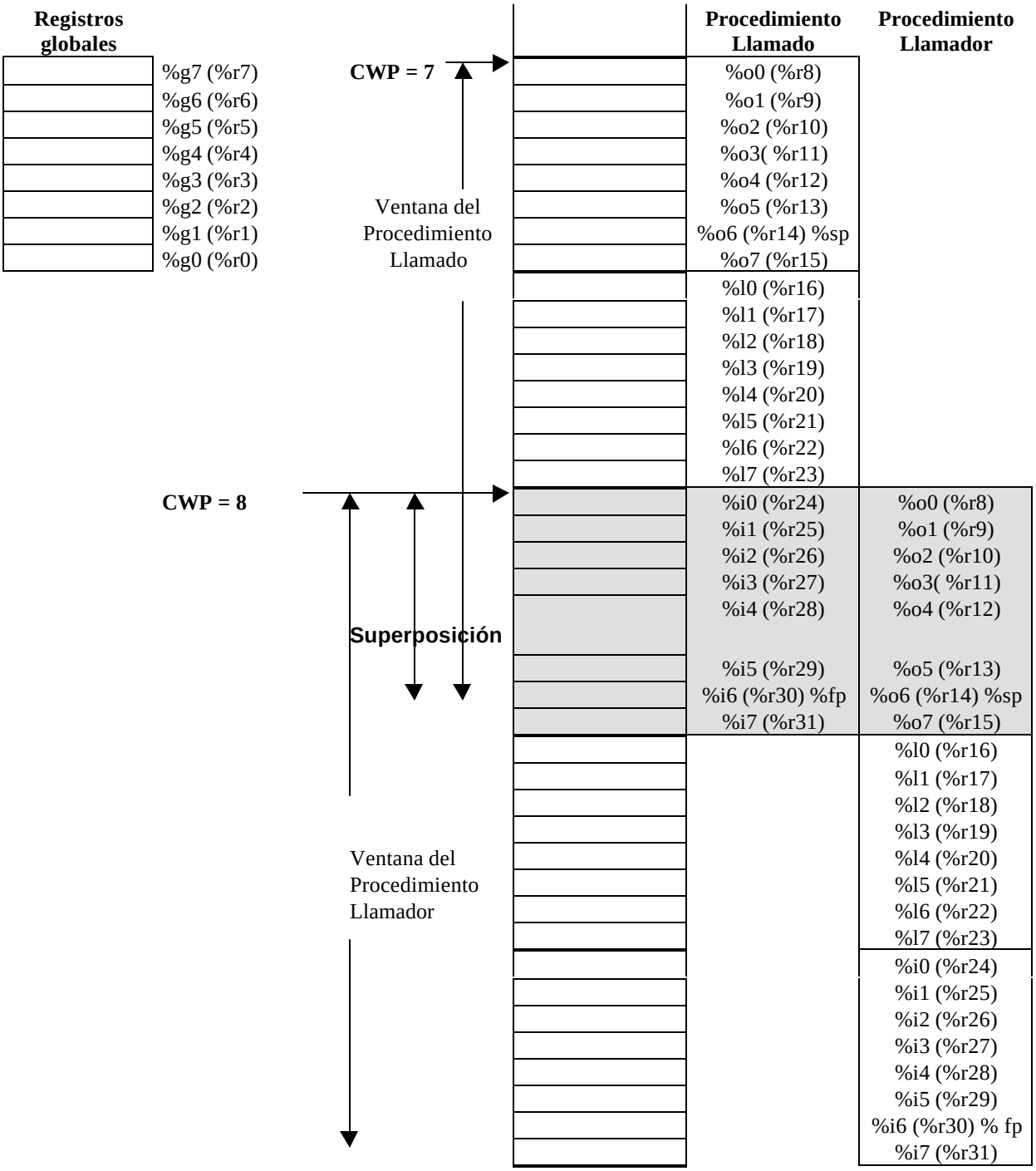


Figura 1: Ventana de registros

Registros Especiales

Además de los registros de uso general, contenidos en la ventana de registros, existen 6 registros especiales:

Registro Y

Utilizado por las operaciones de multiplicación y división. (Ver *Set de Instrucciones*)

Registro WIM (Window Invalid Mask)

Ver apartado sobre *Registros de uso general*.

Registro TBR (Trap Base Register)

Se utiliza para apuntar a la dirección de memoria, dentro de la tabla de traps, donde se encuentre la rutina correspondiente al trap que se debe atender. (Ver *Atención de Interrupciones*)

Registros BASE y LIMITE

El registro BASE apunta a la menor dirección absoluta accesible por el programa actual. El registro LIMITE guarda el tamaño máximo disponible para dicho programa. De esta forma, el espacio de direcciones absolutas accesible en cada momento por un programa en *modo usuario* es el comprendido entre las direcciones BASE y BASE+LIMITE. (Ver *Memoria*)

Registro PSR (Processor Status Register)

Este registro conserva el estado del programa actual. Su interpretación es la siguiente:

Bits	Contenido	Descripción
31..24	Reservado	
23	N – Negative	Flag en 1 si la última operación arrojó resultado negativo
22	Z – Zero	Flag en 1 si la última operación arrojó resultado cero
21	V – Overflow	Flag en 1 si la última operación provocó overflow
20	C – Carry	Flag en 1 si la última operación acarreó un bit.
19..12	Reservado	
11..8	PIL – Processor Interrupt Level	Contiene el mínimo número de interrupción que se debe atender.
7	S – State	1=Modo Supervisor 0=Modo Usuario
6	PS – Previous State	Ultimo estado (modo) en el que estuvo el PSR (Ver sección “Atención de interrupciones”)
5	ET – Enable Trap	1=Traps permitidos 0=Traps deshabilitados
4..0	CWP – Current Window Pointer	Apunta a la ventana de registros actual

Program Counters

El procesador trabaja con dos program counters: PC (Program Counter), que contiene la dirección de la próxima instrucción a utilizar y nPC (Next Program Counter), que conserva el próximo valor del PC.

Cada ciclo de instrucción normal termina copiando el nPC al PC y sumándole 4 al nPC.

Cuando la instrucción es un branch, asigna nPC a PC y actualiza nPC con la dirección a la cual bifurca (si la condición de bifurcación se cumple).

Esta implementación de los Program Counters resulta en que la instrucción inmediatamente posterior a un branch *siempre* se ejecuta.

Modos

El procesador funciona en 2 modos: Supervisor y Usuario. Ciertas instrucciones sólo podrán ejecutarse en modo Supervisor. Asimismo, los registros BASE y LIMITE descritos anteriormente, sólo tendrán efecto bajo el modo Usuario.

Atención de interrupciones

Los pedidos de interrupción son recibidos por el procesador junto con el código del trap a atender.

Para localizar la dirección de la rutina de interrupción, se utiliza el TBR, cuya estructura es la siguiente:

Bits	Contenido	Descripción
31..12	Trap base address	Dirección base de la tabla de atención de traps.
11..4	Trap Type	Trap a atender.
3..0	Constante 0000	

Los primeros 20 bits (trap base address), con los otros 12 bits en cero, nos da la dirección base de la tabla de atención de traps. Ubicando en los bits 11..4 el tipo de trap a atender, el TBR apunta a la dirección de la rutina de atención del mismo. Los últimos 4 bits en cero nos aseguran un espacio de 16 bytes en los cuales guardar la rutina de atención de cada trap.

Se detalla aquí lo que ocurre en la atención de una interrupción y al salir de ella.

Ocurre Trap:

```
DEC (CWP)       ‘ Se cambia a la siguiente ventana de registros.
ET=0            ‘ Se deshabilitan los traps.
PS=S            ‘ Se guarda en Previous State si al entrar al trap se estaba en modo
                  supervisor o usuario
R17=PC          ‘ Se guarda en el registro 17 (Local 1) el Program Counter actual.
R18=nPC         ‘ Se guarda en el registro 18 (Local 2) el Next Program Counter actual.
S=1             ‘ Se setea en modo supervisor.
PC=TBR          ‘ Se setea como nuevo Program Counter la dirección base de la tabla de
                  registros
```

Return from Trap:

```
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.
```

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

Sintaxis del lenguaje ensamdor

sethi $const_{22}$, reg_{rd}

Formato de la instrucción

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

Sintaxis del lenguaje ensamdor

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

Formato de la instrucción

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

Ejemplos:

Instrucciones de decalaje

Sintaxis del lenguaje ensamdor

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

Formato de la instrucción

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.

Ejemplos:

Instrucciones de Salto Relativo

Sintaxis del lenguaje ensamdor

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

Formato de la instrucción

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

Ejemplos:

Instrucciones de Salto Absoluto

Sintaxis del lenguaje ensamdor

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

Formato de la instrucción

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

Sintaxis del lenguaje ensamdor

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

Formato de la instrucción

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.
```

Ver apartado *Atención de Interrupciones.*

Instrucción de llamada a un procedimiento

Sintaxis del lenguaje ensamdor

call $disp_{30}$

Formato de la instrucción

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

Sintaxis del lenguaje ensamdor

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

Formato de la instrucción

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)

Sintaxis del lenguaje ensamdor

operación $regr_{o1}$, reg_{ro2}

operación $siconst_{13}$, reg_{ro1}

Formato de la instrucción

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

Ver apartado *Atención de Interrupciones*

Instrucción NOP

Sintaxis del lenguaje ensamdor

nop

Formato de la instrucción

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

Instrucciones de Lectura de Registros de Estado

Sintaxis del lenguaje ensamdor

rd *reg_{rd},, statereg*

Formato de la instrucción

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

Sintaxis del lenguaje ensamdor

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

Formato de la instrucción

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	$siconst_{13}$	

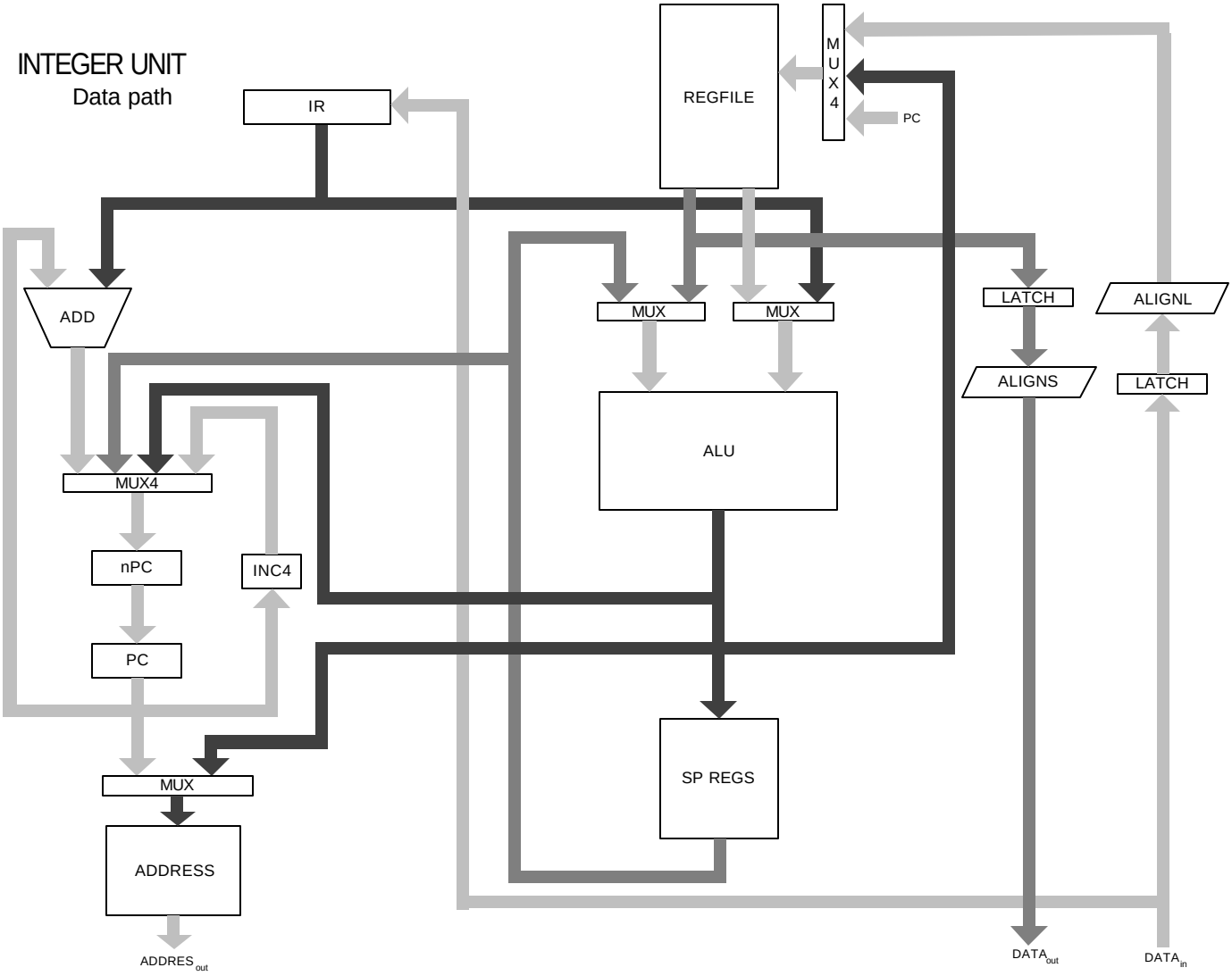
Codificación de los registros

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}

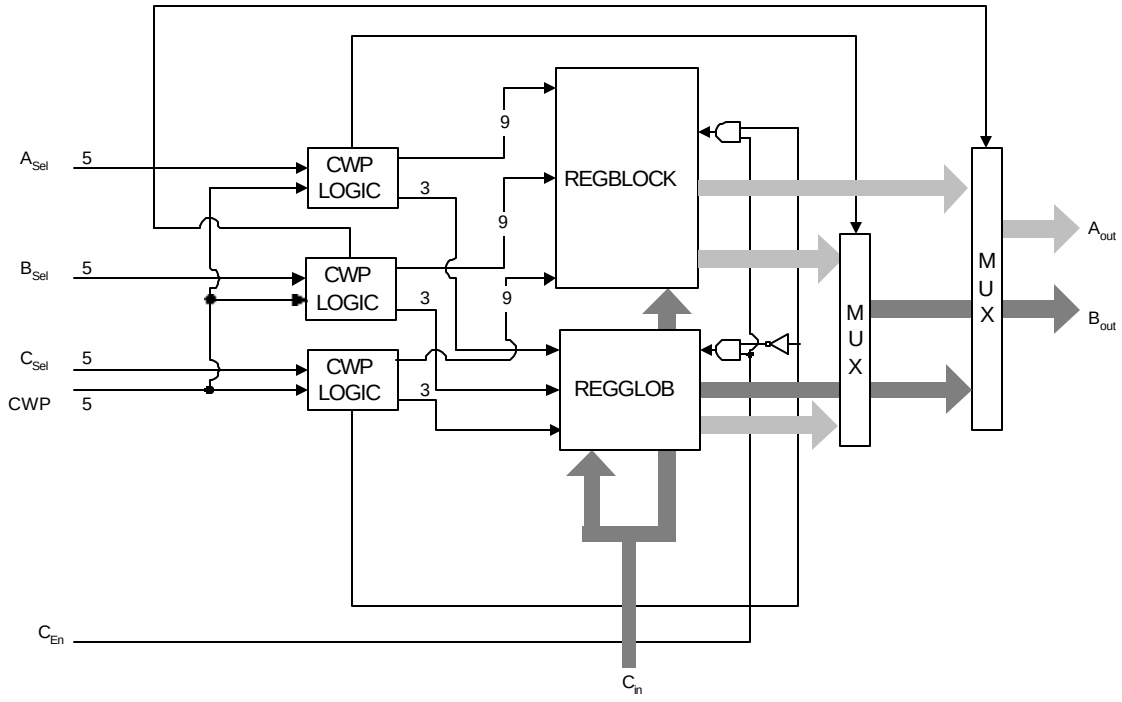
Diagramas de la Arquitectura

Integer Unit - Data Path



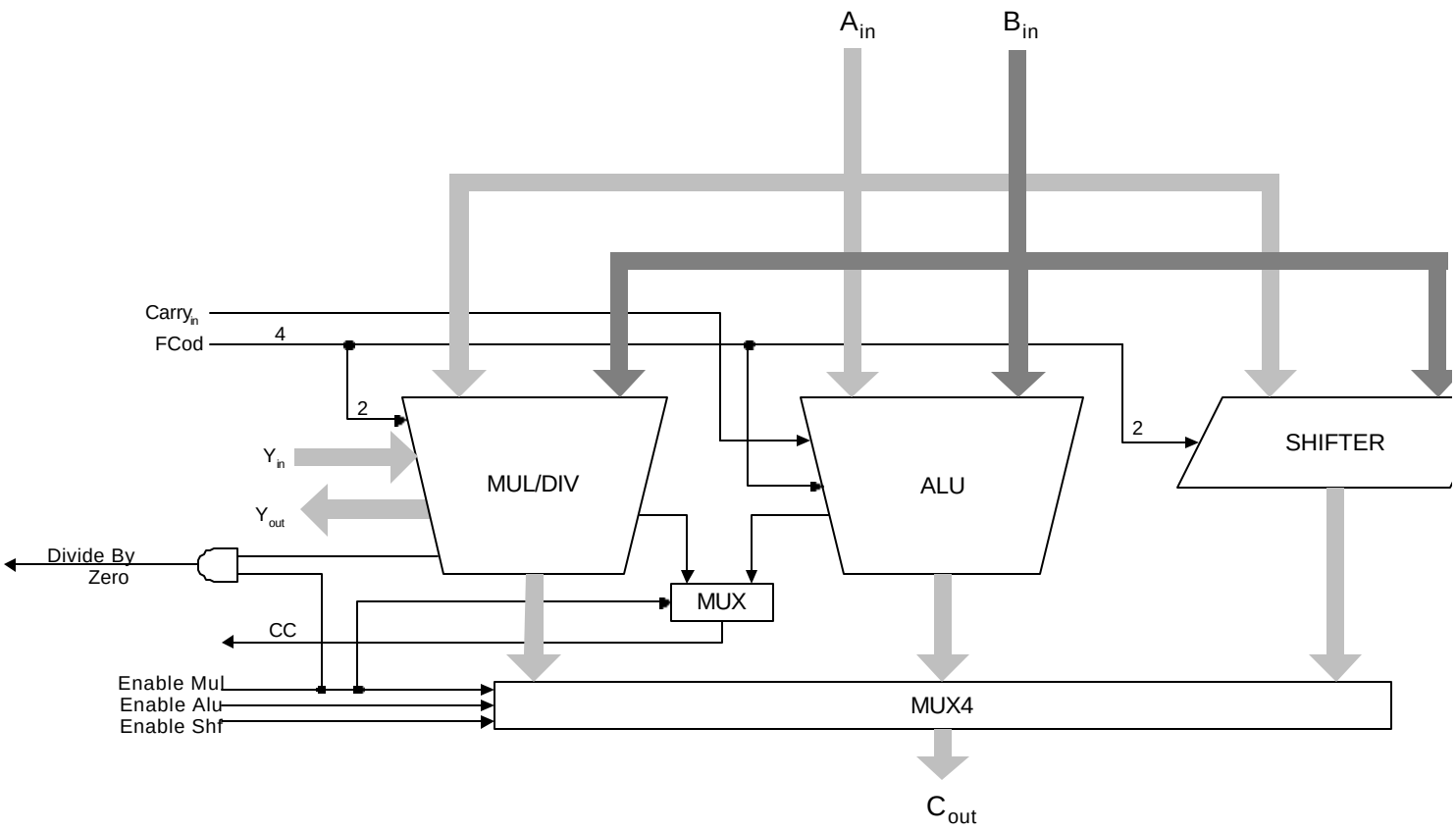
Registros de uso general

REGFILE



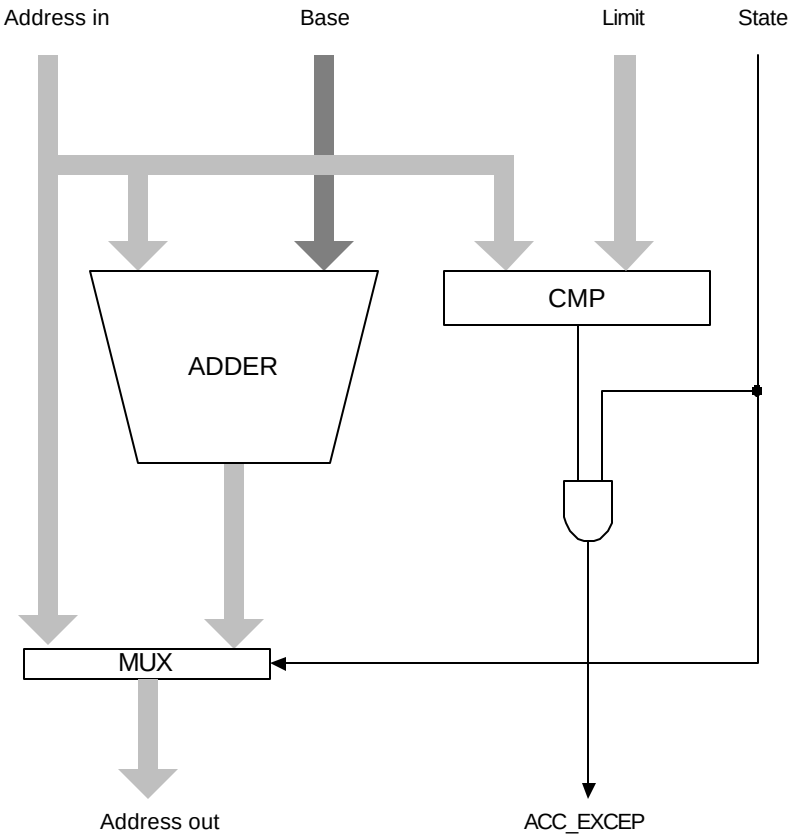
ALU

ALU&F


A in
B in
Carry in
FCod
4
2
2
Y in
Y out
MUL/DIV
ALU
SHIFTER
Divide By Zero
MUX
CC
Enable Mul
Enable Alu
Enable Shf
MUX4
C out

Address Unit

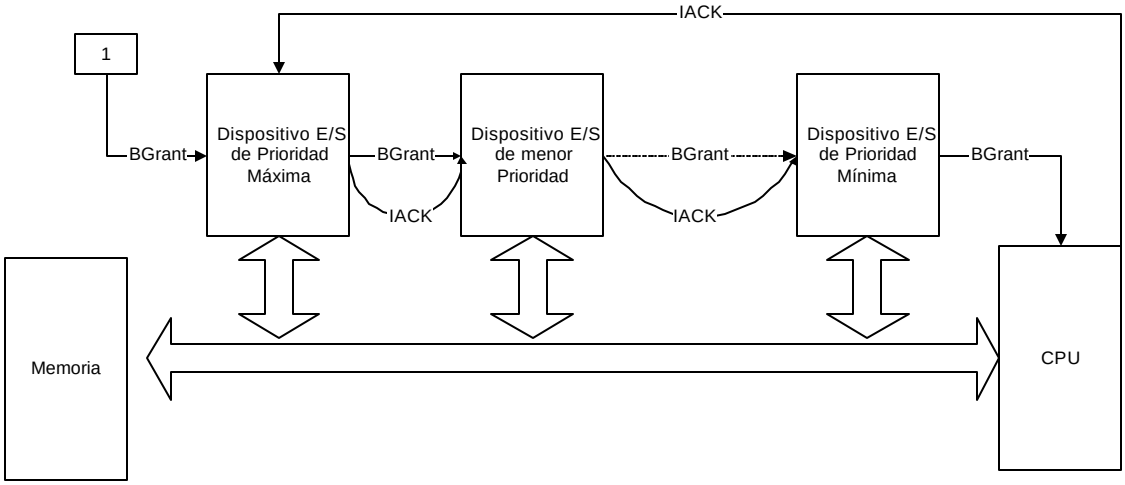
ADDRESS UNIT



BUS

El bus conecta la memoria, el CPU y los dispositivos de entrada/salida. La memoria es mappeada. Esto quiere decir que cada dispositivo “tiene” un conjunto de direcciones de memoria asignadas. Es decir, cualquier dato escrito a dichas direcciones debe ser leído e interpretado por ellos.

El bus se conecta de la siguiente forma:



Toda componente que pueda funcionar como Master del bus (dispositivos entrada/salida, CPU) se conecta, por un lado, al dispositivo de prioridad inmediatamente superior, y por otro al de prioridad inmediatamente inferior, a través de las líneas BGRANT (Bus Grant) e IACK (IRQ Acknowledgment). El dispositivo de mayor prioridad se conecta, en su BGRANTin, a una señal constante de valor 1.

Por otro lado, el bus posee 15 líneas IRQ (IRQ1..IRQ15). Cada dispositivo se conecta a una de ellas, también de acuerdo a su prioridad.

Para comprender mejor el funcionamiento del bus y las componentes de su interfaz, se describe aquí un ciclo completo de bus. Como ejemplo, tomaremos un dispositivo de entrada/salida que quiere escribir un dato en una dirección perteneciente a la Memoria.

El dispositivo que desee colocar la información en el bus, antes de comenzar el ciclo debe estra recibiendo un 1 en su BGRANTin. Coloca entonces un 0 en su BGRANTout, de forma tal que ningún dispositivo de menor prioridad pueda tomar el bus.

El ciclo de bus comenzará cuando el dispositivo que estuviera utilizando el bus antes dé de baja la señal BUSY:

1. Cuando la señal BUSY esté en 0, el dispositivo coloca un 1 en BUSY.
2. Coloca la dirección a la cual quiere escribir en A0-A31.
3. Coloca el dato en DATA0-DATA31.
4. Indica qué bytes de la palabra contenida en la dirección quiere escribir a través del Byte Select Mask, BSEL0-BSEL3.
5. Coloca un 1 (Write) en RD/WRout.
6. Coloca un 1 en AS, indicando que ya colocó toda la información en el BUS.
7. La memoria recibe el AS.
8. La memoria recibe la dirección y la identifica como propia.
9. La memoria graba el dato recibido en la dirección correspondiente.

10. La memoria coloca un 1 en su DTACKout.
11. El dispositivo recibe el 1 en su DTACKin.

Terminado el ciclo, si sigue teniendo un 1 en BGRANTin, puede transmitir otro dato. Si no necesita hacerlo, coloca un 0 en BUSY y, si está recibiendo un 1 en BGRANTin, lo coloca en BGRANTout.

Componentes de la Arquitectura y sus interfaces

ADDER

Suma dos operandos, seteando el control de Carry.

Nombre	In	Out	Descripción
OPA0-OPA31	X		Op A
OPB0-OPB31	X		Op B
RES0-RES31		X	Result
C		X	Carry

Res = OpA + OpB

ALIGNL

Se utiliza para alinear el dato leído en un Load

Nombre	In	Out	Descripción
OP0-OP31	X		Op
SIZE0-SIZE1	X		
A0-A1	X		Bits más bajos de address
SIGN	X		
RES0-RES31		X	Result

Codificación de Size: 00: Word, 01: Byte, 10: Half Word

Si Size = Word, Res = Op
Si Size = Half Word, Res = Op >> (16 * A_1) AND 0x00FF
    Si Sign = 1, se copia el bit 15 de Res a los bits 16-31
Si Size = Byte, Res = Op >> (8 * A_1A_0) AND 0x000F
    Si Sign = 1, se copia el bit 7 de Res a los bits 8-31

ALIGNS

Se utiliza para alinear el dato a grabar en un Store.

Nombre	In	Out	Descripción
OP0-OP31	X		Op
SIZE0-SIZE1	X		
A0-A1	X		Bits más bajos de address
RES0-RES31		X	Result

Codificación de Size: 00: Word, 01: Byte, 10: Half Word

Si Size = Word, Res = Op
Si Size = Half Word, Res = Op << (16 * A_1)

Si Size = Byte, Res = Op << (8 * A_1A_0)

ALU

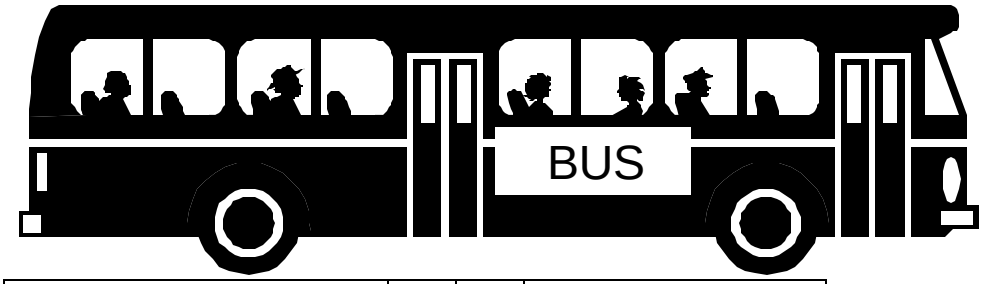
Unidad Aritmético-Lógica.

Nombre	In	Out	Descripción
OPA0-OPA31	X		Op A
OPB0-OPB31	X		Op B
CIN	X		Carry in
FCOD0-FCOD3	X		Function Code
RES0-RES31		X	Result
C		X	Carry
N		X	Negative
Z		X	Zero
V		X	Overflow

FCod		Res
0000	ADD	OpA + OpB
0001	AND	OpA $\wedge$ OpB
0010	OR	OpA $\vee$ OpB
0011	XOR	OpA XOR OpB
0100	SUB	OpA – OpB
0101	ANDN	~(OpA $\wedge$ OpB)
0110	ORN	~(OpA $\vee$ OpB)
0111	XNOR	~(OpA XOR OpB)
1000	ADDX	OpA + OpB + Cin
1100	SUBX	OpA – OpB – Cin

Las operaciones lógicas dejan C y O en 0

BUS



Nombre	In	Out	Descripción
DATA0-DATA31	X	X	Data
A0-A31	X	X	Address
BSEL0-BSEL3	X	X	Byte Select
CLOCK	X	X	Clock
AS	X	X	Address Strobe
RD/WR	X	X	Read or Write
DTACK	X	X	Data Acknowledge
ERR	X	X	Error
RESET	X	X	Reset
IRQ1-IRQ15	X	X	Interrupt Request
BUSY	X	X	Bus Busy

Todas las señales se copian del input al output

CCLOGIC

Se utiliza para decidir si se debe o no bifurcar en los saltos condicionales.

Nombre	In	Out	Descripción
C	X		Carry
N	X		Negative
Z	X		Zero
V	X		Overflow
COND0-COND3	X		Cond
RES		X	Result

Cond		Descripción	Res
0000	N	Never	0
0001	E	Equal	Z
0010	LE	Less or Equal	Z $\vee$ (N XOR V)
0011	L	Less	N XOR V
0100	LEU	Less or Equal Unsigned	C $\vee$ Z
0101	CS	Carry Set	C
0110	NEG	Negative	N
0111	VS	Overflow Set	V
1000	A	Always	1
1001	EN	Not Equal	~Z
1010	G	Greater	~(Z $\vee$ (N XOR V))
1011	GE	Greater or Equal	~(N XOR V)
1100	GU	Greater Unsigned	~(C $\vee$ Z)
1101	CC	Carry Clear	~C
1110	POS	Positive	~N
1111	VC	Overflow Clear	~V

CLOCK

Nombre	In	Out	Descripción
CLCK		X	Clock

El período del reloj debe ser configurable.

CMP

Se utiliza en la Addres Unit para verificar que las direcciones no excedan el límite.

Nombre	In	Out	Descripción
OPA0-OPA31	X		Op A
OPB0-OPB31	X		Op B
EQ		X	Equal
LW/GT		X	lower/not greater

EQ = (Op A = Op B)
LW = (Op A < Op B)

CS

Nombre	In	Out	Descripción
A0-A31	X		Address
AS	X		Address Strobe
CS		X	Chip Select

Debe poder configurarse para responder a un rango de direcciones.
P. Ej: Rango = [0xFFFF0000..0xFFFFFFFF]

$CS = AS \wedge A \in Rango$

CWPLOGIC

Decide si se accede a los registros globales o a la ventana de registros.

Nombre	In	Out	Descripción
CWP0-CWP4	X		Current Window Pointer
SEL0-SEL4	X		Select
GSEL0-GSEL2		X	Global Select
RSEL0-RSEL8		X	Register Select
R/G		X	Register/Global

Si Sel < 8, GSel = Sel y R = 0
Si no, RSel = (CWP * 16 + Sel - 8) % 512 y R = 1

INC4

Se utiliza para actualizar el nPC.

Nombre	In	Out	Descripción
OP0-OP31	X		Op
RES0-RES31		X	Result = Op + 4

INC/DEC

Se utiliza para actualizar el CWP.

Nombre	In	Out	Descripción
OP0-OP4	X		Op
FCOD	X		1 = INC ; 0 = DEC
RES0-RES4		X	Result

IRQLOGIC

Nombre	In	Out	Descripción	Prioridad	Trap Type
IRQ1-IRQ15	X		Interrupt request	32-IRq	0x11-0x1F
PIL0-PIL3	X		Processor interrupt level		
TF		X	Trap found		
TT0-TT7		X	Trap type		

Si alguna IRq > PIL está encendida, se selecciona la de mayor prioridad y se enciende TF

LATCH

Nombre	In	Out	Descripción

IN0-IN31	X		
EIN	X		Enable Input
CLEAR	X		
OUT0-OUT31		X	

Memoria que contiene un registro. Su valor lo llamaremos L.
Cuando EIn = 1, L = In
Si Clear = 1, L = 0
Out siempre es L

MEM

Memoria.

Nombre	In	Out	Descripción
DATA0-DATA31	X	X	Data
A2-A31	X		Address
BSEL0-BSEL3	X		Byte Select
AS	X		Address Strobe
RD/WR	X		Read or Write
DTACK		X	Data Acknowledge
ERR		X	Error
RESET	X		Reset

Debe poder configurarse el tamaño, y se tiene que poder cargar la imagen inicial de un archivo.

Cuando AS=1 se lee A
Si Rd=1
    BSel(M[A]) = BSel(Data)
Si no
    Data = M[A]
DtAck = 1 cuando se terminó la operación
Err = 1 si la dirección no existe
BSel indica qué bytes de la palabra se actualizan (P. Ej si BSel = 0011 solo se actualizan los 16 bits menos significativos.
Reset hace que el contenido de la memoria vuelva a ser la imagen inicial

MUL/DIV

MULtiplier/DIVider.

Nombre	In	Out	Descripción
OPA0-OPA31	X		Op A
OPB0-OPB31	X		Op B
YIN0-YIN31	X		Y In
YOUT0-YOUT31		X	Y Out
FCOD0-FCOD1	X		Function Code
RES0-RES31		X	Result
N		X	Negative
Z		X	Zero
DIV_ZERO		X	Division by zero

FCod

00	UMUL	YOut * 2^{32} + Res = OpA × OpB (Unsigned)
01	SMUL	YOut * 2^{32} + Res = OpA × OpB (Signed)
10	UDIV	Res = YIn * 2^{32} + OpA / OpB (Unsigned)

11 SDIV Res = YIn * 2^{32} + OpA / OpB (Signed)

MUX

Multiplexor de dos entradas.

Nombre	In	Out	Descripción
A0-A31	X		Op A
B0-B31	X		Op B
SELA/SELB	X		Select A/Select B
OUT0-OUT31		X	Out

Si SelA=1, Out = OpA
Si no, Out = OpB

MUX4

Multiplexor de 4 entradas.

Nombre	In	Out	Descripción
A0-A31	X		Op A
B0-B31	X		Op B
C0-C31	X		Op C
D0-D31	X		Op D
SELA	X		Select A
SELB	X		Select B
SELC	X		Select C
SELD	X		Select D
OUT0-OUT31		X	Out

Si SelA=1, Out = OpA
Si SelB=1, Out = OpB
Si SelC=1, Out = OpC
Si SelD=1, Out = OpD

REGBLOCK

Ventana de registros.

Nombre	In	Out	Descripción
ASEL0-ASEL8	X		Select A
BSEL0-BSEL8	X		Select B
CSEL0-CSEL8	X		Select C
CEN	X		Enable C
RESET	X		Reset
AOUT0-AOUT31		X	Output A
BOUT0-BOUT31		X	Output B
CIN0-CIN31	X		Input C

Bloque de 512 registros. Lo llamaremos R
AOut = R[ASel]
BOut = R[BSel]
Si CEn = 1, R[CSel] = Cin
Reset deja todo en 0

REGGLOB

Registros globales.

Nombre	In	Out	Descripción
ASEL0-ASEL3	X		Select A
BSEL0-BSEL3	X		Select B
CSEL0-CSEL3	X		Select C
CEN	X		Enable C
RESET	X		Reset
AOUT0-AOUT31		X	Output A
BOUT0-BOUT31		X	Output B
CIN0-CIN31	X		Input C

Bloque de 8 registros. Lo llamaremos G. G[0] siempre es 0.
AOut = G[ASel]
BOut = G[BSel]
Si CEn = 1 y CSel > 0, G[CSel] = Cin
Reset deja todo en 0

SHIFTER

Nombre	In	Out	Descripción
OPA0-OPA31	X		Op A
OPB0-OPB4	X		Op B
FCOD0-FCOD1	X		Function Code
RES0-RES31		X	Result

FCod		Res
01	SLL	OpA << OpB
10	SRL	OpA >> OpB (Unsigned)
11	SRA	OpA >> OpB (Signed)

SIGNEXT13

Extiende el signo de un operando de 13 bits a 32 bits.

Nombre	In	Out	Descripción
OP0-OP12	X		Op
RES0-RES31		X	Result = signextend(Op)

SIGNEXT22

Extiende el signo de un operando de 22 bits a 32 bits.

Nombre	In	Out	Descripción
OP0-OP21	X		Op
RES0-RES31		X	Result = signextend(Op)

TRAPLOGIC

Manejo de traps.

Nombre	In	Out	Descripción	Prioridad	Trap Type
INST_ACC_EXCEP	X		Instruction access exception	5	0x01
ILLEG_INST	X		Illegal instruction	7	0x02
PRIV_INST	X		Privileged instruction	6	0x03
WIN_OVER	X		Window overflow	9	0x05
WIN_UNDER	X		Window underflow	9	0x06
ADDR_NOT_ALIGN	X		Address not aligned	10	0x07
DATA_ACC_EXCEP	X		Data access exception	13	0x09
INST_ACC_ERR	X		Instruction access error	3	0x21
DATA_ACC_ERR	X		Data access error	12	0x29
DIV_ZERO	X		Division by zero	15	0x2A
DATA_ST_ERR	X		Data store error	2	0x2B
TRAP_INST	X		Trap instruction	16	0x80+TN
TN0-TN6	X		Trap number		
TF		X	Trap found		
TT0-TT7		X	Trap type		

Si alguna entrada (salvo TN) está encendida se selecciona la de mayor prioridad y se enciende TF.

WIMCHECK

Se utiliza para ver si hay window overflow/underflow en un save/restore.

Nombre	In	Out	Descripción
CWP0-CWP4	X		Current Window Pointer
WIM0-WIM31	X		Window Invalid Mask
RES		X	Res = WIM_{CWP}

Si Sel < 8, GSel = Sel y R = 0
Si no, RSel = (CWP * 16 + Sel - 8) % 512 y R = 1

Simulando componentes digitales

La mayoría de los modelos atómicos deben simular el comportamiento de componentes digitales. Pueden recibir en sus líneas de entrada y transmitir en sus líneas de salida uno de dos voltajes. Los circuitos operan de manera continua.

Para simular este comportamiento, un modelo debe recordar el último valor que recibió en cada uno de sus inputs y el que envió por cada uno de sus outputs. Al recibir un mensaje de cambio de algún input, vuelve a calcular sus outputs y envía mensajes solo para aquellos que hayan cambiado.