La arquitectura DLX.

Introducción.

En este capítulo describiremos la estructura básica de una arquitectura sencilla de carga/almacenamiento denominada DLX (Pronunciado "DeLuxe"). La arquitectura del conjunto de instrucciones DLX fue presentada por primera vez en la obra "*Computer Architecture: A Quantitative Approach*" de Jonh Hennessy y Dave Patterson. Según sus autores, DLX es el segundo computador poliinsaturado del mundo.

Se ha elegido la arquitectura del DLX sobre la base de observaciones de las primitivas utilizadas con más frecuencia en los programas. Las instrucciones más sofisticadas (y menos críticas en cuanto a rapidez) son implementadas por software mediante la ejecución de instrucciones múltiples.

Al igual que muchas de las máquinas de cargas/almacenamiento más recientes, DLX hace hincapié en los siguientes aspectos:

- Un conjunto sencillo de instrucciones de carga/almacenamiento. (Filosofía RISC).
- Adecuación al diseño de unidades segmentadas (*pipelining*).
- Un conjunto de instrucciones fácilmente decodificable. (Unidad de Control sencilla).
- Permite que los compiladores que generen código para él sean eficientes.

DLX proporciona un buen modelo de arquitectura para estudio, no sólo debido a la reciente popularidad de este tipo de máquinas, sino también a lo fácil de entender que es su arquitectura. Esta arquitectura se viene utilizando en muchas Universidades en innumerables cursos sobre Arquitectura de Computadores.

Estructura de la arquitectura DLX.

En esta sección describiremos la estructura básica del DLX y definiremos el conjunto de instrucciones del mismo.

- La arquitectura dispone de 32 registros de propósito general (GPRs) de 32 bits; el valor de R0 es siempre 0. Por otra parte, existe un conjunto de 32 registros en coma flotante (FPRs), los cuales pueden ser usados como registros de simple precisión (32 bits) o en parejas par-impar almacenando valores de doble precisión (64 bits). Se accede a los registros en coma flotante de doble precisión mediante los nombres F0, F2, ..., F28 y F30. Como es lógico, se proporcionan operaciones de simple y doble precisión. Existe un conjunto de registros especiales usados para acceder a la información sobre el estado de la máquina. El registro de estado de las operaciones en coma flotante es usado tanto en comparaciones como excepciones de coma flotante. Todos los movimientos desde o hacia los registros de estado se realizan a través de los registros de propósito general; también existe una instrucción de bifurcación que testea el valor del registro de estado de las operaciones en coma flotante.
- La memoria es direccionable por bytes, en modo "*Big Endian*" y con direcciones de 32 bits. Todas las referencias de memoria se realizan a través de cargas o almacenamientos entre memoria y los GPRs o FPRs. Los accesos que involucren a los GPRs pueden realizarse a un byte (8 bits), a media palabra o

halfword (16 bits) o a una palabra (32 bits). Los FPRs pueden ser cargados y almacenados mediante una palabra, para simple precisión, o dos palabras, para doble precisión (usando un par de registros). Todos los accesos a memoria sobre palabras o dobles palabras deben estar alineados, esto es, deben ser direcciones múltiples de cuatro. Además, existen instrucciones para el movimiento de datos entre los GPRs y los FPRs.

- Todas las instrucciones son de 32 bits y deben estar alineadas.
- Existen unos cuantos registros especiales que pueden ser transferidos desde o hacia los registros de propósito general. Un ejemplo de registro especial es el registro de estado de coma flotante, utilizado para almacenar información sobre los resultados de las operaciones en coma flotante. También veremos, en el siguiente capítulo, que existen otros registros especiales relacionados con las operaciones vectoriales; son el registro de longitud vectorial (vlr) y el registro de máscara vectorial (vm)

Tipos de datos

Aunque el tamaño del bus de datos de la arquitectura MIPS sea de 32 bits (palabra), se direccionan bytes individuales. Por tanto, la dirección de una palabra es realmente la misma que la de uno de los 4 bytes de la palabra. Por consiguiente, las direcciones de palabras secuenciales se diferencian en 4.

Dirección	Dato
0	1010110..0
4	1101011..1
8	1000111..1
12	1001111..0
...	...

Fig. 2.1. Direcciones reales de memoria MIPS

Recordemos que el tamaño de un **registro** es de 32 bits (palabra) y el DLX dispone de los siguientes registros:

- R0, ..., R31 : 32 registros enteros de propósito general.
- F0, F1, F2, ..., F31 : 32 registros en coma flotante de simple precisión.
- F0, F2, F4, ..., F30 : 16 registros en coma flotante de doble precisión (dos palabras).

El registro R0 siempre tiene el valor 0. Siempre que se utilice el registro R0, se suministra un 0 y el programador no puede cambiar el valor de ese registro.

Siempre que el DlxvSim lea un **número**, este será aceptado en notación **decimal**, en notación **hexadecimal** si los dos primeros caracteres del número son **0x** (Por ej. 0xf3a), o en notación **octal** si el primer carácter es el **0**.

Los tipos de datos inmediatos que puede manejar el DLX son :

- enteros de 16 bits con signo
- enteros de 16 bits sin signo
- números en coma flotante de simple precisión (una palabra)
- números en coma flotante de doble precisión (dos palabras)
- bytes
- cadenas de caracteres no terminadas por el byte NULL
- cadenas de caracteres terminadas por el byte NULL
- palabras (32 bits)

Expresiones simbólicas pueden ser usadas para especificar **direcciones de memoria**. La forma más simple es introducir un número, el cual será interpretado como la dirección de memoria. De forma más general, las expresiones de direcciones pueden estar formadas por números, símbolos (los cuales deben estar definidos en el fichero ensamblador actualmente cargado en la memoria de la máquina), operadores *, /, %, +, -, <<, >>, &&, |, y ^ (los cuales tienen el mismo significado y precedencia que en "C"), y paréntesis para agrupamiento.

Instrucciones de DLX.

Existen cuatro tipos de instrucciones:

- Cargas y almacenamientos.
- Operaciones sobre la unidad aritmético lógica.
- Bifurcaciones y saltos.
- Operaciones en coma flotante.

Instrucciones de carga y almacenamiento.

Cualquiera de los registros de propósito general o en coma flotante puede ser cargado o almacenado, excepto que el cargar sobre R0 no tiene ningún efecto. Hay un solo modo de direccionamiento: registro base + desplazamiento de 16 bits.

Las cargas de medias palabras (*halfwords*) y de bytes ponen el valor cargado en la parte baja del registro y la parte alta del mismo se rellena con el signo del valor cargado o con ceros, dependiendo del código de operación (que sea carga con signo o sin signo).

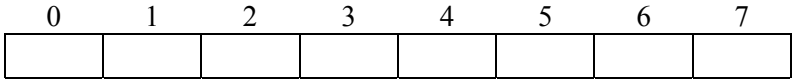
Los números en coma flotante de simple precisión ocupan un solo registro en coma flotante, mientras que los valores de doble precisión ocupan un par. Las conversiones entre simple y doble precisión deben ser realizadas explícitamente. El formato de los números en coma flotante sigue el estándar IEEE 754. La Figura 0.1 proporciona unos ejemplos sobre las operaciones de carga/almacenamiento. Por otra parte, una lista completa de las instrucciones viene reflejada más adelante.

Instrucción Ejemplo	**Instrucción**	**Significado**
`lw r1, 30(r2)`	Cargar palabra	$R1 \leftarrow_{32} M[30+R2]$
`lw r1, 1000(r0)`	Cargar palabra	$R1 \leftarrow_{32} M[1000+0]$
`lb r1, 40(r3)`	Cargar byte (con signo)	$R1 \leftarrow_{32} (M[40+R3]_0)^{24}$ ## $M[40+R3]$
`lbu r1, 40(r3)`	Cargar byte sin signo	$R1 \leftarrow_{32} 0^{24}$ ## $M[40+R3]$
`lh r1, 40(r3)`	Cargar media palabra (con signo)	$R1 \leftarrow_{32} (M[40+R3]_0)^{16}$ ## $M[40+R3]$ ## $M[41+R3]$
`lf f0, 50(r3)`	Cargar simple precisión	$F0 \leftarrow_{32} M[50+R3]$
`ld f0, 50(r2)`	Cargar doble precisión	F0 ## $F1 \leftarrow_{64} M[50+R2]$
`sw 500(r4), r3`	Almacenar palabra	$M[500+R4] \leftarrow_{32} R3$
`sf 40(r3), f0`	Almacenar simple precisión	$M[40+R3] \leftarrow_{32} F0$
`sd 40(r3), f0`	Almacenar doble precisión	$M[40+R3] \leftarrow_{32} F0$; $M[44+R3] \leftarrow_{32} F1$
`sh 502(r2), r3`	Almacenar media palabra	$M[502+R2] \leftarrow_{16} R3_{16..31}$
`sb 41(r3), r2`	Almacenar byte	$M[41+R3] \leftarrow_{8} R3_{24..31}$

Figura 0.1 Ejemplos de instrucciones de carga y almacenamiento en DLX

Todas las instrucciones utilizan un modo de direccionamiento simple y requieren que el valor de la dirección de memoria esté alineado.

Nota: el símbolo ## se utiliza para representar la concatenación de datos. La numeración de cada byte es la que se muestra a continuación:



Por lo tanto, el bit de signo es de 0.

Instrucciones aritméticas y lógicas.

Todas las operaciones sobre la ALU se realizan mediante instrucciones registro-registro. Las operaciones incluyen operaciones aritméticas y lógicas: suma (*add*), resta (*sub*), and, or, xor y desplazamientos (*shifts*). Además se proporcionan las versiones con modo de direccionamiento inmediato de todas esas instrucciones, con valor inmediato de 16 bits y con extensión de signo. Además, la operación lhi ("*load high immediate*") carga la mitad superior del registro, mientras pone la mitad inferior del mismo a ceros. Esto permite construir una constante completa de 32 bits con solamente dos instrucciones.

Hay también instrucciones de comparación, las cuales comparan dos registros (=, ≠, <, >, >=, <=). Si la condición es verdadera, estas instrucciones ponen un 1 en el registro destino (para representar verdadero "*true*"); en cualquier otro caso ponen un cero(para representar falso "*false*"). Debido a que estas instrucciones inicializan unos registros se llaman "s_" (*set if _*). Por ejemplo "slt" (*set if less than*). Existen también formas inmediatas para estas instrucciones de comparación. La Figura 0.2 proporciona algunos ejemplos de las instrucciones aritmético lógicas

Instrucción ejemplo	**Instrucción**	**Significado**
`add r1, r2, r3`	Suma	R1 ← R2 + R3
`addi r1, r2, #3`	Suma inmediato	R1 ← R2 + 3
`lhi r1, #42`	Carga alto inmediato	R1 ← 42 ## 0^{16}
`sll r1, r2, #5`	Desplazamiento lógico a izquierdas	R1 ← R2 << 5
`slt r1, r2, r3`	Comparación (menor que)	if (R2 < R3) R1←1 else R1←0

Figura 0.2 Ejemplos de instrucciones aritmético-lógicas del DLX; ambas se presentan con y sin la forma inmediata.

Si queremos cargar una cte. Como por ejemplo 13 en un registro como R5 haremos

```
ADDI    R5, R0, #13
```

y aparecerá en R5 = 0x0000000D.

Vamos a ver otra forma de cargar una constante en un registro, que además es la más aconsejable si resulta que esta constante es un número real. Sea el número PI = 3.1415926 que queremos cargarlo en el registro de coma flotante doble precisión F2:

```
PI:         .double  3.1415926
...
            LD      F2, PI
```

Instrucciones de salto y bifurcación.

El control de flujo del programa se realiza a través de una serie de saltos y bifurcaciones. Las cuatro instrucciones de salto se diferencian por las dos maneras de especificar la dirección destino y si se hace o no se hace un enlace. Dos instrucciones de saltos utilizan una dirección relativa al contador de programa con un desplazamiento (*offset*) de 26 bits con signo, (el contador de programa corresponde a la instrucción que sigue secuencialmente al salto); las otras dos instrucciones de salto especifican un registro que contiene la dirección de salto. Hay dos formas de salto: salto plano y salto con enlace (usado en llamadas a procedimientos). La última forma de salto almacena la dirección de retorno en el registro R31.

Todas las bifurcaciones son condicionales. La condición de bifurcación es especificada por la instrucción, la cual puede testear si el registro fuente es o no es cero; el registro fuente puede ser el resultado de una comparación. La dirección a la que saltar se especifica mediante un desplazamiento de 16 bits con signo que se suma al contador de programa. La Figura 0.3 proporciona algunos ejemplos típicos de instrucciones de salto y de bifurcación.

Instrucción ejemplo	Instrucción	Significado
`j etiqueta`	Salto	PC ← etiqueta; $((PC+4)-2^{25}) \le$ etiqueta $< ((PC+4)+2^{25})$
`jal etiqueta`	Salto y enlace	R31 ← PC+4; PC ← etiqueta; $((PC+4)-2^{25}) \le$ etiqueta $< ((PC+4)+2^{25})$
`jr r3`	Salto con registro	PC ← R3
`jalr r2`	Salto y enlace con registro	R31 ← PC+4; PC ← R2
`beqz r4, etiqueta`	Bifurcación si igual a cero	if (R4 == 0) PC ← etiqueta; $((PC+4)-2^{15}) \le$ etiqueta $< ((PC+4)+2^{15})$
`bnez r4, etiqueta`	Bifurcación si distinto a cero	if (R4 != 0) PC ← etiqueta; $((PC+4)-2^{15}) \le$ etiqueta $< ((PC+4)+2^{15})$

Figura 0.3 Ejemplos de instrucciones típicas de control de flujo en el DLX.

Todas las instrucciones de control de flujo, excepto aquellas que proporcionan la dirección de un registro, son relativas al PC. Si se utiliza una instrucción de bifurcación con registro operando R0 entonces la bifurcación es incondicional, pero en este caso el compilador preferirá utilizar una instrucción de salto, ya que el desplazamiento es mayor en estas.

Instrucciones de coma flotante.

Las instrucciones en coma flotante manejan los registros en coma flotante e indican si la operación que se va a realizar es de simple o doble precisión. Las operaciones de simple precisión pueden ser aplicadas a cualquiera de los registros, mientras que las operaciones de doble precisión se aplican únicamente a las parejas par-impar (Por ejemplo: F4, F5), las cuales son designadas por el número del registro par. Las instrucciones de carga y almacenamiento de los registros en coma flotante mueven datos entre memoria y registros, tanto en simple como en doble precisión. Las operaciones movf y movd copian un registro de simple precisión (movf) o doble precisión (movd) a otro registro del mismo tipo. Las operaciones movfp2i y movi2fp mueven datos entre un registro en coma flotante de simple precisión y un registro entero de propósito general; mover un valor de doble precisión a dos registros enteros, requiere la utilización de dos instrucciones.

Además, se proporcionan instrucciones de multiplicación y división enteras que actúan sobre registros en coma flotante de 32 bits, así como instrucciones de conversión de entero a coma flotante y viceversa.

Las operaciones en coma flotante son: suma, resta, multiplicación y división; se utiliza un sufijo "d" para doble precisión y un sufijo "f" para simple precisión (Por ejemplo: addd, addf, subd, subf, ...) Las comparaciones de números en coma flotante modifican el registro de estado asociado a las operaciones en coma flotante, el cual puede ser testeado mediante un par de instrucciones de bifurcación: bfpt y bfpf, bifurcar si verdadero (*true*) y bifurcar si falso (*false*). En la Figura 0.4 podemos ver algunos ejemplos típicos de operaciones en coma flotante.

Instrucción ejemplo	**Instrucción**	**Significado**
`addf f1, f2, f3`	Suma simple precisión	`F1 ← F2 + F3`
`addd f0, f2, f4`	Suma doble precisión	`(F0#F1) ← (F2#F3) + (F4#F5)`
`bfpt #eti`	Salta si FPSR es cero	`Si (FPSR==1) PC←PC+4+eti`
`cvtf2d f0, f3`	Convertir simple a doble precisión.	`(F0#F1) ← F3`
`gtd f0, f2`	Comparación de números en coma flotante de doble precisión.	`If ((F0#F1) > (F2#F3)) FPSR←"1"`

Figura 0.4 Ejemplos de instrucciones en coma flotante de simple y doble precisión.

A continuación se proporciona una lista de todas las instrucciones y su significado.

Instrucciones de transferencia de datos.

Transfieren datos entre registros y memoria o entre registros enteros y registros en coma flotante o registros especiales; el único modo de direccionamiento de memoria que se utiliza es un desplazamiento de 16 bits más el contenido de un registro de propósito general.

`lb, lbu, sb`	Cargar byte, cargar byte sin signo, almacenar byte.
`lh, lhu, sh`	Cargar media palabra, cargar media palabra sin signo, almacenar media palabra.
`lw, sw`	Cargar palabra, almacenar palabra.
`lf, ld, sf, sd`	Cargar número en coma flotante de simple precisión (lf), cargar número en coma flotante de doble precisión (ld), almacenar número en coma flotante de simple precisión (sf) y almacenar número en coma flotante de doble precisión (sd).
`movf, movd`	Copiar un registro en coma flotante a otro registro de coma flotante y copiar un par de registros en coma flotante a otro par (si doble precisión).
`movfp2i, movi2fp`	Mover valores de 32 bits de registros en coma flotante a registros enteros y mover valores de 32 bits de registros enteros a registros en coma flotante.

Instrucciones aritmético-lógicas.

Operaciones sobre los datos enteros o lógicos que se encuentran en registros de propósito general; la aritmética con signo causa un *trap* en caso de desbordamiento.

`add, addi, addu, addui`	Suma, suma inmediata (todos los valores inmediatos son de 16 bits), suma sin signo y suma sin signo inmediata.
`sub, subi, subu, subui`	Resta, resta inmediata, resta sin signo, resta sin signo inmediata.
`mult, multu, div,`	Multiplicación, multiplicación sin signo, división y división sin signo. Los

`divu`	operandos deben ser registros de enteros; todas las operaciones toman y devuelven valores de 32 bits.
`and, andi`	AND y AND inmediato.
`or, ori, xor, xori`	OR, OR inmediato, OR exclusiva y OR exclusiva inmediato.
`lhi`	Carga la parte alta del registro con un valor inmediato.
`sll, srl, sra, slli, srli, srai`	Desplazamientos: inmediatos (s_i) o normales (s_); lógicos a la izquierda (ll), lógicos a la derecha (rl) y aritméticos a la derecha (ra)
`s_, s_i`	Comparación y asignación: "_" puede ser LT, GT, LE, GE, EQ, EN.

Instrucciones de control de flujo.

Bifurcaciones y saltos condicionales; relativos al PC o mediante un registro de propósito general.

`beqz, bnez`	Bifurcación si registro igual/no igual a cero; desplazamiento de 16 bits desde PC+4.
`bfpt, bfpf`	Test del bit de comparación del registro de estado de las operaciones en coma flotante y bifurca; desplazamiento de 16 bits desde PC+4.
`j, jr`	Saltos: desplazamiento de 26 bits desde el PC (j) o dirección de salto presente en el registro (jr).
`jal, jalr`	Salta y enlaza: salva el PC+4 en R31, el destino es relativo al PC (jal) o se proporciona en un registro (jalr).
`trap`	Genera una llamada al sistema.
`tfe`	Vuelve al código del usuario desde una excepción; restaurar modo de usuario.

Instrucciones en coma flotante.

Realiza las operaciones en coma flotante, tanto en simple como en doble precisión.

`addd, addf`	Suma números de doble o simple precisión.
`subd, subf`	Resta números de doble o simple precisión.
`multd, multf`	Multiplicación de números de doble o simple precisión.
`divd, divf`	División de números de doble o simple precisión.
`cvtf2d, cvtf2i, cvtd2f, cvtd2i, cvti2f, cvti2d`	Instrucciones de conversión; CVTx2y convierte del tipo x al tipo y, donde x e y pueden ser: i (entero), f (coma flotante de simple precisión) o d (coma flotante de doble precisión). Ambos operandos han de estar en los registros de coma flotante.
`_d, _f`	Comparación en simple o doble precisión: "-" puede ser lt, gt, le, ge, eq, en; modifican, en consecuencia, el registro de estado de las operaciones en coma flotante.

Formatos de las instrucciones.

Todas las instrucciones son de 32 bits con un código de operación principal de 6 bits. Posee tres únicos formatos: tipo R, tipo I y tipo J. Aunque los formatos múltiples complican el hardware, podemos reducir la complejidad manteniendo formatos similares. Por ejemplo, los tres primeros campos de los formatos tipo R y tipo I tienen los mismos nombres, el cuarto campo del tipo I tiene igual longitud que los últimos tres del tipo R. Esta similitud de en la representación de las instrucciones simplifica el diseño hardware.

Los formatos se distinguen por los valores del primer campo: cada formato se asigna a un conjunto de valores del primer campo (op) para que el hardware sepa como tratar los siguientes campos de la instrucción.

MIPS no hace uso de códigos de condición. Si una instrucción genera una condición, los indicadores correspondientes se almacenan en un registro de uso general. Esto elimina la necesidad de una lógica especial para tratar con códigos de condición.

Instrucciones del tipo R

En las instrucciones aritmético/lógicas se deben especificar los dos operandos, sobre los que se realiza la operación, más un tercer operando que almacenará el resultado de la misma. Esto implica la necesidad de especificar tres registros. El formato de dichas instrucciones es el siguiente:

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

- *op* : operación de la instrucción
- *rs* : primer registro del operando fuente
- *rt* : segundo registro del operando fuente
- *rd* : registro del operando destino; obtiene el resultado de la operación
- *shamt* : cantidad de desplazamiento
- *funct* : función; este campo selecciona la variante de la operación del campo op

Instrucciones del tipo I

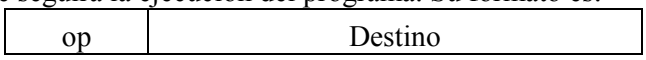
Se utiliza para las instrucciones de transferencia de datos en las que necesitamos especificar dos registros y una dirección. Por tanto, los campos de este formato son:

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

- *op* : operación de la instrucción
- *rs* : registro índice
- *rt* : registro destino u origen de la información
- *dirección* : dirección base de la memoria (para los autores es el desplazamiento respecto al registro índice *rs*)

Instrucciones del tipo J.

Se emplea en algunas instrucciones de bifurcación. Sólo se necesita especificar la dirección de destino donde se seguirá la ejecución del programa. Su formato es:



- *op* : operación de la instrucción
- *destino* : dirección destino del salto

El campo *destino* se concatena con los 4 bits más significativos del contador de programa para formar la dirección absoluta de bifurcación. Como todas las direcciones acaban en 00 (se accede por palabras), *destino* especificada los bits 2 a 27 y los bits 28 a 31 se cogen del PC.

Modos de direccionamiento

Direccionamiento inmediato

El operando es una constante que está en la misma instrucción.

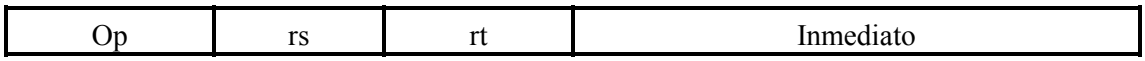


Fig. 3.1. Direccionamiento inmediato

Direccionamiento mediante registro

El operando esta almacenado en un registro.

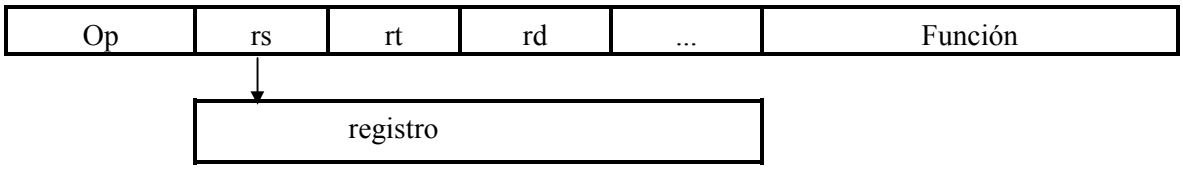


Fig. 3.2. Direccionamiento mediante registro

Direccionamiento relativo a registro

El operando está en la posición de memoria cuya dirección es la suma de un registro y una dirección de la instrucción.

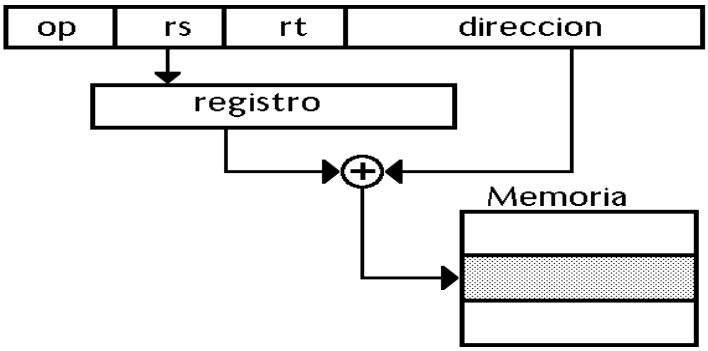


Fig. 3.3. Direccionamiento relativo a registro

Direccionamiento relativo al PC

La dirección es la suma del PC y una constante de la instrucción.

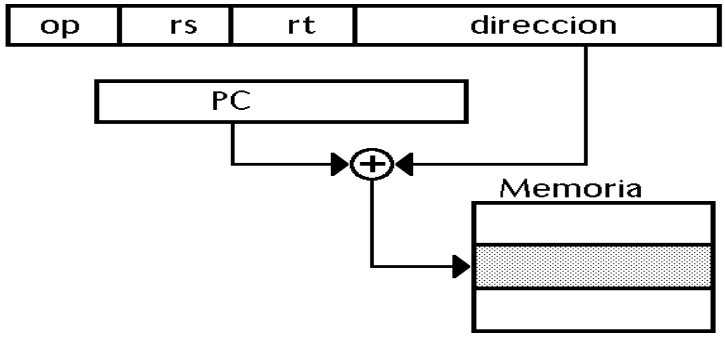


Fig. 3.4. Direccionamiento relativo al PC

En estas instrucciones, en lenguaje ensamblador aparece el salto relativo a una etiqueta. El compilador traduce la dirección de la etiqueta a un valor a sumar a PC+4.

Programación en ensamblador

La representación simbólica de las instrucciones se denomina **lenguaje ensamblador**. Por otra parte, llamamos **lenguaje máquina** al equivalente numérico que ejecuta la máquina. Los programas hechos en lenguaje ensamblador se traducen a lenguaje máquina mediante un **ensamblador**.

El ensamblador del DLX (WinDLX) acepta programas en lenguaje ensamblador del DLX que estén en ficheros con extensión *.s. Estos ficheros contendrán líneas de texto respetando las siguientes normas:

- Las etiquetas están definidas por un grupo de caracteres, no blancos, que comienza con una letra, un signo de subrayado, o el signo del dólar ($), y seguido inmediatamente por dos puntos (:). Estas etiquetas están asociadas con la siguiente dirección en la cual el código del fichero va a ser almacenado. Las etiquetas pueden ser accedidas en cualquier lugar del fichero, y en ficheros cargados con posterioridad si dicha etiqueta ha sido declarada como global (directiva .global).
- Los comentarios se comienzan con un punto y coma (;), y continuarán hasta el final de la línea.
- Las constantes pueden ser introducidas con o sin un signo que las precedan.

Mientras el ensamblador está procesando un fichero, los datos y las instrucciones que van siendo ensambladas son almacenadas en memoria basándose en el puntero de códico o en el puntero de datos. Cuál de los dos es usado no depende del tipo de la información, sino de si la directiva más reciente ha sido .data o .text. Inicialmente el programa es cargado en el segmento de código.

Directivas

El ensamblador soporta numerosas directivas, las cuales afectan a la forma en que se carga en memoria el código ensamblado. Estas directivas deben ser introducidas en el lugar en el que normalmente se introducen las instrucciones y sus argumentos.

A continuación se proporciona una lista de las directivas aceptadas por el ensamblador del DLX, indicando su significado y los resultados que provocan :

- .**align** n : Provoca que el código o los datos que vienen a continuación sean cargados en la parte más alta de la dirección con los n bits más bajos puestos a cero (Es decir, en la siguiente dirección más cercana, que sea igual o más grande que la actual dirección y que además sea múltiplo de 2^{n-1}).
- .**ascii** "cadena1", "cadena2",... : Almacena las cadenas, presentes en la línea de la directiva, como una lista de caracteres. Las cadenas no son terminadas por el byte NULL.
- .**asciiz** "cadena1", "cadena2",... : Es similar a la directiva .ascii, pero cada cadena se almacena terminándola con el carácter NULL (como las cadenas en "C").
- .**byte** "byte1", "byte2",... : Almacena los bytes, listados en la línea de la directiva, de forma secuencial en la memoria de la máquina.
- .**data** [Dirección] : Provoca que el siguiente código y los siguientes datos sean almacenados en la zona de datos (data). Si se proporciona una "Dirección", la información será cargada comenzando en esa dirección, en cualquier otro caso, el último valor del puntero de datos será utilizado. Si estábamos leyendo código basándonos en el puntero de código, almacena esa dirección de manera que podamos continuar desde ella más tarde (usando una directiva .text).
- .**double** numero1,numero2,... : Almacena los números, presentes en la línea de la directiva, de manera secuencial en la memoria como números en coma flotante de doble precisión.
- .**float** numero1,numero2,... : Almacena los números, presentes en la línea de la directiva, de manera secuencial en la memoria como números en coma flotante de simple precisión.
- .**global** etiqueta : Hace que la etiqueta pueda ser referenciada en el código que se encuentre en ficheros cargados con posterioridad al actual.
- .**space** tamaño : Desplaza el puntero actual de almacenamiento "tamaño" bytes hacia adelante (con el propósito de dejar algún espacio libre en memoria).
- .**text** [Dirección] : Provoca que el siguiente código y los siguientes datos sean almacenados en la zona de código. Si se proporciona una "Dirección", la información será cargada comenzando en esa dirección, en cualquier otro caso, el último valor del puntero de código será utilizado. Si estábamos leyendo datos basándonos en el puntero de datos, almacena esa dirección de manera que podamos continuar desde ella más tarde (usando una directiva .data).
- .**word** palabra1,palabra2,... : Almacena las palabras, presentes en la línea de la directiva, de forma secuencial en la memoria.

Llamadas al sistema

El DLX simula las llamadas al sistema (dependiendo del valor de num) mediante la siguiente instrucción:

trap num

Se espera que en el registro R14 se encuentre la dirección de memoria donde está almacenado el primer argumento de la llamada al sistema. Así mismo, se espera que los siguientes argumentos (si los hay)

estén almacenados a continuación de la dirección indicada en R14, y por supuesto en el orden adecuado. El resultado de la trap simulada es almacenado en R1.

En el caso de que haya algún error en las llamadas al sistema, el número del error (es decir, el motivo del error) se almacena en la posición 0 de la memoria. Los posibles valores de error son:

ENOFILE, ENOENT= 2	Fichero no encontrado	EINVACC = 12	Código de acceso no válido
ENOPATH = 3	Camino no encontrado	EINVDRV = 15	Unidad de disco incorrecta
EMFILE = 4	Demasiados ficheros abiertos.	ENMFILE = 18	No se puede crear o abrir más ficheros
EACCESS = 5	Permiso de acceso denegado.	EINVAL = 19	Argumento no válido
EBADF = 6	Número (manejador) de fichero incorrecto	EEXIT = 35	el fichero especificado aún existe
EINVFMT = 11	Formato incorrecto		

Los posibles valores que pueden tener los traps son :

- **trap 0** : Termina la ejecución del programa, pero sin terminar las operaciones pendientes.
- **trap 1** : manejador=**open**(camino, acceso [, modo]). Abre el fichero especificado por *camino* (Normalmente el *camino* será una dirección que apuntará a una zona de memoria donde está almacenado el string que describe el nombre del fichero) y lo prepara para lectura y/o escritura. El parámetro *acceso* especifica el modo de acceso al fichero ,entre los 9 posibles:

O_RONLY = 1	Abre fichero solo para lectura
O_WRONLY = 2	Abre fichero solo para escritura
O_RDWR = 4	Abre fichero para lectura y escritura
O_CREAT = Ox0100	Crea y abre el fichero
O_TRUNC = Ox0200	Abre el fichero, pero truncándolo.
O_EXCL = Ox0400	Abre el fichero de forma exclusiva.
O_APPEND = Ox0800	Añade al final del fichero
O_TEXT = Ox4000	Abre el fichero en modo texto.
O_BINARY = Ox8000	Abre el fichero en modo binario

Y el parámetro *modo* especifica los permisos de acceso al mismo, que pueden ser dos:

S_IREAD = 0X0100	El propietario puede leer.
S_IWRITE = 0X0080	El propietario puede escribir.

En caso de que la llamada se realice con éxito, en el registro R1 se almacenrá el manejador asociado al fichero que acaba de ser abierto. Este valor debe ser guardado para ser utilizado por otras llamadas al sistema. Si esta llamada no ha tenido éxito, aparecerá un -1 en R1 indicando que ha habido algun error y en la posición 0 de memoria aparecerá uno de los siguientes números de error: ENOENT, EMFILE, EACCESS, EINVAC.

- **trap 2** : resultado = **close**(manejador). Cierra un fichero asociado a un *manejador* que ha sido obtenido anteriormente mediante una llamada a open(...).

  Si no hay errores en R1 se almacena un 0, sino un -1 y en la posición de memoria 0 el valor: EBADF.

- **trap 3** : leidos = **read**(manejador, destino, número). Lee de un fichero asociado a un *manejador*, obtenido anteriormente mediante una llamada a open(...), el número de bytes especificado por *número*, almacenando esos bytes en la dirección *destino*.

Como manejador se puede utilizar el 0 (entrada estándar) para leer de la ventana de I/O del simulador.

Si no hay errores, en R1 se almacena el número de bytes leidos ó 0 si se encontró el final de fichero; en caso contrario se almacena un -1 y en la posición de memoria 0 aparecerá: EACCESS ó EBADF.

- **trap 4** : escritos = **write**(manejador, destino, número). Escribe en un fichero asociado a un *manejador*, obtenido anteriormente mediante una llamada a open(...), el número de bytes especificado por *número*, y los cuales están almacenados en la dirección *destino*.

  Como manejador se puede utilizar el 1 (salida estándar) o el 2 (salida estándar de error) para escribir en la vantana del simulador.

  Si no hay errores, en R1 se almacena el número de bytes escritos o en caso contrario se almacena un -1 y en la posición de memoria 0 aparecerá: EACCESS ó EBADF.

- **trap 5** : resultado = **printf**(formato [, argumentos,...]). Envía la salida formateada al dispositivo estándar de salida (normalmente la pantalla, manejador = 0).

  Si no hay errores, en R1 se almacena el número de bytes enviados ó -1 (EOF) para indicar algún error; y en la posición de memoria 0 aparecerá: EACCESS ó EBADF.

- **trap 6** : Termina la ejecución del programa, terminando también las operaciones que queden pendientes en las unidades de proceso. WinDLX no distingue entre *trap 0* y *trap6.*

Observesé que los parámetros de las *traps* de la 1 a la 5 son los mismos que sus correspondientes funciones en "C". En aquellos casos en los que el argumento de una de las llamadas al sistema sea un string, se especificará, como parámetro de la llamada, la dirección de memoria donde éste se encuentre almacenado.

Para finalizar la ejecución de un programa en ensamblador del DLX debemos de utilizar la *trap 0* o la *trap 6.*

Subrutinas

Una subrutina es una forma en la que se estructuran los programas. Las subrutinas hacen más fáciles de comprender los programas y permiten que el código sea reutilizado. Un repertorio de instrucciones debe proporcionar algún método para bifurcar a un procedimiento y después volver desde el procedimiento a la instrucción inmediatamente siguiente al punto de llamada. Los programadores también deben tener convenios que gobiernen cómo pasar los parámetros y cómo soportar el anidamiento de llamadas a procedimientos.

DLX proporciona una instrucción que bifurca a una dirección y simultáneamente guarda la dirección de la siguiente instrucción en el registro R31 :

jal etiqueta26

Se forma un enlace (almacenado en R31) al sitio que llama para permitir que el procedimiento vuelva a la dirección adecuada. Tenemos otra instrucción para realizar la bifurcación de vuelta :

jr R31

Esta instrucción bifurca a la dirección almacenada en R31.

Si un procedimiento quisiera llamar a otro procedimiento, el programador necesitaría guardar el valor antiguo del registro R31, ya que la nueva **jal** machacaría la antigua dirección de vuelta. La estructura de datos ideal para almacenar registros en memoria es la pila.

Necesitamos un convenio que gobierne el paso de argumentos, o parámetros, pasados a un procedimiento. El convenio software de DLX es poner los parámetros en los registros R4 a R7. Si un procedimiento necesita invocar a otro procedimiento, estos registros de parámetros pueden ser guardados y restaurados desde la pila como direcciones de vuelta. En general, si un procedimiento modifica los registros utilizados por la rutina actual, debe haber un convenio para guardar y restaurar los registros a través de llamadas a procedimientos. Nosotros seguiremos el siguiente convenio: El procedimiento invocado es responsable de guardar y restaurar cualquier registro que pueda utilizar. El programa invocador utiliza los registros sin preocuparse de restaurarlos después de una llamada.

¿Qué ocurre si hay más de cuatro parámetros? El procedimiento espera entonces que los cuatro primeros parámetros estén en los registros R4 a R7 y el resto en memoria, direccionable vía el puntero de pila.

En DLX, las pilas crecen desde las direcciones superiores hasta las inferiores. Por tanto, para introducir valores en la pila hay que decrementar el puntero de pila y para sacar valores de la pila hay que incrementar el puntero de pila.

Bibliografía.

"ARQUITECTURA DE COMPUTADORES: UN ENFOQUE CUANTITATIVO".
Jonh Hennessy y Dave Patterson.
Editorial McGraw Hill. 1993.

"THE DLX INSTRUCTION SET ARCHITECTURE HANDBOOK"
Philip M. Sailer y David R. Kaeli.
Morgan Kaufmann Publishers, Inc. San Francisco, California.

Cualquier error detectado en estos apuntes es responsabilidad del autor, el cual agradecerá que para su corrección se le sea notificado enviándole un mensaje a la dirección:

Vicente.Arnau@uv.es