



Arquitectura del CPU

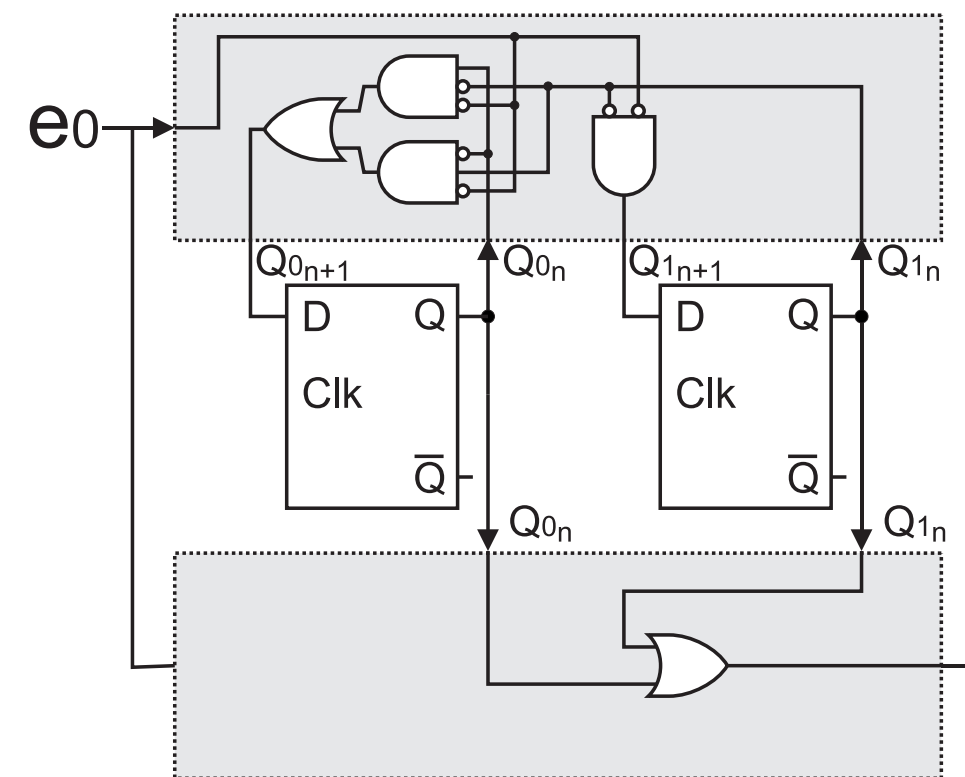
Organización del Computador 1
1er cuatrimestre 2017

COMPUTER LOVE

Agenda

¿De dónde venimos?

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



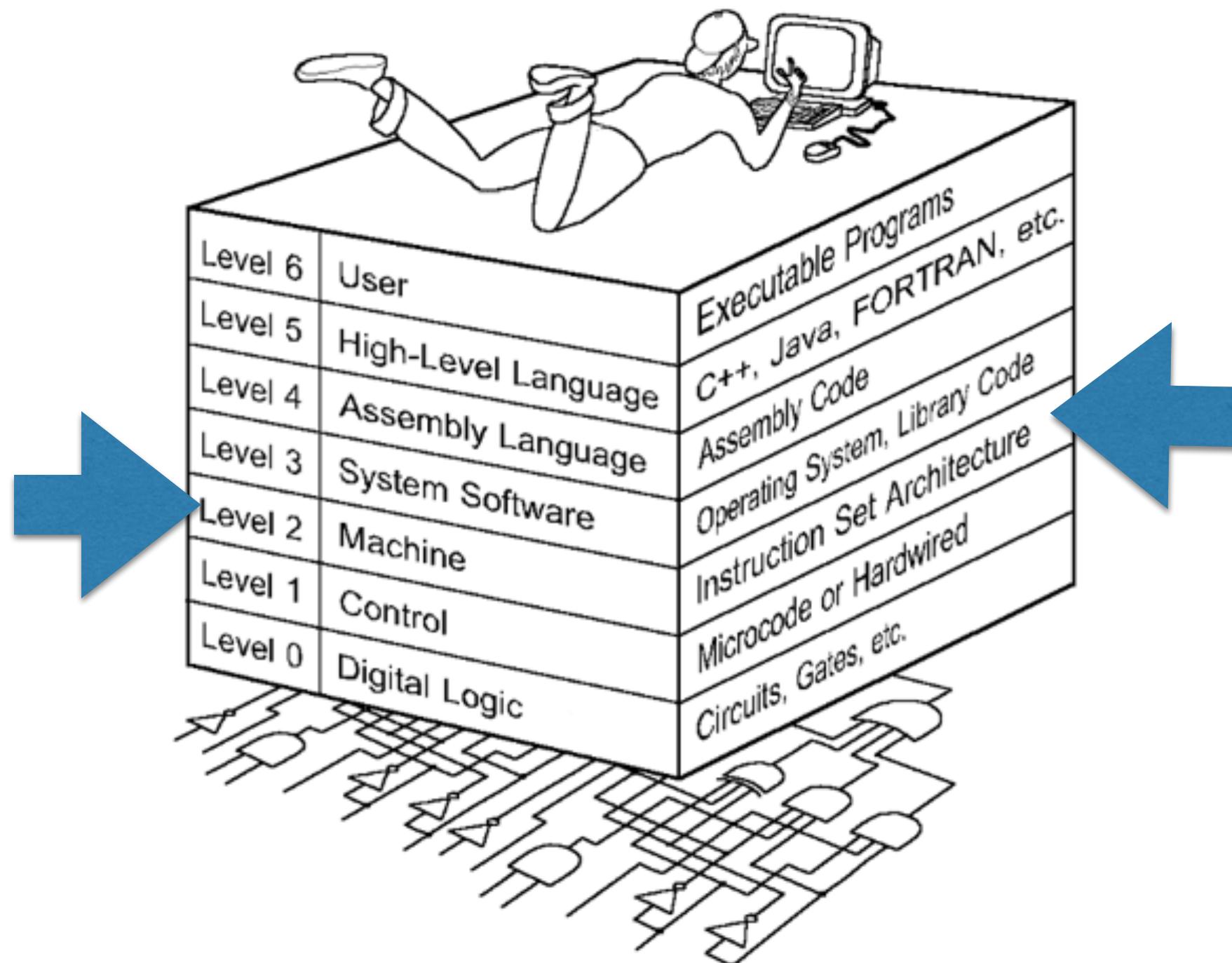
Agenda

¿Adónde vamos?

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



Niveles de Abstracción de una Computadora

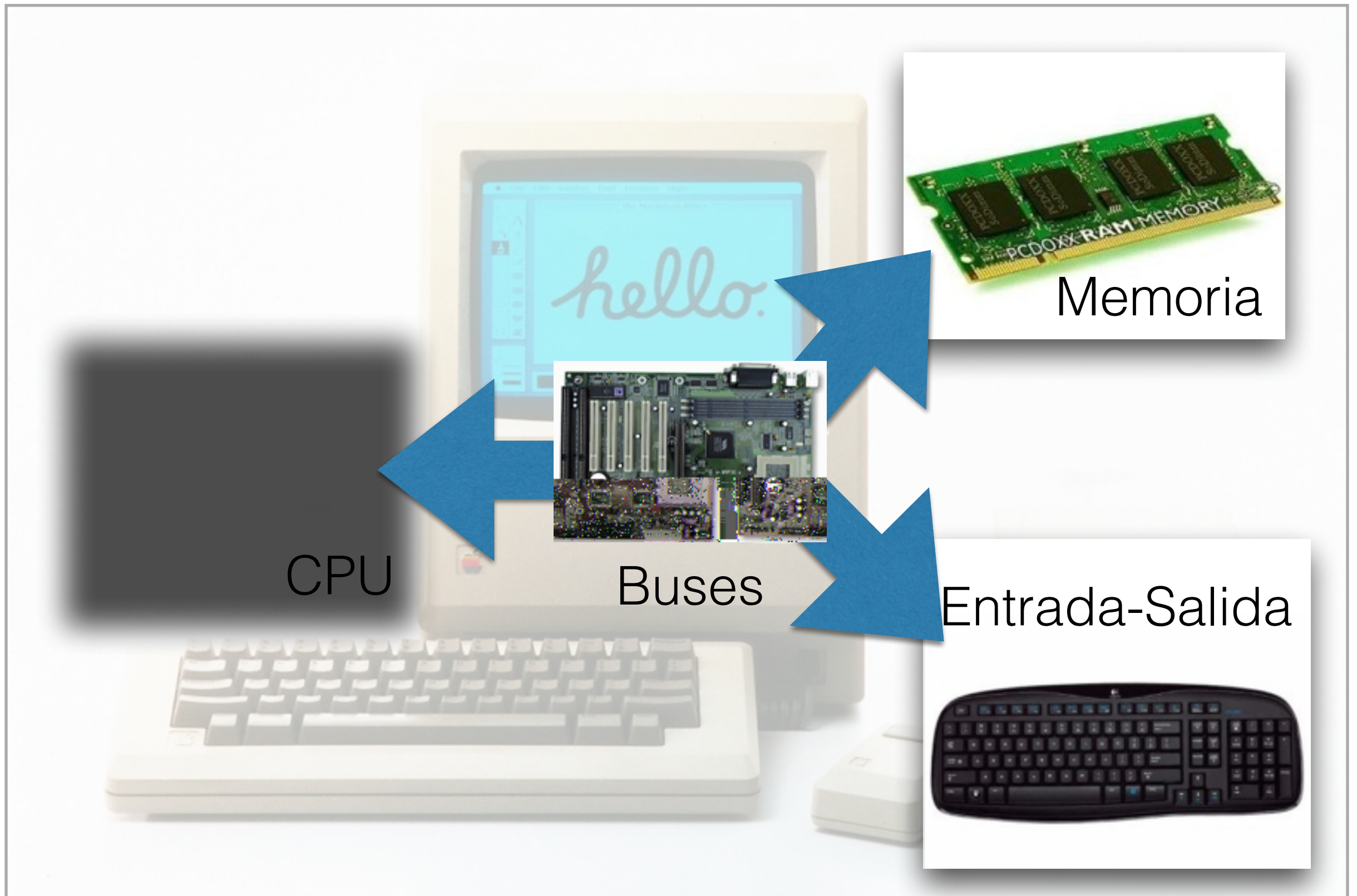


Modelo de Von Neumann

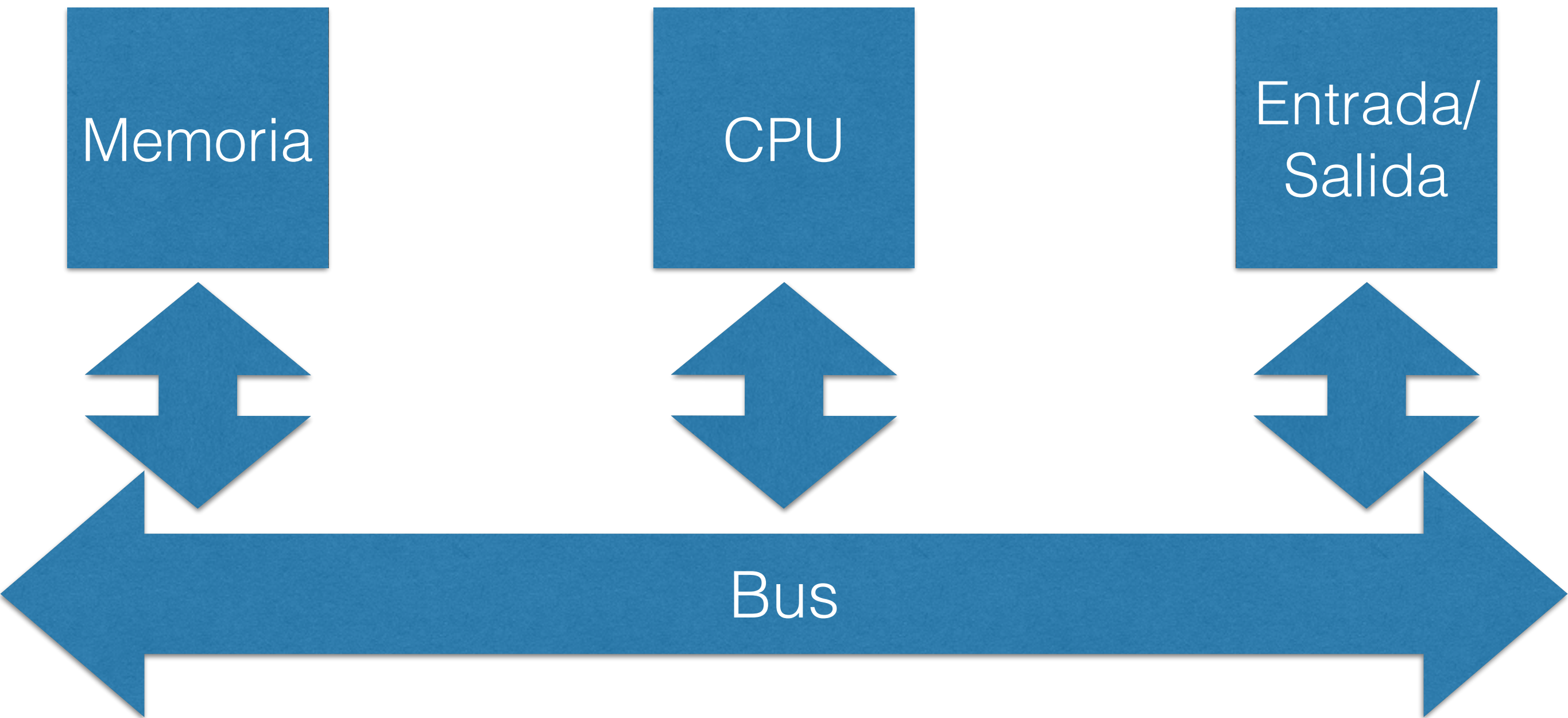
- John Von Neumann
(1903-1957)
- *Primer Informe sobre el EDVAC* (1945)
- Programas son almacenados como datos en memoria
- ¿Qué es un dato? ¿Qué es un programa?

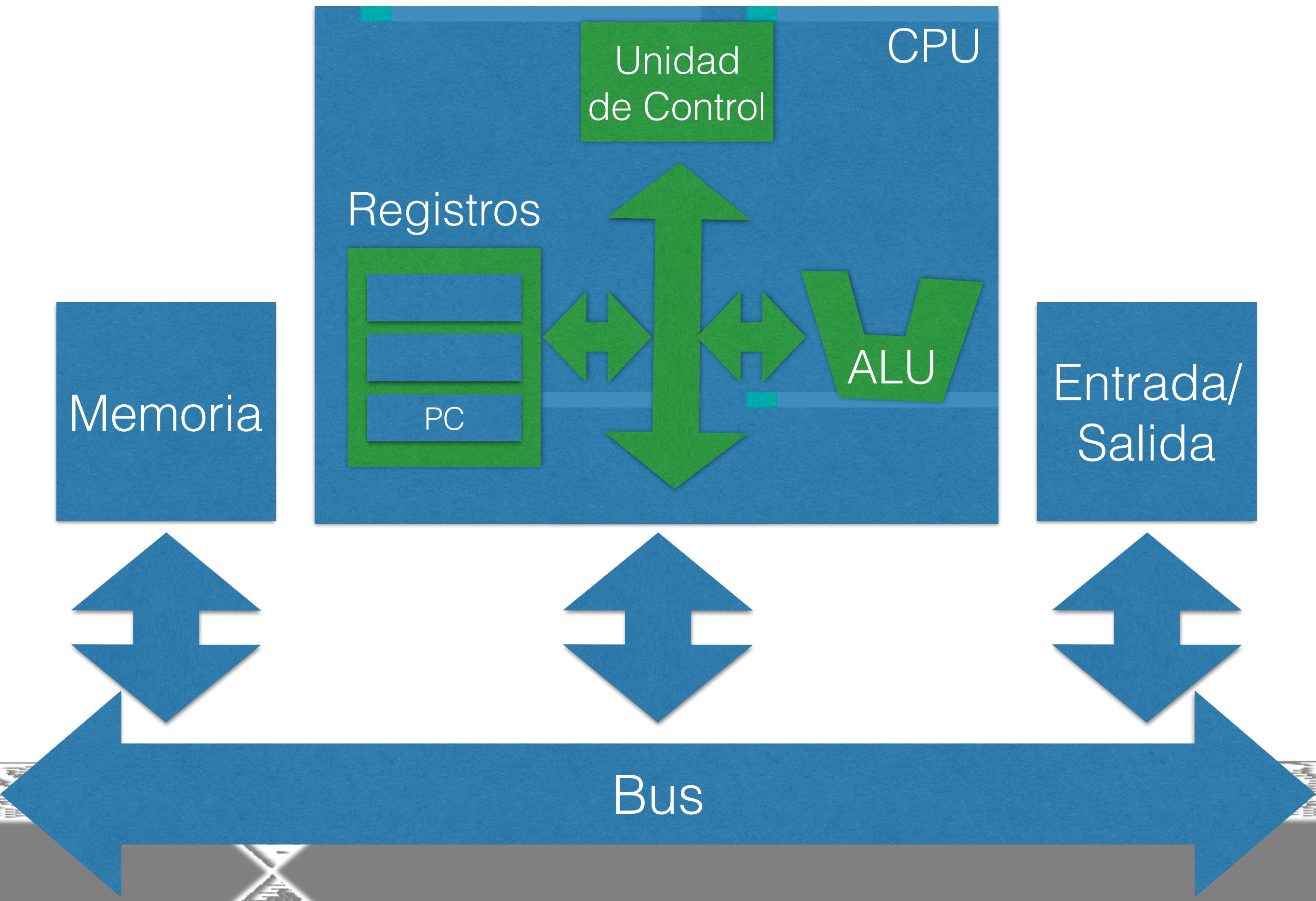






Modelo Von Neumann Organización





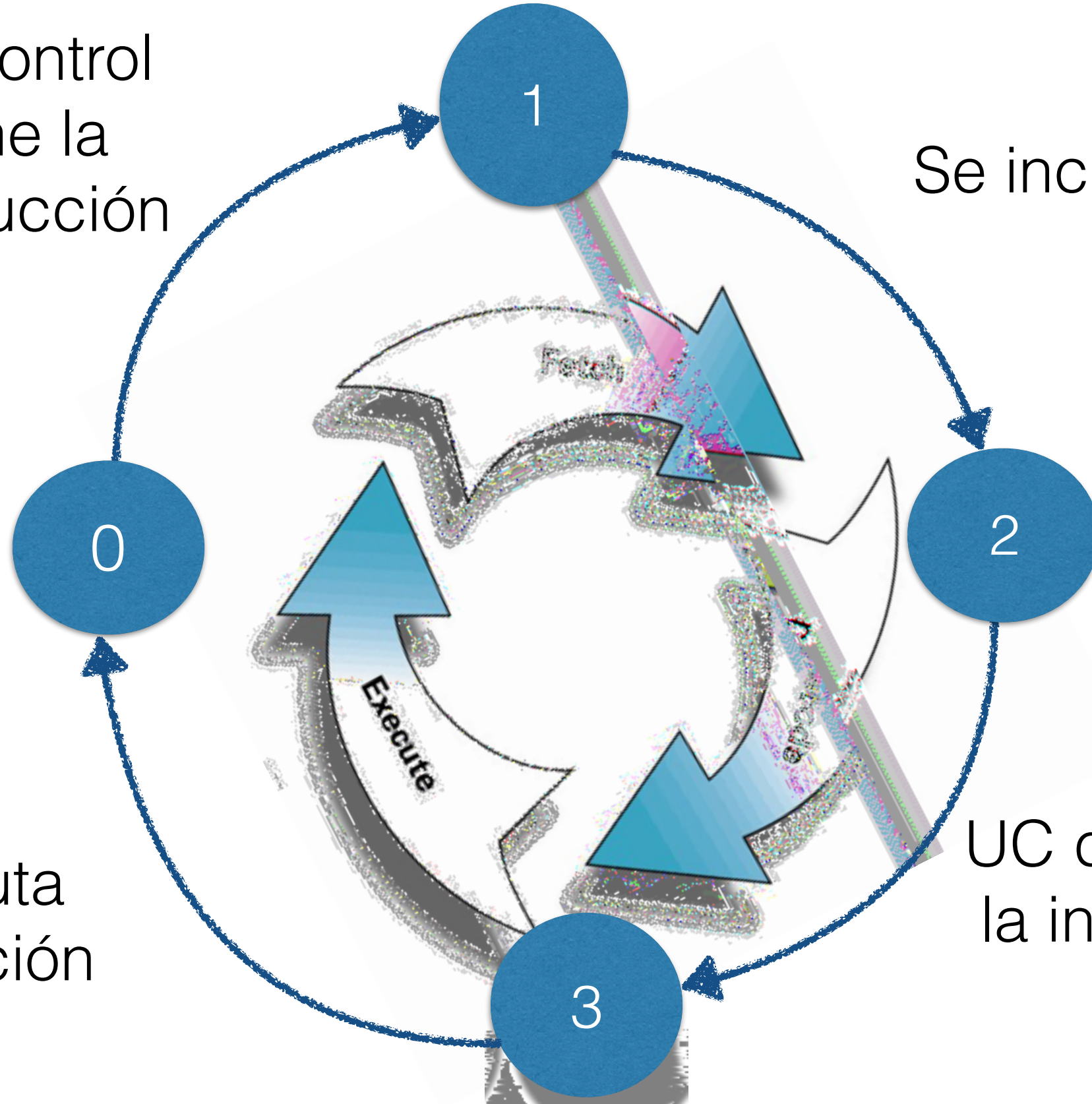
Ciclo de Instrucción

Unidad de Control
(UC) obtiene la
próxima instrucción

Se incrementa el
PC

UC ejecuta
la instrucción

UC decodifica
la 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?

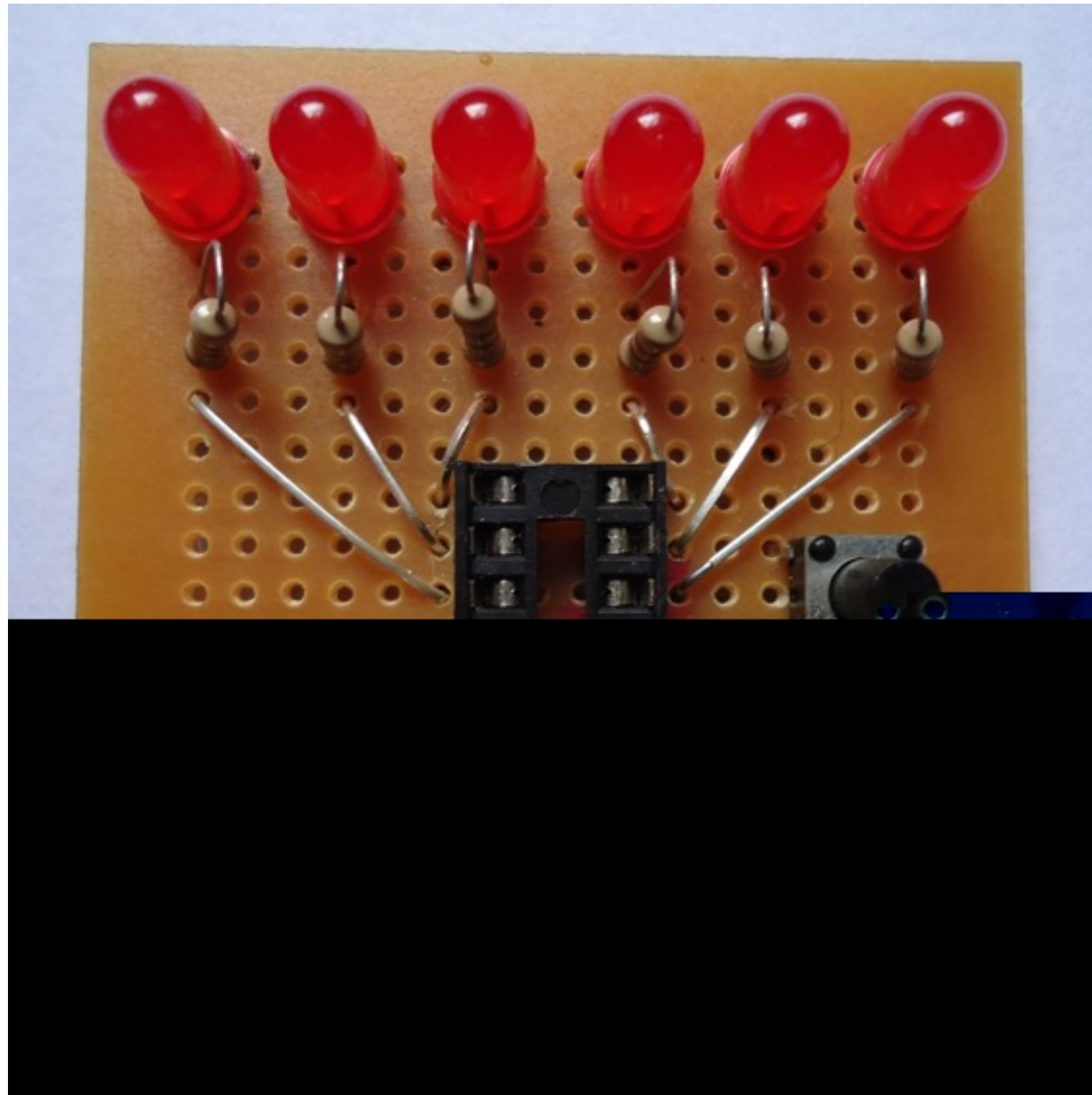
Un Instruction Set Real



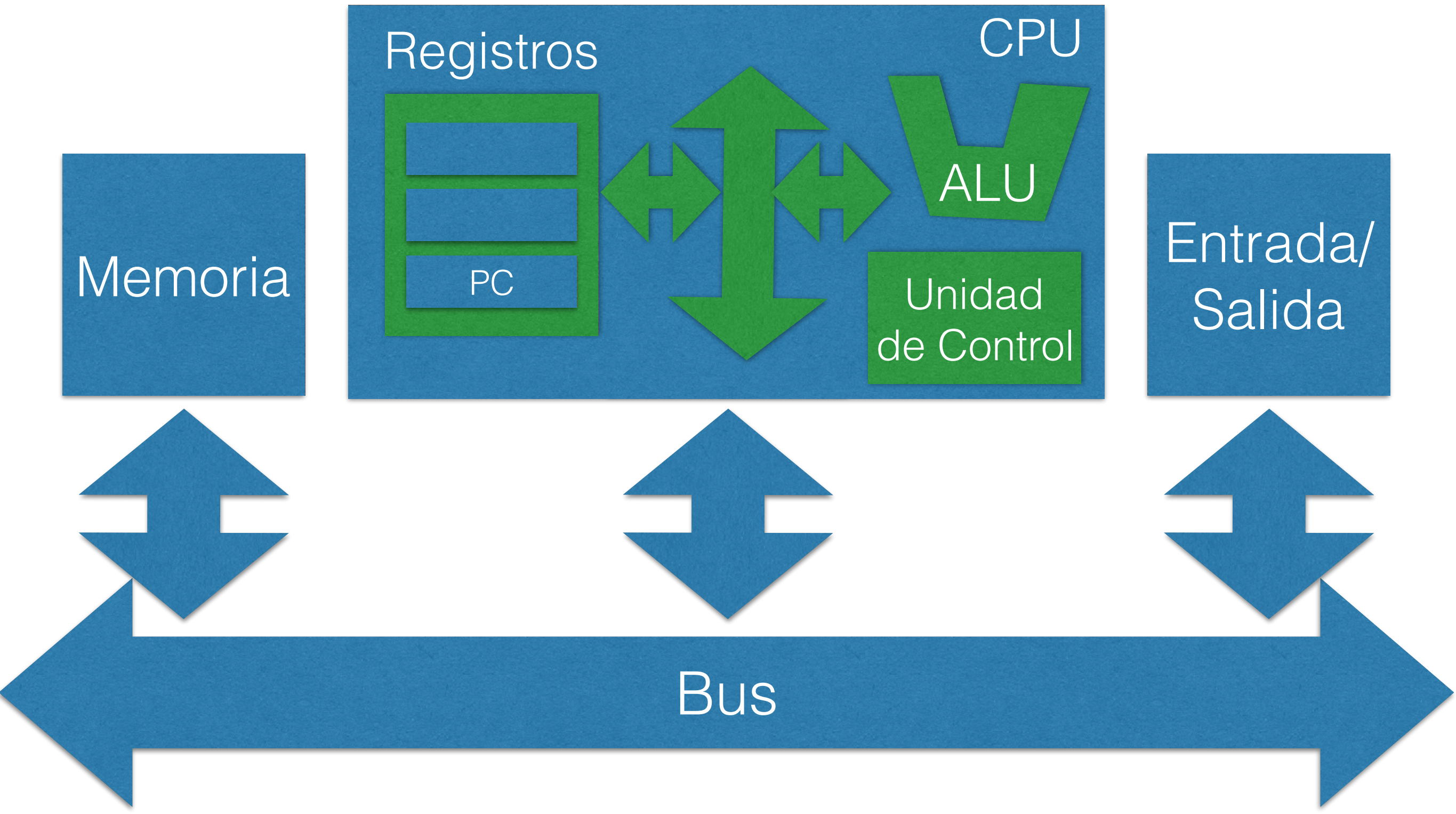
Una Arquitectura Real



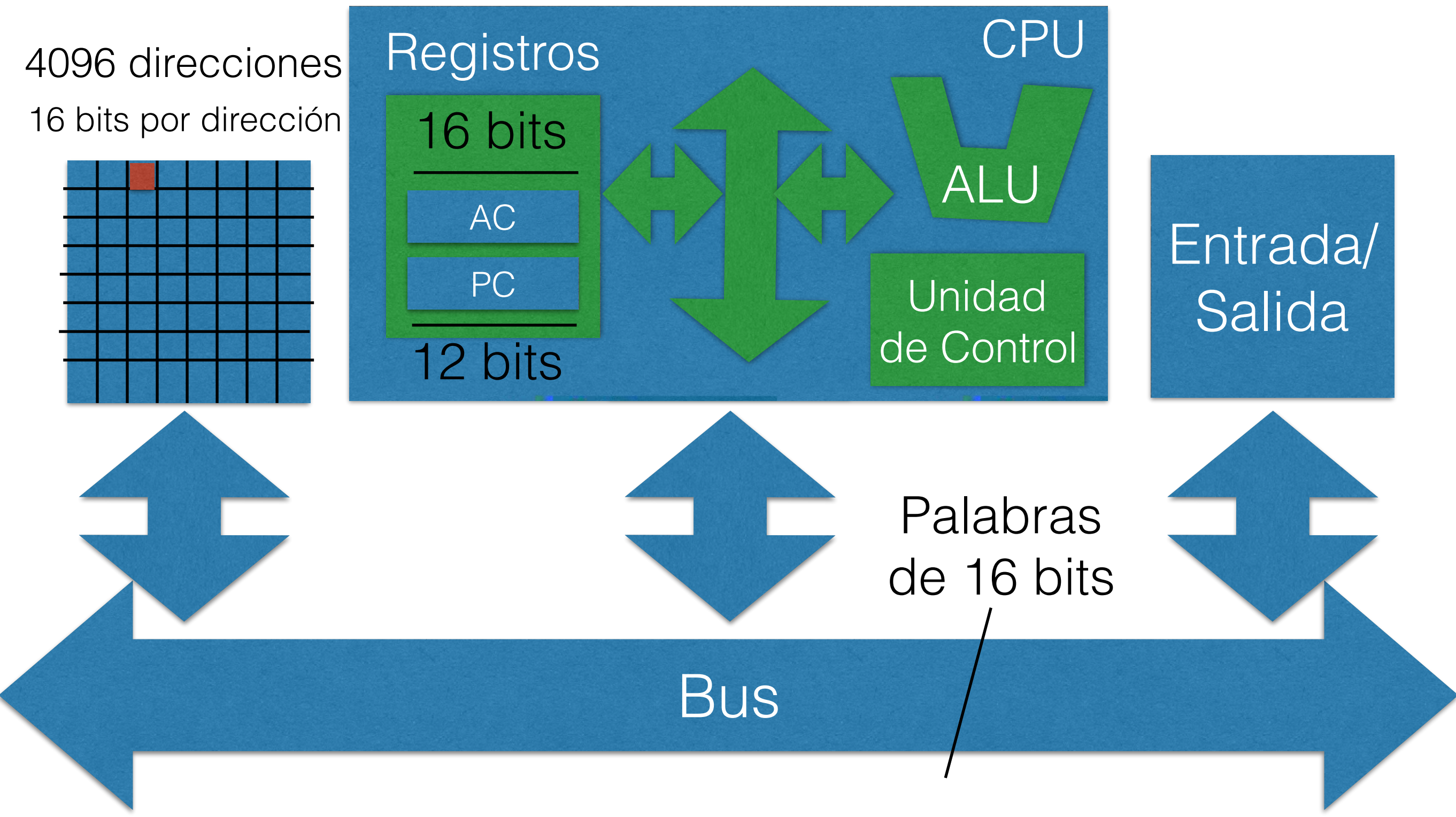
Otra Arquitectura Real



Arquitectura MARIE

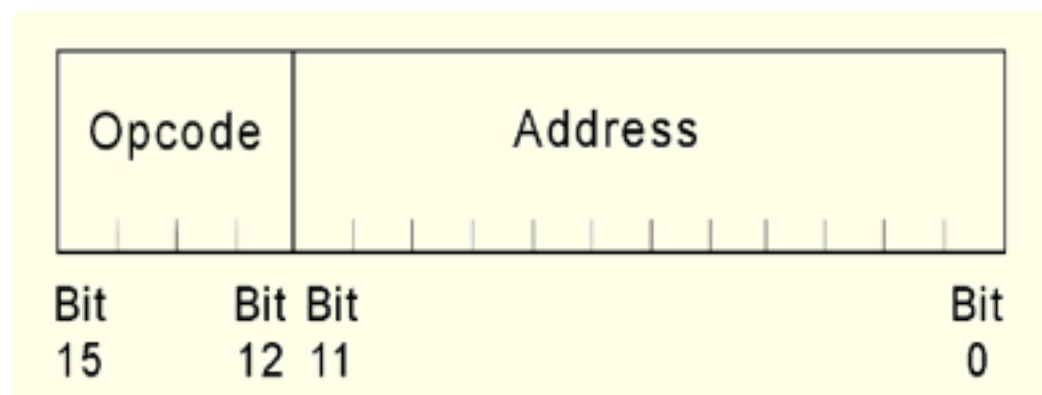


Arquitectura MARIE



MARIE: Arquitectura

- Formato de instrucción Fijo de 16 bits

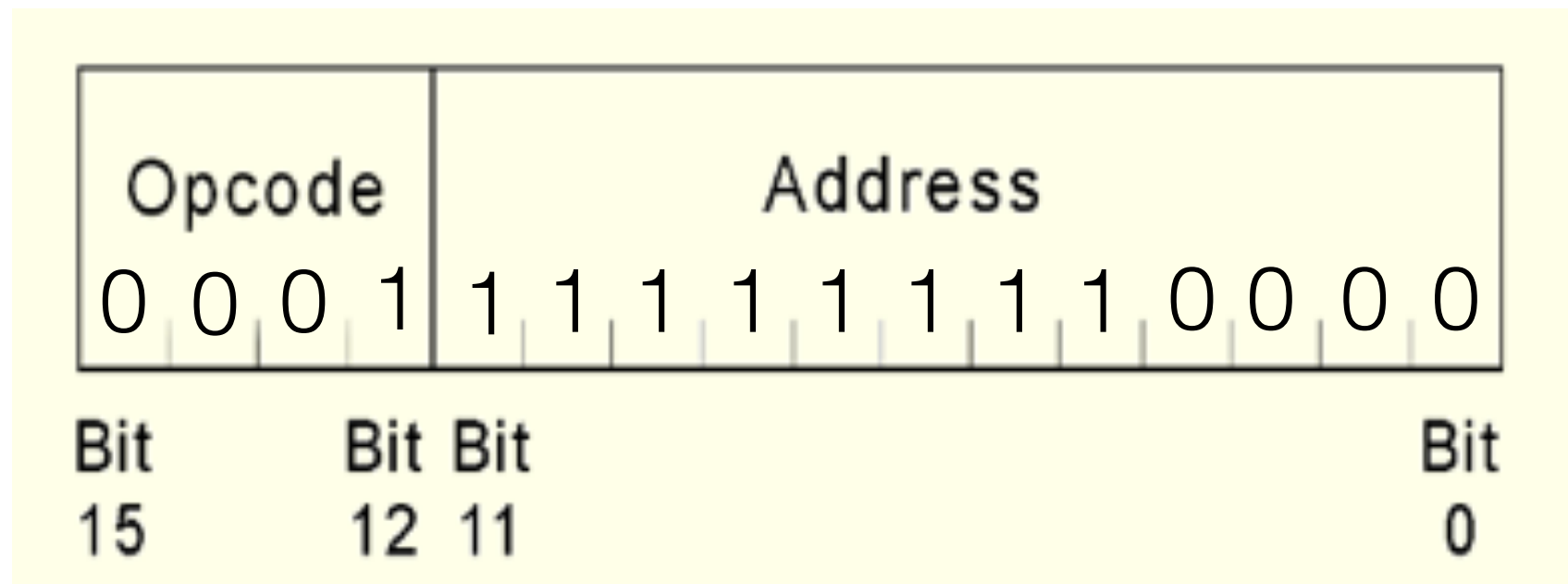
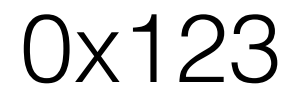


- Instrucciones

Opcode	Instrucción	Efecto
"0001"	Load X	Copia el contenido de la dirección X en AC
"0010"	Store X	Copia el contenido de AC en la dirección X
"0011"	Add X	$AC := AC + [X]$ (complemento a 2)
"0100"	Subt X	$AC := AC - [X]$ (complemento a 2)
"1010"	Clear	$AC := 0$

MARIE: Ejemplo

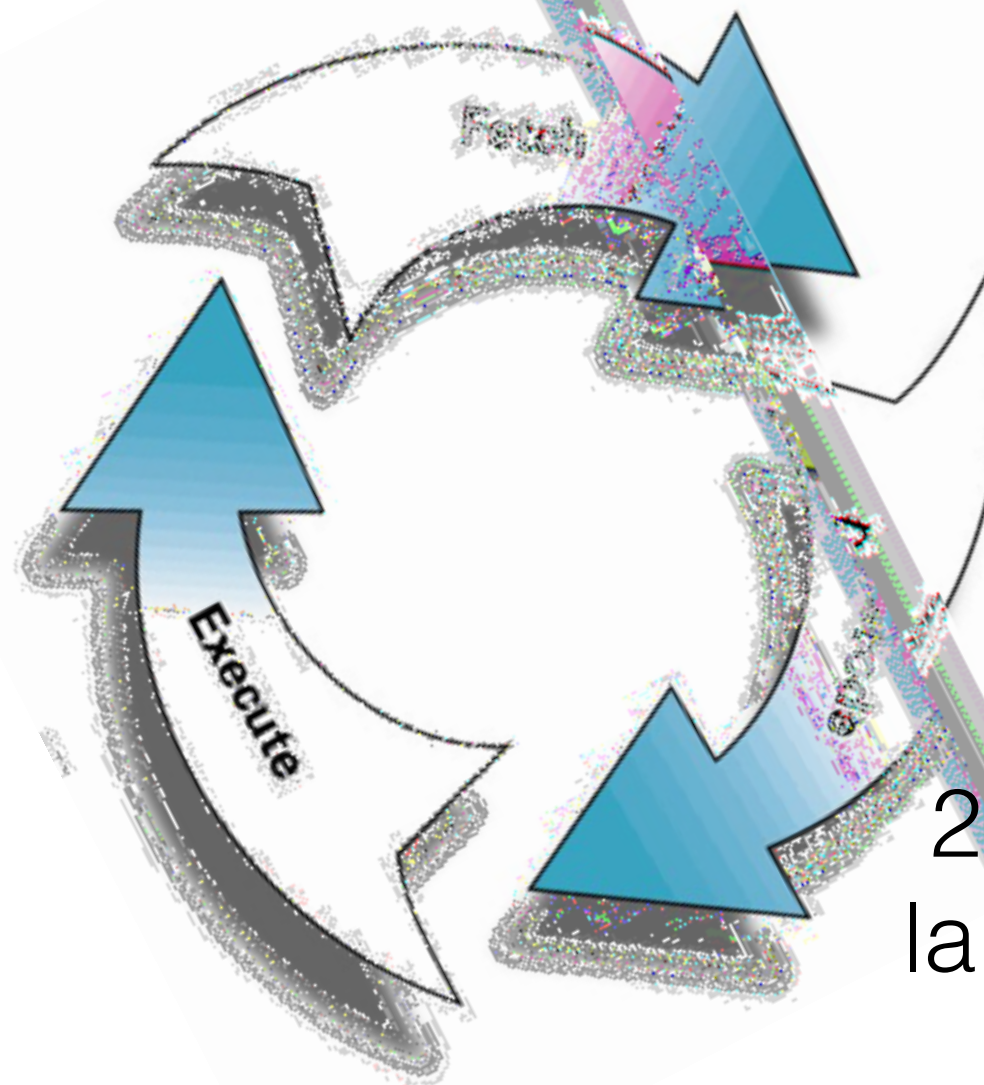
- Con PC=0x123, copiar el contenido de la dirección 0xFF0 en el registro AC



0x123

Opcode				Address											
0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0
Bit 15				Bit 12		Bit 11								Bit 0	

1. La UC lee el contenido de la dirección 0x123, lo escribe en el registro IR, e incrementa el PC a 0x124



3. La UC ejecuta "LOAD 0xFF0"

2. La UC decodifica la instrucción 0x1FF0

Instrucciones de Salto

- Nos permiten alterar la secuencia normal de ejecución del programa
 - IF, WHILE, FOR, etc.
 - Llamadas a procedimientos
- Se dividen de acuerdo a su ejecución
 - Condicionales (debe cumplirse alguna condición)
 - Incondicionales (la ejecución cambia **siempre**)

MARIE: Saltos

Opcode	Instrucción	Efecto
"0000"	JnS X	[X] =PC (copia los menos significativos) y luego $PC := X + 1$
"1000"	Skip Cond	Si Cond=00 y $AC < 0$, entonces $PC := PC + 1$ Si Cond=01 y $AC = 0$, entonces $PC := PC + 1$ Si Cond=10 y $AC > 0$, entonces $PC := PC + 1$
"1001"	Jump X	$PC := X$
"1100"	Jumpi X	$PC := [X]$ (copia los menos significativos)

MARIE

- ¿Cómo podemos escribir este programa usando instrucciones de MARIE?

```
If (AC!=0)
Then AC=0
Endif
```

Opcode	Instrucción	Efecto
"0001"	Load X	Copia el contenido de la dirección X en AC
"0010"	Store X	Copia el contenido de AC en la dirección X
"0011"	Add X	$AC := AC + [X]$ (complemento a 2)
"0100"	Subt X	$AC := AC - [X]$ (complemento a 2)
"1010"	Clear	$AC := 0$

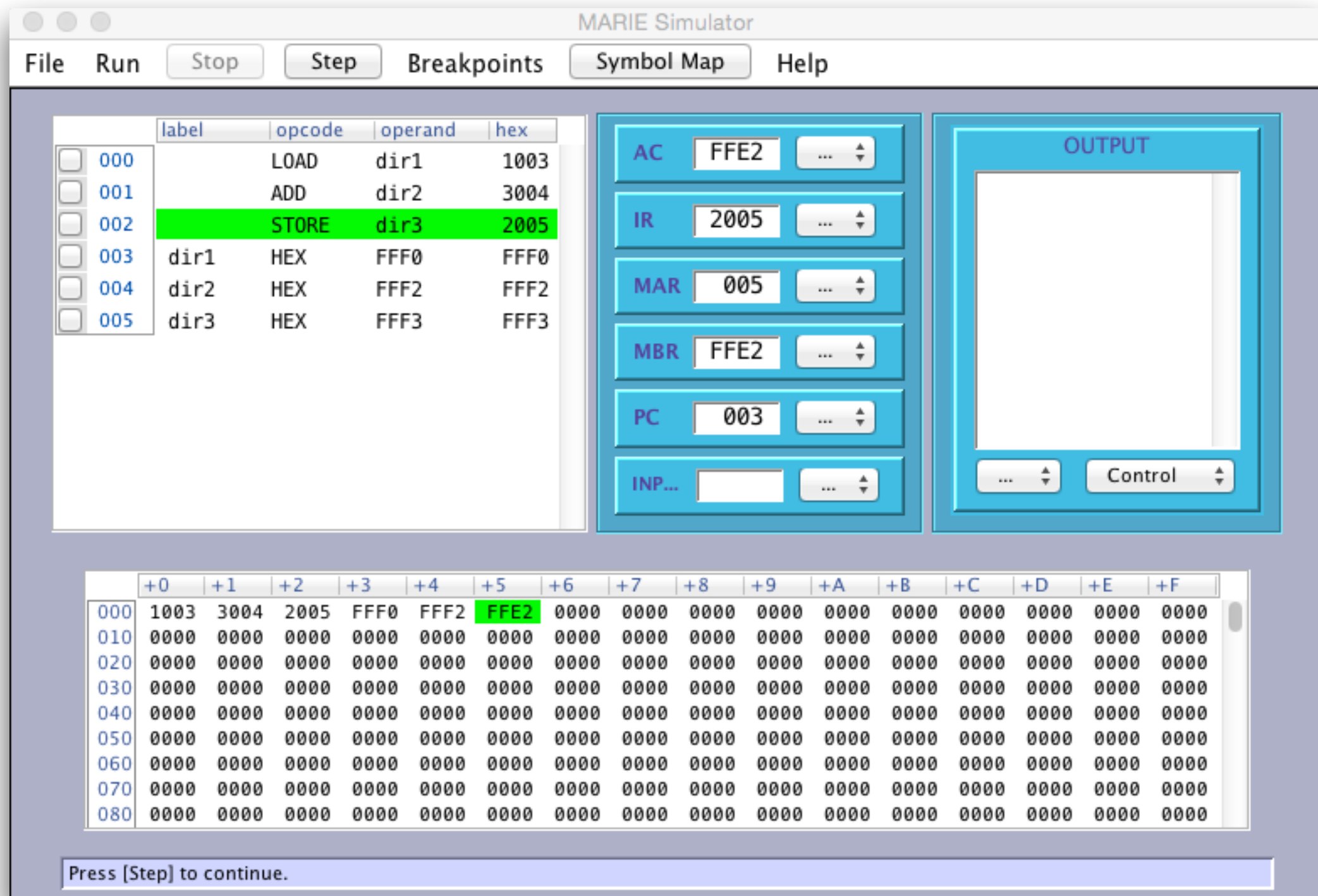
Opcode	Instrucción	Efecto
"0000"	JnS X	$[X] = PC$ (copia los menos significativos) y luego $PC := X + 1$
"1000"	Skip Cond	Si Cond=00 y $AC < 0$, entonces $PC := PC + 1$ Si Cond=01 y $AC = 0$, entonces $PC := PC + 1$ Si Cond=10 y $AC > 0$, entonces $PC := PC + 1$
"1001"	Jump X	$PC := X$
"1100"	Jumpi X	$PC := [X]$ (copia los menos significativos)

If (AC!=0)	1. Skip 01
Then AC=0	2. Clear
Endif	3. ...

MARIE: Set de instrucciones

Opcod	Instrucción	Efecto	
"0001"	Load x	AC := [X]	Memoria
"0010"	Store x	[X] := AC	
"0011"	Add x	AC := AC + [X]	Aritméticas
"0100"	Subt x	AC := AC - [X]	
"1010"	clear	AC:=0	
"1011"	Addi x	AC := AC + [[X]]	
"0000"	JnS x	[X] =PC (copia los menos significativos) y luego PC:=X+1	Control de Ejecución
"1000"	Skip Cond	Si Cond=00 y AC<0, entonces PC:=PC+1 Si Cond=01 y AC=0, entonces PC:=PC+1 Si Cond=10 y AC>0, entonces PC:=PC+1	
"1001"	Jump x	PC:=X	
"1100"	Jumpi x	PC:=[X] (copia los menos significativos)	

MARIE Simulator



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

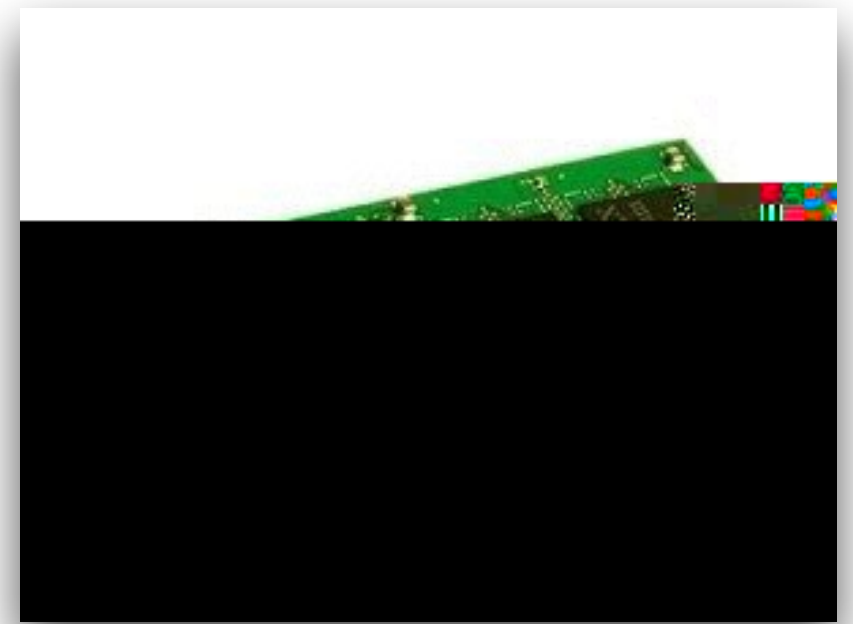


- **Z**ero
- **N**egative
- **C**arry
- o**V**erflow

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.



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

4 bits	6 bits	6 bits	16 bits	16 bits
cod. op.	destino	fuelle	constante destino (opcional)	constante fuente (opcional)

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

0. Modifica los flags según el resultado de $d - f$.
 1. $d \leftarrow d + f$ (suma binaria).
 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	100rrr	Rrrr
Indirecto registro	110rrr	[Rrrr]
Indexado	111rrr	[Rrrr + c16]

c16 es una constante de 16 bits.

Rrrr 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

1 bit	6 bits	6 bits	16 bits
cod. op.	destino	000000	constante destino (opcional)

operacin	cod. op.	efecto	modifica flags
NEG d	1000	$d \leftarrow$ el inverso aditivo de d	S
NOT d	1001	$d \leftarrow$ not d (bit a bit)	S (*)

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	fuelle	constante fuente (opcional)

operacin	cod. op.	efecto	modifica flags
JMP f	1010	$PC \leftarrow f$	no
CALL f	1011	$[SP] \leftarrow PC$, $SP \leftarrow SP - 1$, $PC \leftarrow f$	no

Tipo III: Sin operandos

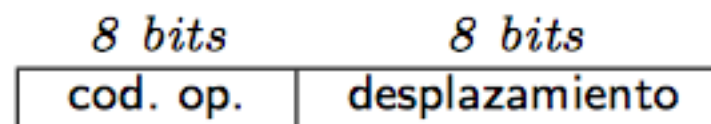
<i>4 bits</i>	<i>6 bits</i>	<i>6 bits</i>
cod. op.	000000	000000

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

Ensamblador (Assembler)

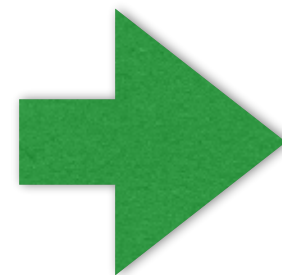
- **Lenguaje** ensamblador \neq **Programa** ensamblador
- **Lenguaje** ensamblador:
 - Lenguaje en el que escribimos programas para Orga1
 - Textual (no confundir con código de máquina)
 - Permite usar etiquetas

Ensamblador (Assembler)

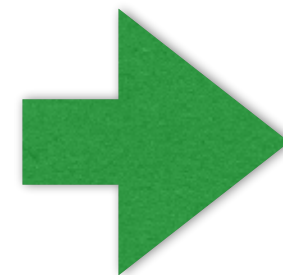
- **Lenguaje** ensamblador != **Programa** ensamblador
- **Programa** ensamblador:
 - Traduce programas en lenguaje ensamblador (texto) a código máquina (ceros y unos)
 - También calcula el valor de etiquetas y desplazamiento

Ensamblador

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



Ensamblador



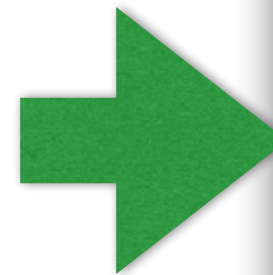
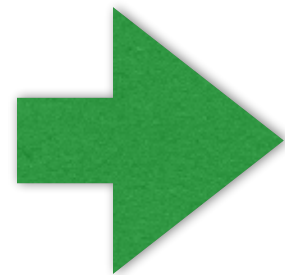
Programa en Lenguaje
Ensamblador

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

1. MOV R0, 0x0010

2. MOV R1, 0x0001

3. ADD R0, R1

MOV

0001 100000 000000

0000 0000 0001 0000

destino
R0

fuelle
cte

Constante
0x0010

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

MOV

destino
R1

fuentes
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

ADD

destino
R0

fuentes
R1

Cargador

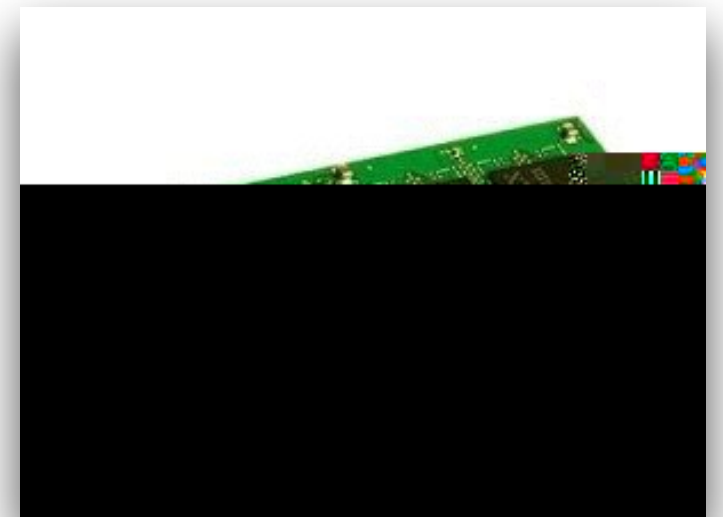
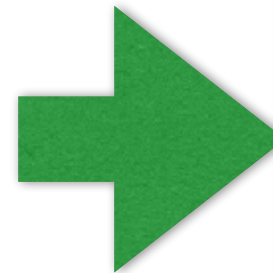
0001 100000 000000

0000 0000 0001 0000

0001 100001 000000

0000 0000 0000 0001

0010 100000 100001



Dirección de Carga: 0x0010

Cargador

0x000F

...

0x0010

0001 100000 000000

0x0011

0000 0000 0001 0000

0x0012

0001 100001 000000

0x0013

0000 0000 0000 0001

0x0014

0010 100000 100001

0x0015

...

MOV R0, 0x0010

MOV R1, 0x0001

ADD R0, R1



Etiquetas

Inicio: MOV R0, 0x0010

MOV R1, 0x0001

ADD R0, R1

JMP **Inicio**

Etiquetas

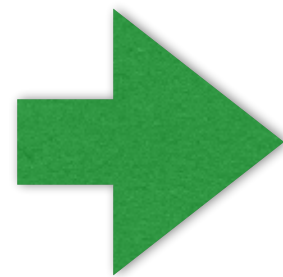
Inicio: MOV R0, 0x0010

MOV R1, 0x0001

ADD R0, R1

JMP **Inicio**

+ Dirección de Carga
del Programa (0x0010)



Ensamblador

Etiquetas

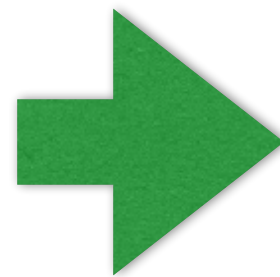
0x0010: MOV R0, 0x0010

0x0012: MOV R1, 0x0001

0x0014: ADD R0, R1

0x0015: JMP **Inicio**

+ Dirección de Carga
del Programa (0x0010)



Etiqueta	Valor
Inicio	0x0010

Etiquetas

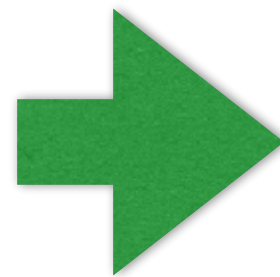
0x0010: MOV R0, 0x0010

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

fuelle
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 0x0F01?

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 0x0F01?

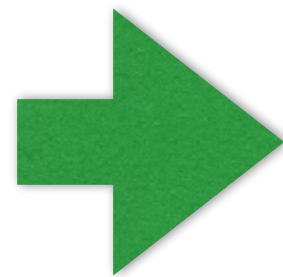
Rta: La codificación
del Salto Absoluto
es distante

Salto Relativos

Inicio: MOV R0, 0x0010

MOV R1, 0x0001

ADD R0, R1



Ensamblador

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

+ Dirección de Carga
del Programa (0x0010)

Salto Relativos

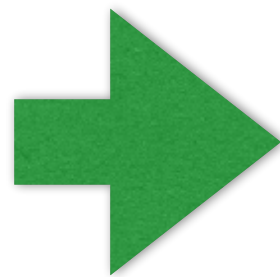
0x0010: MOV R0, 0x0010

0x0012: MOV R1, 0x00001

0x0014: ADD R0, R1

0x0015: JE **Inicio**

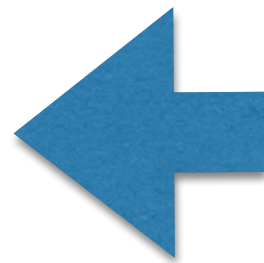
+ Dirección de Carga
del Programa (0x0010)



Etiqueta	Valor
Inicio	0x0010

Salto Relativos

0x0010: MOV R0, 0x0010

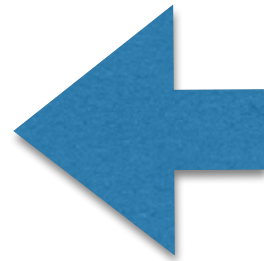


¿Cuánto debe valer el PC para que se ejecute esta instrucción?

0x0012: MOV R1, 0x0001

0x0014: ADD R0, R1

0x0015: JE **Inicio**

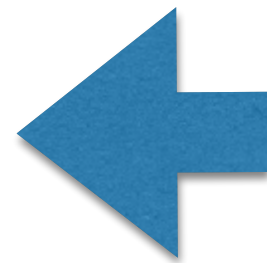


¿Cuál es el valor del PC luego de ejecutarse esta instrucción?

+ Dirección de Carga del Programa (0x0010)

Salto Relativos

0x0010: MOV R0, 0x0010



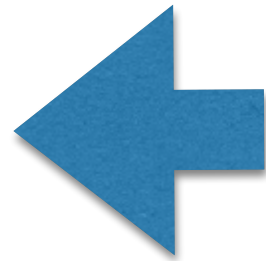
¿Cuánto debe valer el PC para que se ejecute esta instrucción?

0x0012: MOV R1, 0x0001

Rta: PC=0x0010

0x0014: ADD R0, R1

0x0015: JE **Inicio**



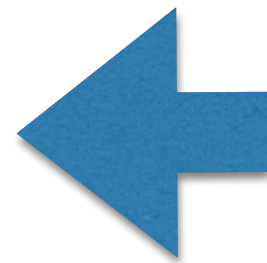
¿Cuál es el valor del PC luego de ejecutarse esta instrucción?

Rta: PC=0x0016

+ Dirección de Carga del Programa (0x0010)

Salto Relativos

0x0010: MOV R0, 0x0010



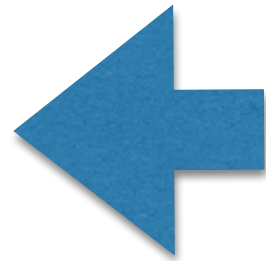
¿Cuánto debe valer el PC para que se ejecute esta instrucción?

0x0012: MOV R1, 0x0001

Rta: PC=0x0010

0x0014: ADD R0, R1

0x0015: JE -6



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

Rta: PC=0x0016

+ Dirección de Carga del Programa (0x0010)

Salto 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)

Salto 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)

Salto 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)



Salto



- 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

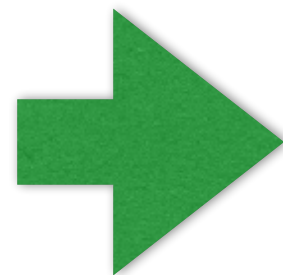
Ensamblador - Datos

V1: DW 0x001F

V2: DW 0x0FF0

ADD R0, [V1]

ADD R0, [V2]



Ensamblador

+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

V1: DW 0x001F

V2: DW 0x0FF0

ADD R0, [V1]

ADD R0, [V2]

+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

0x0010: DW 0x001F

0x0011: DW 0x0FF0

0x0012: ADD R0, [V1]

0x0014: ADD R0, [V2]

Etiqueta

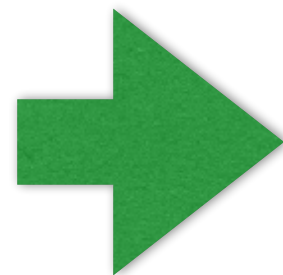
Valor

V1

0x0010

V2

0x0011



+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

0x0010: DW 0x001F

0x0011: DW 0x0FF0

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 0x0FF0

0000 1111 1111 0000

Dato 0x0FF0

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 0x0FF0

0000 1111 1111 0000

0x0012: ADD R0, [0x0010]

0010 100000 001000

0000 0000 0001 0000

0x0014: ADD R0, [0x0011]

ADD

destino
R0

fuelle
dirección

Dirección
0x0010

+ Dirección de Carga
del Programa (0x0010)

Ensamblador - Datos

0x0010: DW 0x001F

0000 0000 0001 1111

0x0011: DW 0x0FF0

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

fuelle
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
- Máquina MARIE
- Máquina ORGA1
- Lenguaje Ensamblador ORGA1
- **Lenguaje** Ensamblador vs. **Programa** Ensamblador



Bibliografía

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