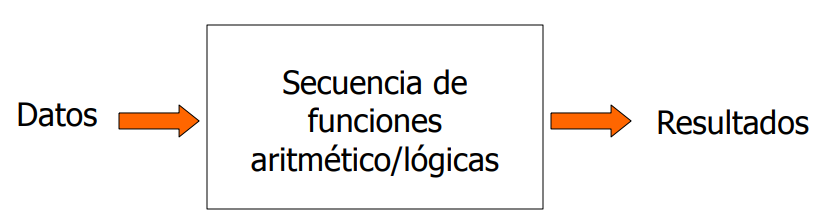
[Resumen clase 1](https://drive.google.com/file/d/1C3BgcYYlPOQfk6Yu5RFVLOUz8T3Ky0wc/view?usp=sharing)

# Repaso Organización de Computadoras

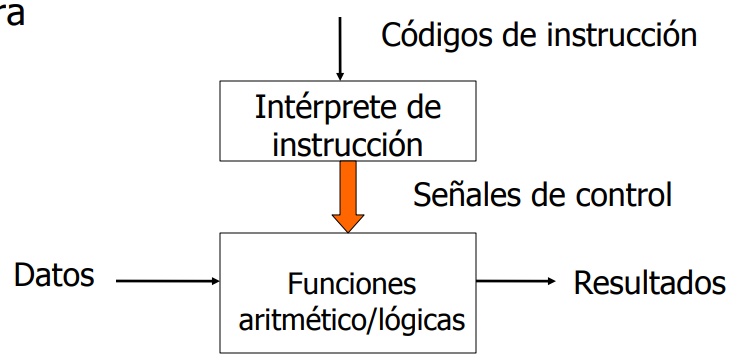
## Programas

**Antes** se tenían sistemas cableados:



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

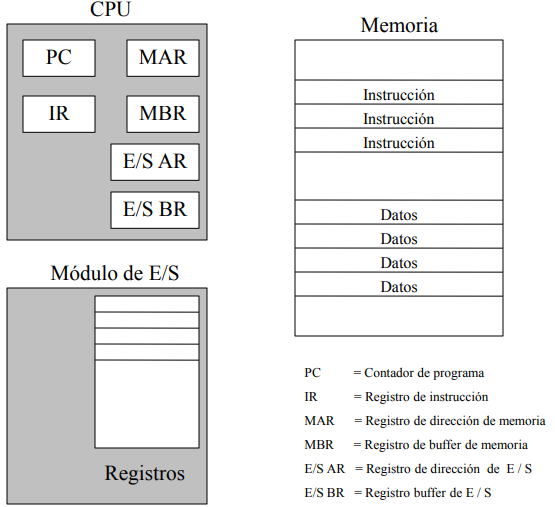
**Ahora**



* Programación en software: En cada paso se efectúa alguna operación sobre los datos
  + 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 debemos cambiar el hardware.**

## Arquitectura Von Neumann

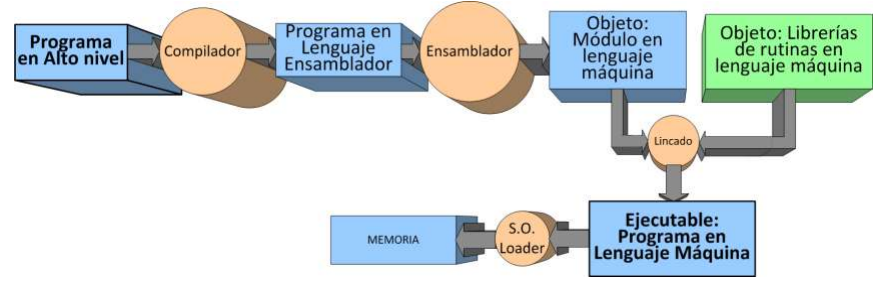
* CPU constituída por:
  + UC
  + ALU
* Datos e instrucciones deben introducirse en el sistema y los resultados se proporcionan mediante componentes de E/S
* Se necesita almacenar temporalmente datos e instrucciones: Memoria principal
* Componentes de una computadora:



## 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, SUB, LOAD
  + De operandos: ADD BX, PEPE (suma los contenidos de BX y PEPE y almacena el resultado en BX)

### Alto nivel a máquina



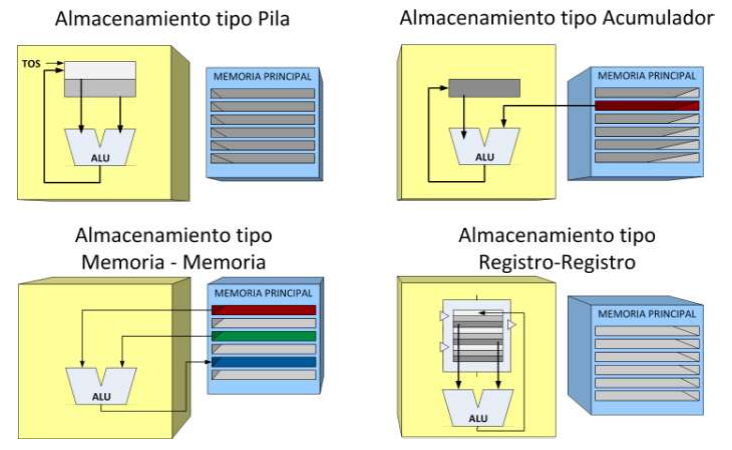
### Elementos de una instrucción

* CodOp
* Referencia a operandos fuentes
* Referencia a operando resultado
* Referencia a la siguiente instrucción

### Dónde se almacenan los operandos

* Memoria principal (o virtual o en memoria caché)
* Registro de la CPU
* Dispositivos de E/S

Alternativas de almacenamiento:



### 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)

### Cantidad de direcciones

* Más direcciones por instrucción
  + Instrucciones más complejas
  + Más registros (las operaciones entre 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

### Diseño del conjunto de instrucciones

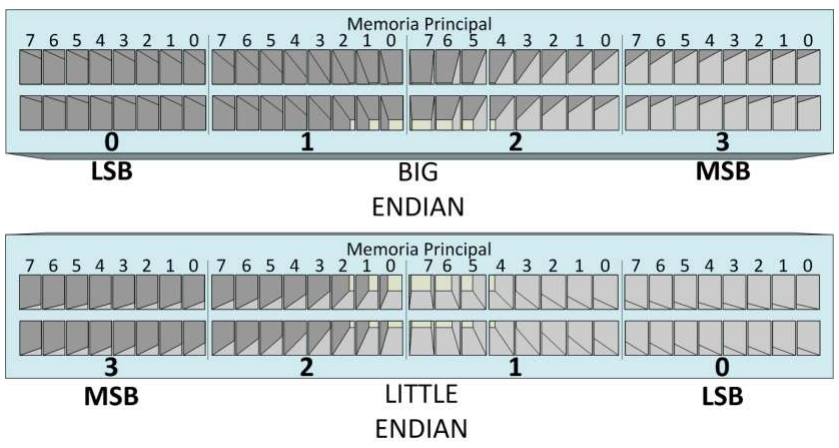
* Tipos de operandos (datos)
  + Direcciones
  + Números (PFijo, PFlotante)
  + Caracteres (ASCII, EBCDIC,...)
  + Datos lógicos (bits -1 o 0-) (flags o indicadores)
* Repertorio de operaciones (cuántas, cuáles y con qué complejidad c/u)
* Formato de instrucciones (longitud de instrucción, número de direcciones, tamaño de los campos)
* Registros (número de registros de la CPU referenciables, qué operaciones en c/u)
* Modos de direccionamiento (cómo se específica la ubicación de un operando o instrucción)

**RISC** (Computadora de Conjunto Reducido de Instrucciones)vs. **CISC** (Computadora de Conjunto Complejo de Instrucciones)

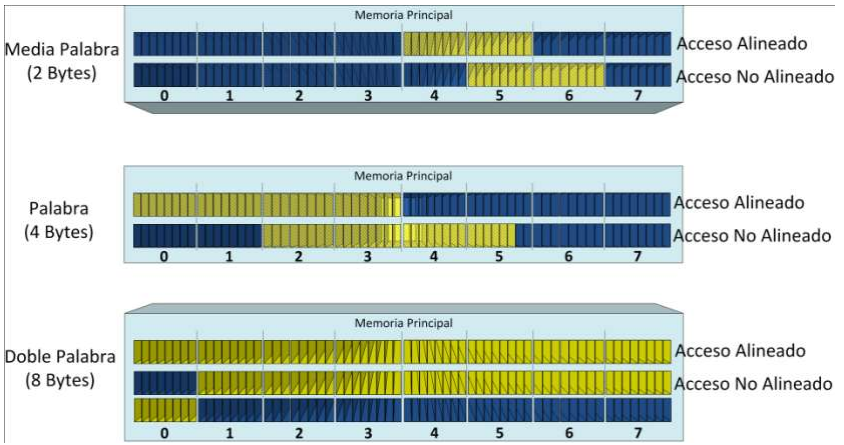
### Orden de los bytes

Suponiendo una memoria direccionable de 1 byte, los números que ocupan más de 1 byte se pueden almacenar (y leer) en bytes consecutivos de 2 formas, según el bit menos significativo (LSB) y el bit más significativo (MSB) (Ej: 98765432H)

* Big Endian: MSB en la dirección con valor numérico más bajo
  + Intel 80x86, Pentium y VAX
* Little Endian: LSB en la dirección con valor numérico más bajo
  + IBM S/370, Motorola 680x0 (Mac) y la mayoría de los RISC



| Dir. de byte | Big | Little |
| --- | --- | --- |
| 00 | 98 | 32 |
| 01 | 76 | 54 |
| 02 | 54 | 76 |
| 03 | 32 | 98 |

Accesos a la memoria

* Los accesos no alineados son más lentos

### Tipos de operaciones

* Transferencia de datos
* Aritméticas
* Lógicas
* Conversión
* E/S
* 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
  + MDD
* Diferentes movimientos → Diferentes instrucciones
  + Reg-reg, Reg-mem o Mem-reg
* O una instrucción y diferentes direcciones
  + MOV destino, fuente

#### Aritméticas

* Operaciones básicas (ADD, SUB, MUL, DIV)
  + Enteros con/sin signo
  + Punto flotante?
* Pueden incluirse otras (INC, DEC, NEG, ABS, Shift left/right -desplaza bits-)

#### Lógicas

Operaciones que manipulan bits individualmente

* Operaciones Booleanas (AND, OR, XOR, NOT)
* Otras (rotate left/right)

#### Conversion

Operaciones para cambiar formatos de datos

* Binario a decimal
* EBCDIC a ASCII

#### E/S

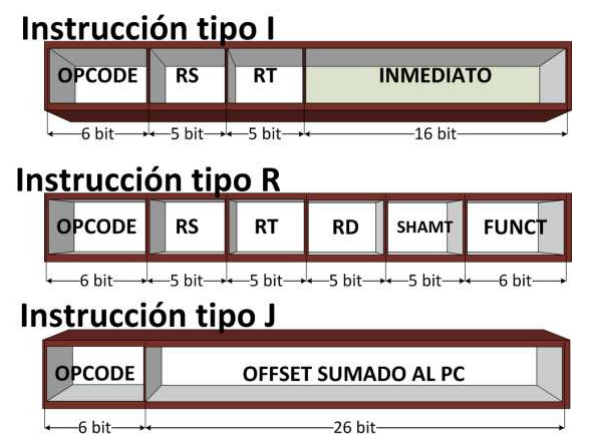
* Pocas instrucciones pero de acciones específicas: IN o OUT
* Se pueden realizar usando instrucciones de movimiento de datos: MOVE
* Se pueden realizar a través de un controlador aparte: DMA (Direct Memory Access)

#### Control de flujo

Modifican el valor contenido en el registro PC

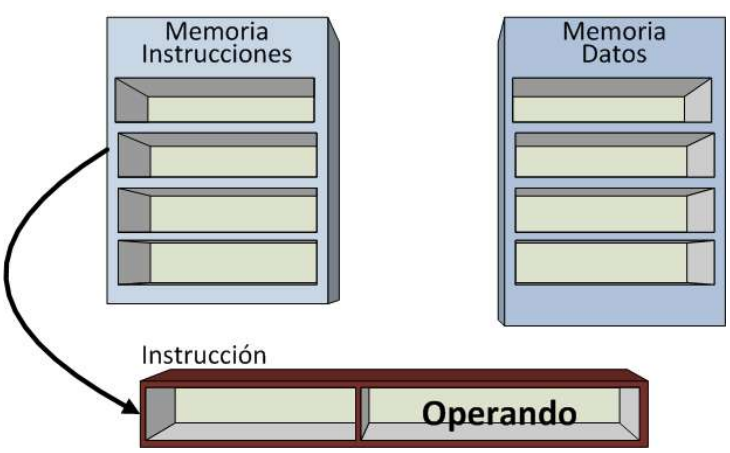
* Salto incondicional: JMP
* Salto condicional: JZ
* Salto con retorno o llamada a subrutina: CALL subrut
  + Para retornar al programa que llamó se usa RET

### Formato de instrucción

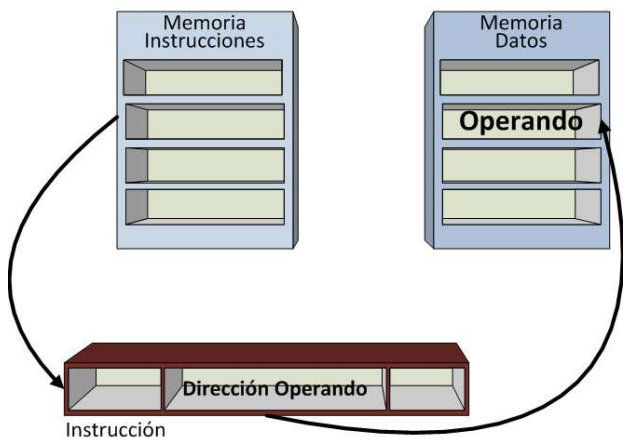


### Modos de direccionamiento

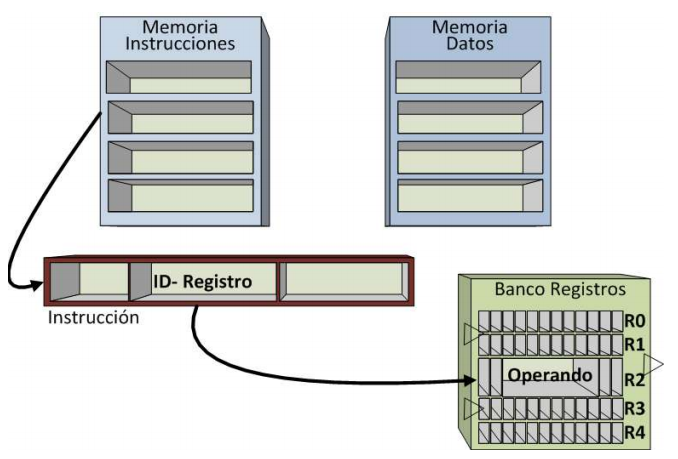
#### Inmediato



#### Directo o absoluto (de memoria)



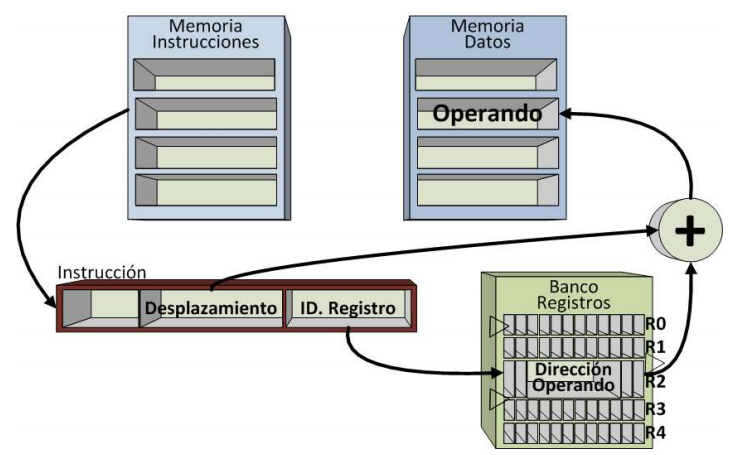
#### Directo de registro



#### Indirecto con desplazamiento

Puede ser:

* Basado, indexado o relativo al PC
* Pila (o relativo al SP)



#### Indirecto de memoria (en desuso)

#### Indirecto con registro

## Ciclo de instrucción

Dos pasos:

* Captación
* Ejecución

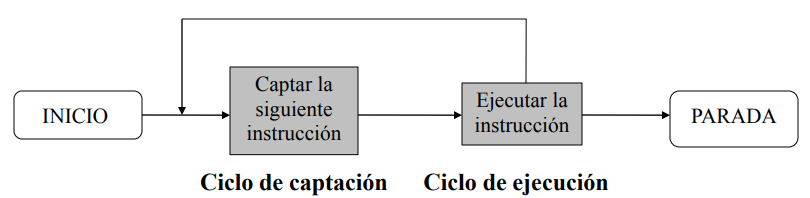
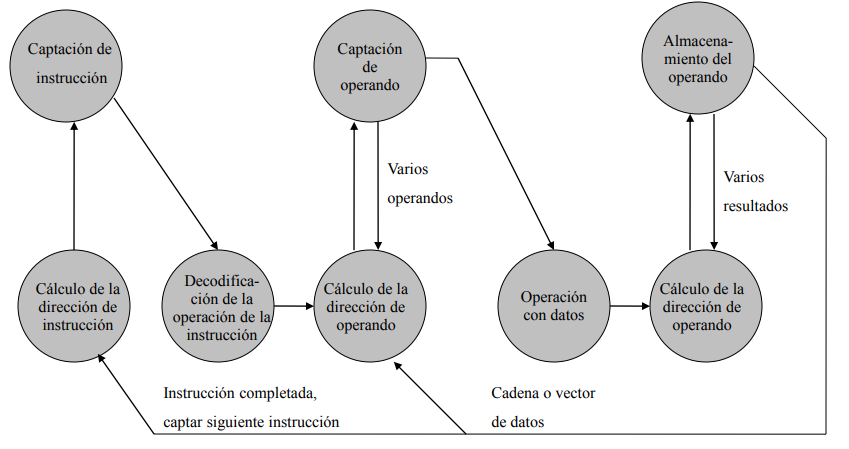
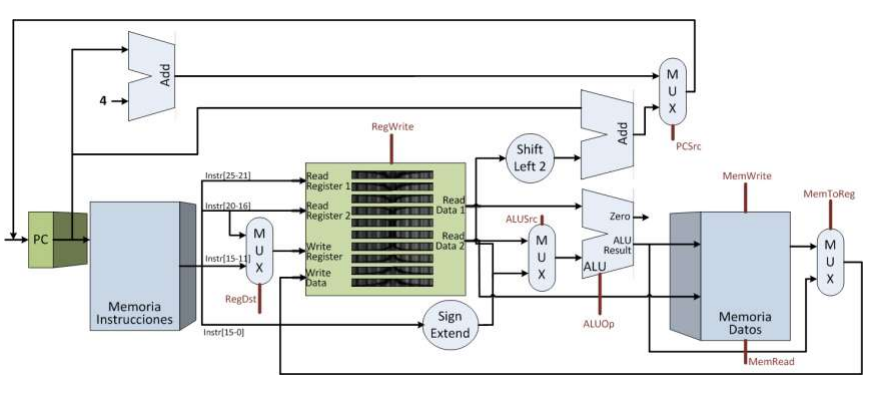


Diagrama de estados del ciclo de instrucción:



Ruta de datos:



### Ciclo de captación

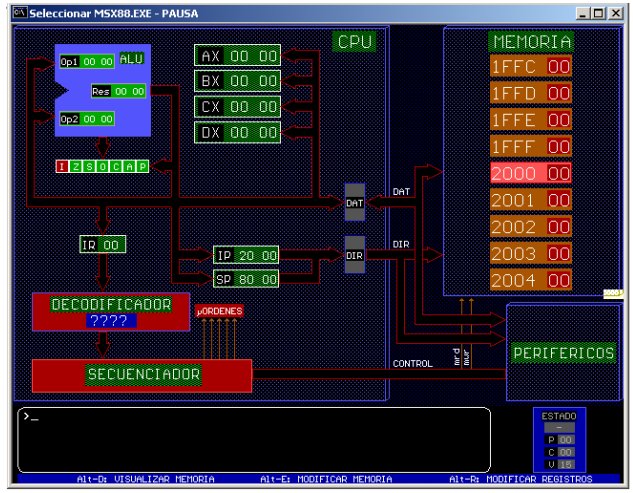
* La dirección de la instrucción que se debe captar esta en el PC
* La UC capta la dirección desde la memoria (la instrucción va al IR)
* Se incrementa PC (salvo que se indique lo contrario)
* La UC interpreta la instrucción captada y lleva a cabo la acción

### Ciclo de ejecución

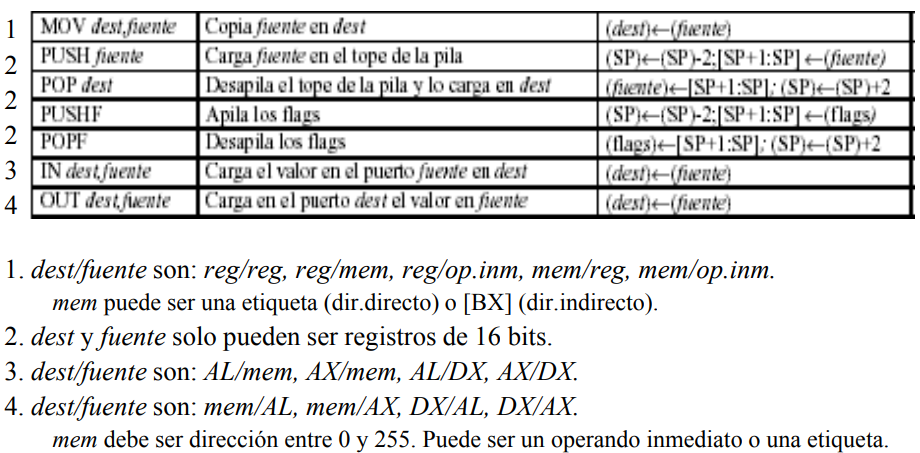
Acciones posibles:

* Procesador-memoria (transferencia de datos CPU-Memoria)
* Procesador-E/S
* Procesamiento de datos (Alguna operación aritmética o lógica con los datos)
* Control (Alteración de la secuencia de ejecución -instrucción de salto-)
* Combinación de las anteriores

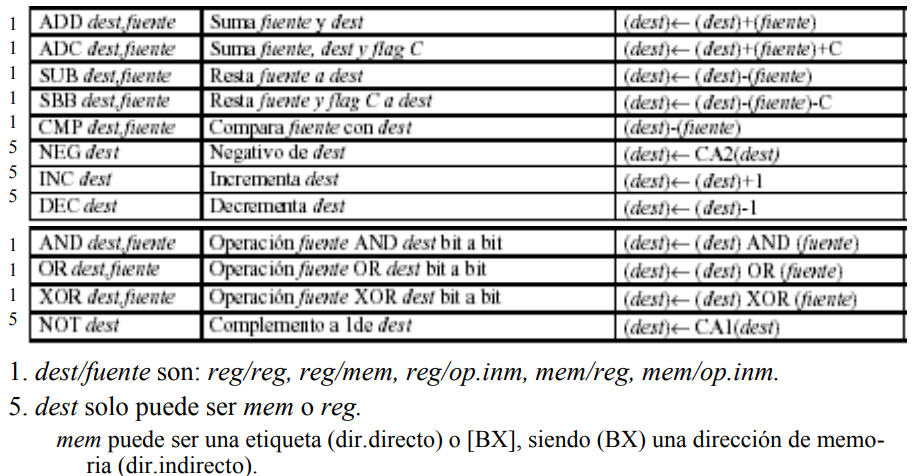
## Simulador



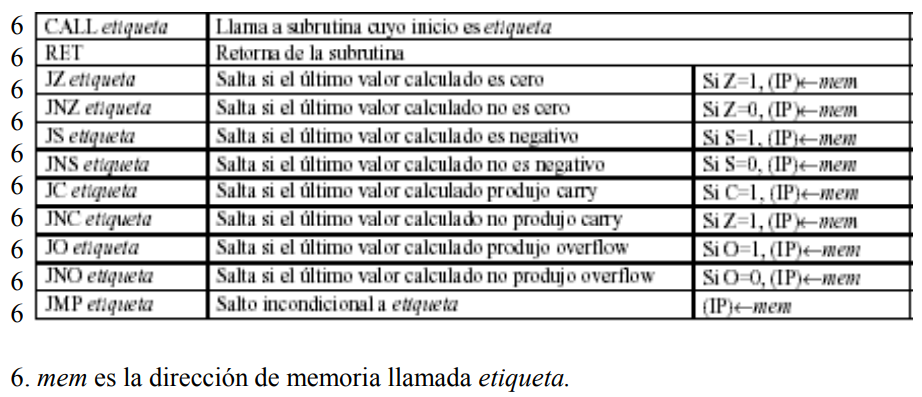
### Instrucciones de transferencia



### Instrucciones aritméticas y lógicas



### Instrucciones de transferencia de control



# Subrutinas

* Innovación en lenguajes de programación
* Programa auto-contenido
* Puede invocarse desde cualquier punto de un programa (instrucción CALL)
* Brinda economía (código reusable) 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

* Via registros
  + Limitado por el número de registros
  + Es importante documentar qué registros se usan
* Vía memoria
  + Se usa un área definida de memoria (RAM)
  + Difícil de estandarizar
* Vía pila (stack)
  + Método más usado
  + El verdadero “pasaje de parámetros”
  + Independiente de memoria y registros
  + Hay que comprender bien su funcionamiento porque la pila es usada por el usuario y por el sistema (en x86, SP apunta al último lugar usado)

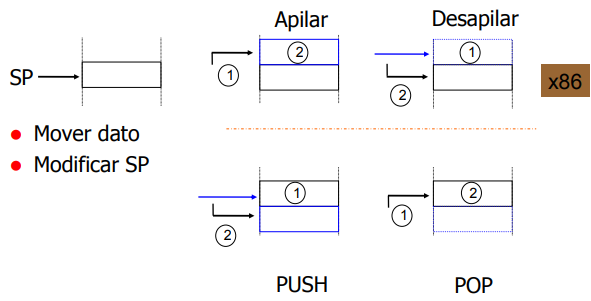
### Funcionamiento de una pila

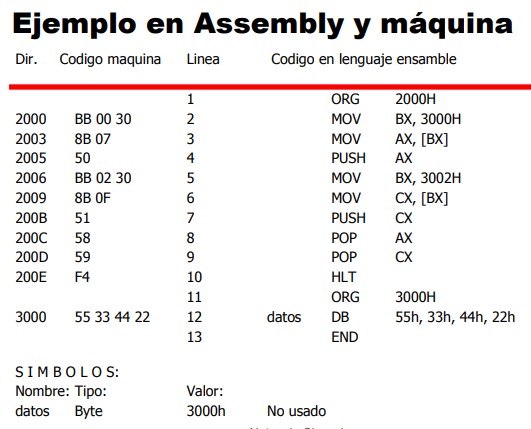
* El operando está (implícito) en la cabeza de la pila
* Requiere un registro puntero de pila (SP) que contiene la dirección de la cabeza de la pila
* Operaciones: PUSH y POP

### Operaciones PUSH y POP

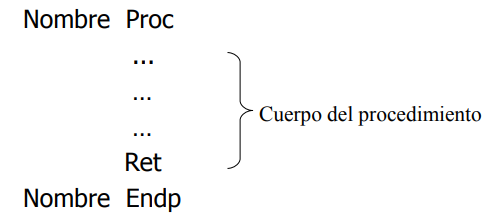
* Secuencia de dos acciones:
  + Movimiento de datos Reg-Mem o Mem-Reg
  + Modificación del puntero antes/después de la anterior
* Tener en cuenta:
  + Dónde apunta el puntero
  + Cómo crece la pila

### Funcionamiento de la pila

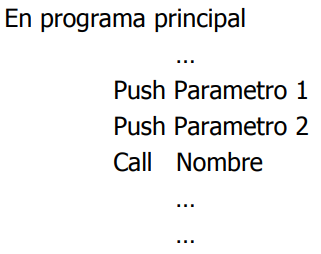


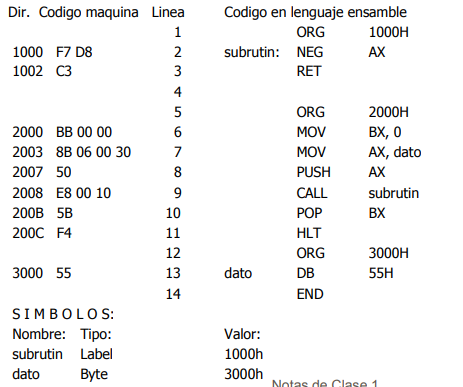
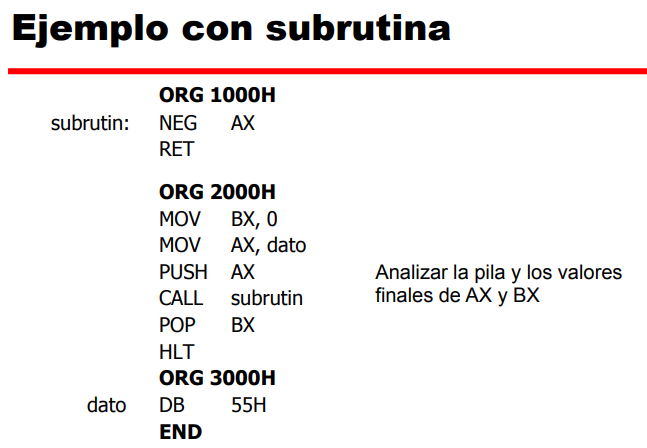


### Definición del procedimiento



### Llamada al procedimiento





### Posibles pasos en un procedimiento

1. **Salvar el estado de BP (viejo BP)**
   1. Push BP

*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'.*

1. **Salvar el estado de SP (BP=SP)**
   1. Mov BP, SP

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

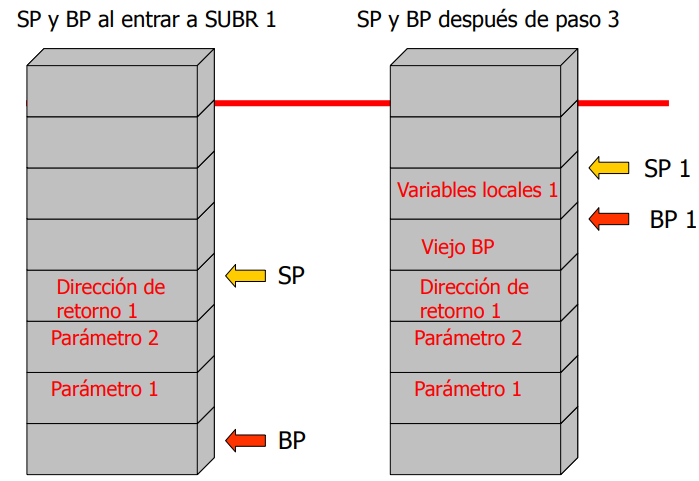
1. **Reservar espacio para datos locales (opcional)**

Se decrementa SP, reservando lugar en la pila

1. sub SP, 2

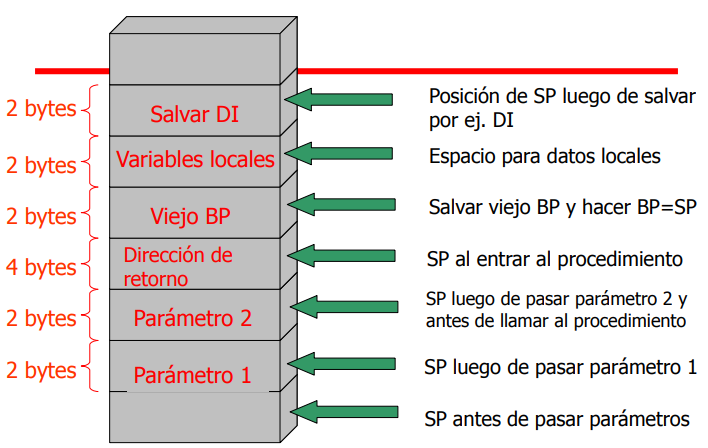
Este ej. reserva 2 bytes para datos locales.

El sistema puede utilizar al SP sin escribir sobre el área de trabajo (o frame) del procedimiento.



1. Salvar valores de otros registros (opcional)
   1. 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.



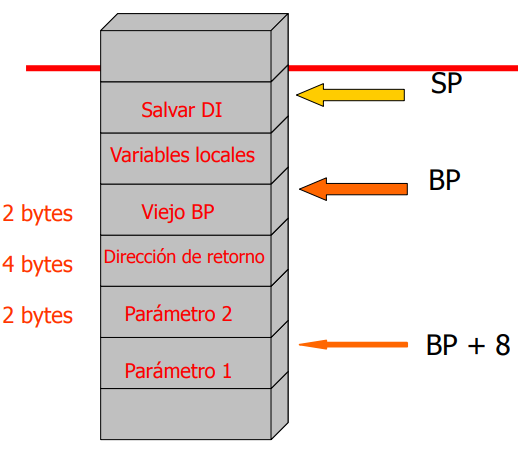
1. Acceder a parámetros

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:

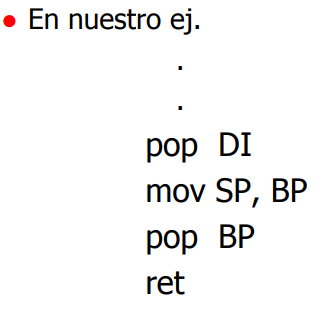
* mov CX, [ BP + 8 ]

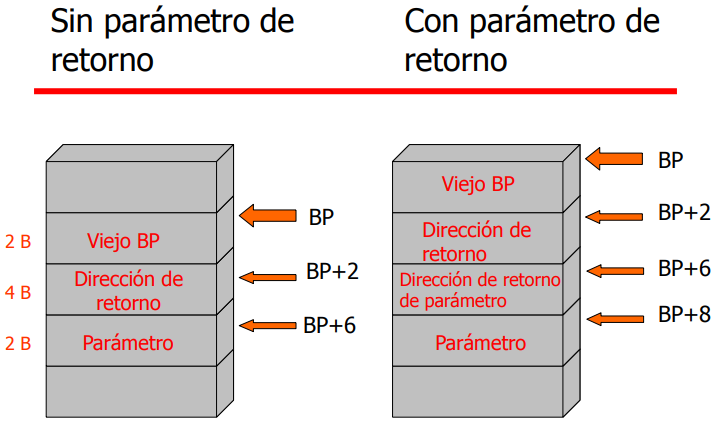


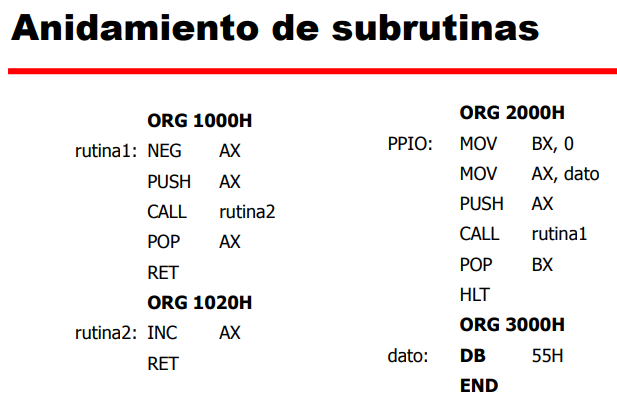
1. Escribir sentencias a ejecutar
2. Retornar parámetro (opcional)
3. Regresar correctamente del procedimiento

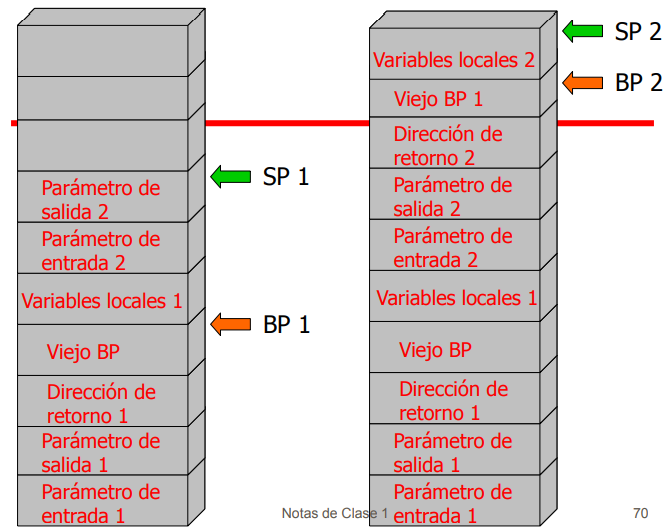
### Salida del procedimiento

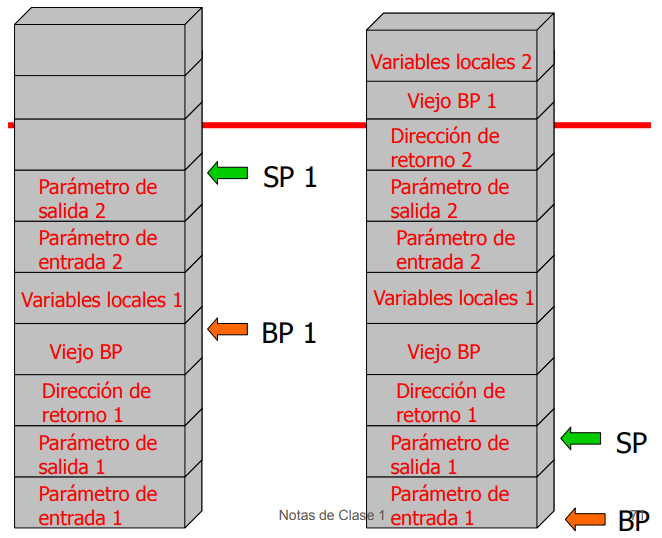
* Los registros salvados en la pila deben ser descargados en orden inverso
* Si se reservó espacio para variables locales, se debe reponer SP con el valor de BP que no cambio durante el procedimiento
* Reponer BP
* Volver al programa que llamó al procedimiento con RET

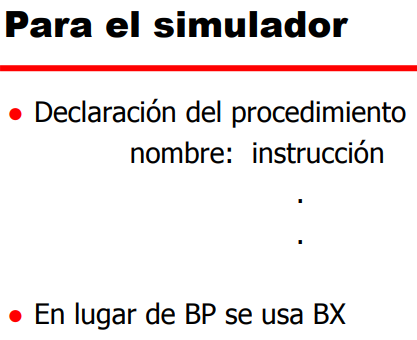


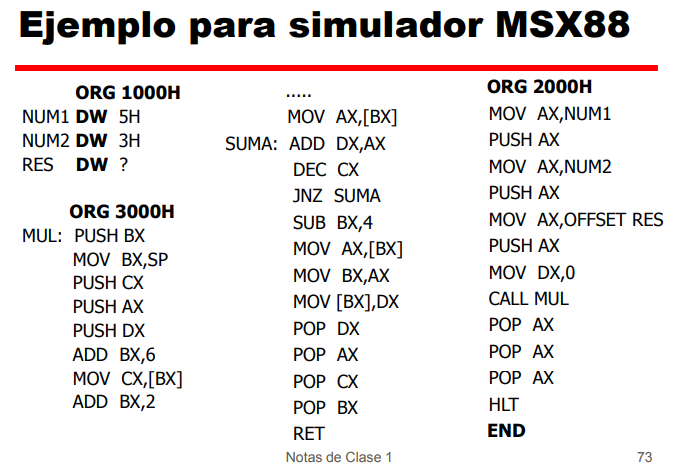


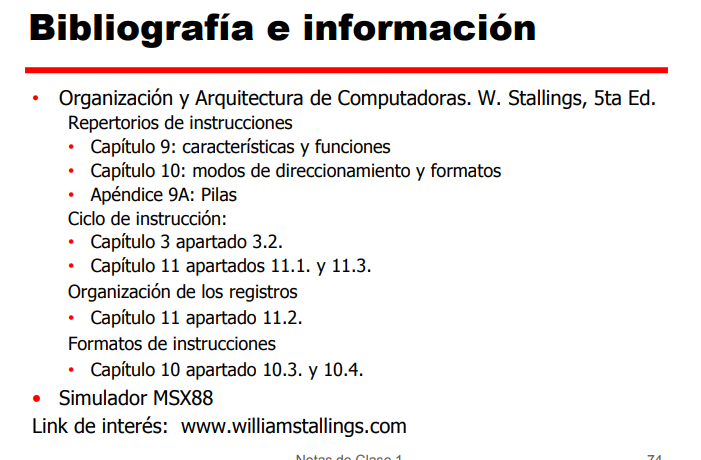












[Anexo clase 1](https://drive.google.com/file/d/1wBvlKTA1RiYkj7CSbaFlY_0CqtKG-FrH/view?usp=sharing)

Maquinas de 4,3,2,1 y 0 direcciones.

