

Arquitectura

Parte 1/2

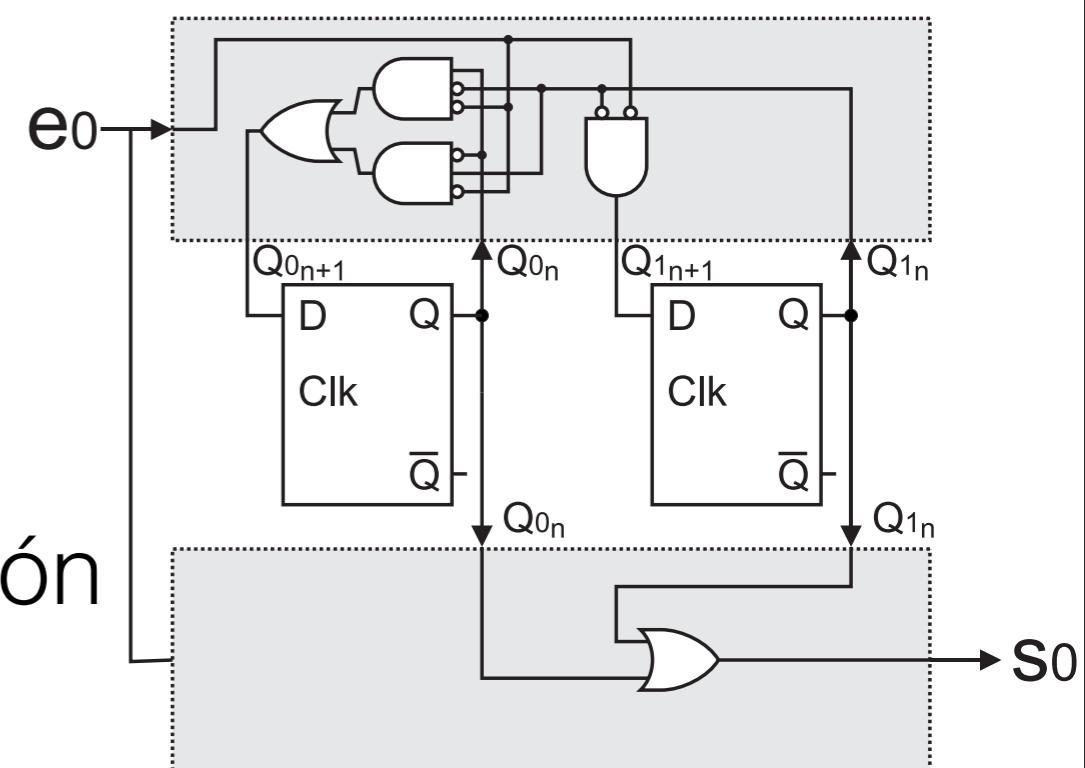
Organización del Computador 1
1er Cuatrimestre 2018

COMPUTER LOVE

Agenda

¿De dónde venimos?

- Introducción: esquema de una computadora
- Representación de la información
- Circuitos Combinatorios
- Circuitos Secuenciales



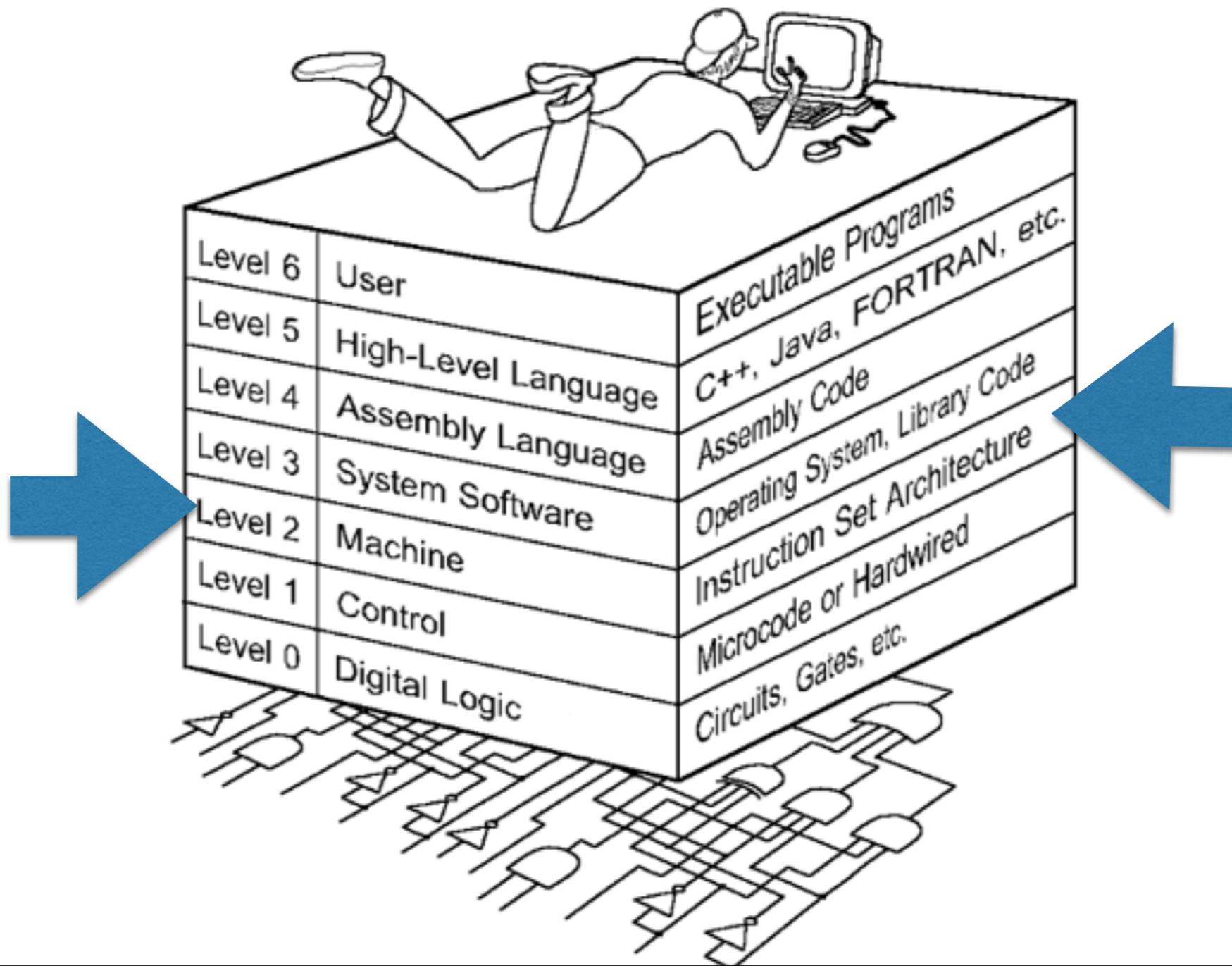
Agenda

¿A dónde vamos?

- Ciclo de Instrucción
- Arquitectura del Computador
 - Memoria/Registros
 - Instrucciones
- Lenguaje Ensamblador



Niveles de Abstracción de una Computadora



Modelo de Von Neumann

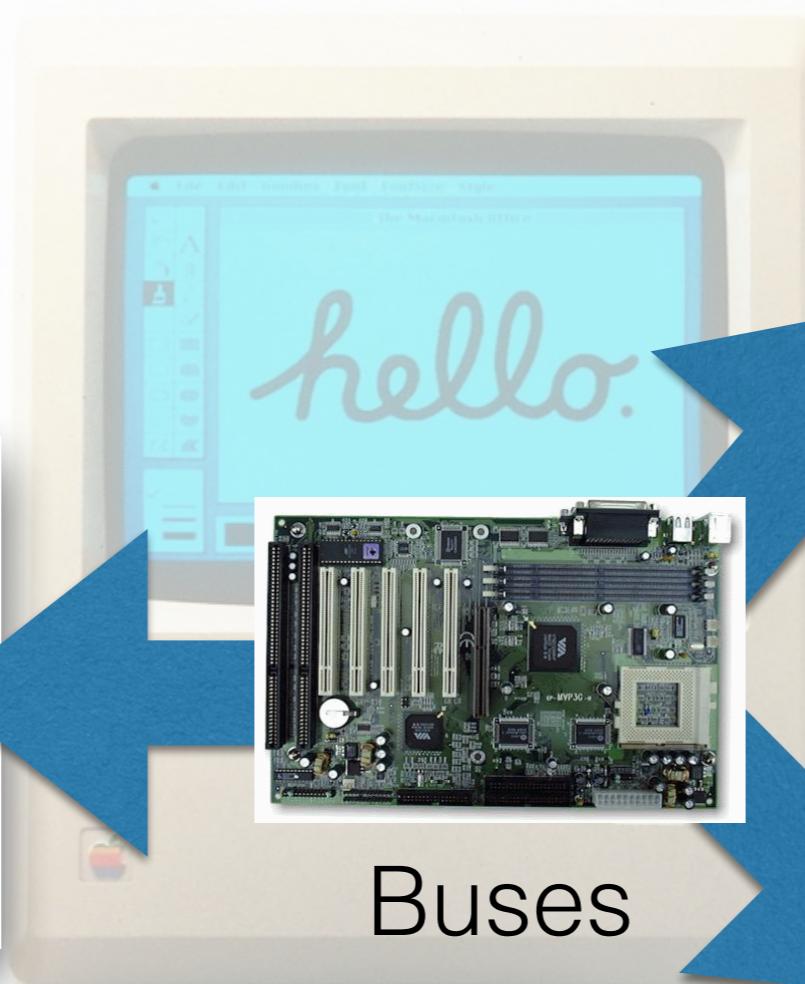
- John Von Neumann
(1903-1957)
- *Primer Informe sobre el EDVAC* (1945)
- Programas son **almacenados** como datos en **memoria**
- ¿Qué es un **dato**? ¿Qué es un **programa**?



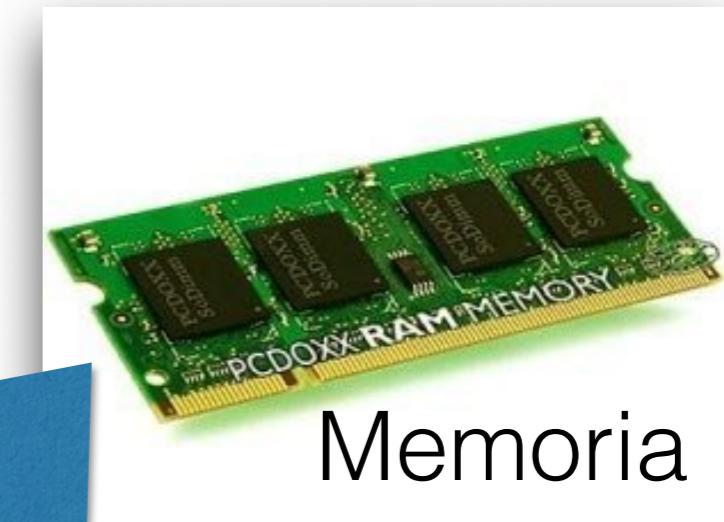




CPU



Buses



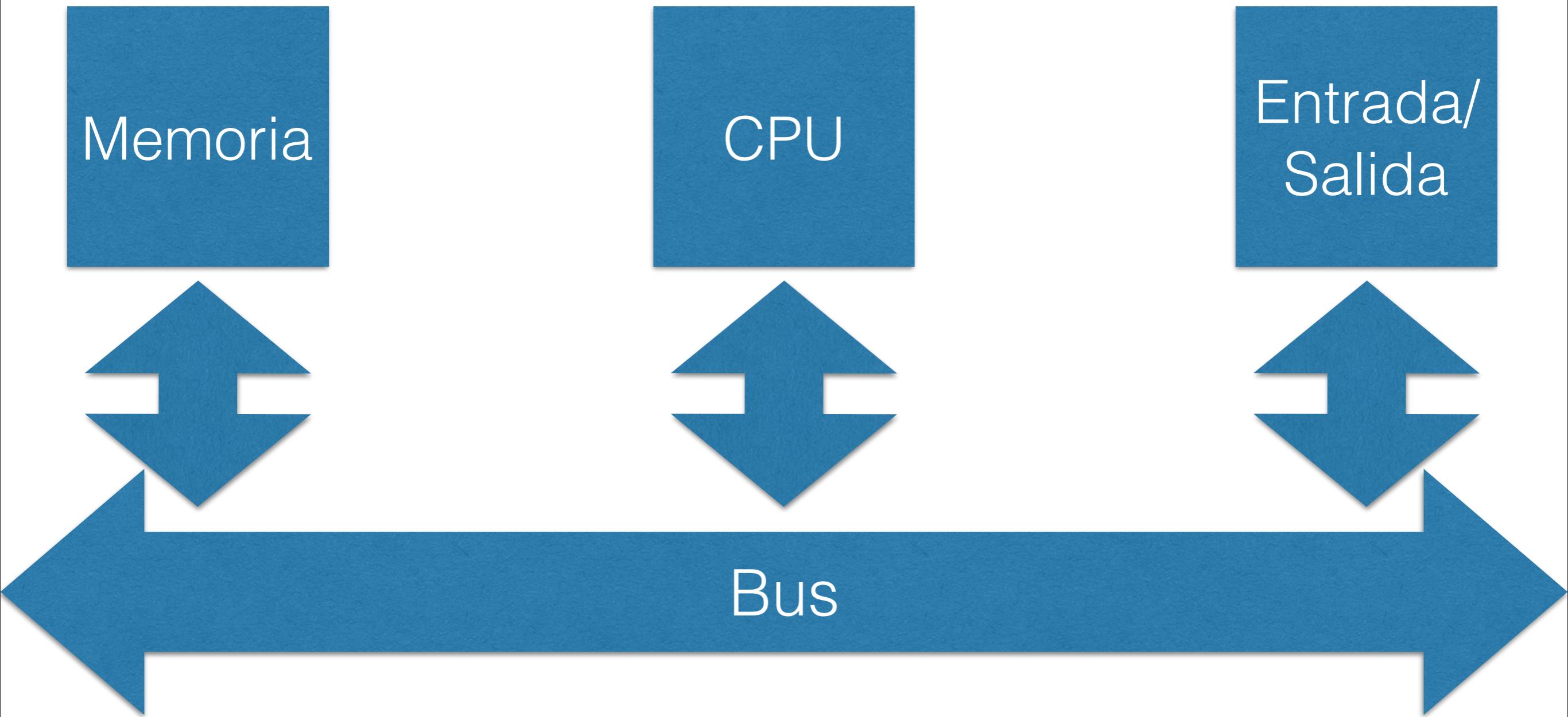
Memoria

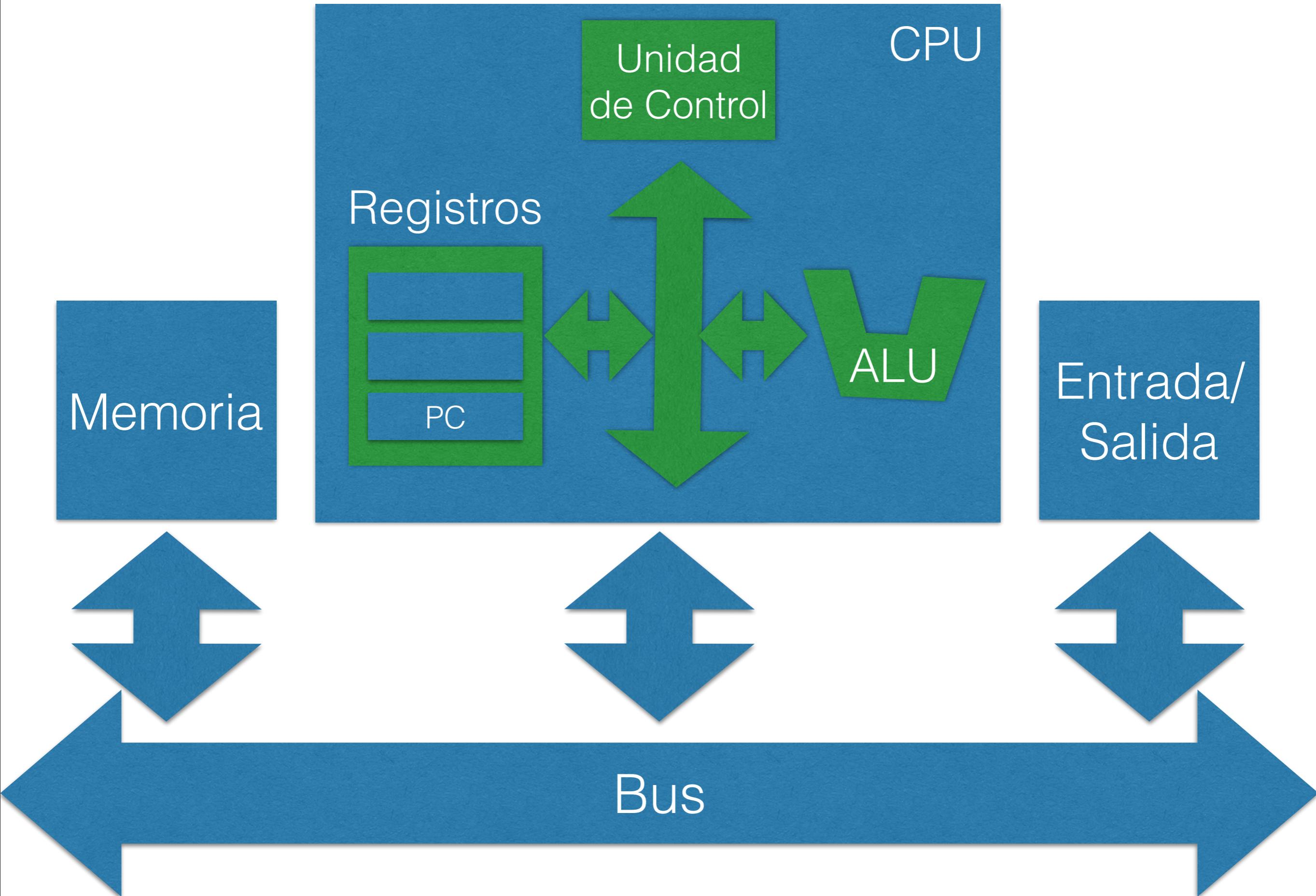
Entrada-Salida



Modelo Von Neumann

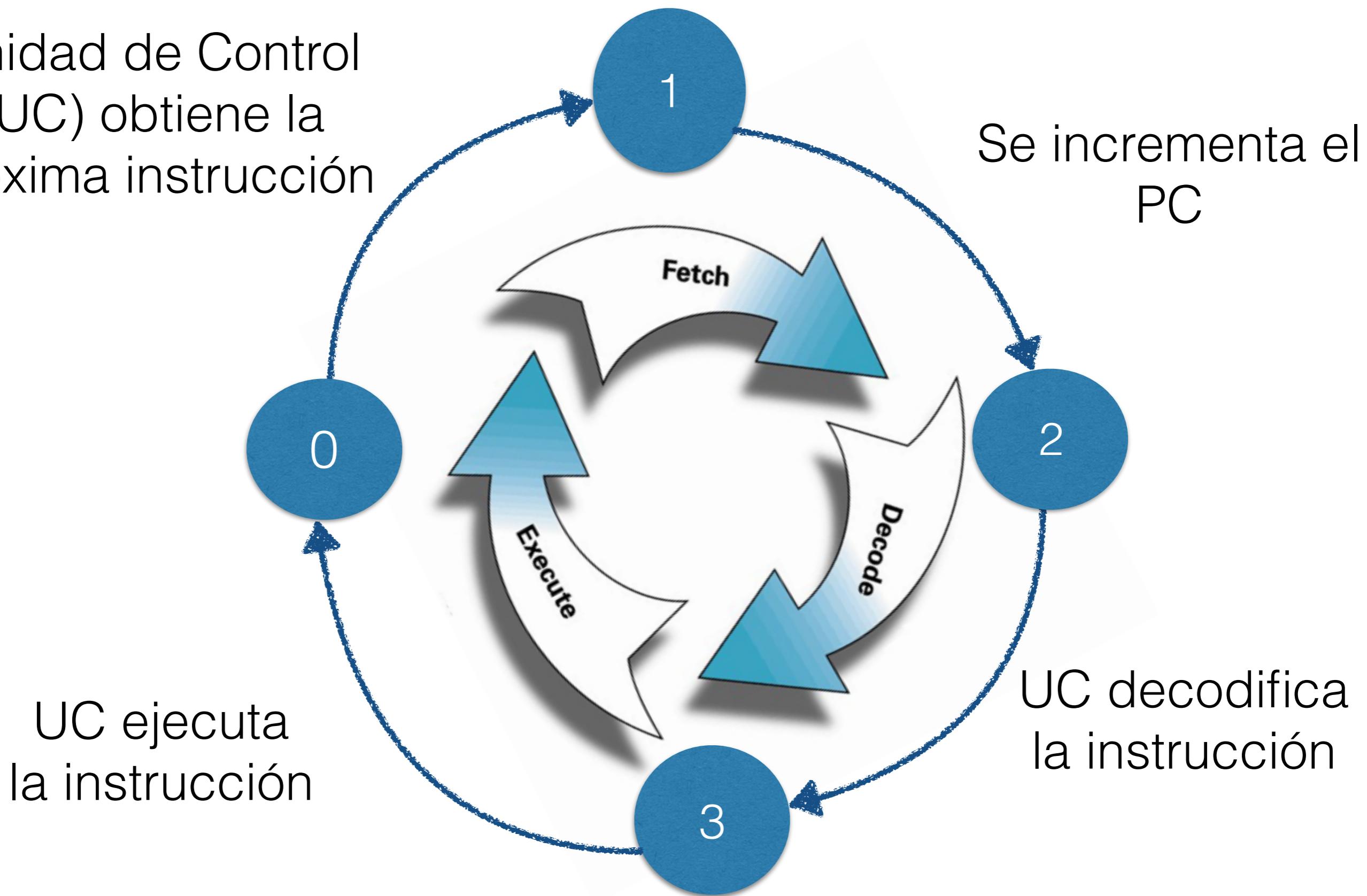
Organización





Ciclo de Instrucción

Unidad de Control
(UC) obtiene la
próxima instrucción



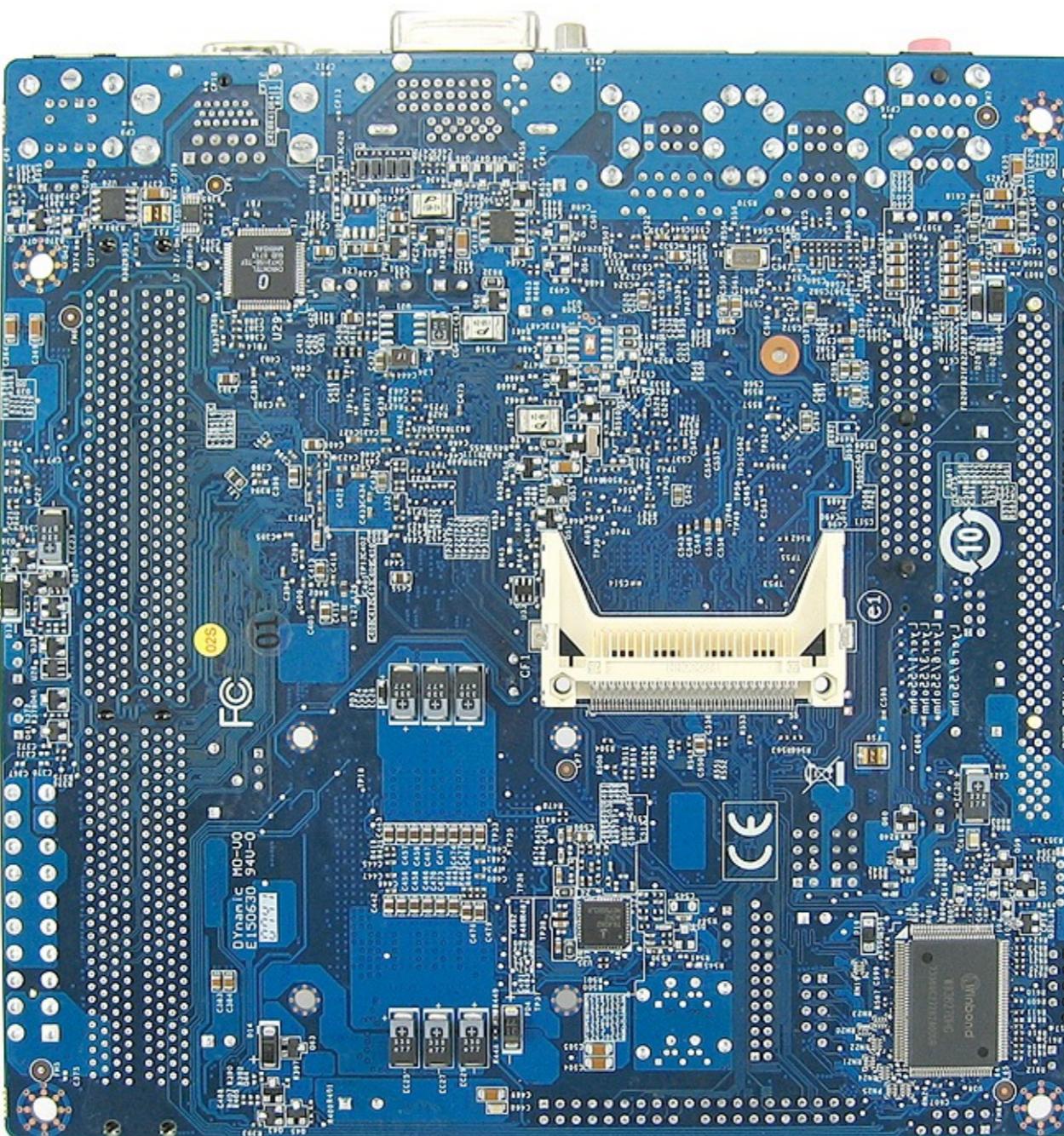
ISA: Instruction Set Architecture

- ¿Qué preguntas debe responder?
 - ¿Qué tipos de dato puedo manejar nativamente?
 - ¿Cómo se almacenan?
 - ¿Cómo se acceden? ¿Tamaño de palabra?
- ¿Qué operaciones (instrucciones) puedo ejecutar?
 - ¿Cómo se codifican?

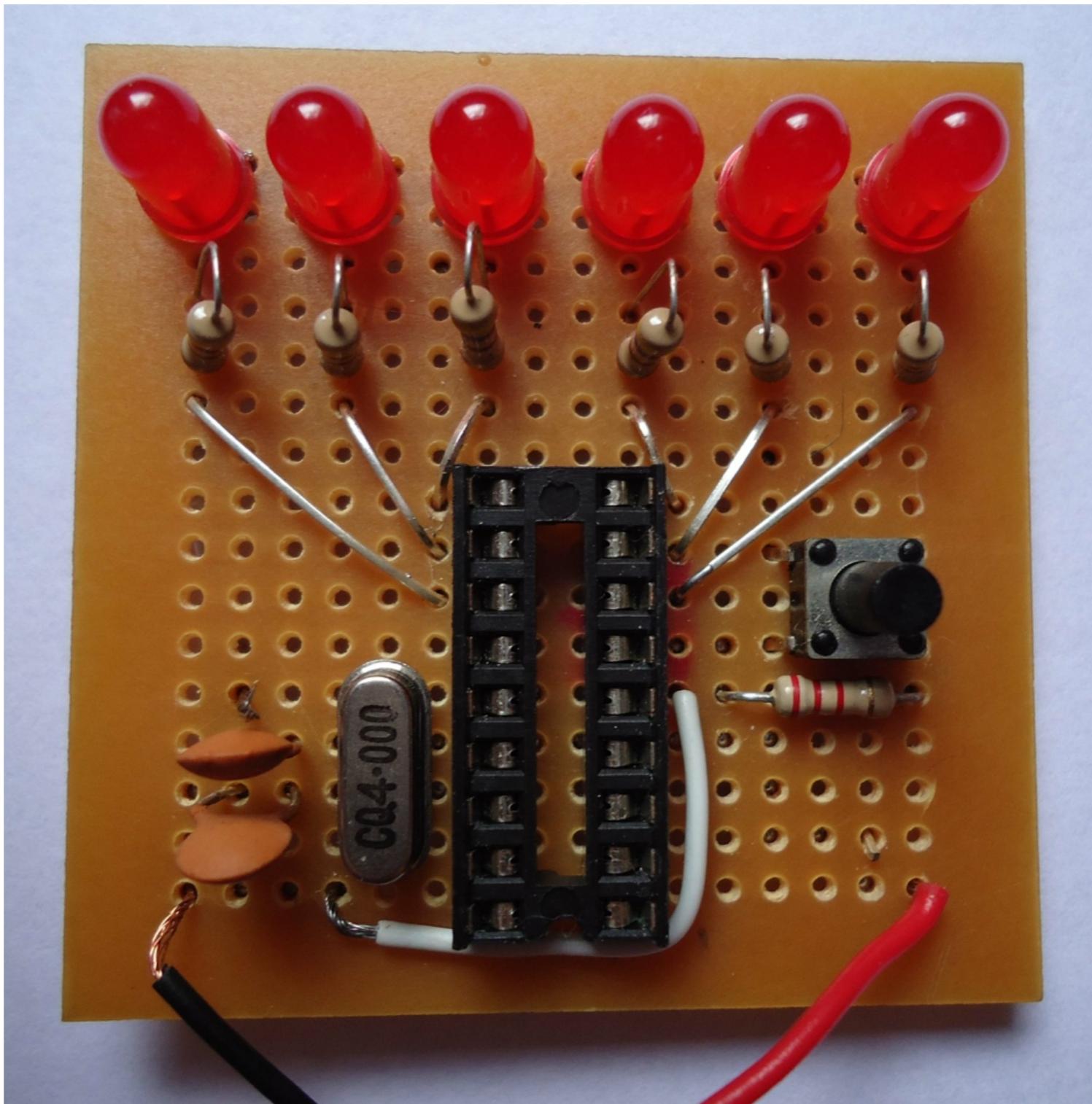
Un Instruction Set Real



Una Arquitectura Real



Otra Arquitectura Real



Arquitectura de una Computadora

- ¿Qué tipos de datos puede manejar “nativamente”?
 - ¿Cómo se almacenan?
 - ¿Cómo se acceden?
- ¿Qué operaciones (instrucciones) puede ejecutar?
 - ¿Cómo se codifican estas operaciones?

Arquitectura de una Computadora

- Un conjunto de respuestas a las preguntas anteriores se “llama” **Instruction Set Architecture** (~ISA)
- Cuando estamos estudiando una ISA, la organización se vuelve un detalle que sólo nos interesa para entender el comportamiento

Métricas de una ISA

- Cantidad total de instrucciones disponibles (“largo”).
 - Complejidad del conjunto de instrucciones:
 - RISC: **R**educed **I**nstruction **S**et **C**omputer
 - CISC: **C**omplex **I**nstruction **S**et **C**omputer
 - Longitud de las instrucciones (“ancho”)
 - Cantidad de memoria que un programa requiere (“largo x ancho”)

¿Qué tipos de datos soporta?

- Tipos de datos
 - Enteros (8, 16, 32 bits; complemento a 2)
 - Números Racionales - Punto fijo
 - Números Racionales - Punto flotante
 - Binary Coded Decimal (BCD), American Standard Code for Information Interchange (ASCII)
- Almacenamiento
 - Big Endian
 - Little Endian

Acceso a los datos

- ¿Dónde se almacenan los datos?
 - ¿Tiene Registros?
 - ¿Cuánta Memoria tiene?
 - ¿Tiene Stack?
 - ¿Tiene Espacio de E/S diferenciado o compartido?
- ¿Cómo se acceden a los datos?
 - Modos de direccionamiento válidos de las instrucciones
 - Directo (memoria), indirecto (puntero en memoria), indexado a registro

Operaciones

- ¿Qué operaciones (instrucciones) puede ejecutar?
 - Movimiento de datos (ej: MOVE, LOAD, STORE...)
 - Aritméticas (ej: ADD, SUB, ...)
 - Lógicas (ej: AND, XOR, ...)
 - E/S (ej: IN, OUT)
 - Transferencia de control (ej: JUMP, CALL, RET)
 - Específicas (ej: MMX)

¿Cómo se codifican las operaciones?



Longitud de Instrucción

- **CodOp**: representa la operación a realizar.
- **Fuente/s**: provee información suficiente para obtener operandos de origen.
- **Destino/s**: provee información suficiente para obtener la ubicación del resultado de la operación.
- **Ref. próx. instr.**: provee información suficiente para determinar el próximo valor del PC.

Arquitectura "Orga1"

- Es una "simplificación" de una arquitectura Intel x86
- Existe un **simulador** de su ejecución
- Existe un **ensamblador** para sus programas
- Es una arquitectura de “*propósito general*”



Registros



- 8 registros de **propósito general** de 16 bits (R0 a R7)
- 5 registros de **propósito específico** de 16 bits
 - PC (Program Counter)
 - SP (Stack Pointer)
 - IR0, IR1, IR2 (Instruction Register)

Flags

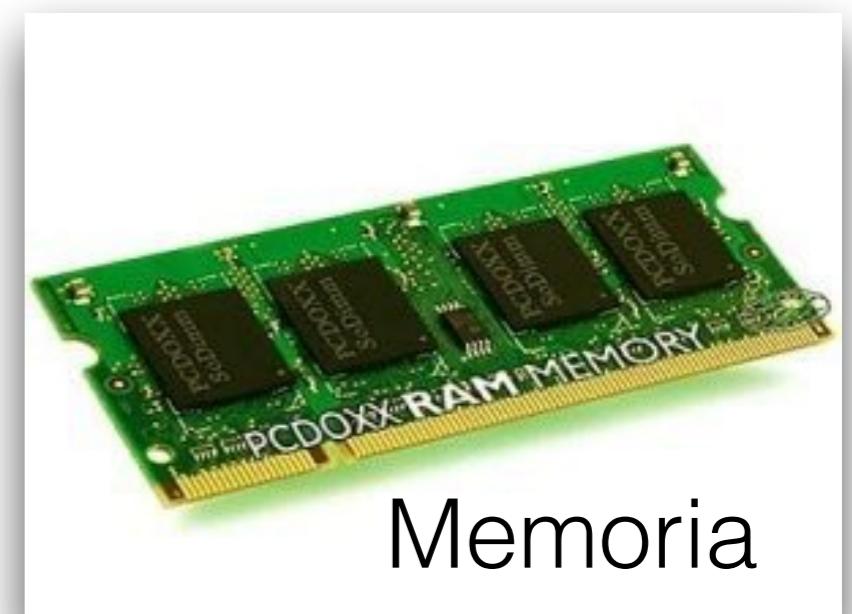
- Son registros de 1 bit que nos proveen información de control
- En la arquitectura Orga1:
 - (**Z**)ero
 - (**N**)egative
 - (**C**)arry
 - o(**V**)erflow
- Se modifican con todas las operaciones aritmético-lógicas (ie, todas salvo MOV, CALL, RET, JMP, Jxx).



Memoria



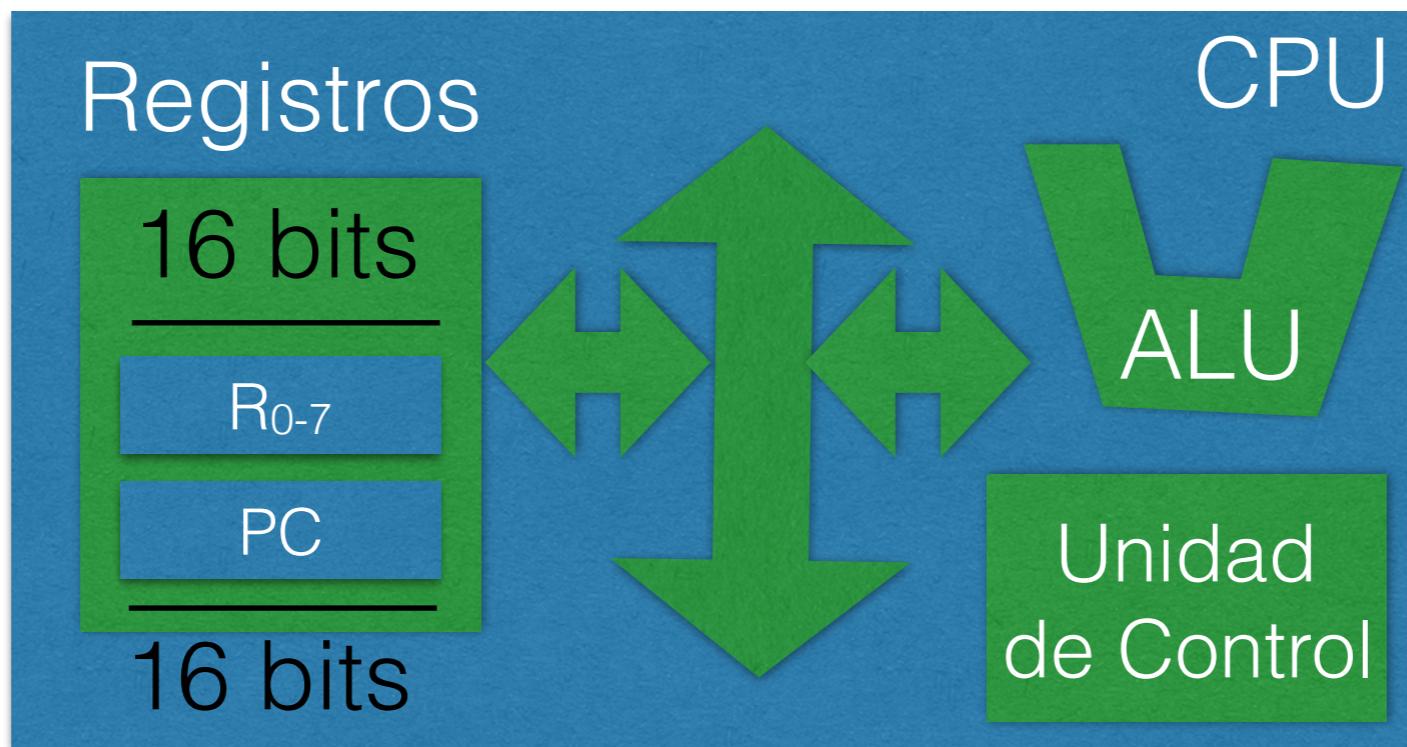
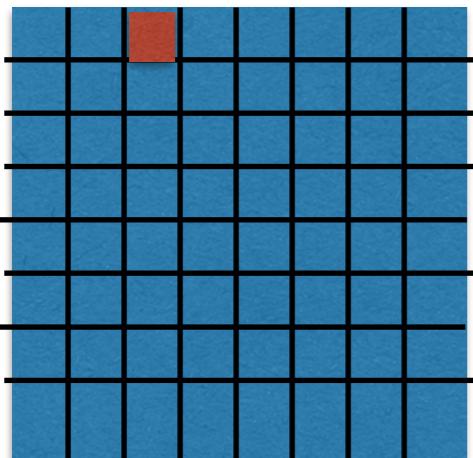
- Direcciones de 16 bits (de 0x0000 a 0xFFEF)
 - Las direcciones faltantes son para E/S (Pr.5)
- Palabras de 16 bits
- Direccionamiento a palabra (65520 palabras)
- Distintos modos de direccionamiento
 - Directo, Indirecto, etc.



Memoria

Arquitectura ORGA1

2^{16} direcciones
16 bits por dirección



Palabras
de 16 bits

Bus

Instrucciones



- Formato de Instrucción de Longitud **variable**
- Instrucciones:
 - Tipo I: 2 operandos (hasta 48 bits)
 - Tipo II: 1 operando (hasta 32bits)
 - Tipo III: sin operandos (16 bits)
 - Tipo IV: desplazamiento (16 bits)

Ciclo de Instrucción



- 1a) **UC** obtiene primer palabra de la instrucción de memoria (de la dirección apuntada por el PC) e incrementa el PC
- 1b) **UC** decodifica la primer palabra de la instrucción
- 1c) Si es necesario: busca más palabras de la instrucción (usando el PC) e incrementa el PC
- 2) **UC** decodifica la instrucción completa
- 3) **UC** ejecuta la instrucción
- 4) Ir a Paso 1a)

Tipo I: 2 operandos

<i>4 bits</i>	<i>6 bits</i>	<i>6 bits</i>	<i>16 bits</i>	<i>16 bits</i>
cod. op.	destino	fuente	constante destino (opcional)	constante fuente (opcional)

operación	cod. op.	efecto	modifica flags
MOV <i>d, f</i>	0001	$d \leftarrow f$	no
ADD <i>d, f</i>	0010	$d \leftarrow d + f$ (suma binaria)	sí
SUB <i>d, f</i>	0011	$d \leftarrow d - f$ (resta binaria)	sí
AND <i>d, f</i>	0100	$d \leftarrow d \text{ and } f$	sí (*)
OR <i>d, f</i>	0101	$d \leftarrow d \text{ or } f$	sí (*)
CMP <i>d, f</i>	0110	Modifica los <i>flags</i> según el resultado de $d - f$.	sí
ADDC <i>d, f</i>	1101	$d \leftarrow d + f + \text{carry}$ (suma binaria)	sí

(*) dejan el flag de *carry* (C) y el de *overflow* (V) en cero.

Modo	Codificación	Resultado
Inmediato	000000	c16
Directo	001000	[c16]
Indirecto	011000	[[c16]]
Registro	100 <i>rrr</i>	R <i>rrr</i>
Indirecto registro	110 <i>rrr</i>	[R <i>rrr</i>]
Indexado	111 <i>rrr</i>	[R <i>rrr</i> + c16]

c16 es una constante de 16 bits.

R*rrr* es el registro indicado por los últimos tres bits del código de operando.

Las instrucciones que tienen como destino un operando de tipo *inmediato* son consideradas como inválidas por el procesador, excepto el CMP.

Tipo II: 1 operando

- Tipo IIa: Instrucciones de un operando destino

4 bits	6 bits	6 bits	16 bits
operación	cod. op.	destino	000000 constante destino (opcional)
NEG d	1000		$d \leftarrow$ el inverso aditivo de d
NOT d	1001		$d \leftarrow \text{not } d$ (bit a bit)

(*) deja el flag de carry (C) y el de overflow (V) en cero.

- Tipo IIb: Instrucciones de un operando fuente

4 bits	6 bits	6 bits	16 bits
cod. op.	000000	fuente	constante fuente (opcional)
JMP f	1010		$\text{PC} \leftarrow f$
CALL f	1011		$[\text{SP}] \leftarrow \text{PC}, \text{SP} \leftarrow \text{SP} - 1, \text{PC} \leftarrow f$

Tipo III: Sin operandos

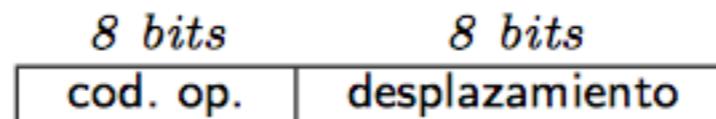
	<i>4 bits</i>	<i>6 bits</i>	<i>6 bits</i>
cod. op.	000000	000000	

operación	cod. op.	efecto
RET	1100	PC \leftarrow [SP+1], SP \leftarrow SP + 1

- No se modifica el valor de los Flags.

Tipo IV: Saltos Relativos Condicionales

- El salto se produce si se cumple la “condición de salto” correspondiente
- $PC := PC + \text{desplazamiento}$
- El desplazamiento se representa en complemento a 2 de 8 bits
- No se modifican los flags



Codop	Operación	Descripción	Condición de Salto
1111 0001	JE	Igual / Cero	Z
1111 1001	JNE	Distinto	not Z
1111 0010	JLE	Menor o igual	Z or (N xor V)
1111 1010	JG	Mayor	not (Z or (N xor V))
1111 0011	JL	Menor	N xor V
1111 1011	JGE	Mayor o igual	not (N xor V)
1111 0100	JLEU	Menor o igual sin signo	C or Z
1111 1100	JGU	Mayor sin signo	not (C or Z)
1111 0101	JCS	Carry / Menor sin signo	C
1111 0110	JNEG	Negativo	N
1111 0111	JVS	Overflow	V

Modos de Direccionamiento

Modos de direccionamiento

- Del hecho de usar distintos tipos de operandos surge la idea de tener distintos **modos de direccionamiento**
- Es decir, distintas formas posibles de acceder a los operandos (y además, optimizar la ejecución)
- Algunos modos de direccionamiento usuales:
 - Inmediato
 - Directo (o absoluto)
 - Indirecto
 - Registro
 - Indirecto con registro
 - Desplazamiento
 - Indexado

Inmediato

- El operando está en la misma instrucción (Cte).
- Ejemplos (Orga1):
 - JMP 0x1110
 - CMP 0x00FF 0x11FF
- No hay acceso adicional a la memoria
- Es rápido (pero poco flexible)



Directo

- El operando está en la dirección de memoria referenciada por DIR.

- Es decir, el operando=[DIR]



- Ejemplo (Orga1):
 - NEG [0x1941]
- Ideal para acceso a **variables**
- Hay un único acceso adicional a memoria

Registro

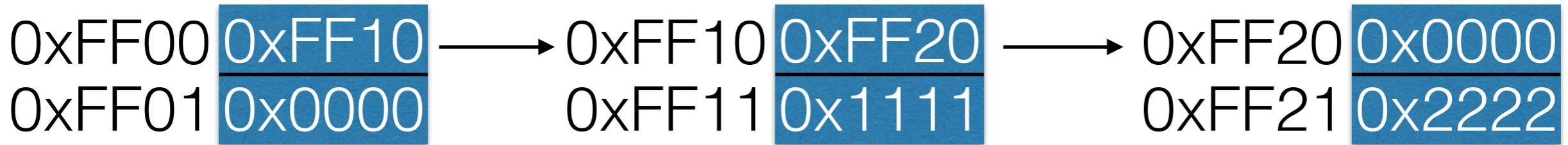
- El operando está en un registro del CPU
 - Es decir, operando=registro n OpCode Rn
- Ejemplo (Orga1):
 - NOT R0
 - Instrucción corta y rápida
 - No requiere acceso a memoria (ya está el operando en el CPU)
 - Espacio de direcciones (ie, cantidad de registros) limitado

Registro Indirecto

- El operando está almacenada en la dirección de memoria indicada por un registro
 - Es decir, operando=[Rn]
- Necesita un acceso a memoria menos que el direccionamiento indirecto
- Cómodo para acceder a arrays

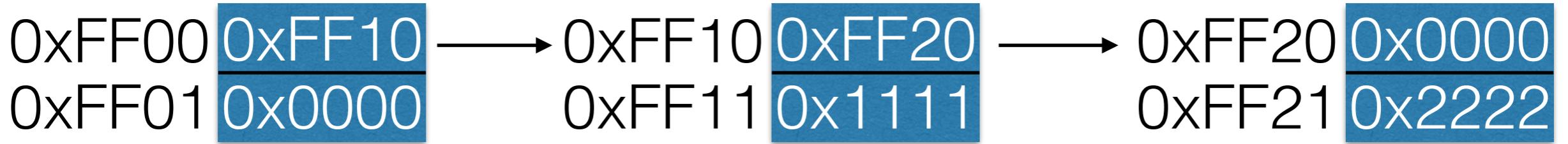


Ejemplo



- Por ejemplo, supongamos que almacenamos una lista dinámica como <PROX_DIRECCION,VALOR>
 - El último elemento tiene PROX_DIRECCION=0x0000
- ¿Cómo escribimos un programa en lenguaje ORGA1 para que cuente la longitud de esta lista y lo almacene en R0?

Ejemplo



R0 := 0

R1 := 0xFF00

Mientras (R1 != 0x0000)

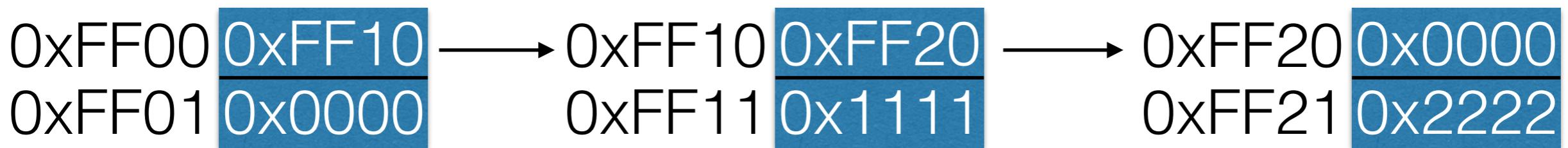
Pseudo-código

R0 := R0 + 1

R1 := [R1]

Fin Mientras

Registro Indirecto



Inicio

```
MOV R0, 0x0000  
MOV R1, 0xFF00  
CMP [R1], 0x0000
```

Mientras: JE FinMientras

```
ADD R0, 0x0001  
MOV R1, [R1]  
JMP Mientras
```

Ensamblador Orga1

FinMientras: ...

Indirecto

- El operando está en la dirección de memoria apuntada por el contenido de la dirección
 - Es decir, operando=[[Dir]]
 - Típico para acceder a listas, arreglos u otras estructuras dinámicas
- Necesita 2 accesos a memoria:
 - 1ro: obtener el puntero P almacenado en Dir
 - 2do: obtener el operando almacenado en P.



Indirecto



R0 := 0

[p_actual] := 0xFF00

Mientras ([p_actual] != 0x0000)

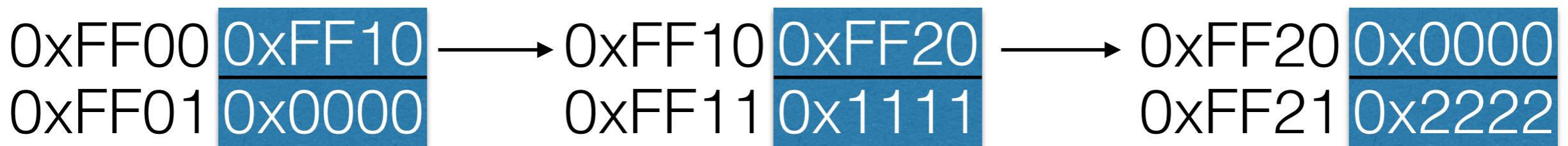
Pseudo-código

R0 := R0 + 1

[actual] := [[p_actual]]

Fin Mientras

Indirecto



p_actual: DW 0x0000

Inicio

MOV R0, 0x0000

MOV [p_actual], 0xFF00

Mientras: CMP [p_actual], 0x0000

JE FinMientras

ADD R0, 0x0001

MOV [p_actual], [[p_actual]]

JMP Mientras

FinMientras: ...

Ensamblador Orga1

Desplazamiento

- El operando está almacenado en la dirección surgida de sumar el valor del registro y del desplazamiento (también llamado offset)

- Es decir, operando=[Rn+D]



- Ideal para acceder a campos de registros
- También para arrays de registros
 - Rn se mueve dentro del array
 - D selecciona el campo

Indexado

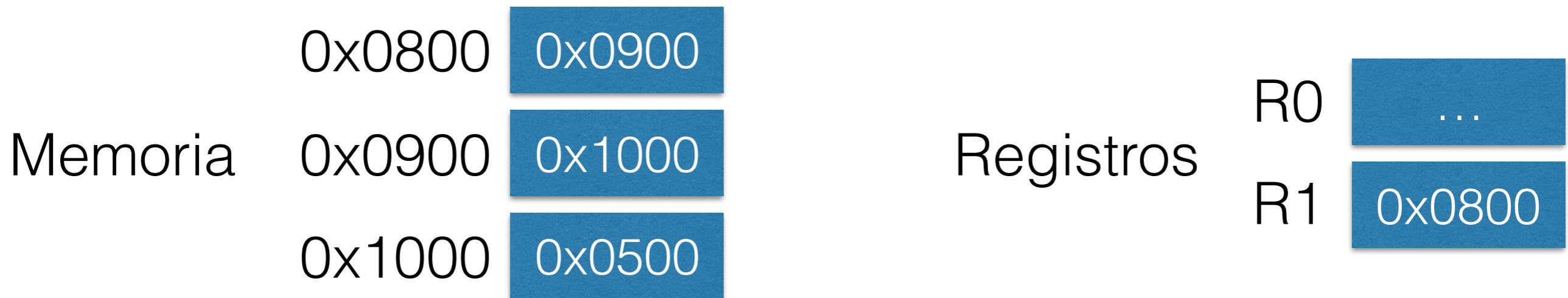
- El operando está almacenado en la dirección producto de sumar una dirección con el valor de un registro
 - Es decir, operando=[Dir+Rn]
- Similar al direccionamiento de desplazamiento
 - Diferencia: D puede ser bastante menor al espacio de direcciones (4 bits vs. 32 bits)
 - Ideal para **arreglos**



Modos de direccionamiento

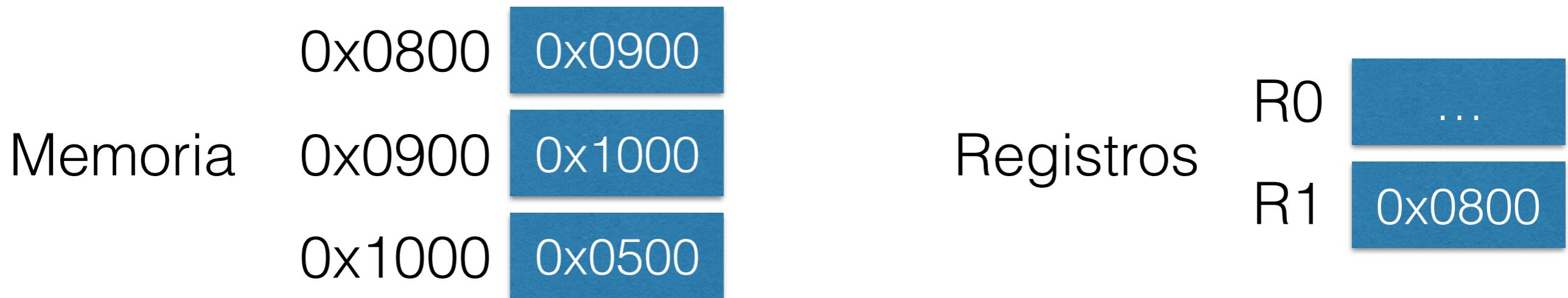
- Facilitan el acceso a los operandos
 - Arreglos
 - Matrices
 - Listas
 - Registros (Records)
- La ausencia de un modo de direccionamiento no impide el acceso al operando.

Modos de direccionamiento



Instrucción Máq. Orga1	Modo de Direccionamiento fuente	Valor final de R0
MOV R0, 0x0800	Inmediato	
MOV R0, [0x0800]	Directo	
MOV R0, [[0x0800]]	Indirecto	
MOV R0, R1	Registro	
MOV R0, [R1]	Indirecto Registro	
MOV R0, [R1+0x0800]	Indexado	

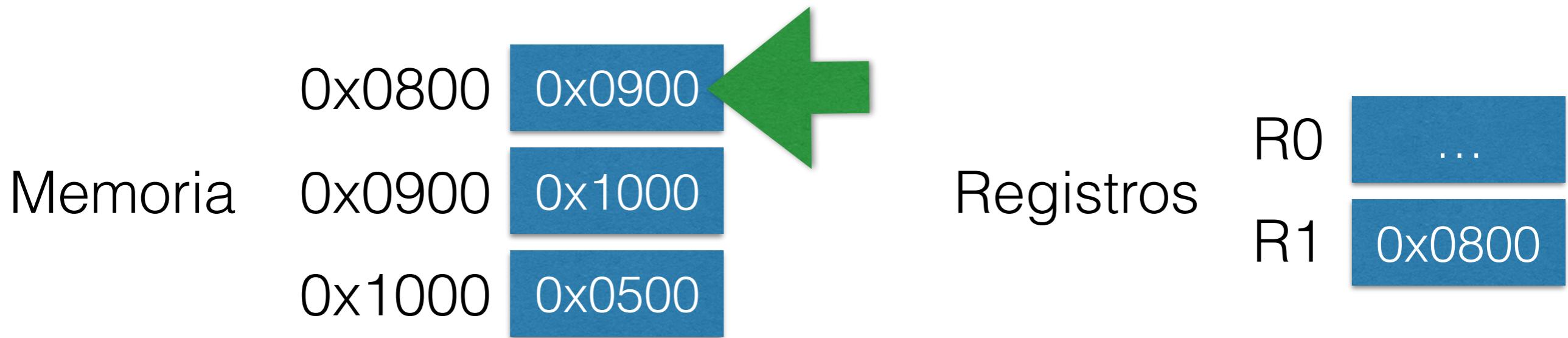
Modos de direccionamiento



Instrucción Máq. Orga1	Modo de Direccionamiento fuente	Valor final de R0
MOV R0, 0x0800	Inmediato	0x0800
MOV R0, [0x0800]	Directo	
MOV R0, [[0x0800]]	Indirecto	
MOV R0, R1	Registro	
MOV R0, [R1]	Indirecto Registro	
MOV R0, [R1+0x0800]	Indexado	

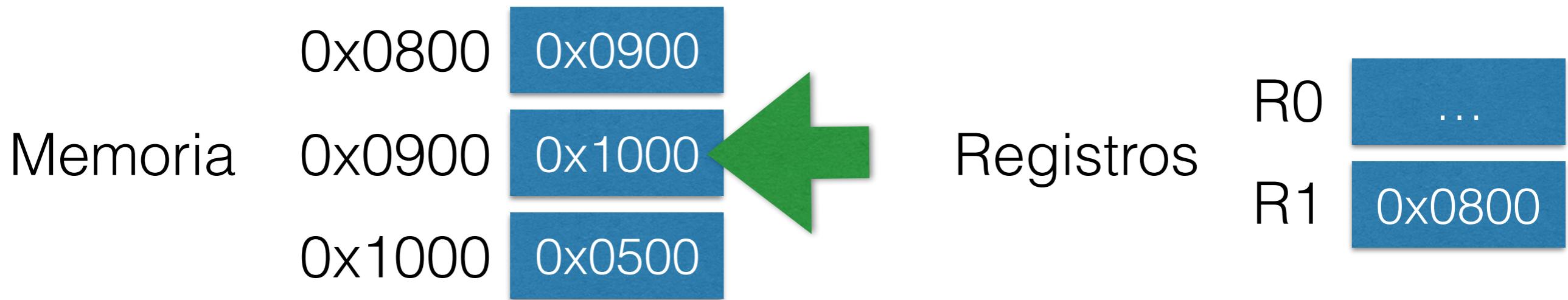


Modos de direccionamiento



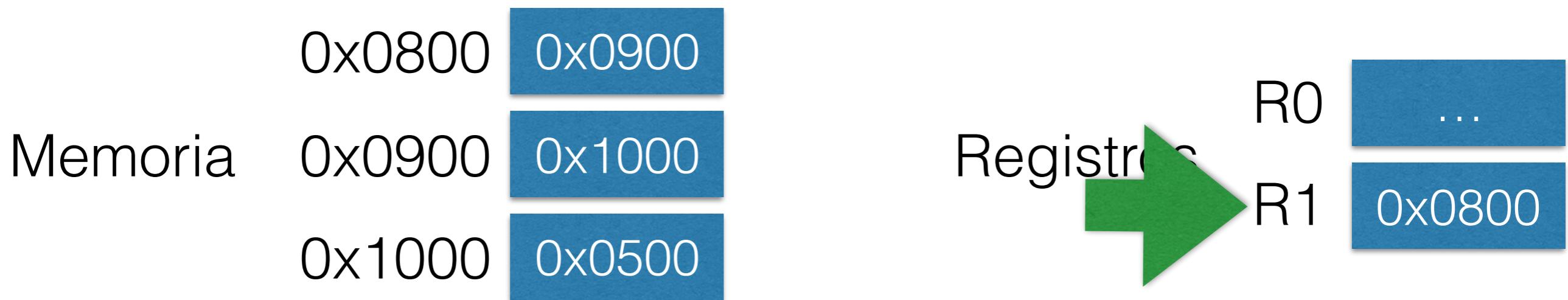
Instrucción Máq. Orga1	Modo de Direccionamiento fuente	Valor final de R0
MOV R0, 0x0800	Inmediato	0x0800
MOV R0, [0x0800]	Directo	0x0900
MOV R0, [[0x0800]]	Indirecto	
MOV R0, R1	Registro	
MOV R0, [R1]	Indirecto Registro	
MOV R0, [R1+0x0800]	Indexado	

Modos de direccionamiento



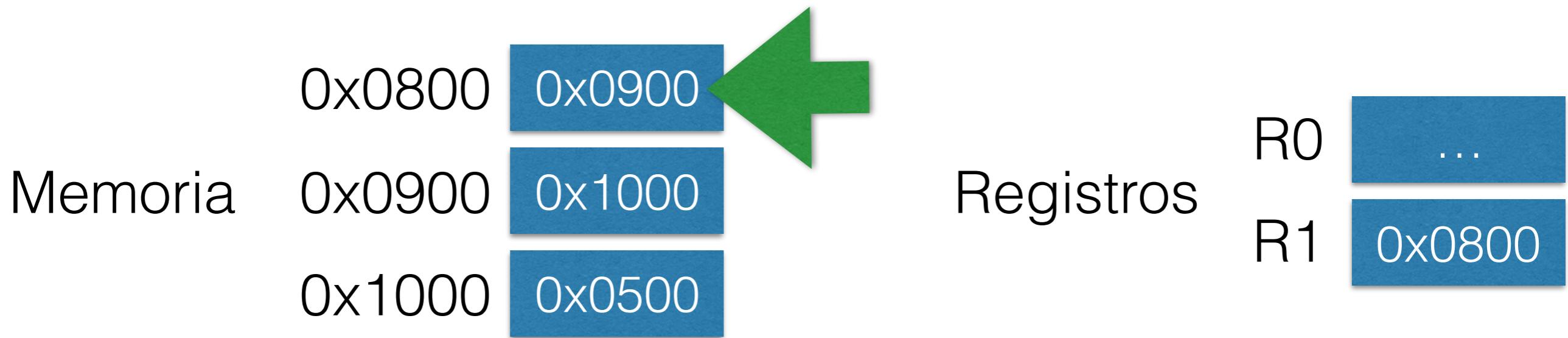
Instrucción Máq. Orga1	Modo de Direccionamiento fuente	Valor final de R0
MOV R0, 0x0800	Inmediato	0x0800
MOV R0, [0x0800]	Directo	0x0900
MOV R0, [[0x0800]]	Indirecto	0x1000
MOV R0, R1	Registro	
MOV R0, [R1]	Indirecto Registro	
MOV R0, [R1+0x0800]	Indexado	

Modos de direccionamiento



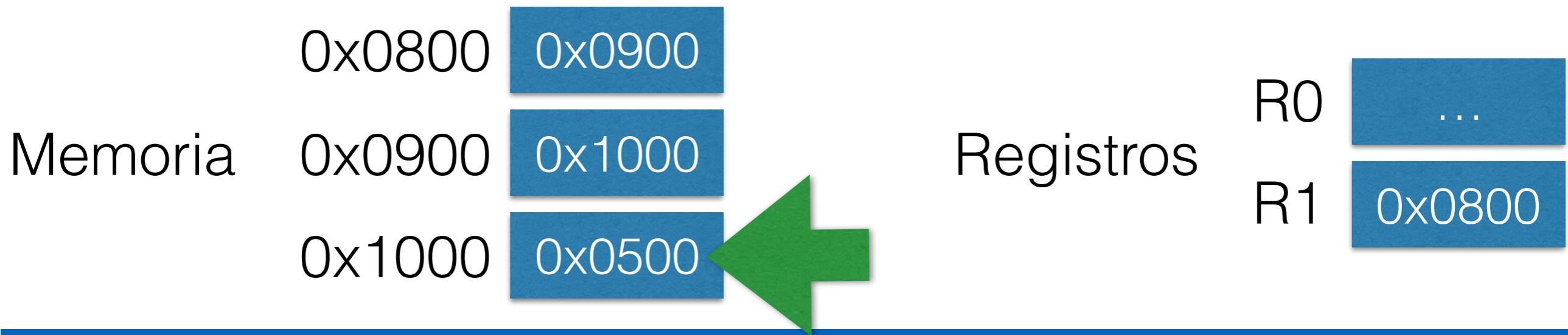
Instrucción Máq. Orga1	Modo de Direccionamiento fuente	Valor final de R0
MOV R0, 0x0800	Inmediato	0x0800
MOV R0, [0x0800]	Directo	0x0900
MOV R0, [[0x0800]]	Indirecto	0x1000
MOV R0, R1	Registro	0x0800
MOV R0, [R1]	Indirecto Registro	
MOV R0, [R1+0x0800]	Indexado	

Modos de direccionamiento



Instrucción Máq. Orga1	Modo de Direccionamiento fuente	Valor final de R0
MOV R0, 0x0800	Inmediato	0x0800
MOV R0, [0x0800]	Directo	0x0900
MOV R0, [[0x0800]]	Indirecto	0x1000
MOV R0, R1	Registro	0x0800
MOV R0, [R1]	Indirecto Registro	0x0900
MOV R0, [R1+0x0800]	Indexado	

Modos de direccionamiento



Instrucción Máq. Orga1	Modo de Direccionamiento fuente	Valor final de R0
MOV R0, 0x0800	Inmediato	0x0800
MOV R0, [0x0800]	Directo	0x0900
MOV R0, [[0x0800]]	Indirecto	0x1000
MOV R0, R1	Registro	0x0800
MOV R0, [R1]	Indirecto Registro	0x0900
MOV R0, [R1+0x0800]	Indexado	0x0500

Ensamblado

Ensamblador (Assembler)

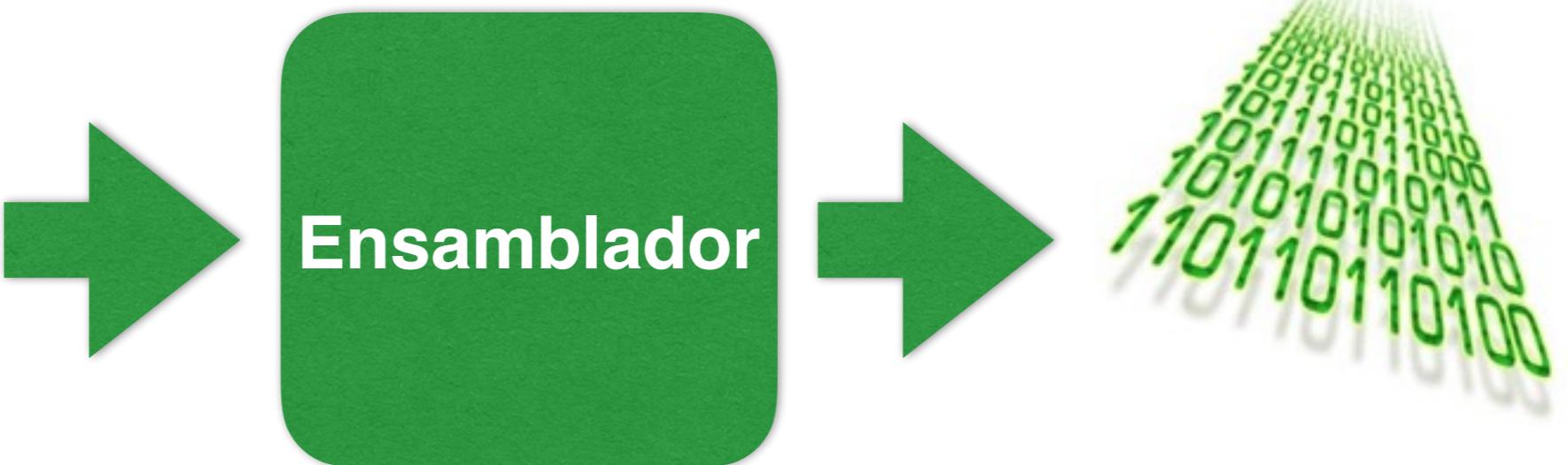
- **Lenguaje** ensamblador != **Programa** ensamblador
- **Lenguaje** ensamblador:
 - Lenguaje en el que escribimos programas para Orga1
 - Textual (no confundir con código de máquina)
 - Permite usar etiquetas

Ensamblador (Assembler)

- **Lenguaje** ensamblador != **Programa** ensamblador
- **Programa** ensamblador:
 - Traduce programas en lenguaje ensamblador (texto) a código máquina (ceros y unos)
 - También calcula el valor de etiquetas y desplazamiento

Ensamblador

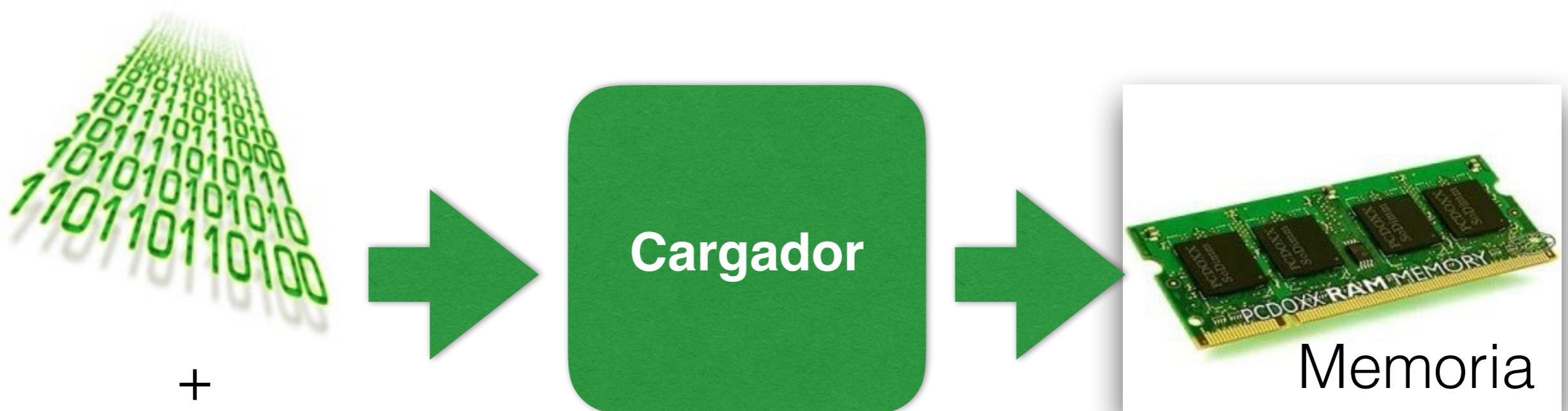
1. MOV R0, 0x0010
2. MOV R1, 0x0001
3. ADD R0, R1



Programa en Lenguaje
Ensamblador

Programa en Lenguaje
Máquina

Ensamblador



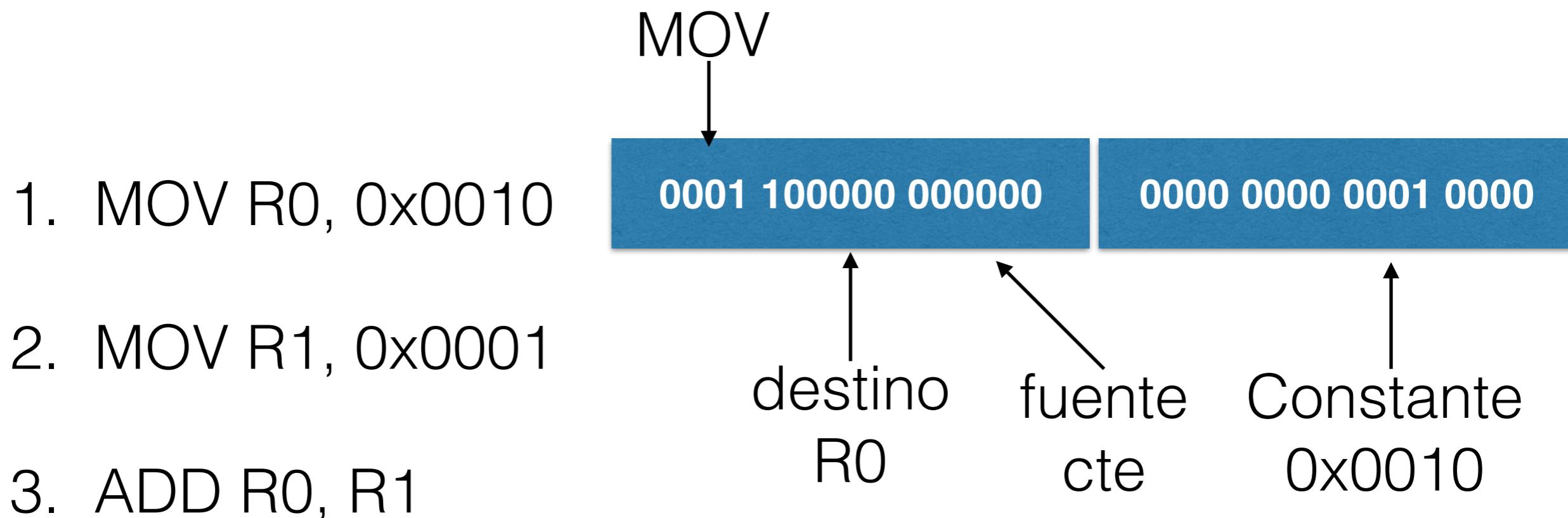
Dirección de Memoria
para cargar el Programa

Programa cargado
en memoria

Ensamblador

1. MOV R0, 0x0010
2. MOV R1, 0x0001
3. ADD R0, R1

Ensamblador



Ensamblador

1. MOV R0, 0x0010



2. MOV R1, 0x0001



3. ADD R0, R1

MOV

destino
R1

fuente
cte

Constante
0x0001

Ensamblador

1. MOV R0, 0x0010

0001 100000 000000

0000 0000 0001 0000

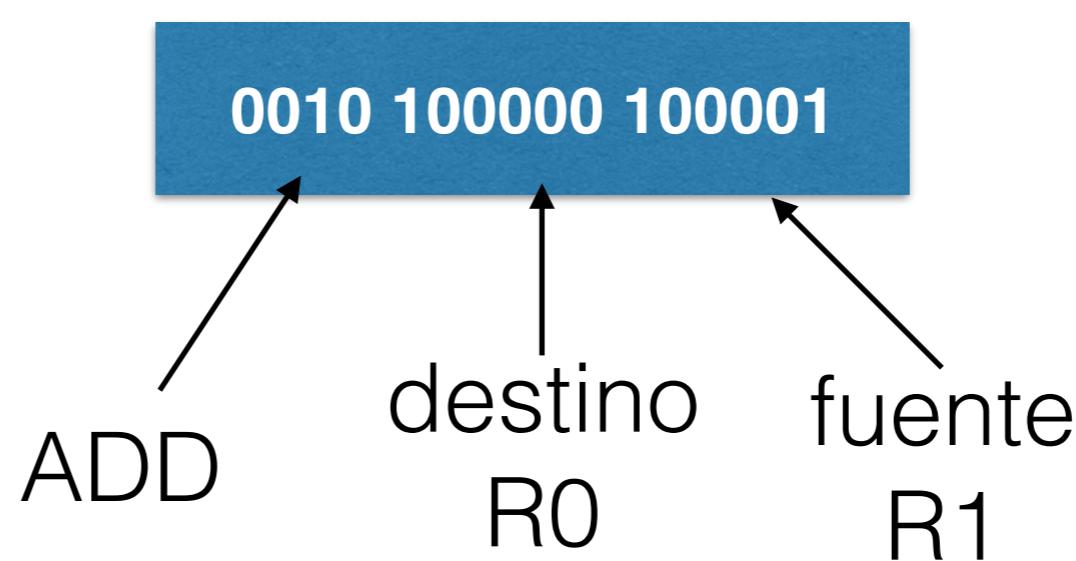
2. MOV R1, 0x0001

0001 100001 000000

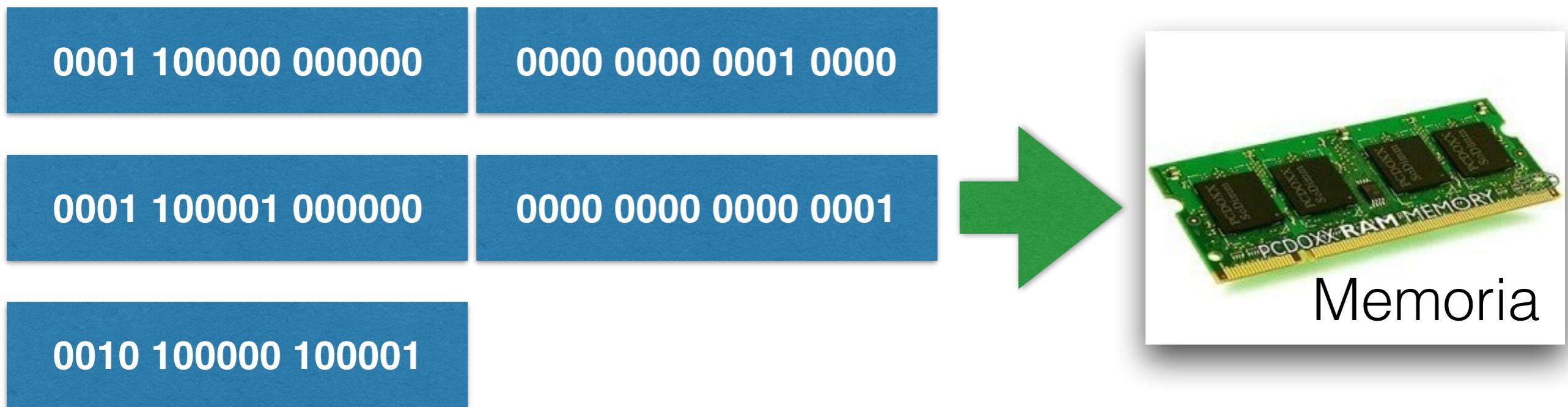
0000 0000 0000 0001

3. ADD R0, R1

0010 100000 100001



Cargador



Dirección de Carga: 0x0010

Cargador

0x000F

...

0x0010

0001 100000 000000

MOV R0, 0x0010

0x0011

0000 0000 0001 0000

0x0012

0001 100001 000000

MOV R1, 0x0001

0x0013

0000 0000 0000 0001

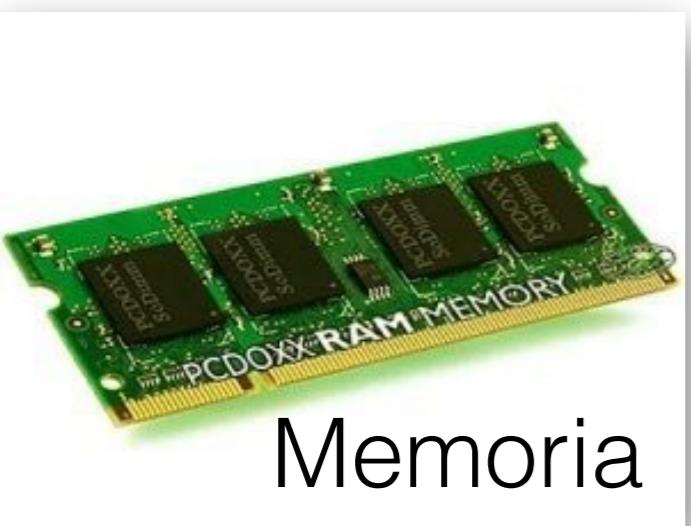
0x0014

0010 100000 100001

ADD R0, R1

0x0015

...



Memoria

Etiquetas

Inicio: MOV R0, 0x0010

MOV R1, 0x0001

ADD R0, R1

JMP **Inicio**

Etiquetas

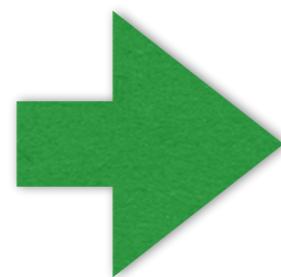
Inicio: MOV R0, 0x0010

MOV R1, 0x0001

ADD R0, R1

JMP **Inicio**

+ Dirección de Carga
del Programa (0x0010)



Etiquetas

0x0010: MOV R0, 0x0010

0x0012: MOV R1, 0x0001

0x0014: ADD R0, R1

0x0015: JMP **Inicio**

+ Dirección de Carga
del Programa (0x0010)

Etiqueta	Valor
Inicio	0x0010

Etiquetas

0x0010: MOV R0, 0x0010

0x0012: MOV R1, 0x0001

0x0014: ADD R0, R1

0x0015: JMP **0x0010**

+ Dirección de Carga
del Programa (0x0010)

Etiqueta	Valor
Inicio	0x0010

Etiquetas

0x0010: MOV R0, 0x0010

0001 100000 000000

0000 0000 0001 0000

0x0012: MOV R1, 0x0001

0001 100001 000000

0000 0000 0000 0001

0x0014: ADD R0, R1

0010 100000 100001

0x0015: JMP **0x0010**

+ Dirección de Carga
del Programa (0x0010)

Etiquetas

0x0010: MOV R0, 0x0010

0001 100000 000000

0000 0000 0001 0000

0x0012: MOV R1, 0x0001

0001 100001 000000

0000 0000 0000 0001

0x0014: ADD R0, R1

0010 100000 100001

0x0015: JMP **0x0010**

0001 000000 000000

0000 0000 0001 0000

+ Dirección de Carga
del Programa (0x0010)

JMP

fuente
cte

Constante
0x0010

Etiquetas

0x0010:	0001 100000 000000
0x0011:	0000 0000 0001 0000
0x0012:	0001 100001 000000
0x0013:	0000 0000 0000 0001
0x0014:	0010 100000 100001
0x0015:	0001 000000 000000
0x0016:	0000 0000 0001 0000

¿Qué pasa si la dirección de carga es 0xF01?

Etiquetas

0x0010: 0001 100000 000000

0x0011: 0000 0000 0001 0000

0x0012: 0001 100001 000000

0x0013: 0000 0000 0000 0001

0x0014: 0010 100000 100001

0x0015: 0001 000000 000000

0x0016: 0000 0000 0001 0000

¿Qué pasa si la dirección de carga es 0xF01?

Rta: La codificación del Salto Absoluto es distinta

Saltos Relativos

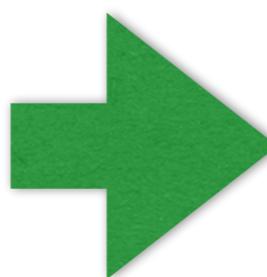
Inicio: MOV R0, 0x0010

MOV R1, 0x0001

ADD R0, R1

JE **Inicio** # Salta si Z=1

+ Dirección de Carga
del Programa (0x0010)



Saltos Relativos

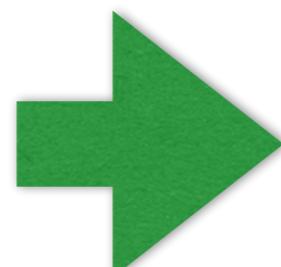
0x0010: MOV R0, 0x0010

0x0012: MOV R1, 0x0001

0x0014: ADD R0, R1

0x0015: JE Inicio

+ Dirección de Carga
del Programa (0x0010)



Etiqueta	Valor
----------	-------

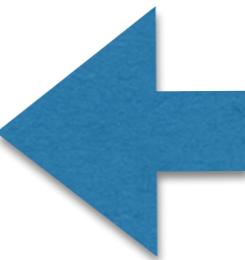
Inicio 0x0010

--

--

Saltos Relativos

0x0010: MOV R0, 0x0010

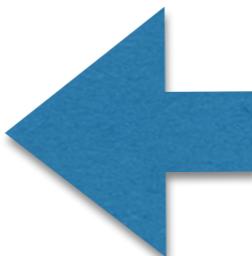


¿Cuánto debe valer el PC para que se ejecute esta instrucción?

0x0012: MOV R1, 0x0001

0x0014: ADD R0, R1

0x0015: JE Inicio

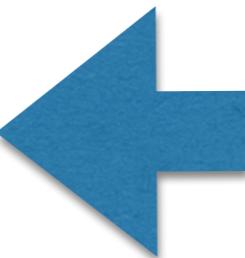


¿Cuál es el valor del PC luego de ejecutarse esta instrucción?

+ Dirección de Carga
del Programa (0x0010)

Saltos Relativos

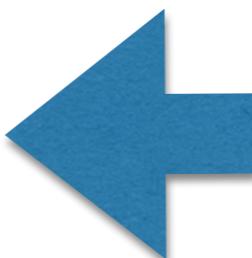
0x0010: MOV R0, 0x0010



¿Cuánto debe valer el PC para que se ejecute esta instrucción?

Rta: PC=0x0010

0x0014: ADD R0, R1



¿Cuál es el valor del PC luego de ejecutarse esta instrucción?

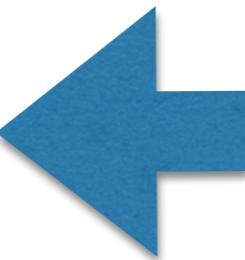
0x0015: JE Inicio

+ Dirección de Carga
del Programa (0x0010)

Rta: PC=0x0016

Saltos Relativos

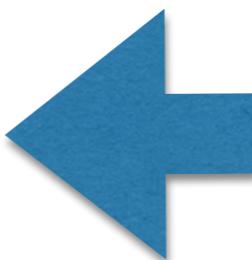
0x0010: MOV R0, 0x0010



¿Cuánto debe valer el PC para que se ejecute esta instrucción?

Rta: PC=0x0010

0x0014: ADD R0, R1



¿Cuál es el valor del PC luego de ejecutarse esta instrucción?

0x0015: JE -6

+ Dirección de Carga
del Programa (0x0010)

Rta: PC=0x0016

Saltos Relativos

0x0010: MOV R0, 0x0010

0001 100000 000000

0000 0000 0001 0000

0x0012: MOV R1, 0x0001

0001 100001 000000

0000 0000 0000 0001

0x0014: ADD R0, R1

0010 100000 100001

0x0015: JE -6

+ Dirección de Carga
del Programa (0x0010)

Saltos Relativos

0x0010: MOV R0, 0x0010

0001 100000 000000

0000 0000 0001 0000

0x0012: MOV R1, 0x0001

0001 100001 000000

0000 0000 0000 0001

0x0014: ADD R0, R1

0010 100000 100001

0x0015: JE -6

11110001 xxxxxx

+ Dirección de Carga
del Programa (0x0010)

↑
JE

desplazamiento -6
(complemento a 2 de 6bits)

Saltos Relativos

0x0010: MOV R0, 0x0010

0001 100000 000000

0000 0000 0001 0000

0x0012: MOV R1, 0x0001

0001 100001 000000

0000 0000 0000 0001

0x0014: ADD R0, R1

0010 100000 100001

0x0015: JE -6

11110001 111010

+ Dirección de Carga
del Programa (0x0010)

↑
JE

desplazamiento -6
(complemento a 2 de 6bits)



Saltos



- Saltos **Absolutos**: Utilizan la dirección final del programa
 - El ensamblado (codificación) del programa es distinta de acuerdo a la dirección de carga.
 - Ejemplo: Saltar a la dirección 0xFF01
- Saltos **Relativos**: Utilizan un desplazamiento
 - El ensamblado (codificación) del programa es independiente de la dirección de carga
 - Ejemplos: Saltar 6 posiciones atrás, Saltar 15 posiciones hacia adelante, etc.

Ensamblador - Datos

- Para definir datos utilizamos el keyword “DW”(Define Word)
- DW no es una instrucción
- DW es una **directiva** al ensamblador

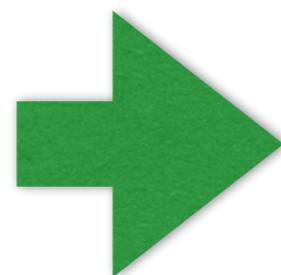
Ensamblador - Datos

V1: DW 0x001F

V2: DW 0xFF0

ADD R0, [V1]

ADD R0, [V2]



+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

V1: DW 0x001F

V2: DW 0xFF0

ADD R0, [V1]

ADD R0, [V2]

+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

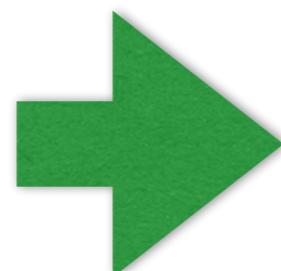
0x0010:DW 0x001F

0x0011:DW 0xFF0

0x0012:ADD R0, [V1]

0x0014:ADD R0, [V2]

+ Dirección de Carga
del Programa (0x0010)



Etiqueta	Valor
----------	-------

V1	0x0010
V2	0x0011

--

Ensamblador - Datos

0x0010: DW 0x001F

0x0011: DW 0xFF0

0x0012: ADD R0, [0x0010]

0x0014: ADD R0, [0x0011]

+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

0x0010: DW 0x001F

0000 0000 0001 1111

Dato 0x001F

0x0011: DW 0xFF00

0000 1111 1111 0000

Dato 0xFF00

0x0012: ADD R0, [0x0010]

0x0014: ADD R0, [0x0011]

+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

0x0010: DW 0x001F

0000 0000 0001 1111

0x0011: DW 0xFF0

0000 1111 1111 0000

0x0012: ADD R0, [0x0010]

0010 100000 001000

0000 0000 0001 0000

0x0014: ADD R0, [0x0011]

ADD

destino
R0

fuente
dirección

Dirección
0x0010

+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

0x0010: DW 0x001F

0000 0000 0001 1111

0x0011: DW 0xFF0

0000 1111 1111 0000

0x0012: ADD R0, [0x0010]

0010 100000 001000

0000 0000 0001 0000

0x0014: ADD R0, [0x0011]

0010 100000 001000

0000 0000 0001 0001

+ Dirección de Carga
del Programa (0x0010)

ADD

destino
R0

fuente
dirección

Dirección
0x0011

Programa en Memoria

0x0010:	0000 0000 0001 1111
0x0011:	0000 1111 1111 0000
0x0012:	0010 100000 001000
0x0013:	0000 0000 0001 0000
0x0014:	0010 100000 001000
0x0015:	0000 0000 0001 0001

Resumen

- **Lenguaje** Ensamblador:
 - Forma más cómoda de escribir programas para una computadora
- **Programa** Ensamblador:
 - Transforma un programa ensamblador en su codificación en ceros y unos
 - Resuelve etiquetas
 - Posee “directivas” (ejemplo: para indicar datos)

Resumen

- Arquitectura Von Neuman
 - Ciclo de Instrucción
 - Modos de direccionamiento
 - Máquina ORGA1
 - Lenguaje Ensamblador ORGA1
 - **Lenguaje Ensamblador vs. Programa Ensamblador**



Bibliografía

- Capítulo 5 – Tanenbaum
- Capítulo 4 y 5 – Null
- Capítulo 10 – Stallings