

# Arquitectura de las computadoras

## **Electrónica IV**

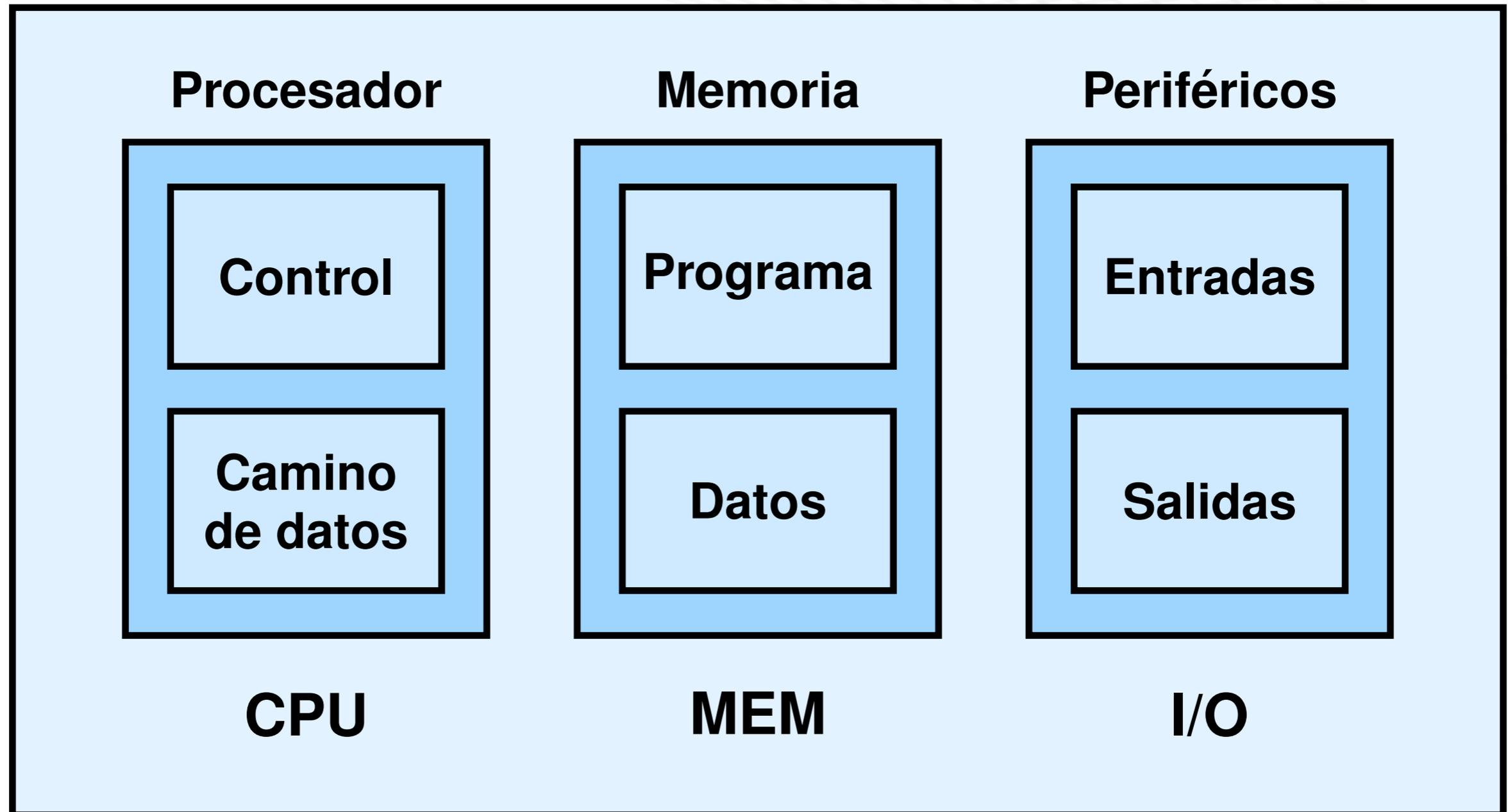
**Mg.Ing. Esteban Volentini** ([evolentini@herrera.unt.edu.ar](mailto:evolentini@herrera.unt.edu.ar))

<https://facetvirtual.facet.unt.edu.ar/course/view.php?id=165>

# Arquitectura de una computadora

---

## Computadora



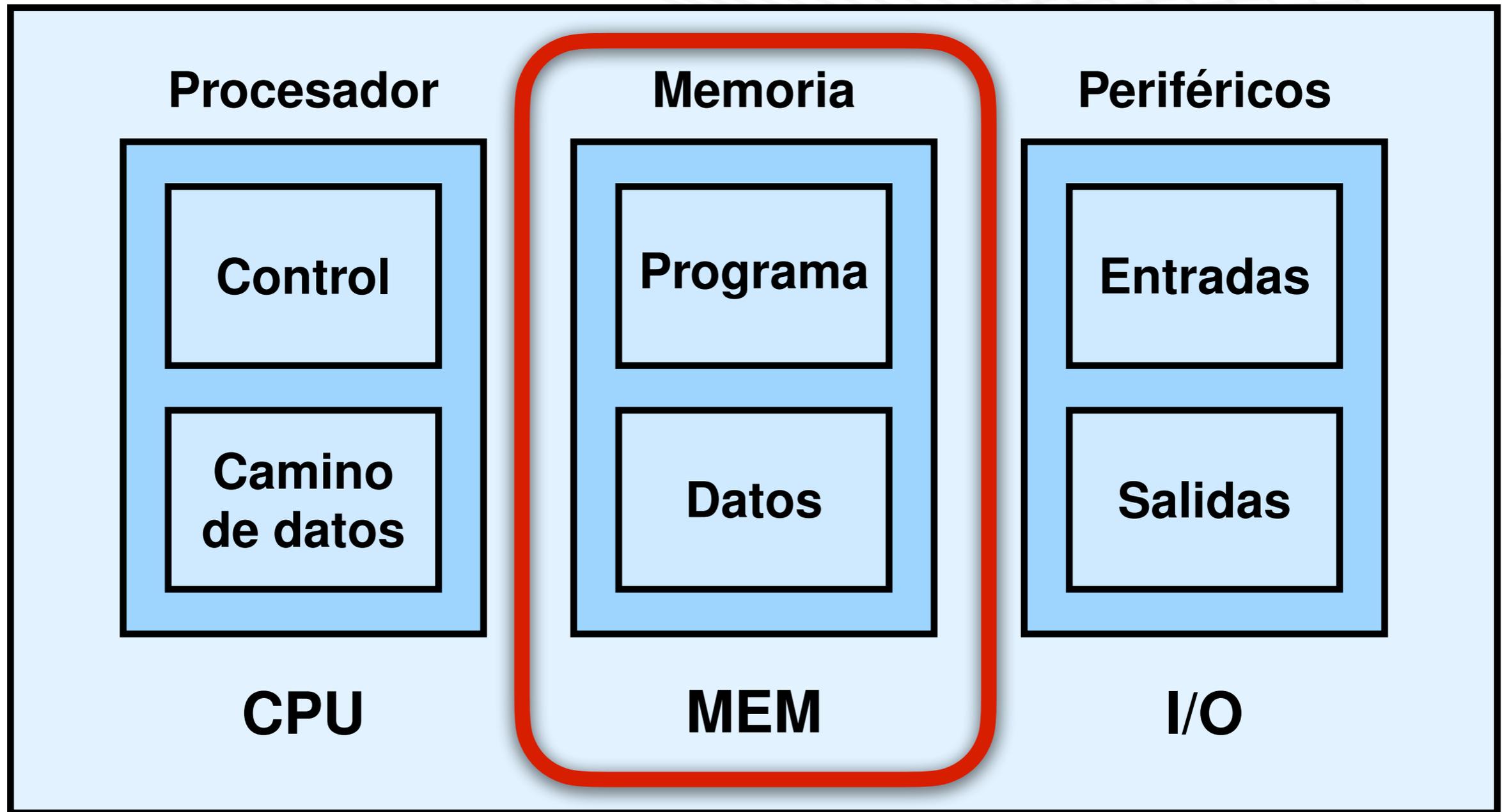
# Responsabilidades

---

- ▶ La memoria almacena las instrucciones y los datos
- ▶ El procesador ejecuta los programas para transformar los datos
- ▶ Ejemplo de datos
  - ▶ La posición A contiene el dato X
  - ▶ La posición B contiene el dato Y
- ▶ Ejemplo de programa
  - ▶ Tome el valor X almacenado en la posición A
  - ▶ Súmele el valor Y almacenado en la posición B
  - ▶ Guarde el resultado en la posición C

# La memoria

## Computadora



# Memoria principal y secundaria

---

- ▶ La memoria principal esta gestionada en forma directa por el procesador
  - ▶ El tiempo de acceso es el mismo para cualquier celda, independiente de la secuencia previa de acceso (Random)
  - ▶ Se utiliza para almacenar las instrucciones y los datos
- ▶ La memoria secundaria se gestiona como un periférico de entrada/salida
  - ▶ El tiempo de acceso puede variar en función de la secuencia previa de lecturas y/o escrituras
  - ▶ Se utiliza para almacenar bloques de datos que se transfieren desde y hacia la memoria principal mientras no son utilizados

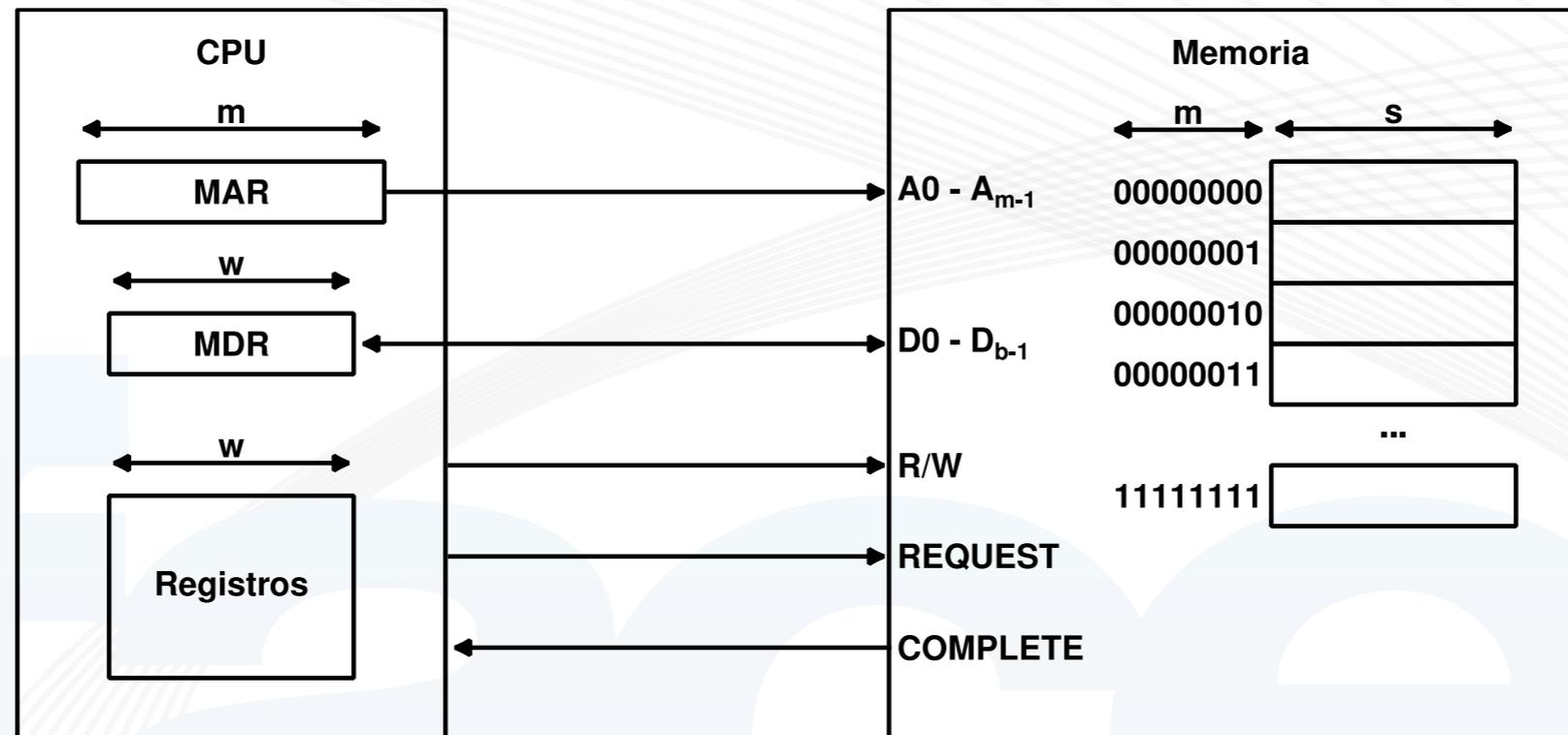
# La memoria principal

- ▶ Funciona como una gran tabla unidimensional
- ▶ La dirección de memoria es el número de fila, es decir el índice en la tabla
- ▶ Cada fila puede almacenar un valor de tamaño fijo (n bits)
- ▶ La celda básica utilizada para almacenar cada bit determina las características de la memoria

0	n bits
1	n bits
2	n bits
3	n bits
4	n bits
5	n bits
6	n bits

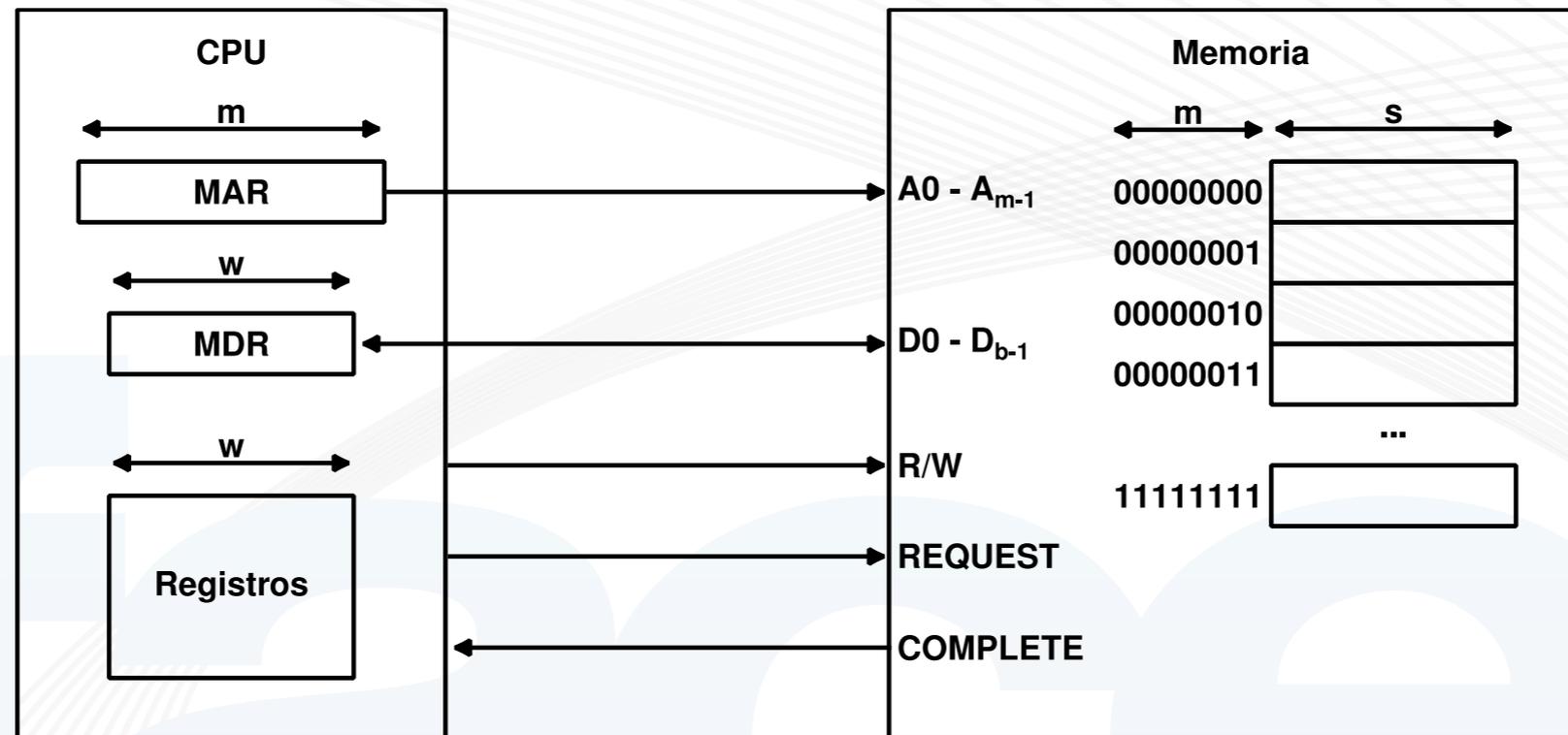
...

# Conexión de la memoria



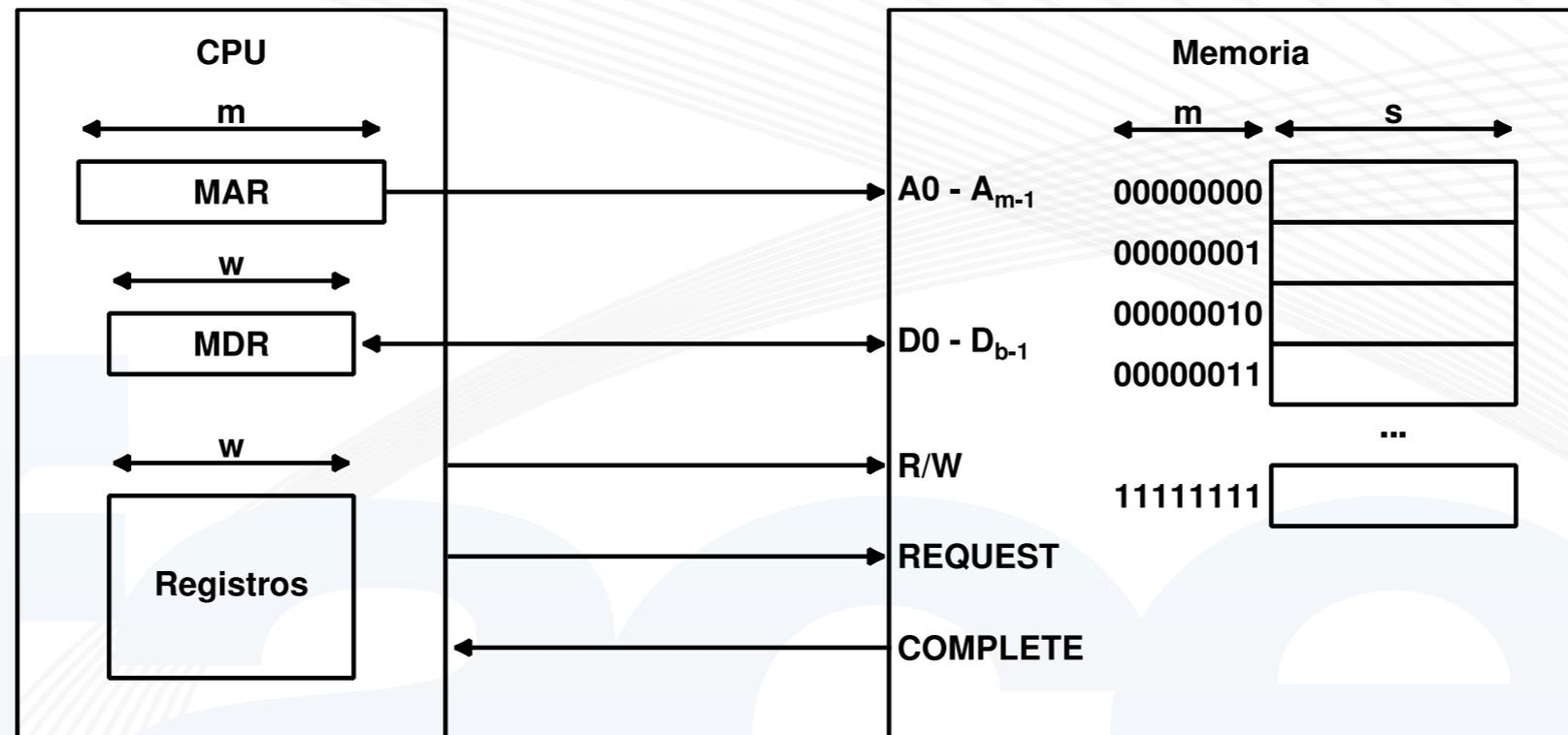
- ▶ Memory Address Register (MAR)
  - ▶ Contiene la dirección de los datos a leer o escribir en memoria
  - ▶  $2^m$  es la dimensión del “espacio de direcciones”
- ▶ Memory Data Register (MDR)
  - ▶ Contiene los datos a escribir o que se leen de memoria
  - ▶  $w$  es el “ancho de palabra” del procesador
  - ▶  $s$  es el "ancho" de una dirección de memoria

# Lectura de la memoria



- ▶ El procesador almacena en MAR la dirección, fija el valor adecuado a la línea R/W y, finalmente, activa la línea REQUEST
- ▶ La memoria recupera el dato de la dirección recibida por las líneas  $A_x$ , lo envía las líneas  $D_x$  y, finalmente, activa COMPLETE
- ▶ El procesador almacena el dato recibido en el registro MDR

# Escritura en la memoria



- ▶ El procesador almacena en MAR la dirección, en MDR el dato a escribir, fija el valor adecuado a la línea R/W y, finalmente, activa la línea REQUEST
- ▶ La memoria almacena el dato recibido por las líneas  $D_x$  en la dirección recibida por las líneas  $A_x$  y, finalmente, activa COMPLETE

# Propiedades de memoria

Simbolo	Definición	CPU08	8086	CortexM4
w	Ancho de palabra del CPU	8 bits	16 bits	32 bits
m	Bits en una dirección de memoria	16 bits	20 bits	32 bits
s	Bits en la menor unidad direccionable	8 bits	8 bits	8 bits
b	Ancho del Data Bus	8 bits	16 bits	32 bits
$2^m$	Capacidad de palabras en memoria	$2^{16}$ words	$2^{20}$ words	$2^{32}$ bytes
$2^m * s$	Espacio de Memoria en bits	$2^{16} \times 8$ bits	$2^{20} \times 8$ bits	$2^{32} \times 8$ bits

**¡El ancho de palabra de CPU no siempre es igual al ancho del data bus!**

# Espacio de memoria

---

- ▶ Es la máxima cantidad de memoria que se puede conectar a un procesador
- ▶ Esta dado por su capacidad de direccionamiento
- ▶ En general no todo se usa
- ▶ La ubicación de las partes usadas forman el Mapa de Memoria

# Direccionamiento a palabra

---

- ▶ En general el ancho de la memoria debería ser igual que el ancho de la palabra del procesador
- ▶ Pero no se pueden leer o escribir datos más chicos que una fila de memoria
- ▶ Sin embargo en los procesadores más modernos los programas todavía utilizan muchos datos pequeños (de 8 bits o incluso de menos)
- ▶ Se desperdicia mucha memoria al almacenar datos pequeños en filas de memoria con muchos bits

# Direccionamiento a bytes

---

- ▶ En consecuencia lo mas común es mantener la memoria organizada en bytes
- ▶ De esta forma se pueden operar con datos de un byte como mínimo
- ▶ Para datos de mayor tamaño se deberían realizar varias lecturas sucesivas
- ▶ Esta opción tiene problemas de velocidad

# Direccionamiento a bytes

---

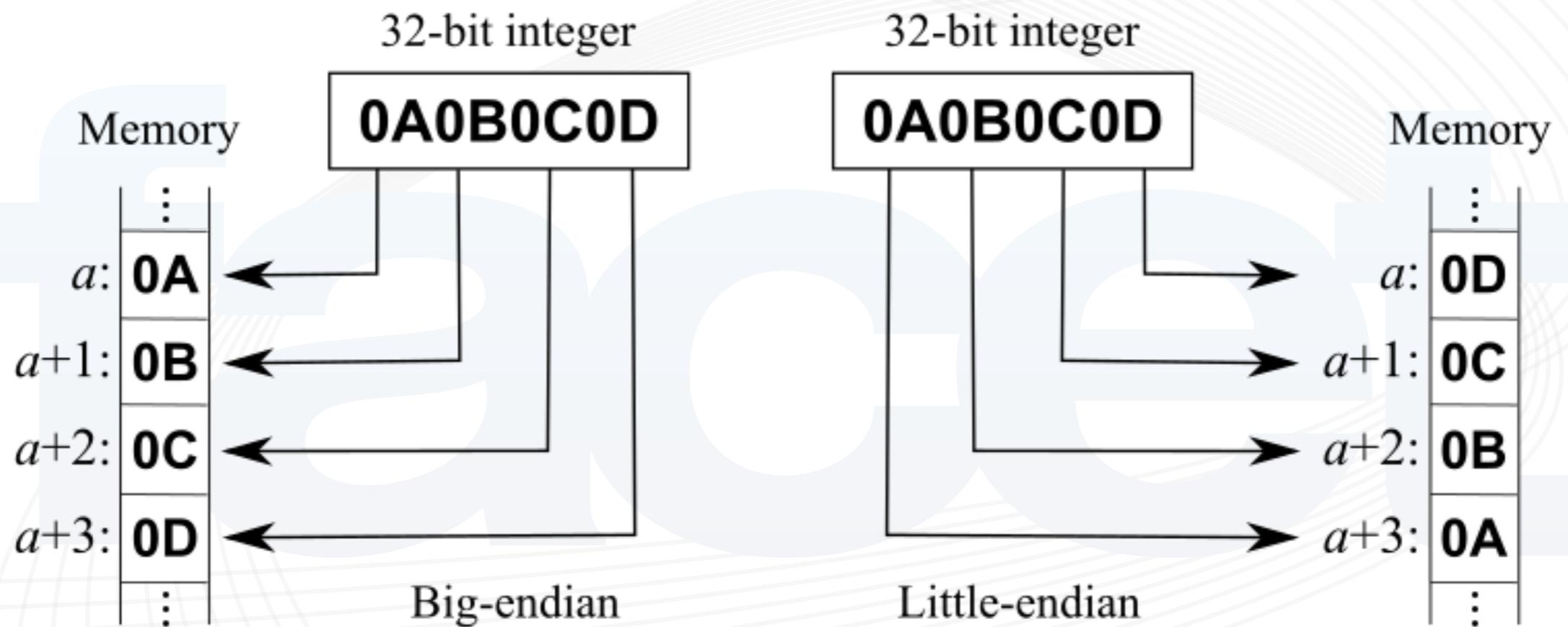
- ▶ Para mejorar la performance se lee una palabra completa en cada acceso
- ▶ Cada byte de la palabra tiene una dirección y se puede individualizar
- ▶ Cada palabra se representa por la dirección del primer byte que la compone
- ▶ Cada palabra ocupa más de una dirección de memoria

# Big-endian y Little-endian

---

- ▶ En una memoria organizada por bytes los números que ocupan más de un byte se pueden almacenar de dos formas diferentes:
- ▶ **Big-endian**
  - ▶ El Byte más significativo en la dirección baja
  - ▶ Es más fácil para leer (humanos)
- ▶ **Little-endian**
  - ▶ El Byte menos significativo en la dirección baja
  - ▶ Es más fácil para operar (maquinas)

# Big-endian y Little-endian



# Memorias de lectura y escritura

---

- ▶ Son memorias donde la lectura y la escritura demoran la misma cantidad de tiempo
- ▶ **SRAM: Static Random Access Memory**
  - ▶ La celda de almacenamiento es un flip-flop
  - ▶ El contenido se mantiene indefinidamente con  $V_{cc}$
  - ▶ Baja densidad, alto consumo, cara, rápida
- ▶ **DRAM: Dynamic Random Access Memory**
  - ▶ La celda básica de almacenamiento es un capacitor
  - ▶ Necesita ser “refrescada” regularmente
  - ▶ Alta densidad, bajo consumo, barato, lento

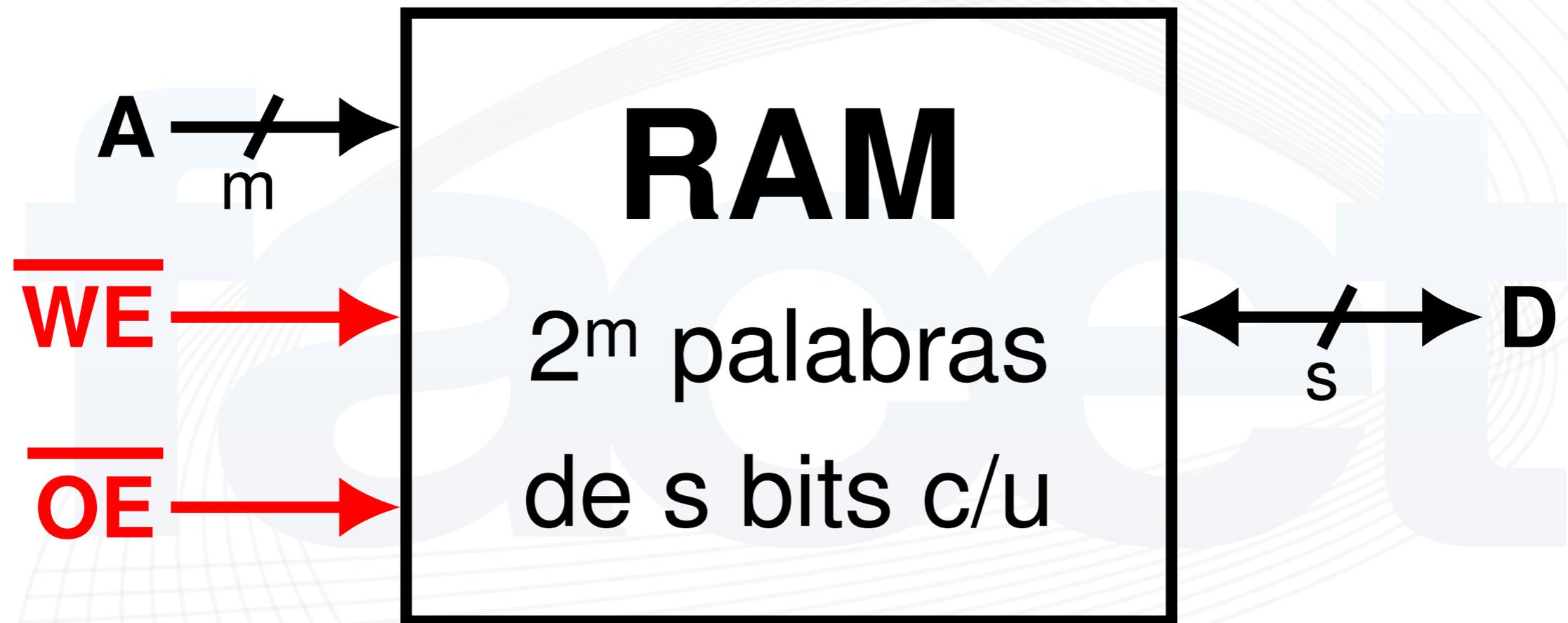
# Memorias de solo lectura

---

- ▶ Son memorias donde no se puede escribir, o la escritura requiere un proceso diferente que demora más tiempo que la lectura
- ▶ ROM: Read Only Memory
  - ▶ La celda de almacenamiento es una conexión eléctrica
  - ▶ El valor almacenado se define al fabricar el chip de memoria
- ▶ PROM: Programmable Read-Only Memory
  - ▶ La celda básica de almacenamiento es un diodo
  - ▶ Se puede programar una sola vez quemando ciertos diodos
- ▶ EPROM: Erasable Programmable Read-Only Memory
  - ▶ La celda básica es un “pozo de energía potencia”
  - ▶ Se puede programar eléctricamente y borrar con luz UltraVioleta
- ▶ EEPROM: Electrically Erasable Programmable Read-Only Memory
  - ▶ La celda básica es un “pozo de energía potencia”
  - ▶ Se puede programar y borrar eléctricamente una cantidad limitada de veces

# Diagrama Lógico RAM

---



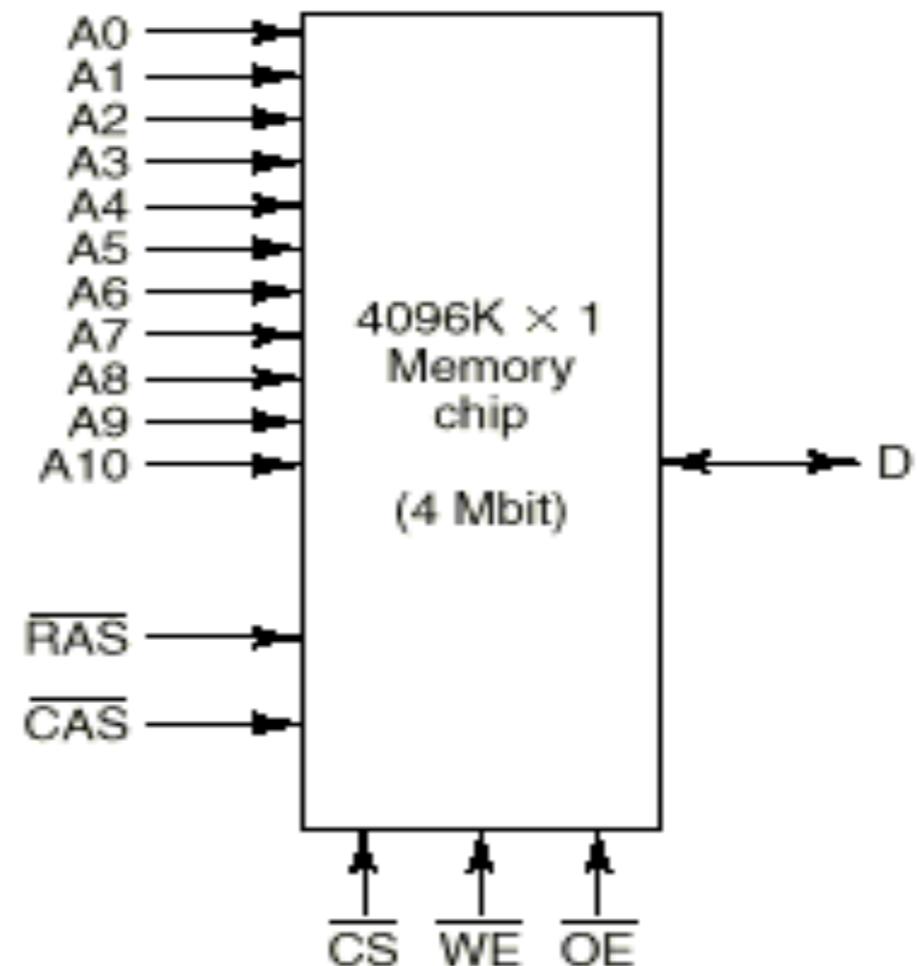
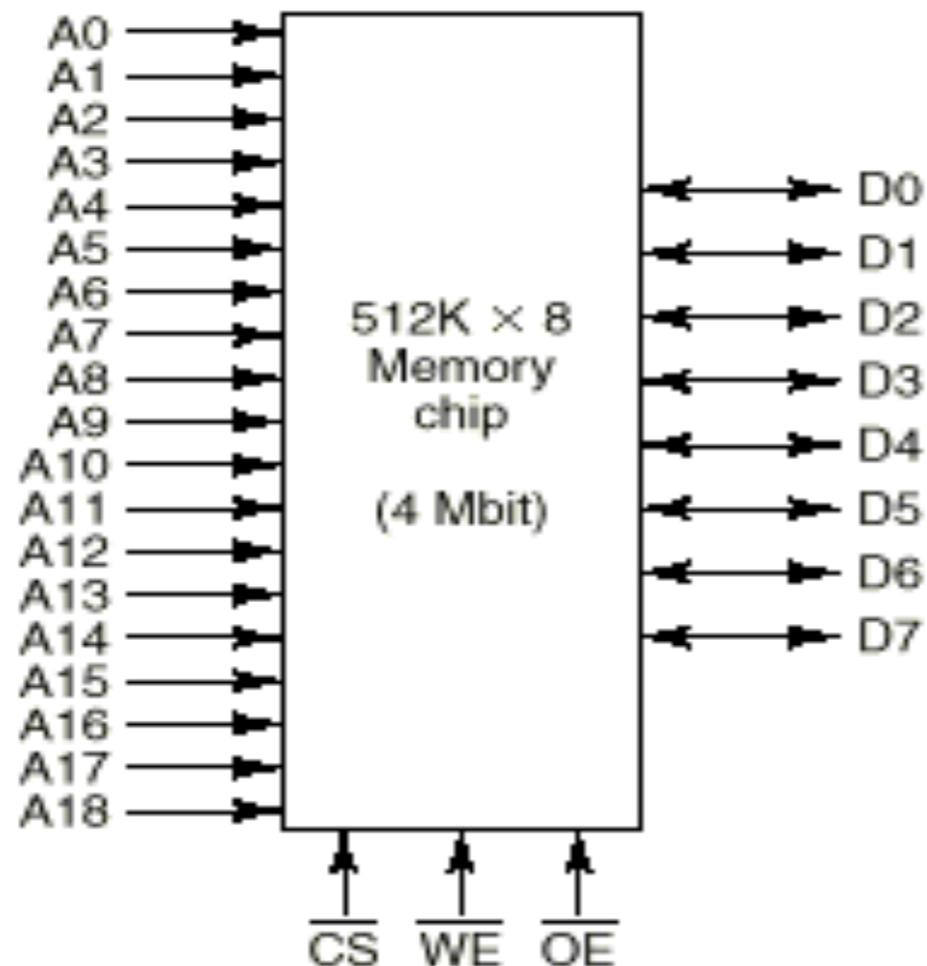
# Diagrama Lógico RAM

---

- ▶ **Señales de control:** Write Enable y Output Enable
  - ▶ Generalmente se activan en “Low” ( $\overline{\text{WE}}$ ,  $\overline{\text{OE}}$ )
  - ▶  $\overline{\text{OE}}$  permite ahorrar terminales, unificando Din y Dout
- ▶ **WRITE:**  $\overline{\text{WE}}$  se activa (0),  $\overline{\text{OE}}$  se desactiva (1)
  - ▶ D funciona como entrada de datos
- ▶ **READ:**  $\overline{\text{WE}}$  se desactiva (1),  $\overline{\text{OE}}$  se activa (0)
  - ▶ D es la salida de datos
- ▶ **INDEFINIDO:** SI tanto  $\overline{\text{WE}}$  como  $\overline{\text{OE}}$  se activan:
  - ▶ Resultado no definido. ¡¡No hacerlo!!

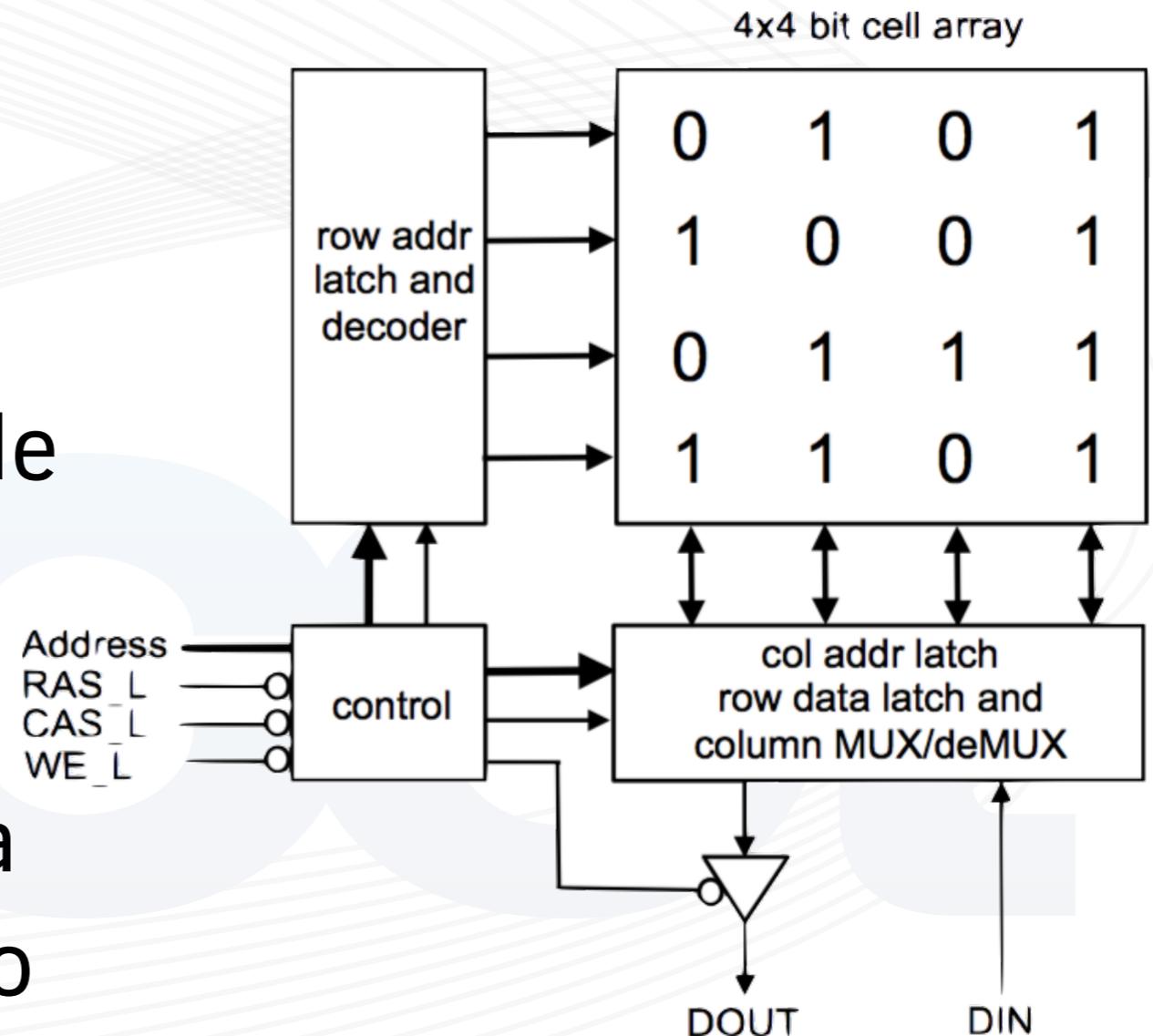
# Memorias SRAM y DRAM

- ▶ ¿Cómo se puede verificar el tamaño en KB de estos chips?
- ▶ Algunas RAM unifican /WE y /OE en un único R/W
- ▶ ¿Para qué traen incorporados CS?



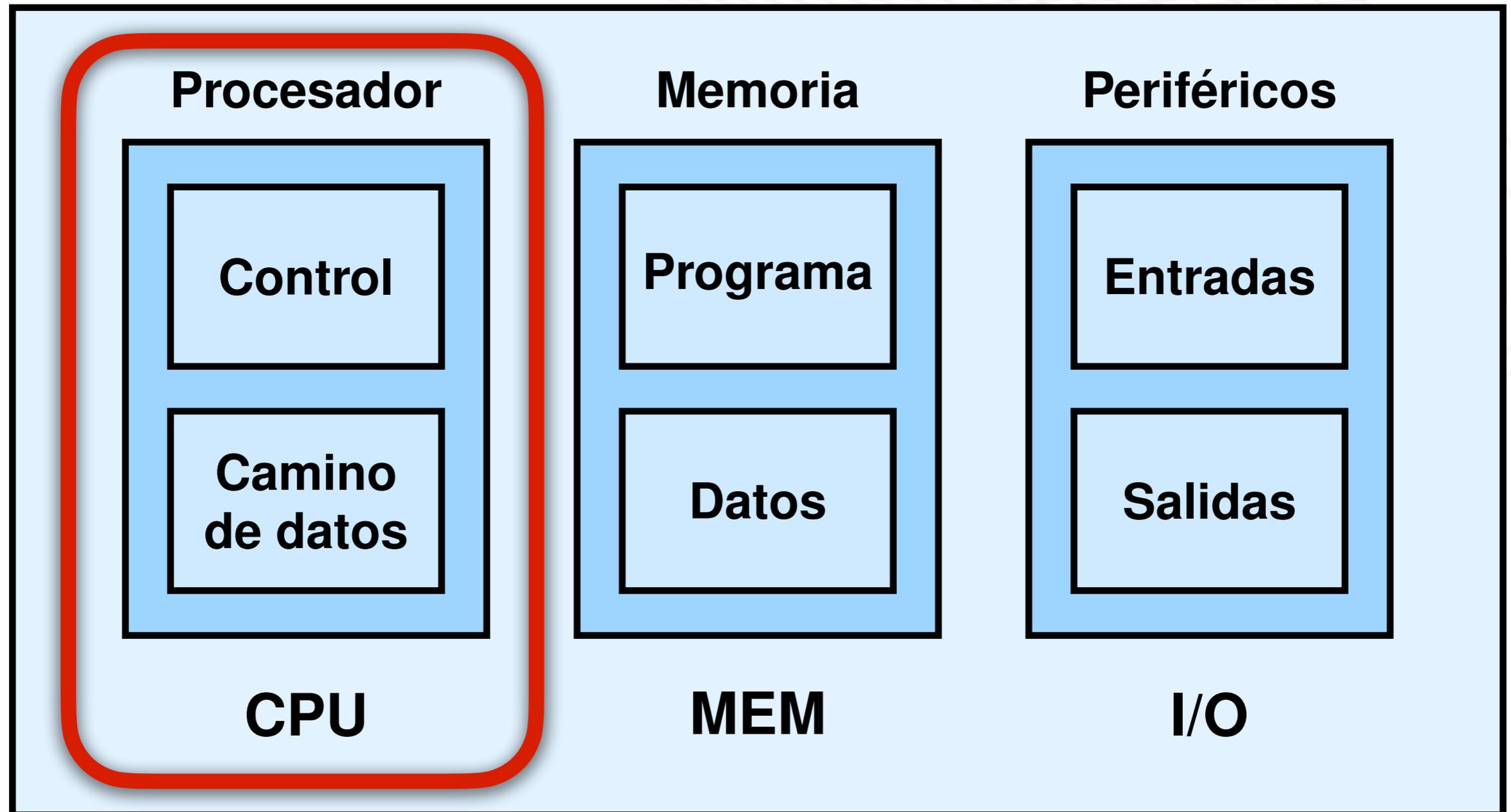
# Direcciones partidas en DRAM

- ▶ Las memorias se construyen en forma de cuadrados
- ▶ Los tiempos de acceso de la celda DRAM permiten partir la dirección
- ▶ Usar esta técnica en una SRAM afectaría el tiempo de acceso total de la memoria



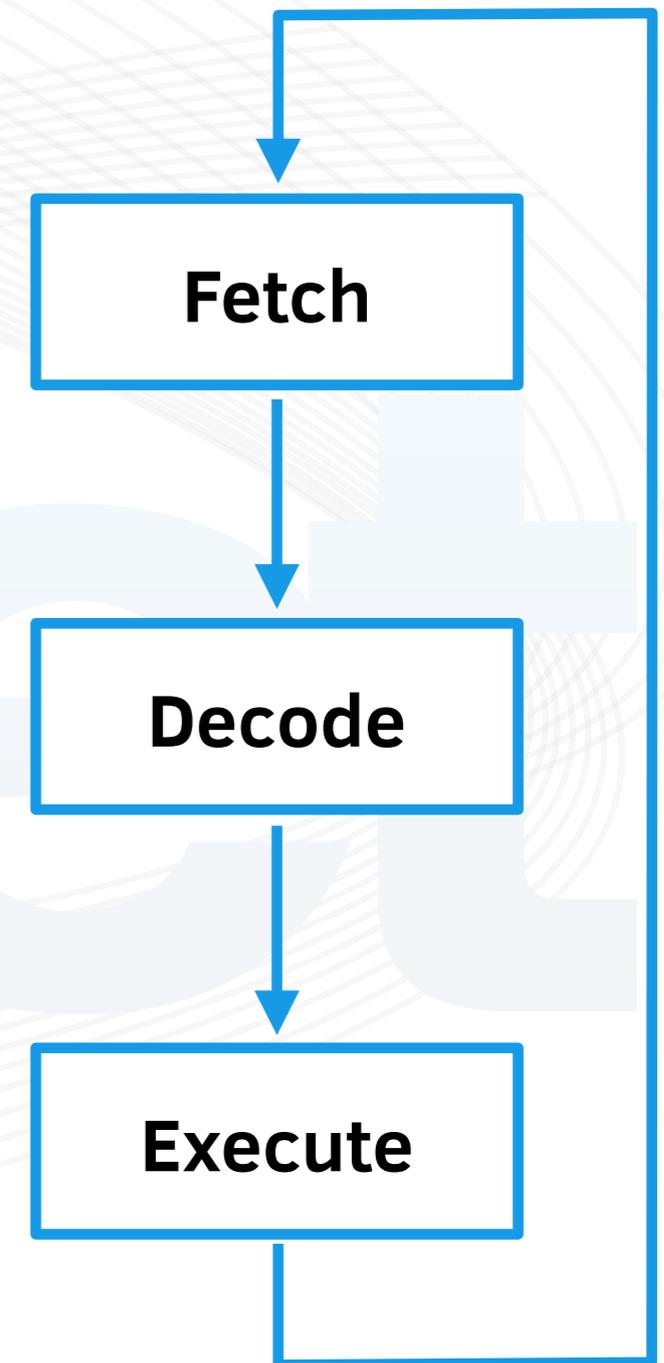
# El procesador

## Computadora



# Ciclo de la instrucción

- ▶ El procesador es una maquina de estados finito que repite permanentemente una secuencia
- ▶ Busca una instrucción en memoria
- ▶ Determina las acciones necesarias para que la instrucción genere el resultado esperado
- ▶ Ejecuta las acciones determinadas en el paso anterior para obtener el resultado esperado
- ▶ Es equivalente a un interprete del ISA



# Componentes del ISA

---

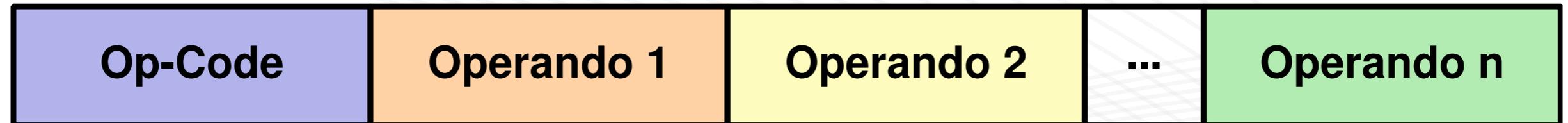
- ▶ **El ISA es todo lo que necesita conocer el programador del CPU**
- ▶ Celdas de almacenamiento
  - ▶ Registros de propósito general y especial en el CPU
  - ▶ Muchas celdas de propósito general de igual tamaño en memoria
  - ▶ Almacenamiento relacionado con dispositivos de Entrada/Salida
- ▶ Formato de las instrucciones
  - ▶ Tamaño y significado de los diferentes campos de las instrucciones
  - ▶ Modos de direccionamiento disponibles para cada instrucción
- ▶ El set de instrucciones de la máquina
  - ▶ Es el repertorio completo de las operaciones de la máquina.
  - ▶ Emplea celdas de almacenamiento, formatos de las instrucciones y especifica los resultados del ciclo de cada instrucción

# Tres clases de instrucciones

---

- ▶ **Movimiento de datos**
  - ▶ Tienen siempre una fuente y un destino
  - ▶ Load: la fuente es memoria y el destino un registro
  - ▶ Store: la fuente es un registro y el destino es memoria
  - ▶ Hay casos con fuentes y destino, ambos en memoria o en registros
- ▶ **Instrucciones Aritméticas y Lógicas (procesamiento)**
  - ▶ Procesar uno o más operandos fuentes y guardar el resultado
  - ▶ Add, Sub, Shift, etc.
- ▶ **Control de flujo de instrucciones (saltos)**
  - ▶ Alterar el flujo normal de control en lugar de ejecutar la siguiente instrucción de la dirección contigua
  - ▶ Saltos condicionales o incondicionales

# Formato de la instrucción



- ▶ La instrucción se divide en campos:
- ▶ Op-Code: Qué hace la instrucción
  - ▶ Con n bits se pueden especificar hasta  $2^n$  operaciones distintas
- ▶ Operandos: Con qué datos se debe operar
  - ▶ En general desde 0 a 3 operandos
  - ▶ El dato puede estar en el campo de la misma instrucción, en memoria o en un registro
  - ▶ Modo de direccionamiento, la forma en que se obtiene el operando a partir de la información suministrada en la instrucción
- ▶ Puede haber más de un formato, dependiendo del Op-Code

# Que se especifica en una instrucción

---

- add x3, x2, x1** ▶ ¿Qué operación realizar?
- ▶ Op-code: add, load, branch, etc
- add x3, x2, x1** ▶ ¿Dónde están los operandos?
- ▶ En registros del CPU, en memoria, E/S
  - ▶ Pueden ser parte de la instrucción.
- add x3, x2, x1** ▶ ¿Dónde se guarda el resultado?
- ▶ En un registro, memoria o E/S
- add x3, x2, x1** ▶ ¿Como continua el programa?
- ▶ Lugar de memoria donde buscar la siguiente instrucción
- j lazo**

# Modos de direccionamiento

---

- ▶ Para escribir un programa se requiere un algoritmo que opera sobre los datos
- ▶ Estos datos puede estar organizados en diferentes formas
- ▶ El modo de direccionamiento determina como se obtiene el dato a partir del la información proporcionada en la instrucción
- ▶ Los modos de direccionamiento permiten acceder a las diferentes estructuras de datos de forma eficiente

# Modos de direccionamiento

---

- ▶ **Registro** para variables temporarias muy usadas
  - ▶ El dato está almacenado en un registro de propósito general y en la instrucción especifica un número del registro
  - ▶ Si R2 contiene 3 y R3 contiene 4
  - ▶ **ADD R1, R2, R3 ; R1 ← 7**
- ▶ **Directo** para variables globales
  - ▶ El dato está almacenado en memoria y la instrucción especifica la dirección de memoria como una constante
  - ▶ Si la dirección de memoria 0x1000 contiene el valor 5
  - ▶ **LD R1, (0x1000) ; R1 ← 5**

# Program Status Word (PSW)

---

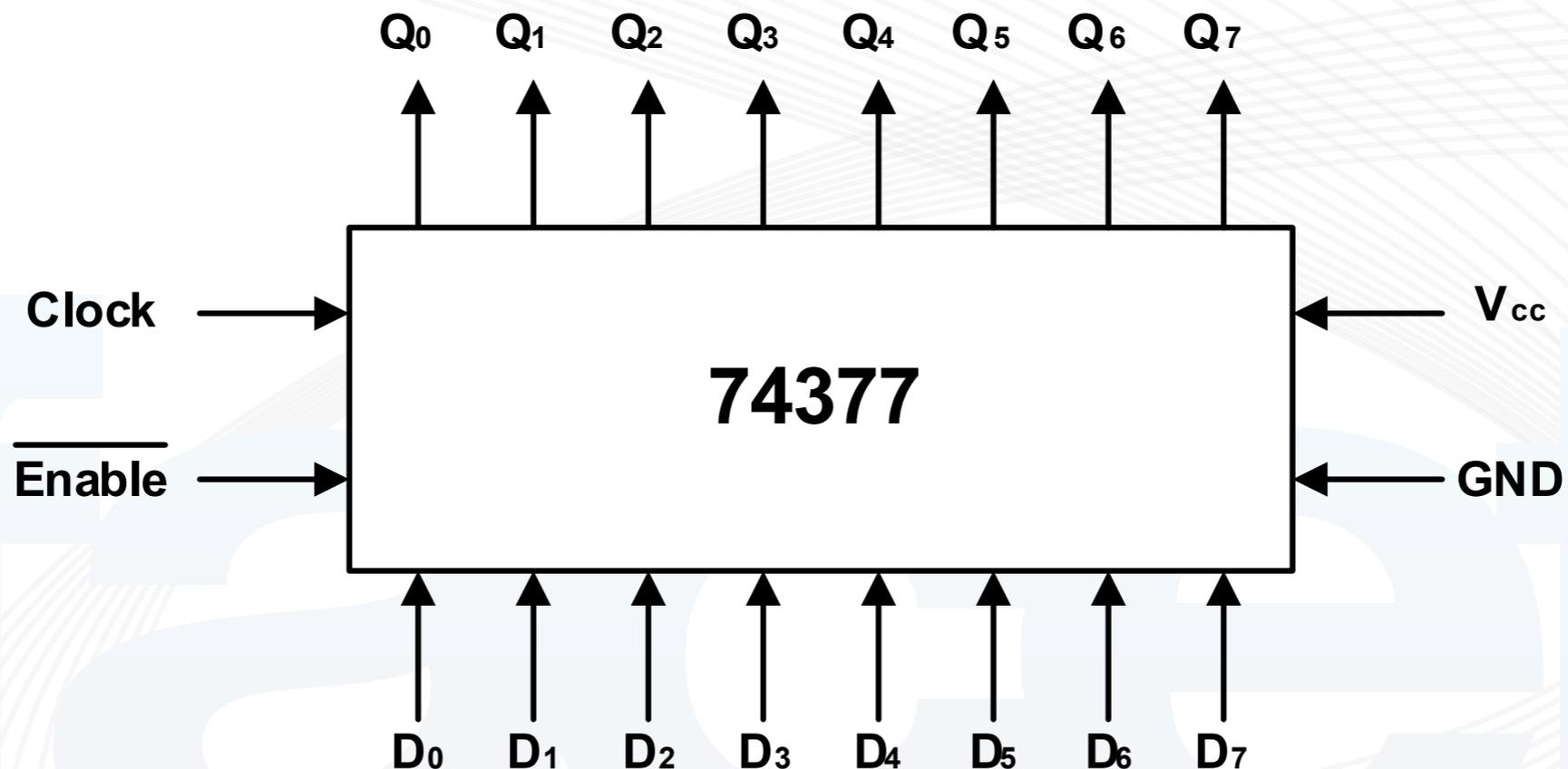
- ▶ Registro de código de condición (CCR)
  - ▶ Compuesto por banderas que dan información sobre el resultado de la última operación
    - ▶ Carry, Negative, Overflow, Zero, ...
  - ▶ Sirven para las instrucciones de salto
    - ▶ BZ (salte si cero)
    - ▶ BN (salte si negativo)
  - ▶ Algunas arquitecturas no utilizan CCR
- ▶ Otra información de estado del CPU

# El procesador por dentro

---

- ▶ Se divide en dos grandes componentes:
  - ▶ El camino de datos, esta formado por los registros, la ALU y las conexiones necesarios para almacenar y transformar los datos
  - ▶ La unidad de control, es una máquina de estados finitos que genera la señales de control para actuar sobre el camino de datos
- ▶ Construiremos un camino de datos desde cero

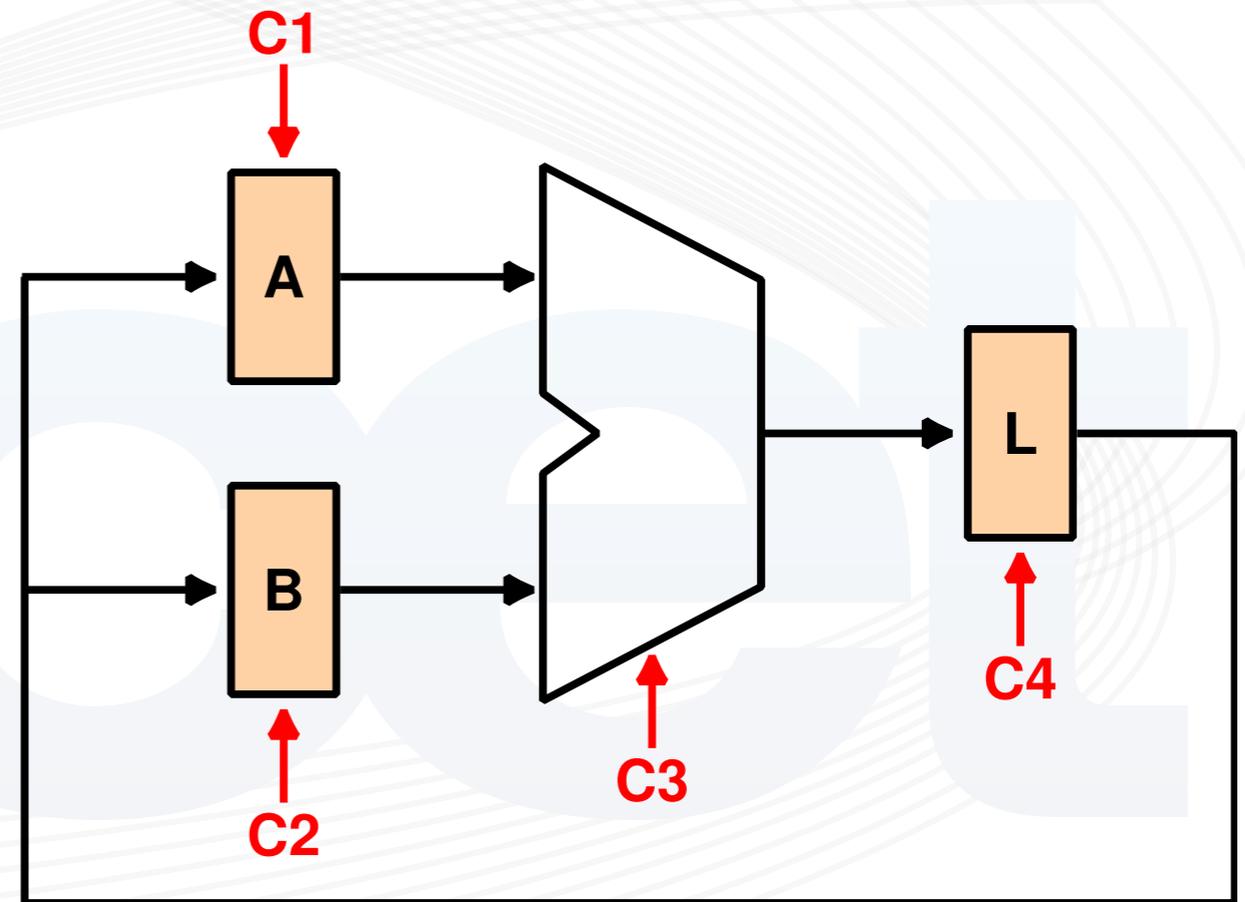
# Presentación de un registro



- ▶ Las salidas  $Q_i$  toman el valor de las entradas  $D_i$  cuando la señal  $\overline{\text{Enable}}$  está activa (en bajo) y se produce un flanco ascendente en la señal Clock

# Un camino de datos básico

- ▶ C1, C2, C3 y C4 son señales de control
- ▶ ¿De que depende la cantidad de bits de C3?
- ▶ ¿Para que esta el registro L?

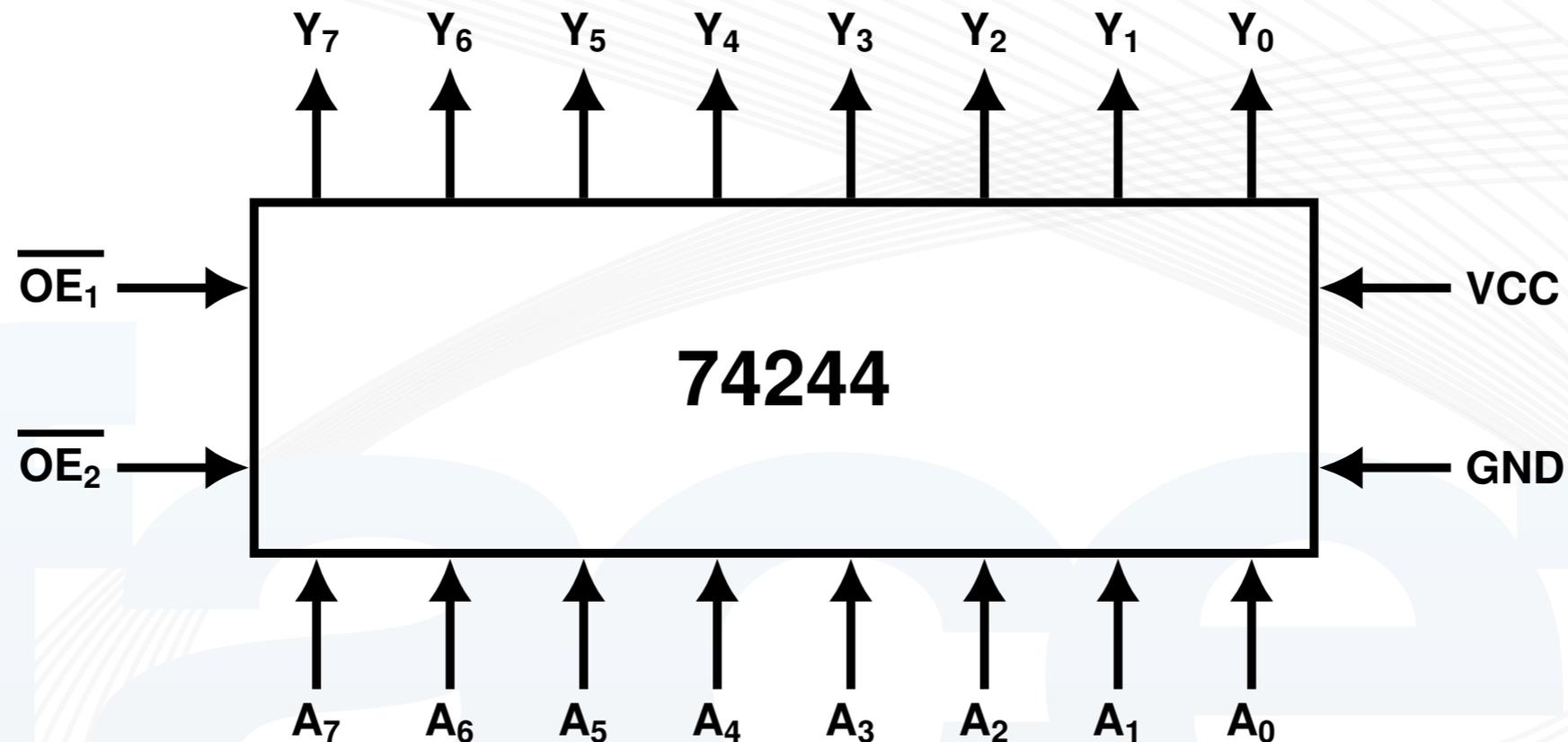


# Camino de datos más complejo

---

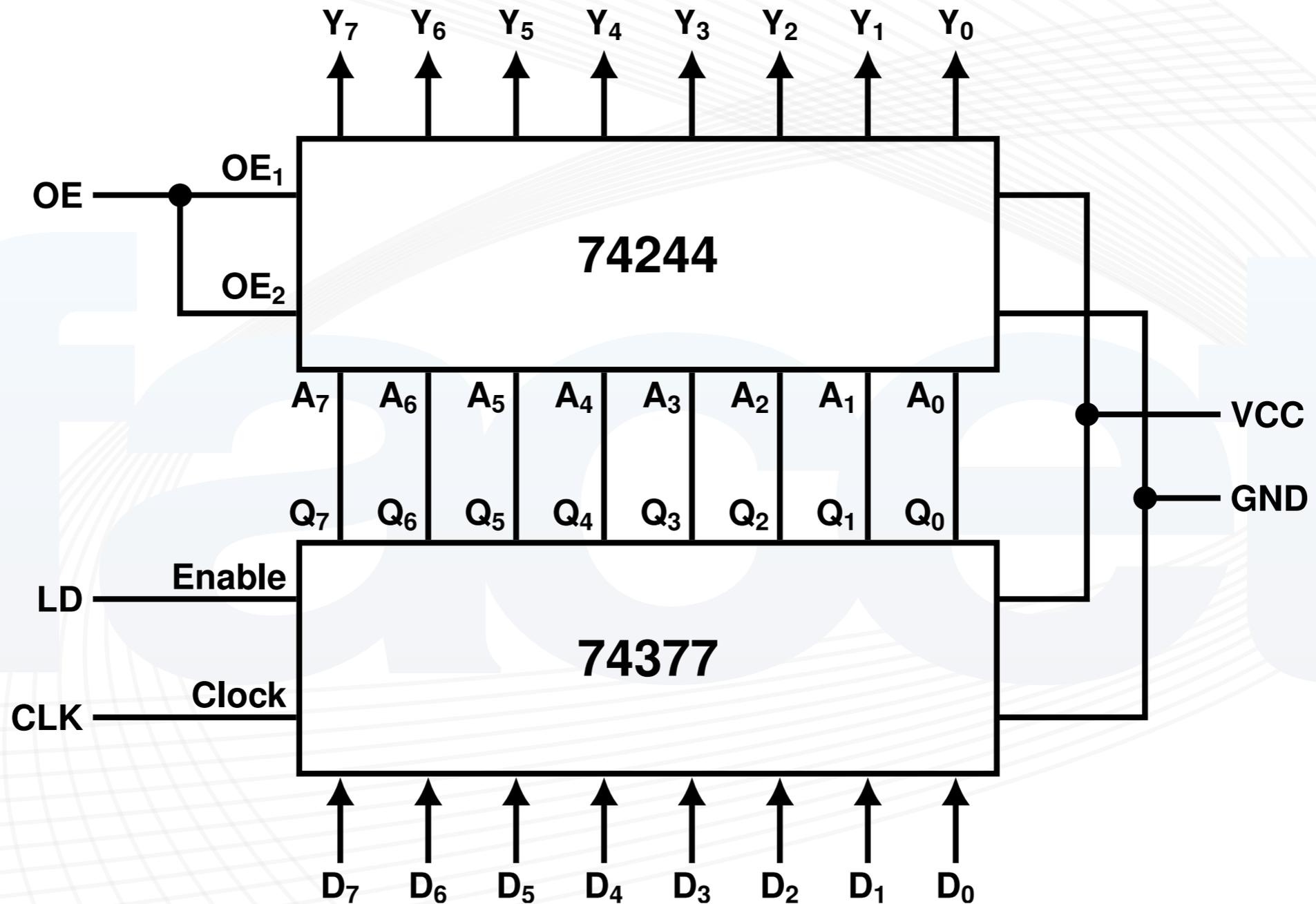
- ▶ Se desea conectar siete registros y una ALU
  - ▶ R1, R2,..., R7 y ALU
  - ▶ Intercambiar contenidos de registros ( $R_i \leftrightarrow R_j$ )
  - ▶ Utilizar cualquier registro como operando ( $R_i \rightarrow \text{ALU}$ )
  - ▶ Guardar el resultado en cualquier registro ( $\text{ALU} \rightarrow R_j$ )
- ▶ Primera solución
  - ▶ Conectar todos contra todos
  - ▶ Evaluar ventajas y desventajas

# Presentación de un buffer 3state



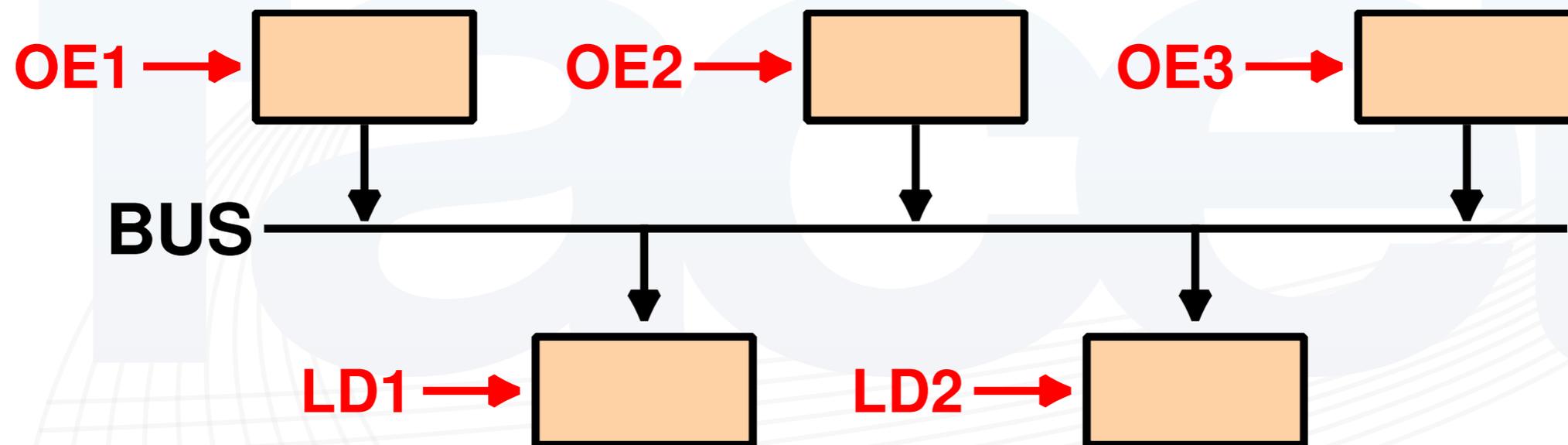
- ▶ Las salidas  $Q_i$  toman el valor de las entradas  $D_i$  cuando la señal  $/OE$  está activa (en bajo)
- ▶ Cuando la señal  $/OE$  está inactiva (en alto) las salidas permanecen  $Q_i$  en alta impedancia (desconectadas)

# Conexión de un registro a un bus



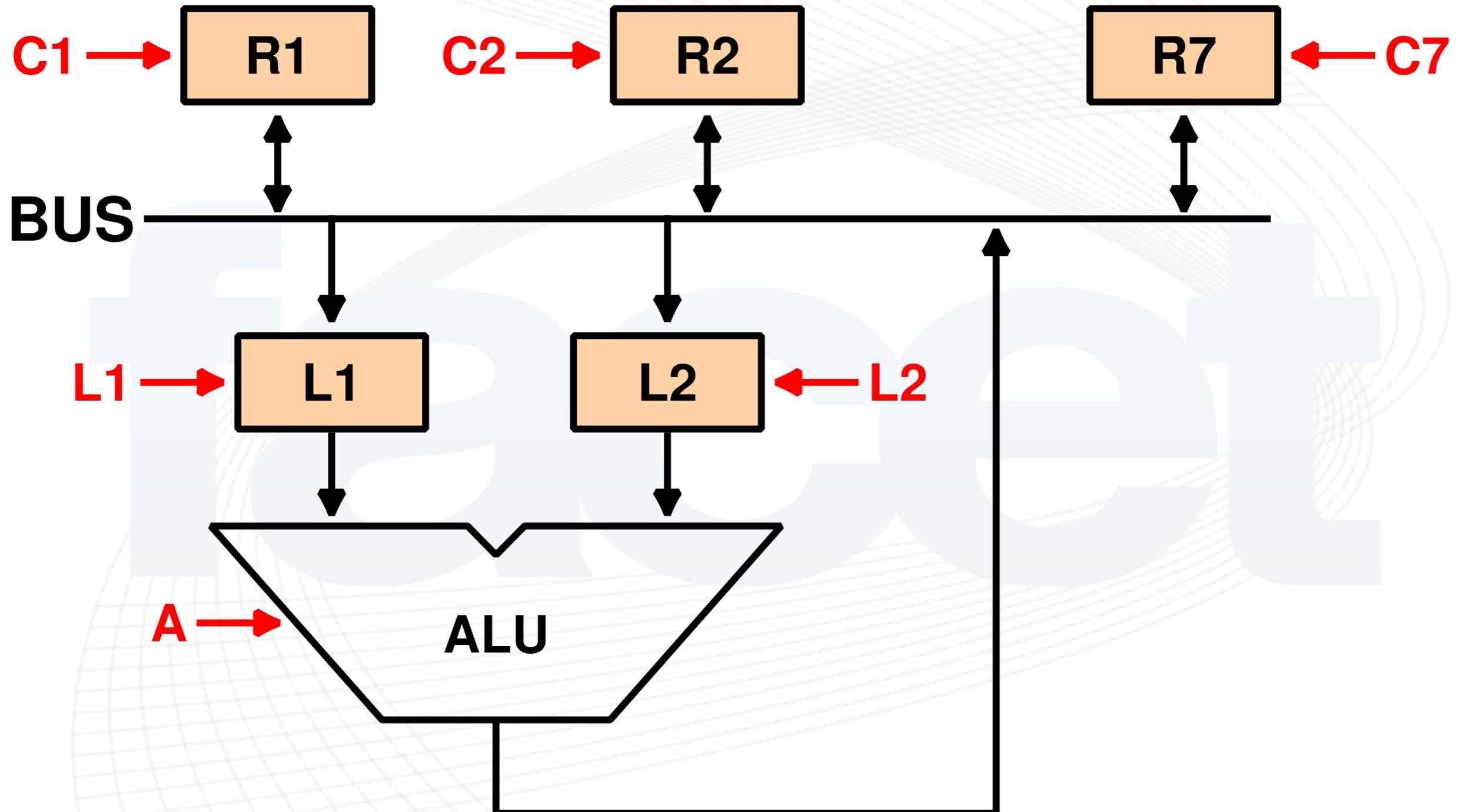
# Concepto de bus

- ▶ Salida 3 State:
  - ▶ Conexión de salidas directa a un conjunto de conductores (Bus)
  - ▶ Se sube con OE=1 (Output Enable)
  - ▶ Se baja con LD=1 (Load)



- ▶ Se sube y se baja en varios lados
- ▶ ¿Cuántos pueden subir a la vez? ¿Y bajar?

# Segunda solución, un bus



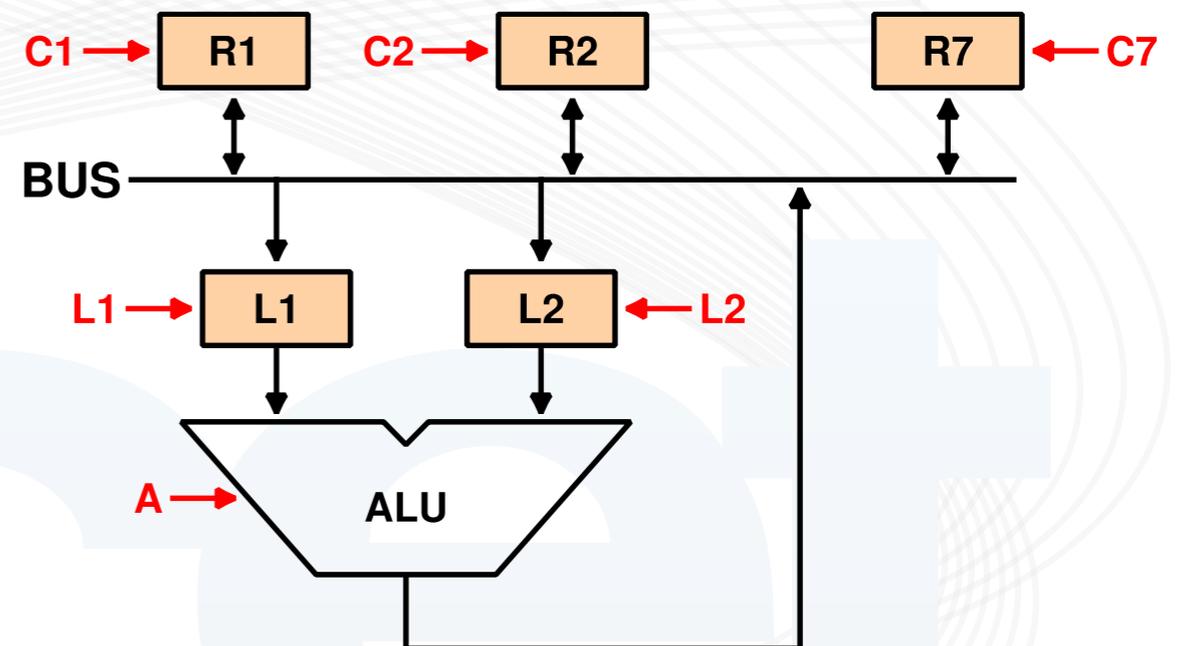
# Observaciones

## ▶ ¿Es solución?

- ▶ Probar  $R1 = R2 + R3$
- ▶ Probar  $R3 \leftarrow R4$
- ▶ Calcular los ciclos de reloj
- ▶ Ventajas y desventajas
- ▶ El BUS es bidireccional

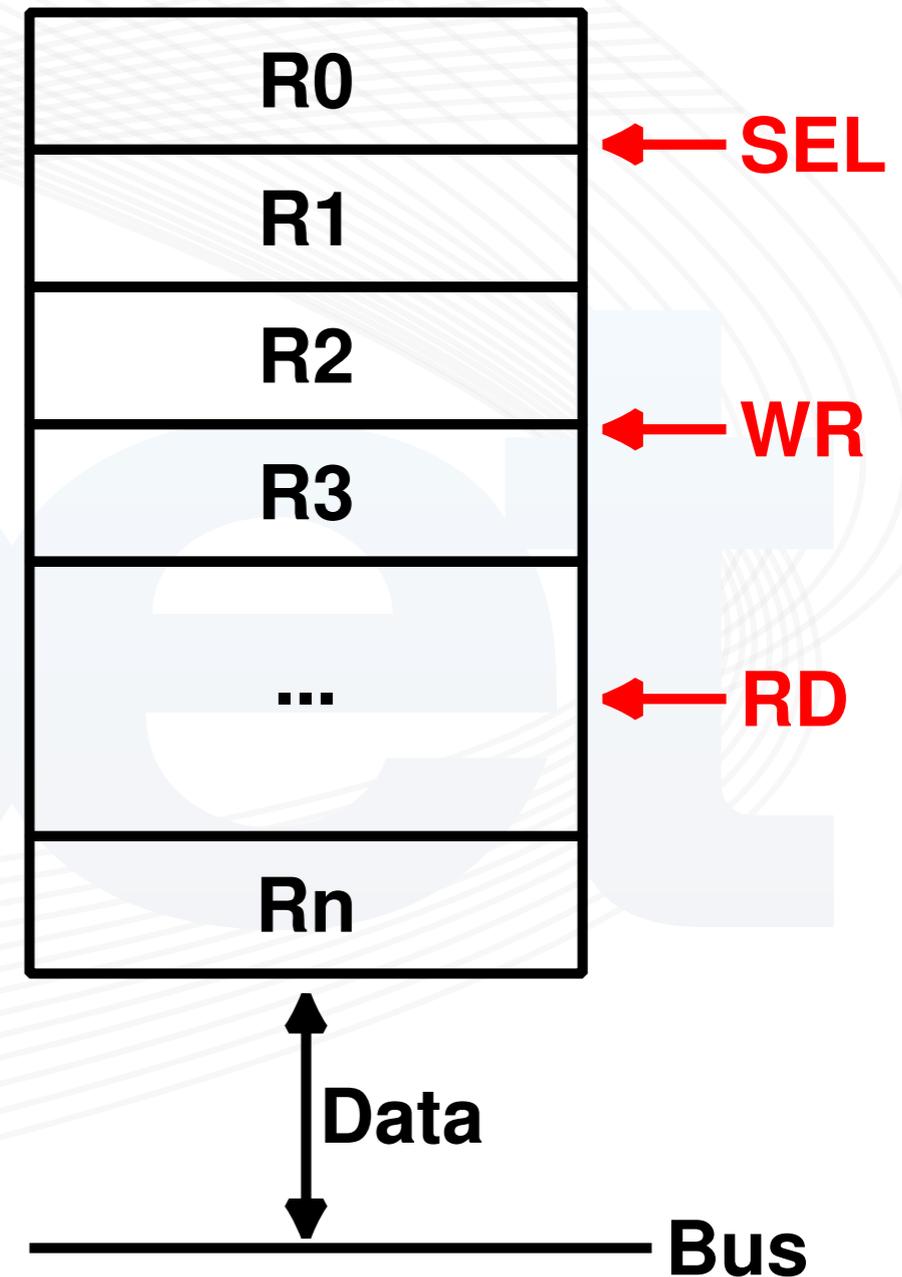
## ▶ Observaciones

- ▶ Solo un registro se conecta a la vez
- ▶ Muchas señales de control =  $O(n)$
- ▶ ¿Se podría eliminar el registro L2?

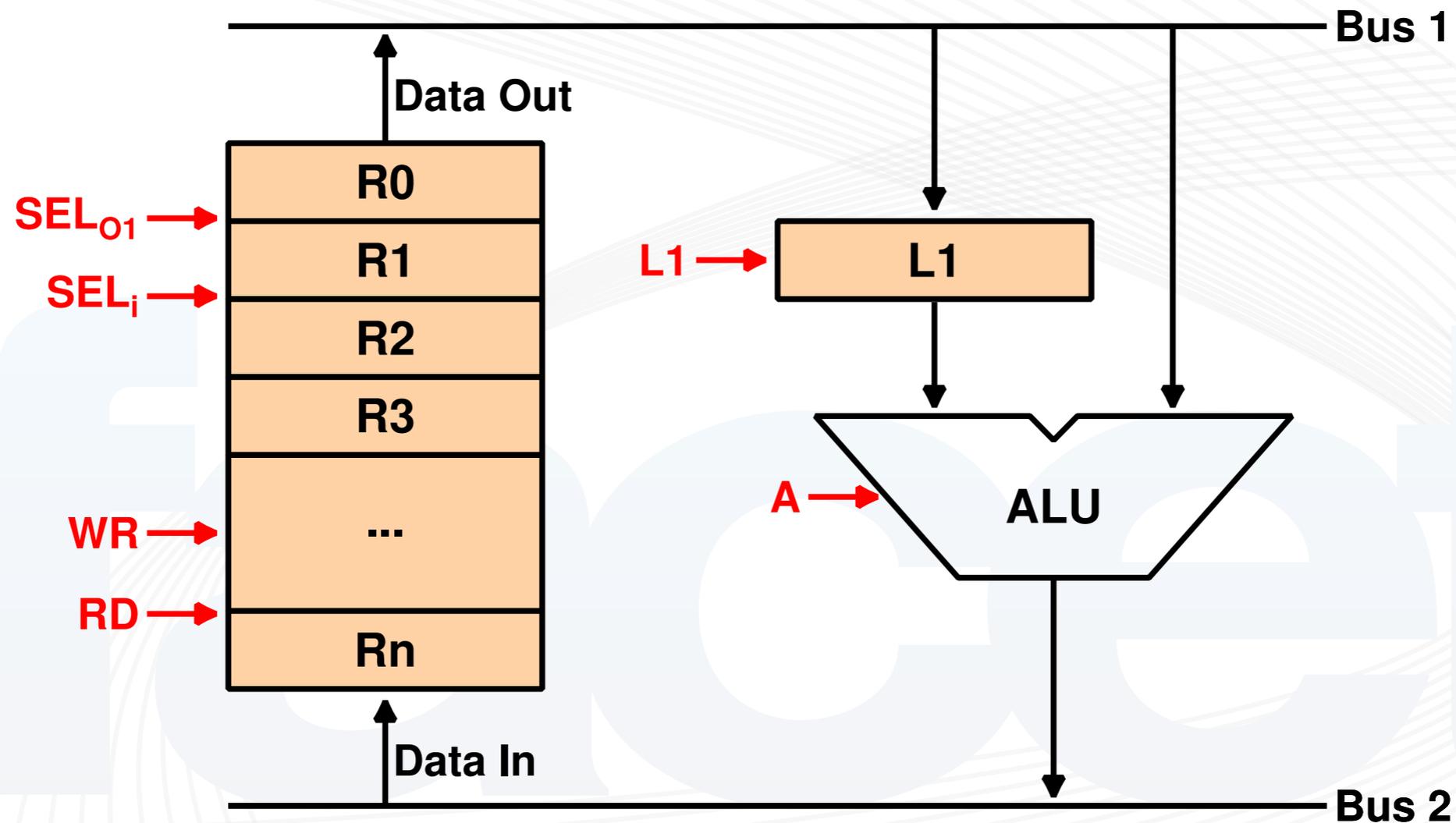


# Banco de registros

- ▶ Entradas de control:  
SEL, WR (LD), RD (OE)
- ▶ Ventaja: integración similar a una memoria
- ▶ SEL require  $\log_2(N)$  cantidad de líneas

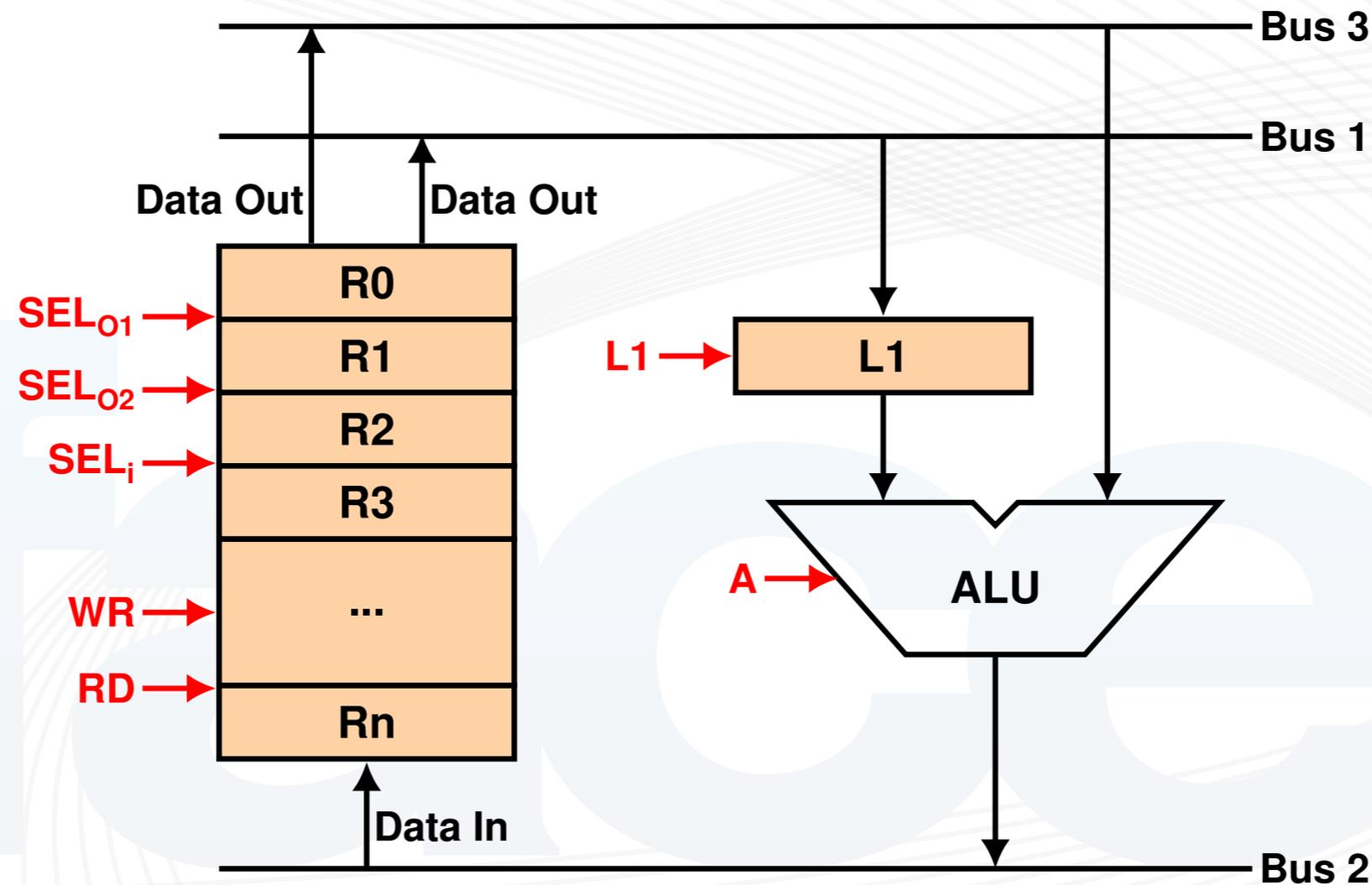


# Solución con dos buses



- ▶ Probar  $R1 = R1 + R2$  y  $R2 \leftrightarrow R3$ , calcular ciclos de clock.
- ▶ ¿Cuánto más rápido es la operación de suma?
- ▶ ¿Puede eliminarse el registro temporal L1?

# Solución con tres buses



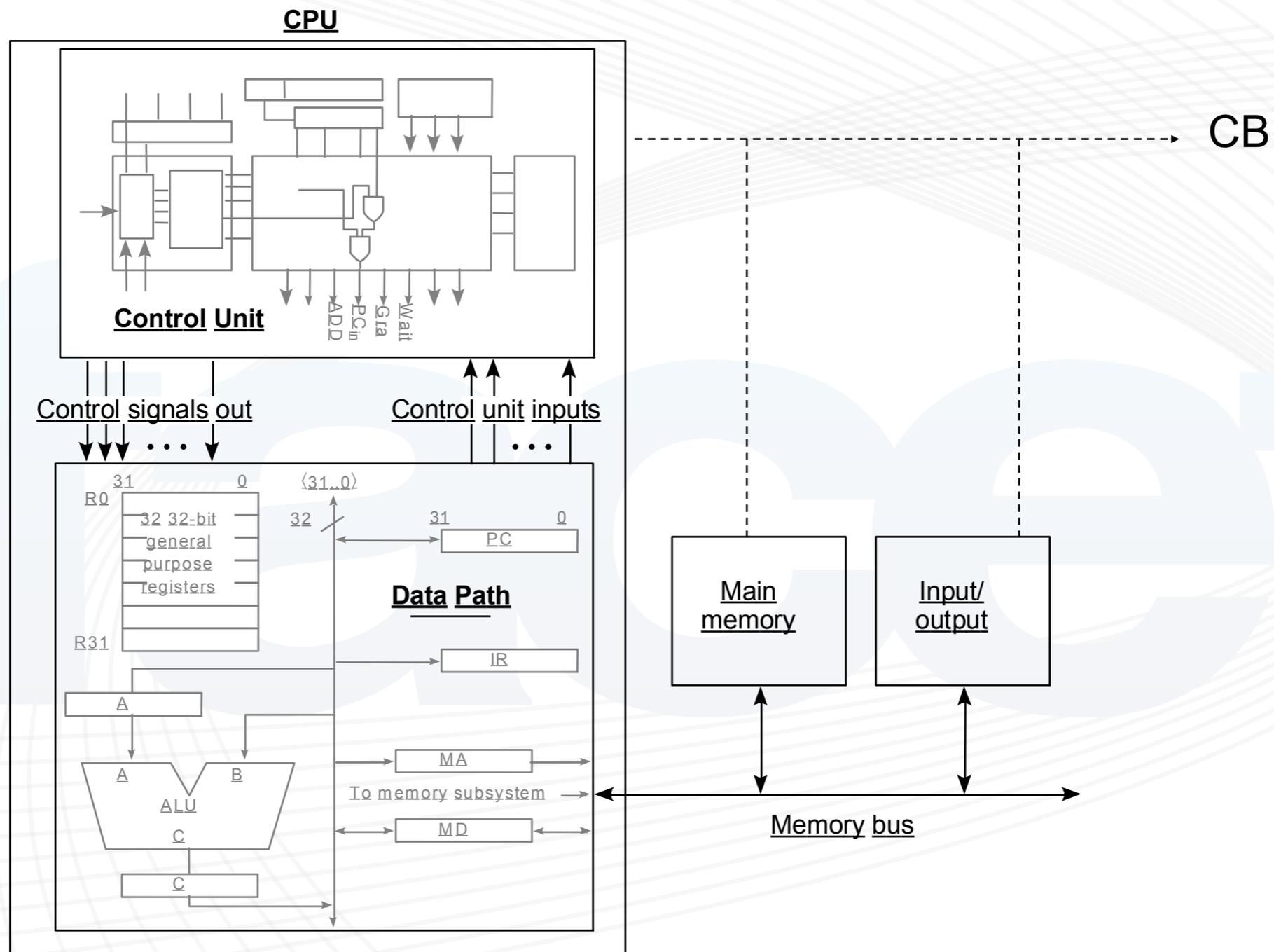
- ▶ Probar  $R1 = R1 + R2$  y  $R2 \leftrightarrow R3$ , calcular ciclos de clock.
- ▶ ¿Cuánto más rápido es la operación de suma?
- ▶ ¿Puede eliminarse el registro temporal L1?

# CPU - Registros Adicionales

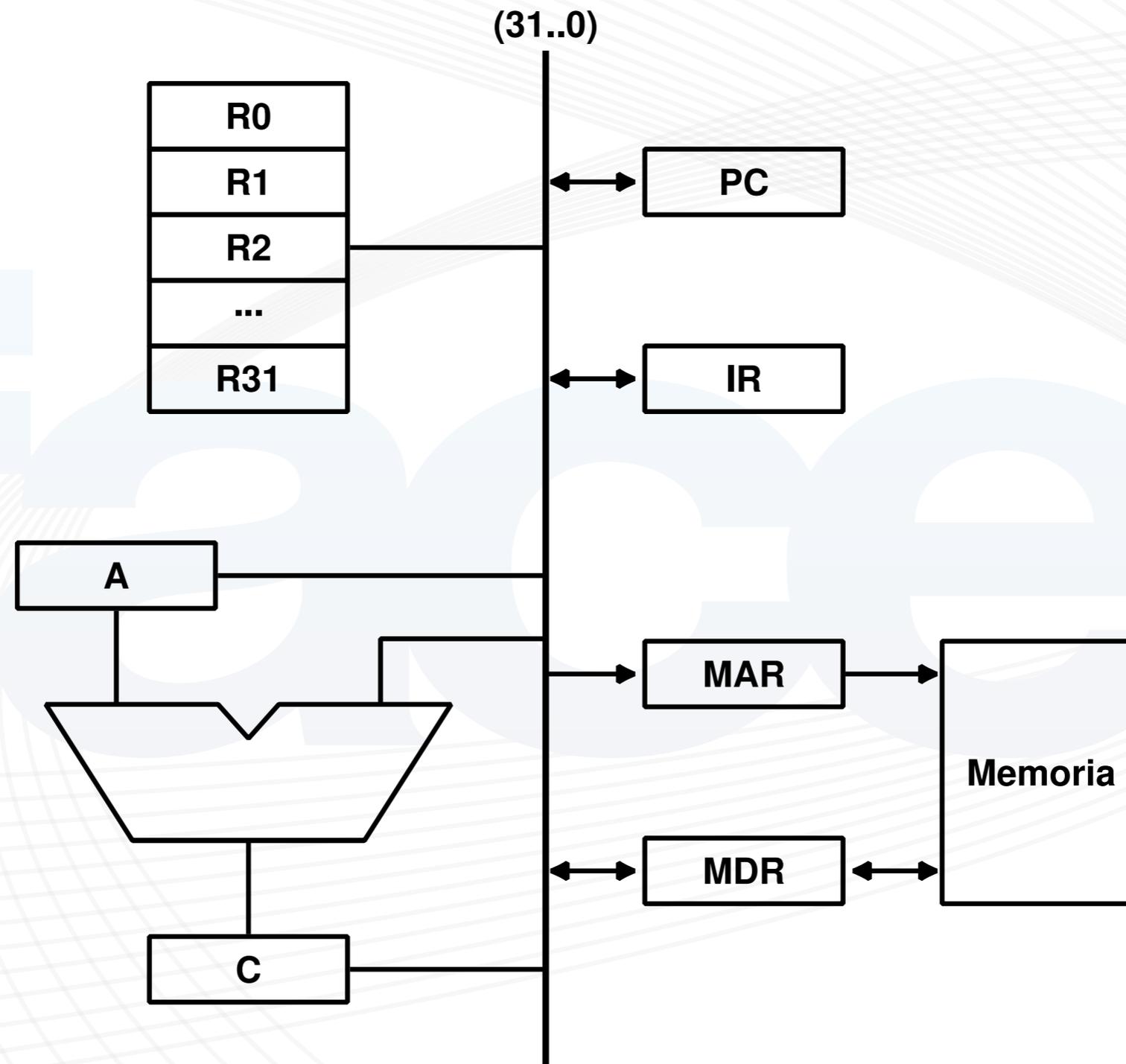
---

- ▶ Para completar un procesador será necesario agregar algunos registros adicionales:
  - ▶ Program Counter (PC): registro que contiene la dirección de memoria donde debe buscarse la próxima instrucción
  - ▶ Instruction Register (IR): registro que contiene la instrucción que se está ejecutando
  - ▶ Memory Address Register (MAR): registro que contiene la dirección de memoria que se quiere leer o escribir
  - ▶ Memory Data Register (MDR): registro que almacena el dato que se lee o que se quiere escribir en memoria

# Diagrama de bloques del procesador

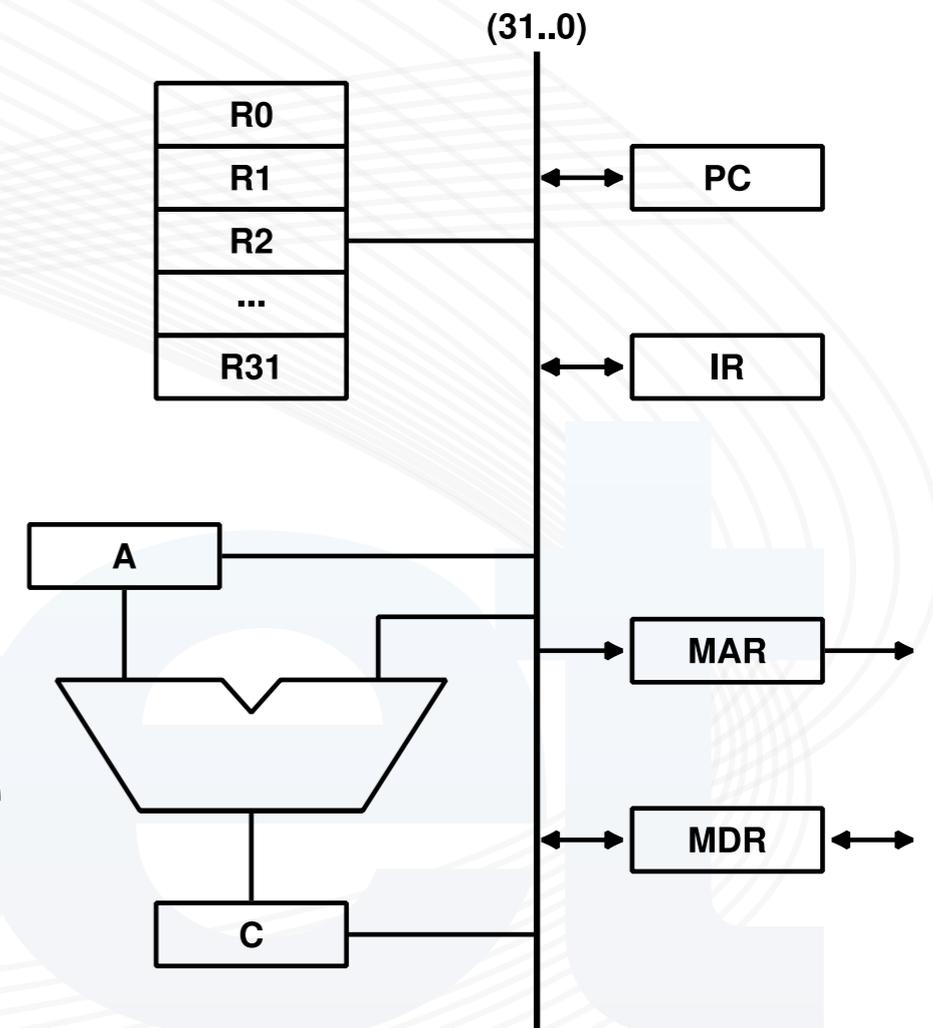


# Camino de datos simplificado



# Comentarios adicionales

- ▶ La ALU, los buses y todos los registros son de 32 bits
- ▶ El primer operando de la ALU siempre está en el registro temporal A
- ▶ El resultado de la ALU siempre se almacena en el registro temporal C
- ▶ El segundo operando de la ALU siempre proviene del bus
- ▶ La memoria se organiza en palabras de 32 bits pero permite el direccionamiento a Bytes
- ▶ El tiempo de acceso a memoria es de 1T



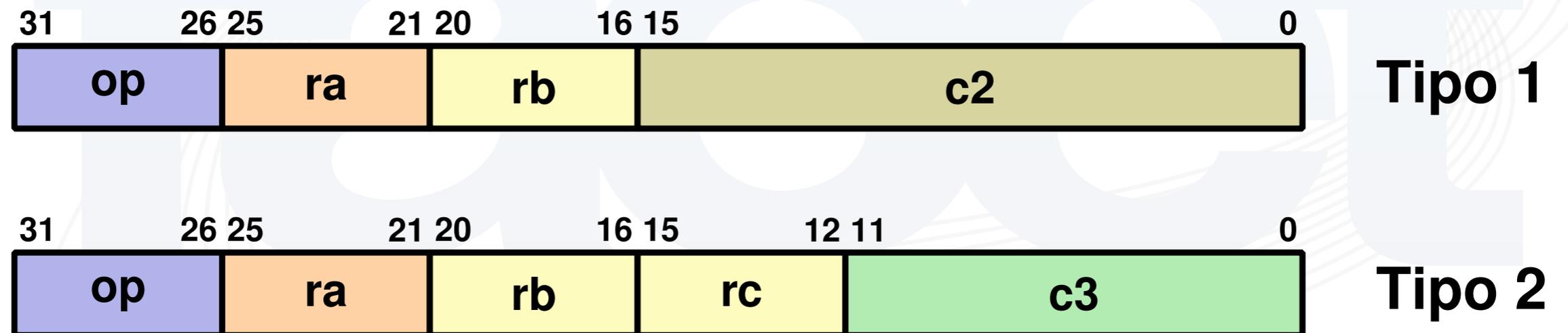
# Descripción Formal de Instrucciones

---

- ▶ RTN – Register Transfer Notation o RTL.
  - ▶ Descripción Abstracta: dice qué hace una instrucción pero no cómo se implementa
  - ▶ Descripción Concreta: dice cómo se implementa una instrucción en un camino de datos concreto
- ▶ Se trata de diseño, por lo que puede haber distintas implementaciones, con ventajas y desventajas
- ▶ Para un ISA existe una única descripción abstracta, pero pueden existir múltiples descripciones concretas
  - ▶ El camino de datos no forma parte del ISA

# Instrucciones del Procesador

- ▶ Formatos de las instrucciones



# Instrucciones del Procesador

---

- ▶ El RTN abstracto del todo el procesador sería el siguiente:
  - ▶ `(IR ← M[PC] : PC ← PC + 4 ; execution) ;`
- ▶ Donde `execution` es la ejecución de la instrucción y se define como:
  - ▶ `execution := (...`
    - ▶ `add (:= op=12) ⇒ R[ra] ← first_op + second_op :`
    - ▶ `ld (:= op=1) ⇒ R[ra] ← M[address] :`
    - ▶ `st (:= op=3) ⇒ M[address] ← R[ra] :`
  - ▶ `...)`
- ▶ Donde `first_op` es el primer operando y se define como:
  - ▶ `first_operand := (rb=0 ⇒ 0 : rb≠0 ⇒ R[rb]) ;`
- ▶ Y donde `address` es la dirección de memoria y se define como:
  - ▶ `address := first_operand + 16@c2<15>#c2<15..0>`

# Ejemplo de programa

---

## ▶ Datos

- ▶ `0x00000100` 2 (valor de la palabra)
- ▶ `0x00000104` 3
- ▶ `0x00000108` (lugar para resultado)

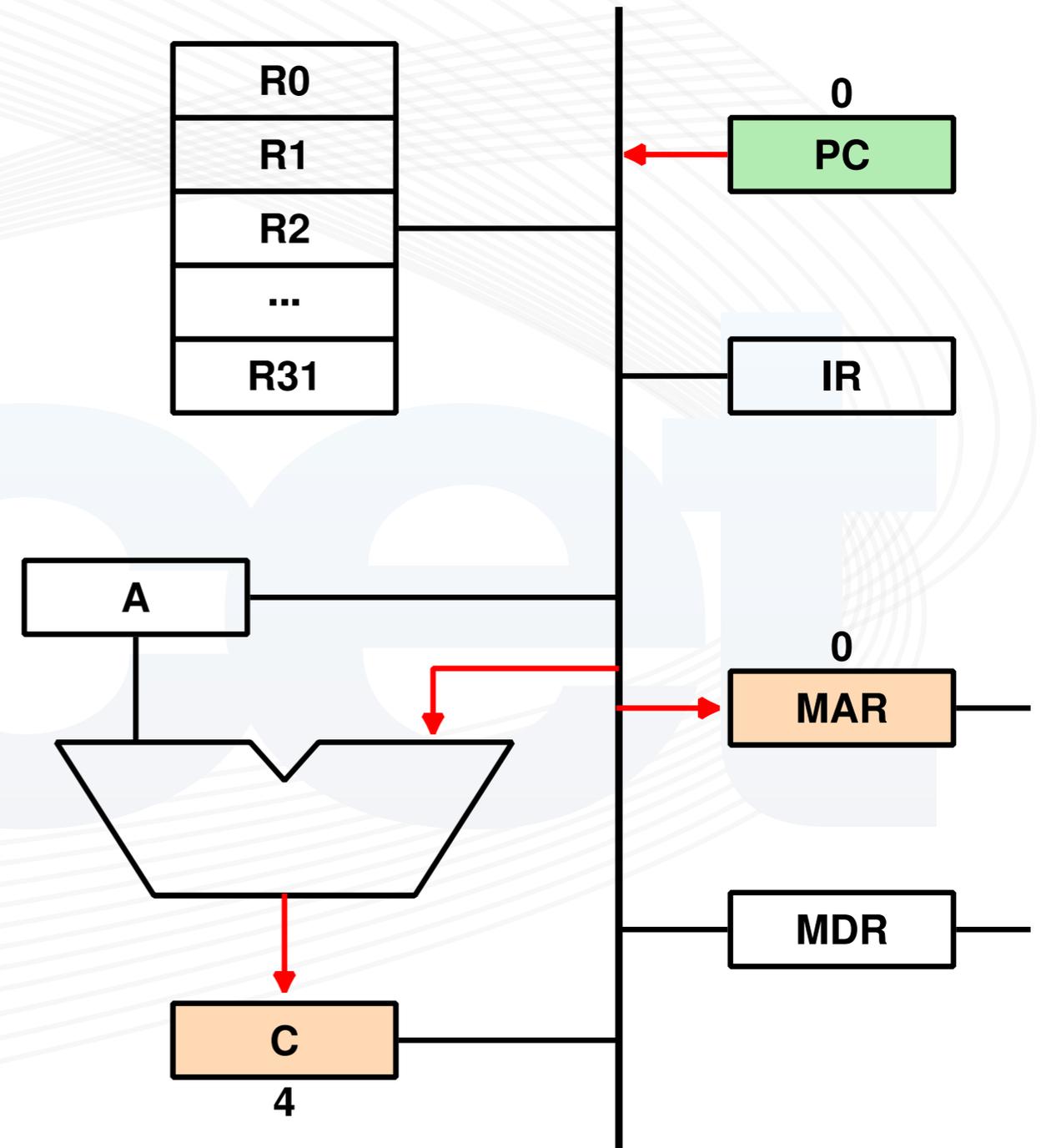
## ▶ Programa

- ▶ `0x00000000` LD R1, R0, 0x0100
  - ▶ `0x00000004` LD R2, R0, 0x0104
  - ▶ `0x00000008` ADD R3, R2, R1
  - ▶ `0x0000000C` ST R3, R0, 0x0108
- ▶ El programa y los datos están en memoria
- ▶ ¿Valor inicial PC?

# Ejecución de LD R1, R0, 0x0100

T0 : MAR  $\leftarrow$  PC :

C  $\leftarrow$  PC + 4 ;



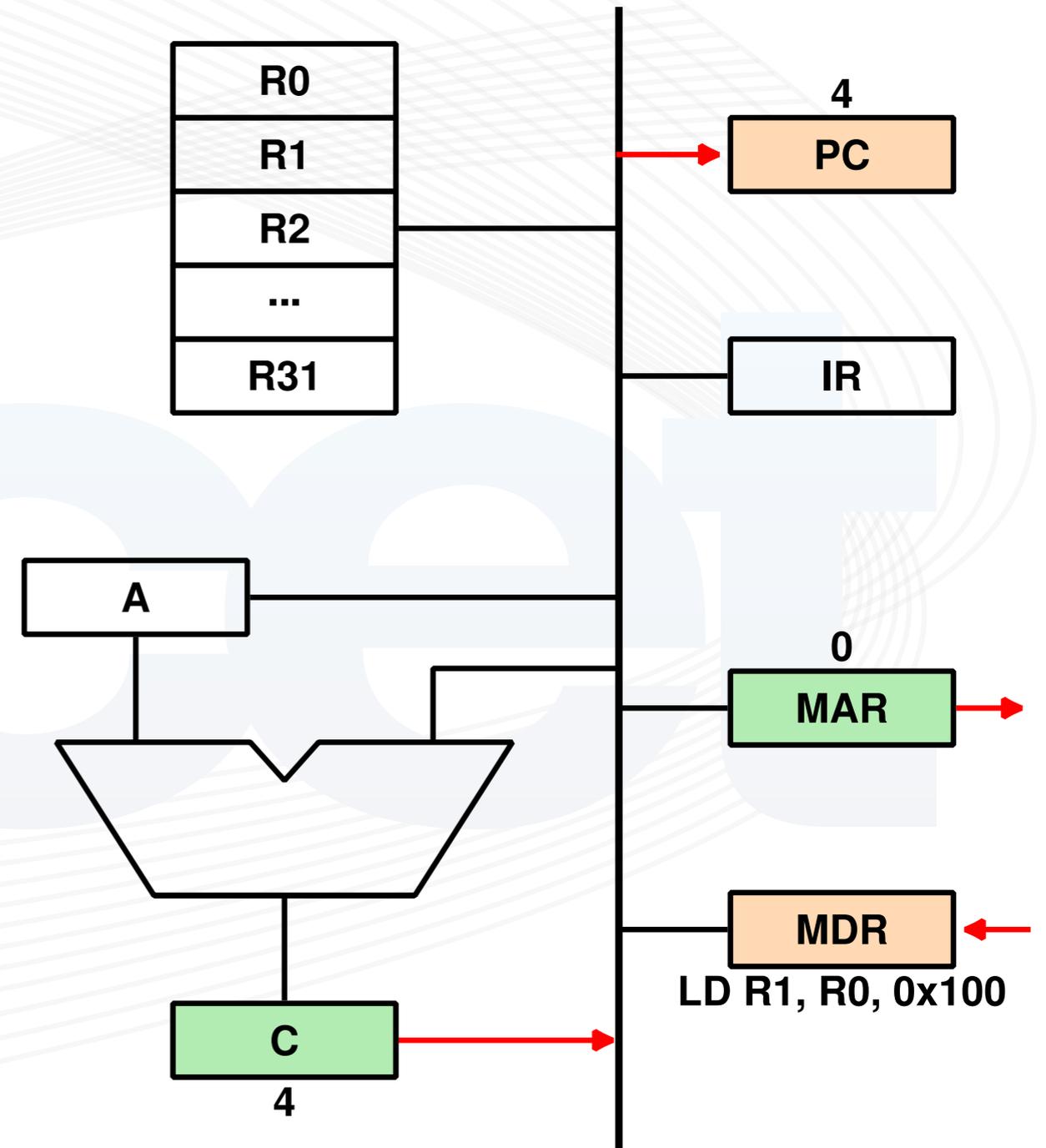
# Ejecución de LD R1, R0, 0x0100

T0:  $MAR \leftarrow PC:$

$C \leftarrow PC + 4;$

T1:  $MDR \leftarrow M[MAR]:$

$PC \leftarrow C;$



# Ejecución de LD R1, R0, 0x0100

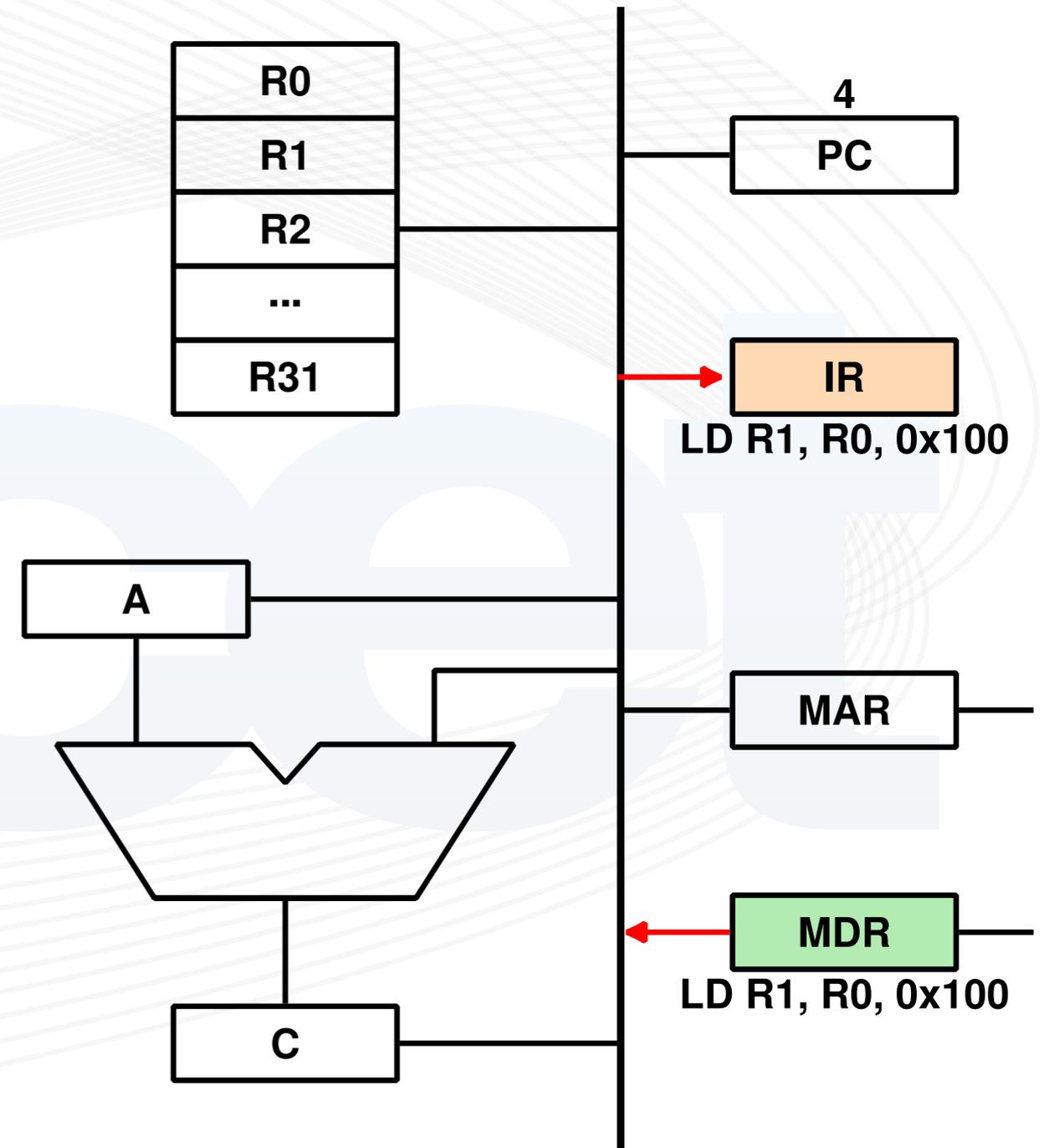
T0: MAR  $\leftarrow$  PC:

C  $\leftarrow$  PC + 4;

T1: MDR  $\leftarrow$  M[MAR]:

PC  $\leftarrow$  C;

T2: IR  $\leftarrow$  MDR;



# Ejecución de LD R1, R0, 0x0100

T0: MAR  $\leftarrow$  PC:

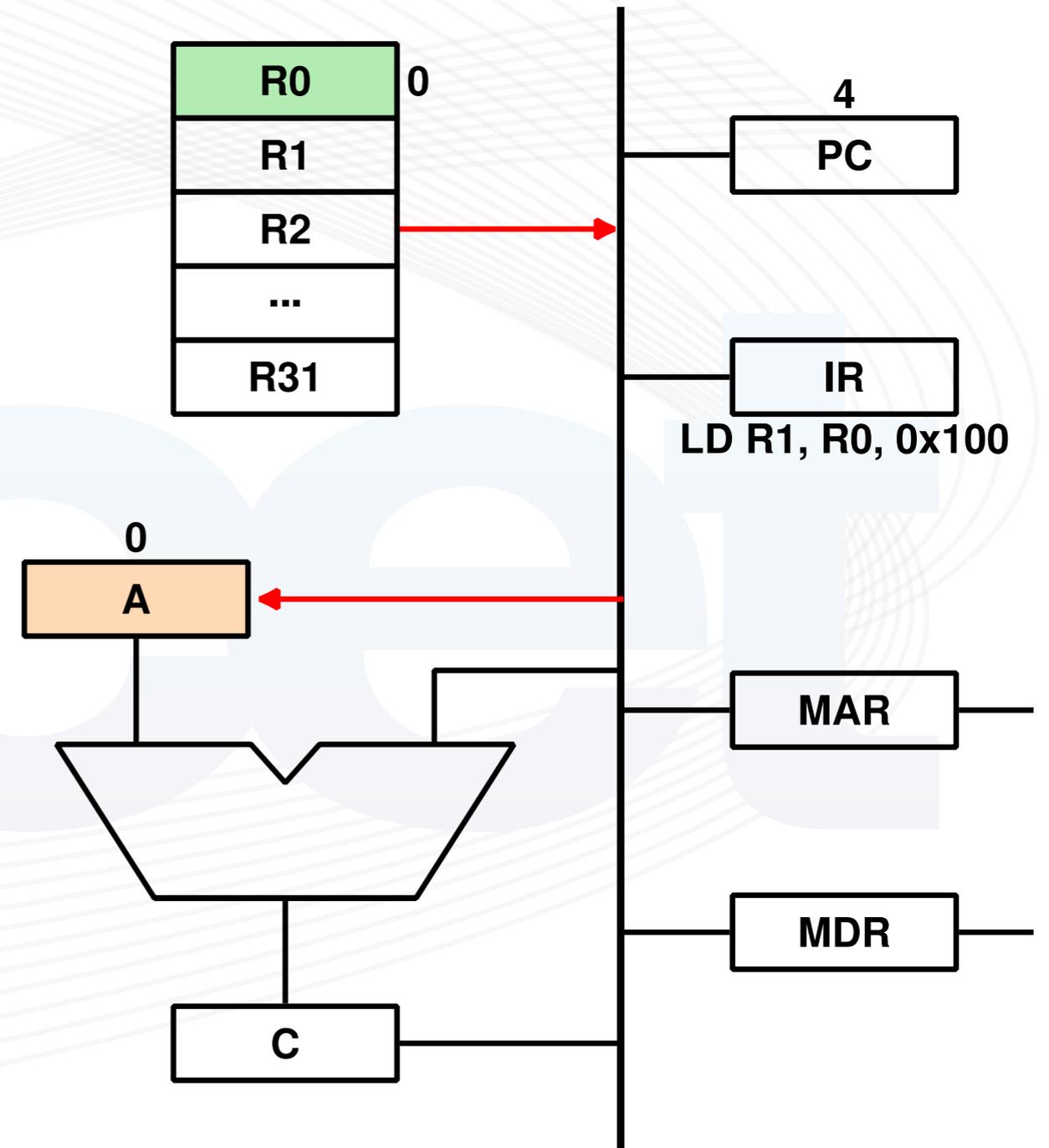
C  $\leftarrow$  PC + 4;

T1: MDR  $\leftarrow$  M[MAR]:

PC  $\leftarrow$  C;

T2: IR  $\leftarrow$  MDR;

T3: A  $\leftarrow$  (rb=0  $\Rightarrow$  0:  
rb $\neq$ 0  $\Rightarrow$  R[rb]);



# Ejecución de LD R1, R0, 0x100

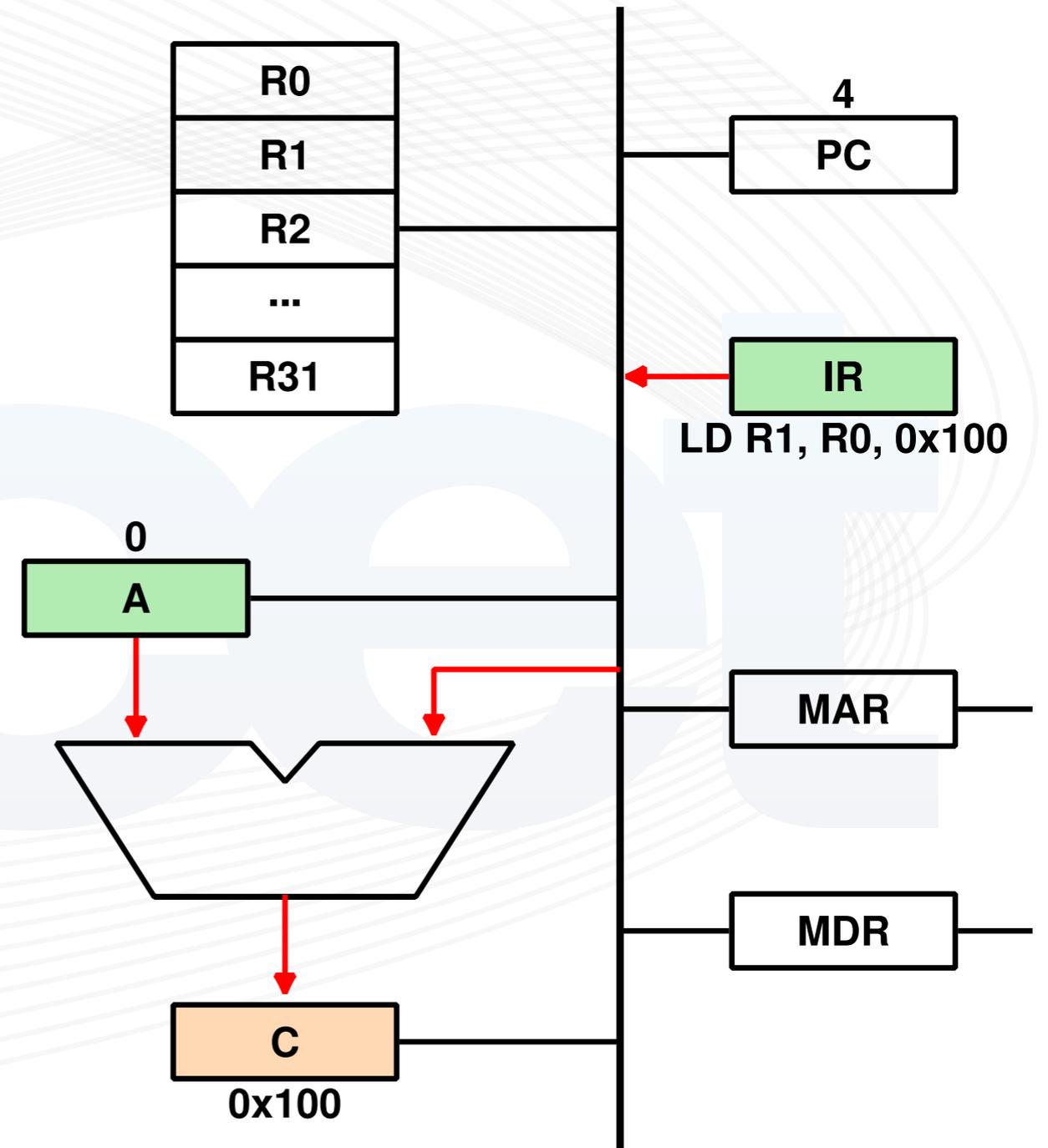
T0:  $MAR \leftarrow PC$ ;  
 $C \leftarrow PC + 4$ ;

T1:  $MDR \leftarrow M[MAR]$ ;  
 $PC \leftarrow C$ ;

T2:  $IR \leftarrow MDR$ ;

T3:  $A \leftarrow (rb=0 \Rightarrow 0 : rb \neq 0 \Rightarrow R[rb])$ ;

T4:  $C \leftarrow A + (16 @ IR(15) \# IR < 15 .. 0 >)$ ;



# Ejecución de LD R1, R0, 0x0100

T0: MAR  $\leftarrow$  PC:

C  $\leftarrow$  PC + 4;

T1: MDR  $\leftarrow$  M[MAR]:

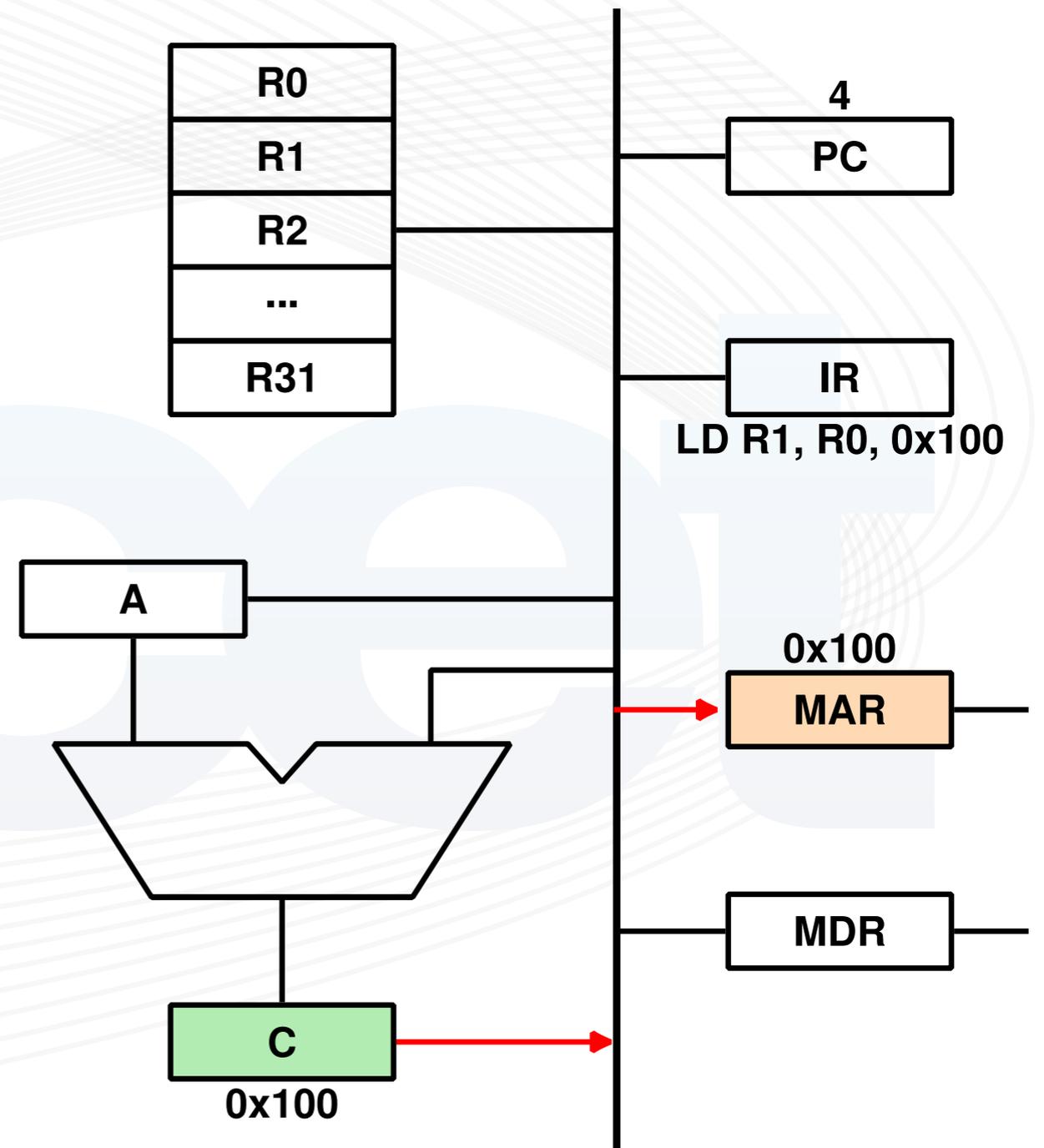
PC  $\leftarrow$  C;

T2: IR  $\leftarrow$  MDR;

T3: A  $\leftarrow$  (rb=0  $\Rightarrow$  0:  
rb $\neq$ 0  $\Rightarrow$  R[rb]);

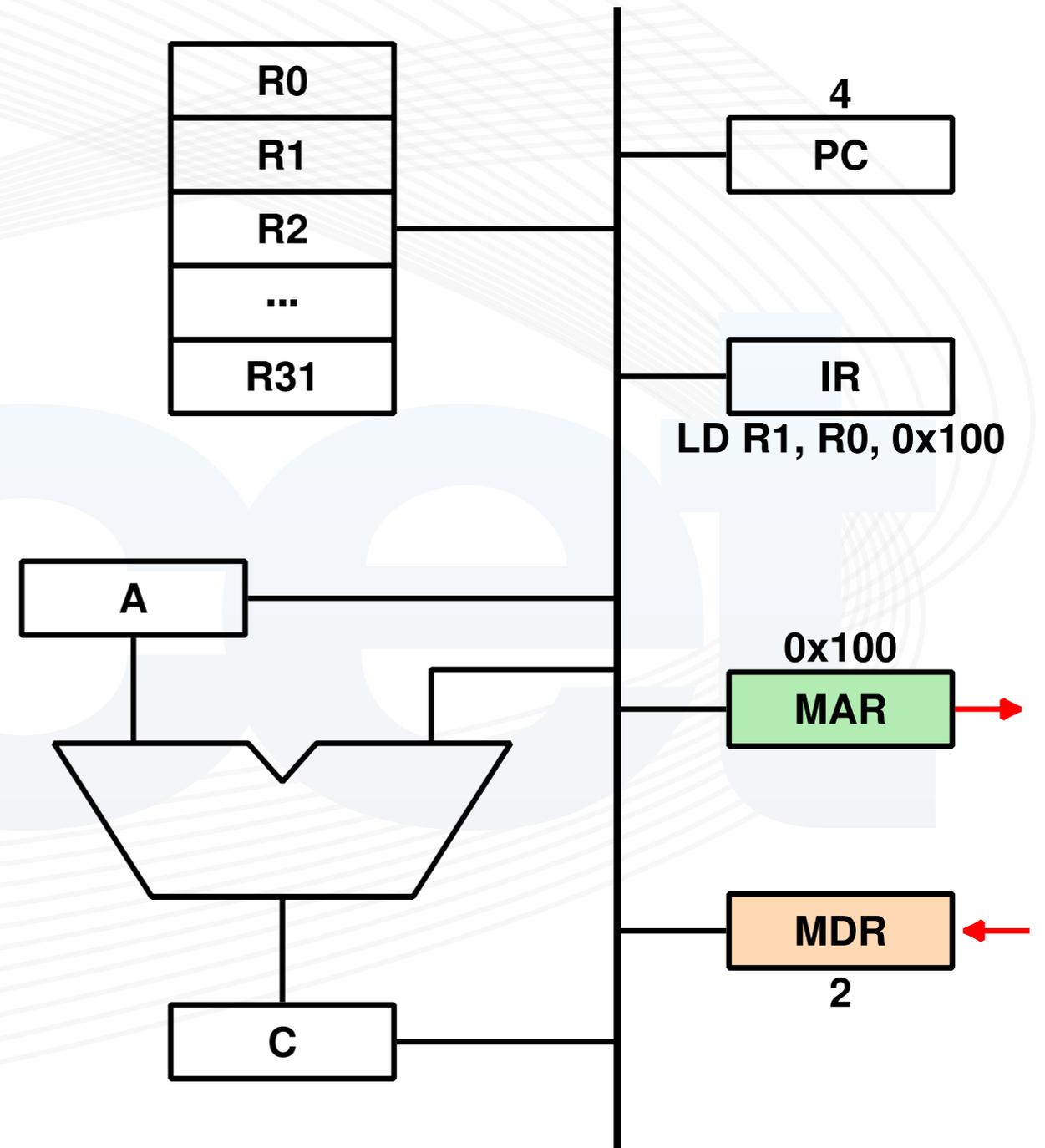
T4: C  $\leftarrow$  A + (16@IR(15)  
#IR<15..0>);

T5: MAR  $\leftarrow$  C;



# Ejecución de LD R1, R0, 0x0100

T0:  $MAR \leftarrow PC$ ;  
 $C \leftarrow PC + 4$ ;  
T1:  $MDR \leftarrow M[MAR]$ ;  
 $PC \leftarrow C$ ;  
T2:  $IR \leftarrow MDR$ ;  
T3:  $A \leftarrow (rb=0 \Rightarrow 0$ ;  
 $rb \neq 0 \Rightarrow R[rb])$ ;  
T4:  $C \leftarrow A + (16 @ IR(15)$   
 $\#IR < 15 \dots 0 >)$ ;  
T5:  $MAR \leftarrow C$ ;  
T6:  $MDR \leftarrow M[MAR]$ ;



# Ejecución de LD R1, R0, 0x0100

T0: MAR  $\leftarrow$  PC:

C  $\leftarrow$  PC + 4;

T1: MDR  $\leftarrow$  M[MAR]:

PC  $\leftarrow$  C;

T2: IR  $\leftarrow$  MDR;

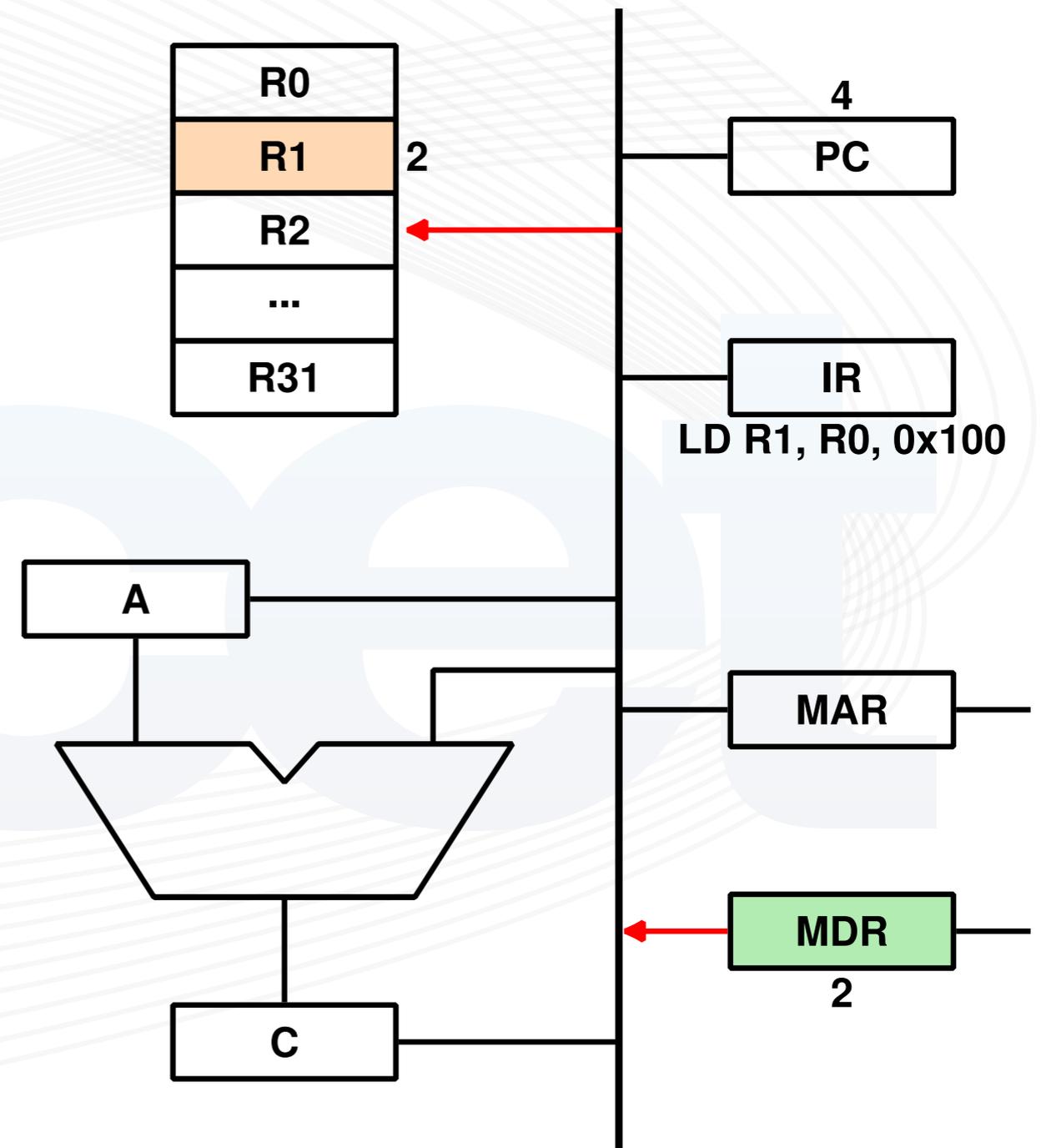
T3: A  $\leftarrow$  (rb=0  $\Rightarrow$  0:  
rb $\neq$ 0  $\Rightarrow$  R[rb]);

T4: C  $\leftarrow$  A + (16@IR(15)  
#IR<15..0>);

T5: MAR  $\leftarrow$  C;

T6: MDR  $\leftarrow$  M[MAR];

T7: R[ra]  $\leftarrow$  MDR;



# LD R1, R0, 0x0100

Ciclo	RTN concreto	Valores en los registros
T0	$\text{MAR} \leftarrow \text{PC} : \text{C} \leftarrow \text{PC} + 4 ;$	MAR=0, C=4
T1	$\text{MDR} \leftarrow \text{M}[\text{MAR}] : \text{PC} \leftarrow \text{C} ;$	MDR=M[0]=LD, PC=4
T2	$\text{IR} \leftarrow \text{MDR} ;$	IR=LD
T3	$\text{A} \leftarrow (\text{rb}=0 \Rightarrow 0 : \text{rb} \neq 0 \Rightarrow \text{R}[\text{rb}]) ;$	A=0
T4	$\text{C} \leftarrow \text{A} + (16 @ \text{IR}(15) \# \text{IR} < 15 \dots 0 >) ;$	C=0x00000100
T5	$\text{MAR} \leftarrow \text{C} ;$	MAR=0x00000100
T6	$\text{MDR} \leftarrow \text{M}[\text{MAR}] ;$	MDR=M[0x100]=2
T7	$\text{R}[\text{ra}] \leftarrow \text{MDR} ;$	R[1]=2

# LD R2, R0, 0x0104

Ciclo	RTN concreto	Valores en los registros
T8	$\text{MAR} \leftarrow \text{PC} : \text{C} \leftarrow \text{PC} + 4 ;$	MAR=4, C=8
T9	$\text{MDR} \leftarrow \text{M}[\text{MAR}] : \text{PC} \leftarrow \text{C} ;$	MDR=M[4]=LD, PC=8
T10	$\text{IR} \leftarrow \text{MDR} ;$	IR=LD
T11	$\text{A} \leftarrow (\text{rb}=0 \Rightarrow 0 : \text{rb} \neq 0 \Rightarrow \text{R}[\text{rb}]) ;$	A=0
T12	$\text{C} \leftarrow \text{A} + (16 @ \text{IR}(15) \# \text{IR} < 15 \dots 0 >) ;$	C=0x00000104
T13	$\text{MAR} \leftarrow \text{C} ;$	MAR=0x00000104
T14	$\text{MDR} \leftarrow \text{M}[\text{MAR}] ;$	MDR=M[0x104]=3
T15	$\text{R}[\text{ra}] \leftarrow \text{MDR} ;$	R[2]=3

# ADD R3, R2, R1

Ciclo	RTN concreto	Valores en los registros
T16	$\text{MAR} \leftarrow \text{PC} : \text{C} \leftarrow \text{PC} + 4 ;$	MAR=8, C=12
T17	$\text{MDR} \leftarrow \text{M}[\text{MAR}] : \text{PC} \leftarrow \text{C} ;$	MDR=M[8]=ADD, PC=12
T18	$\text{IR} \leftarrow \text{MDR} ;$	IR=ADD
T19	$\text{A} \leftarrow (\text{rb}=0 \Rightarrow 0 : \text{rb} \neq 0 \Rightarrow \text{R}[\text{rb}] ) ;$	A=R[2]=3
T20	$\text{C} \leftarrow \text{A} + \text{R}[\text{rc}] ;$	C=A+R[1]=3+2=5
T21	$\text{R}[\text{ra}] \leftarrow \text{C} ;$	R[3]=5

# ST R3, R0, 0x0108

Ciclo	RTN concreto	Valores en los registros
T22	$\text{MAR} \leftarrow \text{PC} : \text{C} \leftarrow \text{PC} + 4 ;$	MAR=12, C=16
T23	$\text{MDR} \leftarrow \text{M}[\text{MAR}] : \text{PC} \leftarrow \text{C} ;$	MDR=M[12]=ST, PC=16
T24	$\text{IR} \leftarrow \text{MDR} ;$	IR=ST
T25	$\text{A} \leftarrow (\text{rb}=0 \Rightarrow 0 : \text{rb} \neq 0 \Rightarrow \text{R}[\text{rb}]) ;$	A=0
T26	$\text{C} \leftarrow \text{A} + (16 @ \text{IR}(15) \# \text{IR} < 15 \dots 0 >) ;$	C=0x00000108
T27	$\text{MAR} \leftarrow \text{C} ;$	MAR=0x00000108
T28	$\text{MDR} \leftarrow \text{R}[\text{ra}] ;$	MDR=R[3]=5
T29	$\text{M}[\text{MAR}] \leftarrow \text{MDR} ;$	M[0x108]=MDR=2

# Conclusiones del ejemplo

---

- ▶ Realización concreta con un bus: SRC-1
  - ▶ La búsqueda es la misma para todas las instrucciones
  - ▶ Una transferencia abstracta puede necesitar varias transferencias concretas en función del camino de datos
- ▶ El CPU es sólo una máquina.
  - ▶ Siempre hace lo mismo según el ciclo de la instrucción
- ▶ El Programa es quien da la “inteligencia”
  - ▶ Un programa sintetiza un algoritmo
- ▶ Al final el PC apunta a la siguiente instrucción
  - ▶ El procesador continúa indefinidamente buscando y ejecutando instrucciones