INSTRUCCIONES: EL LENGUAJE DEL PROCESADOR

ARQUITECTURA DEL PROCESADOR II

La tecnología de las computadoras modernas, requiere que los profesionales de las distintas especialidades en ciencias de la computación, entiendan tanto de software como de hardware. [1] La interacción en diversos niveles entre el software y el hardware, constituye un buen marco para entender los fundamentos de las ciencias de la computación.
En los últimos treinta años, surgieron numerosos nuevos modelos de computadoras, cada una de estas apariciones revolució la industria; y cada revolución fue superada sólo porque alguien construyó un mejor modelo de computadora. Esta carrera por la innovación ha llevado a un progreso nunca antes visto, desde el inicio de la computación electrónica a fines de 1940. Haciendo una comparación entre la *industria del transporte* y la *industria de las computadoras*, si los medios de transporte hubiesen avanzado tanto como las computadoras, por ejemplo hoy día sería posible viajar de Nueva York a Londres en un segundo pagando un centavo.

Como parte de esa evolución, fueron necesarios algunos cambios y adaptaciones que sirvieron para mejorar la terminología empleada para designar distintos recursos. Una de estas mejoras se dió con las unidades que designan elementos de almacenamiento. Por razones históricas y de costumbre, los sistemas de almacenamiento secundario y de comunicaciones comenzaron a utilizar potencias de diez para designar grandes contidades de bytes.

Decimal term	Abbreviation	Value	Binary term	Abbreviation	Value	% Larger
kilobyte	KB	10^3	kibibyte	KiB	2^{10}	2%
megabyte	MB	10^6	mebibyte	MiB	2^{20}	5%
gigabyte	GB	10^9	gibibyte	GiB	2^{30}	7%
terabyte	TB	10^{12}	tebibyte	TiB	2^{40}	10%
petabyte	PB	10^{15}	pebibyte	PiB	2^{50}	13%
exabyte	EB	10^{18}	exbibyte	EiB	2^{60}	15%
zettabyte	ZB	10^{21}	zebibyte	ZiB	2^{70}	18%
yottabyte	YB	10^{24}	yobibyte	YiB	2^{80}	21%

FIGURA 1. Potencias de diez vs potencias de dos

La Figura 1 resume la comparación entre algunas potencias de diez y de dos, con los nombres usados, los valores correspondientes y el porcentaje de diferencia entre ellos. Por ejemplo el término terabyte (TB) se usó originalmente para designar 10^{12} (1.000.000.000.000)

[1] Este material de estudio es la traducción de la sección 2.2 del libro de Patterson y Hennessy [1]

bytes, que no se corresponde con ninguna potencia de dos (que es la cantidad de símbolos empleados en el sistema binario.) Una cifra cercana a 10^{12} que sea potencia de dos es 2^{40} (1.099.511.627.776), que es un 10 % más grande; desde hace algún tiempo, es el valor elegido para designar los mil millones de bytes. Hoy en día se emplea tebibyte (TiB) 2^{40} en reemplazo de terabyte (TB) 10^{12}.

La aclaración anterior es oportuna en este documento porque las tres secciones que siguen se destinan a describir el conjunto de instrucciones del procesador MIPS, a detallar los operandos de las instrucciones en memoria y a la representación de instrucciones dentro de las computadoras.

1. Operaciones del hardware de la computadora

Todo procesador debe ser capaz de realizar aritmética. La siguiente notación en lenguaje de ensamble del MIPS

$add\, a, b, c$

instruye a la computadora para sumar las dos variable b y c dejando el resultado en a. Esta notación es rígida ya que cada instrucción aritmética del MIPS realiza una única operación y siempre debe tener exactamente tres variables. Por ejemplo, supongamos que deseamos ubicar la suma de las cuatro variables b, c, d y e en a. (Por ahora se deja sin especificar que es una "variable", más adelante se explica en detalle.) La siguiente secuencia de instrucciones suma cuatro variables:

$add\, a$, b, c # La suma de b y c es dejada en a
$add\, a$, a, d # Ahora la suma de b, c y d es dejada en a
$add\, a$, a, e # Finalmente la suma de b, c, d y e es dejada en a

De esa manera se requieren tres instrucciones para sumar cuatro variables.
En cada línea, las palabras a la derecha del símbolo # son *comentarios* para el lector humano, por lo que la computadora las ignora. Notar que a diferencia de otros lenguajes de programación, cada línea de este lenguaje puede contener a lo más una instrucción. Otra diferencia con el lenguaje C es que siempre termina al final de la línea.
El número natural de operandos para una operación como la suma es tres: los dos números a ser sumados y el destino de la suma. Se requiere que cada instrucción tenga exactamente tres operandos, no más ni menos, siguiendo la filosofía de mantener el hardware simple: el hardware para un número variable de operandos es más complicado que el hardware para un número fijo de operandos. Esta situación ilustra el primero de los tres principios que subyacen en el diseño de hardware:

Principio de Diseño 1: La simplicidad favorece la regularidad.

A continuación se presentan dos ejemplos que muestran la relación de programas escritos en lenguaje de alto nivel con programas en esta notación más primitiva.
Ejemplo. Compilación de dos sentencias C en MIPS. Este segmento de un programa C contiene cinco variables: a, b, c, d y e. Dado que Java evolucionó de C, este ejemplo y

algunos pocos de los que siguen, funcionan bien para ambos lenguajes de programación:

$a = b + c;$

$d = a - e;$

La traducción de C a instrucciones assembly de MIPS es realizada por el *compilador*. Muestre el código MIPS producido por el compilador para las operaciones anteriores.

Respuesta. Una instrucción MIPS opera con dos operandos fuentes y ubica el resultado en el operando destino. Por lo que las dos sentencias simples de arriba se compilan directamente en estas dos instrucciones del lenguaje de ensamble del MIPS:

add a, b, c

sub d, a, e

□

Ejemplo. Compilación de una asignación C compleja para MIPS. Una sentencia algo más compleja contiene cinco variables f, g, h, i y j:

$f = (g + h) - (i + j);$

¿Qué puede producir un compilador C?.

Respuesta. El compilador debe dividir esta sentencia C en varias instrucciones assembly, debido a que solamente una operación es realizada por instrucción MIPS. La primera instrucción MIPS calcula la suma de g y h. Se debe guardar el resultado en algún lugar por lo que el copilador emplea una variable temporal llamada $t0$:

add $t0$, g, h # La variable temporal $t0$ contiene $g + h$

Aunque la siguiente operación es una resta, se debe calcular la suma de i con j antes que se pueda restar. Así la segunda instrucción, ubica la suma de i con j en la variable del compilador denominada $t1$:

add $t1$, i, j # La variable temporal $t1$ contiene $i + j$

Finalmente la instrucción de resta, hace la substracción entre el resultado de la primer suma, con el resultado de la segunda suma y deja el resultado en la variable f:

sub f, $t0$, $t1$ # En f se carga $t0$ - $t1$, que es (g+h)-(i+j)

□

2. Operandos del hardware de las computadoras

A diferencia de los programas en lenguaje de alto nivel, los operandos de las instrucciones aritméticas están restringidos, ya que tiene un número limitado de lugares especiales construidos directamente en hardware, llamados *registros*. Los registros son primitivas usadas en el diseño de hardware que también son visibles al programador una vez que la computadora se completa, de esa manera se puede pensar que los registros son los ladrillos para cosntruir computadoras. El tamaño de un registro en la arquitectura del MIPS es de 32 bits; los grupos de 32 bits son tan comunes que se les dá el nombre de *palabra* en la arquitectura MIPS.

Una diferencia importante entre las variables de un lenguaje de programación y los registros del procesador, es que estos últimos están limitados en cantidad: 32 en las computadoras actuales. En el MIPS, cada uno de los tres operandos de las instrucciones aritmeticas,

deben ser alguno de los 32 registros de 32 bits. La razón del límite de 32 registros puede ser encontrado en el segundo de los tres principios subyacentes en el diseño del hardware:

Principio de Diseño 2: Lo pequeño es más rápido.

Un número muy grande de registros puede incrementar el tiempo de ciclo de reloj simplemente porque se necesitan señales eléctricas más largas cuando ellas deben viajar más lejos. Las ideas generales como "lo pequeño es más rápido" no son absolutas; 31 registros no son más rápidos que 32. Pero la realidad detrás de tales observaciones causa que los diseñadores de computadoras los toman seriamente. En este caso, el diseñador debe balancear el anhelo de los programas de contar con más registros, con el deseo de los diseñadores de lograr ciclos de reloj pequeños. Otra razón para no usar más de 32 registros es el número de bits que podría requerir en el formato de la instrucción, como se verá más adelante.

El uso efectivo de los registros es crítico para el desempeño del programa.

Se podría simplemente escribir instrucciones usando números de 0 a 31 para los registros, pero la convención de MIPS es usar dos letras luego de un signo pesos ($) como nombre de registro. Un resumen de nombres para los registros se observa en la Figura 2.

Name	Register number	Usage	Preserved on call?
`$zero`	0	The constant value 0	n.a.
`$v0–$v1`	2–3	Values for results and expression evaluation	no
`$a0–$a3`	4–7	Arguments	no
`$t0–$t7`	8–15	Temporaries	no
`$s0–$s7`	16–23	Saved	yes
`$t8–$t9`	24–25	More temporaries	no
`$gp`	28	Global pointer	yes
`$sp`	29	Stack pointer	yes
`$fp`	30	Frame pointer	yes
`$ra`	31	Return address	yes

FIGURA 2. Convenciones de registros en MIPS

Se usaran los nombres $\$s0$ al $\$s7$ para registros que se corresponden con las variables de programas en C y Java. Los nombres de registro $\$t0$ al $\$t9$ para registros temporarios necesarios para compilar los programas de alto nivel en instrucciones MIPS.

Ejemplo. Compilación de una asignación C usando registros. Es trabajo del compilador asociar variables del programa con registros. Al considerar, por ejemplo, la sentencia de asignación de un ejemplo anterior:

$$f = (g + h) - (i + j);$$

Las variables f, g, h, i y j son asignadas a los registros $\$s0$,$\$s1$,$\$s2$,$\$s3$,$\$s4$, respectivamente. ¿Cuál es el código MIPS compilado?

Respuesta. El programa compilado es muy similar al ejemplo anterior, salvo que se reemplazan los nombres de las variables con los registros mencionados arriba más dos registros temporales $\$t0$ y $\$t1$, los cuales se corresponden con las variables temporales de antes:

add $\$t0$, $\$s1$, $\$s2$ # La variable temporal $\$t0$ contiene $g + h$
add $\$t1$, $\$s3$, $\$s4$ # La variable temporal $\$t1$ contiene $i + j$
sub $\$s0$, $\$t0$, $\$t1$ # En $\$so$ se carga $t0$ - $t1$, que es (g+h)-(i+j)

□

Operandos en memoria. Los lenguajes de computación admiten variables simples que contienen un único elemento de dato (como en los ejemplos considerados), pero también admiten agrupaciones de datos más complejas –arreglos y estructuras. Estas estructuras de datos complejas pueden contener varios elementos de datos más que los registros de una computadora. ¿Cómo puede una computadora representar y acceder a estas agrupaciones de datos?.

Antes de responder, es oportuno recordar cuales son los cinco componentes principales de una computadora: camino de datos, control, memoria, entrada y salida. El procesador sólo puede colocar una pequeña cantidad de datos en los registros, pero la memoria de la computadora puede contener miles de millones de elementos de datos. Por esa razón es que las agrupaciones de datos (arreglos y estructuras) se conservan en memoria.

Como se explicó antes, las operaciones en el MIPS se hacen solamente entre registros, por lo que son necesarias operaciones para transferir datos entre la memoria y los registros. Estas son llamadas *instrucciones de transferencia de datos*. Para acceder a una palabra de datos en memoria, la instrucción debe proveer la *dirección* de memoria. La memoria es simplemente un gran arreglo de una única dimensión, donde la dirección actúa como el índice del arreglo, comenzando en la dirección cero. Por ejemplo en la Figura 3, la dirección del tercer elemento de datos es 2, y el valor de Memoria[2] es 10.

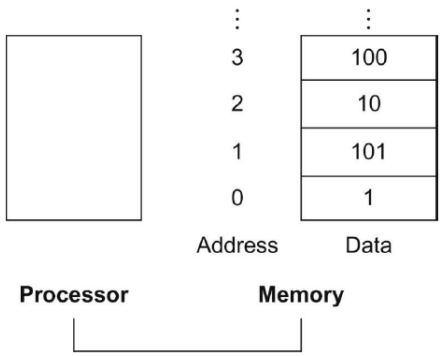

FIGURA 3. Direcciones de memoria y contenidos en esas posiciones. Como MIPS direcciona a byte, la dirección Memoria[2] es incorrecta si los elementos fuesen palabras.

La instrucción de transferencia de datos que copia datos desde memoria a los registros, tradicionalmente se llama *load*. El formato de la instrucción load es el nombre de la operación seguido por el registro a cargar, luego una constante y el registro usado para acceder a la memoria. La suma de la porción constante de la instrucción y el contenido del segundo registro constituyen la dirección de memoria. En MIPS el nombre de esta instrucción es *lw*, por las palabras en inglés *load word* (cargar palabra).

Ejemplo. Compilación de una asignación cuando un operando está en memoria. Se asume que A es un arreglo de 100 palabras y que el compilador ha asociado las variables g y h con los registros $\$s1$ y $\$s2$ como antes. Se asume que la dirección de cominezo, o *dirección base* del arreglo está en $\$s3$. Imite la tarea del compilador con la siguiente sentencia de asignación:

$$g = h + A[8];$$

Respuesta. Aunque hay una única operación en esta sentencia de asignación, uno de los operandos ($A[8]$) está en memoria, por lo que primero se debe transferir a un registro. La dirección de este elemento del arreglo, es la suma de la base del arreglo A, guardado en el registro $\$s3$, más el número para seleccionar el elemento 8. El dato debiera ser ubicado en un registro temporario para usar en la siguiente instrucción. En base a la Figura 3 la primer instrucción compilada es:

lw $\$t0, 8(\$s3)$ #El registro temporario $\$t0$ recibe $A[8]$

(Se hará un pequeño ajuste de esta instrucción, por ahora se usará esta versión simplificada.) La instrucción que le sigue puede operar con el valor que tiene $\$t0$ (igual a $A[8]$) porque está en un registro. La instrucción debe sumar h (conservada en $\$s2$) con $A[8]$ (conservado en $\$t0$) y poner la suma en el registro correspondiente a g (asociado con $\$s1$):

add $\$s1$, $\$s2$, $\$t0$ # g = h + A[8]

La constante en la instrucción de transferencia (que en este ejemplo tiene el valor 8) es llamada *desplazamiento*, y el registro sumado para formar la dirección ($\$s3$) es llamado el *registro base*.□

Además de asociar variables con registros, el compilador asigna agrupaciones de datos, como arreglos y estructuras, en posiciones de memoria. El compilador puede entonces ubicar la dirección inicial en las instrucciones de transferencia de datos.

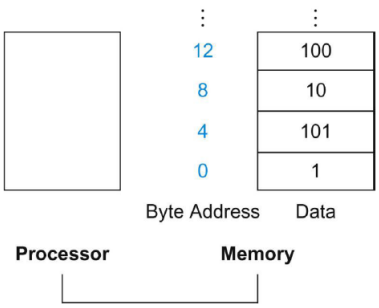

FIGURA 4. Direcciones reales en MIPS y contenido de las palabras.

Debido a que los *bytes* de 8 bits son útiles en varios programas, practicamente todas las computadoras de hoy día direccionan a bytes individuales. Por lo tanto, la dirección de una palabra coincide con la dirección de uno de los bytes dentro de una palabra, y las direcciones de palabras consecutivas difieren en 4. Por ejemplo, la Figura 4 muestra las direcciones MIPS reales para las palabras de la Figura 3; la dirección a byte de la tercer palabra es 8.

En MIPS las palabras deben comenzar en una dirección que sea múltiplo de 4. Este requerimiento es llamado *restricción de alineación* y varias arquitecturas la tienen. Las computadoras se dividen en aquellas que usan la dirección del byte de más a la izquierda o "extremo grande" (en inglés "big end") como la dirección de la palabra versus aquellas que usan el byte de más a la derecha o "extremo pequeño" (en inglés "little end"). MIPS es del grupo "big-endian". Por ejemplo en una máquina "big-endian", la directiva *.byte* 0, 1, 2, 3 resulta en una palabra de memoria conteniendo:

# byte			
0	1	2	3

mientras que en una máquina "little-endian", la palabra podría contener:

# byte			
3	2	1	0

Se otorga poca consideración a que convención usa el hardware debido a que el orden sólo importa si se accede al mismo dato de distintas formas: como palabra y como cuatro bytes.

El direccionamiento a byte también afecta el índice del arreglo. Para conseguir la dirección de byte adecuada en el código anterior, el desplazamiento a ser sumado al registro base $\$s3$ debe ser 4×8 o 32, de esa manera la dirección de carga seleccionará $A[8]$ y no $A[8/4]$.

La instrucción complementaria al load tradicionalmente es llamada *store*; esta instrucción copia datos desde un registro a memoria. El formato del store es similar al del load: el nombre de la operación seguido por el registro a ser almacenado, luego el desplazamiento para seleccionar el elemento del arreglo y finalmente el registro base. Nuevamente, la dirección MIPS es especificada en parte por una constante y en parte por el contenido de un registro. El nombre real es *sw*, establecido por *store word* o *almacenar palabra*.

Ejemplo. Compilación al usar load y store. Asumiendo que la variable h está asociada con el registro $\$s2$ y que la dirección base del arreglo A está en $\$s3$. ¿Cuál es el código MIPS generado para la siguiente sentencia de asignación C?.

$$A[12] = h + A[8];$$

Respuesta. Aunque hay una sola operación en la sentencia C, ahora dos de los operandos están en memoria, así que se necesitan más instrucciones MIPS. Las dos primeras instrucciones son iguales al ejemplo anterior, excepto que ahora se usa el desplazamiento correcto para que el direccionamiento de la instrucción load seleccione $A[8]$ y la instrucción de suma deje el resultado en $\$t0$:

lw $\$t0$, 32($\$s3$) #Temporariamente el registro $t0 conserva $A[8]$

add $\$t0$, $\$s2$, $\$t0$ #Temporariamente el registro $t0 conserva $h + A[8]$

La instrucción final almacena la suma en $A[12]$, usando $48(4 \times 12)$ como el desplazamiento y el registro $\$s3$ como el registro base.

sw $\$t0$, 48($\$s3$) #Almacena $h + A[8]$ en $A[12]$

Load word y store word son las instrucciones que copian palabras entre la memoria y los registros en la arquitectura del MIPS. Otras marcas de computadoras usan otras instrucciones junto con load y store para transferir datos. Una arquitectura con tales alternativas es el intel x86.□

Muchos programas tienen más variables que los registros disponibles en las computadoras. En consecuencia, el compilador procura mantener las variables más usadas en registros y deja las restantes en memoria, usando loads y stores para mover las variables entre la memoria y los registros. El proceso de colocar las variables menos comunmente usadas (o las que serán necesarias más tarde) en memoria se llama *derramado* (en inglés *spilling*) de registros.

El principio de hardware que relaciona tamaño con velocidad, sugiere que la memoria debe ser más lenta que los registros debido a que la capacidad de la memoria es grande, mientras que la capacidad de los registros es poca. Indudablemente este es el caso; los accesos a los datos son más rápidos si el dato está en registro en lugar de memoria. Además, el dato es más útil cuando está en un registro. Una instrucción aritmética del MIPS, puede leer dos registros, opertar con ellos y guardar el resultado en un tercer registro. Una instrucción de transferencia de datos en MIPS solamente lee o escribe un operando, sin operar sobre él. Así, los registros requieren menor tiempo de acceso y tienen un mejor rendimiento (en inglés "throughtput") que la memoria, haciendo que los datos en registros sean más rápidos de acceder y más simples de usar. Acceder a los registros también usa menos energía que los accesos a memoria. Para alcanzar el desempeño máximo y ahorar energía, una arquitectura del conjunto de instrucciones debe tener un número suficiente de registros, y los compiladores deben usar los registros de manera eficiente.

Operandos constantes o inmediatos. Muchas veces un programa usará una constante en una operación – por ejemplo al incrementar un índice para que apunte al siguiente elemento del arreglo. En efecto, más de la mitad de las instrucciones aritméticas en MIPS tienen una constante como operando cuando se corre el benchmark [2] SPEC CPU2006.

[2]Benchmark es una técnica utilizada para medir el rendimiento de un sistema o una de sus componentes; un benchmark es el resultado de la ejecución de un programa informático (o un conjunto de programas) en una máquina, con el objetivo de estimar el rendimiento de un elemento concreto, y poder comparar los resultados entre máquinas similares. La palabra benchmark es un anglicismo traducible al español como "comparativa". Wikipedia

Usando solamente las instrucciones que se han visto hasta aquí, la única manera de operar con una constante es cargándola desde memoria mediante un load. (La constante debiera haber sido ubicada en memoria cuando el programa fue cargado.)

Por ejemplo, para sumar la constante 4 al registro $\$s3$, se podría usar el código:

lw $\$t0$, $DirConstante4(\$s1)$ #$t0 = constante 4

add $\$s3$, $\$s3$, $\$t0$ # $s3 = $s3 + $t0 ($t0=4)

asumiendo que $\$s1$ + $DirConstante4$ es la dirección en memoria de la constante 4.

Una alternativa para evitar el uso de la instrucción load es ofrecer versiones de las instrucciones aritméticas en las cuales un operando es una constante. Esta instrucción de suma rápida con un operando constante es llamada *suma inmediata* o *addi*. Para sumar 4 al registro $\$s3$, simplemente hay que escribir:

addi $\$s3$, $\$s3$, 4 # $s3 = $s3 + 4

Los operandos constantes ocurren frecuentemente y al incluir constantes en las instrucciones aritméticas, las operaciones son mucho más rápidas y usan menos energía que si las constantes fuesen cargadas desde memoria.

La constante cero tiene otro rol, se usa para simplificar el conjunto de instrucciones, ofreciendo algunas variantes útiles. Por ejemplo, la operación *move* es justamente una instrucción de suma donde un operando es cero. Por eso, MIPS destina el registro $zero como el operando cero; es un registro que está físicamente conectado al valor cero. (Como se puede esperar, es el registro número 0.) Considerando que el uso de constantes es algo frecuente, está más que justificada la inclusión de este tipo de operandos; éste es un ejemplo de uso de la gran idea de hacer el **caso frecuente más rápido**.

3. Representando instrucciones en la computadora

Ahora es posible explicar las diferencias entre la manera en que los humanos instruyen a las computadoras y la forma en que las computadoras ven a las instrucciones. Las instrucciones se mantienen en la computadora como una serie de señales electrónicas altas y bajas y pueden ser representadas como números. En efecto, cada parte de una instrucción puede ser considerada un número individual y al ubicar estos números uno al lado del otro se forma la instrucción.

Considerando que los registros son nombrados en las instrucciones, debe haber una convención para hacer corresponder nombres con números. En el lenguaje assembly del MIPS, los registros $\$s0$ al $\$s7$ se corresponden con los registros 16 al 23 y los registros $\$t0$ al $\$t7$ se corresponden con los registros del 8 al 15. Por esa razón, $\$s0$ significa registro 16, $\$s1$ significa registro 17, $\$s2$ significa registro 18, ... $\$t0$ significa registro 8, $\$t1$ significa registro 9, y así siguiendo.

Ejemplo. Traducción de instrucción assembly a instrucción de máquina. El refinamiento del lenguaje del MIPS es un ejemplo de traducción. A continuación se observa la versión en lenguaje real del MIPS de la instrucción presentada simbólicamente como:

add $\$t0$, $\$s1$, $\$s2$

primero como una combinación de números decimales y luego como números binarios.

Respuesta. La representación decimal es:

0	17	18	8	0	32

Cada uno de los segmentos de la instrucción se llaman *campos*. En combinación el primero y el último campo (conteniendo en este caso 0 y 32) le informan a la computadora MIPS que esta instrucción es una suma. El segundo campo contiene el número de registro que es el primer operando fuente de la operación de suma (17=$s1), el trecer campo guarda el número de registro del segundo operando (18=$s2). El cuarto campo tiene el número de registro que recibirá la suma (8=$t0). El quinto campo no se usa en esta instrucción, por lo que es puesto en cero. Esta instrucción suma el contenido del registro $s1 con el contenido del registro $s2 y ubica la suma en el registro $t0. Si los campos de la misma instrucción se representan en binario, quedaría:

000000	10001	10010	01000	00000	100000

□

Esta disposición de la instrucción es llamada *formato de instrucción*. Como se verifica al contar los bits, las instrucciones del MIPS requieren exactamente 32 bits – el mismo tamaño que la palabra de datos. En concordancia con el principio de diseño que establece que la simplicidad favorece la regularidad, todas las instrucciones MIPS son de 32 bits.

Esta representación de las instrucciones es diferente al lenguaje assembly, la versión numérica de las instrucciones se denomina *lenguaje de máquina* y una secuencia de estas instrucciones se denomina *código de máquina*. En general, para simplificar la lectura y escritura de instrucciones se usa representación hexadecimal.

Campos MIPS. Los campos MIPS tienen nombres asignados, para simplificar su designación:

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

El significado del nombre de los campos en las instrucciones MIPS:

- *op*: Operación básica de la instrucción, tradiccionalmente llamado el **opcod**.
- *rs*: El primer registro fuente.
- *rt*: El segundo registro fuente.
- *rd*: El registro destino. Se carga con el resultado de la operación.
- *shamt*: Cantidad de corrimientos en la operación *shift*.
- *funct*: Función. Este campo frecuentemente es llamado el *código de función*; selecciona la variante específica de la operación del campo op.

Surge un problema cuando una instrucción necesita campos más largos que aquellos considerados más arriba. Por ejemplo, la instrucción para cargar una palabra debe especificar dos registros y una constante. Si para especificar la dirección se empleara un campo de 5 bits, similar a los mostrados en el formato anterior, para especificar la constante con el desplazamiento desde el registro base, quedaría limitada al rango $[-2^4, 2^4 - 1]$. Este desplazamiento constante se usa para seleccionar elementos en arreglos o estructuras de datos y frecuentemente necesita ser más grande que ese rango limitado. El campo de 5 bits sería demasiado pequeño para ser de utilidad.

Por lo tanto, tenemos un conflicto entre el deseo de mantener todas las instrucciones del mismo tamaño y el deseo de tener un único formato de instrucción. Esto conduce al último principio de diseño de hardware:

Principio de diseño 3: Los buenos diseños requieren buenos compromisos.

El compromiso elegido por los diseñadores del MIPS es ajustar todas las instrucciones al mismo largo, lo que requiere diferentes clases de formatos de instrucción para diferentes clases de instrucciones. Por ejemplo el formato anterior es llamado *Tipo-R* (por registro) o *Formato-R*. Una segunda clase de formato de instrucción es llamado *Tipo-I* (por inmediato) o *Formato-I* y es usado por las instrucciones de transferencia de datos y las inmediatas. Los campos del Formato-I son:

op	rs	rt	constante o dirección
6 bits	5 bits	5 bits	16 bits

La dirección de 16 bits significa que una instrucción para cargar una palabra, puede acceder a cualquier palabra en la región de $\pm 2^{15}$ o 32,768 bytes ($\pm 2^{13}$ o 8,192 palabras) de la dirección en el registro base rs. De la misma manera la suma inmediata está limitada a constantes que no excedan $\pm 2^{15}$.

Queda claro que más de 32 registros podrían ser una complicación para este formato debido a que los campos rs y rt necesitarían más bits, haciendo difícil ajustar todo en una palabra.

Al considerar la siguiente instrucción para cargar una palabra:

lw $\$t0$, 32($\$s3$)

Aqui, se ubica un 19 (por $s3) en el campo rs, un 8 (por $t0) en el campo rt y un 32 en el campo de la dirección. Notar que el significado del campo rt ha cambiado; en una instrucción de carga de palabra el campo *rt* especifica el registro *destino*, es decir el que recibe el resultado de la carga.

Aunque múltiples formatos complican el hardware, se puede reducir la complejidad haciendo similares los formatos. Por ejemplo los tres primeros campos del Formato-R y del

formato-I son del mismo tamaño y tienen los mismos nombres; el largo del cuarto campo en el Tipo-I es igual a la suma de los largos de los tres últimos campos del Tipo-R.

Los formatos se distinguen por los valores del primer campo: cada formato tiene asignado un conjunto distinto de valores en el primer campo (op), de esa manera el hardware sabe si tratar la última mitad de la instrucción como tres campos (Tipo-R) o como un único campo. La figura 5 muestra los números usados en cada campo de las instrucciones MIPS consideradas aquí.

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

FIGURA 5. Decodificación de las instrucciones en MIPS

Ejemplo. Traducción de lenguaje assembly a lenguaje de máquina. Este ejemplo cubre el camino desde lo que escribe el programador hasta lo que ejecuta el procesador. Si $t1 tiene la base del arreglo A y $s2 corresponde a h, la sentencia de asignación:

$$A[300] = h + A[300];$$

se compila en :

lw $\$t0$, 1200($\$t1$)
add $\$t0$, $\$s2$, $\$t0$
sw $\$t0$, 1200($\$t1$)

¿Cuál es el código en lenguaje de máquina para estas tres operaciones?

Respuesta. Por conveniencia, primero se representa la instrucción en lenguaje de máquina usando números en decimal. Usando la Figura 5 se puede hacer la traducción de las tres instrucciones:

op	rs	rt	rd	dir/shamt	func
35	9	8	1200		
0	18	8	8	0	32
34	9	8	1200		

La instrucción lw es identificada por 35 (ver la Figura 5) en el primer campo (op). El registro base 9 ($t1) es especificado en el segundo campo (rs) y el registro destino 8 ($t0) es especificado en el tercer campo (rt). El desplazamiento para seleccionar $A[1200]$ (1200=300 X 4) es encontrado en el campo final (dirección).

La instrucción add que sigue es especificada con 0 en el primer campo (op) y 32 en el último campo (función). Los tres registros de operandos (18, 8 y 8) son encontrados en los campos segundo, tercero y cuarto y corresponden a $s2, $t0 y $t0.

La instrucción *sw* es identificada con 43 en el primer campo. El resto de esta última instrucción es igual a la instrucción *lw*.

Dado que $1200_{diez} = 0000\,0100\,1011\,0000_{dos}$ el equivalente binario a la forma decimal es:

op	rs	rt	rd	dir/shamt	func
100011	01001	01000	0000 0100 1011 0000		
000000	01010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Notar la similitud de las representaciones binarias de la primer y última instrucción. La única diferencia está en el tercer bit desde la izquierda.

La Tabla 1 resume las porciones del lenguaje de máquina del MIPS descriptos en este documento. La similitud en la representación binaria de las instrucciones relacionadas simplifica el diseño del hardware. Estas similitudes son otro ejemplo de la regularidad en la arquitectura MIPS.

Nombre	Formato	Ejemplo						Comentario
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
addi	I	8	18	17	100			addi $s1,$s2,100
lw	I	35	18	17	100			lw $s1, 100($s2)
sw	I	43	18	17	100			sw $s1, 100($s2)
Tamaño campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Las inst MIPS son de 32 bits
Formato-R	R	op	rs	rt	rd	shamt	función	Formato de inst aritméticas
Formato-I	I	op	rs	rt	Dirección			Formato de transf. de datos

Tabla 1. Arquitectura MIPS considerada en el documento

Referencias

[1] Patterson, David A. and Hennessy, John L., Computer Organization and Design, Fifth Edition: The Hardware/Software Interface, Morgan Kaufmann Publishers Inc., 2013