



Universidad Nacional  
de Entre Ríos

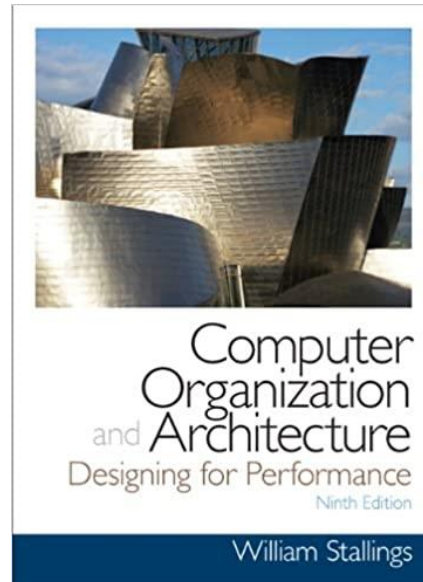
Tecnicatura universitaria en desarrollo web

# 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

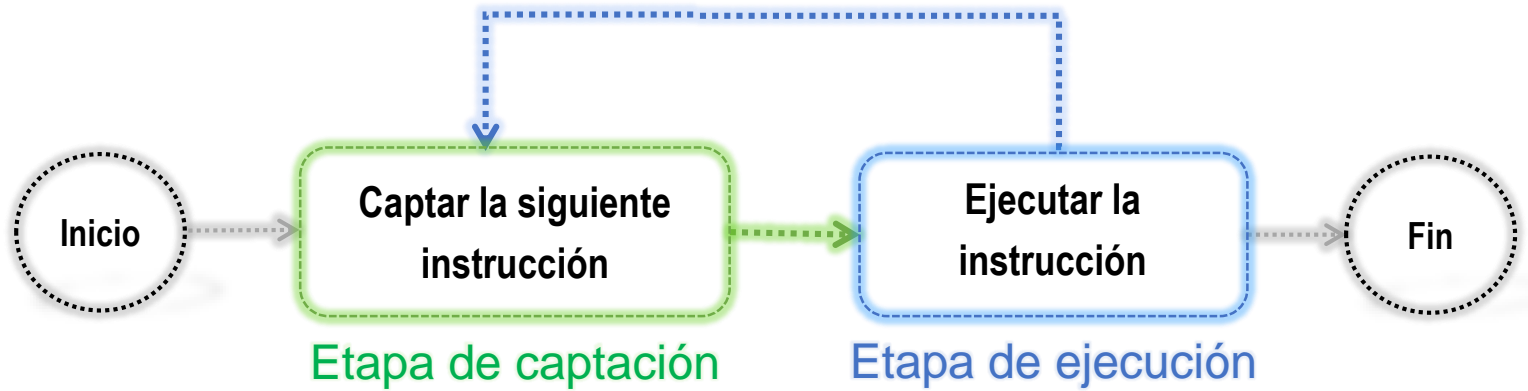
### ➤ Ciclo de la instrucción



- 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

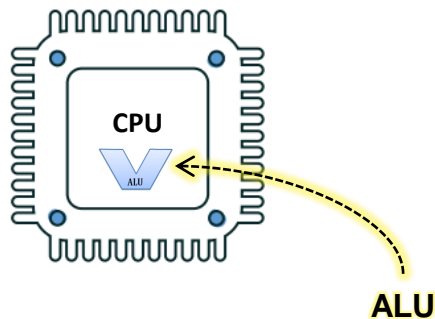
❑ Código maquina:

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
```



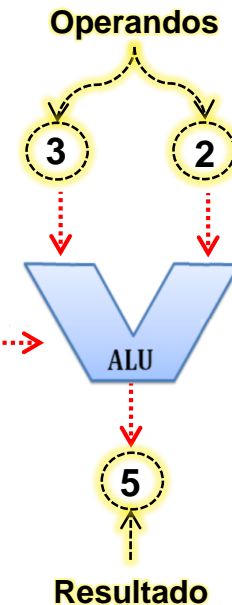
□ **C:**

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

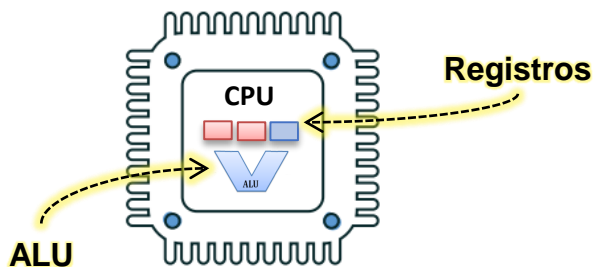
□ **Ensamblador:**

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

**Código operación** → **add**



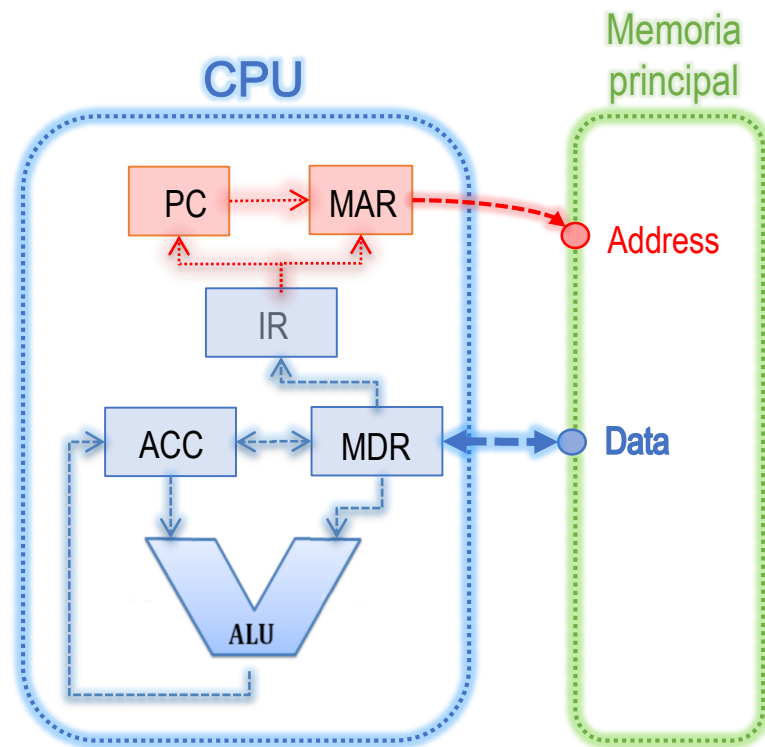
¿de dónde salen los datos **x e y**?, ¿dónde están ese 3, ese 2 y el número que representa la operación de suma? En algún lado tienen que estar almacenados, ¿no?



## 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:

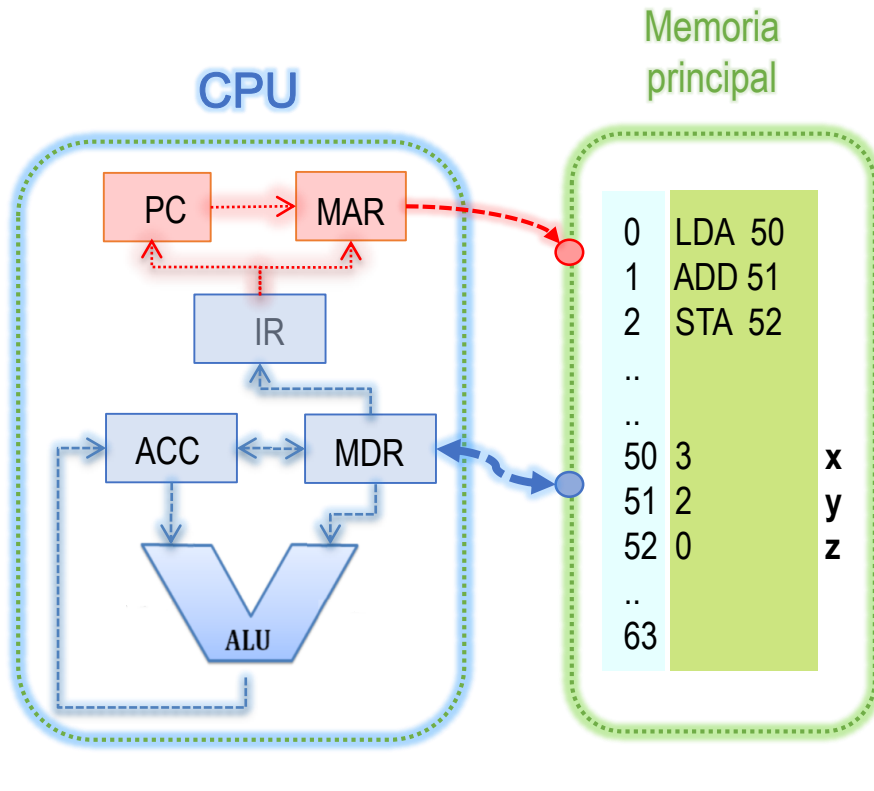
```
int z = 0, x = 3, y = 2;
z = x + y;
```

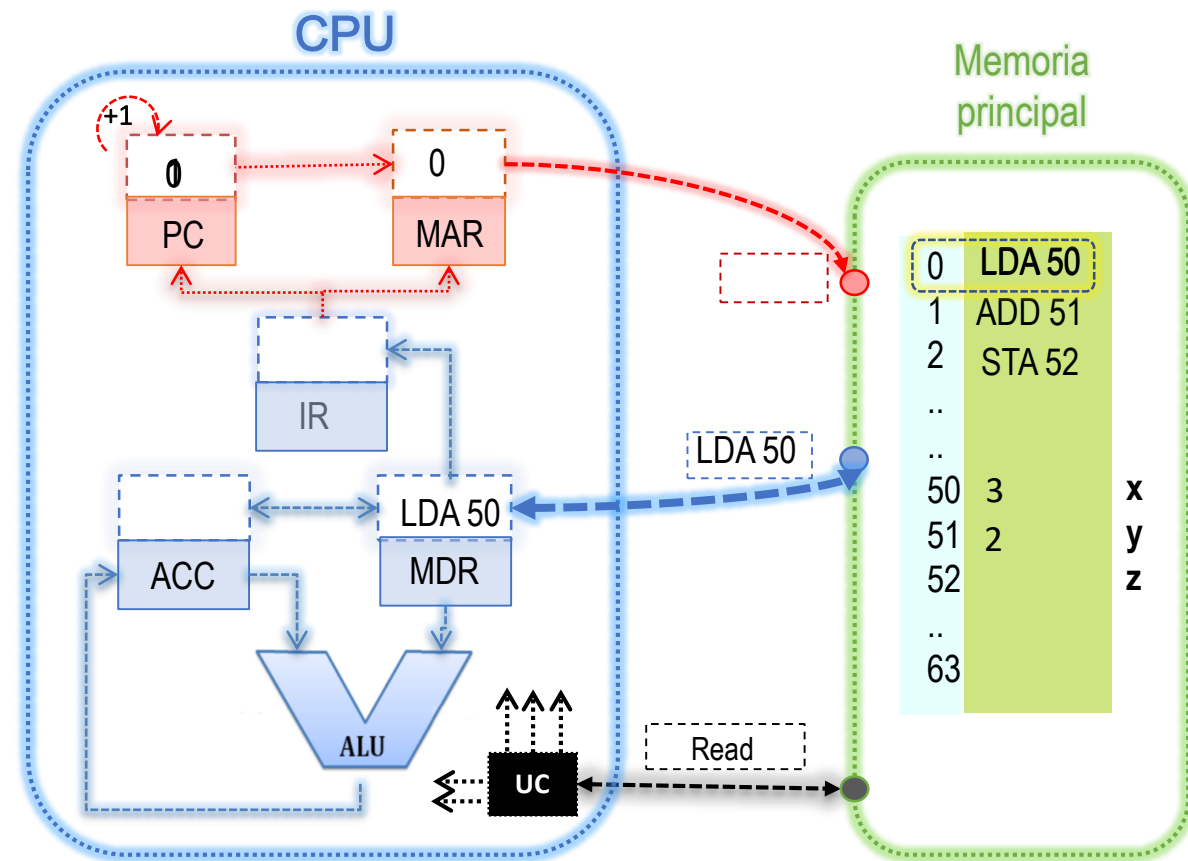
❑ Ensamblador:

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

.data

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

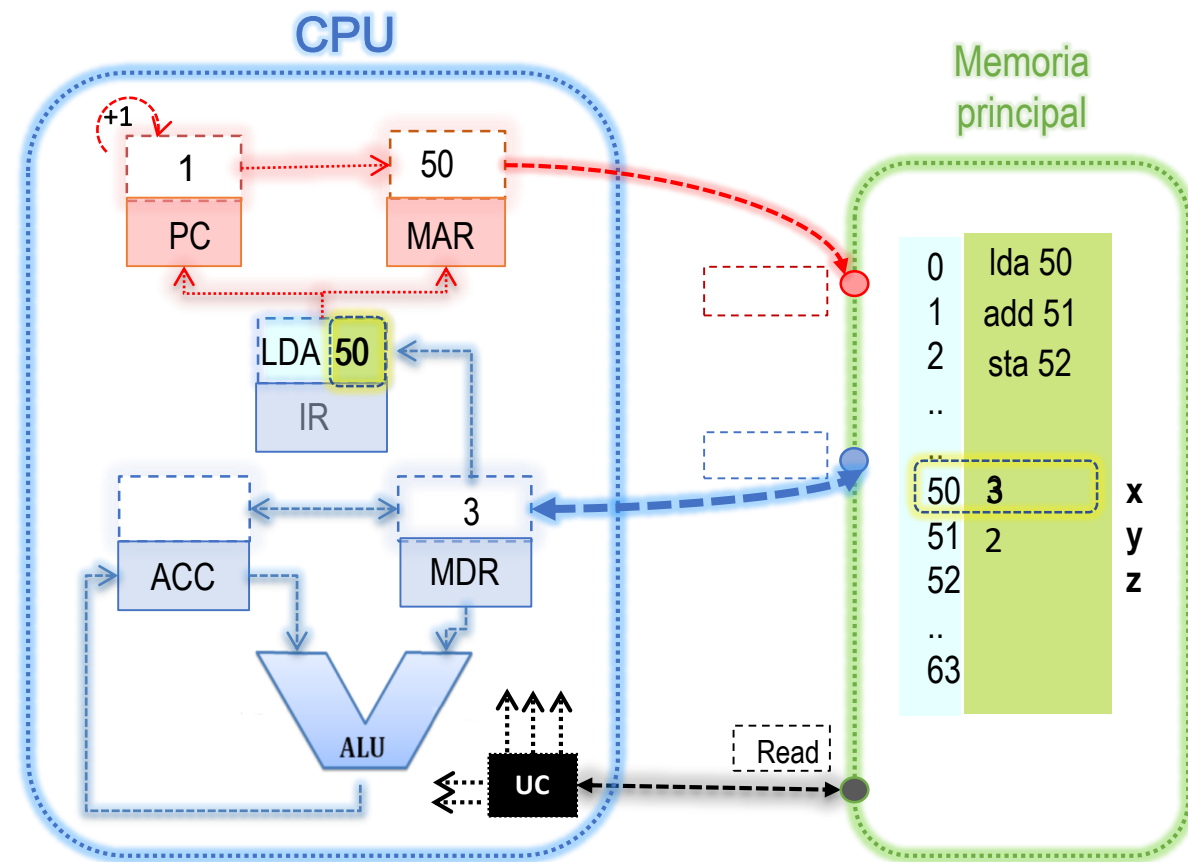




## Etapa de captación

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:  $IR \leftarrow MDR$



## 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 ← PC
2: MDR ← read(Memoria[MAR]) ;
   PC ← PC + 1
3: IR ← MDR
```

Cargar: **LDA**

Decodificar

Guardar: **STA**

```
1: MAR ← IR(dirección)
2: MDR ← read(Memoria[MAR])
3: ACC ← MDR
```

Sumar: **ADD**

```
1: MAR ← IR(dirección)
2: MDR ← read(Memoria[MAR])
3: ACC ← ACC + MDR
```

```
1: MAR ← IR(dirección)
2: MDR ← ACC
3: write(Memoria[MAR]) ← MDR
```



El CPU funciona a 20 Hz, entonces:

La duración de un ciclo de CPU es:  
 $1/\text{Hz} = 1/20 = \mathbf{0.05 \text{ segundos}}$

Captación

- 1 (0,05 segundos) :  $\text{MAR} \leftarrow \text{PC}$
- 2 (0,05 segundos) :  $\text{MDR} \leftarrow \text{read}(\text{Memoria}[\text{MAR}])$   
 $\text{PC} \leftarrow \text{PC} + 1$   
**Wait** (0,2 segundos =  $4 * 0,05 \text{ segundos}$ )
- 3 (0,05 segundos) :  $\text{IR} \leftarrow \text{MDR}$



La memoria RAM funciona a 10 Hz, entonces:

- Como la memoria RAM es más lenta el CPU debe esperar a que la memoria responda.
- La duración del ciclo de la memoria RAM es  $\rightarrow 1/\text{HZ} = 1/10 = \mathbf{0.1 \text{ segundos}}$ .
- Cada orden de lectura o escritura (read/write) a memoria RAM tarda 2 ciclos de RAM, entonces una operación en memoria tarda **0.2 segundos**.
- 0.2 segundos de una operación de memoria RAM representa 4 ciclos de CPU ( $0.05 \text{ ciclos de CPU} * 4 = 0.2 \text{ segundos}$ ), entonces el CPU espera (wait) por 4 ciclos cada vez que hay una operación de lectura o escritura en la memoria RAM

Ejecución

Cargar de memoria: LDA dirección

Decodificar

Guardar a memoria: STA dirección

- 1 (0,05 segundos) :  $\text{MAR} \leftarrow \text{IR}(\text{dirección})$
- 2 (0,05 segundos) :  $\text{MDR} \leftarrow \text{read}(\text{Memoria}[\text{MAR}])$   
**Wait** (0,2 segundos)
- 3 (0,05 segundos) :  $\text{ACC} \leftarrow \text{MDR}$

Sumar: ADD dirección

- 1 (0,05 segundos) :  $\text{MAR} \leftarrow \text{IR}(\text{dirección})$
- 2 (0,05 segundos) :  $\text{MDR} \leftarrow \text{read}(\text{Memoria}[\text{MAR}])$   
**Wait** (0,2 segundos)
- 3 (0,05 segundos) :  $\text{ACC} \leftarrow \text{ACC} + \text{MDR}$

- 1 (0,05 segundos) :  $\text{MAR} \leftarrow \text{IR}(\text{dirección})$
- 2 (0,05 segundos) :  $\text{MDR} \leftarrow \text{ACC}$
- 3 (0,05 segundos) :  $\text{write}(\text{Memoria}[\text{MAR}]) \leftarrow \text{MDR}$   
**Wait** (0,2 segundos)

# Preguntas?