Universidad Nacional de Entre Ríos

Tecnicatura universitaria en desarrollo web

Repertorio de instrucciones

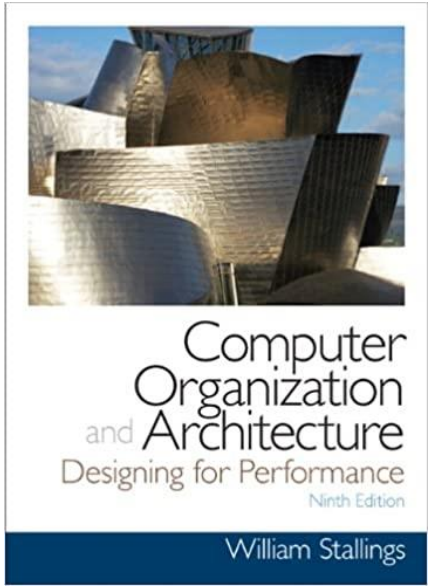
Semana 5 – Arquitectura de computadoras

Docente: José Ruiz {jose.ruiz@uner.edu.ar}

UNER virtual

Esta presentación esta basada en el libro de:

- William Stallings, Computer Organization and Architecture, 9th Edition, 2017.
  - Arquitectura acumulador: THE VON NEUMANN MACHINE: 2.1 / A BRIEF HISTORY OF COMPUTERS, pag 17.
  - Operandos de una instrucciones: CHAPTER 12 / INSTRUCTION SETS: CHARACTERISTICS AND FUNCTIONS, Number of Addresses, pag 410.



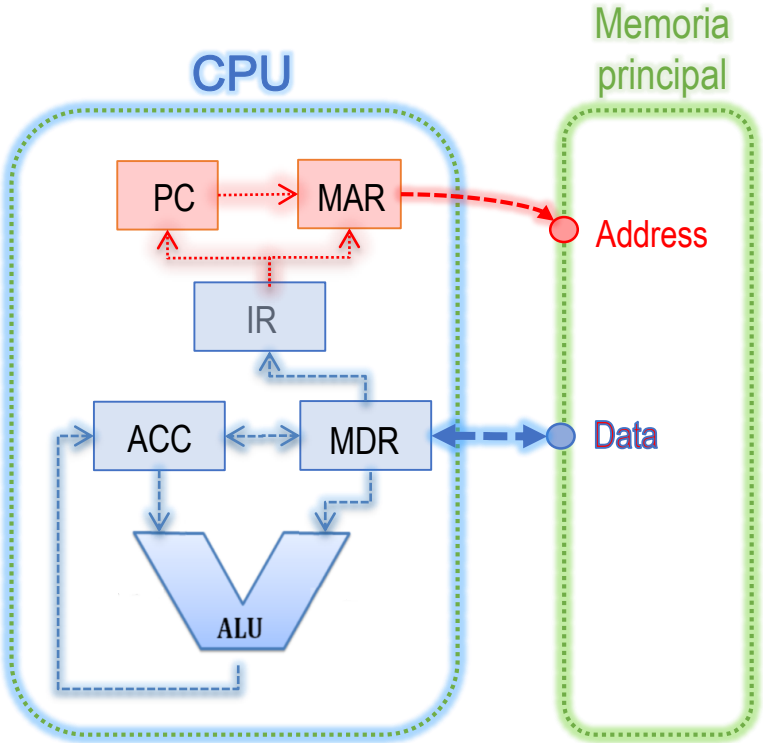
Archivos presentación y ejemplos se alojan en:

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

Repertorio de instrucciones

- Tipos de instrucciones
  - Banderas
- Modos de direccionamiento
- Operandos de la instrucción

Arquitectura acumulador (ACC)



- **Repertorio de instrucciones**

Código operación (C_7 C_6)	Nemónico		Efecto
0 = 00	LDA	dirección	ACC ← Memoria[dirección]
1 = 01	STA	dirección	Memoria[dirección] ← ACC
2 = 10	ADD	dirección	ACC ← ACC + Memoria[dirección]
3 = 11	HLT		Detiene CPU

Instrucciones de control

Traducción de la estructura de control «if» entre python y ensamblador

A la ALU se le agrega un flag o bandera que indica cuando el resultado cumple cierto requisito.

Python:

```
x = 3
y = 3
if x == y:
    print('x es igual que y')
```

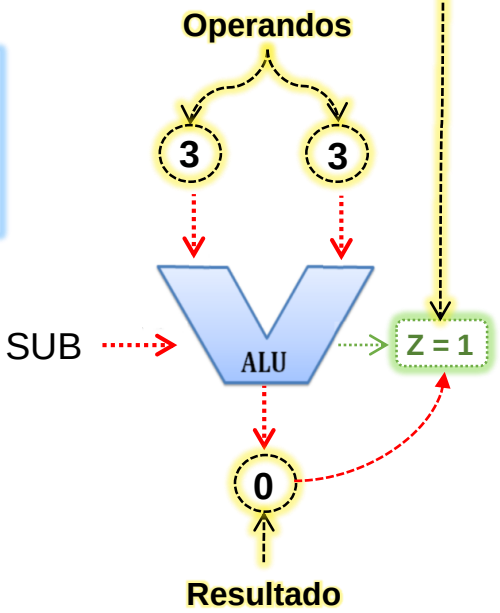
Se traduce

Ensamblador:

```
LDA x
SUB y
JMZ Print-Igual

Print-Igual:
```

```
.data
x db 3
y db 3
```

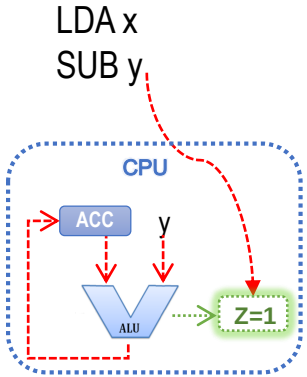
Una instrucción en **alto nivel** equivale a varias instrucciones en **bajo nivel**.



Flags - Banderas

Cero - (resultado=0) – Zero – Flag Z

1 byte = 8 bits, se puede representar 2^8 = 256 valores **(0-255)**



Si el resultado de la ALU es cero → el flag Zero = 1

Instrucciones de control

Traducción de la estructura de control «if» entre python y ensamblador

Python

```
x = 2
y = 3
if x < y:
    print('x es menor que y')
```

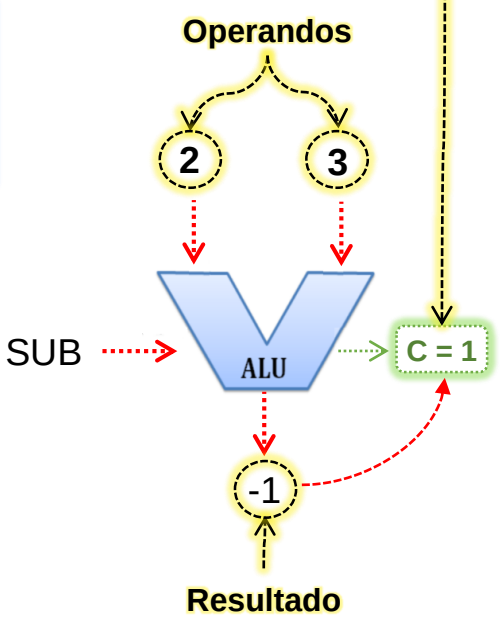
Se traduce

Acumulador

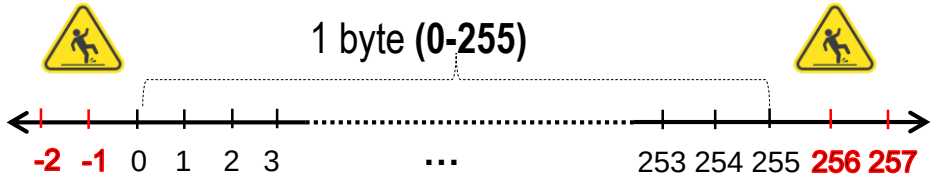
```
LDA x
SUB y
JMC Print-Menor

Print-Menor:
```

```
.data
x db 2
y db 3
```

A la ALU se le agrega un flag o bandera que indica cuando el resultado cumple cierto requisito.

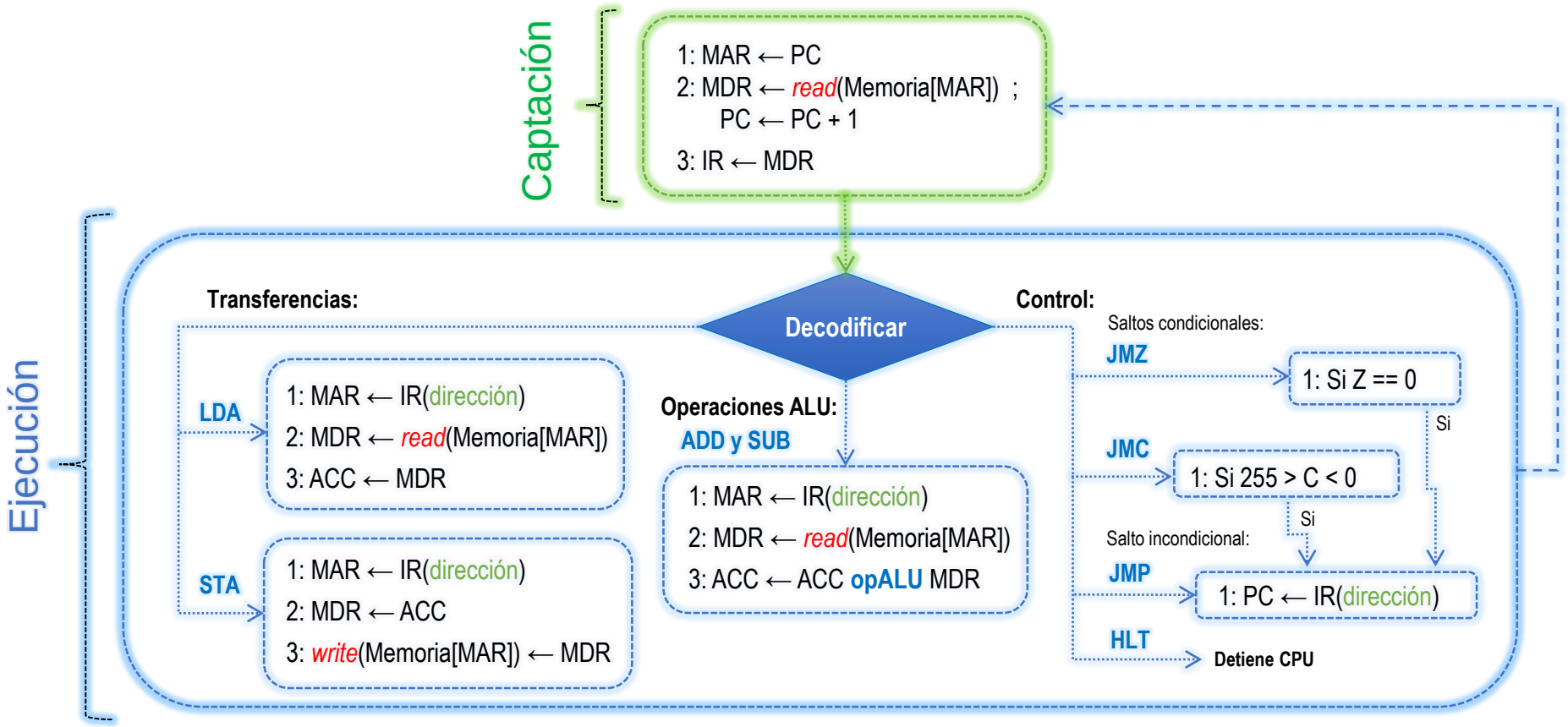


Acarreo – (o me llevo) – Carry – Flag C



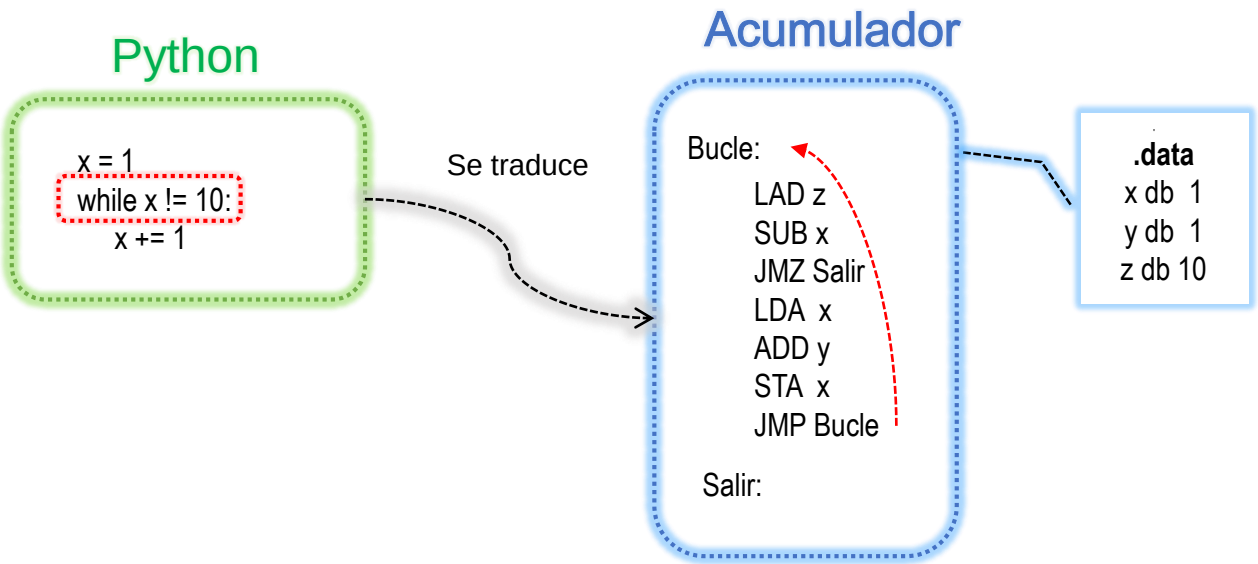
Se activa el Carry = 1: si el resultado es mayor o menor al rango soportado por el registro de 8 bits (0-255) → indica que el resultado es erróneo.

Ciclo de instrucción – Etapa de captación y ejecución



Instrucciones de control

Traducción de la estructura de control «while» entre python y ensamblador



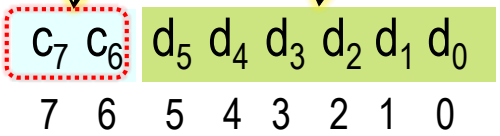
Tipos de instrucciones

Tipo de instrucciones

Tipo	Código operación / Nemónico	Efecto
Movimiento	0 = 000 → LDA dirección 1 = 001 → STA dirección	ACC ← Memoria[dirección] Memoria[dirección] ← ACC
Aritmética	2 = 010 → ADD dirección 3 = 011 → SUB dirección	ACC ← ACC + Memoria[dirección] ACC ← ACC – Memoria[dirección]
Control	4 = 100 → JMP dirección 5 = 101 → JMZ dirección 6 = 110 → JMC dirección 7 = 111 → HLT	PC ← dirección Si ACC == 0 entonces PC ← dirección Si 255 > ACC < 0 entonces PC ← dirección Detiene CPU

Hay que ampliar el código de operación **(c)**

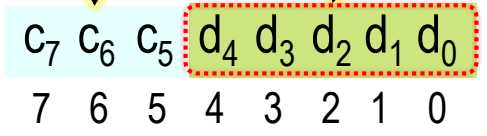
Dirección **(d)**



Pero la cantidad de posiciones direccionales se reduce $2^5 = 32$

Código de operación **(c)**

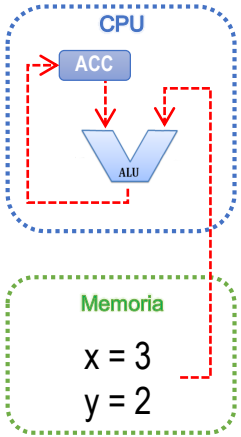
Dirección **(d)**



Operandos de la instrucción

Acumulador

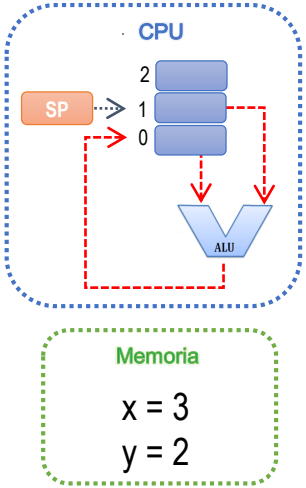
Las operaciones tienen como operando implícito el registro (ACC) y para el otro operando se proporciona su dirección en la memoria y el resultado se coloca en el ACC.



Stack

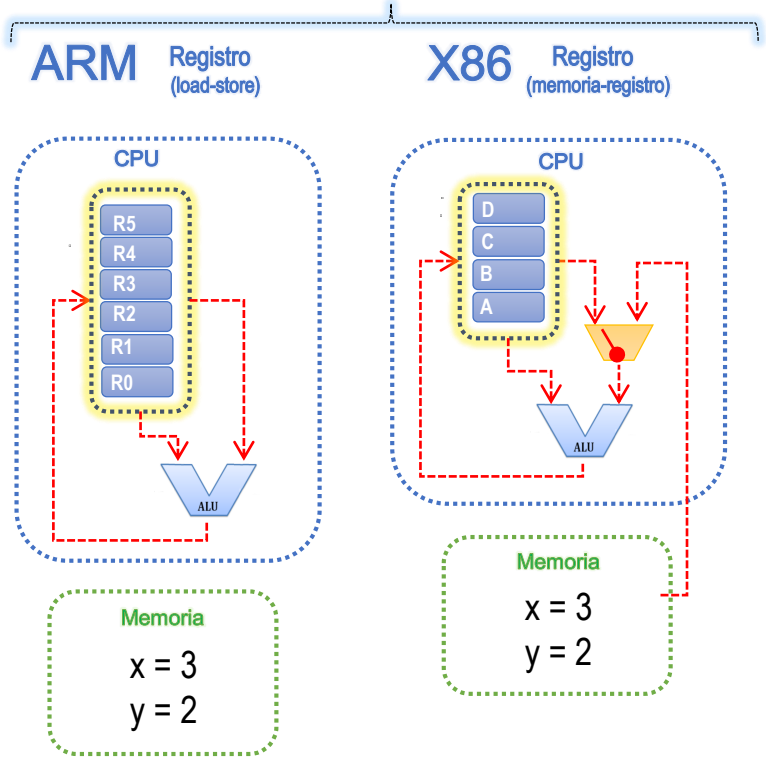
Cuenta con una pila y las operaciones se realizan sobre los elementos almacenados en ella accesibles desde el tope (SP).
Un stack o pila posee dos operaciones:

- **Push** que permite colocar un dato en la primera posición libre.
- **Pop** que permite retirar el último dato que se encuentra en la pila.



Registros (R)

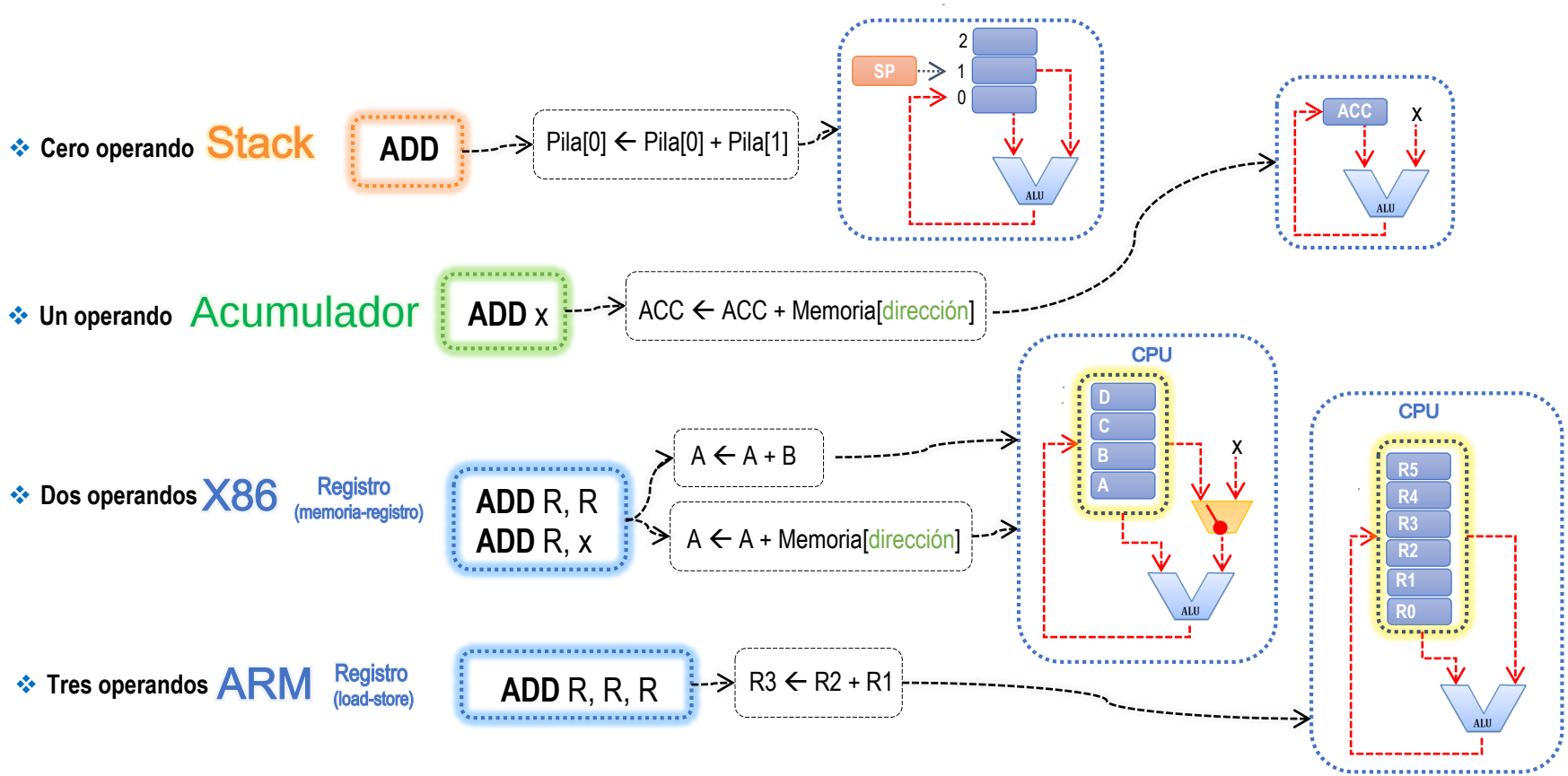
Se cuenta con un banco de registros y las operaciones se realizan entre ellos.



Operandos de la instrucción

Operandos de la instrucción

Se clasifican según el numero de operandos **explícitos** en la instrucción:



Operandos de la instrucción

Z = X+Y

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

Acumulador

```
LDA  x
STA  x
ADD x
```

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

Stack

```
PUSH  x
POP    x
ADD
```

```
PUSH x
PUSH y
ADD
POP z
```

```
PUSH x =>  Pila[0] = Memoria[x]
PUSH y =>  Pila[1] = Memoria[y]
ADD     =>   Pila[0] = Pila[0] + Pila[1]
POP z   =>   Memoria[z] = Pila[0]
```

Registros (R)

X86 Registro (memoria-registro)

```
MOV R, x
MOV x, R
ADD R, x
```

```
MOV A, x
ADD A, y
MOV w, A
```

ARM Registro (load-store)

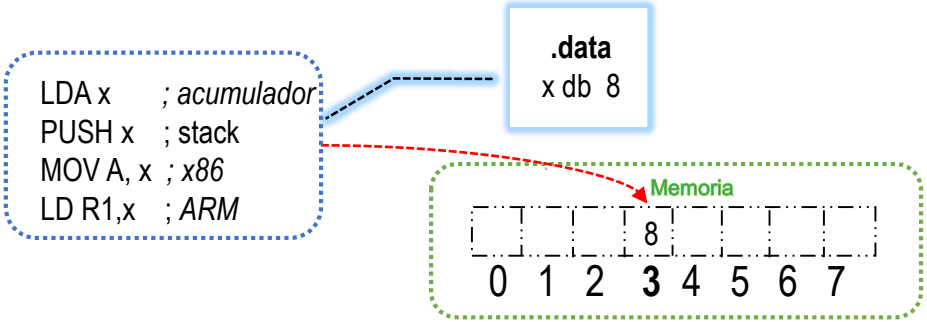
```
LD R, x
ST R, x
ADD R, R, R
```

```
LD R1, x
LD R2, y
ADD R3, R2, R1
ST  R3, z
```

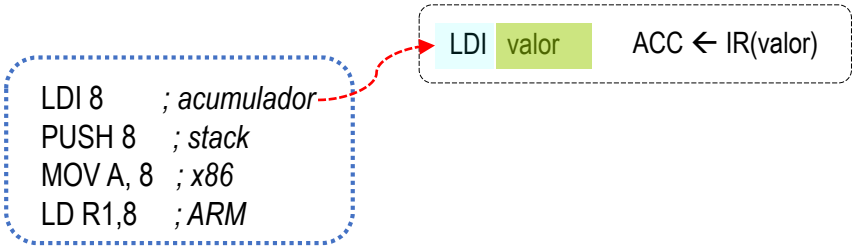
Modos de direccionamiento

- **Modos de direccionamiento:** Indican la manera que se obtienen los operandos de una instrucción.

  - **Direccionamiento directo**: dentro de la instrucción se encuentra la dirección del operando.

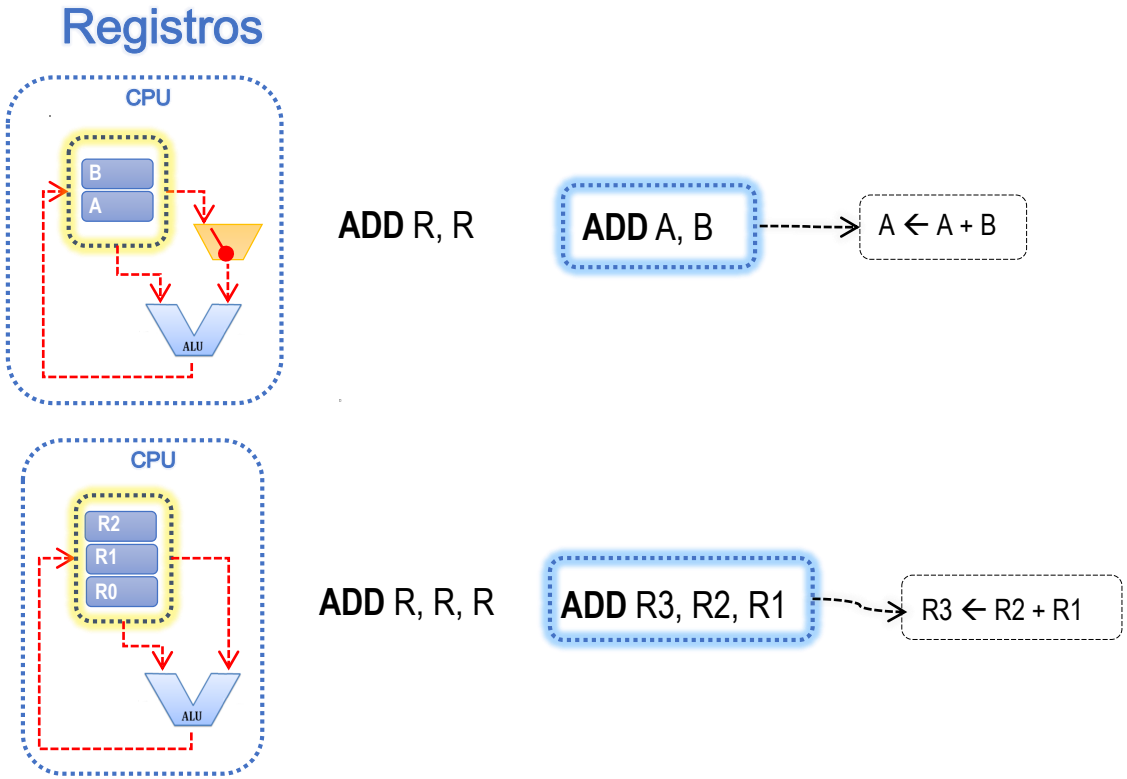


  - **Direccionamiento inmediato**: el valor del operando forma parte de la instrucción.



Modos de direccionamiento

- **Por registro:** solo intervienen los registros del procesador.



Ejercicio

W = Z - (X+Y)

```
.data
x db 3
y db 2
z db 7
w db 0
```

Acumulador

```
LDA  x
STA  x
ADD x
SUB x
```

```
LDA x
ADD y
STA t
LDA z
SUB t
STA w
```

Stack

```
PUSH  x
POP  x
ADD
SUB
```

```
PUSH z
PUSH x
PUSH y
ADD
SUB
POP w
```

ARM Registro (load-store)

```
LD R, x
ST x, R
ADD R, R, R
SUB R, R, R
```

```
LD R1, x
LD R2, y
ADD R3, R2, R1
LD R1, z
SUB R2, R1, R3
ST  R2, w
```

X86 Registro (memoria-registro)

```
MOV R, x
MOV x, R
ADD R, x
SUB R, x
MOV R, R
```

```
MOV A, x
ADD A, y
MOV B, A
MOV A, z
SUB A, B
MOV w, A
```

Preguntas?