



Universidad Nacional
de Entre Ríos

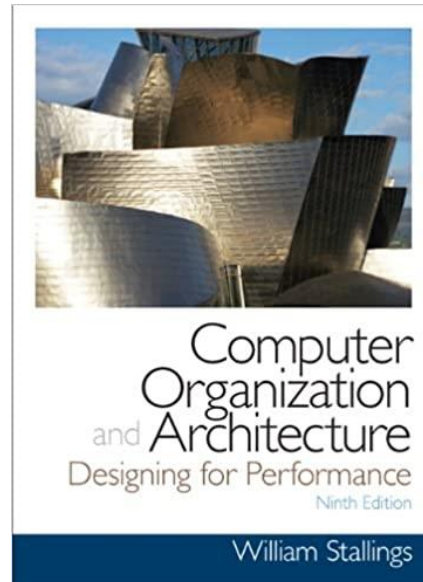
Tecnicatura universitaria en desarrollo web

Diseño de CPU

Semana 3 – Arquitectura de computadoras

Esta presentación esta basada en el libro de:

- ❑ William Stallings, Computer Organization and Architecture, 9th Edition, 2017




Archivos presentación y ejemplos se alojan en:

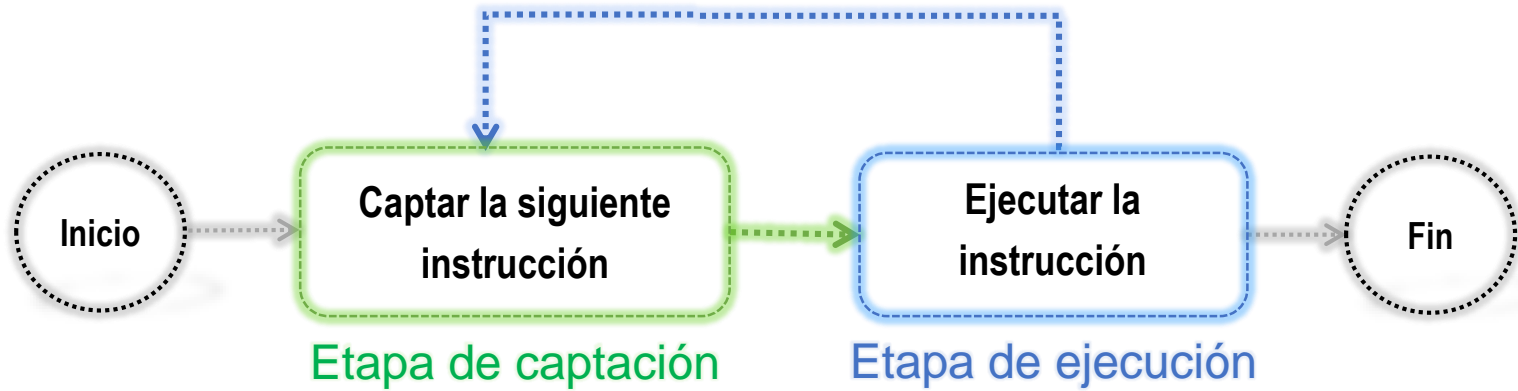


<https://github.com/ruiz-jose/tudw-arq.git>

Diseño de CPU

- Ciclo de la instrucción 
 - Etapa de captación y ejecución

- Componentes de la computadoras:
 - ALU y registros
 - Arquitectura acumulador (ACC)



Lenguaje alto nivel

❑ C:

```
Int z =0, x=3, y=2;
z = x + y;
```

Se traduce

Lenguaje bajo nivel

❑ Ensamblador:

```
load x
add y
store z
```

Mapeo 1 a 1

Lenguaje de maquina

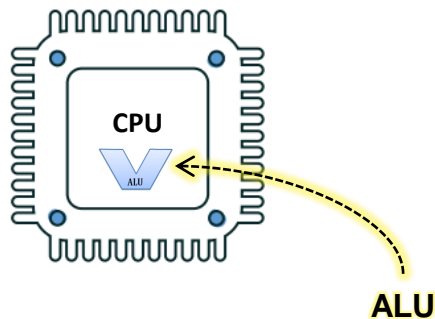
❑ Código maquina:

00	110010
10	110011
01	110100

Una instrucción en **alto nivel** equivale a varias instrucciones en **bajo nivel**.

.data

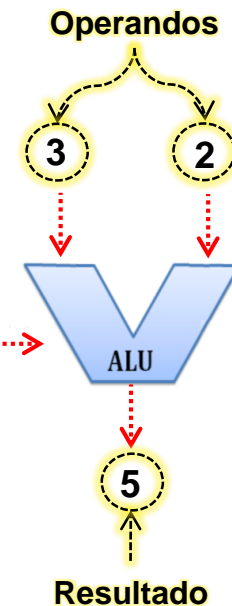
```
x db 3
y db 2
z db 0
```



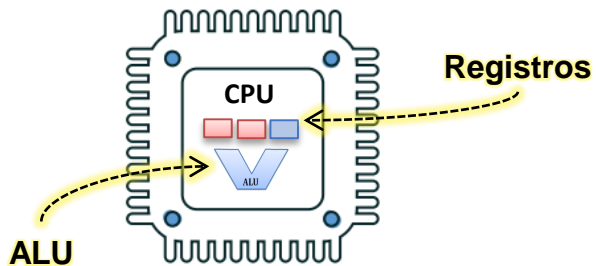
□ **C:**
`Int z =0, x=3, y=2;`
`z = x + y;`

□ **Ensamblador:**
`load x`
`add y`
`store z`

Código operación \dashrightarrow **add**



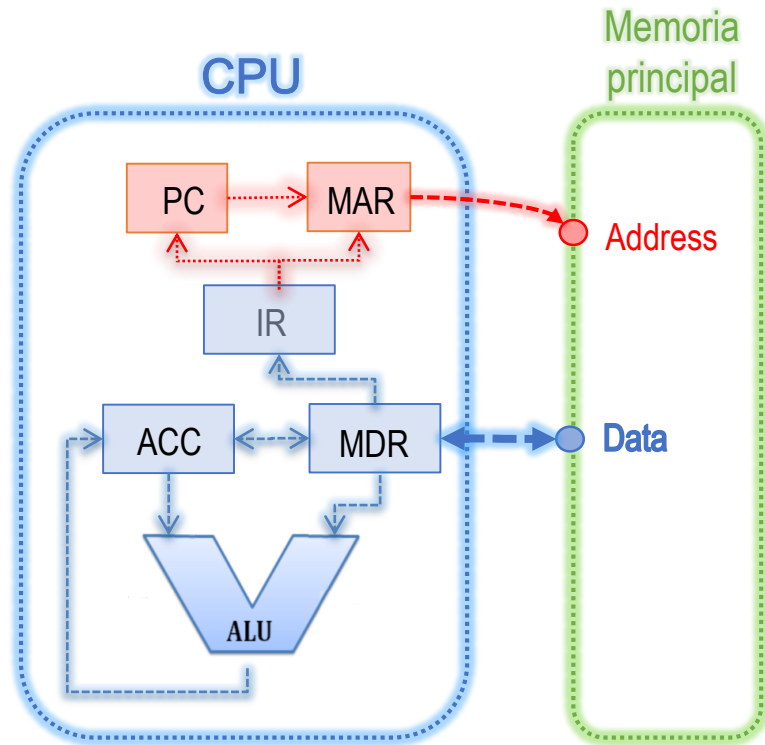
¿de dónde salen los datos **x e y**?, ¿dónde están ese 3, ese 2 y el número que representa la operación de suma? En algún lado tienen que estar almacenados, ¿no?



Registros:

- **Contador de programa de PC:** contiene la dirección de la próxima instrucción que se ejecutará
- **Registro de direcciones de memoria MAR:** contiene la ubicación de la memoria de los datos a los que se debe acceder.
- **Registro de datos de memoria MDR:** contiene datos que se transfieren a/o desde la memoria.
- **Acumulador ACC:** se almacenan resultados aritméticos y lógicos intermedios.
- **Registro de instrucción IR:** contiene la instrucción actual durante el procesamiento.

Arquitectura acumulador (ACC)



En la lenguaje ensamblador de la **arquitectura acumulador** se utilizan los nemónico **LDA** (load acc) y **STA** (store acc) para indicar transferencia desde o a memoria en vez de **load** y **store**, entonces ensamblador seria:

load x → **LDA x**

store z → **STA z**

❑ **C:**

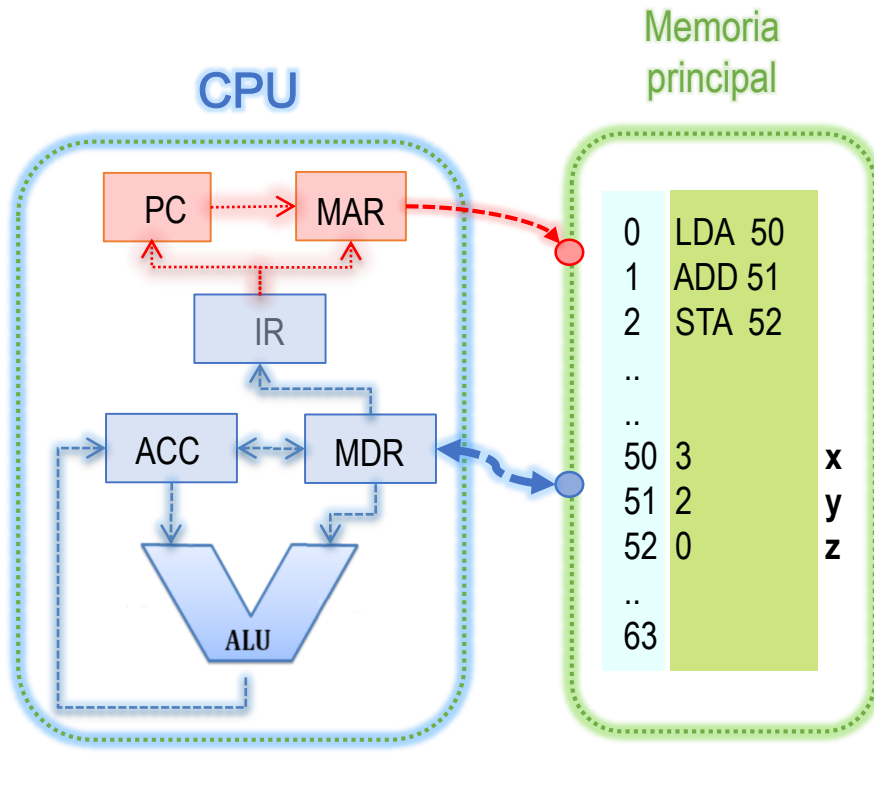
```
Int z =0, x=3, y=2;
z = x + y;
```

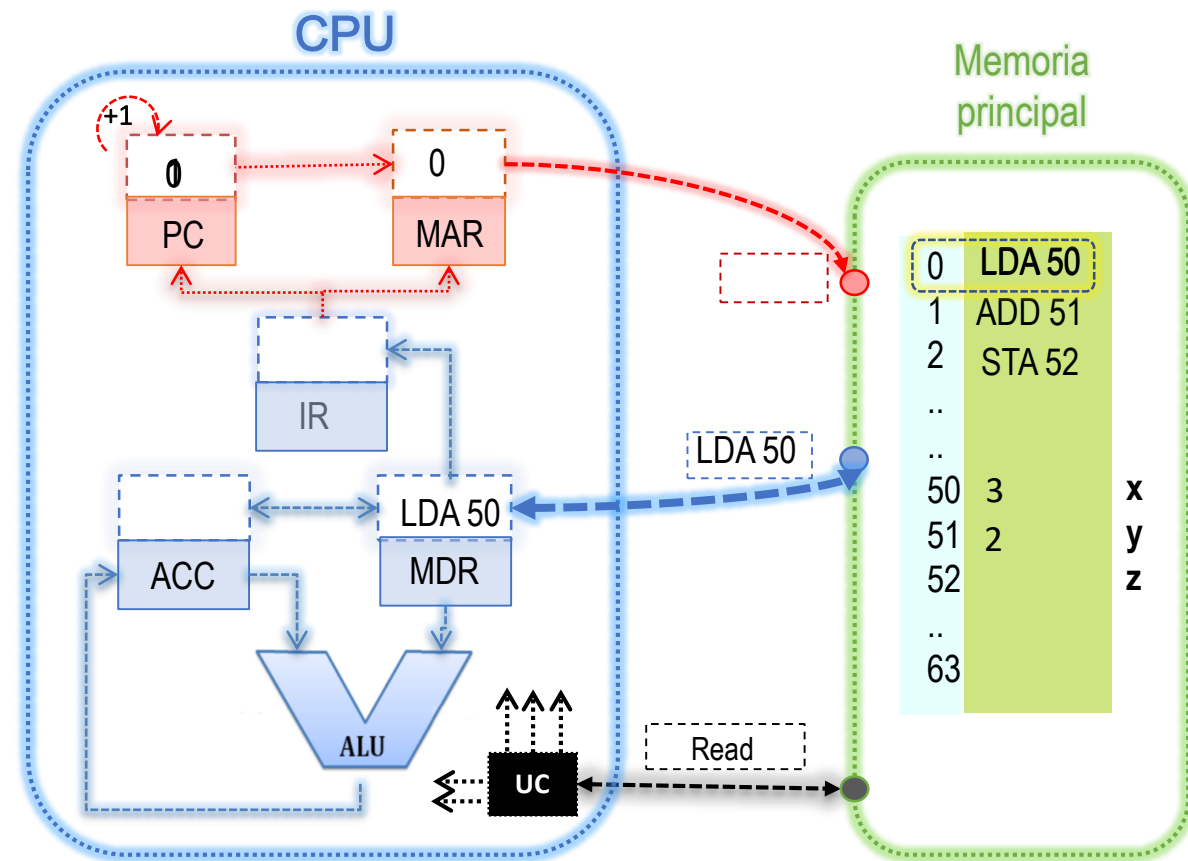
❑ **Ensamblador:**

```
LDA x
ADD y
STA z
```

.data

```
x db 3
y db 2
z db 0
```



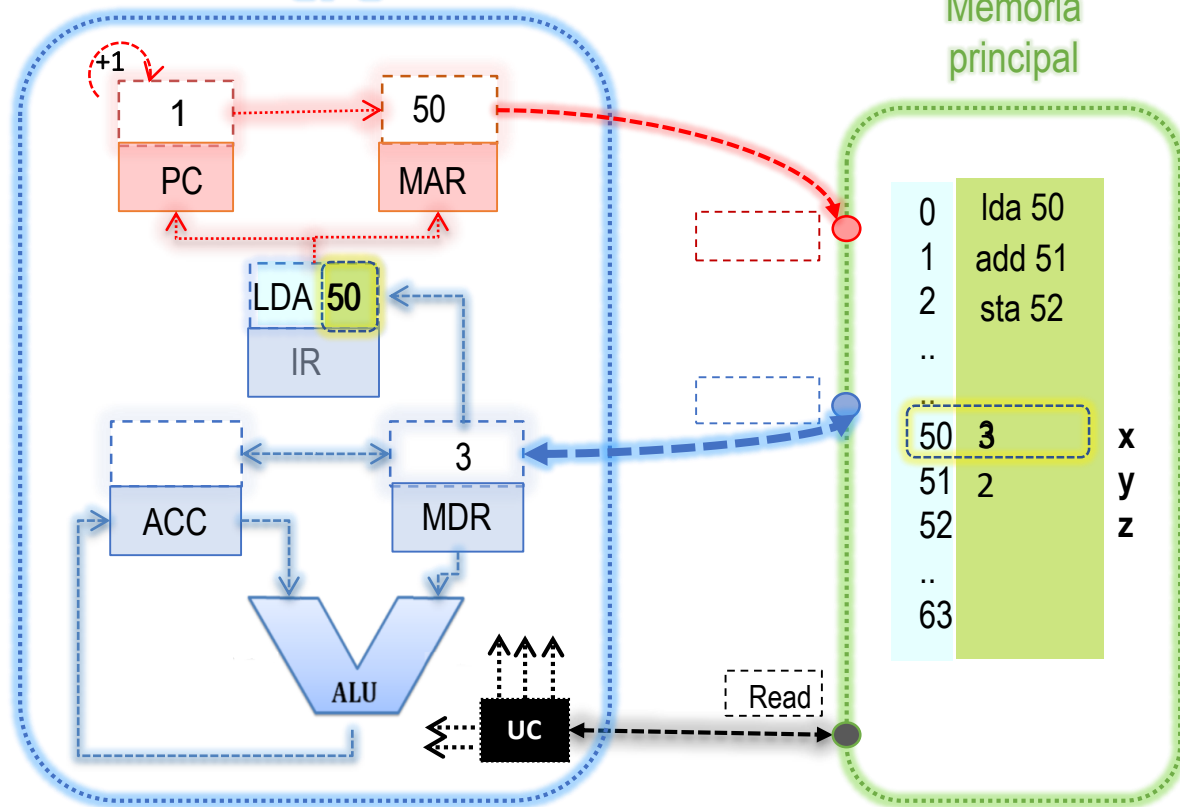


Etapa de captación

La secuencia de pasos de la **unidad de control (UC)** para captar una instrucción.

- 1: $MAR \leftarrow PC$
- 2: $MDR \leftarrow \text{read}(\text{Memoria}[MAR])$;
 $PC \leftarrow PC + 1$
- 3: $IR \leftarrow MDR$

CPU



Memoria principal

0	lda 50
1	add 51
2	sta 52
..	
50	3
51	2
52	
..	
63	

x
y
z

Etapa de ejecución

La secuencia de pasos de la **unidad de control (UC)** para ejecutar una instrucción.

Cada tipo de instrucción tiene sus **propios pasos de ejecución**.

1: $MAR \leftarrow IR(\text{dirección})$

2: $MDR \leftarrow \text{read}(\text{Memoria}[MAR])$

3: $ACC \leftarrow MDR$



Ejecución

Captación

1: $MAR \leftarrow PC$
2: $MDR \leftarrow \text{read}(\text{Memoria}[MAR])$;
 $PC \leftarrow PC + 1$
3: $IR \leftarrow MDR$

Cargar: **LDA**

Decodificar

Guardar: **STA**

1: $MAR \leftarrow IR(\text{dirección})$
2: $MDR \leftarrow \text{read}(\text{Memoria}[MAR])$
3: $ACC \leftarrow MDR$

Sumar: **ADD**

1: $MAR \leftarrow IR(\text{dirección})$
2: $MDR \leftarrow \text{read}(\text{Memoria}[MAR])$
3: $ACC \leftarrow ACC + MDR$

1: $MAR \leftarrow IR(\text{dirección})$
2: $MDR \leftarrow ACC$
3: $\text{write}(\text{Memoria}[MAR]) \leftarrow MDR$

Preguntas?