



# Organización de Computadoras 2009

---

## Clase 7



# Temas de Clase

---

- Formatos de instrucción
- Modos de direccionamiento

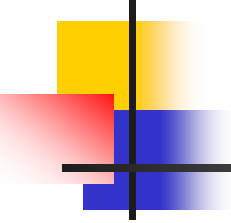


# Elementos de una instrucción de máquina

---

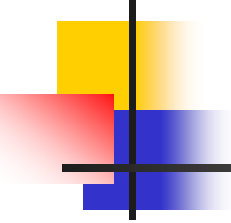
- Código de operación
  - especifica la operación a realizar (ej. suma).
  - es un código binario.
- Referencia del operando fuente
  - Establece dónde se encuentra el operando.
  - la operación puede involucrar uno ó más operando fuente (o de entrada).

# Elementos de una instrucción de máquina (2)



---

- Referencia del operando resultado
  - establece dónde almacenar el resultado
- Referencia de la siguiente instrucción
  - le dice a la CPU donde buscar la siguiente instrucción después de la ejecución de la instrucción anterior.
  - en la mayoría de los casos se ubica a continuación de la instrucción actual.



# Elementos de una instrucción de máquina (3)

---

- ❖ Los operandos fuente y resultado pueden estar en tres lugares :
  - Memoria
  - Registro de la CPU
  - Dispositivo de E/S



# Representación de instrucciones

---

- Dentro de la computadora cada instrucción está representada mediante una secuencia de bits
- La secuencia se divide en campos en correspondencia a los elementos que la componen.
- Este esquema se conoce como *formato de la instrucción*.



# Representación de instrucciones (2)

---

- Es difícil para el programador tratar con las representaciones binarias de las instrucciones de máquina. Por lo tanto, se usa una *representación simbólica*.
- Los códigos de operación se representan por medio de abreviaturas, llamadas *mnemónicos* que indican la operación.



# Representación de instrucciones (3)

---

- ❖ Los ejemplos más comunes son:  
(algunos ya los vimos en el Ingreso)
- ❖ ADD    adición (suma )
- ❖ SUB    sustracción (resta)
- ❖ MOV    movimiento de datos
- ❖ AND, OR, XOR    operaciones lógicas



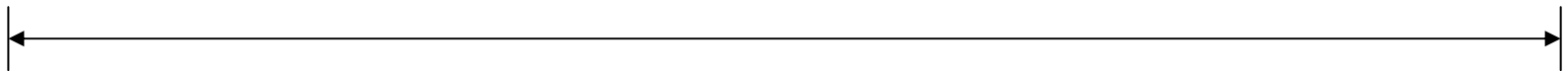
# Representación de instrucciones (4)

- Los operandos también se pueden representar de manera simbólica.

Ej: MOV reg1 , memoY

- instrucción que copia el valor contenido en la posición de memoria llamada **memoY**, a un registro denominado **reg1**.

Código de operación	Referencia al operando	Referencia al operando
---------------------	------------------------	------------------------



M bits



# Tipos de instrucciones

---

- En lenguajes de alto nivel escribimos:

$$X := X + Y$$

- Esta instrucción suma los valores almacenados en las posiciones de memoria X e Y.
- Esto puede implicar cargar registros, sumarlos y luego almacenar el resultado en memoria.



## Tipos de instrucciones (2)

---

- Una instrucción de alto nivel puede requerir varias instrucciones de máquina.
- El lenguaje de alto nivel expresa operaciones en forma “concisa” usando variables.
- El lenguaje de máquina expresa las operaciones en forma “básica” involucrando movimiento de datos y uso de registros.



## Tipos de instrucciones (3)

---

- Cualquier programa escrito en lenguaje de alto nivel se debe convertir a un lenguaje de máquina para ser ejecutado.
- El conjunto de instrucciones de máquina debe ser capaz de expresar cualquiera de las instrucciones de un lenguaje de alto nivel.



# Tipos de instrucciones (4)

---

- Podemos categorizar las instrucciones de máquina como de:
  - ❖ Procesamiento de datos
    - ❖ operaciones aritméticas y lógicas.
  - ❖ Almacenamiento de datos
    - ❖ transferencias dentro del sistema.
  - ❖ Instrucciones de E/S
    - ❖ transferencia de datos entre la computadora y los mecanismos externos.
  - ❖ Control



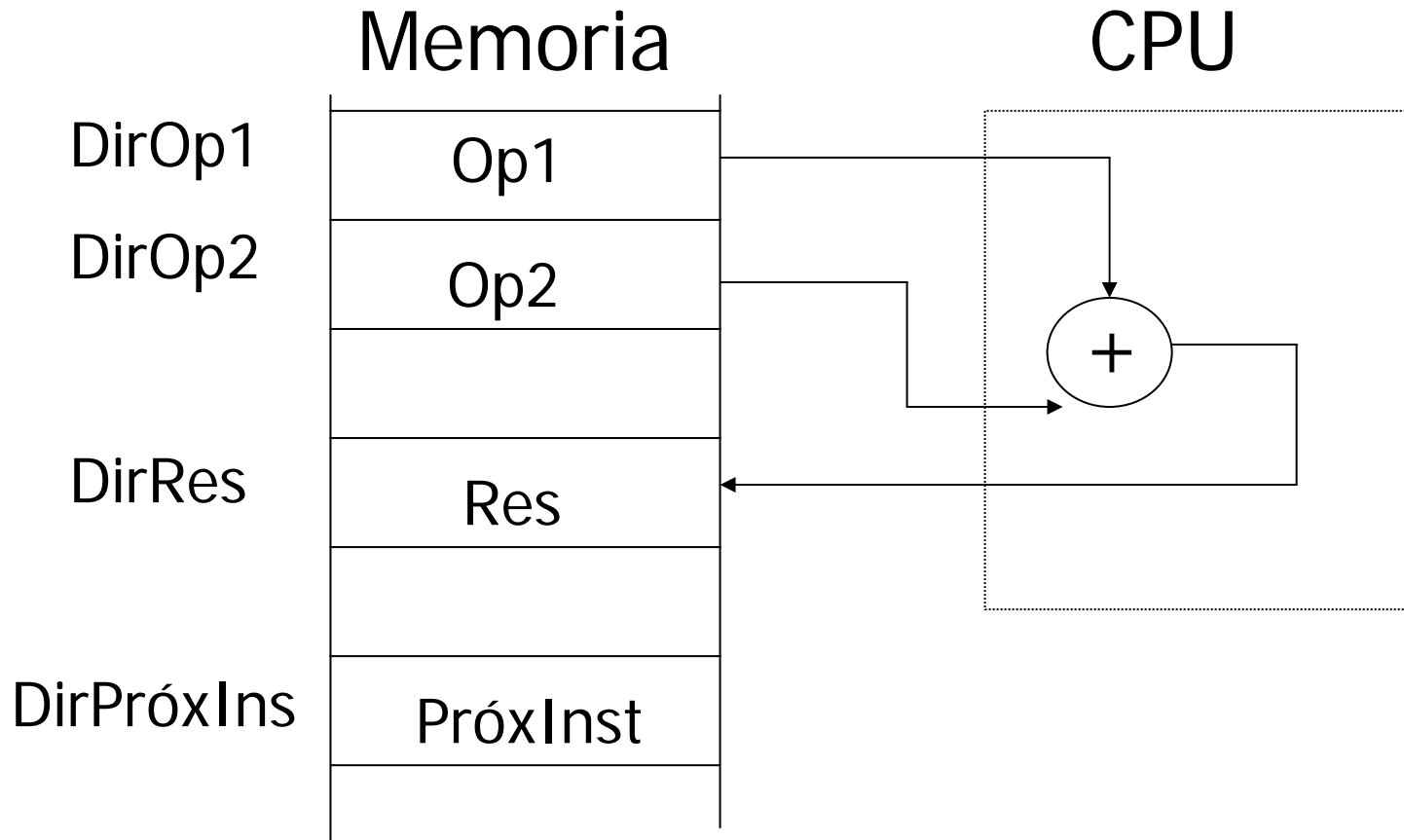
# Número de direcciones

- ¿Cuántas direcciones se necesitan?
- Dos direcciones para hacer referencia a los operandos, una donde almacenar el resultado y la dirección de la próxima instrucción. Por lo tanto necesitaríamos cuatro direcciones

Add DirRes, DirOp1, DirOp2, DirPróxIns

Add	DirRes	DirOp1	DirOp2	DirPróxIns
-----	--------	--------	--------	------------

# Máquina para 4 direcciones





## Máquina para 4 direcciones (2)

---

- ✓ Direcciones explícitas para operandos, resultado y próxima instrucción.
- ✓ Son "raras", cada campo de dirección tiene que tener bits para "acomodar" una dirección completa.
- ✓ Ej. si dirección = 24 bits, la instrucción tiene 96 bits de referencias.





# Máquina para 3 direcciones

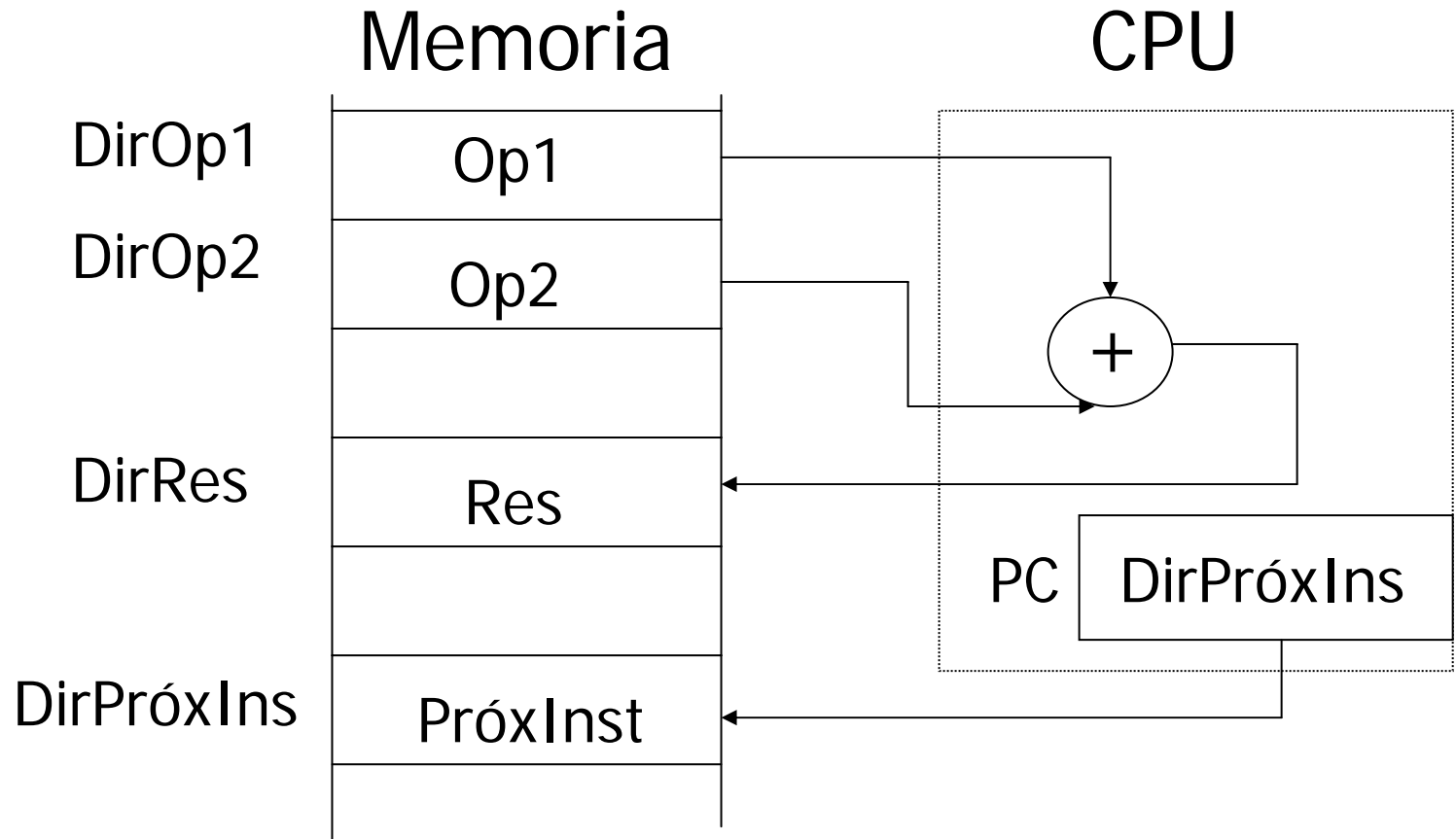
---

Add DirRes, DirOp1, DirOp2

Add	DirRes	DirOp1	DirOp2
-----	--------	--------	--------

- Dirección de la próxima instrucción está almacenada en un registro de la CPU, llamado Contador de Programa PC.
- Referencias = 72 bits. Todavía larga.

# Máquina para 3 direcciones (2)





# Máquina para 2 direcciones

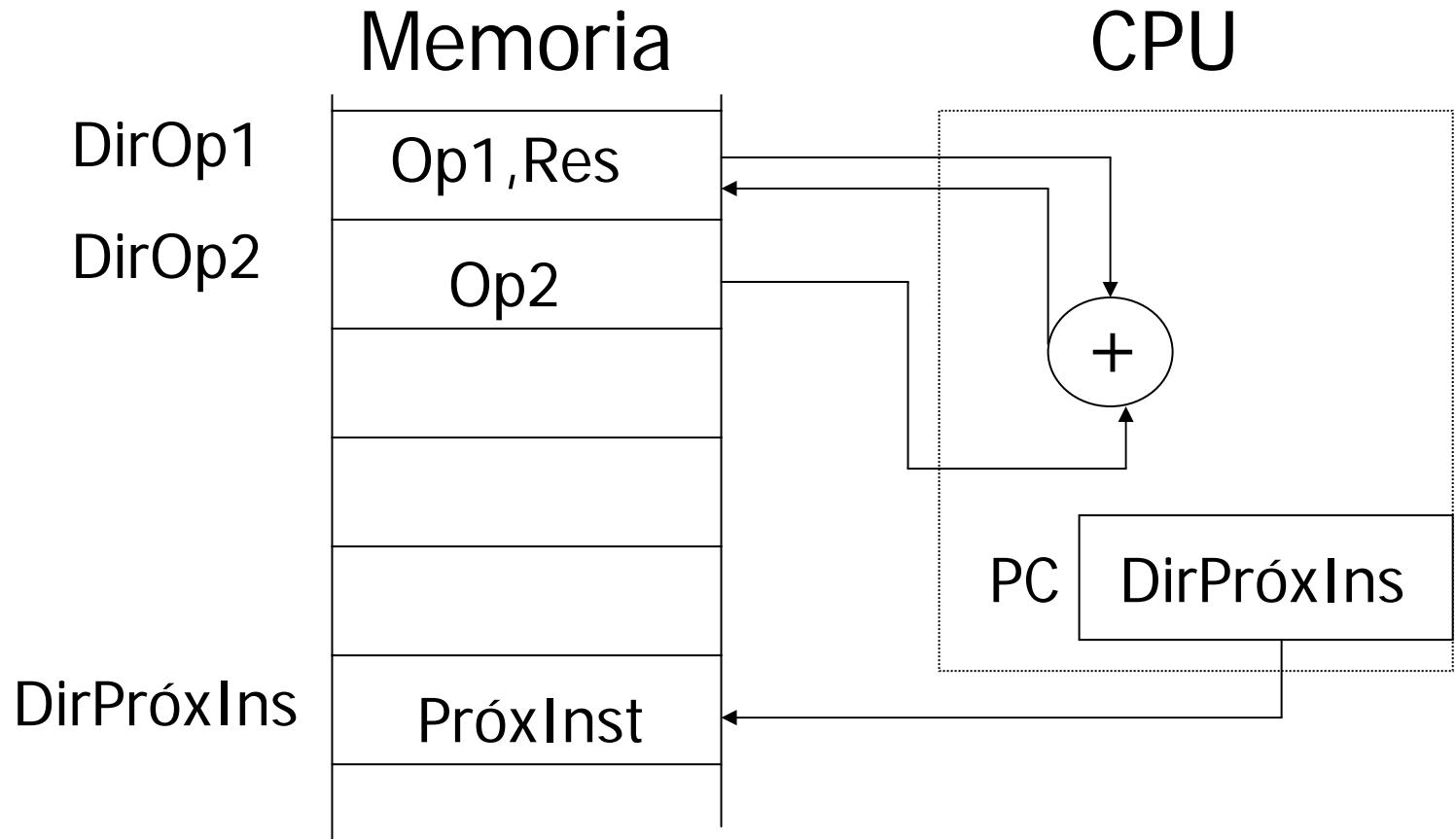
---

Add DirOp1, DirOp2

Add	DirOp1	DirOp2
-----	--------	--------

- Reduce el tamaño de la instrucción.
  - 48 bits de referencias.
- Hay que mover el Op1 a un registro temporal.
- Menos elección donde guardar el resultado.

# Máquina para 2 direcciones (2)





# Máquina para 1 dirección

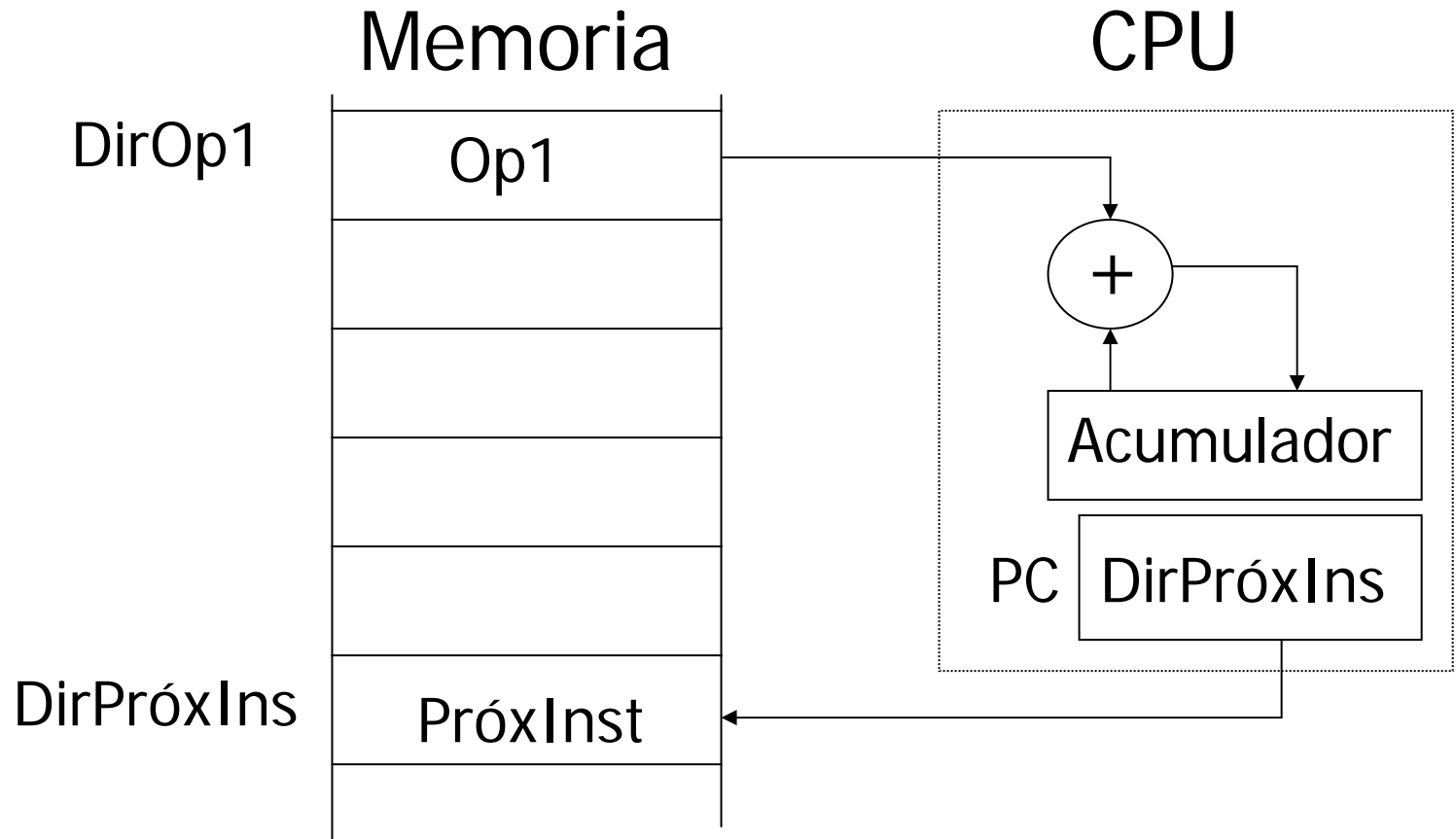
---

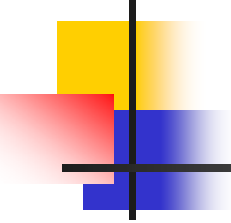
Add DirOp1

Add	DirOp1
-----	--------

- Registros especiales en la CPU (**acumulador**)
- Instrucciones para cargar y descargar el acumulador.
- Un operando y resultado en lugar predefinido
- Instrucción más corta (24 bits de referencias)

# Máquina para 1 dirección (2)





Ej. evaluar  $a = (b + c) * d - e$

---

3 direcciones

add a, b, c  
mul a, a, d  
sub a, a, e

3 instruc./3 acc. MI  
9 acc. MD

2 direcciones

mov a, b  
add a, c  
mul a, d  
sub a, e

4 instruc./4 acc. MI  
12 acc. MD

1 dirección

load b  
add c  
mul d  
sub e  
store a

5 instruc./5 acc. MI  
5 acc. MD



# Diseño del conjunto de instrucciones (1)

---

- ✓ El conjunto de instrucciones es el medio que tiene el programador para controlar la CPU.
- ✓ Hay que tener en cuenta:
  - ✓ Tipos de operaciones
    - ✓ cuántas y cuáles
  - ✓ Tipos de datos
    - ✓ cuáles





# Diseño del conjunto de instrucciones (2)

---

- ✓ Formato de instrucciones
  - ✓ longitud (bits), N° de direcciones, tamaño de cada campo
- ✓ Registros
  - ✓ cantidad que se pueden referenciar mediante instrucciones y su uso
- ✓ Direccionamiento
  - ✓ la manera de especificar la dirección de un operando o una instrucción (la próxima).



# Tipos de operaciones

---

- ✓ Transferencia de datos: Mov (load/store)
- ✓ Aritméticas: Add, Sub, Inc, Dec, Mul
- ✓ Lógicas: And, Or, Xor, Not
- ✓ Conversión
- ✓ E/S: In, Out
- ✓ Transferencia de control: salto, bifurcación
- ✓ Control del sistema: usadas por S.O.



# Tipos de datos

---

Los más importantes:

- Direcciones
- Números: enteros, p. fijo, p. flotante
- Caracteres: ASCII, BCD.
- Datos lógicos



# Modos de direccionamiento

---

- ✓ Como vimos, en una instrucción se utilizan bits para expresar el **código de operación**: nos dice **qué** hacer. También se necesitan una "gran" cantidad de bits para especificar de dónde provienen los datos.
- ✓ ¿Cómo podemos reducir el tamaño de estas especificaciones?



## Modos de direccionamiento (2)

---

Hay 2 métodos generales:

1. Si un operando va a usarse varias veces puede colocarse en un registro.

- ❖ Usar registro para una variable tiene 2 ventajas
  - ❖ el acceso es más rápido
  - ❖ se necesitan menos bits.

Ej. si hay 32 reg. se necesitan 5 bits para especificar c/u de ellos (menos bits que las dir. de mem.).



## Modos de direccionamiento (3)

---

### 2. Especificar uno ó más operandos en forma implícita.

Ejemplos:  $\text{reg2} = \text{reg2} + \text{fuente1}$  ; el acumulador.

Los mdd tienen como objetivo:

- ✓ disminuir la cantidad de bits en la instrucción
- ✓ la dirección puede que no se conozca hasta el momento de ejecutar el programa
- ✓ manejo más eficiente de datos (arreglos)



# Modos de direccionamiento (4)

---

- Inmediato
- Directo
- Por registro
- Indirecto por memoria
- Indirecto por registro
- Por desplazamiento
- Del stack



# Mdd Inmediato

---



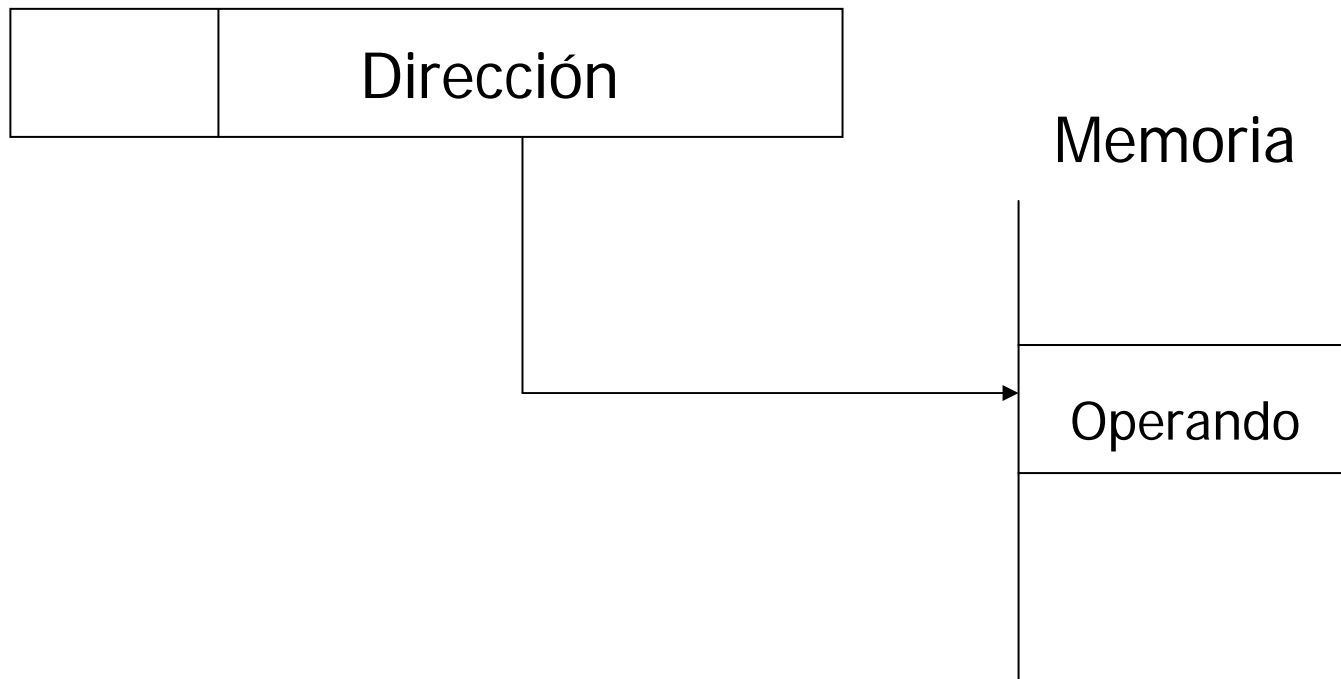
- El operando se obtiene automáticamente de la memoria al mismo tiempo que la instrucción.
- No requiere una referencia extra a memoria de datos
- Se utiliza para definir constantes y para inicializar variables.
- Desventaja: tamaño del operando limitado por el tamaño del campo de direccionamiento.





# Mdd Directo

---





## Directo (2)

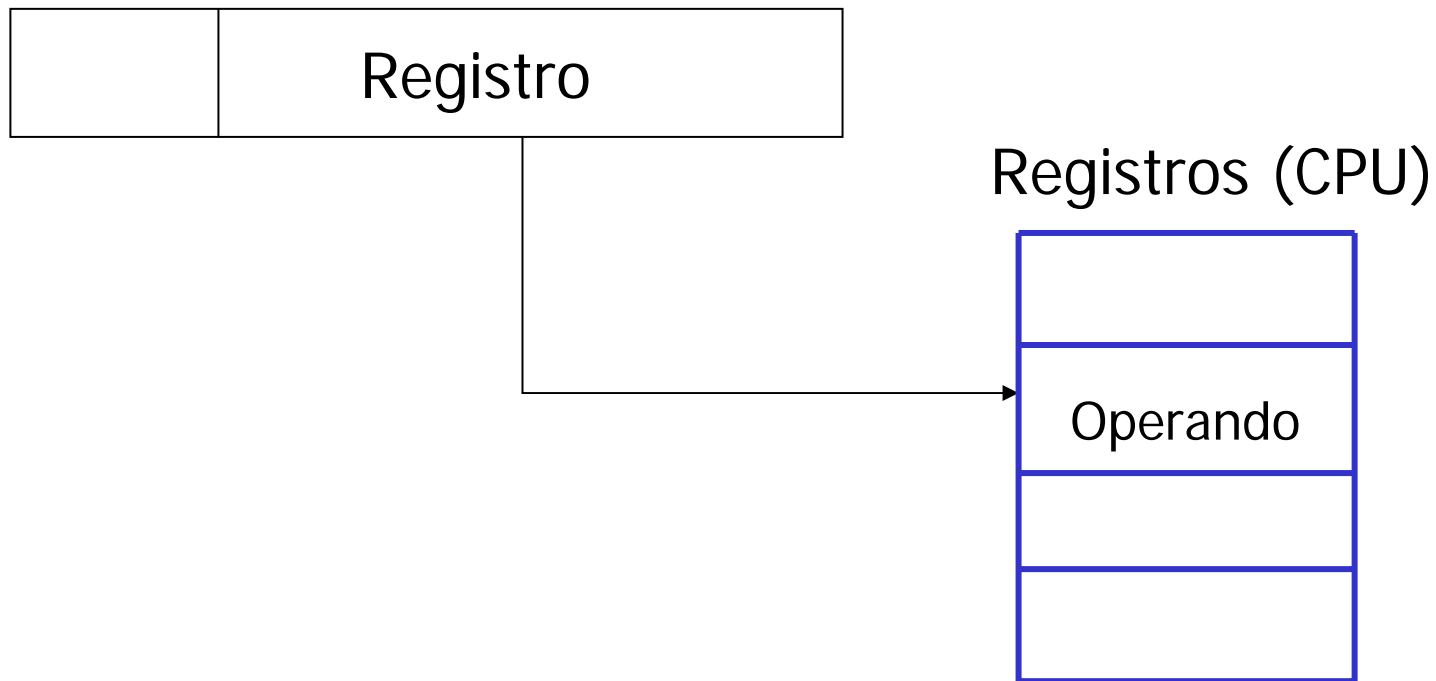
---

- El campo de dirección tiene la dirección efectiva del operando.
- Es simple, pero tiene un espacio limitado de direcciones por cantidad de bits del campo.
- Uso: acceder a variables globales, cuya dirección se conoce en el momento de compilación.



# Mdd Por registro

---



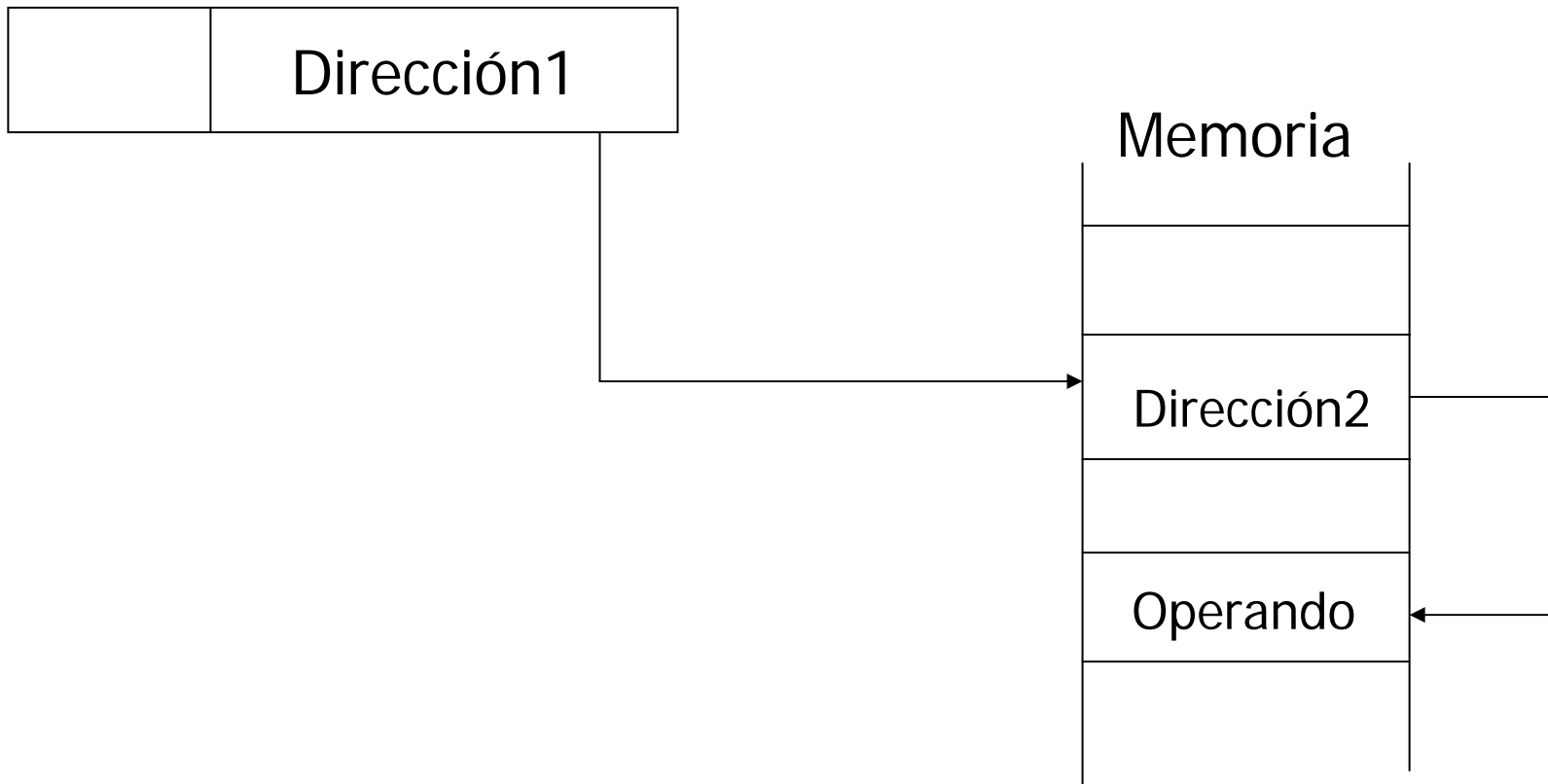


## Por registro (2)

---

- Conceptualmente igual al directo, pero se especifica un **registro** en lugar de una posición de memoria.
- La referencia a registro usa menos bits que la especificación de la dirección y no requiere acceso a memoria de datos.
- Desventaja: los registros no son muchos y es un recurso preciado.

# Mdd Indirecto por memoria



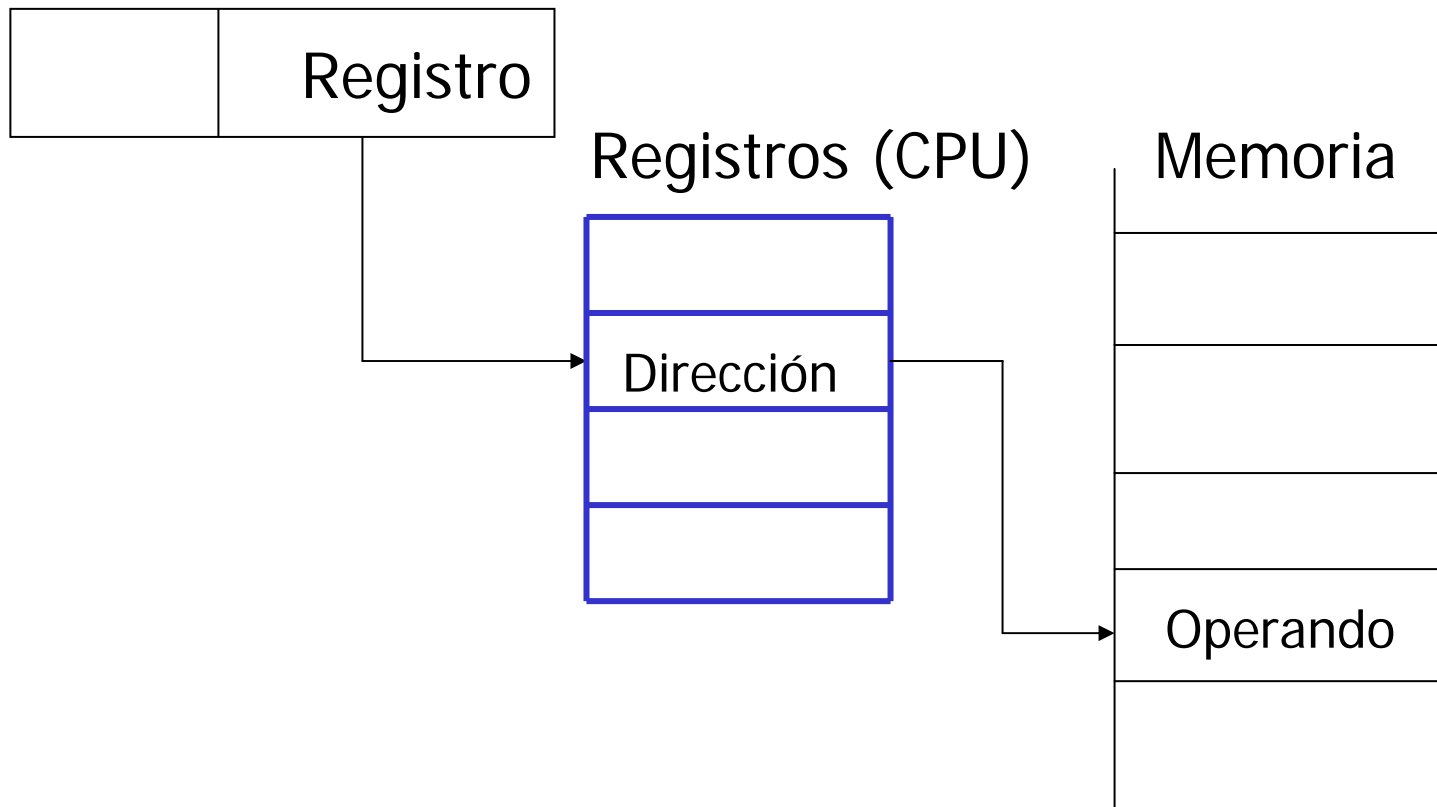


## Indirecto por memoria (2)

---

- En la instrucción está la dirección de la dirección del operando. Trata de solucionar el problema del directo. Así, con una dirección de menos bits en la instrucción, se apunta a una dirección de más bits.
- Ventaja: espacio de direccionamiento mayor
- Desventaja: múltiples accesos a memoria.

# Mdd Indirecto por registro





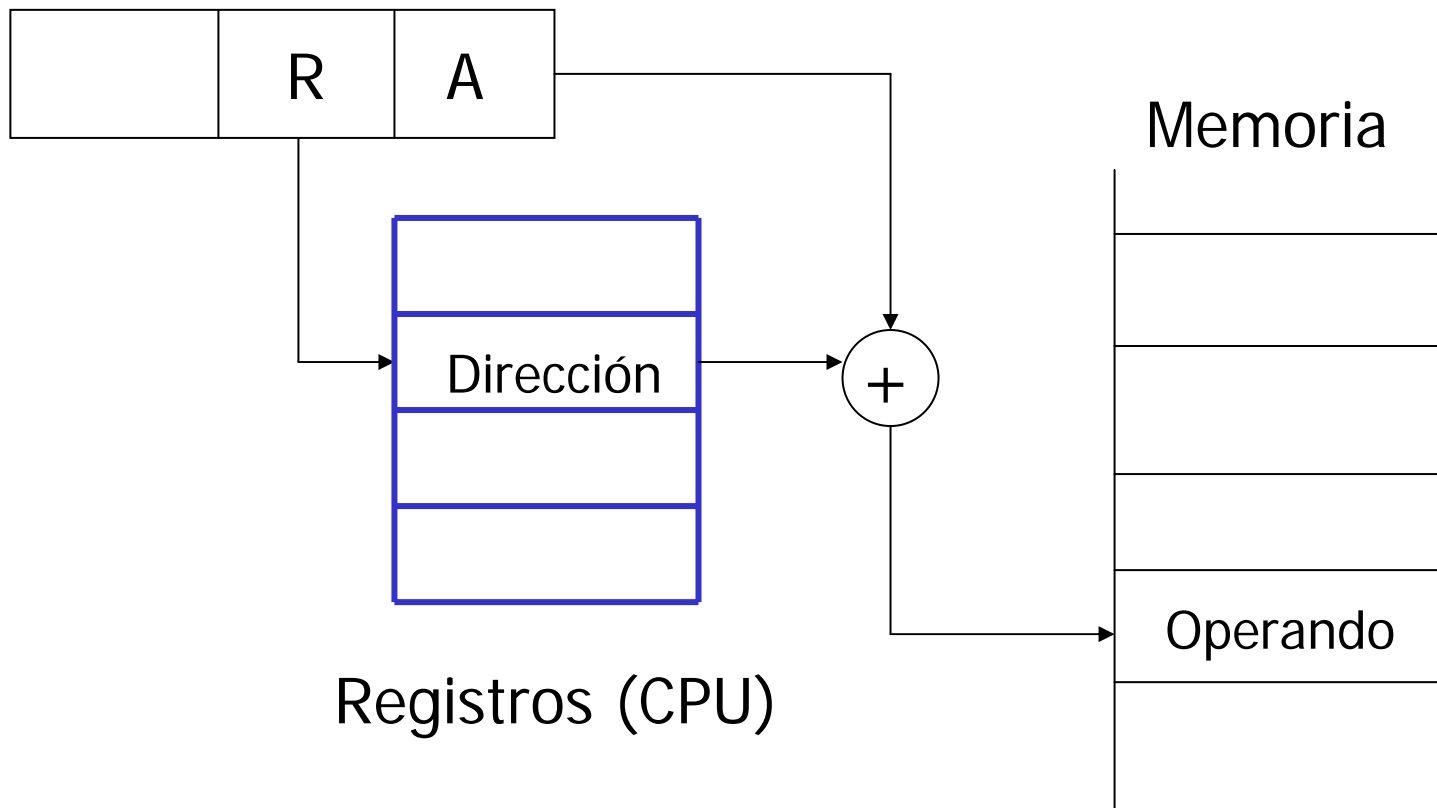
## Indirecto por registro (2)

---

- En la instrucción se especifica el registro que tiene almacenada la dirección.
- Ventaja: menos bits para especificar el registro que la posición de memoria. Espacio de direccionamiento grande, accede una vez menos a memoria que el indirecto. La dirección así usada se llama apuntador.



# Mdd Por desplazamiento





## Por desplazamiento (2)

---

- Combina capacidades de indirecto y directo. Requiere que la instrucción tenga dos campos de dirección. Estos dos campos se suman para producir la dirección efectiva. Los más comunes:
  - Relativo
  - De registro base
  - Indexado



# Relativo

---

- El registro referenciado de manera implícita es el contador de programa PC.
- La dirección de la instrucción actual se suma al campo de dirección para producir la dirección efectiva.
  - El campo de dirección se trata como un número en  $\text{Ca2}$ .



# De registro base

---

- El registro referenciado contiene una dirección de memoria y el campo de dirección tiene un desplazamiento.



# Indexado

---

- Se direcciona la memoria con un registro más un desplazamiento.
  - Es “igual” al anterior pero se intercambian los papeles del registro y del desplazamiento.
- La Indexación proporciona un mecanismo eficiente para realizar operaciones iterativas.
- Se utiliza un registro llamado **índice**
  - algunas máquinas incrementan ó decrementan este registro como parte de la instrucción (Autoindexación)



# Del stack

---

- El stack ó pila es un arreglo lineal de localidades de memoria. Es una lista ó cola donde el último en entrar es el primero en salir. Es una zona de memoria reservada.
- Asociado con la pila o stack hay un registro apuntador (o registro puntero de pila), cuyo valor es la dirección tope de pila o stack.

# MSX88: inst. de transferencia

1	MOV <i>dest,fuente</i>	Copia <i>fuentes</i> en <i>dest</i>	$(dest) \leftarrow (fuente)$
2	PUSH <i>fuentes</i>	Carga <i>fuentes</i> en el tope de la pila	$(SP) \leftarrow (SP) - 2; [SP+1:SP] \leftarrow (fuente)$
2	POP <i>dest</i>	Desapila el tope de la pila y lo carga en <i>dest</i>	$(fuente) \leftarrow [SP+1:SP]; (SP) \leftarrow (SP) + 2$
2	PUSHF	Apila los flags	$(SP) \leftarrow (SP) - 2; [SP+1:SP] \leftarrow (flags)$
2	POPF	Desapila los flags	$(flags) \leftarrow [SP+1:SP]; (SP) \leftarrow (SP) + 2$
3	IN <i>dest,fuente</i>	Carga el valor en el puerto <i>fuentes</i> en <i>dest</i>	$(dest) \leftarrow (fuente)$
4	OUT <i>dest,fuente</i>	Carga en el puerto <i>dest</i> el valor en <i>fuentes</i>	$(dest) \leftarrow (fuente)$

1. *dest/fuentes* son: *reg/reg*, *reg/mem*, *reg/op.inm*, *mem/reg*, *mem/op.inm*.

*mem* puede ser una etiqueta (dir.directo) o [BX] (dir.indirecto).

2. *dest* y *fuentes* solo pueden ser registros de 16 bits.

3. *dest/fuentes* son: *AL/mem*, *AX/mem*, *AL/DX*, *AX/DX*.

4. *dest/fuentes* son: *mem/AL*, *mem/AX*, *DX/AL*, *DX/AX*.

*mem* debe ser dirección entre 0 y 255. Puede ser un operando inmediato o una etiqueta.

# Inst. aritméticas y lógicas

1	ADD <i>dest,fuente</i>	Suma <i>fuentes</i> y <i>dest</i>	$(dest) \leftarrow (dest) + (fuente)$
1	ADC <i>dest,fuente</i>	Suma <i>fuentes</i> , <i>dest</i> y flag <i>C</i>	$(dest) \leftarrow (dest) + (fuente) + C$
1	SUB <i>dest,fuente</i>	Resta <i>fuentes</i> a <i>dest</i>	$(dest) \leftarrow (dest) - (fuente)$
1	SBB <i>dest,fuente</i>	Resta <i>fuentes</i> y flag <i>C</i> a <i>dest</i>	$(dest) \leftarrow (dest) - (fuente) - C$
1	CMP <i>dest,fuente</i>	Compara <i>fuentes</i> con <i>dest</i>	$(dest) - (fuente)$
5	NEG <i>dest</i>	Negativo de <i>dest</i>	$(dest) \leftarrow \text{CA2}(dest)$
5	INC <i>dest</i>	Incrementa <i>dest</i>	$(dest) \leftarrow (dest) + 1$
5	DEC <i>dest</i>	Decrementa <i>dest</i>	$(dest) \leftarrow (dest) - 1$
1	AND <i>dest,fuente</i>	Operación <i>fuentes</i> AND <i>dest</i> bit a bit	$(dest) \leftarrow (dest) \text{ AND } (fuente)$
1	OR <i>dest,fuente</i>	Operación <i>fuentes</i> OR <i>dest</i> bit a bit	$(dest) \leftarrow (dest) \text{ OR } (fuente)$
1	XOR <i>dest,fuente</i>	Operación <i>fuentes</i> XOR <i>dest</i> bit a bit	$(dest) \leftarrow (dest) \text{ XOR } (fuente)$
5	NOT <i>dest</i>	Complemento a 1 de <i>dest</i>	$(dest) \leftarrow \text{CA1}(dest)$

1. *dest/fuente* son: *reg/reg*, *reg/mem*, *reg/op.inm*, *mem/reg*, *mem/op.inm*.

5. *dest* solo puede ser *mem* o *reg*.

*mem* puede ser una etiqueta (*dir.directo*) o *[BX]*, siendo (*BX*) una dirección de memoria (*dir.indirecto*).





# Inst. transf. de control

6	CALL <i>etiqueta</i>	Llama a subrutina cuyo inicio es <i>etiqueta</i>	
6	RET	Retorna de la subrutina	
6	JZ <i>etiqueta</i>	Salta si el último valor calculado es cero	Si $Z=1$ , $(IP) \leftarrow mem$
6	JNZ <i>etiqueta</i>	Salta si el último valor calculado no es cero	Si $Z=0$ , $(IP) \leftarrow mem$
6	JS <i>etiqueta</i>	Salta si el último valor calculado es negativo	Si $S=1$ , $(IP) \leftarrow mem$
6	JNS <i>etiqueta</i>	Salta si el último valor calculado no es negativo	Si $S=0$ , $(IP) \leftarrow mem$
6	JC <i>etiqueta</i>	Salta si el último valor calculado produjo carry	Si $C=1$ , $(IP) \leftarrow mem$
6	JNC <i>etiqueta</i>	Salta si el último valor calculado no produjo carry	Si $Z=1$ , $(IP) \leftarrow mem$
6	JO <i>etiqueta</i>	Salta si el último valor calculado produjo overflow	Si $O=1$ , $(IP) \leftarrow mem$
6	JNO <i>etiqueta</i>	Salta si el último valor calculado no produjo overflow	Si $O=0$ , $(IP) \leftarrow mem$
6	JMP <i>etiqueta</i>	Salto incondicional a <i>etiqueta</i>	$(IP) \leftarrow mem$

6. *mem* es la dirección de memoria llamada *etiqueta*.



# mas información ...

---

## Repertorios de instrucciones

- Capítulo 9: características y funciones
- Capítulo 10: modos de direccionamiento y formatos
- Apéndice 9A: Pilas
  - Stallings. 5ta Ed.
- Lenguaje Assembly
  - Apunte 4 de cátedra
- Simulador MSX88
  - Descargas en página web de cátedra