

Conceptos de Arquitectura de Computadoras

Clase 1

Régimen de cursado

- Los Trabajos Prácticos son con registro de asistencia (Miércoles y Viernes).
- Cada Trabajo Práctico tendrá actividades en aula y en sala de PC.
- Aprobación
 - Promoción
 - Con Examen Final

Aprobación de cursada

- Se realizarán 2 evaluaciones parciales de trabajos prácticos con dos fechas de recuperación.
- Las 2 evaluaciones parciales deben ser Aprobadas.
- Las evaluaciones se tomarán en horarios de cursada.
- Para rendir las evaluaciones: 66 % de asistencia.
- En la segunda fecha de recuperación de evaluaciones, optarán por rendir el Parcial 1, el Parcial 2 o ambos.

Promoción

- Se deberá Aprobar, con valoración equivalente o superior a nota 4 (cuatro), una evaluación corta de teoría durante la cursada y en fecha definida en cronograma.
- Se deberán Aprobar, los 2 parciales de prácticas en la primera fecha prevista.
- Cumplidas las dos condiciones anteriores, se deberá Aprobar, con nota 6 (seis) o superior una evaluación teórica que se tomará antes del 2º recuperatorio de parciales de práctica. La nota obtenida es la nota de Promoción.
- Las evaluaciones se tomarán en horarios de cursada.

Bibliografía

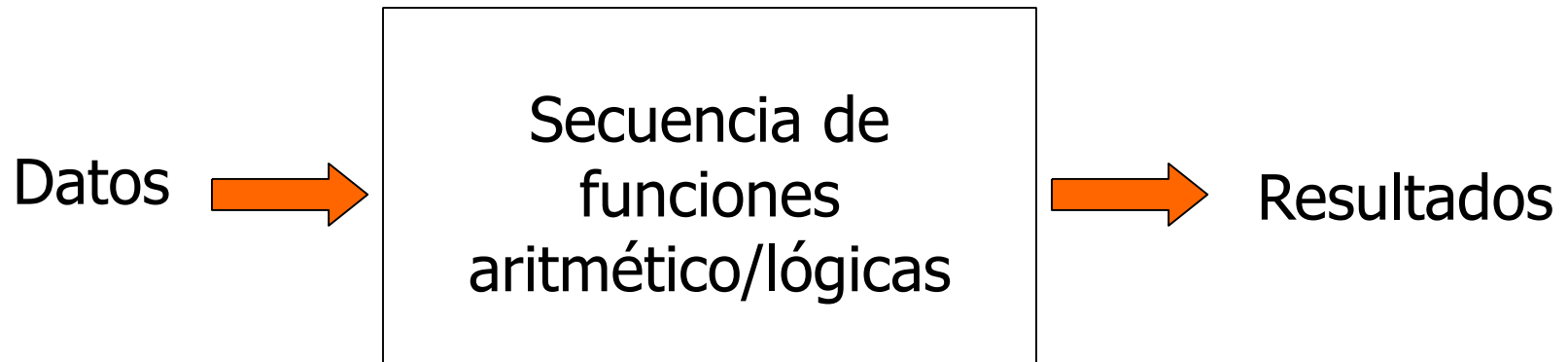
- ***Organización y Arquitectura de Computadoras – Diseño para optimizar prestaciones***, Stallings W., Editorial Prentice Hall.
- ***Organización de Computadoras***, Tanenbaum A., Editorial Prentice Hall.
- ***Arquitectura de Computadores - Un enfoque cuantitativo***, Hennessy & Patterson., Editorial Mc Graw Hill.
- ***Diseño y evaluación de arquitecturas de computadoras***, Beltrán M. y Guzmán A., Editorial Prentice Hall.
- ***Computer Organization and Embedded Systems***, 6th ed. Hamacher C., Vranesic Z., Zaky S., Manjikian N., Editorial Mc Graw Hill
- ***Computer Organization and Architecture, 10/E***. Stallings W., Editorial Pearson

Temas de clase

- Temas básicos
 - Programas
 - Arquitectura Von Neumann
 - Repertorio de instrucciones
 - Ciclo de instrucción
 - Simulador
- Subrutinas
 - Pasaje de argumentos

Concepto de programa

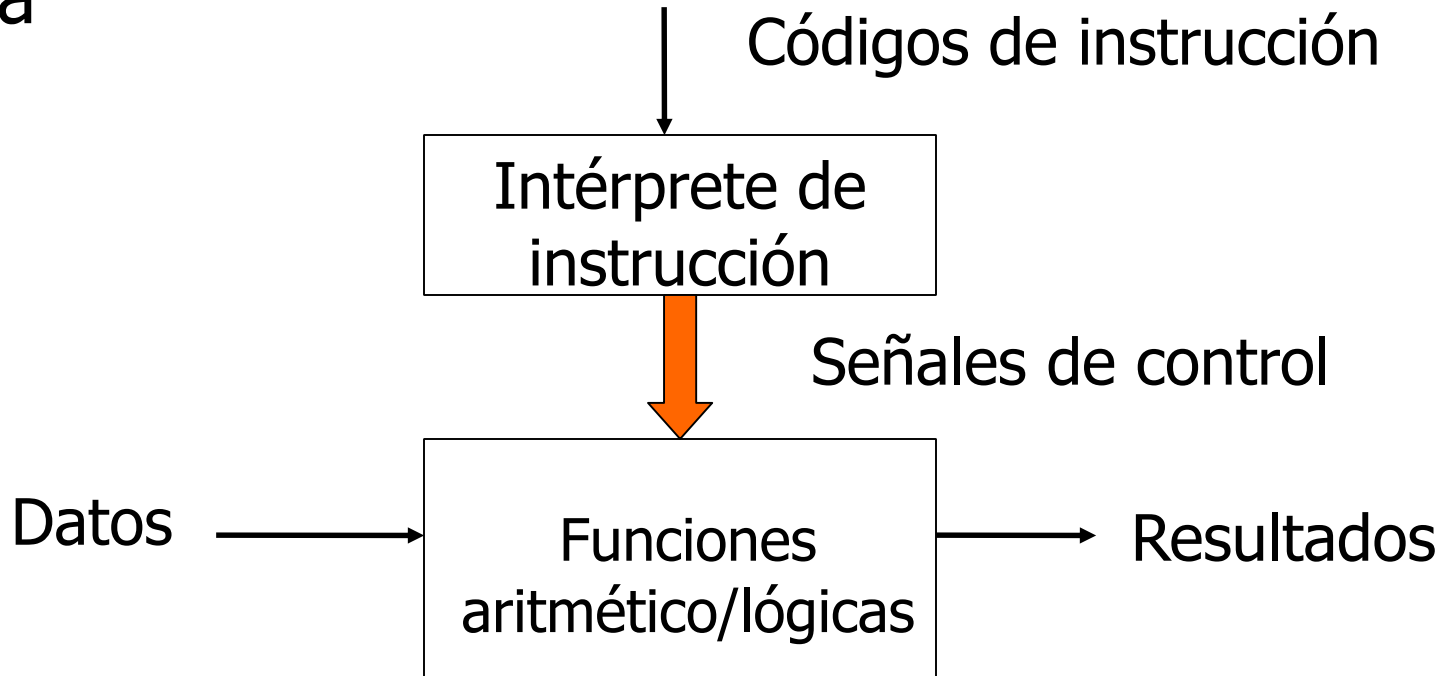
- Antes se tenían sistemas cableados



- Programación en hardware: cuando cambiamos las tareas, debemos cambiar el hardware

Concepto de programa (2)

- Ahora



- Programación en software: en cada paso se efectúa alguna operación sobre los datos

Concepto de programa (3)

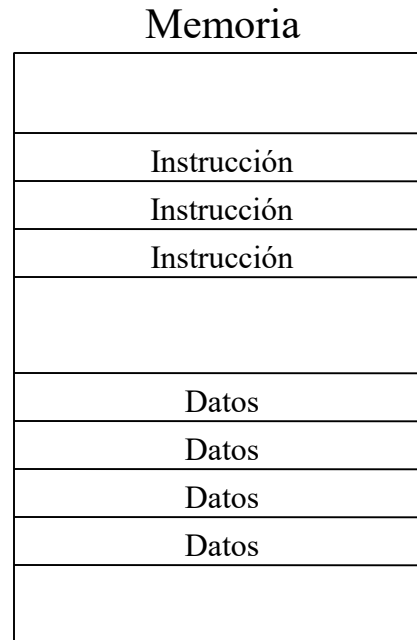
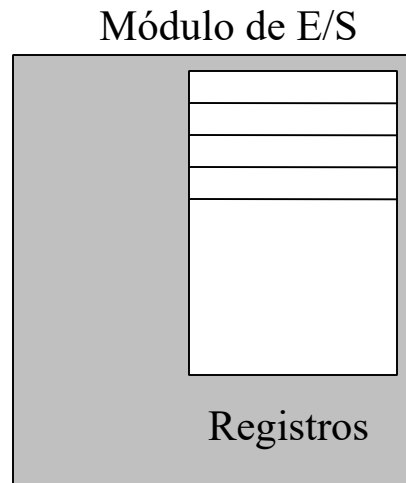
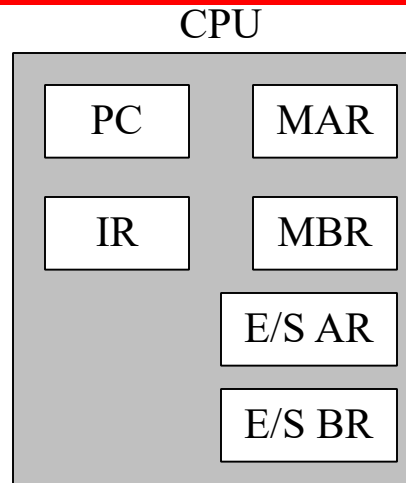
- Para cada paso se necesita un nuevo conjunto de señales de control.
- Las instrucciones proporcionan esas señales de control.
- Aparece el nuevo concepto de programación.

No hay que cambiar el hardware !!!

Arquitectura Von Neumann

- La unidad central de procesamiento (CPU) está constituida por la unidad de control (UC) y la unidad aritmético-lógica (ALU).
- Datos e instrucciones deben introducirse en el sistema y los resultados se proporcionarán mediante componentes de entrada/salida (E/S).
- Se necesita almacenar temporalmente datos e instrucciones:
 - Memoria Principal

Componentes de una computadora

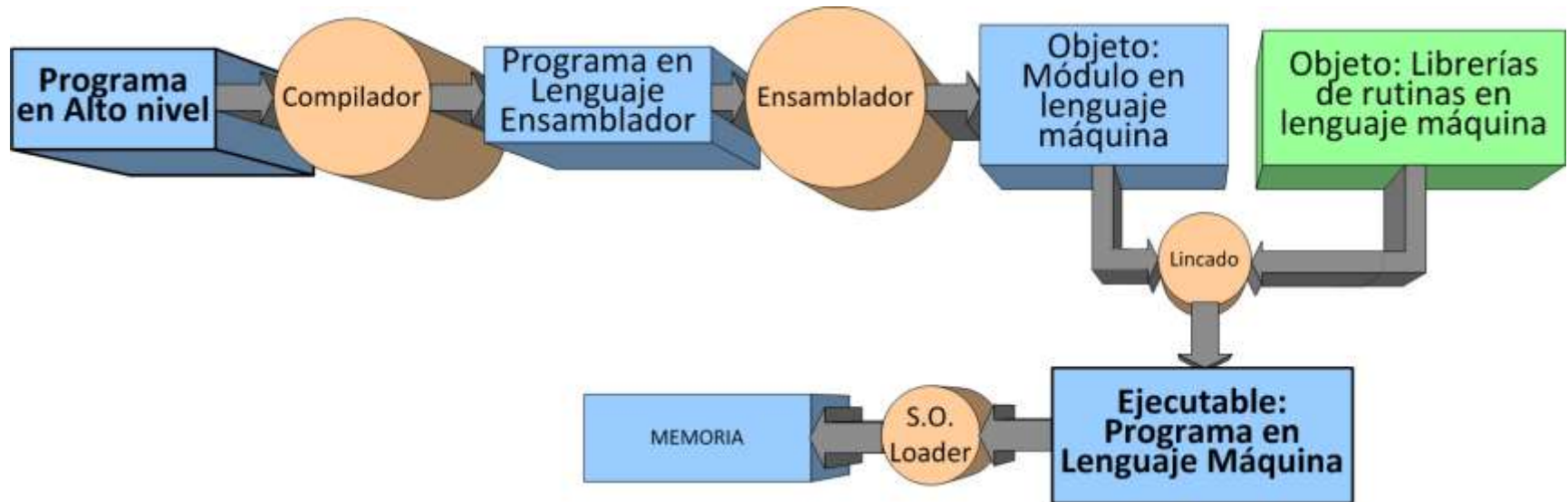


PC = Contador de programa
IR = Registro de instrucción
MAR = Registro de dirección de memoria
MBR = Registro de buffer de memoria
E/S AR = Registro de dirección de E / S
E/S BR = Registro buffer de E / S

Repertorio de instrucciones

- Es el conjunto completo de instrucciones que se realizan en una CPU.
 - Código máquina
 - Binario
- Representado simbólicamente por un conjunto de códigos de ensamblaje
 - de operaciones:
 - ADD (sumar), SUB (restar), LOAD (cargar datos en un registro)
 - de operandos:
 - ADD BX, PEPE; sumar contenidos de reg BX y dirección PEPE, el resultado se guarda en reg BX

Alto nivel a máquina



Elementos de una instrucción

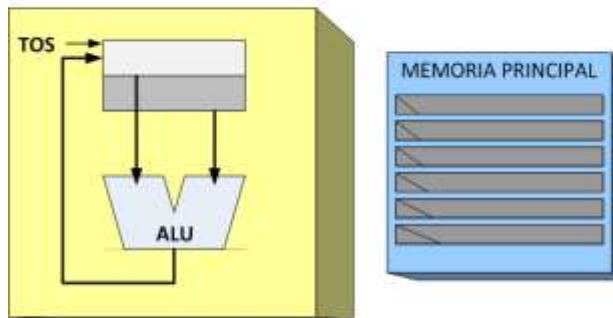
- Código de operación (“Cod Op”)
- Referencia a operandos fuentes
- Referencia al operando resultado
- Referencia a la siguiente instrucción

¿Dónde se almacenan operandos?

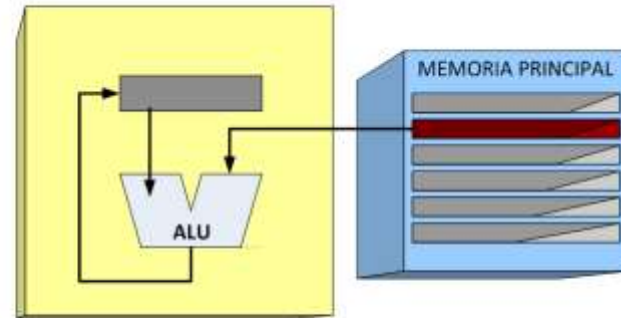
- Memoria principal
 - o memoria virtual o en memoria cache
- Registro de la CPU
- Dispositivo de E/S

Alternativas de almacenamiento

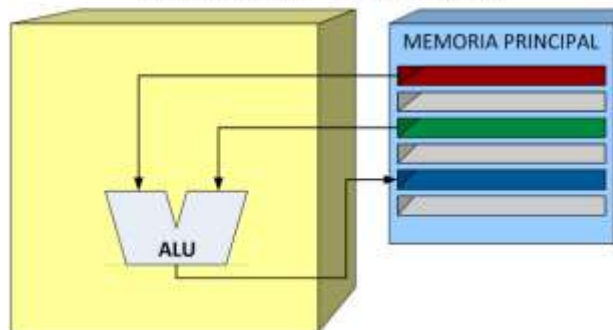
Almacenamiento tipo Pila



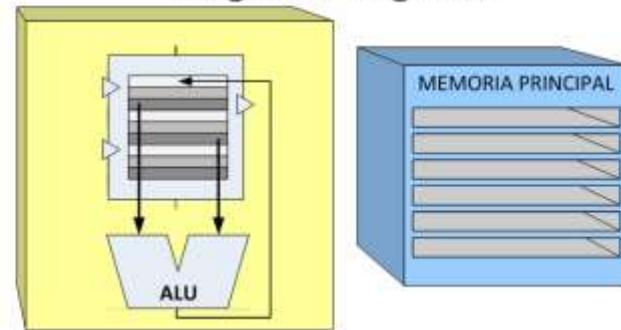
Almacenamiento tipo Acumulador



Almacenamiento tipo
Memoria - Memoria



Almacenamiento tipo
Registro-Registro



Tipos de instrucciones

- Procesamiento de datos:
 - instrucciones aritmético-lógicas
- Almacenamiento de datos:
 - instrucciones de memoria
- Transferencia de datos:
 - instrucciones de E/S
- Control:
 - instrucciones de testeo y flujo del programa

¿Cuántas direcciones?

- Más direcciones por instrucción
 - Instrucciones más complejas
 - Más registros:
 - Las operaciones entre los registros son más rápidas.
 - Menos instrucciones por programa
- Menos direcciones por instrucción
 - Instrucciones menos complejas
 - Más instrucciones por programa
 - La captación/ejecución de las instrucciones es más rápida

Decisiones en el diseño del conjunto de instrucciones (1)

- Tipos de operandos (datos)
- Repertorio de operaciones
 - ¿Cuántas operaciones se considerará?
 - ¿Cuáles operaciones se realizarán?
 - ¿Cuán compleja será cada una de ellas?
- Formatos de instrucciones:
 - Longitud de instrucción
 - Número de direcciones
 - Tamaño de los campos

Decisiones en el diseño del conjunto de instrucciones (2)

- Registros
 - Número de registros de la CPU referenciables
 - ¿En qué registros se pueden ejecutar qué operaciones?
- Modos de direccionamiento
 - ¿cómo es especificada la ubicación de un operando o una instrucción?
- **RISC** contrapuesto a **CISC**
(Computadora de conjunto reducido de instrucciones) a
(Computadora de conjunto complejo de instrucciones)

Tipos de operandos

- Direcciones
- Números
 - punto fijo ó punto flotante
- Caracteres
 - ASCII, EBCDIC ...etc.
- Datos lógicos
 - Bits (1 ó 0)
Ej: flags o indicadores

Orden de los bytes

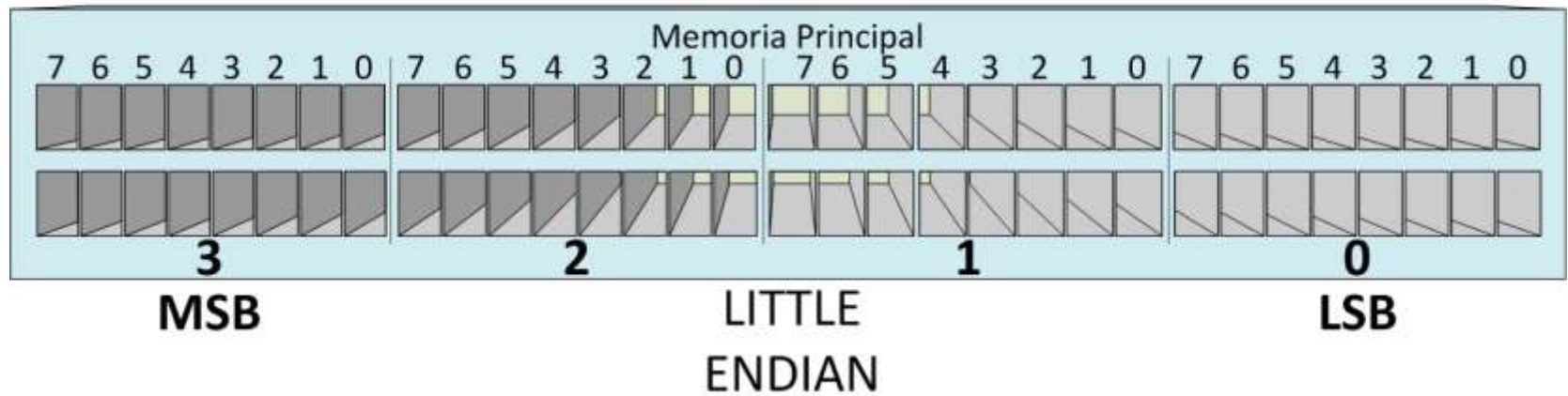
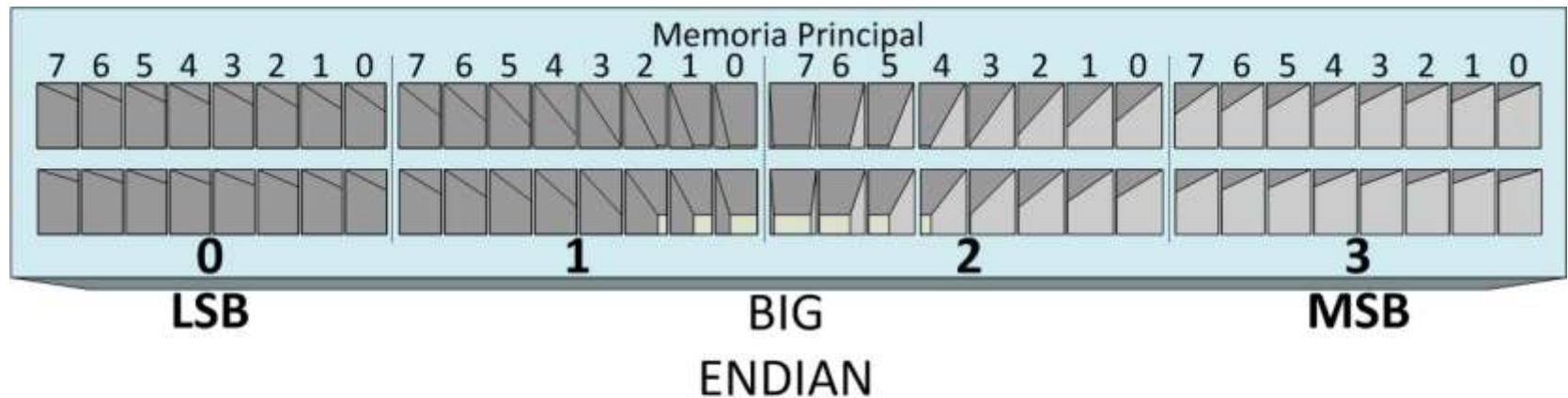
Supongamos una memoria direccionable de a byte

- ¿En qué orden se leen aquellos números que ocupan más de un byte?

Ejemplo:

La palabra doble 98765432H (32 bits) se puede almacenar en 4 bytes consecutivos de las siguientes 2 formas:

Orden de los bytes (2)



Orden de los bytes (3)

Dir. de byte	Forma 1	Forma 2
00	98	32
01	76	54
02	54	76
03	32	98

¿cuál forma uso?

Big endian: el byte más significativo en la dirección con valor numérico más bajo

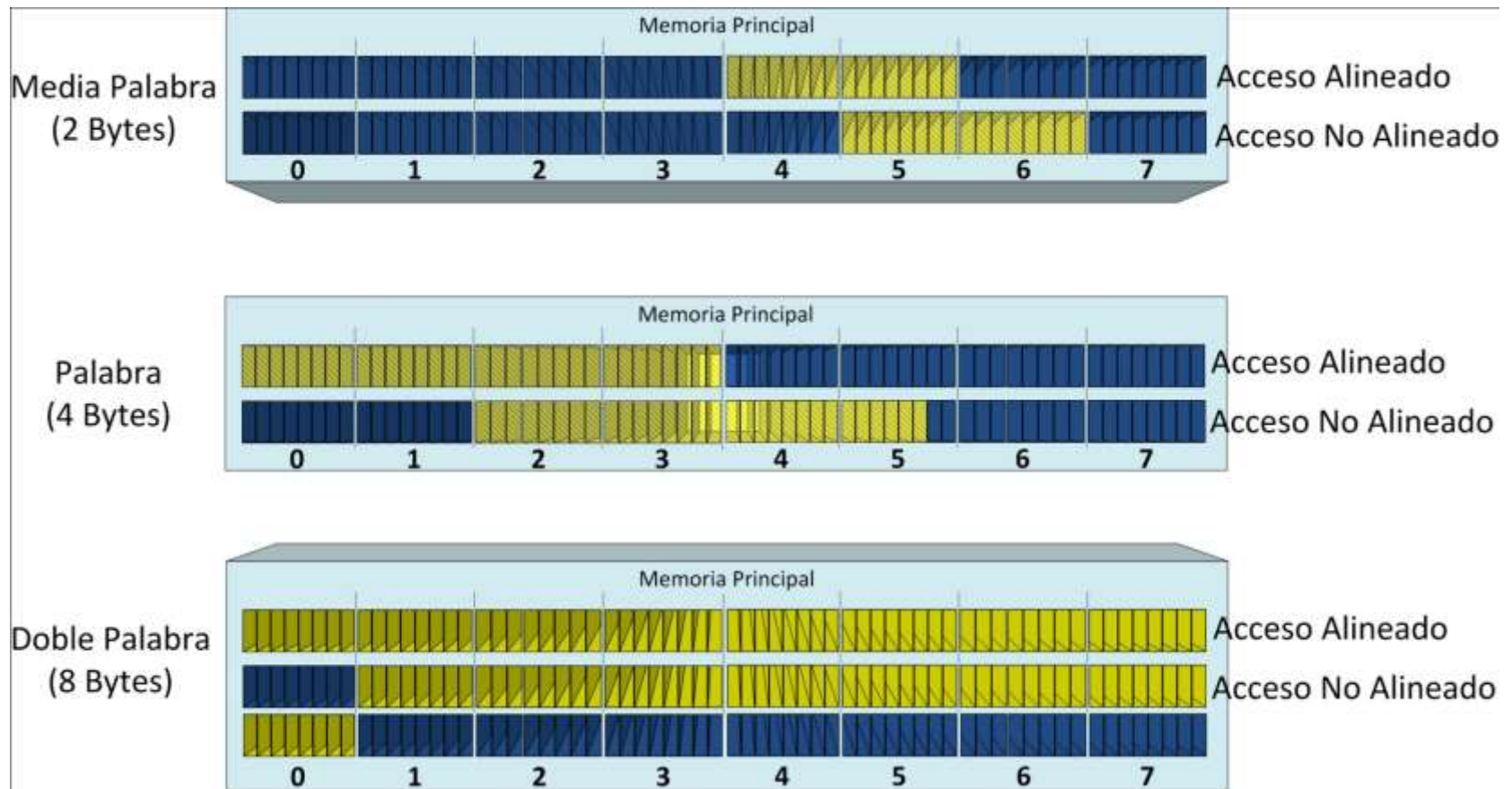
Little endian: el byte menos significativo en la dirección con valor numérico más bajo

Problema

- Intel 80x86, Pentium y VAX son “little-endian”.
- IBM S/370, Motorola 680x0 (Mac) y la mayoría de los RISC son “big-endian”.

Incompatibilidad !!!

Accesos a la memoria



Problema

- Si se permiten, los accesos no alineados son mas lentos!!!!

Tipos de operaciones

- Transferencias de datos
- Aritméticas
- Lógicas
- Conversión
- Entrada/Salida
- Control del sistema
- Control de flujo

Transferencia de datos

- Debe especificarse:
 - Ubicación del operando fuente
 - Ubicación del operando destino
 - Tamaño de los datos a ser transferidos
 - Modo de direccionamiento
- Diferentes movimientos -> diferentes instrucciones
 - Reg-Reg, Reg-Mem o Mem-Reg
- O una instrucción y diferentes direcciones
 - MOV destino, fuente ; copia fuente a destino

Aritméticas

- Operaciones básicas:
Add, **Sub**tract, **M**ultiply y **D**ivide
 - Números enteros sin/con signo.
 - ¿Números en punto flotante?
- Pueden incluirse otras operaciones ...
 - **I**ncrement o **D**ecrement (en 1 el operando)
 - **N**egate: cambia el signo del operando (Ca2).
 - **A**bsolute: toma el valor absoluto del operando.
 - Shift left/right: desplaza bits a izq/der un lugar

Lógicas - Conversión

Operaciones que manipulan bits individualmente

- Operaciones Booleanas.
AND, OR, XOR, NOT
- Otras operaciones
 - Rotate left/right: rota las posiciones de los bits a izq/der

Operaciones para cambiar formatos de datos

- Conversión de binario a decimal o de EBCDIC a ASCII

Entrada/Salida

- Pocas instrucciones pero de acciones específicas
 - IN ó OUT
- Se pueden realizar utilizando instrucciones de movimiento de datos
 - MOVE
- Se pueden realizar a través de un controlador aparte: DMA (Direct Memory Access)

Control de flujo

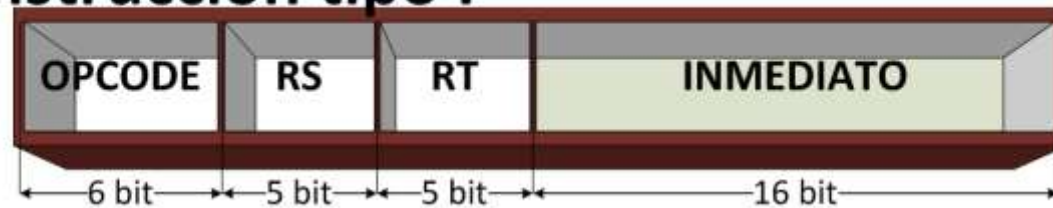
Modifican el valor contenido en el registro PC

- Salto Incondicional
 - **JMP** equis ; saltar a la posición 'equis'
- Salto Condicional
 - **JZ** equis ; saltar a la posición 'equis', si bandera Z=1
- Salto con retorno o llamada a subrutina
 - **CALL** subrut ; saltar a la posición 'subrut'

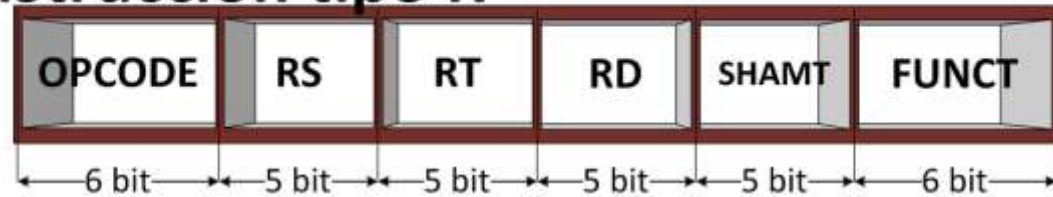
Para retornar al programa que llamó, se debe utilizar la instrucción **RET** como última instrucción del cuerpo de subrutina

Formatos de instrucción

Instrucción tipo I



Instrucción tipo R



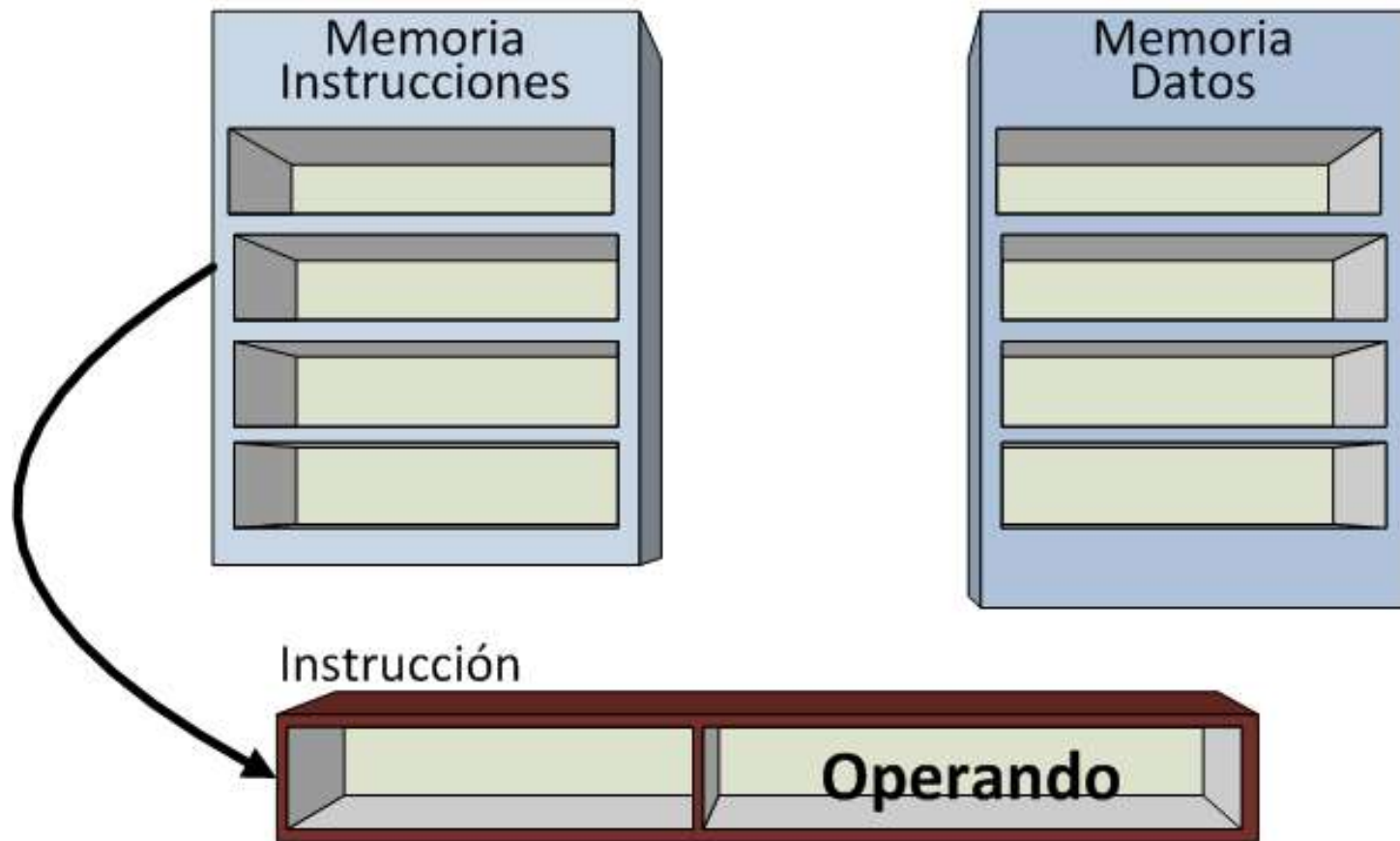
Instrucción tipo J



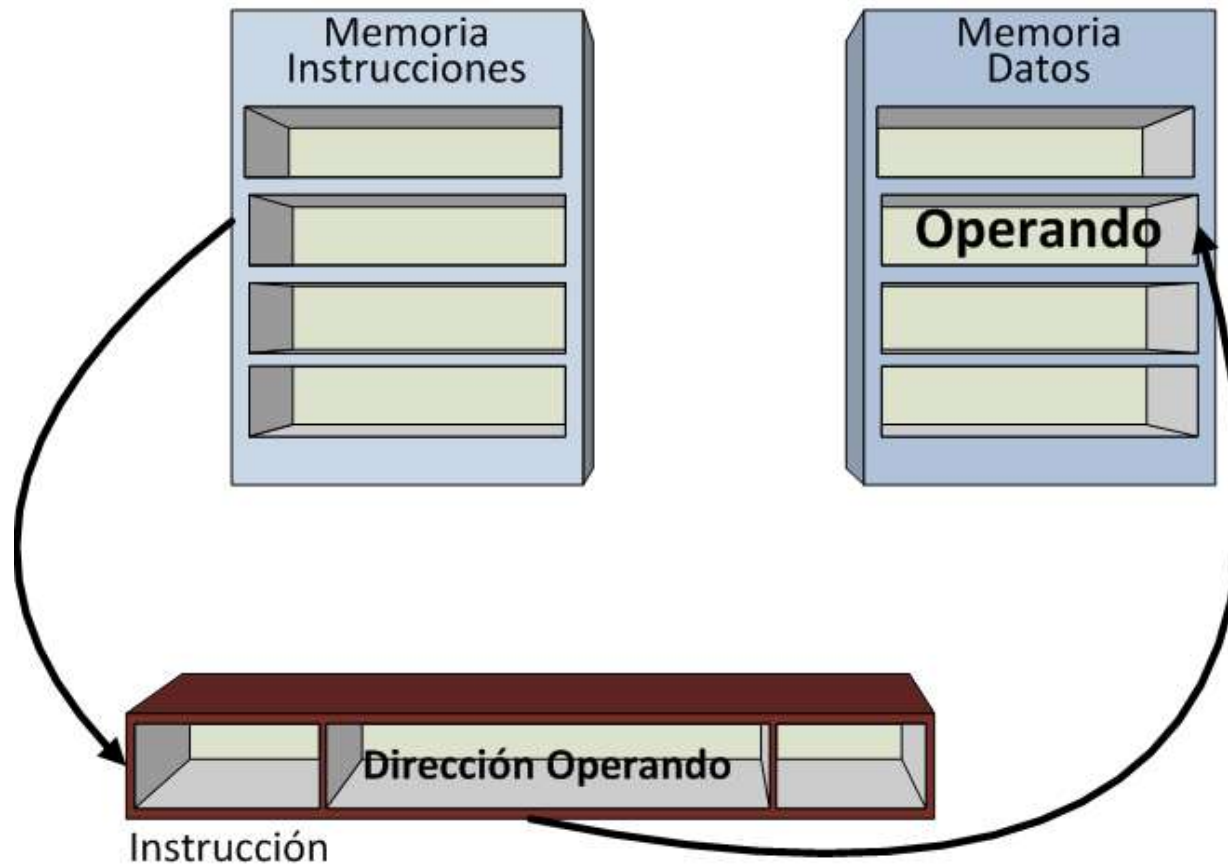
Modos de direccionamiento

- Inmediato
- Directo de memoria o Absoluto
- Directo de Registro
- Indirecto de memoria (en desuso)
- Indirecto con registro
- Indirecto con Desplazamiento
 - basado, indexado o relativo al PC
 - Pila (o relativo al SP)

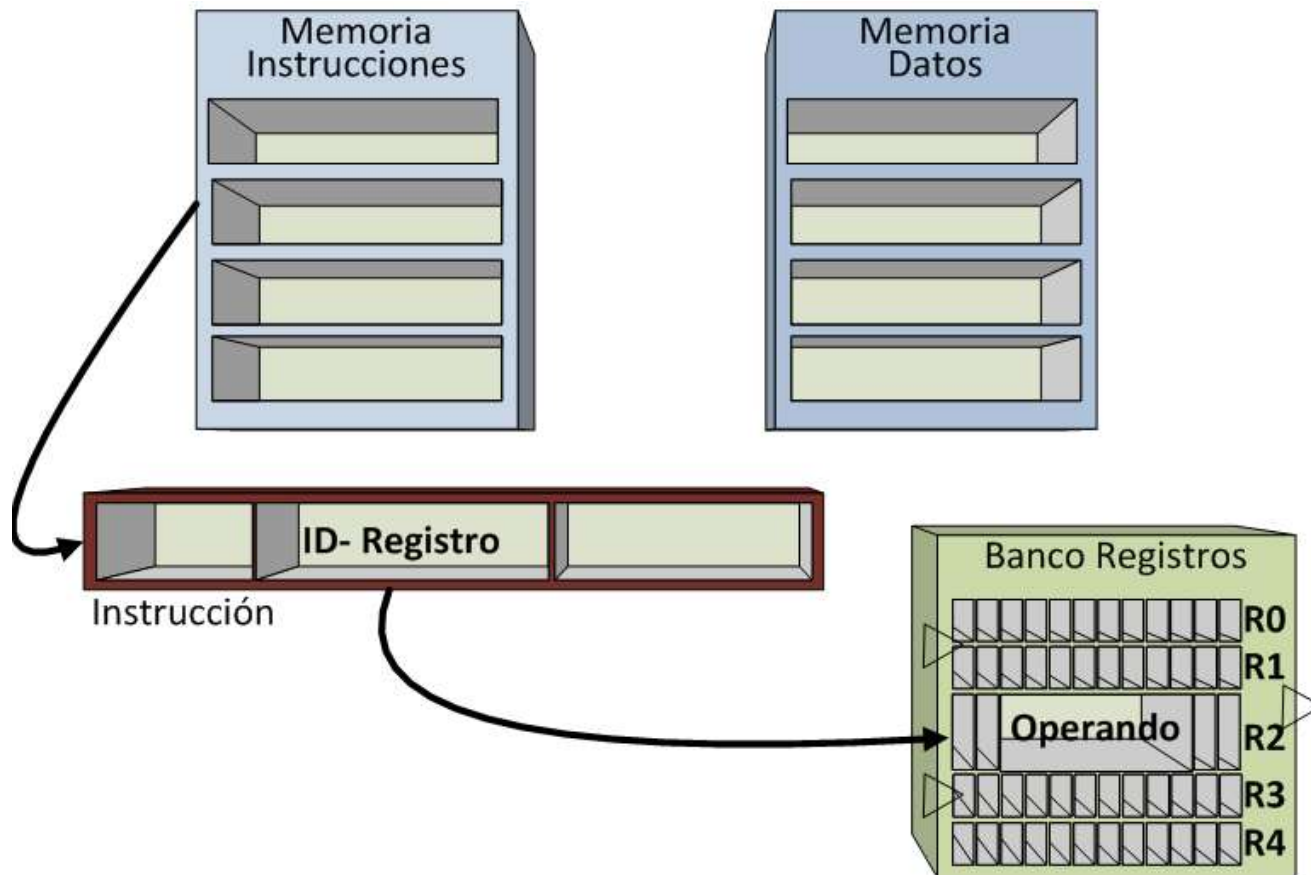
MDD Inmediato



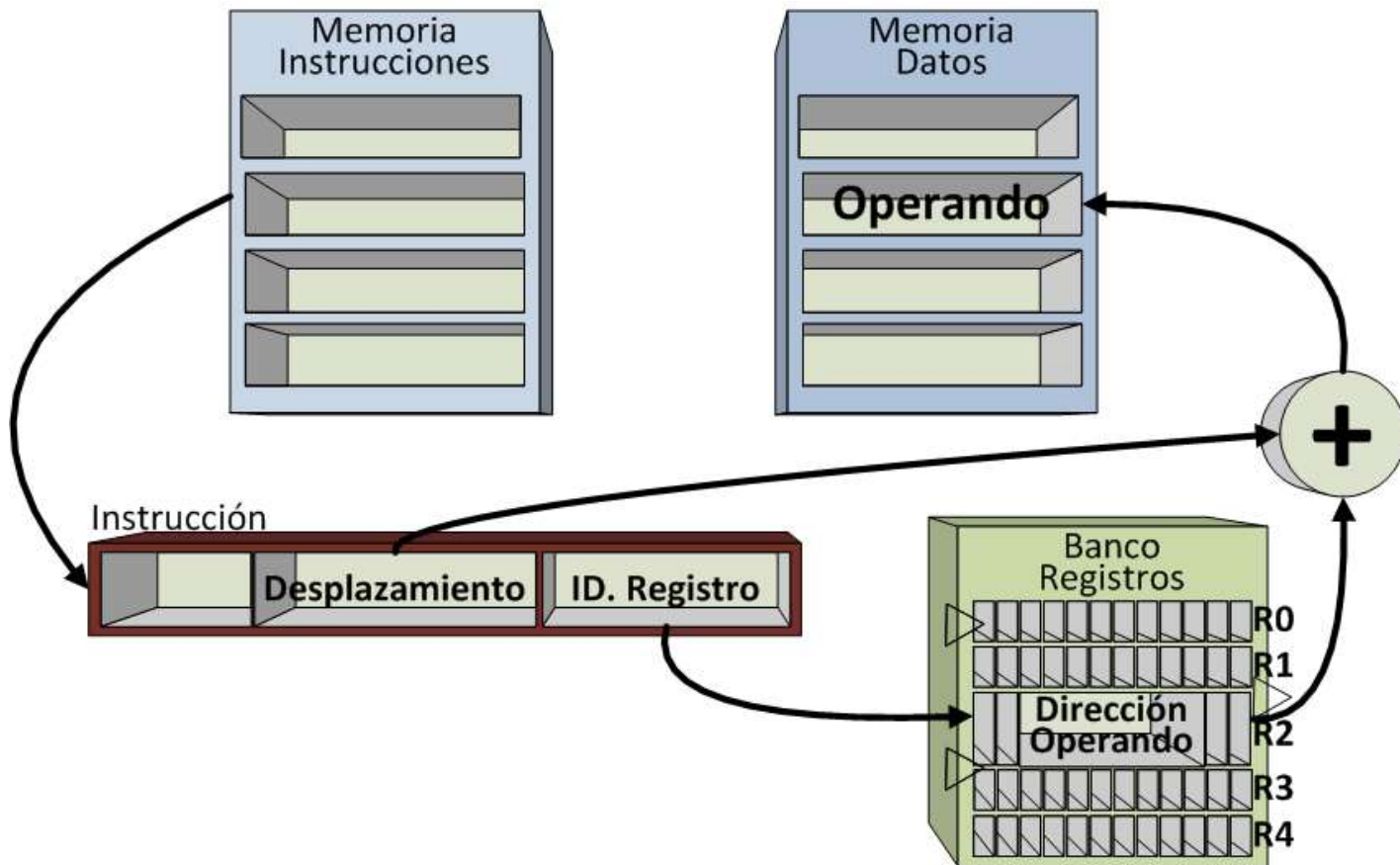
MDD Directo o Absoluto (de memoria)



MDD **Directo** de Registro

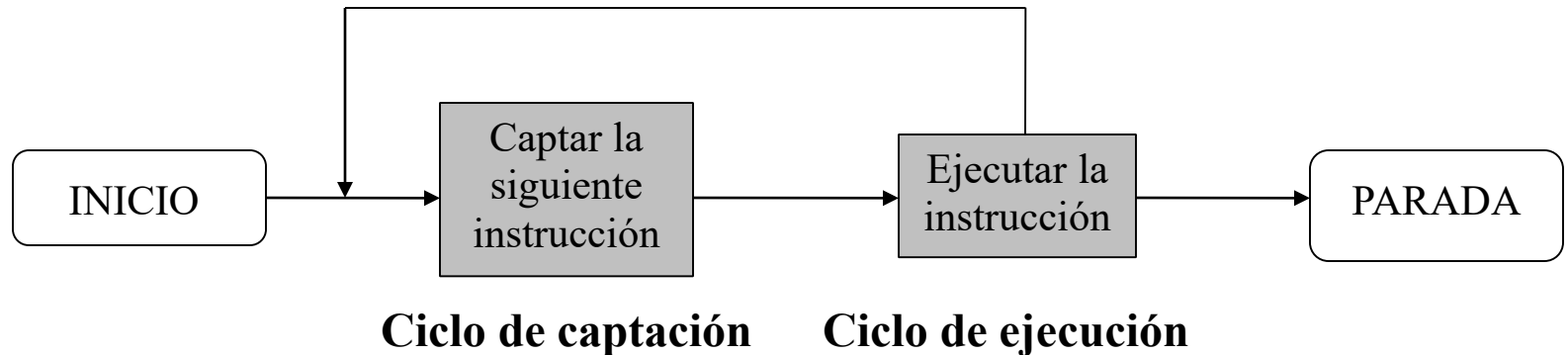


MDD Indirecto con desplazamiento



Ciclo de instrucción básico

- Dos pasos:
 - Captación
 - Ejecución



Ciclo de captación

- La dirección de la instrucción que se debe captar se encuentra en el registro Contador de Programa (PC)
- La UC **capta** la instrucción desde la Memoria
 - La instrucción va al registro de instrucción (IR)
- El registro PC se incrementa
 - a no ser que se indique lo contrario.
- La UC interpreta la instrucción captada y debe llevar a cabo la acción requerida

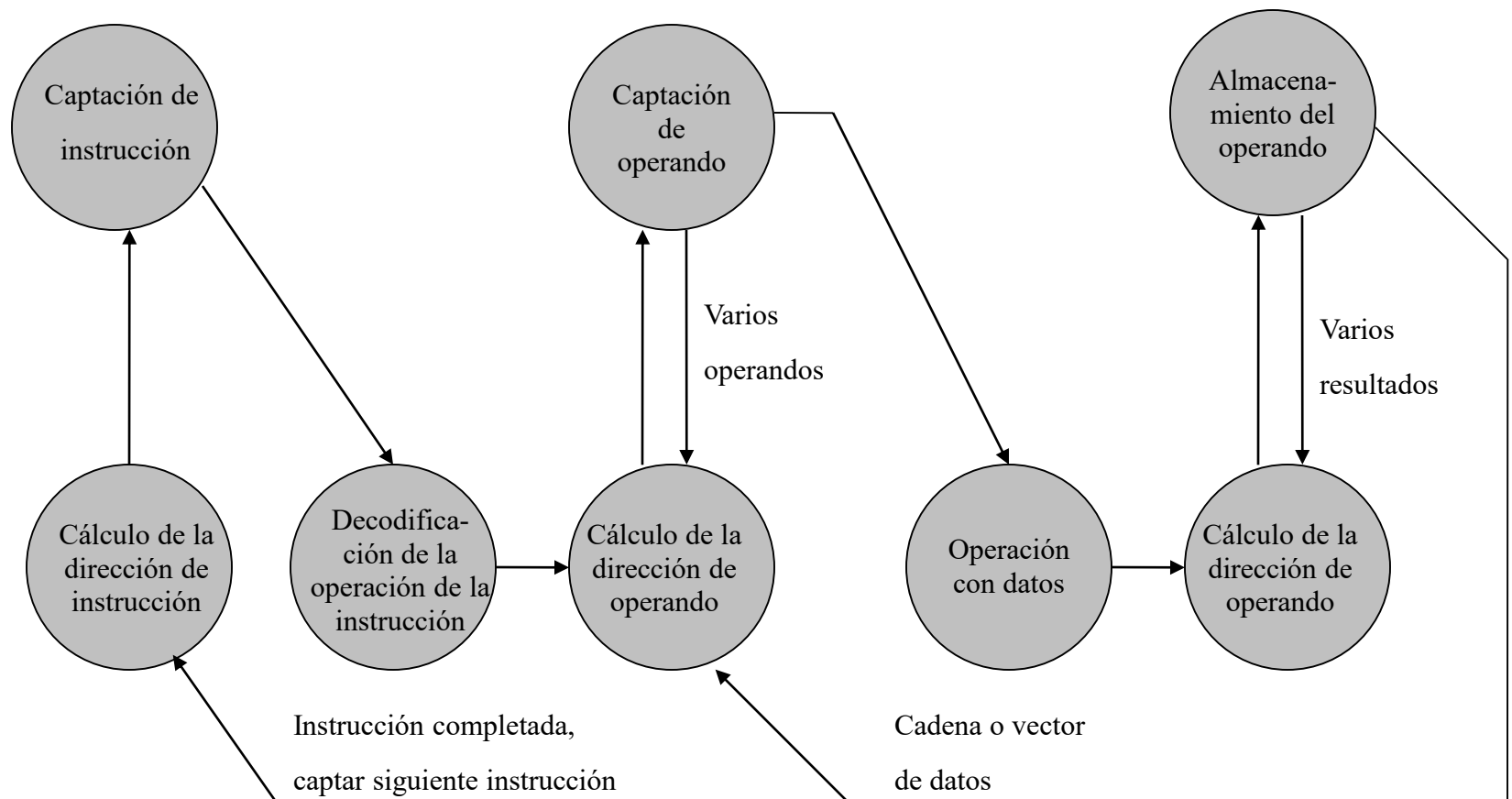
Ciclo de ejecución

Acciones posibles:

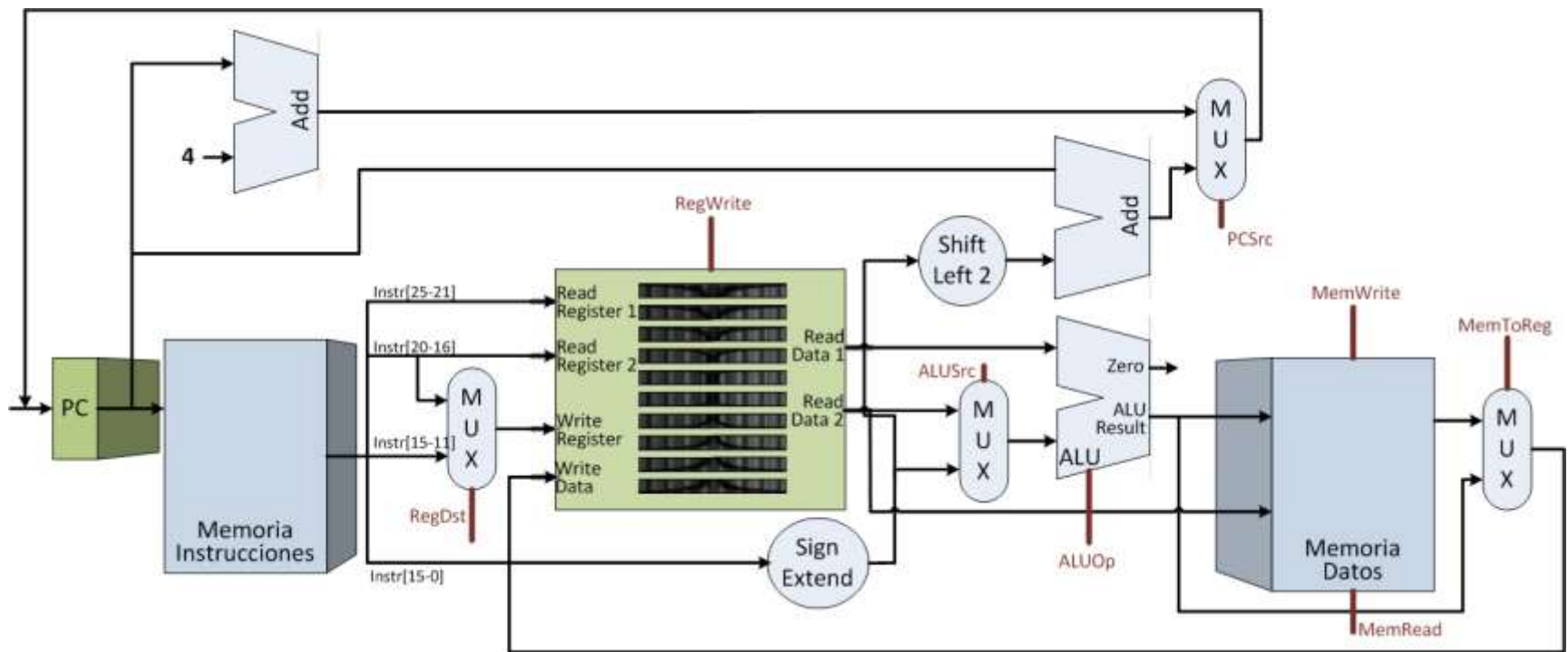
- Procesador - memoria
 - Transferencia de datos CPU - Memoria.
- Procesador - E/S
 - Transferencias de datos CPU y módulo de E/S.
- Procesamiento de datos
 - Alguna operación aritmética o lógica con los datos.
- Control
 - Alteración de la secuencia de ejecución.
 - Instrucción de salto

ó combinación de las acciones anteriores

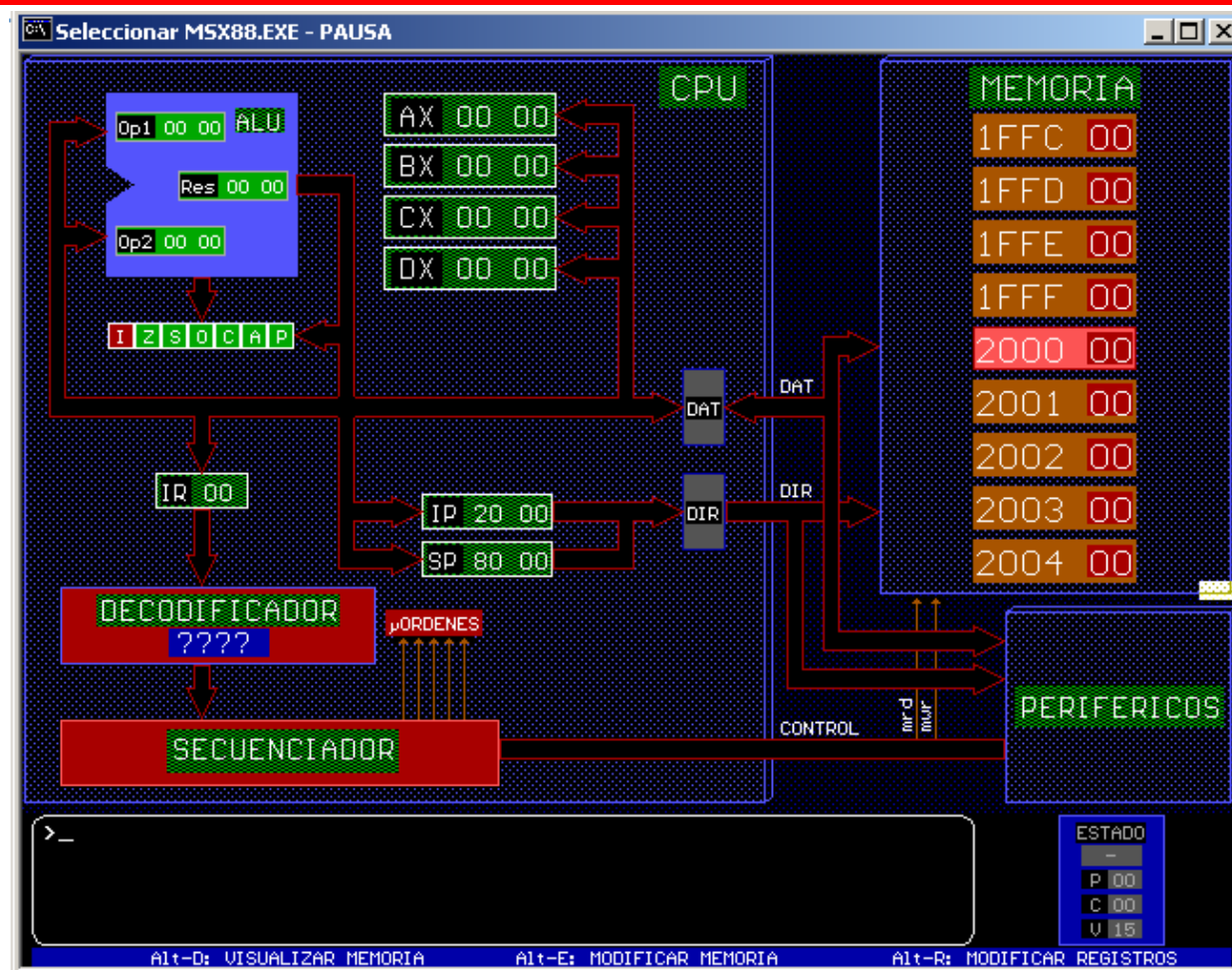
Diagrama de estados del ciclo de instrucción



Ruta de Datos



Simulador MSX88



Simulador VonSim [\(https://vonsim.github.io/\)](https://vonsim.github.io/)

01
10

VonSim

▶ Iniciar

⏹ Detener

⚙

```
1 ; ¡Bienvenido a VonSim!
2 ; Este es un ejemplo de código que calcula los primeros
3 ; n números de la sucesión de Fibonacci, y se guardan
4 ; partir de la posición 1000h de la memoria.
5
6     n equ 10    ; Calcula los primeros 10 números
7
8     org 1000h
9 inicio db 1
10
11     org 2000h
12     mov bx, offset inicio + 1
13     mov al, 0
14     mov ah, inicio
15
16 bucle: cmp bx, offset inicio + n
17        jns fin
18        mov cl, ah
19        add cl, al
20        mov al, ah
21        mov ah, cl
22        mov (bx), cl
23        inc bx
24        jmp bucle
25 fin:   hlt
26        end
```

No hay ningún archivo abierto

Listo para compilar

CPU

left 00 00

right 00 00

ADD

result 00 00

CF ZF SF IF OF

IR 00

Unidad de control

Decodificador

Detenido

???

AX 00 00

BX 00 00

CX 00 00

DX 00 00

id 00 00

IP 00 00

SP 00 00

ri 00 00

MAR 00 00

MBR 00

rd

wr

Memoria

Fijar dirección

< 2000h >

2000h	00	2008h	00
2001h	00	2009h	00
2002h	00	200Ah	00
2003h	00	200Bh	00
2004h	00	200Ch	00
2005h	00	200Dh	00
2006h	00	200Eh	00
2007h	00	200Fh	00

Documentación · GitHub · Reportar un error · © Copyright 2017-2025 — III-LIDI, FI, UNLP

VonSim: inst. de transferencia

1	MOV <i>dest,fuente</i>	Copia <i>fuentes</i> en <i>dest</i>	$(dest) \leftarrow (fuente)$
2	PUSH <i>fuentes</i>	Carga <i>fuentes</i> en el tope de la pila	$(SP) \leftarrow (SP) - 2; [SP+1:SP] \leftarrow (fuente)$
2	POP <i>dest</i>	Desapila el tope de la pila y lo carga en <i>dest</i>	$(fuente) \leftarrow [SP+1:SP]; (SP) \leftarrow (SP) + 2$
2	PUSHF	Apila los flags	$(SP) \leftarrow (SP) - 2; [SP+1:SP] \leftarrow (flags)$
2	POPF	Desapila los flags	$(flags) \leftarrow [SP+1:SP]; (SP) \leftarrow (SP) + 2$
3	IN <i>dest,fuentes</i>	Carga el valor en el puerto <i>fuentes</i> en <i>dest</i>	$(dest) \leftarrow (fuente)$
4	OUT <i>dest,fuentes</i>	Carga en el puerto <i>dest</i> el valor en <i>fuentes</i>	$(dest) \leftarrow (fuente)$

1. *dest/fuentes* son: *reg/reg*, *reg/mem*, *reg/op.inm*, *mem/reg*, *mem/op.inm*.
mem puede ser una etiqueta (dir.directo) o [BX] (dir.indirecto).
2. *dest* y *fuentes* solo pueden ser registros de 16 bits.
3. *dest/fuentes* son: *AL/mem*, *AX/mem*, *AL/DX*, *AX/DX*.
4. *dest/fuentes* son: *mem/AL*, *mem/AX*, *DX/AL*, *DX/AX*.
mem debe ser dirección entre 0 y 255. Puede ser un operando inmediato o una etiqueta.

Inst. aritméticas y lógicas

1	ADD <i>dest,fuente</i>	Suma <i>fuentes</i> y <i>dest</i>	$(dest) \leftarrow (dest) + (fuente)$
1	ADC <i>dest,fuente</i>	Suma <i>fuentes</i> , <i>dest</i> y flag <i>C</i>	$(dest) \leftarrow (dest) + (fuente) + C$
1	SUB <i>dest,fuente</i>	Resta <i>fuentes</i> a <i>dest</i>	$(dest) \leftarrow (dest) - (fuente)$
1	SBB <i>dest,fuente</i>	Resta <i>fuentes</i> y flag <i>C</i> a <i>dest</i>	$(dest) \leftarrow (dest) - (fuente) - C$
1	CMP <i>dest,fuente</i>	Compara <i>fuentes</i> con <i>dest</i>	$(dest) - (fuente)$
5	NEG <i>dest</i>	Negativo de <i>dest</i>	$(dest) \leftarrow CA2(dest)$
5	INC <i>dest</i>	Incrementa <i>dest</i>	$(dest) \leftarrow (dest) + 1$
5	DEC <i>dest</i>	Decrementa <i>dest</i>	$(dest) \leftarrow (dest) - 1$
1	AND <i>dest,fuente</i>	Operación <i>fuentes</i> AND <i>dest</i> bit a bit	$(dest) \leftarrow (dest) \text{ AND } (fuente)$
1	OR <i>dest,fuente</i>	Operación <i>fuentes</i> OR <i>dest</i> bit a bit	$(dest) \leftarrow (dest) \text{ OR } (fuente)$
1	XOR <i>dest,fuente</i>	Operación <i>fuentes</i> XOR <i>dest</i> bit a bit	$(dest) \leftarrow (dest) \text{ XOR } (fuente)$
5	NOT <i>dest</i>	Complemento a 1 de <i>dest</i>	$(dest) \leftarrow CA1(dest)$

1. *dest/fuente* son: *reg/reg*, *reg/mem*, *reg/op.inm*, *mem/reg*, *mem/op.inm*.

5. *dest* solo puede ser *mem* o *reg*.

mem puede ser una etiqueta (*dir.directo*) o *[BX]*, siendo (*BX*) una dirección de memoria (*dir.indirecto*).

Inst. transf. de control

6	CALL <i>etiqueta</i>	Llama a subrutina cuyo inicio es <i>etiqueta</i>	
6	RET	Retorna de la subrutina	
6	JZ <i>etiqueta</i>	Salta si el último valor calculado es cero	Si $Z=1$, $(IP) \leftarrow mem$
6	JNZ <i>etiqueta</i>	Salta si el último valor calculado no es cero	Si $Z=0$, $(IP) \leftarrow mem$
6	JS <i>etiqueta</i>	Salta si el último valor calculado es negativo	Si $S=1$, $(IP) \leftarrow mem$
6	JNS <i>etiqueta</i>	Salta si el último valor calculado no es negativo	Si $S=0$, $(IP) \leftarrow mem$
6	JC <i>etiqueta</i>	Salta si el último valor calculado produjo carry	Si $C=1$, $(IP) \leftarrow mem$
6	JNC <i>etiqueta</i>	Salta si el último valor calculado no produjo carry	Si $C=0$, $(IP) \leftarrow mem$
6	JO <i>etiqueta</i>	Salta si el último valor calculado produjo overflow	Si $O=1$, $(IP) \leftarrow mem$
6	JNO <i>etiqueta</i>	Salta si el último valor calculado no produjo overflow	Si $O=0$, $(IP) \leftarrow mem$
6	JMP <i>etiqueta</i>	Salto incondicional a <i>etiqueta</i>	$(IP) \leftarrow mem$

6. *mem* es la dirección de memoria llamada *etiqueta*.

Subrutinas

- Innovación en lenguajes de programación
- Programa auto-contenido
- Puede invocarse desde cualquier punto de un programa
 - mediante instrucción CALL
- Brinda economía (código usado varias veces) y modularidad (subdivisión en unidades pequeñas).
- Requiere pasaje de argumentos (parámetros)
 - por valor (copia de una variable)
 - por referencia (dirección de la variable)

Pasaje de argumentos a subrutinas

- Vía registros
 - El número de registros es la principal limitación
 - Es importante documentar que registros se usan
- Vía memoria
 - Se usa un área definida de memoria (RAM).
 - Difícil de estandarizar

Pasaje de argumentos a subrutinas

- Vía pila (stack)
 - Es el método más ampliamente usado.
 - El verdadero “pasaje de parámetros”.
 - Independiente de memoria y registros.
 - Hay que comprender bien como funciona porque la pila (stack) es usada por el usuario y por el sistema.

En x86, SP apunta al último lugar usado

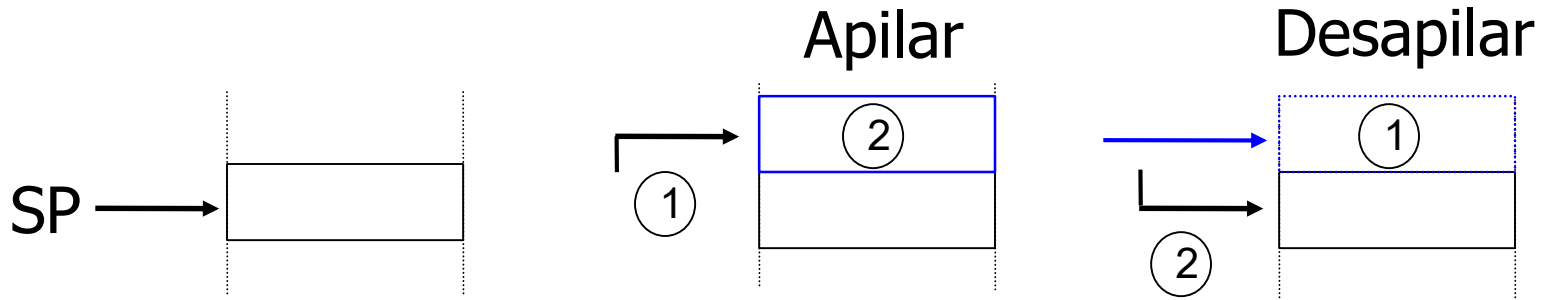
Funcionamiento de una pila

- El operando está (de forma implícita) en la cabeza de la pila
- Se requiere un registro Puntero de Pila (SP)
 - Contiene la dirección de la cabeza de la pila
- Operaciones sobre la pila
 - **PUSH** ; operación de Apilar
 - **POP** ; operación de Desapilar
 - Son inversas entre sí

Operaciones de apilar/desapilar

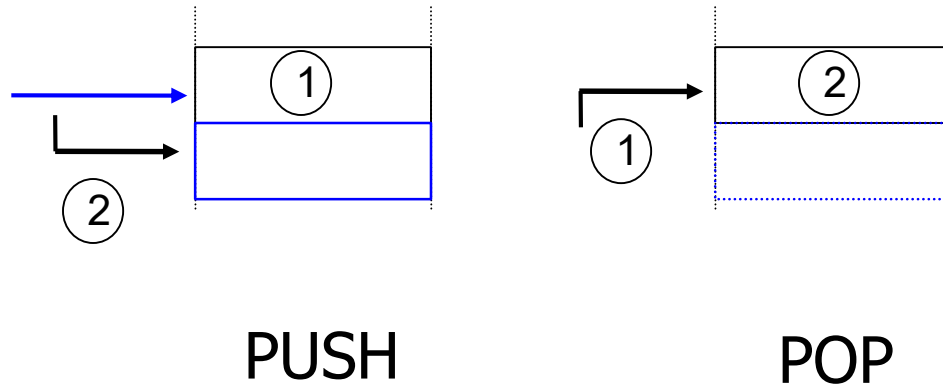
- Secuencia de dos acciones:
 - 1- Movimiento de datos Reg-Mem ó Mem- Reg
 - 2- Modificación del puntero antes/después de la anterior
- Tener en cuenta:
 - dónde apunta el puntero
 - cómo crece la pila

Funcionamiento de la pila



x86

- Mover dato
- Modificar SP



Ejemplo en Assembly

Ejemplo:

```
ORG 2000H  
MOV BX, 3000H  
MOV AX, [BX]  
PUSH AX  
MOV BX, 3002H  
MOV CX, [BX]  
PUSH CX  
POP AX  
POP CX  
HLT  
ORG 3000H  
datos DB 55h, 33h, 44h, 22h  
END
```


Definición del procedimiento

Nombre Proc

...

...

...

Ret

Nombre Endp

} Cuerpo del procedimiento

Llamada al procedimiento

En programa principal

...

Push Parametro 1

Push Parametro 2

Call Nombre

...

...

Ejemplo con subrutina

```
                ORG 1000H  
subrutina:  NEG    AX  
            RET
```

```
                ORG 2000H  
MOV     BX, 0  
MOV     AX, dato  
PUSH    AX  
CALL    subrutina  
POP     BX  
HLT
```

Analizar la pila y los valores
finales de AX y BX

```
                ORG 3000H  
dato:  DB     55H  
                END
```

Posibles pasos en un procedimiento

1. Salvar el estado de BP (viejo BP)
2. Salvar estado de SP ($BP=SP$)
3. Reservar espacio para datos locales (opcional)
4. Salvar valores de otros registros (opcional)
5. Acceder a parámetros
6. Escribir sentencias a ejecutar
7. Retornar parámetro (opcional)
8. Regresar correctamente del procedimiento

Pasos... (1)

- El procedimiento comenzaría con:
 push BP
 mov BP, SP
- Esto establece a BP como puntero de referencia y es usado para acceder a los parámetros y datos locales en la pila. SP no puede ser usado para éste propósito porque no es un registro base ó índice. El valor de SP puede cambiar pero BP permanece 'quieto'.

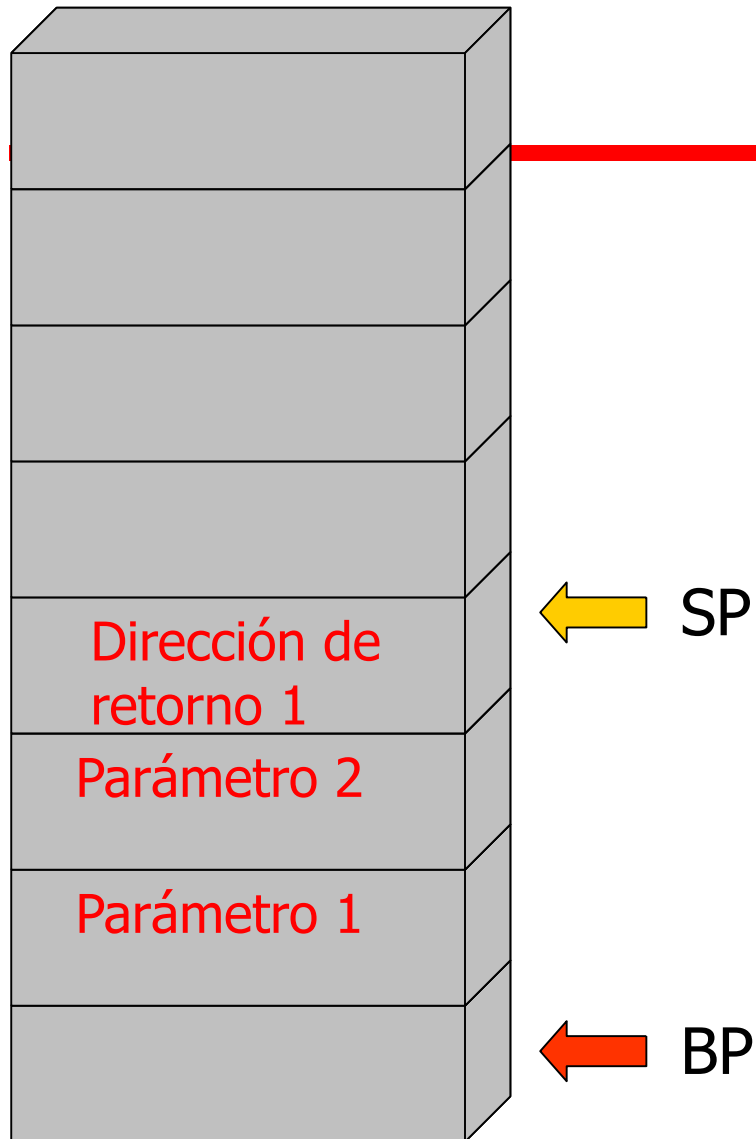
Pasos... (2)

- Así la primera instrucción salva BP y la segunda carga el valor de SP en BP (en el momento de entrar al procedimiento).
- BP es el puntero al área de la pila asignada al procedimiento (frame pointer).
- Para acceder a los datos se deberá sumar un desplazamiento fijo a BP.

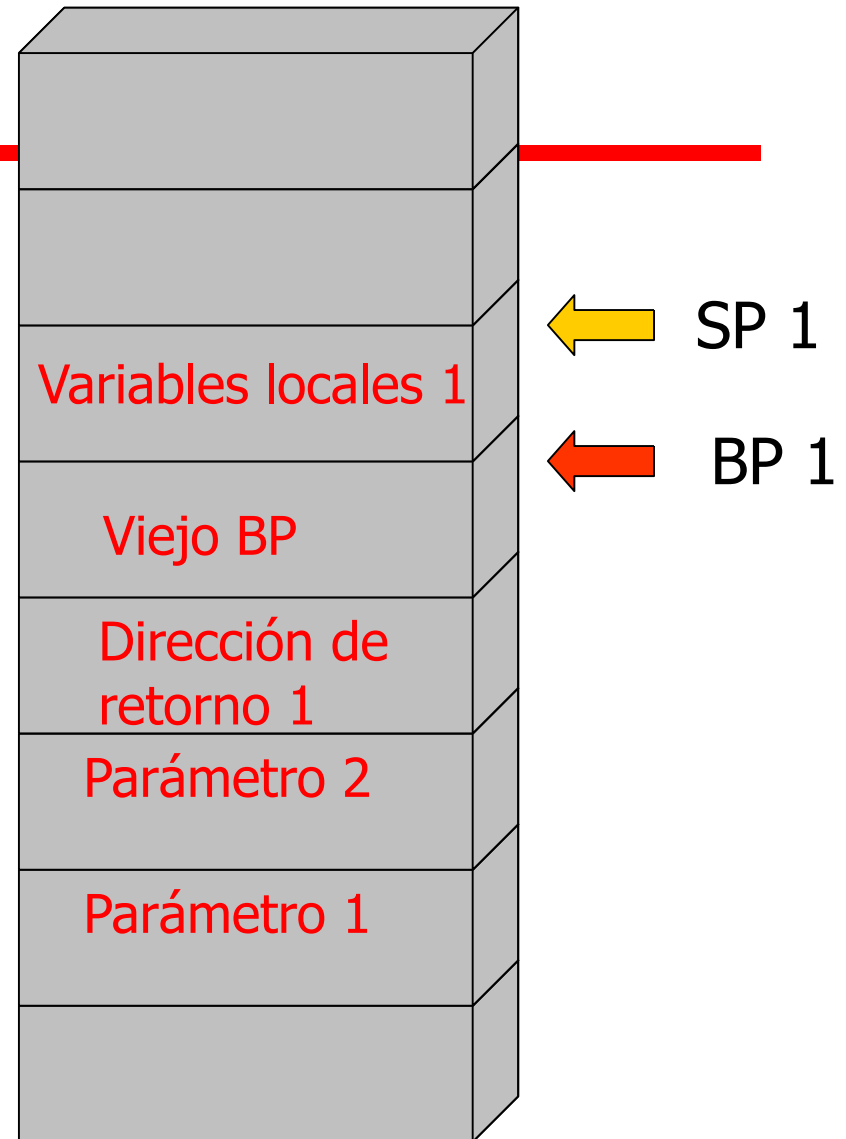
Pasos... (3) (Opcional)

- Reservar espacio para variables locales
 - se decrementa SP, reservando lugar en la pila
sub SP, 2
 - Este ej. reserva 2 bytes para datos locales.
- El sistema puede utilizar al SP sin escribir sobre el área de trabajo (o frame) del procedimiento.

SP y BP al entrar a SUBR 1



SP y BP después de paso 3

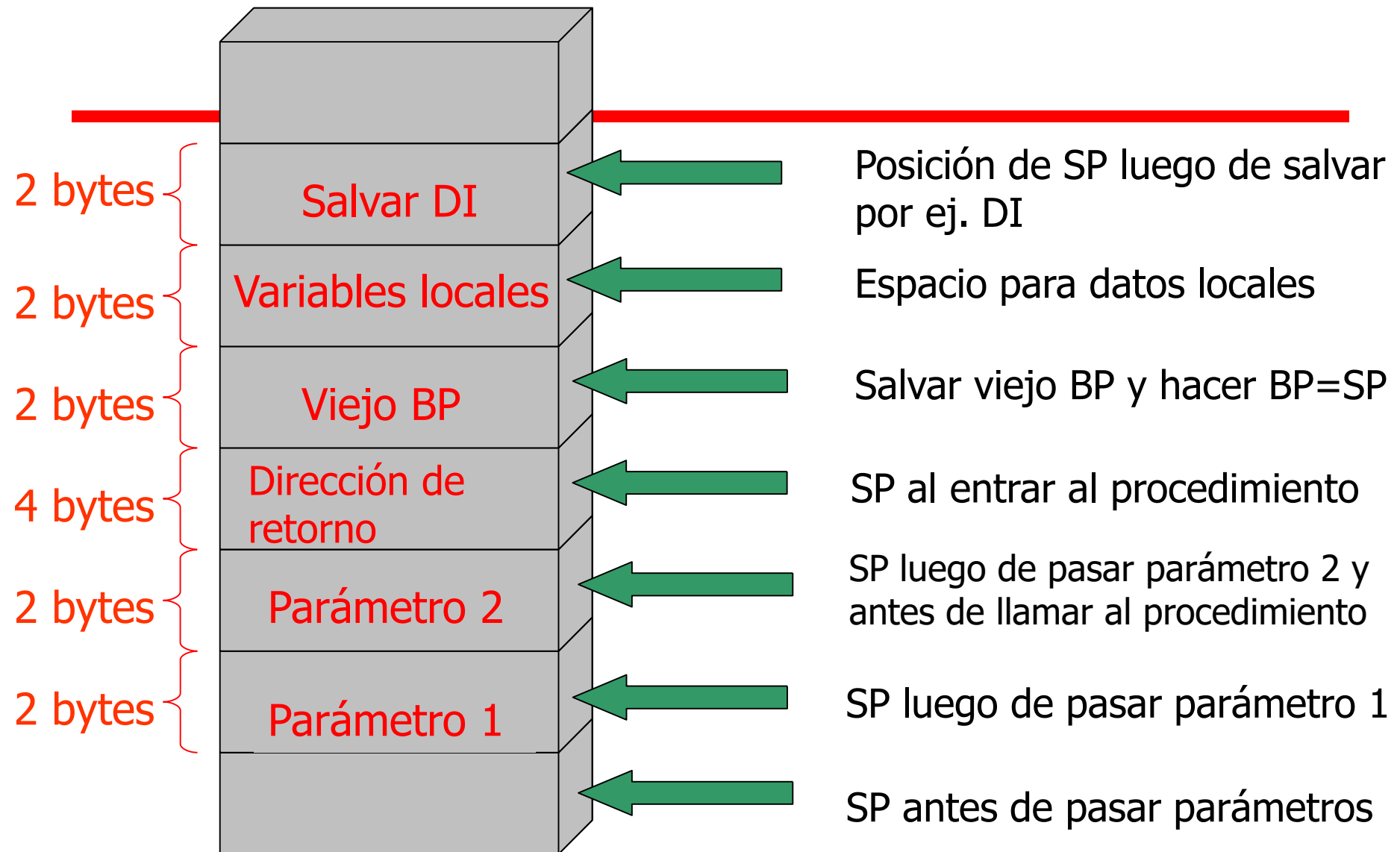


Pasos... (4) (Opcional)

- Salvar otros registros
 - por ej. DI

push DI

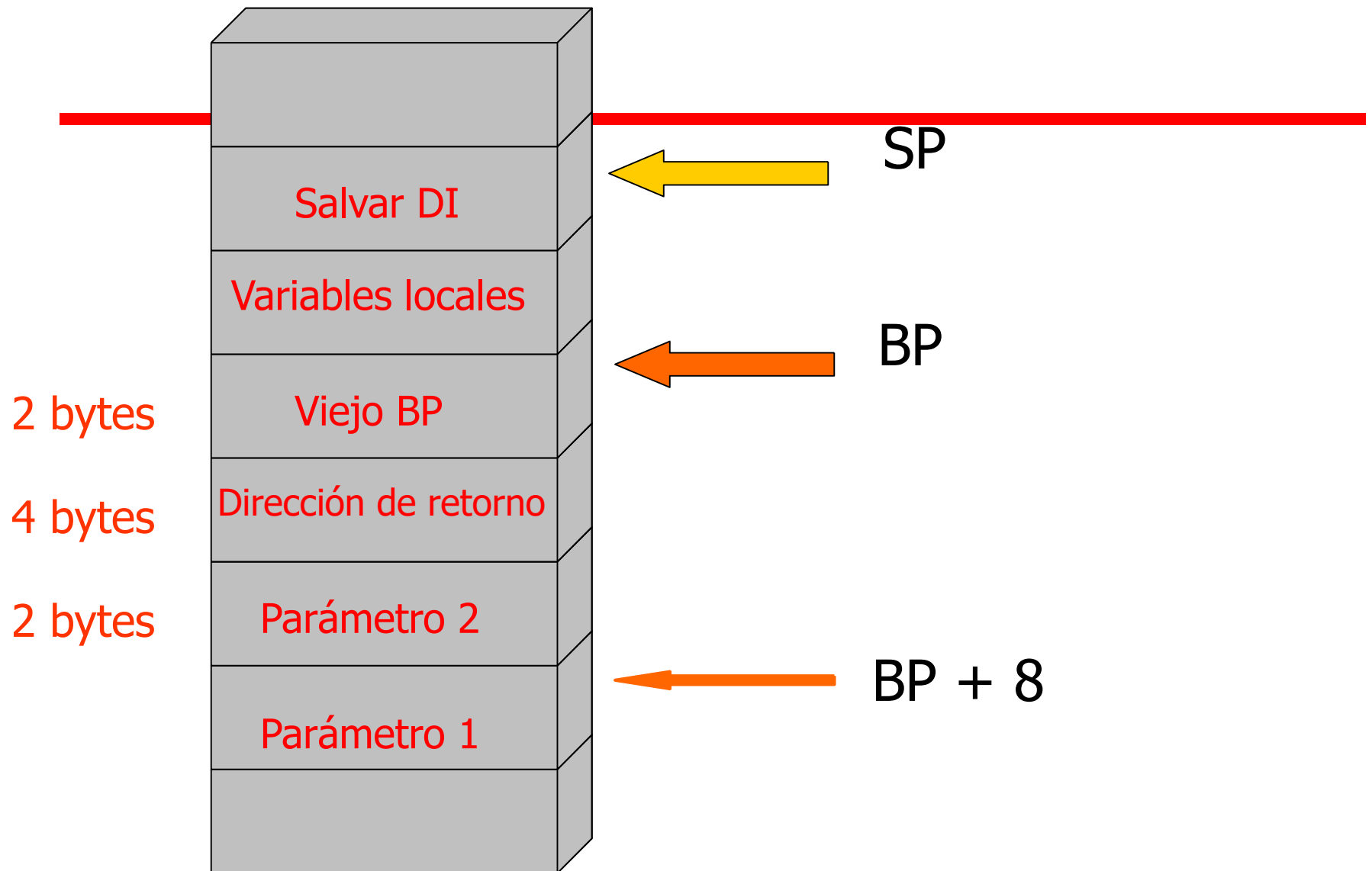
- Si el procedimiento no cambia el valor de los registros, éstos no necesitan ser salvados.
Normalmente los registros son salvados después de establecer el puntero (frame pointer) y los datos locales.



Pasos... (5)

acceso a los parámetros

- En general el desplazamiento de BP para acceder a un parámetro es igual a:
- 2 (es el tamaño de BP apilado) + tamaño de dirección de retorno + total de tamaño de parámetros entre el buscado y BP
- Para acceder al Parámetro 1 deberá ser:
`mov CX, [BP + 8]`



Salida del procedimiento (1)

- Los registros salvados en la pila deben ser descargados en orden inverso.
- Si se reservó espacio para variables locales, se debe reponer SP con el valor de BP que no cambió durante el procedimiento.
- Reponer BP.
- Volver al programa que llamó al procedimiento con RET.

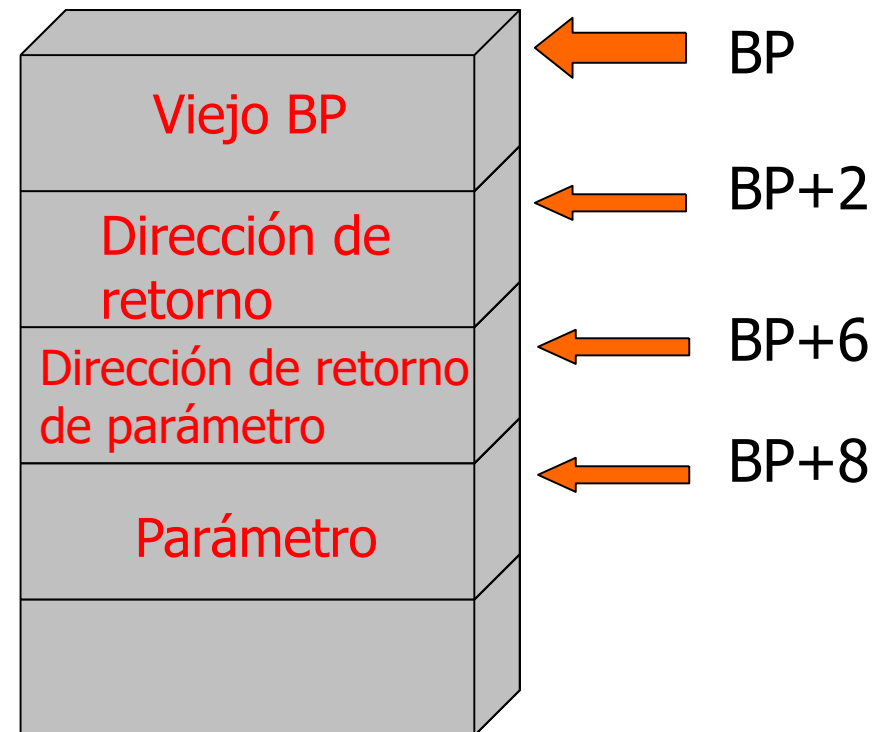
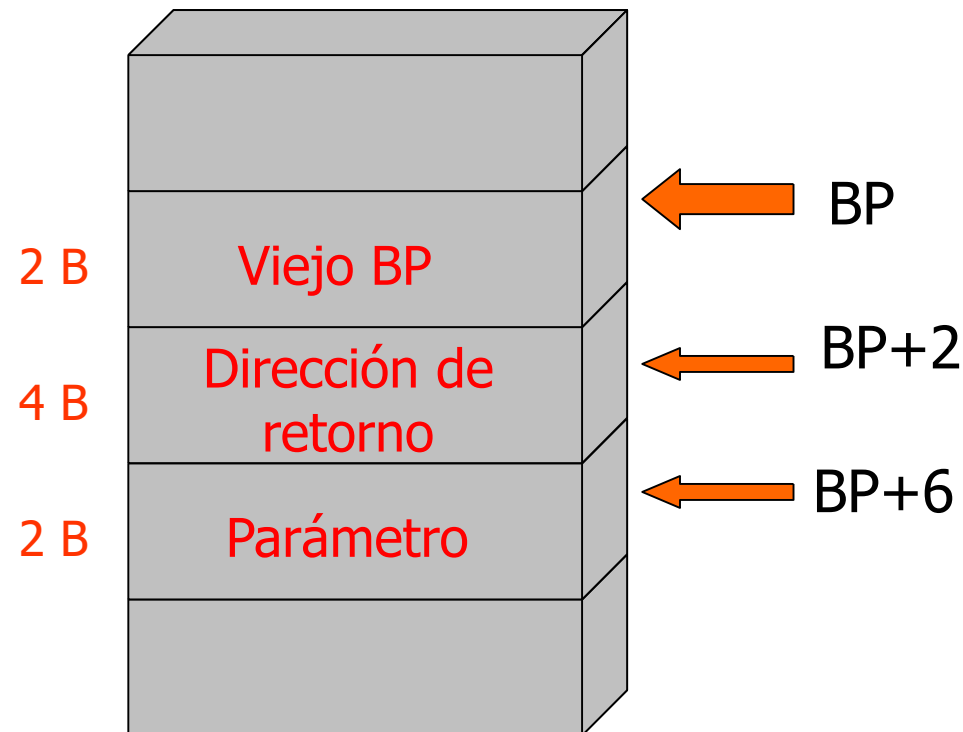
Salida del procedimiento (2)

- En nuestro ej.

```
    .  
    .  
pop  DI  
mov  SP, BP  
pop  BP  
ret
```

Sin parámetro de retorno

Con parámetro de retorno



Anidamiento de subrutinas

ORG 1000H

```
rutina1: NEG    AX
          PUSH   AX
          CALL   rutina2
          POP    AX
          RET
```

ORG 1020H

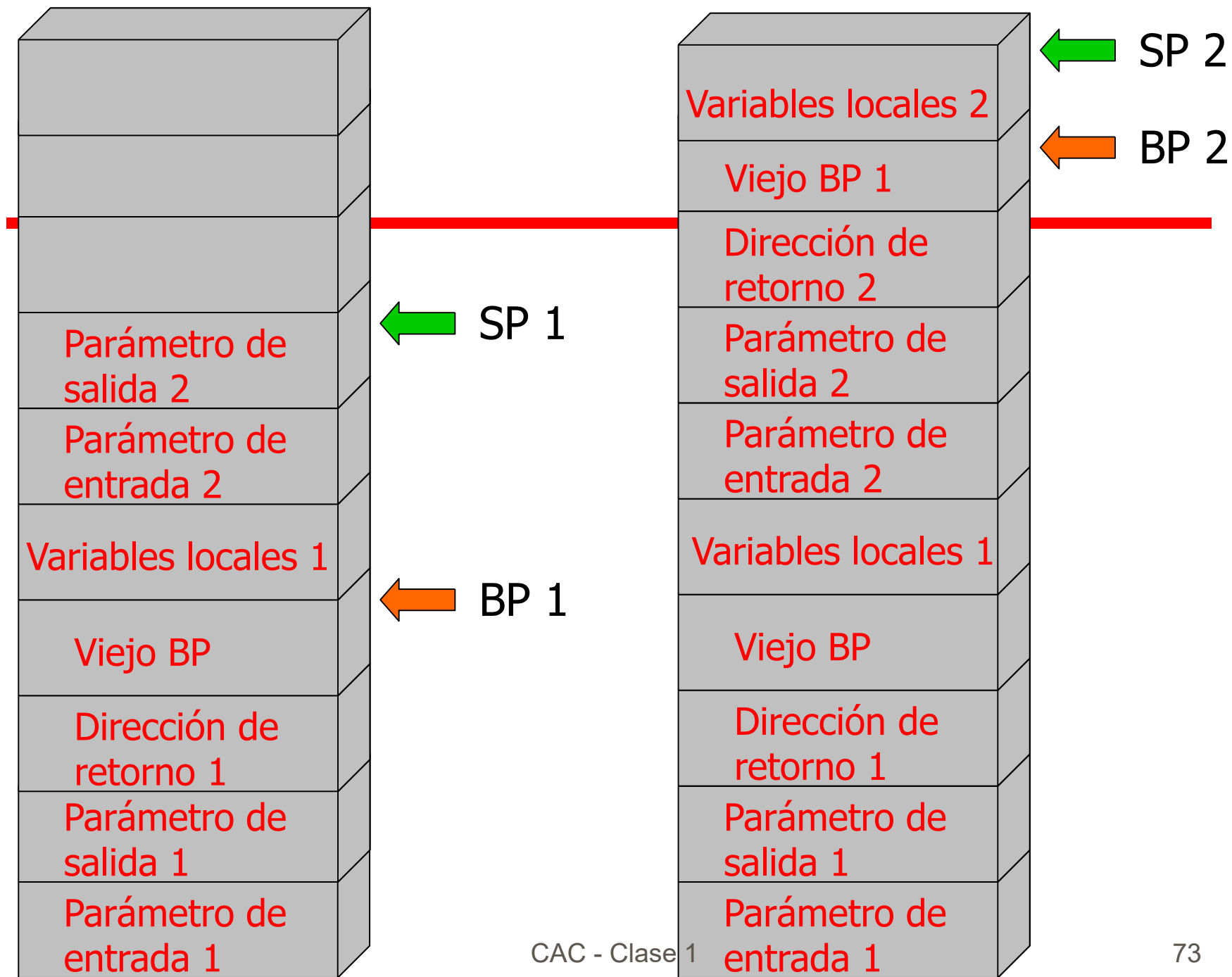
```
rutina2: INC    AX
          RET
```

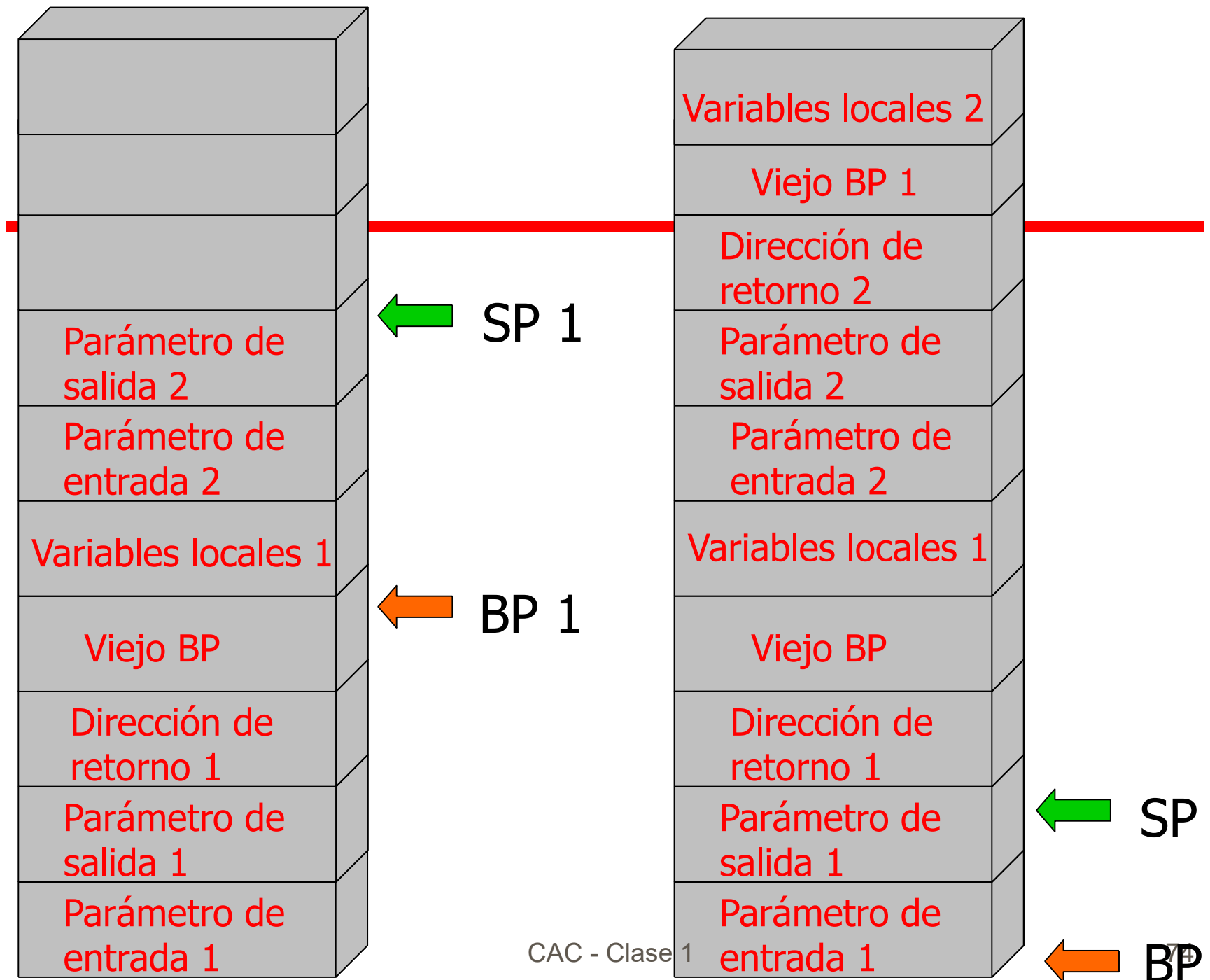
ORG 2000H

```
PPIO:  MOV    BX, 0
        MOV    AX, dato
        PUSH   AX
        CALL   rutina1
        POP    BX
        HLT
```

ORG 3000H

```
dato:  DB      55H
        END
```



Para el simulador

- Declaración del procedimiento
nombre: instrucción
.
.
- En lugar de BP se usa BX

Ejemplo para simulador

ORG 1000H

NUM1 **DW** 5H

NUM2 **DW** 3H

RES **DW** ?

ORG 3000H

MUL: PUSH BX
MOV BX,SP
PUSH CX
PUSH AX
PUSH DX
ADD BX,6
MOV CX,[BX]
ADD BX,2

.....

MOV AX,[BX]
SUMA: ADD DX,AX
DEC CX
JNZ SUMA
SUB BX,4
MOV AX,[BX]
MOV BX,AX
MOV [BX],DX
POP DX
POP AX
POP CX
POP BX
RET

ORG 2000H

MOV AX,NUM1
PUSH AX
MOV AX,NUM2
PUSH AX
MOV AX,OFFSET RES
PUSH AX
MOV DX,0
CALL MUL
POP AX
POP AX
POP AX
HLT
END

Bibliografía e información

- Organización y Arquitectura de Computadoras. W. Stallings, 5ta Ed.
 - Repertorios de instrucciones
 - Capítulo 9: características y funciones
 - Capítulo 10: modos de direccionamiento y formatos
 - Apéndice 9A: Pilas
 - Ciclo de instrucción:
 - Capítulo 3 apartado 3.2.
 - Capítulo 11 apartados 11.1. y 11.3.
 - Organización de los registros
 - Capítulo 11 apartado 11.2.
 - Formatos de instrucciones
 - Capítulo 10 apartado 10.3. y 10.4.
- Simulador VonSim (<https://vonsim.github.io/>)
 - Documentación: (<https://vonsim.github.io/docs/>)

Link de interés: www.williamstallings.com