ARQUITECTURA SPARC: FORMATO DE INSTRUCCIÓN

De Diego Varona, Rubén

Romay López, Oscar Manuel

Vega Martínez, Jorge

INTRODUCCIÓN

La representación de la instrucción en la computadora se denomina **formato**.

Sabemos que un programa está formado por una secuencia de instrucciones, cada una de las cuales especifica una acción particular. La parte esencial de la instrucción es lo que llamamos **código de operación**, que señala la acción que debe ejecutarse.

Muchas instrucciones contienen además los datos que usan o especifican donde están. Así, por ejemplo, una instrucción que compare dos caracteres para ver si son iguales, debe especificar qué caracteres se han de comparar. Todo lo relativo a la especificación de dónde está el dato o donde hay que almacenar el resultado se llama **direccionamiento**.

INSTRUCCIONES EN SPARC

La arquitectura SPARC tiene cerca de 50 instrucciones enteras, unas pocas más que el anterior diseño RISC, pero menos de la mitad del número de instrucciones enteras del 6800 de Motorola.

Las instrucciones de SPARC se pueden clasificar en cinco categorías:

- **LOAD** y **STORE** (La única manera de acceder a la memoria). Estas instrucciones usan dos registros o un registro y una constante para calcular la dirección de memoria a direccionar.

- Instrucciones **Aritméticas/Lógicas/Shift**. Ejecutan operaciones aritméticas, lógicas y de desplazamiento de bits. Estas instrucciones calculan el resultado si es una función de 2 operandos y guardan el resultado en un registro.

- **Operaciones del Coprocesador**. La IU extrae las operaciones de punto flotante desde las instrucciones del bus de datos y los coloca en la cola para la FPU. La FPU ejecuta los cálculos de punto flotante con un número fijo en unidad aritmética de punto flotante, (el número es dependiente de la aplicación). Las operaciones de punto flotante son ejecutadas concurrentemente con las instrucciones de la IU y con otras operaciones de punto flotante cuando es necesario. La arquitectura

SPARC también especifica una interfaz para la conexión de un coprocesador adicional.

- **Instrucciones de Control de Transferencia.** Estas incluyen jumps, calls, traps y branches. El control de transferencia es retardado usualmente hasta después de la ejecución de la próxima instrucción, así el pipeline no es vaciado porque ocurre un control de tiempo. De este modo, los compiladores pueden ser optimizados por ramas retardadas.

- **Instrucciones de control de registros Read/Write**. Estas instrucciones se incluyen para leer y grabar el contenido de varios registros de control. Generalmente la fuente o destino está implícito en la instrucción.

EJECUCIÓN DE LAS INSTRUCCIONES.

El procesador accede a las instrucciones desde la memoria para ser ejecutadas, anuladas o TRAPPED. Las instrucciones son codificadas en cuatro formatos principales y divididas en once categorías generales.

La instrucción en la posición de memoria especificada por el PC (contador de programa) es traída al frente y posteriormente ejecutada. La ejecución de la instrucción puede cambiar el procesador visible de programa y/o el estado de la memoria. Como efecto secundario de su ejecución, nuevos valores serán asignados al PC (program counter) y al siguiente contador de programa (nPC) (next program counter).

Una instrucción puede generar una excepción si encuentra alguna condición que la haga imposible completar una ejecución normal. Este tipo de excepción a su vez puede generar una precisa TRAP. Otros eventos también pueden causar TRAPS: una excepción causada por una instrucción anterior (DEFERRED TRAP), una interrupción o un error asíncrono (DISRPTING TRAP o TRAP INTERRUMPIDA) o una solicitud de restablecimiento
(Una trap reset). Si se produce una trap, el control es vectorizado en una tabla de traps.

Si una TRAP no se produce y la instrucción no es una transferencia de control, entonces el siguiente contador de programa (nPC) se copia en el PC y el npc se incrementa en 4 (ignorando el desbordamiento, en caso de que exista). Si la instrucción es una instrucción de control de transferencia, el

siguiente nPC se copia en el PC y la dirección de destino se escribe en el nPC. Así, los dos contadores de programa establecen un modelo de ejecución retardada.

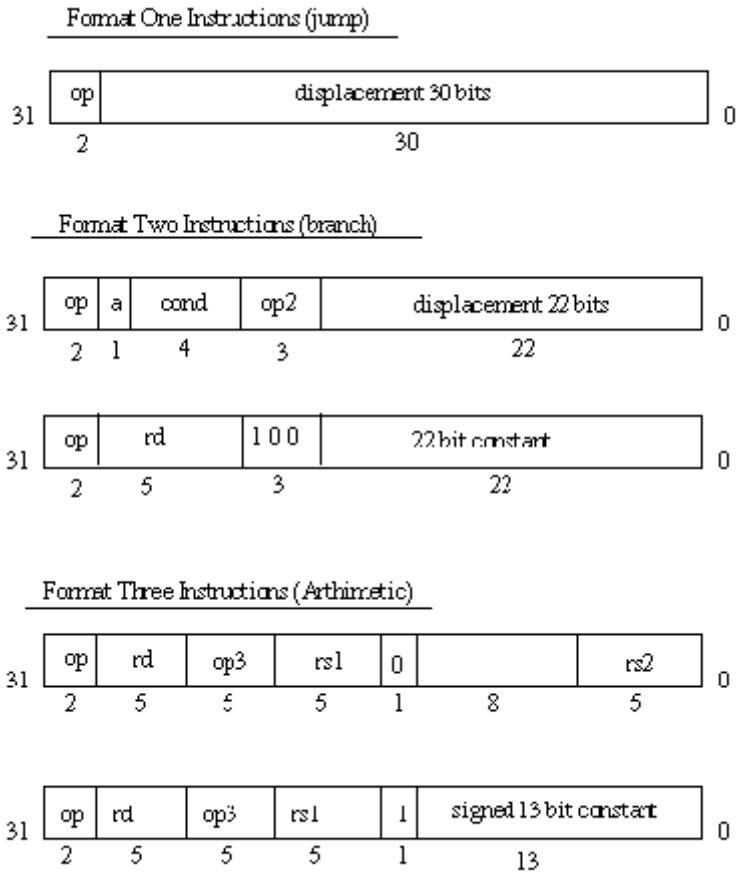
Para cada acceso a la instrucción y cada acceso a los datos normales, IU(Integer Unit) añade un identificador de espacio de direccionamiento o ASI (adress space identifier) de 8 bits a la dirección de memoria de 64 bits. Las instrucciones alternas Load/Store pueden proporcionar un ASI arbitrario con su dirección de dato, o usar el valor actual de ASI contenido en el registro ASI.

FORMATO DE INSTRUCCIONES.

Como la mayoría de las máquinas RISC, las máquinas SPARC y MIPS (siglas de ***M****icroprocessor without* ***I****nterlocked* ***P****ipeline* ***S****tages*) tienen muchas instrucciones divididas en unos pocos tipos de instrucciones máquina. Al tener sólo unos pocos tipos de instrucciónes , la arquitectura de la máquina es más fácil de diseñar y más fácil de optimizar.

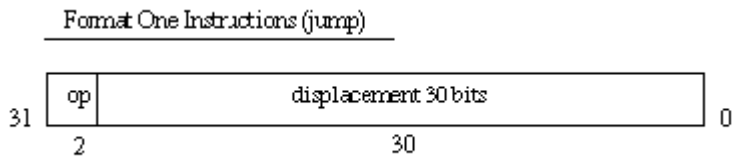
Tanto la máquina MIPS como SPARC usan instrucciones de treinta y dos bits de largo. Ambos equipos tienen tres tipos de instrucciones. En el SPARC se les llama instrucciones de la forma 1 2 y 3.

La disposición de las instrucciones SPARC es:



Formato uno - Instrucciones de salto:

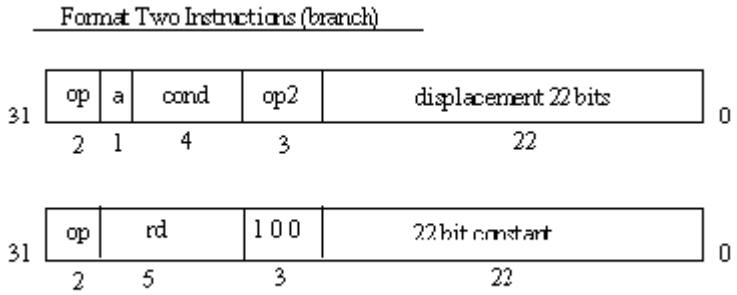
La instrucción de llamada SPARC , utilizada para transferir el control a cualquier parte del espacio de direcciones de 32 bits es así:



Sólo hay una instrucción en la máquina SPARC que es de la forma número uno llamada (CALL) o instrucción de llamada. Cuando se encuentra esta instrucción, el control se transfiere inmediatamente a la nueva ubicación dada por la constante de 30 bits Cómo se determina la ubicación es la constante es desplazado a la izquierda dos bits (para crear una palabra de 32 bits) y el PC se establece en este valor más el PC actual. El cambio de dirección es relativa al contador de programa (PC) para que el programa pueda ser destinado a las memorias sin afectar a las direcciones especificadas por las instrucciones de llamada.

Formato dos - Intrucciones Branch y Sethi:

Tienen la siguiente forma:



Al igual que en el ejemplo anterior, los dos primeros bits especifican el tipo de instrucción, el *cond* es la condición de bifurcación y la op2 es el operando con el que se compara. Junto con el control de la máquina las transferencias a la ubicación especificada por la constante de 22 bits.

SPARC proporciona instrucciones de bifurcación múltiple. El tipo de bifurcación está determinada por los bits de cond. Si se toma la bifurcación, entonces el PC está configurado para la constante que aparece a la izquierda por 2 y añadida al contador de programa. Hay que tener en cuenta que esto sólo permite una bifurcación de hasta 2 ^ 21 posiciones de memoria. Para realizar bifurcaciones a través de todo el rango de memoria, se requiere un manejo especial.

Otro ejemplo de la forma número 2: (Segunda imagen)

00	ª (5)	100	constante de 22 bits

Esta instrucción de la forma número dos se conoce como la instrucción Sethi. Esta instrucción se utiliza para cargar una instrucción de treinta y dos bits en un registro. Para relizarlo, esta instrucción cargará los veintidós bits para posteriormente llamar a una instrucción “OR” encargada de cargar los 10 bits mas bajos de la palabra.

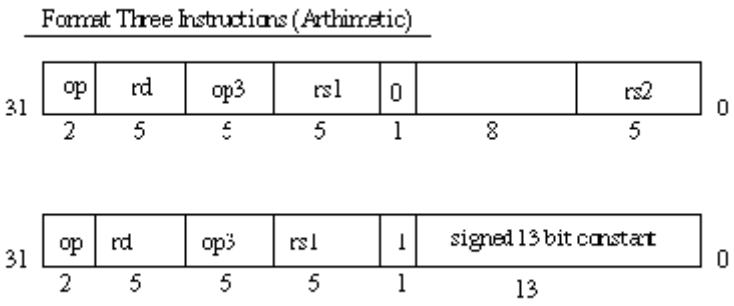
Formato tres: - Instrucciones algebraicas:

Este tipo de instrucciones son las más comunes. Estas son las instrucciones algebraicas o l LOAD/STORE (excepto Sethi).

Estas instrucciones tienen un registro de destino, llamado RD, un instrucción específica llamada OP3, y registro fuente, llamado rs1. La otra fuente de la operación puede ser o bien otro registro fuente, llamado rs2 o una constante de 13 bits (para el caso de operaciones inmediatas).

Algo a tener en cuenta es que al ser este formato el que contiene las instrucciones de LOAD/STORE, hay un problema potencial con ellos al poseer únicamente una constante de 13 bits. ¿Cómo puede direccionarse la memoria en un rango tan limitado (2 ^ 13)? Para estos casos, la dirección de la instrucción está relacionada con el indicador de puntero. En comparación con la máquina MIPS, el direccionamiento de memoria tendrá que ser más lento ya que las instrucciones MIPS formato que-naturalmente, puede abordar una amplia memoria más grande y por lo tanto no será necesaria la instrucción adicional para cargar una dirección en el registro antes de la instrucción de la carga (como la máquina SPARC tiene que hacer).

El formato de instrucción número 3 es:



A continuación presentamos algunas diapositivas con ejemplos de instrucciones sparc.

CARTILLA REDUCIDA DE INSTRUCCIONES SPARC

CARTILLA SPARC (Reducida)	
INSTRUCCIÓN	**OPERACIÓN**
ADD reg_1, reg_o_imm, reg_2	$reg_2 \leftarrow reg_1 + reg\_o\_imm$
ADDX reg_1, reg_o_imm, reg_2	$reg_2 \leftarrow reg_1 + reg\_o\_imm + c$
SUB reg_1, reg_o_imm, reg_2	$reg_2 \leftarrow reg_1 - reg\_o\_imm$
SUBX reg_1, reg_o_imm, reg_2	$reg_2 \leftarrow reg_1 - reg\_o\_imm - c$

AND reg_1, reg_o_imm, reg_2	$reg_2 \leftarrow reg_1 \wedge reg\_o\_imm$
OR reg_1, reg_o_imm, reg_2	$reg_2 \leftarrow reg_1 \vee reg\_o\_imm$
XOR reg_1, reg_o_imm, reg_2	$reg_2 \leftarrow reg_1 \oplus reg\_o\_imm$

SLL reg_1, reg_o_imm, reg_2	corre reg_1, reg_o_imm bits a la izq. y coloca el resultados en reg_2 . Rellena con ceros.
SRL reg_1, reg_o_imm, reg_2	corre reg_1, reg_o_imm bits a la der. y coloca el resultados en reg_2 . Rellena con ceros.
SRA reg_1, reg_o_imm, reg_2	corre reg_1, reg_o_imm bits a la der. y coloca el resultados en reg_2 . Extiende signo.

LDSB *direcc*, *reg*	carga en *reg* el byte que esta en la dirección *direcc*. Expande el signo
LDUB *direcc*, *reg*	carga en *reg* el byte que esta en la dirección *direcc*. No expande el signo
LDSH *direcc*, *reg*	carga en *reg* la media palabra que esta en la dirección *direcc*. Expande el signo
LDUH *direcc*, *reg*	carga en *reg* la media palabra que esta en la dirección *direcc*. No expande el signo
LD *direcc*, *reg*	carga en *reg* la palabra que esta en la dirección *direcc*.

STB *reg*, *direcc*	coloca los 8 bits menos significativos de *reg* en *direcc*
STH *reg*, *direcc*	coloca los 16 bits menos significativos de *reg* en *direcc*
ST *reg*, *direcc*	coloca a *reg* en *direcc*

SWAP *direcc*, *reg*	intercambia los valores de *direcc* y *reg*

SETHI *const22*, *reg*	pone a cero los 10 bits menos significativos de *reg* y coloca a *const22*, en los 20 bits mas significativos .

CALL \EMBED Equation *desp30*	pasa el control a PC+4*$desp30$ y almacena el PC en %15
JMPL *direcc*, *reg*	pasa el control a *direcc* y almacena el PC en *reg*

CARTILLA SPARC (Reducida)	
INSTRUCCIÓN	**OPERACIÓN**
BNE \EMBED Equation $desp22$	salta a PC+4*desp22 si no Z
BE \EMBED Equation $desp22$	" si Z
BG \EMBED Equation $desp22$	" si no Z y N=V
BLE \EMBED Equation $desp22$	" si Z o N≠V
BGE \EMBED Equation $desp22$	" si N=V
BL \EMBED Equation $desp22$	" si N≠V
BGU \EMBED Equation $desp22$	" si no C y no Z
BLEU \EMBED Equation $desp22$	" si C or Z
BCC \EMBED Equation $desp22$	" si no C
BCS \EMBED Equation $desp22$	" si C
BPOS \EMBED Equation $desp22$	" si no N
BNEG \EMBED Equation $desp22$	" si N
BVC \EMBED Equation $desp22$	" si V
BVC \EMBED Equation $desp22$	" si no V

RDPSR reg	coloca el valor del PSR en reg
WRPSR reg, reg_o_imm	coloca en PSR el resultado de reg xor reg_o_imm
RDWIM reg	coloca el valor de WIM en reg
WRWIM reg, reg_o_imm	coloca en WIM el resultado de reg xor reg_o_imm
RDTBR reg	coloca el valor del TBR en reg
WRTBR reg, reg_o_imm	coloca en TBR el resultado de reg xor reg_o_imm

SAVE reg_1, reg_o_imm, reg_2	almacena la ventana del procedimiento llamador
RESTORE reg_1, reg_o_imm, reg_2	restablece la ventana del procedimiento llamador