



Módulo 09
Modos de Direcccionamiento
(Pt. 2)

 Organización de Computadoras
Depto. Cs. e Ing. de la Comp.
Universidad Nacional del Sur 


Copyright

- Copyright © 2011-2023 A. G. Stankevicius
- Se asegura la libertad para copiar, distribuir y modificar este documento de acuerdo a los términos de la **GNU Free Documentation License**, Versión 1.2 o cualquiera posterior publicada por la Free Software Foundation, sin secciones invariantes ni textos de cubierta delantera o trasera
- Una copia de esta licencia está siempre disponible en la página <http://www.gnu.org/copyleft/fdl.html>
- La versión transparente de este documento puede ser obtenida de la siguiente dirección:
<http://cs.uns.edu.ar/~ags/teaching>

 Organización de Computadoras - Mg. A. G. Stankevicius 2

Contenidos

- Tipos de instrucciones
- Formato de instrucción
- Modos de direccionamiento
 - Directo vs. Indirecto
 - Absoluto vs. Relativo
- Código automodificable
- Código independiente de la posición
- La arquitectura **OCUNS**

 Organización de Computadoras - Mg. A. G. Stankevicius 3

Modos implementados en HW

- Los diseñadores de las arquitecturas han ensayado diversos modos de direccionamiento para atender a estos cuestionamientos
- No obstante, pocas arquitecturas implementan la totalidad de los modos de direccionamiento
- En general se adopta un subconjunto que es implementado en hardware, dejando el resto para ser simulado vía software

Modos implementados en HW

- Por caso, el Motorola 6809 implementa en HW:

Nombre	Mnemónico	Op. Accedido
Registro	Ri	Ri
Registro Indirecto	(Ri)	M[Ri]
Absoluto	addr	M[addr]
Absoluto Indirecto	(addr)	M[M[addr]]
Inmediato	#const	const
Auto-incremento	(Ri)+	M[Ri]; Ri++
Auto-decremento	-(Ri)	--Ri; M[Ri]
PC Relativo	PC(const)	M[PC + const]
Base	Ri(addr)	M[Ri + addr]
Indexado	addr(Ri)	M[addr + Ri]
Base indexado	Ri(Rj)	M[Ri + Rj]

Inmediato

- Interpretación: el contenido del campo argumento es el operando

- Ejemplos:

instrucción

add	#5
-----	----

→ add #5 (pila)

→ add R0, #5 (CISC)

→ add R0, R0, #5 (RISC)

0000:	28	A0
:	:	00
9FFF:	1E	F5
A000:	02	02
A001:	A0	F1
A002:	05	EA
:	:	:
FFFF:	B2	00
	memoria	pila del programa

Inmediato

• Características:

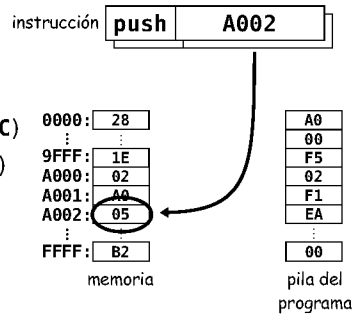
- También se lo conoce como modo literal
- Altamente eficiente, pues no requiere acceder a memoria (el operando es obtenido al acceder inicialmente a memoria para traer la instrucción)
- El rango puede estar acotado a la cantidad de bits disponibles en el formato de la instrucción
- Se lo utiliza para operar con valores constantes
- En caso de no estar soportado por el hardware, se lo puede simular a nivel de lenguaje ensamblador

Absoluto

• Interpretación: el contenido del campo argumento es la ubicación en memoria del operando

• Ejemplos:

- `push A002` (stack)
- `mov R0, [A002]` (CISC)
- `load R0, A002` (RISC)



Absoluto

• Características:

- También se lo conoce como modo memoria directo
- Implica un acceso adicional a memoria (aparte del acceso inicial para traer la instrucción)
- La instrucción contiene la dirección efectiva del argumento
- El espacio de direcciones está acotado por la cantidad de bits permitidos por el formato de instrucción
- No es práctico para recorrer estructuras de datos (¿por qué razón?)

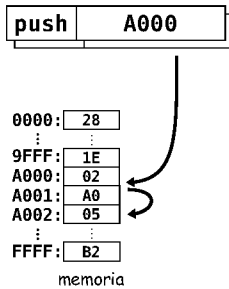
Código automodificable

- El modo directo es apropiado para acceder a tipos de datos simples (por caso, variables), no así a tipos de datos complejos
 - Para recorrer tipos complejos (tales como arreglos o registros), tenemos que modificar en tiempo de ejecución al campo argumento de la instrucción
 - Este tipo de código se lo denomina automodificable
 - El emplear código automodificable complica la interpretación del significado de un programa
 - La alternativa más adecuada consiste en hacer uso de referencias indirectas

Directo vs. indirecto

- La idea detrás del modo indirecto consiste en agregar un paso adicional en la búsqueda del argumento:

- En el modo directo accedemos al valor deseado
 - En el indirecto accedemos a una dirección de memoria en la cual encontraremos el valor deseado
- Este paso adicional implica un segundo acceso a memoria

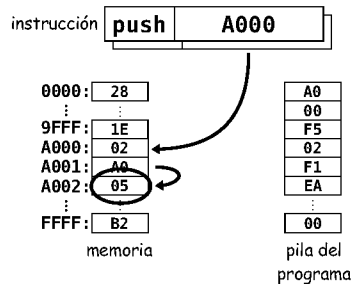


Absoluto indirecto

- Interpretación:** el contenido del campo argumento es la ubicación en memoria de la dirección del operando

Ejemplo:

- push (A000) (stack)



Absoluto indirecto

• Características:

- También se lo conoce como memoria indirecto
- Implica dos accesos adicionales a memoria (aparte del acceso inicial para traer la instrucción)
- Extremadamente flexible, permite recorrer cualquier estructura de datos, si bien está actualmente en desuso por lo oneroso en tiempo de ejecución
- De ser requerida su flexibilidad, se lo puede simular haciendo uso de otros modos de direccionamiento

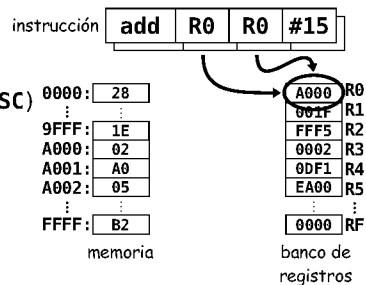
Registro

• Interpretación: el contenido del campo argumento es la ubicación del operando dentro del banco de registros

• Ejemplos:

→ add R0, #15 (CISC)

→ add R0, R0, #15 (RISC)



Registro

• Características:

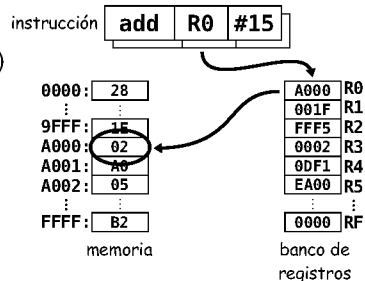
- También se lo conoce como registro directo
- Análogo al modo absoluto, usando al banco de registros del procesador como espacio de direcciones
- Altamente eficiente ya que no requiere acceder a memoria (aparte del acceso inicial)
- Se puede codificar con muy pocos bits (puesto que direcciona un espacio bastante acotado)
- Junto al modo registro indirecto permite simular otros modos de direccionamiento

Registro indirecto

- **Interpretación:** el contenido del campo argumento es la ubicación dentro del banco de registros de la dirección del operando

- **Ejemplo:**

→ `add [R0], #15 (CISC)`



Registro indirecto

- **Características:**

- Análogo al modo absoluto indirecto, pero recuperando la dirección del operando del banco de registros
- Se puede codificar con muy pocos bits (puesto que direcciona un espacio bastante acotado)
- No obstante, al ser indirecto retiene la capacidad de direccionar la totalidad del espacio de direcciones
- Resulta altamente flexible, permitiendo recorrer estructuras de datos complejas
- Más eficiente que el modo absoluto indirecto, ya que sólo requiere un acceso a memoria (aparte del inicial)

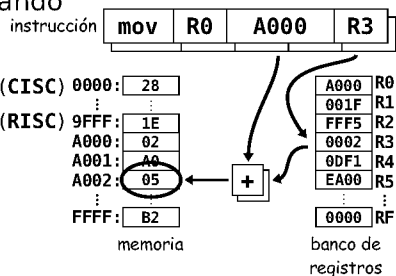
Indexado

- **Interpretación:** el campo argumento contiene una dirección de base fija y un desplazamiento variable que sumados dan la ubicación en memoria del operando

- **Ejemplo:**

→ `mov R0, [A000+R3] (CISC)`

→ `Load R0, A000(R3) (RISC)`



Indexado

• Características:

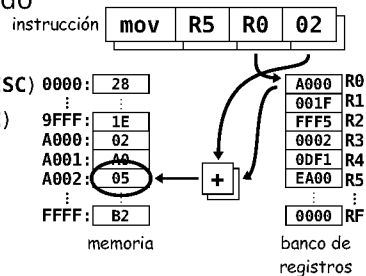
- Relativamente eficiente, puesto que requiere sólo un acceso a memoria (aparte del acceso inicial)
- La dirección de base de la estructura tiene que ser una dirección completa
- El desplazamiento para poder variar se debe almacenar en un registro del procesador
- Está pensado para recorrer múltiples elementos de una única estructura de datos

Base

- Interpretación: el campo argumento contiene un desplazamiento fijo y una dirección de base variable que sumados dan la ubicación en memoria del operando

• Ejemplo:

- `mov R5, [R0 + 02]` (CISC)
- `Load R5, R0(02)` (RISC)



Base

• Características:

- Relativamente eficiente, puesto que requiere sólo un acceso a memoria (aparte del acceso inicial)
- La base para poder variar se debe almacenar en un registro del procesador
- El desplazamiento usualmente está acotado a una cierta cantidad de bits
- Usualmente se codifica este desplazamiento en dos complemento
- Está pensado para acceder a un único elemento en múltiples estructuras de datos

¿Base o indexado?

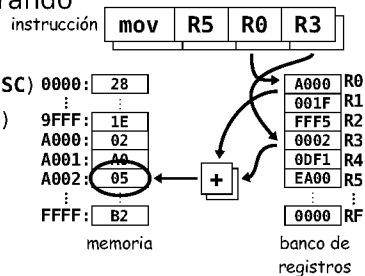
- Un error frecuente (incluso en el final) consiste en confundir los modos base e indexado
- La clave para distinguirlos radica en observar qué codificamos dentro de la instrucción:
 - En el modo indexado codificamos una dirección completa, esto es, un entero positivo que necesita de tantos bits como bits tengan las direcciones de memoria
 - En el modo base codificamos un desplazamiento signado, el cual es práctico incluso en caso de contar con pocos bits

Base-indexado

- Interpretación: el campo argumento contiene una dirección de base y un desplazamiento, ambos variables, que sumados dan la ubicación en memoria del operando

- Ejemplo:

- `mov R5, [R0 + R3]` (CISC)
- `load R5, R0(R3)` (RISC)



Base-indexado

- Características:

- Relativamente eficiente, puesto que requiere sólo un acceso a memoria (aparte del acceso inicial)
- Tanto la base como el desplazamiento para poder variar deben ser almacenados en sendos registros del procesador
- Máxima flexibilidad, permite recorrer múltiples elementos almacenados en múltiples estructuras de datos

Base-indexado indirecto

- La combinación de los modos base-indexado y absoluto indirecto genera dos posibles modalidades de uso:
 - **Pre-indexado indirecto:** el registro base y el registro índice son sumados para obtener la dirección en memoria de la dirección del operando
 - **Post-indexado indirecto:** el registro base contiene la dirección en memoria de un valor el cual sumado al registro índice permiten obtener la dirección del operando

Base-indexado indirecto

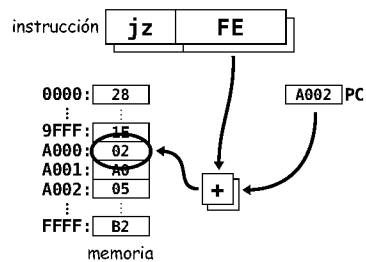
- **Características:**
 - La misma crítica realizada acerca del modo absoluto indirecto se aplica en este nuevo contexto
 - La modalidad pre-indexado indirecto es útil para acceder a tablas de interrupciones
 - La modalidad post-indexado indirecto es útil para acceder a los campos de un registros, ya que por lo general éstos son gestionados a través de un puntero (es decir, la dirección en memoria del comienzo de la estructura)

PC relativo

- **Interpretación:** el campo argumento contiene un desplazamiento fijo el cual debe ser sumado al contenido del registro PC para obtener la dirección referida

- **Ejemplo:**

→ `jz label`



PC relativo

● Características:

- Requiere un único acceso a memoria (aparte del acceso inicial para traer la instrucción)
- El desplazamiento, un entero signado, suele estar representado en dos complemento
- Los saltos condicionales usualmente hacen uso de este modo de direccionamiento para hacer referencia al destino de los mismos

Código independiente de la posición

- Los modos de direccionamiento base y relativo permiten la construcción de código cuyo funcionamiento es independiente de la posición en la cual termine siendo cargado en memoria
- Este tipo de código se denomina justamente independiente de la posición, o bien **PIC** (por su sigla en inglés, Position Independent Code)
- La clave a fin de obtener este tipo de código radica en evitar toda forma de direccionamiento absoluto

Código independiente de la posición

- El advenimiento de modelos avanzado de gestión de memoria hizo que este tipo de código sea relevante solo en ciertos dominios de aplicación muy específicos
 - Por caso, las librerías de vinculación dinámica, que son compartidas por múltiples aplicaciones al mismo tiempo
 - Esta también es una característica deseable para el código creado con el objeto de ser ejecutado en dispositivos embebidos (algo frecuente en el internet de las cosas o **IoT** por su sigla en inglés)

Código independiente de la posición

- El código independiente de la posición se clasifica en dos grandes categorías:
 - **Estático**: cuando el código inicialmente puede ser cargado en cualquier locación de memoria, pero una vez cargado en memoria deja de ser relocable
 - **Dinámico**: cuando el código puede ser cargado en cualquier locación de memoria y de ser necesario también puede ser relocado en todo momento
- En general, para obtener código **PIC** dinámico se debe hacer uso exclusivo de referencias relativas

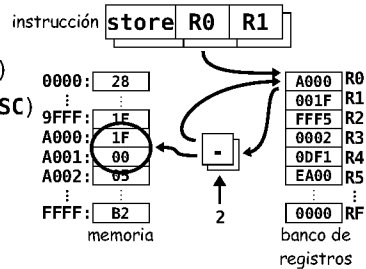
Auto-decrement

- **Interpretación**: el campo argumento contiene una referencia a un registro el cual primero es decrementado y luego es usado como la dirección en memoria del operando

- **Ejemplo**:

→ `mov -(R0), R1 (CISC)`

→ `store R1, -(R0) (RISC)`



Auto-decrement

- **Características**:

→ Relativamente eficiente, puesto que requiere sólo un acceso a memoria (aparte del acceso inicial)

→ Equivale a ejecutar la siguiente secuencia de instrucciones (asumiendo una arquitectura de 16 bits):

`dec R0; dec R0; store R1, (R0)`

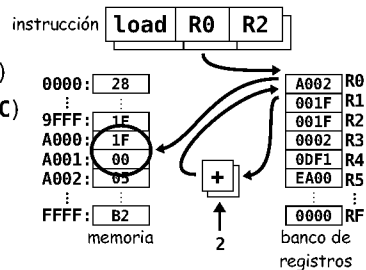
→ Permite gestionar con facilidad una estructura de pila, especialmente a la hora de implementar la operación de apilado (**push**)

Auto-increment

- **Interpretación:** el campo argumento contiene una referencia a un registro es usado como la dirección en memoria del operando y luego es incrementado

- **Ejemplo:**

- `mov R2, (R0)+` (CISC)
- `Load R2, (R0)+` (RISC)



Auto-increment

- **Características:**

- Relativamente eficiente, puesto que requiere sólo un acceso a memoria (aparte del acceso inicial)
- Equivale a ejecutar la siguiente secuencia de instrucciones (nuevamente, asumida una arquitectura de 16 bits):

`Load R2, (R0); inc R0; inc R0`

- Permite gestionar con facilidad una estructura de pila, especialmente a la hora de implementar la operación de desapilado (**pop**)

La arquitectura OCUNS

- La arquitectura **OCUNS** nos servirá de banco de prueba sobre el cual pondremos en práctica las nociones recién introducidas acerca de los modos de direccionamiento
- Es una arquitectura estilo **RISC**
- Sus instrucciones son de tamaño fijo de 16 bits
- El espacio de direcciones es de 256 bytes
- Cuenta con 16 registros de propósito general, si bien el registro **F** se encuentra cableado a cero
- No existe físicamente, es de papel y lápiz



Set de instrucciones

OPCODE	DESCRIPCIÓN	FORMATO	PSEUDOCÓDIGO
0	add (add)	I	$R[d] \leftarrow R[s] + R[t]$
1	sub (subtract)	I	$R[d] \leftarrow R[s] - R[t]$
2	and (and)	I	$R[d] \leftarrow R[s] \& R[t]$
3	xor (xor)	I	$R[d] \leftarrow R[s] \wedge R[t]$
4	lsh (left shift)	I	$R[d] \leftarrow R[s] \ll R[t]$
5	rsh (right shift)	I	$R[d] \leftarrow R[s] \gg R[t]$
6	load (load)	I	$R[d] \leftarrow \text{mem}[\text{offset} + R[s]]$
7	store (store)	I	$\text{mem}[\text{offset} + R[d]] \leftarrow R[s]$
8	lda (load address)	II	$R[d] \leftarrow \text{addr}$
9	jz (jump zero)	II	if $(R[d] == 0)$ $PC \leftarrow PC + \text{addr}$
A	jl (jump greater)	II	if $(R[d] > 0)$ $PC \leftarrow PC + \text{addr}$
B	call (jump and link)	II	$R[d] \leftarrow PC; PC \leftarrow \text{addr}$
C	jmp (jump register)	III	$PC \leftarrow R[d]$
D	inc (increment)	III	$R[d] \leftarrow R[d] + 1$
E	dec (decrement)	III	$R[d] \leftarrow R[d] - 1$
F	hlt (halt)	III	exit

Formato de instrucción

- El formato de instrucción adoptado hace uso de la técnica de expansión del opcode para determinar el tipo y número de los argumentos

FORMATO $\overset{1}{5} \overset{1}{4} \overset{1}{3} \overset{1}{2} \overset{1}{1} 0 9 8 7 6 5 4 3 2 1 0$

I	0 × × ×	dest. d	src. s	src. t/off.
II	1 0 × ×	dest. d	address addr	
III	1 1 × ×	dest. d	-	

- Nótese que en este caso en particular no se está usando esta técnica para ir en busca de bits adicionales para el opcode

Modos de direccionamiento

- La arquitectura **OCUNS** cuenta con un reducido número de modos de direccionamiento:
 - El modo registro directo está disponible en las operaciones aritmético lógicas (formatos **I**, **II** y **III**)
 - El modo registro más desplazamiento está disponible en las instrucciones de transferencia de información desde y hacia memoria (formatos **I** y **II**)
 - El modo absoluto y **PC**-relativo está disponible en las instrucciones de transferencia de control (formato **II**)

Lenguaje máquina

- Consideremos el siguiente fragmento de programa en lenguaje ensamblador:
`add R0, R1, R2`
`sub R3, R4, R5`
`hlt`
- El lenguaje máquina asociado resulta:
`add R0, R1, R2` \Leftrightarrow `0000000000010010` = `0012h`
`sub R3, R4, R5` \Leftrightarrow `0001001101000101` = `1345h`
`hlt` \Leftrightarrow `1111000000000000` = `F000h`

Naturaleza RISC

- La naturaleza **RISC** de la arquitectura **OCUNS** se evidencia por caso en el hecho de que no cuenta con el modo inmediato
- Por caso, ¿de qué manera se puede poner a cero el registro **R0**?
 - `add R0, R0, R0`
 - `sub R0, R0, R0`
 - `xor R0, R0, R0`
 - ...

Otras características

- Esta arquitectura seguirá siendo usada como banco de prueba de los restantes conceptos de bajo nivel que veremos en las siguientes clases:
 - Proceso de ensamblado de un programa
 - Vinculación y carga en memoria
 - Relocación dinámica de código
 - El rol del sistema operativo en la atención de las interrupciones y de los traps
 - Llamada a procedimientos y pasaje de parámetros
 - Programación de bajo nivel de la entrada/salida

¿Preguntas?
