Arquitectura de Computadoras

CURSO 2022

Turno:

Clase 1

Resumen clase 1

Temas básicos – repaso conceptos de Organización

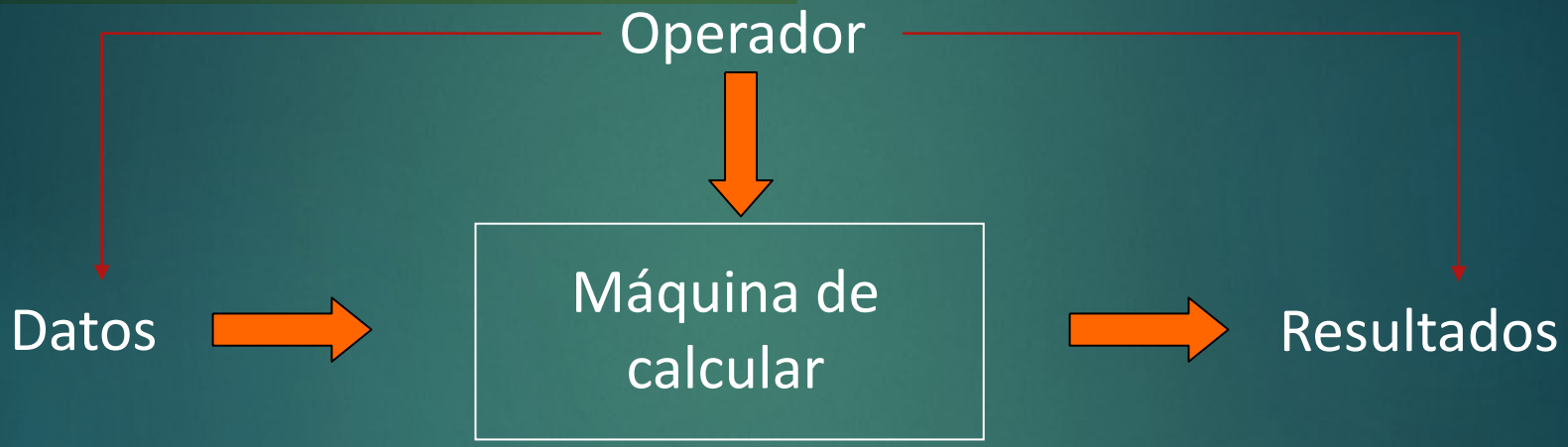
- Programas e instrucciones
- Modelo de von Neumann
- Lenguajes
- Instrucciones de máquina
- Ciclo de instrucción
- Simulador MSX-88
- Pila
- Subrutinas

Concepto de “programa”

- Se puede considerar como antecedente más cercano de los Sistemas de Cómputo a las máquinas de calcular.
- Las primitivas máquinas de calcular eran de tipo mecánico, capaces de realizar una sola operación (por ejemplo suma o resta) a la vez.
- Si el cálculo a resolver requería un solo tipo de operación (por ejemplo sumas o restas repetitivas) solo era necesario ingresar los datos de cada operación.
- Si, por el contrario, también se cambiaba la operación, se requería además modificar la máquina.
- En ambos casos, era necesaria la intervención de una persona (“operador”) para hacer esas tareas.

Concepto de “programa”

- Conceptualmente, esta forma de operación se puede asimilar al siguiente modelo:



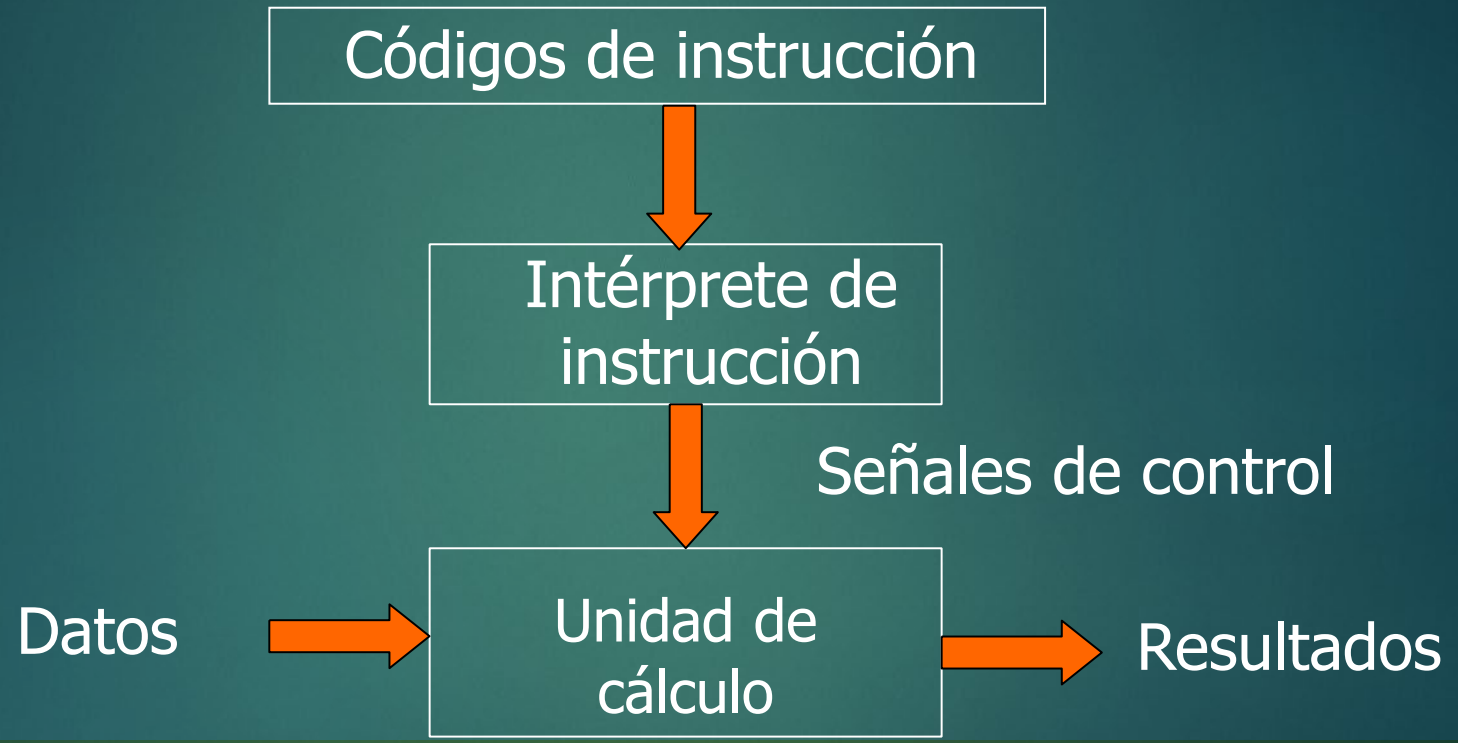
- Este modelo en el que un “operador” tiene que actuar (modificar) sobre la máquina (es decir la “Unidad de Cálculo”) de acuerdo a la tarea a ejecutar, se puede considerar como un “modelo de Programación en hardware”, porque cuando se cambia la tarea, se requiere cambiar el medio físico que la ejecuta.

Concepto de “programa”

- En un Sistema de cómputo, la secuencia de tareas está almacenada en la máquina como instrucciones.
- Las instrucciones son interpretadas por la propia máquina en una unidad especial (Unidad de Control CU o Intérprete de instrucciones por hardware).
- Las instrucciones son convertidas en señales que comandan: la entrada de los datos, el tipo de operación, y la salida de los resultados.
- La Unidad de cálculo es universal y capaz de resolver distintos tipos de operaciones. El tipo de operación a resolver lo obtiene desde el Intérprete de instrucciones a partir de la instrucción leída.

Concepto de “programa”

- El modelo de operación cambia de la siguiente manera:



- Este nuevo modelo, en el que las operaciones se definen por medio de instrucciones almacenadas en memoria, se puede considerar como un “modelo de Programación en software”, porque el cambio de la tarea, solo require modificar la secuencia de instrucciones.

Concepto de “programa”

- En el modelo de Programación en software, la información requerida para ejecutar cada paso de la tarea es proporcionada por la secuencia de instrucciones a ejecutar.
- Cada instrucción es convertida en señales que son enviadas a las unidades funcionales.
- Cambiar la respuesta del Sistema implica únicamente modificar la secuencia de instrucciones (“programa”), que es fácilmente modificable (el “software”) debido a que está almacenada en una memoria dentro de la propia máquina.
- Por otra parte, las unidades funcionales (el “hardware”) no deben modificarse, su respuesta se modifica desde el programa (“software”).

Concepto de “programa”

- Cada instrucción contiene la información necesaria para resolver esa acción.
- Se requiere de un “intérprete por hardware” (la Unidad de control) que convierta la instrucción (en realidad decodifica) en el conjunto de señales requeridos por el hardware.
- Básicamente, un programa es una secuencia de instrucciones que realiza una tarea determinada.
- El programa puede ser modificado de una manera relativamente fácil (mucho más fácil que modificar el hardware).
- No hay que cambiar el hardware, es siempre el mismo. Cambia su comportamiento (y los resultados que produce) de acuerdo a la secuencia de acciones que ejecuta.

Concepto de “programa”

En definitiva, un Programa es:

- Una secuencia de acciones almacenadas en el sistema de cómputo.
- Cada acción es una instrucción (de máquina).
- Cada instrucción es convertida en señales.
- Diferentes señales de control se necesitan para cada operación.
- la Unidad de Control (CU) obtiene toda la información necesaria para resolver (ejecutar) la acción definida en la instrucción.

Modelo de von Neumann - IAS

- Las primeras máquinas (“Generación 1”) con características típicas de un Sistema de cómputo surgieron a principios de la década del 40.
- Impulsado por resolver cálculos matemáticos más o menos complejos en un tiempo relativamente pequeño.
- La Generación 1 eran máquinas destinadas a resolver básicamente problemas matemáticos (cuentas).
- El ámbito de aplicación era el de las áreas científica y militar.
- Usaban 2 tipos de tecnologías:
  - válvulas electrónicas (como las usadas en los primitivos televisores y radios, entre otros equipos eléctrónicos)
  - dispositivos electromecánicos (relés)

Modelo de von Neumann - IAS

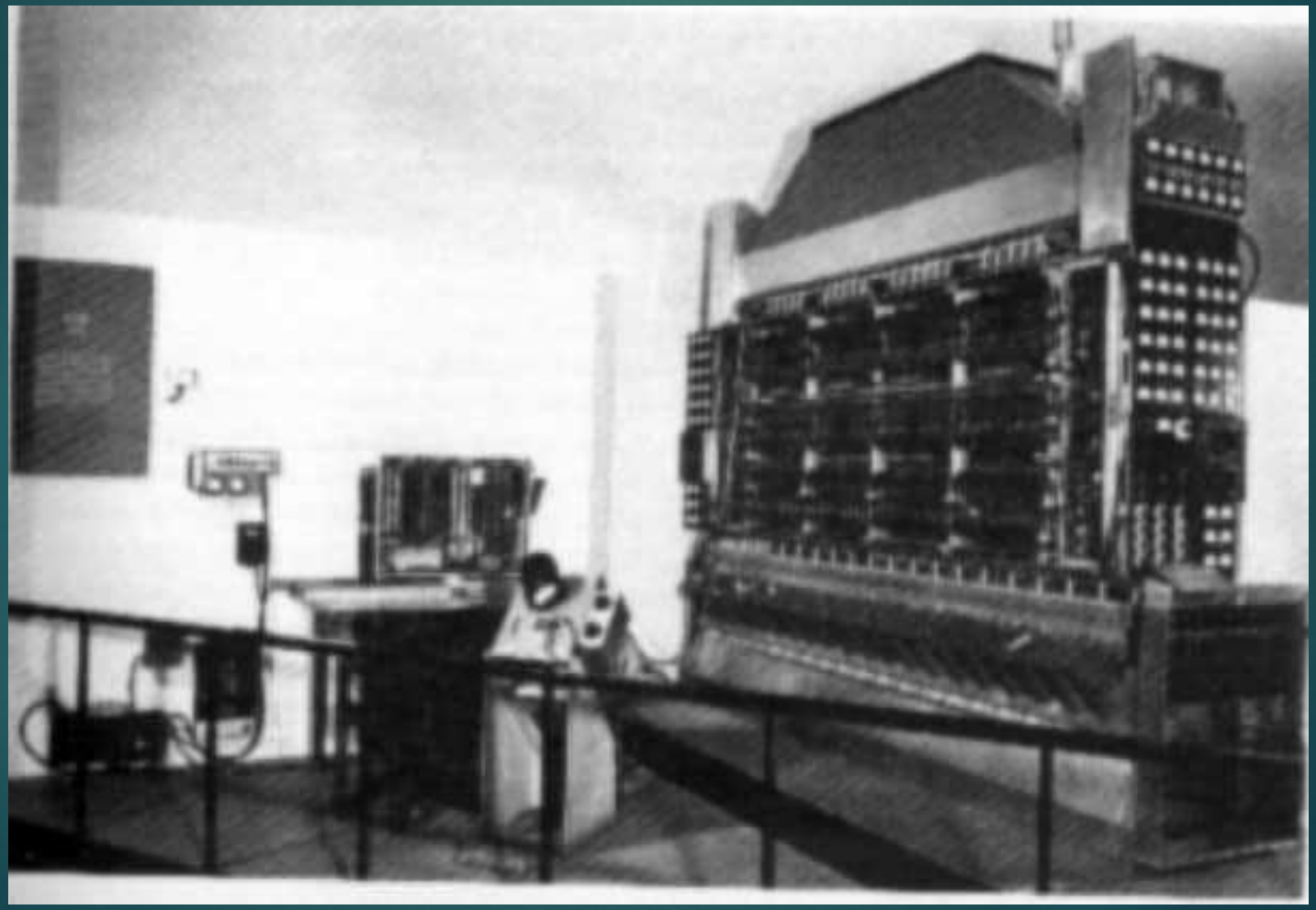
- A mediados de la década del 40, había varios sitios donde se estaban desarrollando computadoras con características parecidas.
- Pero hubo una que sobresalió sobre el resto debido a que usaba conceptos novedosos no empleados en otros desarrollos.
- La máquina se conoció como IAS, y fué desarrollada por John von Neumann en la Universidad de Princeton (NJ).
- El Proyecto comenzó en 1942 y los últimos ajustes se completaron entre 1951 y 1952.

Modelo de von Neumann - IAS

- Una de las novedades más importantes de la IAS era que operaba en binario (en lugar de decimal).
- Tenía una memoria única (para instrucciones y datos), con 4096 palabras de 40 bits.
- El repertorio de instrucciornes estaba compuesto por 21 instrucciones de 20 bits (incluyendo transferencia de datos, saltos, aritméticas, cálculo de dirección)
- El banco de registros de almacenamiento en la CPU estaba compuesto por:
  - Registro Buffer de Memoria (MBR)
  - Registro de Direcciones de Memoria (MAR)
  - Registros de Instrucción y Buffer de Instrucción
  - Registro Contador de Programa (Program Counter)
  - Registros Acumulador y Multiplicador/Cociente

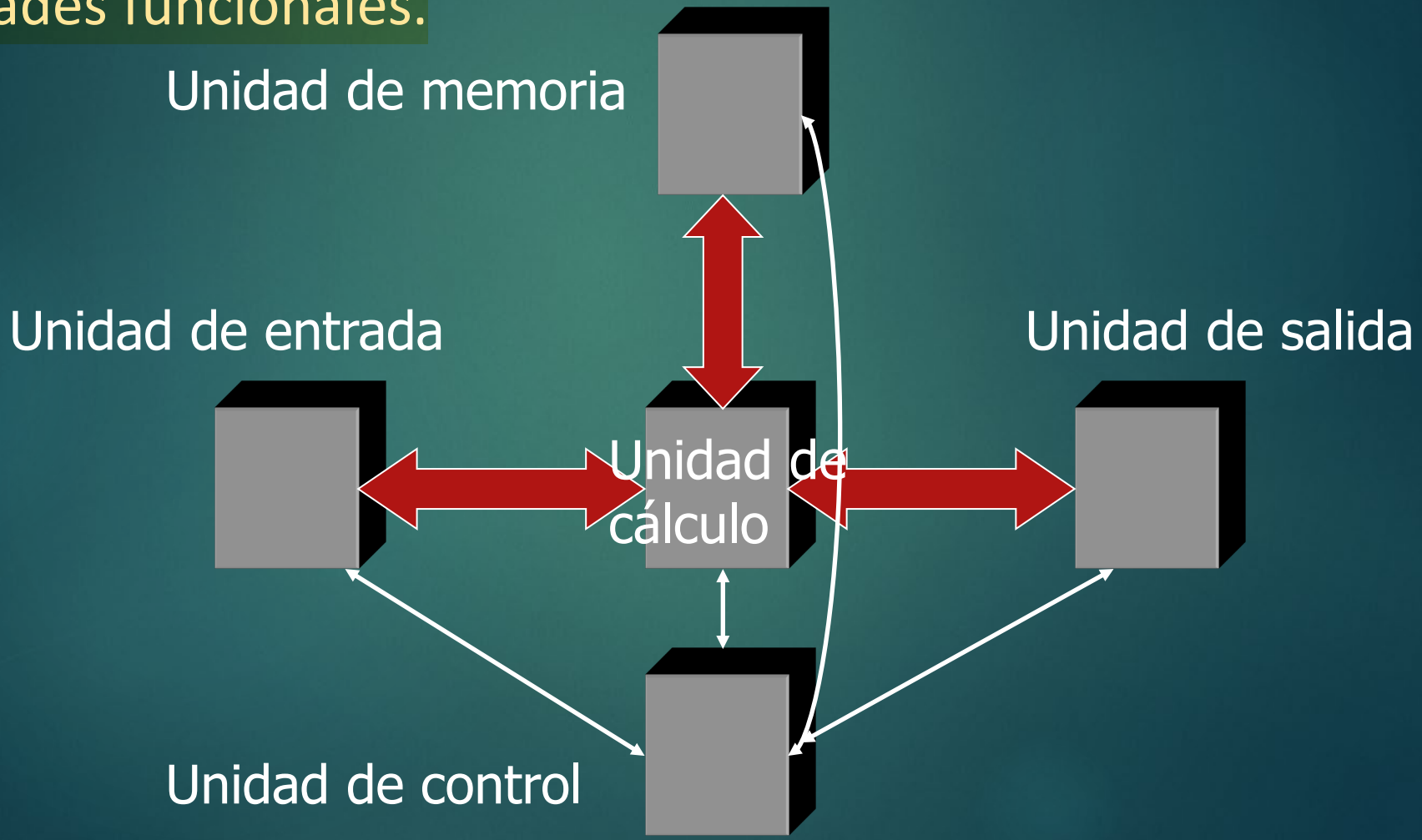
Modelo de von Neumann - IAS

En la imagen siguiente se muestra una foto de la IAS.



Modelo de von Neumann de 5 subsistemas

El modelo de von Neumann de la IAS estaba basado en 5 Unidades funcionales.



Modelo de von Neumann de 5 subsistemas

Las 5 Unidades funcionales eran:

- Unidad de Control (UC)
- Unidad Aritmético-Lógica (ALU)
- Unidad de Memoria
- Unidad de Entrada (de datos)
- Unidad de Salida (de datos)

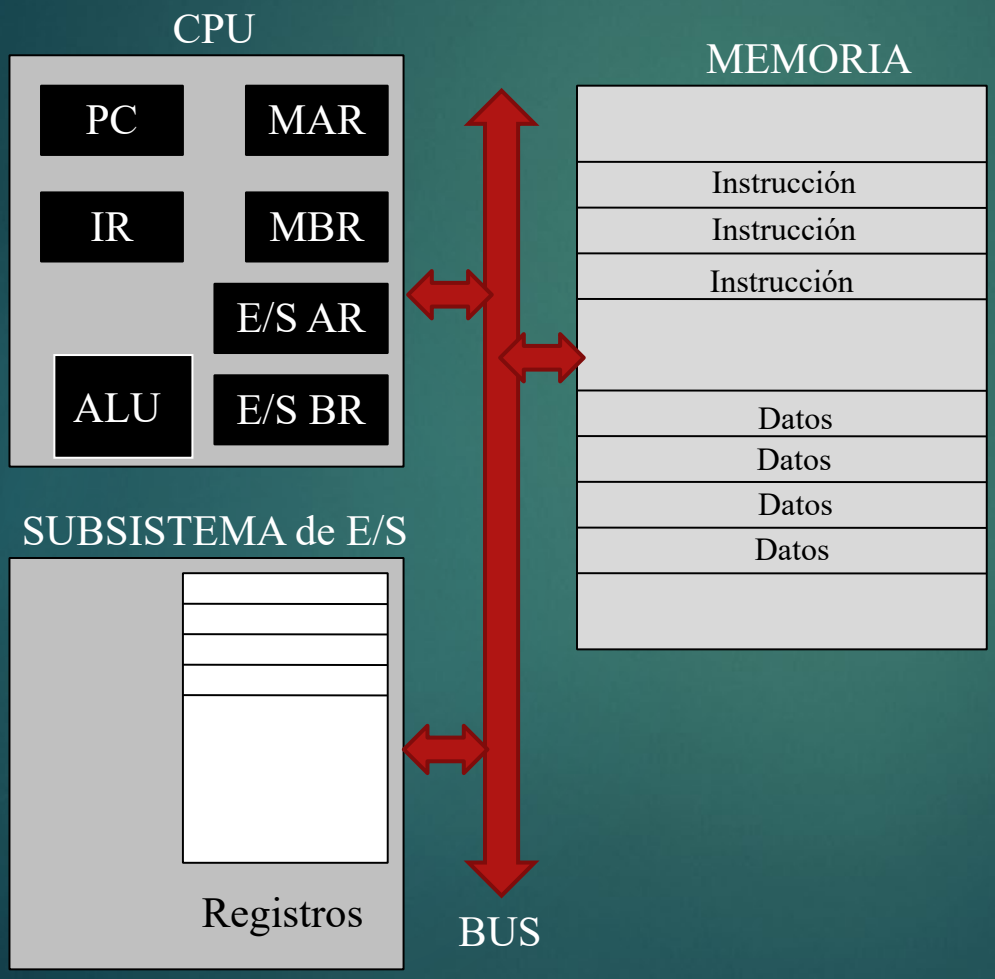
Este modelo de Arquitectura se puede simplificar en un modelo basado en 3 Unidades funcionales, modelo que perdura actualmente en muchas máquinas de propósito general.

Modelo de von Neumann de 3 Subsistemas

- La UNIDAD CENTRAL DE PROCESAMIENTO (CPU) constituída por:
  - Unidad de Control (UC)
  - Unidad Aritmético-Lógica (ALU)
  - Registros.
- La UNIDAD DE ENTRADA/SALIDA (E/S) para introducir y extraer los datos e instrucciones.
- La UNIDAD DE MEMORIA para almacenar temporalmente datos e instrucciones.

Modelo de von Neumann de 3 Subsistemas

Modelo de von Neumann con 3 subsistemas



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

Lenguajes

Lenguaje de máquina o absoluto

- El lenguaje que la máquina es capaz de “entender” y ejecutar se conoce como “lenguaje absoluto o de máquina”, y es del tipo binario.
- Las instrucciones están codificadas mediante un conjunto de bits almacenados en palabras de memoria (1 palabra o más).
- La representación de las instrucciones en memoria es puramente binaria.

Lenguajes

Lenguaje simbólico (Assembly)

- Dado que es prácticamente imposible para un programador tratar con las representaciones binarias de las instrucciones de máquina, se usan otras representaciones más "amigables".
- Estas representaciones tienen distintos grados o niveles de abstracción.
- En el nivel siguiente al nivel de lenguaje de máquina, en cuanto a nivel de abstracción, está la representación simbólica (Lenguaje Assembly).
- En la representación simbólica, los campos que forman la instrucción se pueden representar con textos o números.

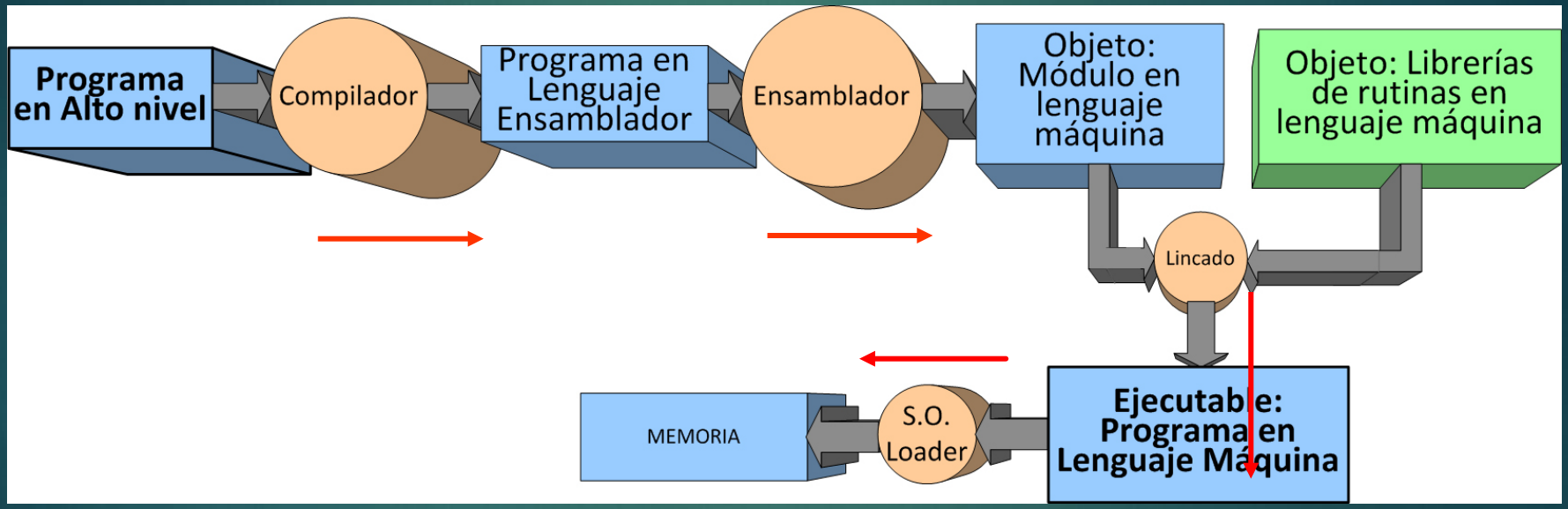
Lenguajes

Lenguaje simbólico (Assembly)

- En el lenguaje Assembly, los diferentes campos de la instrucción se pueden reemplazar por números o símbolos más fáciles de usar que el binario absoluto.
- Por ejemplo, el campo binario de la instrucción que contiene la información del tipo de instrucción (campo de código de operación COP) puede ser reemplazado, en el lenguaje Assembly por mnemónicos más sencillos de memorizar:
  - ADD (significa código de operación de) adición (suma)
  - SUB sustracción (resta)
  - MOV movimiento de datos
  - AND, OR, XOR operaciones lógicas

Lenguajes

El proceso de transformación (compilación) de un programa escrito en lenguaje de alto nivel (HLL) hasta el lenguaje de máquina típicamente es el siguiente.



Elementos de una instrucción de máquina

- Una instrucción de máquina es un código binario compuesto por una cantidad determinada de bits.
- Los bits son usados por la Unidad de Control (CU) para continuar y completar el proceso de ejecución de la instrucción.
- Para ello, el código debe contener suficiente “información” para que la Unidad de Control pueda resolverla.
- Las instrucciones tienen distinto grado de complejidad. Van desde las que no hacen nada, hasta las que operan sobre múltiples datos.

Elementos de una instrucción de máquina

- En general las instrucciones más complejas requieren de múltiples pasos o acciones. Las más sencillas se resuelven en pocos pasos.
- Un ejemplo de instrucción “relativamente compleja” es la que busca 2 operandos y produce un resultado. Para esta instrucción la CPU necesita definir:
  - 1: tipo de operación
  - 2: Lugar del 1er operando (donde está)
  - 3: Lugar del 2do operando (donde está)
  - 4: Lugar del resultado (donde guardarlo)
  - 5: donde está la próxima instrucción

Elementos de una instrucción de máquina

- Dado que hay diferentes datos que proveer, la instrucción está organizada en conjuntos de bits, comúnmente denominados campos, que proveerán esos datos.
- La cantidad de bits de un campo dependen de la cantidad de información que debe proveer.
- Por ejemplo, si un procesador dispone de 256 tipos de instrucciones, y un campo se usa para especificar el tipo de instrucción (conocido comúnmente como campo de Código de Operación COP), entonces ese campo requiere de al menos 8 bits.

Elementos de una instrucción de máquina

Los campos que típicamente pueden estar incluidos en una instrucción son:

- 1.- Campo de Identificación de la instrucción (Código de operación):
  - Especifica la operación a realizar (ej. sumar).
- 2.- Campos de Referencias a operandos:
  - Establece la información de referencia a el o los operandos fuentes (si es que se requieren). La información de referencia puede ser de distinto tipo (de acuerdo al modo de direccionamiento).
  - La operación puede involucrar uno ó más operando fuente (o de entrada). Por ejemplo en una suma se requiere especificar 2 operandos.

Elementos de una instrucción de máquina

- 3.- Campo de Referencia del resultado:
  - Establece dónde almacenar el resultado.
  - La operación puede involucrar uno o más resultados (de salida). Por ejemplo, en una suma se produce 1 resultado solamente.
- 4.- Campo de Referencia de la siguiente instrucción
  - Provee a la CPU con información para determinar donde buscar la siguiente instrucción después de la ejecución de la instrucción anterior.
  - En la mayoría de los casos, la próxima instrucción se ubica a continuación de la instrucción actual, es decir, en la siguiente posición de memoria. Esta situación se conoce como acceso secuencial a instrucciones consecutivas de memoria.

Taxonomía basada en formato de instrucción

- Una forma de clasificación de las máquinas está basado en el formato de instrucción.
- La clasificación identifica la cantidad de campos que contienen direcciones de memoria.
- Suponiendo el caso de una instrucción de suma de 2 operandos en memoria, y resultado almacenado en memoria, se necesitan:
  - 2 campos de direcciones para hacer referencia a los operandos.
  - 1 campo de dirección para hacer referencia a donde almacenar el resultado.
  - 1 campo de dirección de referencia de la ubicación de la próxima instrucción.

Taxonomía basada en formato de instrucción

En base a lo anterior, se pueden tener 5 tipos de maquinas básicos:

- Máquina de 4 direcciones:

- Máquina de 3 direcciones:

- Máquina de 2 direcciones:

Taxonomía basada en formato de instrucción

- Máquina de 1 dirección:

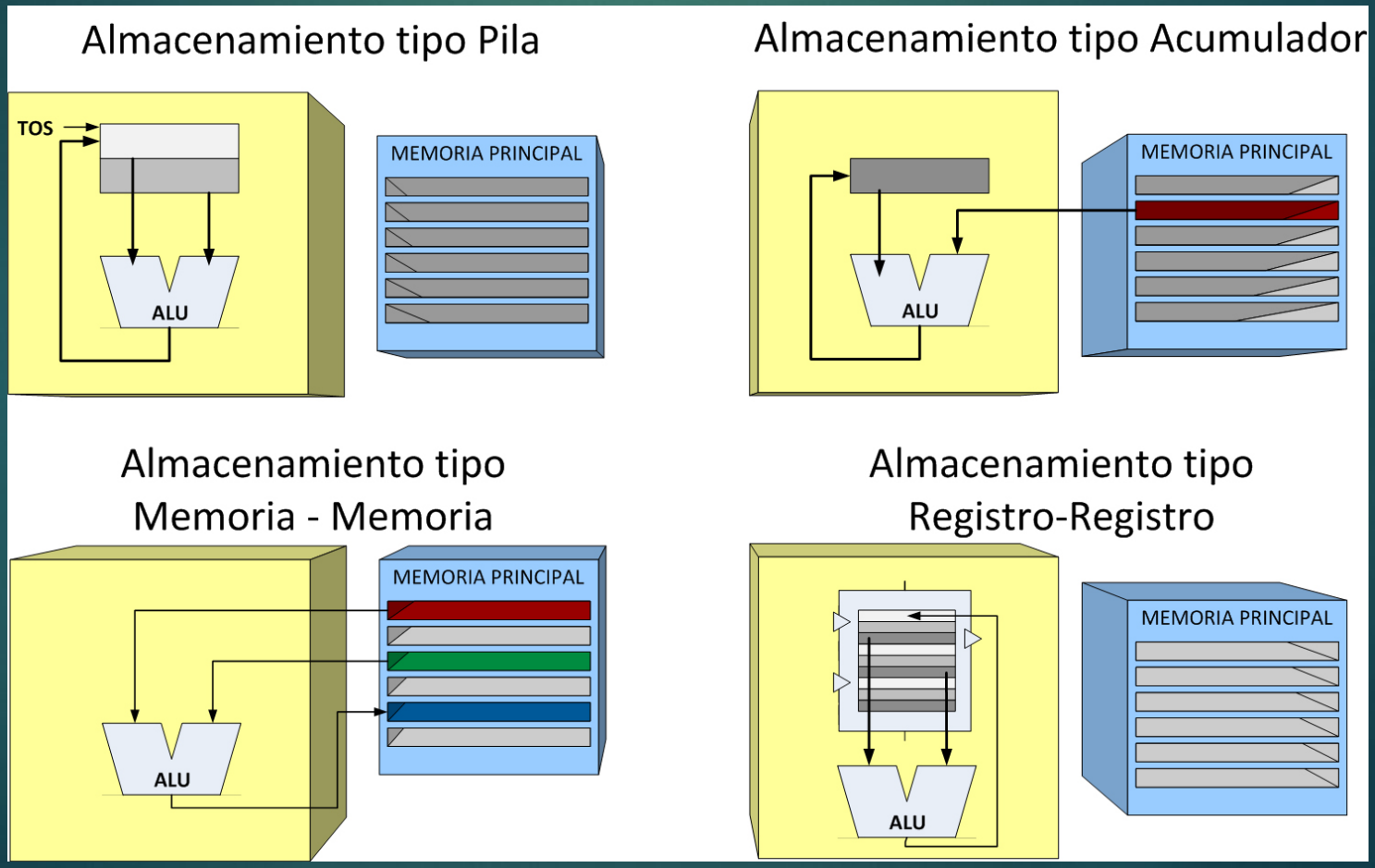
- Máquina de 0 dirección (máquina tipo Pila):

Taxonomía basada en los tipos de almacenamientos de datos

- Otra forma de clasificación de las máquinas está basada en los tipos de almacenamiento de los datos.
- Los datos pueden estar almacenados en 3 lugares físicos:
  - Memoria: cache, principal, virtual. En todos los casos se requiere para referenciar el dato, una dirección de memoria.
  - Registro de la CPU: la referencia es un número de registro
  - Dispositivo de E/S: la referencia es un número de registro de E/S

Taxonomía basada en los tipos de almacenamientos de datos

- Suponiendo una instrucción con referencias a 2 operandos y 1 resultado, podemos tener 4 tipos de máquinas básicas.



Decisiones en el diseño del conjunto de instrucciones

Cuando se diseña la arquitectura de un procesador son muchos los aspectos que se tienen en cuenta. Algunos de los más relevantes son:

1) Formato de instrucción
2) Tipo de datos
3) Repertorio de instrucciones
4) Cantidad y tipo de registros
5) Modos de direccionamiento

Decisiones en el diseño del conjunto de instrucciones

1) Formato de instrucción

Se refiere a la estructura que tendrán las instrucciones. 2 aspectos son relevantes en el diseño del formato de instrucción:

- Formato fijo o variable
  - Formato fijo: todas las instrucciones tienen el mismo tamaño (bits). La captación y ejecución es más sencilla pero aumenta la cantidad de instrucciones de los programas.
  - Formato variable: la longitud de las instrucciones varía con el tipo de instrucción y modo de direccionamiento. La captación y ejecución es más compleja pero disminuye la cantidad de instrucciones de los programas.

Decisiones en el diseño del conjunto de instrucciones

- Cantidad de campos de direcciones
  - A mayor cantidad de referencias de direcciones, se tienen instrucciones más complejas y largas, menos instrucciones por programa, y captación y ejecución de las instrucciones en forma más lenta.
  - A menor cantidad de referencias de direcciones, se tiene instrucciones más simples y cortas, más instrucciones por programa, captación y ejecución de las instrucciones más rápida.

Decisiones en el diseño del conjunto de instrucciones

2) Tipos de datos

Una decisión relevante, referida a las características del procesador y del repertorio de instrucciones, es la cantidad y tipo de datos que puede soportar.

Los tipos de datos más comunes son:

- Numérico: Con/sin signo, punto fijo / punto flotante, BCD, etc.
- Caracter: ASCII, EBCDIC, etc.
- Datos lógicos: manipulación de bits

Decisiones en el diseño del conjunto de instrucciones

3) Repertorio de instrucciones

En el diseño del repertorio de instrucciones, se deben considerar, de manera prioritaria:

- La cantidad de instrucciones (operaciones)
- Tipos de instrucciones
- Características de las instrucciones (sencillas, complejas, etc.)

Decisiones en el diseño del conjunto de instrucciones

4) Administración y uso de Registros

Dado que los registros disponibles en un procesador son un recurso escaso y tienen un fuerte impacto en la velocidad de ejecución de los programas, deben ser diseñados en form asumamente cuidadosa, en particular:

- Tamaño y cantidad de registros
- Tipos de registros: como va a ser el uso y a que instrucciones se van a aplicar.

Decisiones en el diseño del conjunto de instrucciones

5) Modos de direccionamiento

Un aspecto importante en el diseño del repertorio de instrucciones es el referido a los modos de direccionamiento.

Las decisiones de diseño de los modos de direccionamiento se orientan a:

- Cantidad de modos de direccionamiento
- Tipos de modos de direccionamiento

Decisiones en el diseño del conjunto de instrucciones

Los diseños de las arquitecturas de procesadores se han orientado hacia 2 estrategias:

- Aumentar la complejidad y prestaciones del procesador
- Simplificar y optimizar el Procesadores

- El primer camino conduce a procesadores identificados genéricamente como tipo **CISC** (Computadores de Conjunto Complejo de Instrucciones)
- El segundo corresponde a las familiaas de procesadores **RISC** (Computadores de Repertorio de Instrucciones Reducido)

Decisiones en el diseño del conjunto de instrucciones

- Los procesadores CISC básicamente tienen:
  - Repertorio de instrucciones muy amplio
  - Muchos modos de direccionamiento
  - Formato de instrucción variable
- Los procesadores RISC se caracterizan por tener:
  - Repertorio de instrucciones simplificado
  - Pocos modos de direccionamiento
  - Formato de instrucción fijo

Técnicas de almacenamiento

- Las técnicas de almacenamiento se refieren a la forma en que los datos se guardan en memoria, particularmente en los casos en que ocupan más de una palabra de memoria.
  - Ejemplo:

Supongamos que tenemos el siguiente dato de 32 bits (4 bytes):

98765432H (32 bits)

y supongamos tener una memoria organizada en palabras de 8 bits (1 byte por palabra).

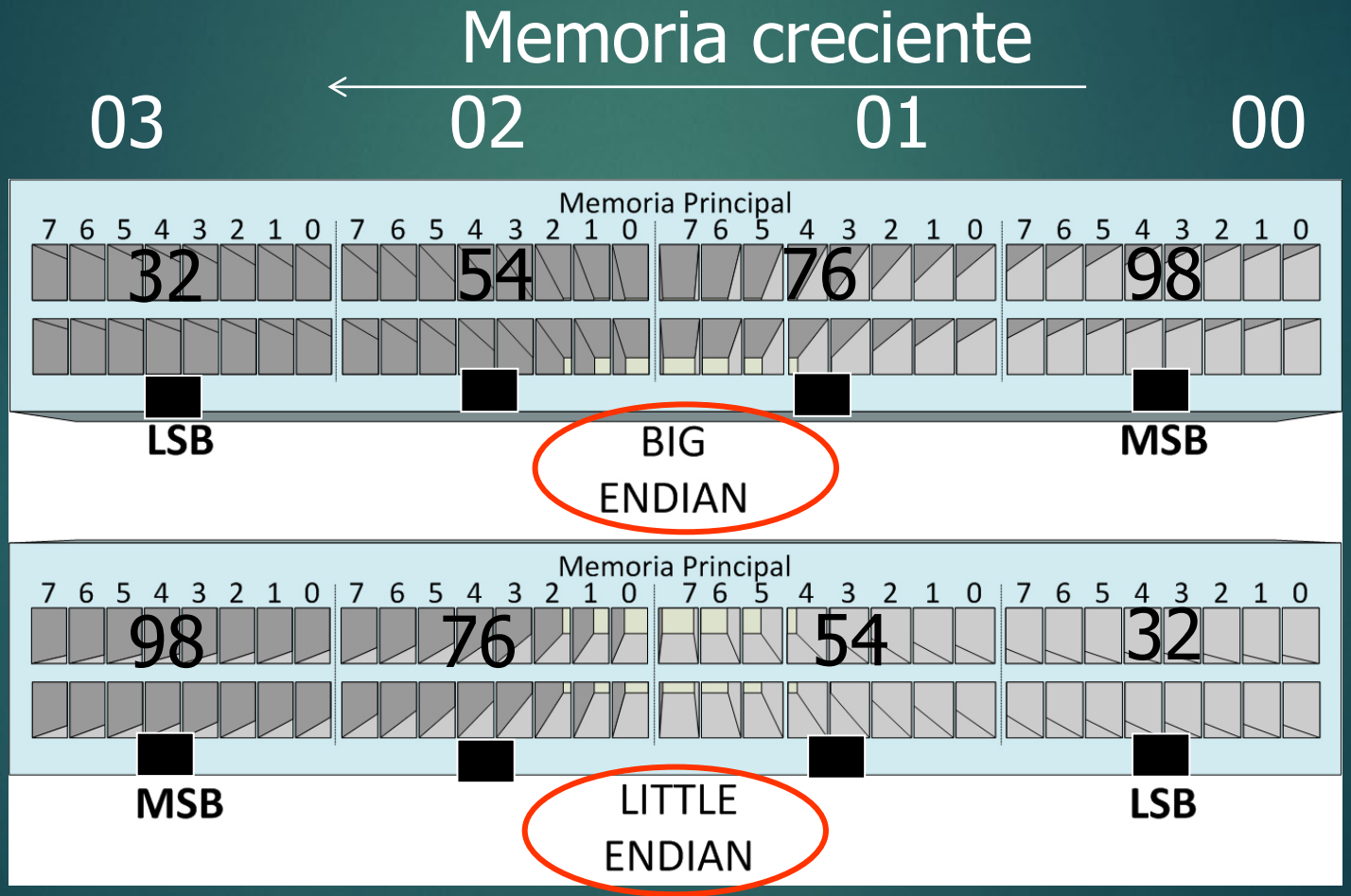
- Existen 2 formas de almacenar el dato (en múltiples palabras de memoria) de acuerdo al orden en que se almacenan las diferentes partes del dato:

Técnicas de almacenamiento

- Las 2 formas se llaman:
  - Big-endian: el byte más significativo se almacena en la dirección con valor numérico más bajo (es decir en la dirección mas baja)
  - Little-endian: el byte mas significativo se almacena en la dirección con valor numérico más alto (es decir, en la dirección más alta).
- En la figura siguiente se muestran las 2 formas.

Técnicas de almacenamiento

Almacenamiento Big-endian y Little-endian



Técnicas de almacenamiento

Es decir que, físicamente, en la memoria quedan almacenados así:

Dir. de memoria	Big-endian	Little-endian
xxx00	98	32
xxx01	76	54
xxx02	54	76
xxx03	32	98

Técnicas de almacenamiento

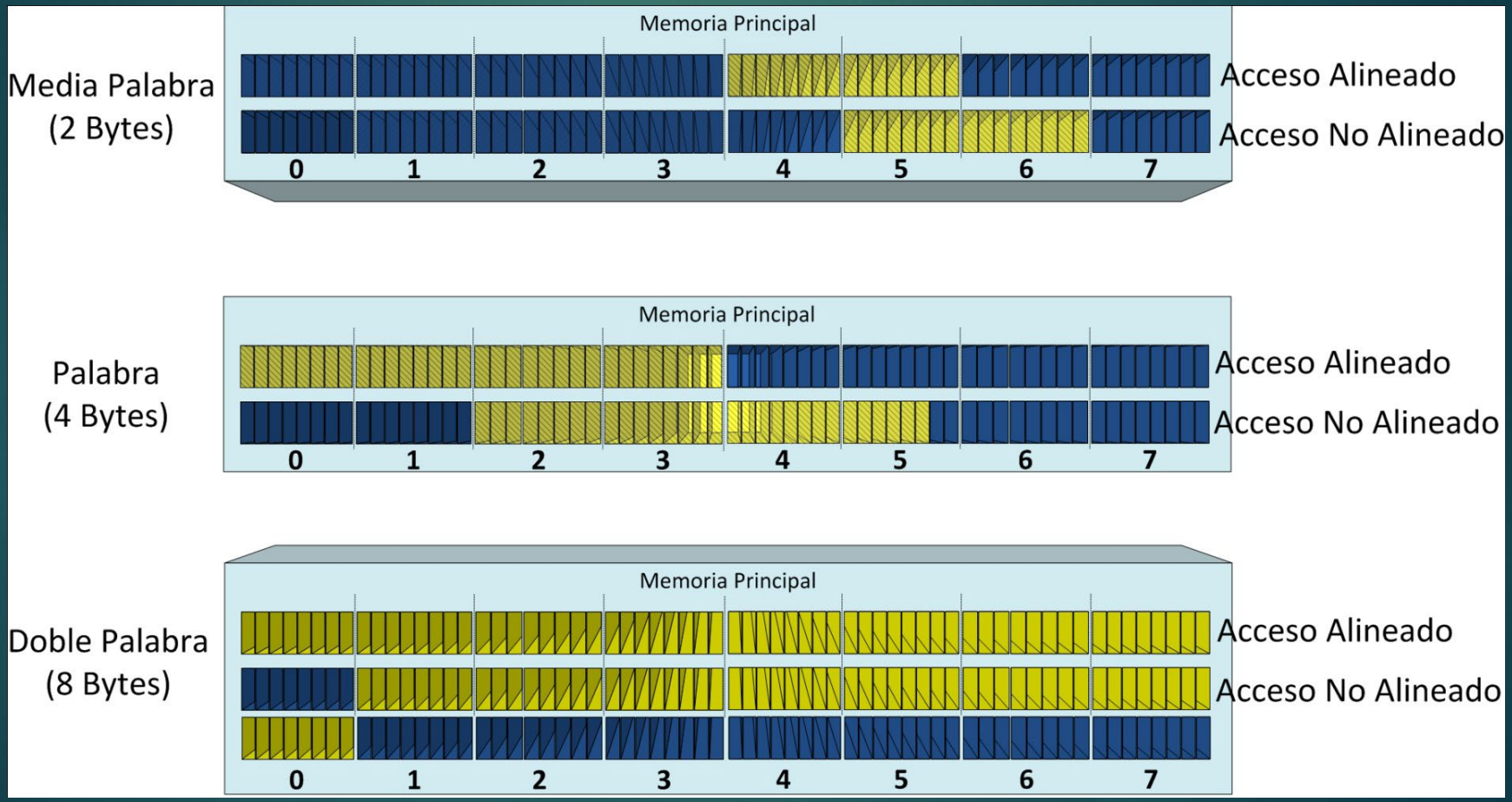
- Ejemplo de procesadores con almacenamiento Little-endian:
  - Intel 80x86
  - Pentium
  - VAX
- Ejemplo de procesadores con almacenamiento Big-endian:
  - IBM S/370
  - Motorola 680x0 (antiguos procesadores de las Mac)
  - La mayoría de los procesadores RISC
- Notar que las 2 formas de almacenamiento son incompatibles entre sí.

Técnicas de almacenamiento

- El otro aspecto relacionado con el almacenamiento de los datos, es el referido a las direcciones de memoria donde se guardan los datos, particularmente en los casos en que ocupan más de una palabra de memoria.
- Existen 2 formas de almacenar los datos múltiples en memoria:
  - Alineados: si un dato de n bytes empieza en una dirección de memoria múltiplo de 2^m, donde $m=\log_2 n$, con $n>1$. En otras palabras, los m bits menos significativos son 0.
  - No alineados: si no cumple la condición anterior.
- En la figura siguiente se muestran las 2 técnicas para los casos de datos de 2, 4 y 8 bytes.

Técnicas de almacenamiento

Almacenamiento alineado y desalineado



Técnicas de almacenamiento

- En los ejemplos anteriores se puede apreciar que:
  - Si el dato es de 2 bytes, el almacenamiento alineado ocurre cuando se almacena a partir de una dirección múltiplo de 2.
  - Si el dato es de 4 bytes, el almacenamiento alineado ocurre cuando se almacena a partir de una dirección múltiplo de 4.
  - Si el dato es de 8 bytes, el almacenamiento alineado ocurre cuando se almacena a partir de una dirección múltiplo de 8.
- Algunos procesadores permiten únicamente acceso alineado. Otros procesadores admiten ambos tipos de accesos, aineados y no alienados. Otros, solo alineados.
- En general, los accesos no alineados son más lentos que los accesos alineados.

Repertorio de instrucciones

Los tipos de instrucciones más comunes que se pueden encontrar en el repertorio de instrucciones de un procesador pertenecen a alguna de las siguientes categorías:

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

Repertorio de instrucciones

Instrucciones de Transferencia de datos

- Son las más típicas. En general se require especificar:
  - Ubicación del operando fuente (usando alguno de los modos de direccionamiento)
  - Ubicación del operando destino (usando aguno de los modos de direccionamiento)
  - Tamaño de los datos a ser transferidos
- En el lenguaje Assembly, los mnemónicos de las instrucciones de movimiento de datos pueden ser:
  - Diferentes mnemónicos para diferentes instrucciones. Por ejemplo: TFR Reg-Reg, LOAD Reg-Mem, STORE Mem-Reg
  - Un solo mnemónico y diferentes formatos. Por ejemplo: MOV destino, fuente ; MOV reg1,reg2

Repertorio de instrucciones

Instrucciones Aritméticas

- Son también muy communes.
- Las instrucciones típicas provistas en un procesador convencional son: Add, Substract, Multiply y Divide.
- Un aspecto importante de estas instrucciones es el tipo de dato que puede manejar: números enteros sin/con signo, números en punto flotante, etc.
- Otras instrucciones aritméticas pueden ser:
  - Increment o Decrement (operando +/- 1)
  - Negate: cambia el signo del operando (en Ca2).
  - Absolute: toma el valor absoluto del operando.
  - Shift left/right: desplazamiento de bits a izq/der un lugar

Repertorio de instrucciones

Instrucciones Lógicas y de rotación y desplazamiento

- Las instrucciones lógicas realizan operaciones que manipulan bits individualmente o en conjunto
- Las más comunes son las funciones de tipo booleana, es decir: AND, OR, XOR, NOT
- Las instrucciones de rotación y desplazamiento son, por ejemplo: Rotate (rota a izq/der), Rotate through carry (rota a través del carry a izq/der), Shift (desplaza a izq/der).

Repertorio de instrucciones

Instrucciones de conversion de tipo

- Son instrucciones poco communes orientadas a cambiar formatos de datos.
- Algunos ejemplos:
  - Conversión de binario a decimal
  - Conversión de EBCDIC a ASCII

Repertorio de instrucciones

Instrucciones de ENTRADA/SALIDA

- Algunos procesadores tienen instrucciones específicas de Entrada/Salida. Los que no tienen este tipo de instrucciones, usan las mismas instrucciones de movimiento de datos con la memoria, para realizar las transferencias de E/S.
- Las instrucciones de E/S típicamente son pocas, con acciones específicas. Por ejemplo: IN, OUT
- SI disponen de instrucciones específicas de E/S, se require disponer, asimismo, de un espacio de direccionamiento separado para E/S y Memoria (típicamente en procesadores de la familia Intel 80x86, por ejemplo).

Repertorio de instrucciones

Instrucciones de control (de flujo de programa)

- Son instrucciones que pueden modificar el valor contenido en el registro PC.
- Las instrucciones más communes de este tipo son:
  - Salto Incondicional: JMP destino
  - Salto Condicional: JZ destino
  - Salto con retorno (llamada a subrutina): CALL subrut
  - Retorno de subrutina: RET
  - Interrupción por software

Repertorio de instrucciones

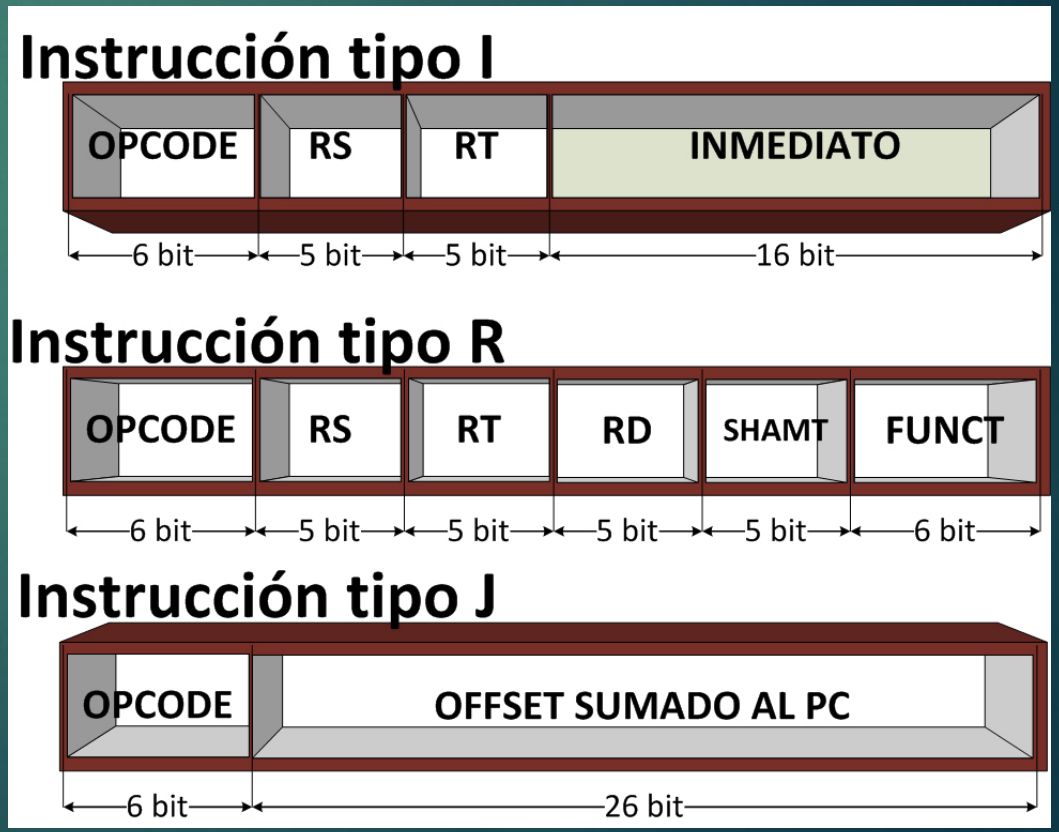
Instrucciones de control de Sistema (control de la CPU)

- Modifican algún estado de operación de la CPU.
- Las instrucciones más comunes de este tipo son:
  - Habilitación o inhibición de interrupciones
  - Modo de operación: supervisor/usuario
  - Halt: suspensión de actividades

Formato de instrucción fijo

- Los formatos de las instrucciones pueden ser fijos o variables.
- En los formatos de instrucción de longitud fija, el tamaño de la instrucción es el mismo para todas las instrucciones.

Por ejemplo, el formato de instrucción de un procesador tipo RISC es el siguiente (solo 3 tipos de instrucciones todas de 32 bits).

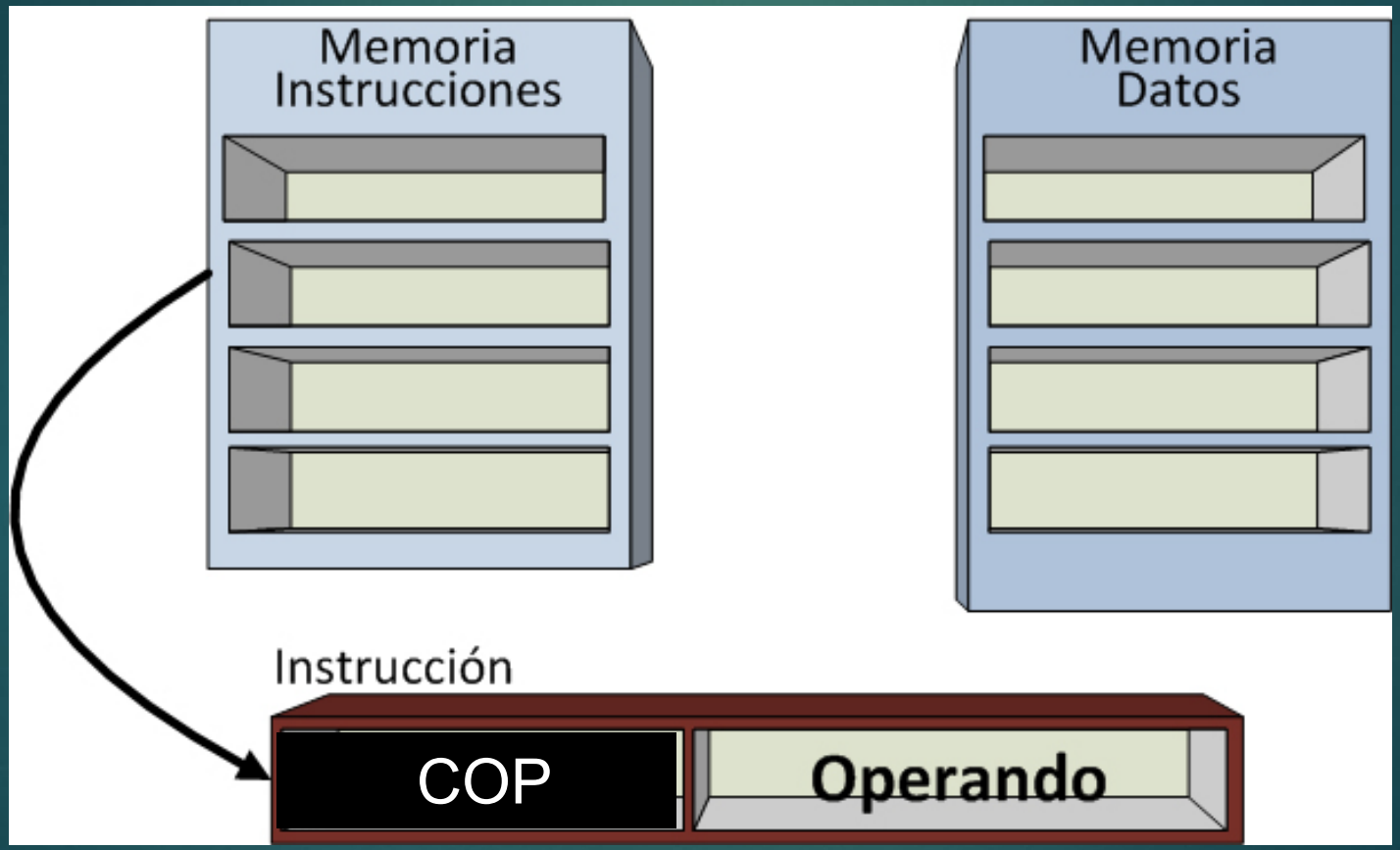


Modos de direccionamiento

- Los modos de direccionamiento son las diferentes formas de especificar el lugar donde reside un operando, donde se va a guardar el resultado, o donde buscar la próxima instrucción a ejecutar.
- En general, los procesadores disponen de una variedad más o menos amplia de modos de direccionamiento. Algunos de los más típicos son:
  - Inmediato
  - Directo: a memoria (“absoluto”), a Registro
  - Indirecto: a memoria (en desuso), con registro
  - Base: base, base Indexado
  - Relativo al PC
  - Pila (o relativo al SP)

Modos de direccionamiento

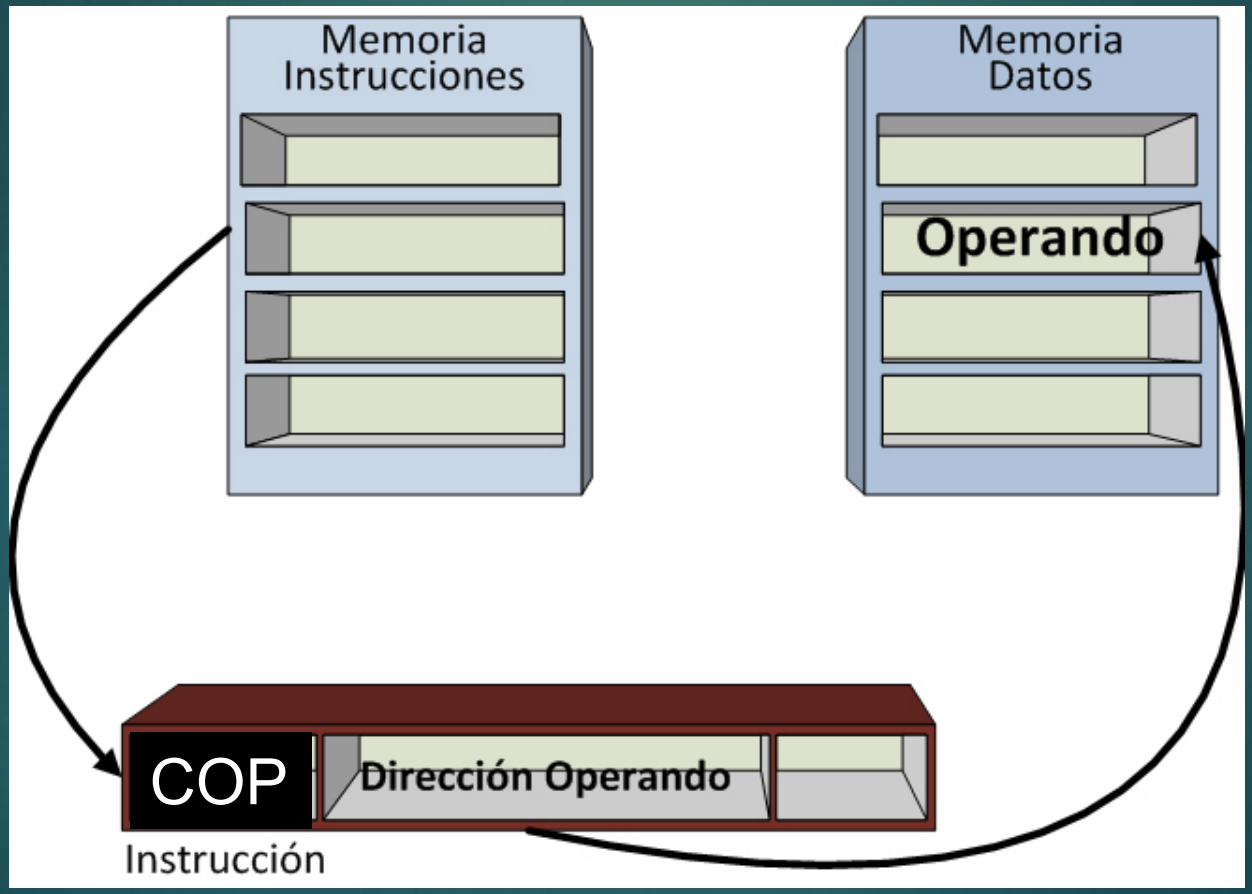
- Esquema del modo de direccionamiento Inmediato



COP: Código de operación

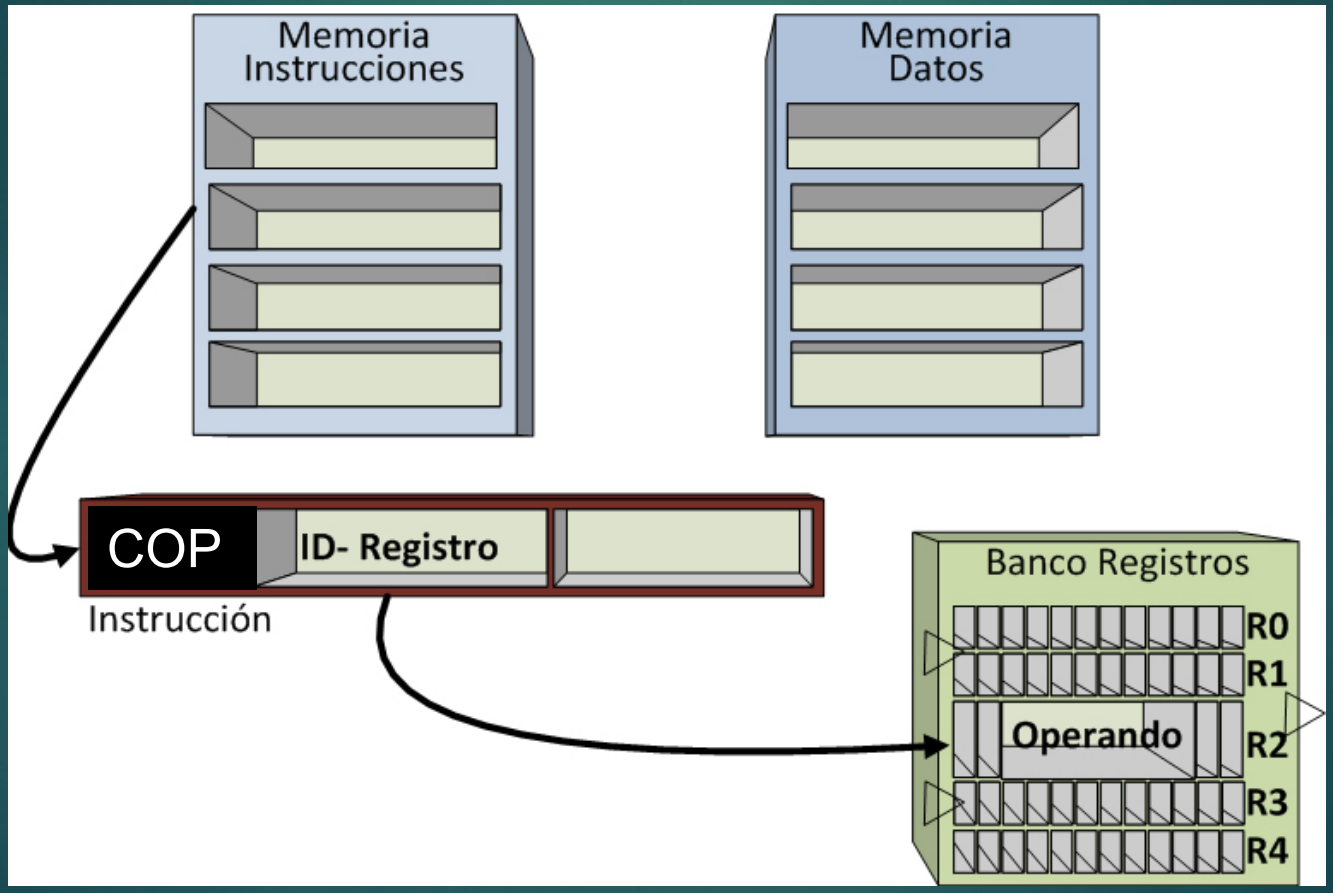
Modos de direccionamiento

- Esquema del modo de direccionamiento Directo a memoria o absoluto



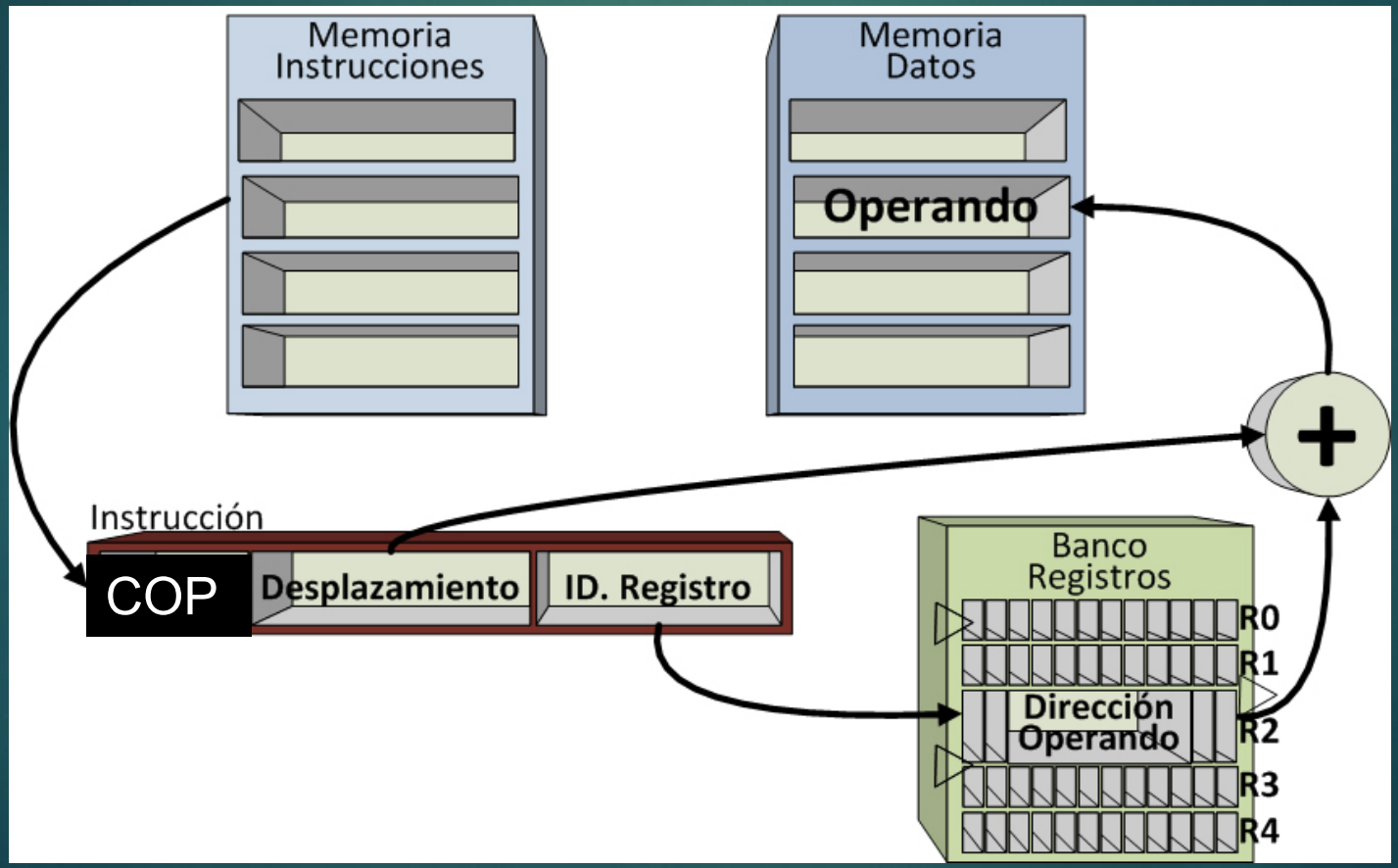
Modos de direccionamiento

- Esquema del modo de direccionamiento Directo a Registro



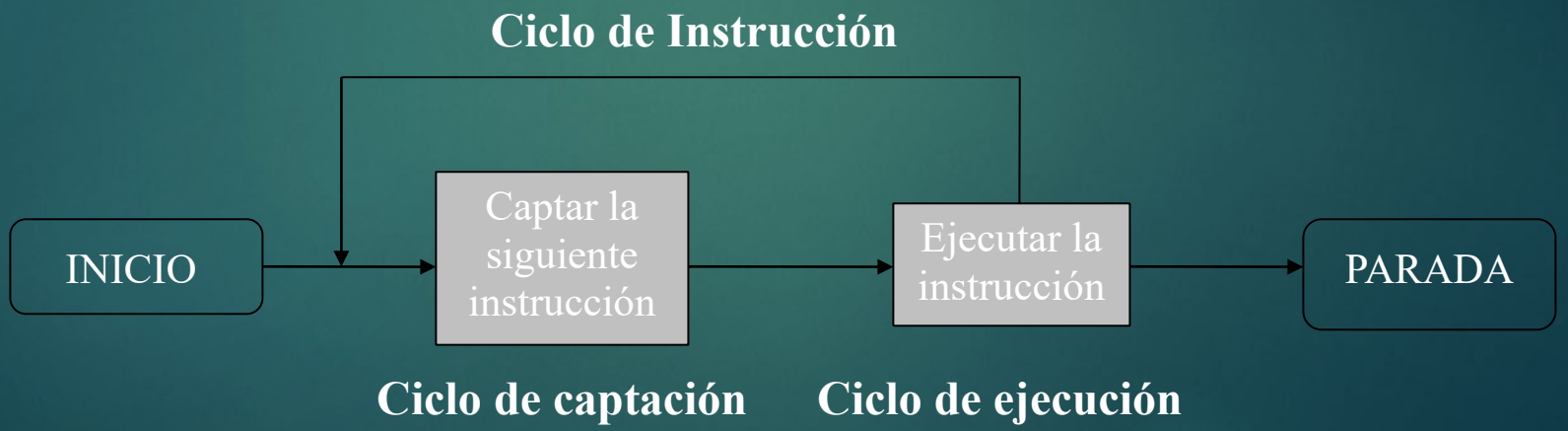
Modos de direccionamiento

- Esquema del modo de direccionamiento Indirecto con desplazamiento (Base con desplazamiento)



Ciclo de instrucción básico

- El ciclo de instrucción es la secuencia de acciones que realiza el procesador para resolver las instrucciones.
- En su forma más sencilla, se puede descomponer en 2 acciones básicas:
  - Captación de la instrucción
  - Ejecución de la operación



Ciclo de instrucción básico

- Ciclo de captación de la instrucción

El ciclo de captación, o lectura de la instrucción, require las siguientes acciones:

- La dirección de la instrucción que se debe captar se carga en el registro Contador de Programa (PC).
- La UC envía al bus de direcciones el valor almacenado en el PC.
- La UC lee la instrucción desde la Memoria, y la almacena en un registro temporal denominado Registro de Instrucción (IR).
- El PC se incrementa en 1 (se prepara para apuntar a la próxima instrucción).
- La UC decodifica (“interpreta”) la instrucción captada.

Ciclo de instrucción básico

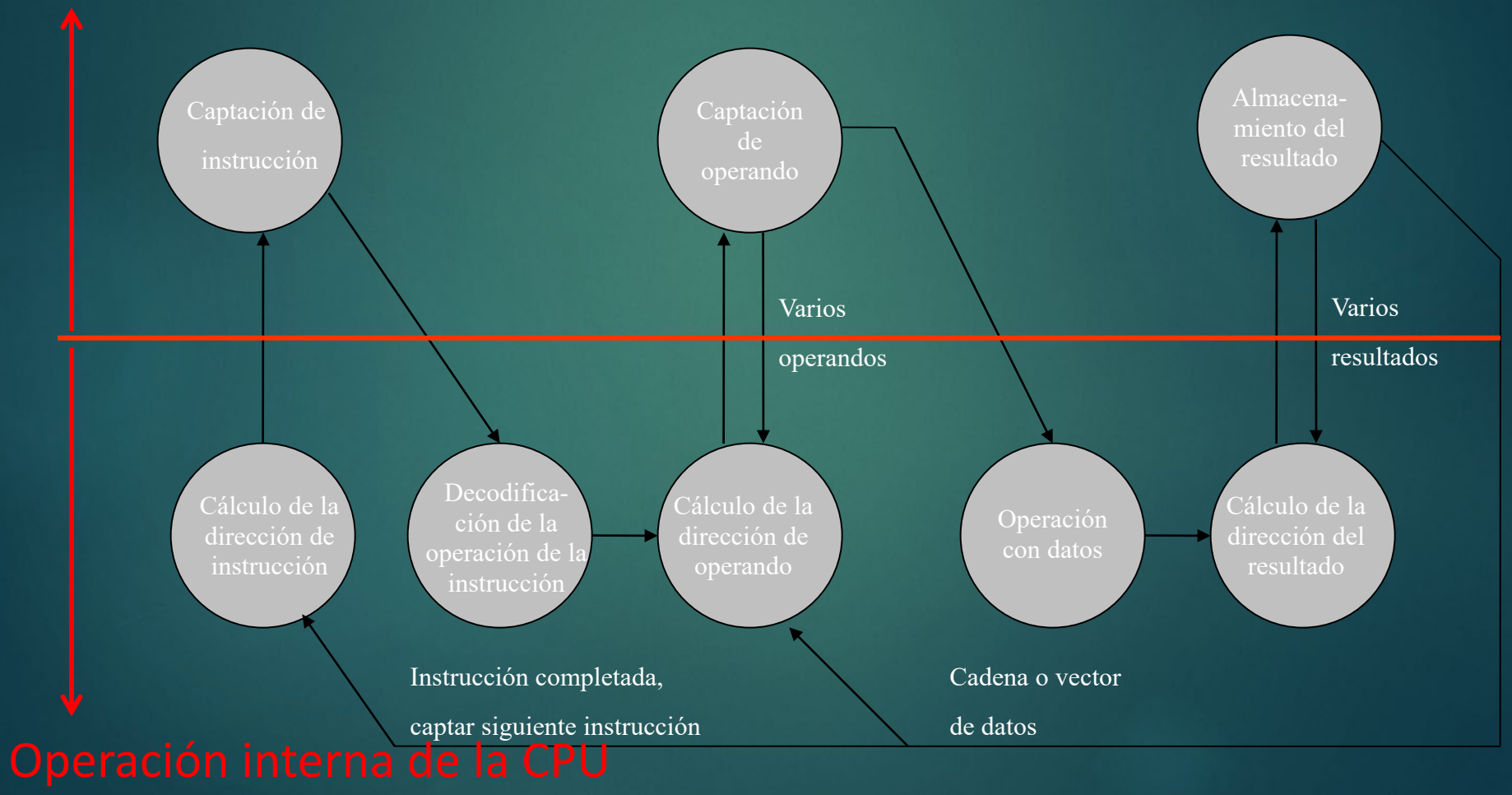
- Ciclo de ejecución de la instrucción

El ciclo de ejecución puede ser muy variado dependiento del tipo de instrucción a ejecutar. Algunas de las posibles acciones en este ciclo son:

- Acciones procesador – memoria: por ejemplo transferencia de datos CPU - Memoria.
- Acciones procesador - E/S: por ejemplo, transferencias de datos entre la CPU y un dispositivo de E/S.
- Acciones de procesamiento de datos: operaciones aritméticas o lógicas con los datos.
- Acciones de control: por ejemplo alteración de la secuencia de ejecución (salto, salto a subrutina).
- Acciones combinadas de las anteriores

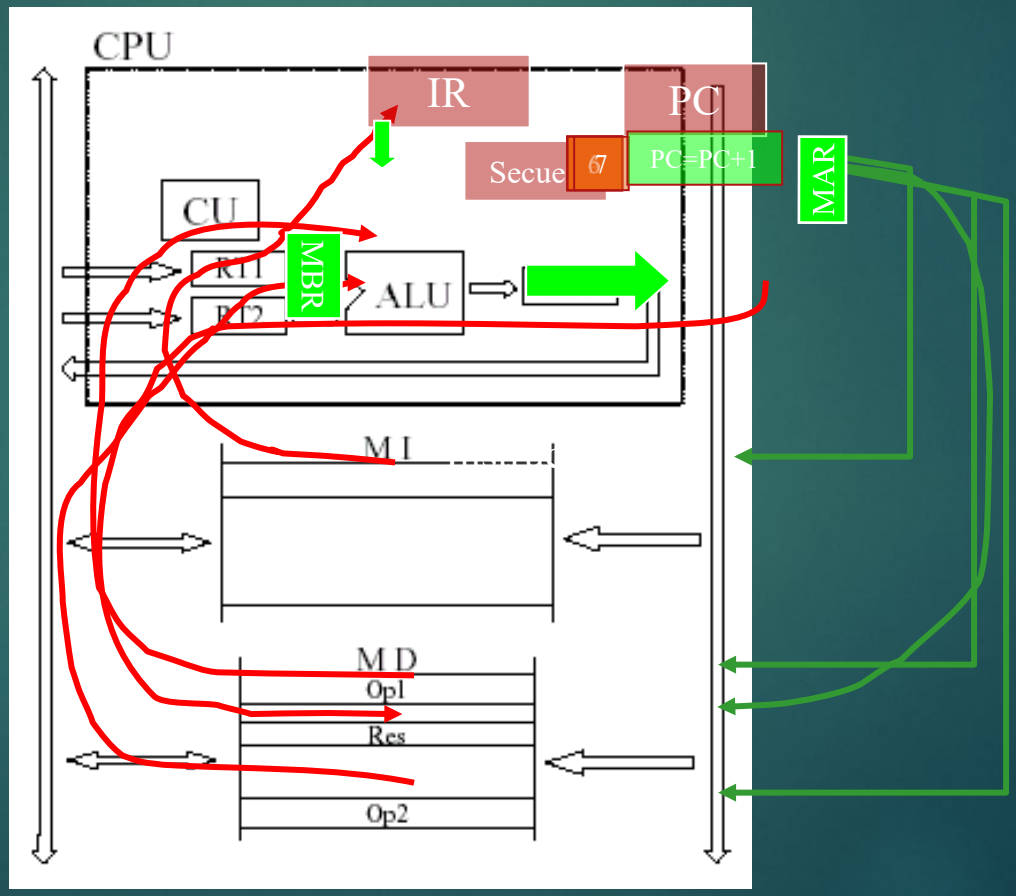
Ciclo de instrucción detallado

Modelo de ciclo de instrucción detallado



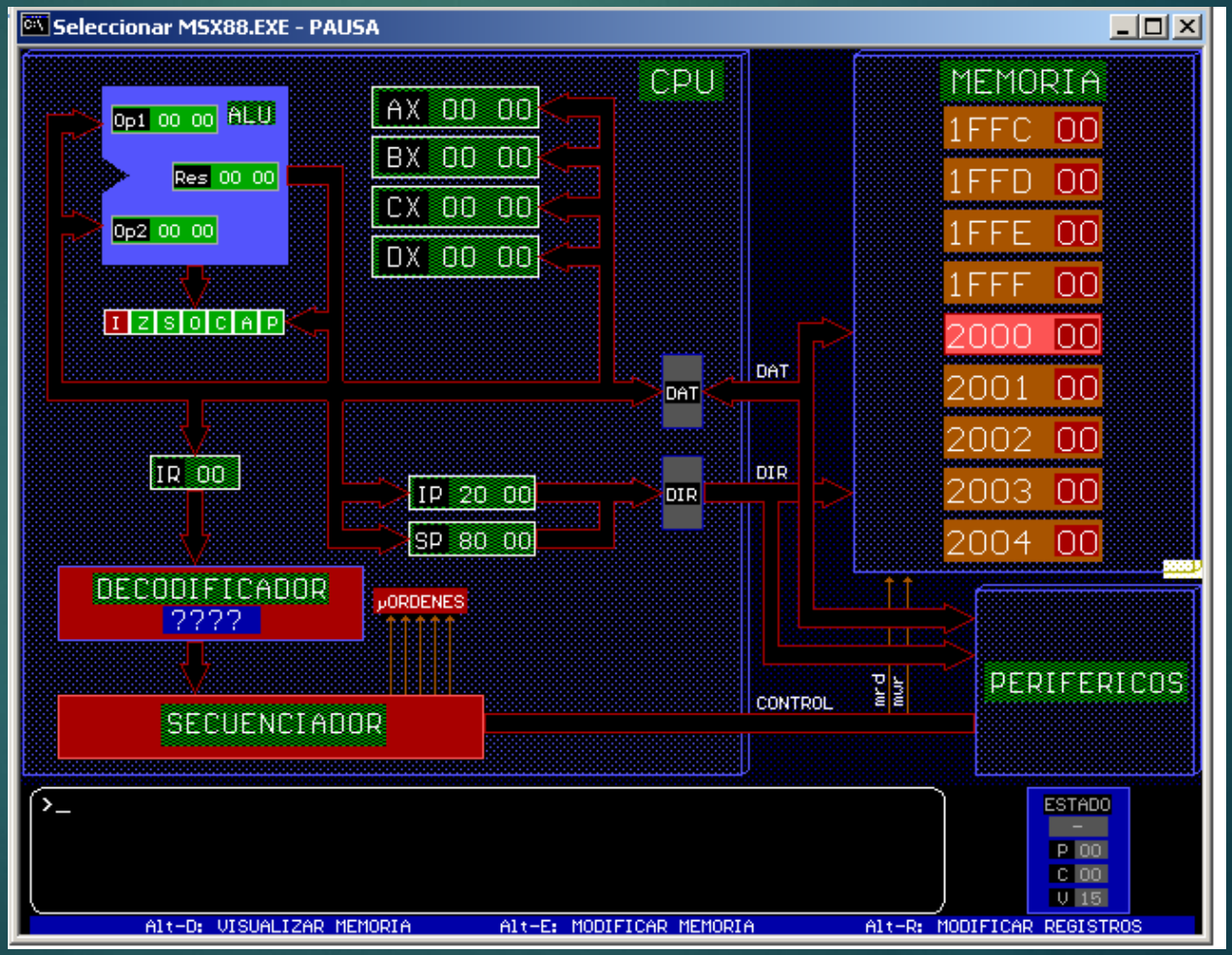
Análisis de la ejecución de una instrucción a nivel de hardware

La figura siguiente representa la secuencia de acciones de un procesador para resolver una instrucción de 2 operandos en memoria.



Simulador MSX88

Pantalla del simulador MSX88



Repertorio de instrucciones del MSX88

Instrucciones de movimiento de datos del MSX88

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

1. *dest/fuente* 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 *fuente* solo pueden ser registros de 16 bits.
3. *dest/fuente* son: *AL/mem, AX/mem, AL/DX, AX/DX.*
4. *dest/fuente* 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.

Repertorio de instrucciones del MSX88

Instrucciones aritméticas y lógicas del MSX88

1	ADD *dest,fuente*	Suma *fuente* y *dest*	(*dest*)← (*dest*)+(*fuente*)
1	ADC *dest,fuente*	Suma *fuente*, *dest* y *flag C*	(*dest*)← (*dest*)+(*fuente*)+C
1	SUB *dest,fuente*	Resta *fuente* a *dest*	(*dest*)← (*dest*)-(*fuente*)
1	SBB *dest,fuente*	Resta *fuente* y *flag C* a *dest*	(*dest*)← (*dest*)-(*fuente*)-C
1	CMP *dest,fuente*	Compara *fuente* con *dest*	(*dest*)-(*fuente*)
5	NEG *dest*	Negativo de *dest*	(*dest*)← CA2(*dest*)
5	INC *dest*	Incrementa *dest*	(*dest*)← (*dest*)+1
5	DEC *dest*	Decrementa *dest*	(*dest*)← (*dest*)-1
1	AND *dest,fuente*	Operación *fuente* AND *dest* bit a bit	(*dest*)← (*dest*) AND (*fuente*)
1	OR *dest,fuente*	Operación *fuente* OR *dest* bit a bit	(*dest*)← (*dest*) OR (*fuente*)
1	XOR *dest,fuente*	Operación *fuente* XOR *dest* bit a bit	(*dest*)← (*dest*) XOR (*fuente*)
1	NOT *dest*	Complemento a 1de *dest*	(*dest*)← CA1(*dest*)
5			

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 memo-ria (dir.indirecto).

Repertorio de instrucciones del MSX88

Instrucciones de control de programa del MSX88

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

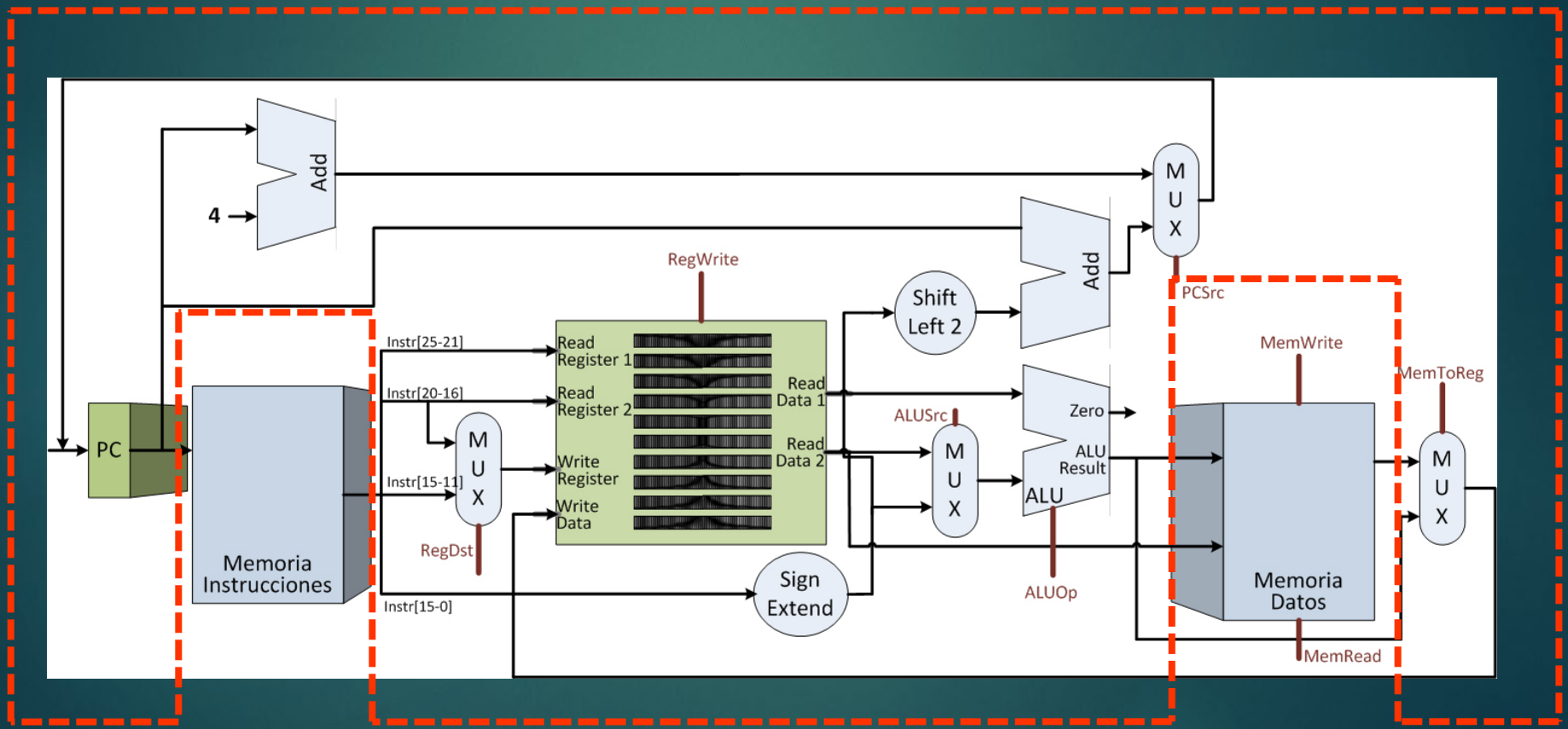
6. *mem* es la dirección de memoria llamada *etiqueta*.

Modelo de Organización del nanoMIPS

- Para el análisis de algunos temas de las próximas clases, se va a emplear una máquina de características muy distintas de las usadas hasta ahora.
- La máquina se identifica con el nombre de nanoMIPS.
- Es un modelo simplificado de una familia de máquinas tipo RISC de uso comercial ,conocidas como MIPS.
- En la siguiente figura se muestra, en forma simplificada, la sección del nanoMIPS encargada del flujo de los datos e instrucciones (es decir, la trayectoria de los datos).

Modelo de Organización del nanoMIPS

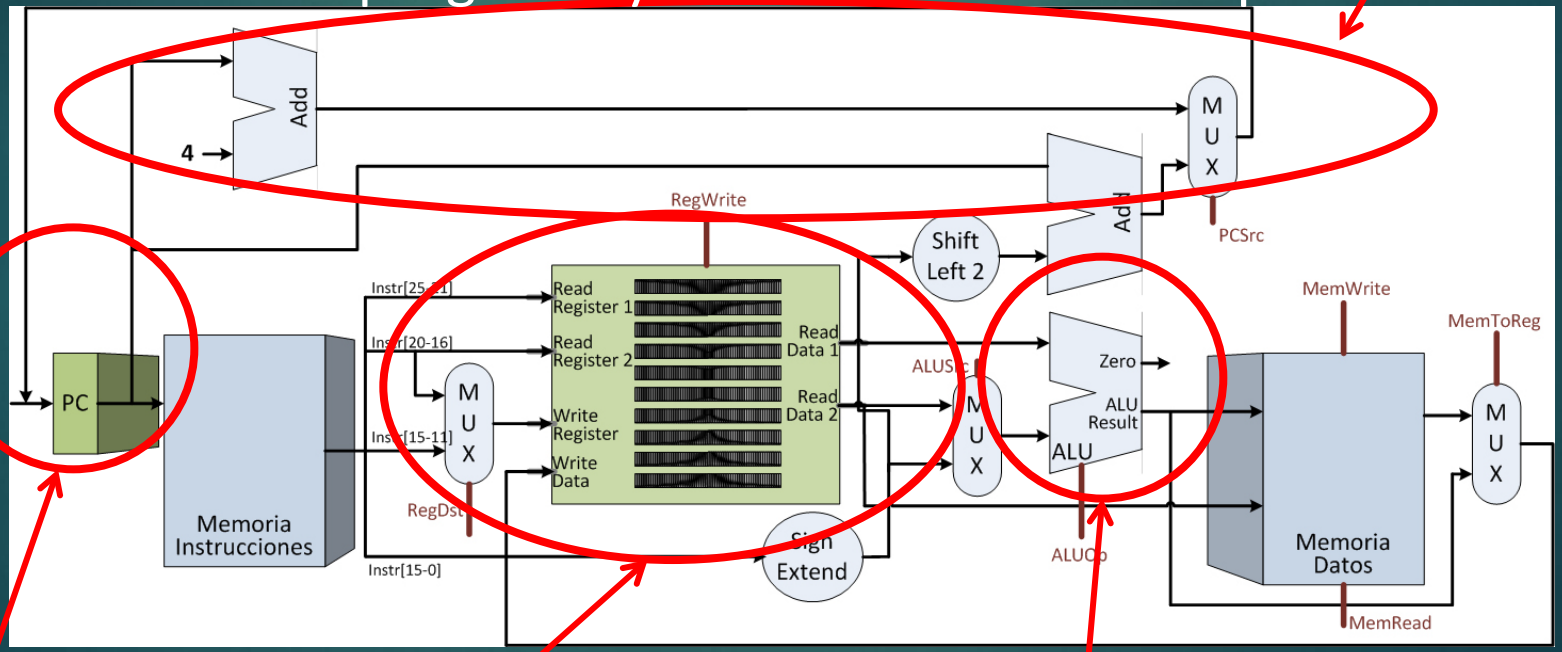
- Principales bloques funcionales encargados del flujo de los datos e instrucciones del nanoMIPS.



Modelo de Organización del nanoMIPS

- Las principales unidades del nanoMIPS son:
  - Banco de registros
  - ALU de datos
  - Contador de programa y cálculo de la dir. de la próxima instruct.

CALCULO PROXIMO VALOR CONTADOR DE PROGRAMA



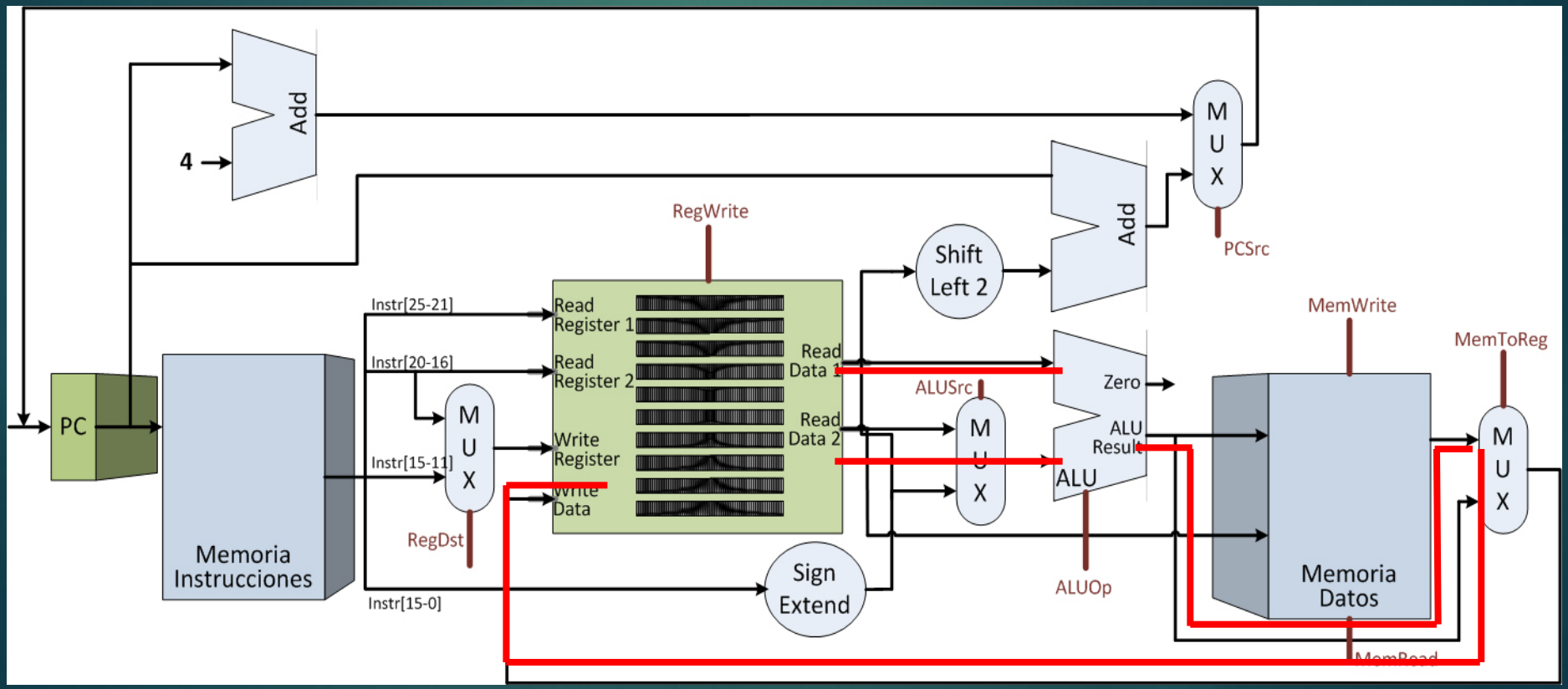
CONTADOR DE PROGRAMA

REGISTROS

ALU PARA DATOS

Modelo de Organización del nanoMIPS

- En la figura siguiente se muestra un ejemplo de la trayectora de los datos para un operación entre 2 registros y el resultado guardado en otro registro.



Modelo de Organización del nanoMIPS

- En la figura anterior, se ha marcado con la línea roja, la trayectora de los datos, desde que salen de los 2 registros hasta que entran al tercero (caso típico de una operación aritmética de 2 operandos y 1 resultado).
- El camino de los datos desde los 2 registros, pasando por la ALU, y almacenándose en un tercer registro, ha sido diseñada para optimizar el tiempo de ejecución.
- Esa operación puede resolverse en un solo ciclo de reloj.
- Notar que la máquina dispone de 2 unidades de memoria separadas: instrucciones y datos.
- Esta arquitectura de memorias separadas de instrucciones y datos se conoce como “Arquitectura tipo Harvard”.

PILA

- La Pila es una estructura ordenada de datos tipo LIFO (Last In- First Out).
- El acceso a la estructura se hace únicamente desde el tope (o cabeza) de la estructura.
- El acceso se hace mediante un registro específico denominado Puntero de Pila (SP, Stack Pointer).
- Sobre la Pila se pueden realizar 2 operaciones básicas, inversas entre sí:
  - Almacenar o “apilar”
    - Mnemónico en Assembly PUSH
  - Extraer o “desapilar”
    - mnemónico en Assembly POP

PILA

- Las operaciones de apilado y desapilado requieren 2 acciones que se ejecutan secuencialmente:
  - 1- Movimiento de datos: que pueden ser del tipo
    - Registro a memoria
    - Memoria a registro
    - Memoria a memoria
  - 2- Operación del puntero: antes y/o después del movimiento de datos (Stack Pointer – SP)
- Al haber movimiento de datos y de puntero durante el apilado o desapilado, hay que tener en cuenta:
  - Dónde apunta el puntero de pila SP
  - Cómo se mueve el SP, porque determina como “crece” la pila (que área de memoria va ocupando)

PILA

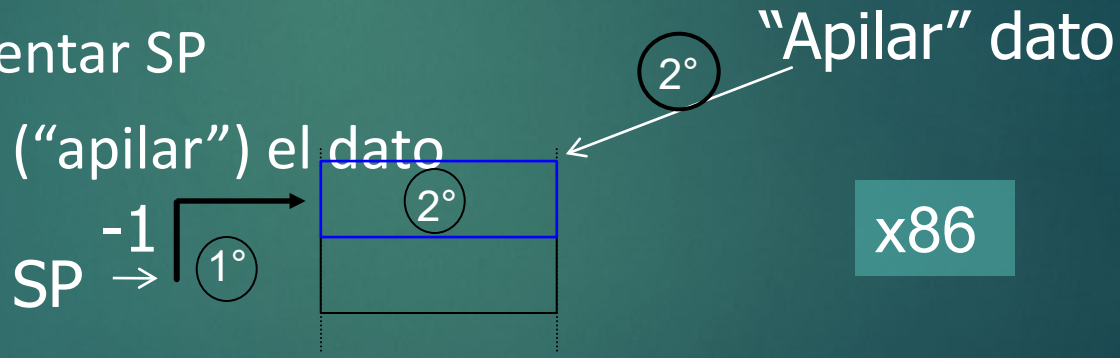
Manejo de la Pila en procesadores de la familia i80x86

En los procesadores de la familia Intel, la secuencia del movimiento del dato a la pila y la actualización del SP es la siguiente.

- **PUSH:**

1° -> decrementar SP

2° -> guardar (“apilar”) el dato



Es decir, primero se decrementa el SP, y luego se transfiere el dato al “tope” de la pila (lugar a donde “apunta” el SP).

Así, los datos se van almacenando en posiciones decrecientes de memoria. La pila “crece” hacia posiciones de memoria inferiores.

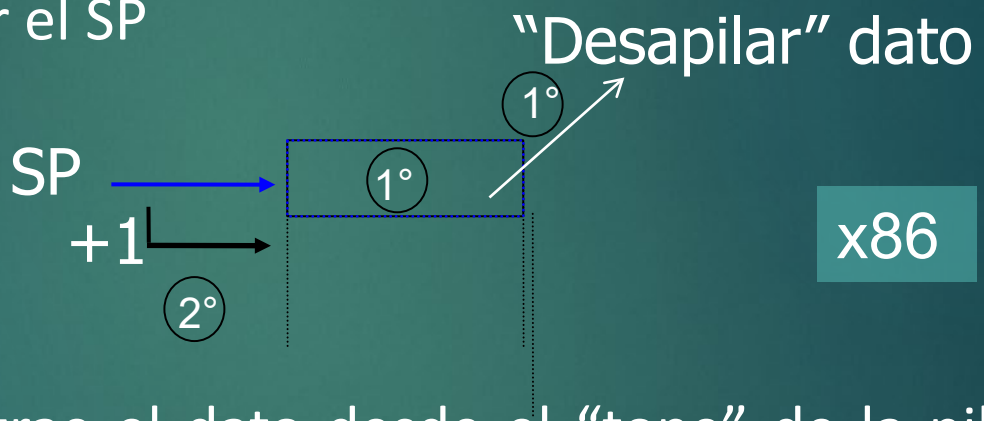
PILA

Manejo de la Pila en procesadores de la familia i80x86

- **POP:**

  1° -> extraer (“desapilar”) el dato

  2° -> incrementar el SP



Es decir, primero se extrae el dato desde el “tope” de la pila (lugar a donde “apunta” el SP) , y luego se incrementa el SP. La operación de POP es exactamente inversa de la de PUSH.

El SP queda apuntando a la dirección donde está el último dato guardado (tope de la pila), el dato extraído permanece en la memoria pero no forma parte de la estructura de datos de la pila (copia no válida)

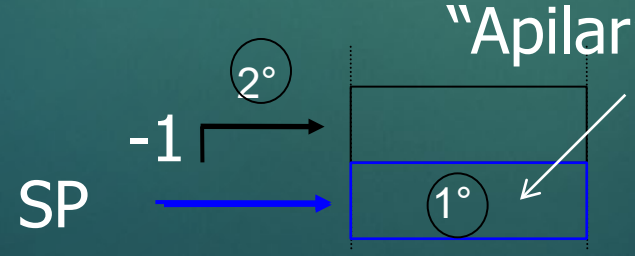
PILA

Manejo de la Pila en procesadores de la familia M68xxx

- Algunos procesadores administran la pila de una manera distinta.
- Por ejemplo, en los procesadores de la familia Motorola, la secuencia del movimiento del dato a la pila y la actualización del SP es la siguiente.
- PUSH:

  1° -> guardar (“apilar”) el dato

  2° -> decrementar SP



68xxx

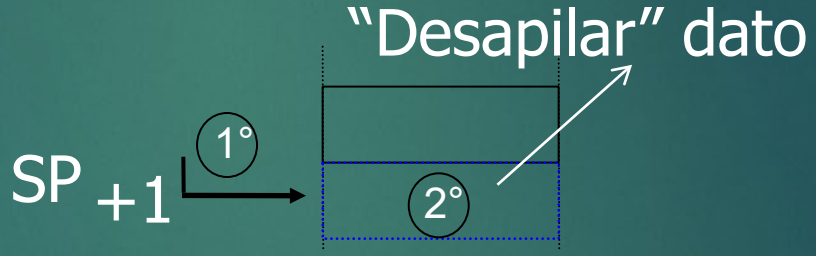
PILA

Manejo de la Pila en procesadores de la familia M68xxx

- POP:

  1° -> incrementar el SP

  2° -> extraer (“desapilar”) el dato



68xxx

Igual que antes, los datos se van almacenando en posiciones decrecientes de memoria. Es decir, la pila “crece” hacia posiciones de memoria inferiores.

Además el PUSH y el POP son exactamente inversas entre sí.

Pero ahora el SP apunta siempre a la primer posición libre, por encima del tope de la pila.

Instrucciones de uso de la PILA

Instrucciones para la PILA en Lenguaje Assembly

- Para operar con la Pila, los procesadores disponen de instrucciones de carga (PUSH) y descarga de datos (POP).
- En el ejemplo siguiente se muestra un programa donde se usa la Pila para intercambiar datos entre los registros AX y CX usando las instrucciones de PUSH y POP.
- Notar que después de ejecutado el programa, la Pila queda exactamente como estaba previamente. Es decir, el SP va a quedar con el mismo valor que tenía cuando comenzó el programa.

Instrucciones de uso de la PILA

Ejemplo de uso de instrucciones para la PILA en Lenguaje Assembly

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

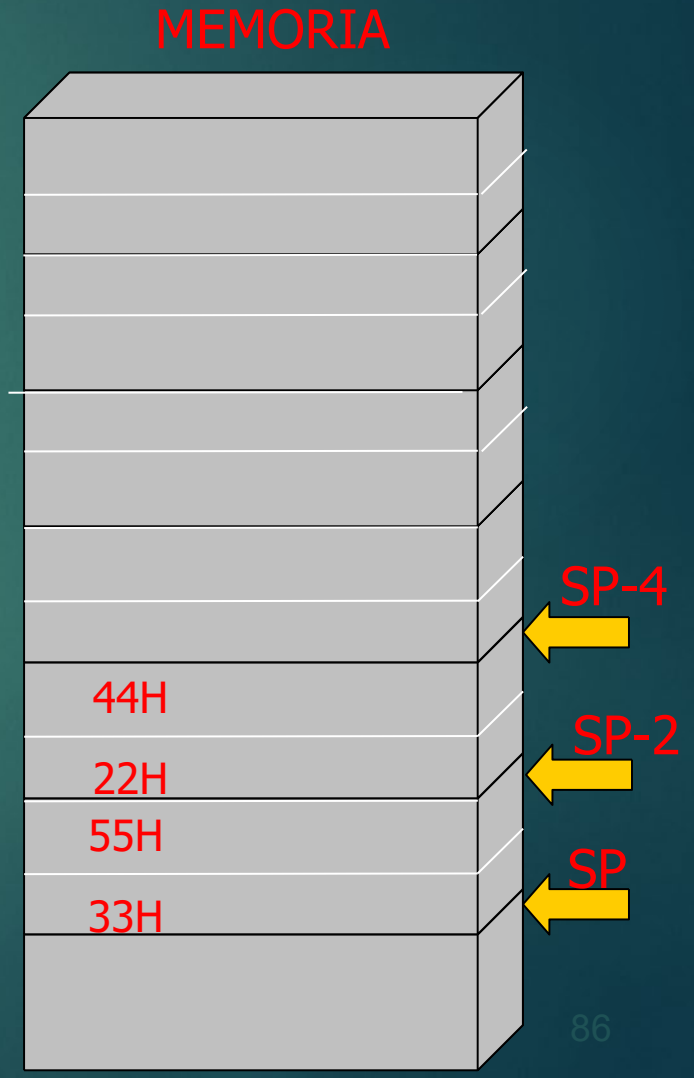
Instrucciones de uso de la PILA

- En el ejemplo anterior se ejecutan en forma consecutiva 2 instrucciones PUSH, y a continuación 2 instrucciones POP.
- En el PUSH AX, el SP se decrementa en 2 y la CPU almacena AX (3355H) en el tope de la pila (el dato de 16 bits se almacena en 2 palabras consecutivas de la pila).
- Idem con PUSH CX (2244H).
- En el POP AX, se carga AX desde el tope de la PILA, que contenía 2244H y el SP se incrementa en 2.
- Idem con POP CX (3355H).
- Al final del programa, el SP contiene el valor que tenía antes de la ejecución del programa.

Instrucciones de uso de la PILA

Movimiento de datos y SP durante la ejecución del programa

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



Subrutinas

- DEFINICIÓN

Una subrutina es una sección de código, que recibe el control en un punto de entrada, y lo devuelve en un punto de salida.

- OBJETIVO

El objetivo de la subrutina es realizar una tarea definida, para lo cual se le transfiere el control. Una vez finalizada la tarea lo devuelve al programa que la invocó en el punto donde la invocó.

Subrutinas

- VENTAJAS

El uso de subrutina tiene muchas ventajas. Entre otras:

- Economía de programa: el código puede ser usado varias veces
- Modularidad: se puede subdividir el programa en unidades pequeñas, más fácilmente verificables.

- OBSERVACIONES

Para realizar su trabajo, se requieren pasar parámetros entre el programa que invoca y la subrutina invocada.

Subrutinas

- Invocación

Una subrutina se invoca mediante una instrucción específica, disponible en el repertorio de instrucciones del procesador. Por ejemplo, un mnemónico típico usado para invocar una subrutina es la instrucción: CALL

- Terminación

La subrutina termina mediante una instrucción específica de retorno. Por ejemplo: RET

- Ejemplo

A continuación se muestra una típica sección de código Assembly con un programa (principal) invocando una subrutina (también llamado Procedimiento).

Subrutinas

En **Programa principal**:

```
...
CALL   Nombre (del procedimiento)
Dir. Ret        Prox. Instrucción
...
```

En **Procedimiento**:

```
Nombre  Proc
        ...
        ...
        RET
Nombre  Endp
```

Cuerpo del procedimiento

Subrutinas

- En el ejemplo anterior, el programa principal invoca la subrutina en la instrucción CALL Nombre (del procedimiento).
- Al invocarla, le transfiere el control. Es decir, el PC se carga con la dirección de comienzo de la subrutina, y la CPU comienza a ejecutar las instrucciones de la subrutina.
- Cuando la subrutina complete su tarea, la última instrucción que ejecuta es la de RET.
- En ese momento el PC se carga con la dirección de la instrucción siguiente al CALL en el programa principal. De esa manera el programa principal continúa con su tarea en el punto siguiente a la invocación de la subrutina.

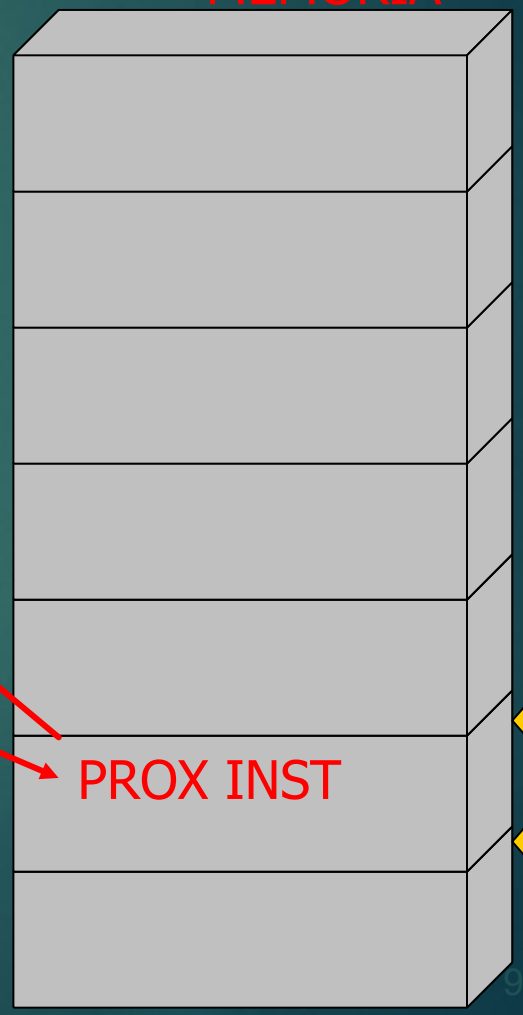
Subrutinas

- Para poder recuperar la dirección de la instrucción siguiente al CALL en el programa principal, esta dirección debe haber sido guardada previamente.
- El lugar donde se guarda esta dirección es la Pila. Cada vez que se ejecuta una instrucción de CALL, se guarda en la Pila el valor actual del PC.
- Cuando se ejecuta una instrucción RET, el PC se carga con la dirección almacenada en el tope de la Pila.
- Mediante este mecanismo, al terminar la subrutina, se restablece en el PC la dirección de la instrucción siguiente al CALL.
- Este mecanismo se muestra en la figura siguiente, que corresponde al programa del ejemplo previo.

Subrutinas

Movimiento de la Pila durante la ejecución del programa

```
Ejemplo:        ORG    2000H
                MOV    AX, 0001H
                CALL   SUB1
PROX INST       MOV    BX,0002H
                …
                …
                ORG    3000H
SUB1            PROC
                …
                RET
SUB1            ENDP
```



Subrutinas

A modo de ejemplo, se necesita analizar el comportamiento de la Pila y los registro para el siguiente ejercicio.

```
            ORG 1000H
SUB1:       NEG     AX
            RET

            ORG 2000H
            MOV     BX, 0
            MOV     AX, dato
            PUSH    AX
            CALL    SUB1
            POP     BX
            HLT
            ORG 3000H
dato:       DB      55H,00H
            END
```

Subrutinas

Análisis del ejemplo anterior

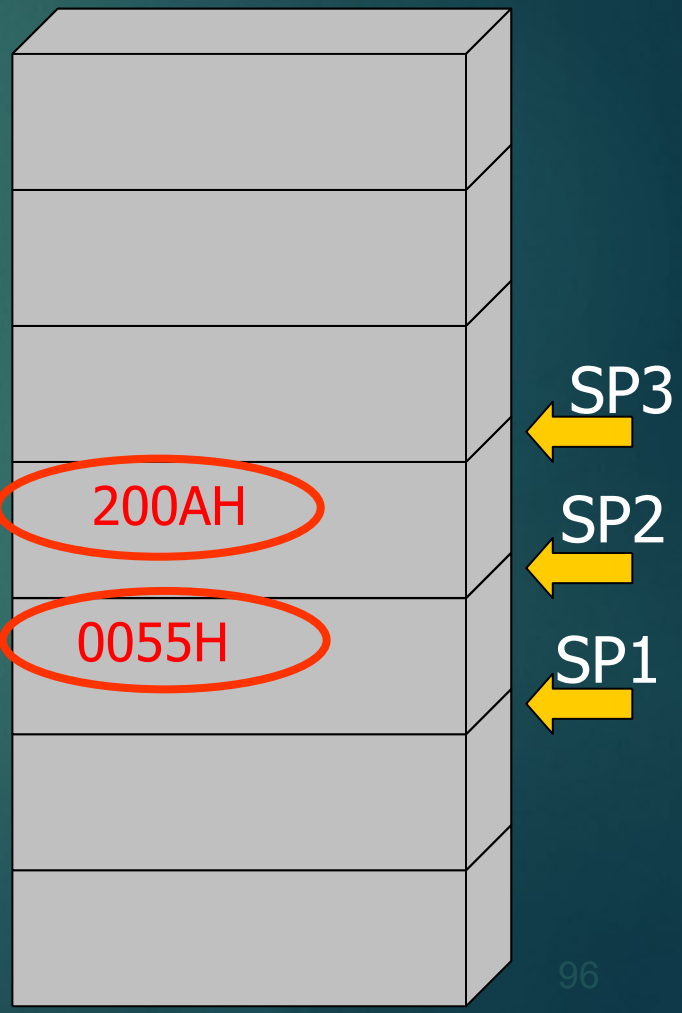
- El punto de entrada del programa principal es 2000H.
- La instrucción PUSH AX apila el registro AX.
- A continuación se invoca la subrutina SUB1. Antes de transferir el control a SUB1 se apila la dirección de la instrucción siguiente (es decir, la de POP BX).
- Se transfiere el control a la subrutina SUB1.
- La subrutina SUB1 termina en la instrucción RET, que desapila el valor almacenado en el tope de la pila, con el que se carga el PC. El nuevo valor del PC es el que corresponde a la dirección de memoria donde está la instrucción POP BX.
- La instrucción POP BX carga en BX el dato desde el tope de la pila.

Subrutinas

Movimiento de la Pila del ejemplo anterior

El programa ensamblado queda:

```
1000H NEG AX
1001H RET
      …
      …
2000H  MOV BX, 0
2003H  MOV AX, 0055H
2006H  PUSH AX
2007H  CALL 1000H
200AH  POP BX
200BH  HLT
```



Detalles de uso de subrutinas

- Para que una subrutina realice una tarea, requiere de datos.
- Los datos (argumentos o parámetros) se los proporciona el programa que invoca a la subrutina.
- A su vez, la subrutina produce 1 o más resultados. Los resultados deben ser devueltos al programa que invocó la subrutina.
- Existen 3 métodos para pasar parámetros a subrutinas.

Detalles de uso de subrutinas

- 1.- Vía registros
  - Los parámetros se pasan a través de los registros de la CPU.
  - Es un método sencillo, pero limitado por el número de registros disponibles.
  - Dado que se van a modificar los contenidos de los registros, es importante documentar los registros a usar.
- 2.- Vía memoria
  - Los parámetros se transfieren a través de un área definida de memoria (RAM).
  - Difícil de estandarizar, debido a las dificultades en asignar un área de memoria.

Detalles de uso de subrutinas

- 3.- Vía pila (stack)
  - Los datos se pasan a través de la Pila.
  - Es el método más ampliamente usado porque presenta varias ventajas respecto de los otros métodos.
  - La principal ventaja es que es independiente de la memoria y registros.
  - Los registros no tienen que ser modificados en las subrutinas.
  - Pero hay que manejar correctamente, porque la Pila va a ser usada por el usuario (en la invocación y en la subrutina) y por el sistema.
  - Recordar que en el x86, el SP apunta al último lugar usado (último dato entrado)

Detalles de uso de subrutinas

Programa principal con llamada a Subrutina

- Para invocar una subrutina con pasaje de parámetros vía Pila, en el programa que invoca se hace lo siguiente:

```
...
Push Parámetro 1
Push Parámetro 2
CALL   SUB1 (del procedimiento)
DirRet  Prox. Instrucción
```

Detalles de uso de subrutinas

Programa principal con llamada a Subrutina

- Previo a la invocación de la subrutina (es decir previo al CALL) se deben apilar los parámetros requeridos por la subrutina (en el ejemplo, se apilan 2 parámetros).
- Tener en cuenta que las 2 instrucciones de Push modifican el SP. El programa principal deberá restablecer a partir de Prox. Instrucción (luego del retorno de la subrutina) el valor correcto del SP (por ejemplo con 2 instrucciones POP).
- El movimiento de los datos y la Pila, se muestran en la filmina siguiente.

Detalles de uso de subrutinas

Movimiento de la Pila durante la invocación a la subrutina

Procedimiento

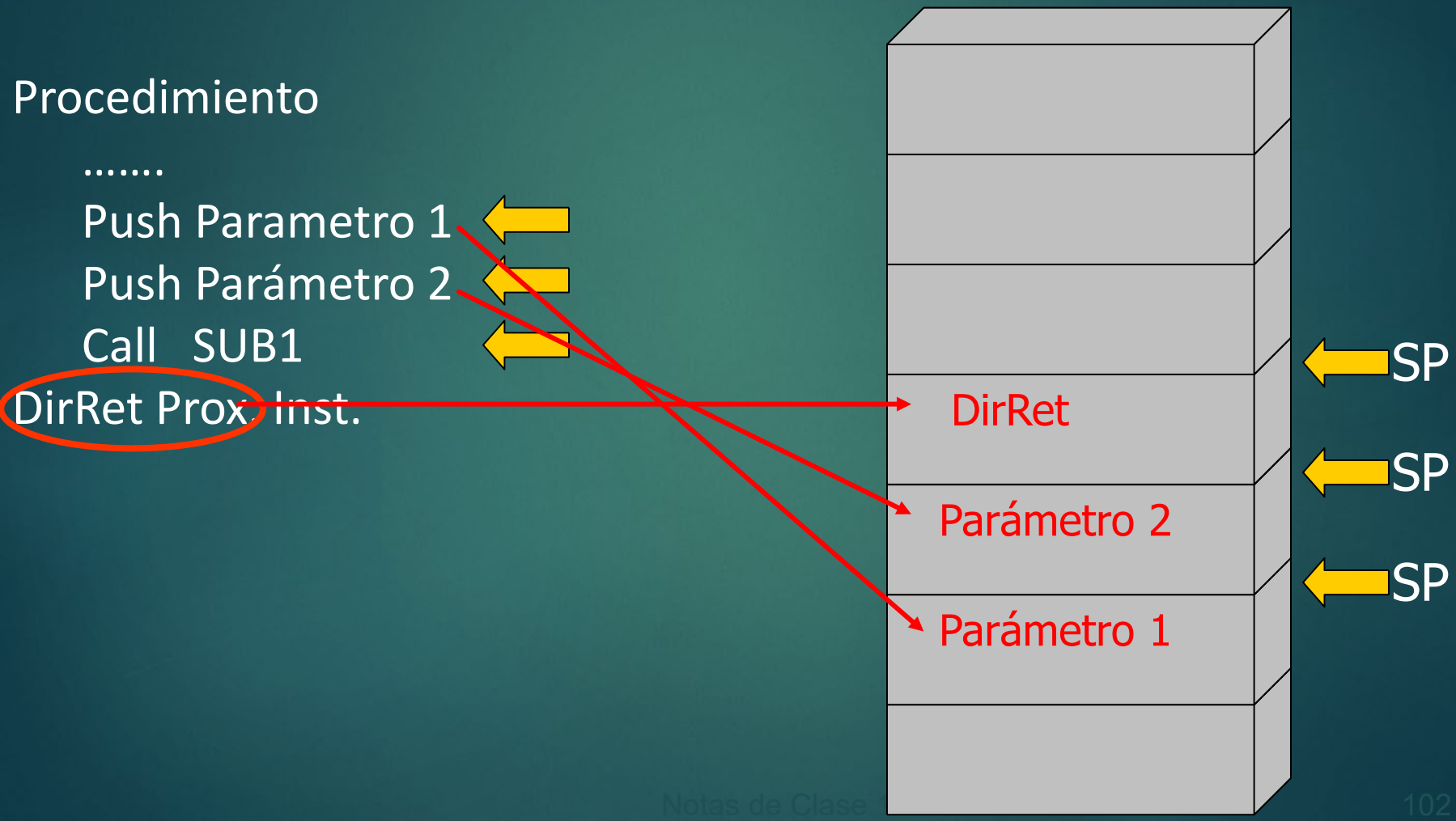
.....

Push Parametro 1

Push Parámetro 2

Call SUB1

DirRet Prox. Inst.



Detalles de uso de subrutinas

Estructura de la subrutina

- Para la descripción que sigue se va a usar como referencia el procesador (80x86).
- Hay que tener en cuenta que el 80x86 no dispone de un modo de direccionamiento relativo al SP (direccionamiento Base). Algunas operaciones deben hacerse debido a esta "limitación" del procesador.
- Para el acceso a la estructura de datos de la Pila se requiere un registro base (dado que no se puede usar el SP), típicamente el BP.
- La subrutina no debería modificar los contenidos de los registros al momento de su invocación. Si la subrutina usa un registro lo debería salvaguardar al comienzo, y restaurarlo al final.

Detalles de uso de subrutinas

Estructura de la subrutina

- La Subrutina requiere de una serie de acciones de manera que quede correctamente estructurada.
- La parte de la subrutina que ejecuta la acción específica se llama cuerpo de la subrutina.
- Previo al cuerpo de la subrutina, se requiere una serie de acciones de “preparación”.
- Posterior al cuerpo de la subrutina, se deben ejecutar otras acciones más para restablecer y adecuar la subrutina, y devolver el control al programa que la invocó en las condiciones correctas.

Detalles de uso de subrutinas

Estructura de la subrutina

1. Salvar el estado de BP (viejo BP)
2. Apuntar con BP al tope de la pila (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 ← cuerpo
7. Restaurar los registros modificados (opcional)
8. Restaurar el valor original de SP y BP
9. Retornar parámetro (opcional)
10. Regresar (correctamente del procedimiento)

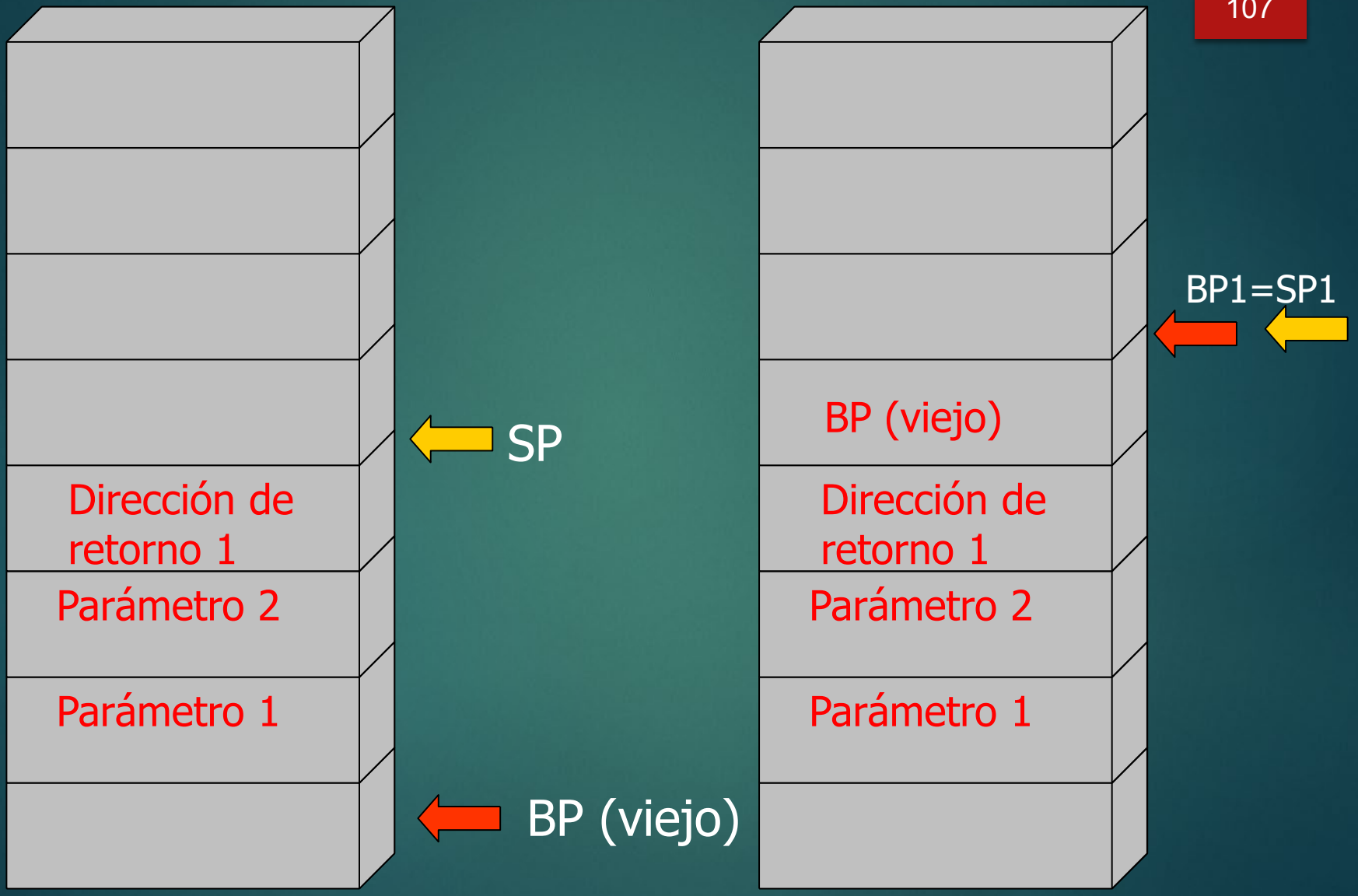
Detalles de uso de subrutinas

Estructura de la subrutina

1. Salvar el BP (viejo):	*push BP*
2. Inicializa BP:	*mov BP, SP*

- Tal como se mencionó anteriormente, se usará el puntero base BP para el acceso a la estructura de datos que se formará en la Pila.
- Como se modificará el BP, siguiendo el concepto de que los registros no deben ser alterados al finalizar la subrutina, la primer instrucción tiene como objetivo salvaguardar el valor actual del BP.
- Una vez hecho esto, la segunda instrucción inicializa el puntero BP para que apunte a la estructura de datos que se formará en la Pila.

Detalles de uso de subrutinas



SP y BP antes de invocar SUBR 1

SP y BP después del paso 2 en SUBR1

Detalles de uso de subrutinas

- BP es el puntero al área de la pila asignada al procedimiento (normalmente llamado “Frame pointer”). Para acceder a los datos en el área de la Pila asignada al procedimiento se deberá sumar un desplazamiento fijo al BP.
- De esta manera se 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(es decir, no hay un direccionamiento base o indexado que use el SP). El valor de SP puede cambiar, pero típicamente BP permanece 'quieto'.

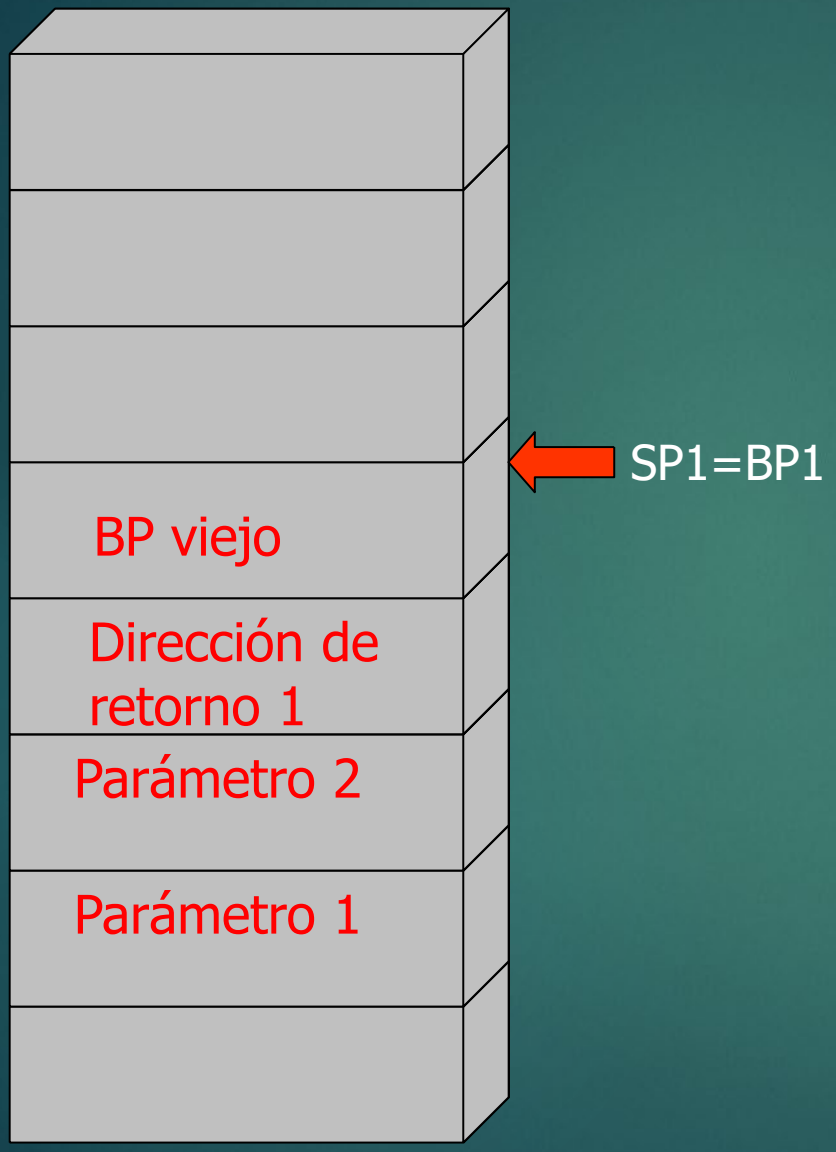
Detalles de uso de subrutinas

Estructura de la subrutina

3. Reserva espacio para variables locales *sub SP,2*

- El tercer paso en la subrutina consiste en reservar espacio para almacenar variables locales (variables temporales usadas por la subrutina y que al finalizer se deben descartar).
- Eso se puede hacer, por ejemplo moviendo el SP una cantidad determinada de bytes.
- En el ejemplo anterior, se reservan 2 bytes para variables locales, decrementando el SP en 2.
- El sistema puede utilizar al SP sin escribir sobre el área de trabajo (o frame) del procedimiento.

Detalles de uso de subrutinas



Pila y punteros después del paso 2

Pila y punteros después del paso 3

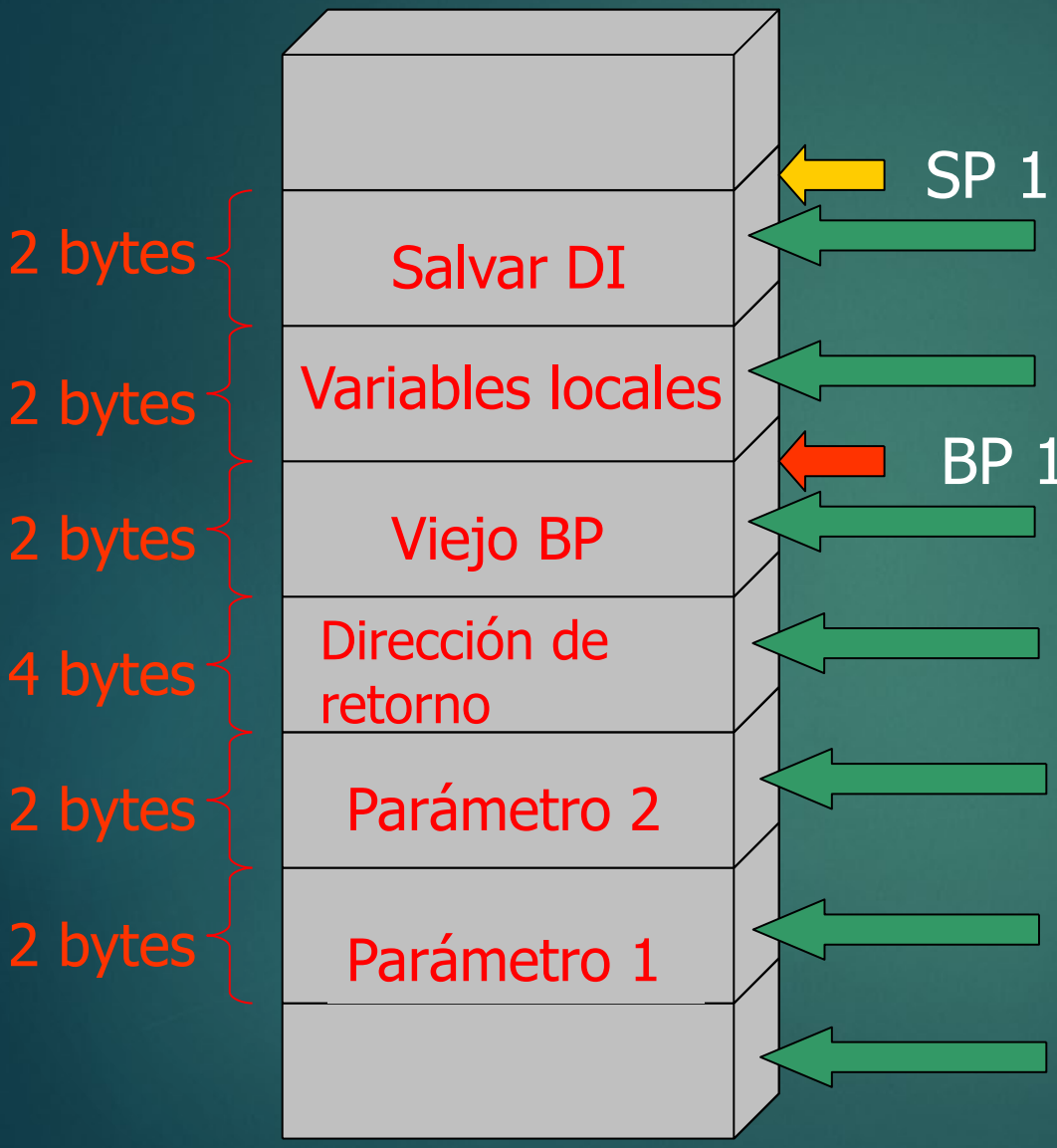
Detalles de uso de subrutinas

Estructura de la subrutina

4. Salva otros registros
usados en la subrutina *push DI*

- En el caso que la subrutina necesite usar registros de la CPU, es aconsejable salvarlos antes, para restablecerlos previo a retornar de la subrutina.
- En el paso 4 se salva el ó los registros que sean necesarios. En este ejemplo se salva el registro DI.
- No necesitan salvarse los registros que no son modificados.
- Normalmente los registros son salvados después de establecer el puntero (frame pointer) y reservar el área para los datos locales.

Detalles de uso de subrutinas



Pila y punteros después del paso 4

Detalles de uso de subrutinas

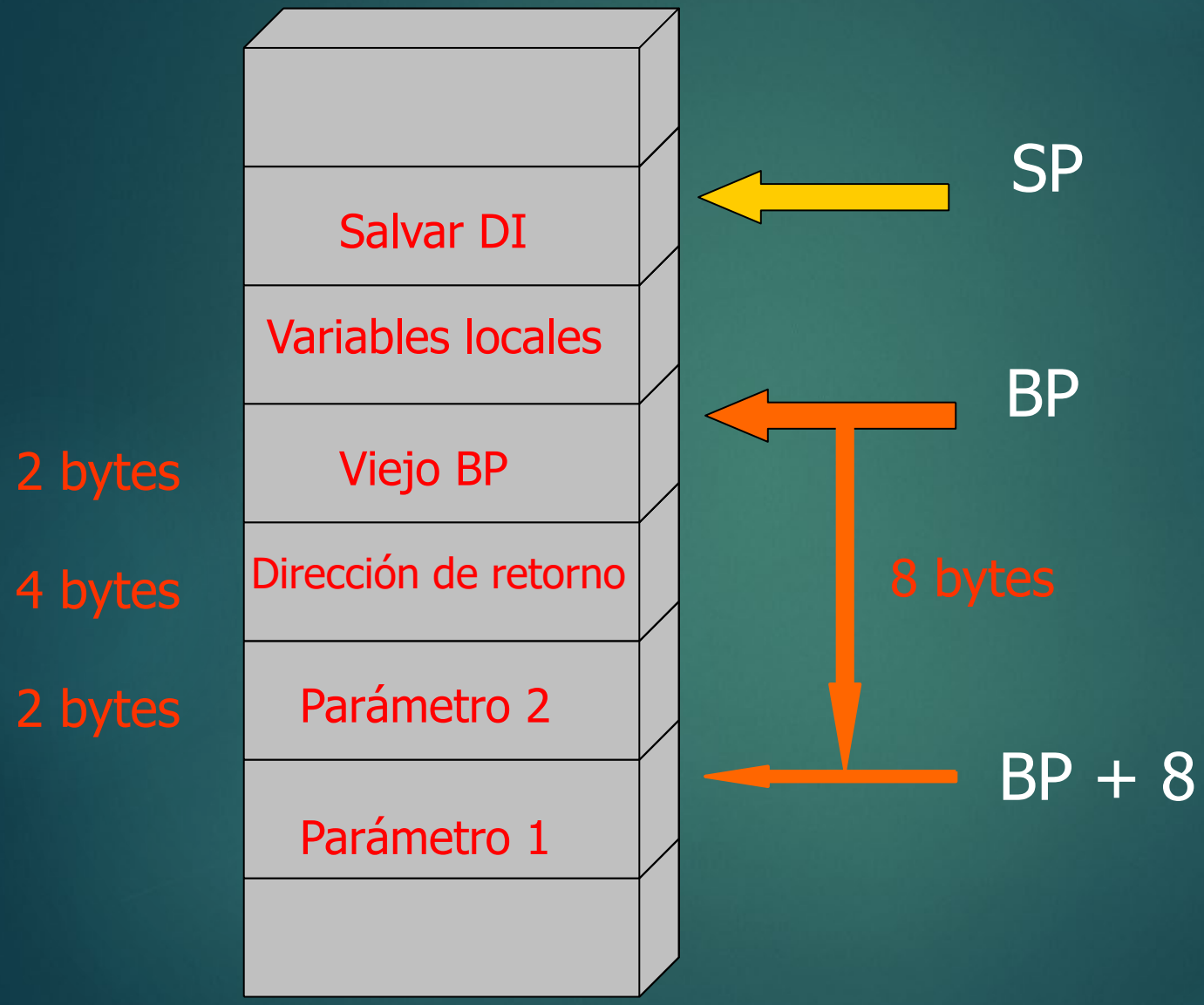
Estructura de la subrutina

5. Acceder a un parámetro *mov CX, [BP + 8]*

- En general el acceso a un parámetro se hace sumando un desplazamiento a BP (BP no se modifica), correspondiente a un direccionamiento base con offset.
- El offset que se suma al BP se obtiene sumando todos los bytes intercalados entre la posición actua del BP y la posición del parámetro a buscar.
- Por ejemplo, para acceder al Parámetro 1, el offset es de:
  - 4+2+2= 8 bytes

  y el desplazamiento es BP+8.

Detalles de uso de subrutinas



Acceso a Parámetro 1

Detalles de uso de subrutinas

Estructura de la subrutina

6. Cuerpo de la subrutina ….

….

- En el paso 6 se escriben las instrucciones propias de la tarea que desarrolla la subrutina, y que constituyen el cuerpo del procedimiento.

Detalles de uso de subrutinas

Estructura de la subrutina

7. Restaurar los registros modificados *pop DI*

- En el paso 7 se restauran los valores originales en los registros usados durante la subrutina.
- En el ejemplo, solo se usó el DI (que fué salvado en el paso 4), por lo que solo ese registro es recuperado.

Detalles de uso de subrutinas

Estructura de la subrutina

8. Restaurar SP y BP	*mov SP,BP* *pop BP*

- En el paso 8 se restauran los valores originales de SP y BP.
- La primer instrucción (mov SP, BP) restablece el valor de SP que pudo haberse movido cuando se reservó espacio para las variables locales (paso 3). Notar que el BP no se movió durante el acceso a la estructura de datos en la Pila.
- La segunda instrucción (pop BP), restaura el valor original (el valor “viejo”) que tenía el BP antes de ingresar a la subrutina.

Detalles de uso de subrutinas

Estructura de la subrutina

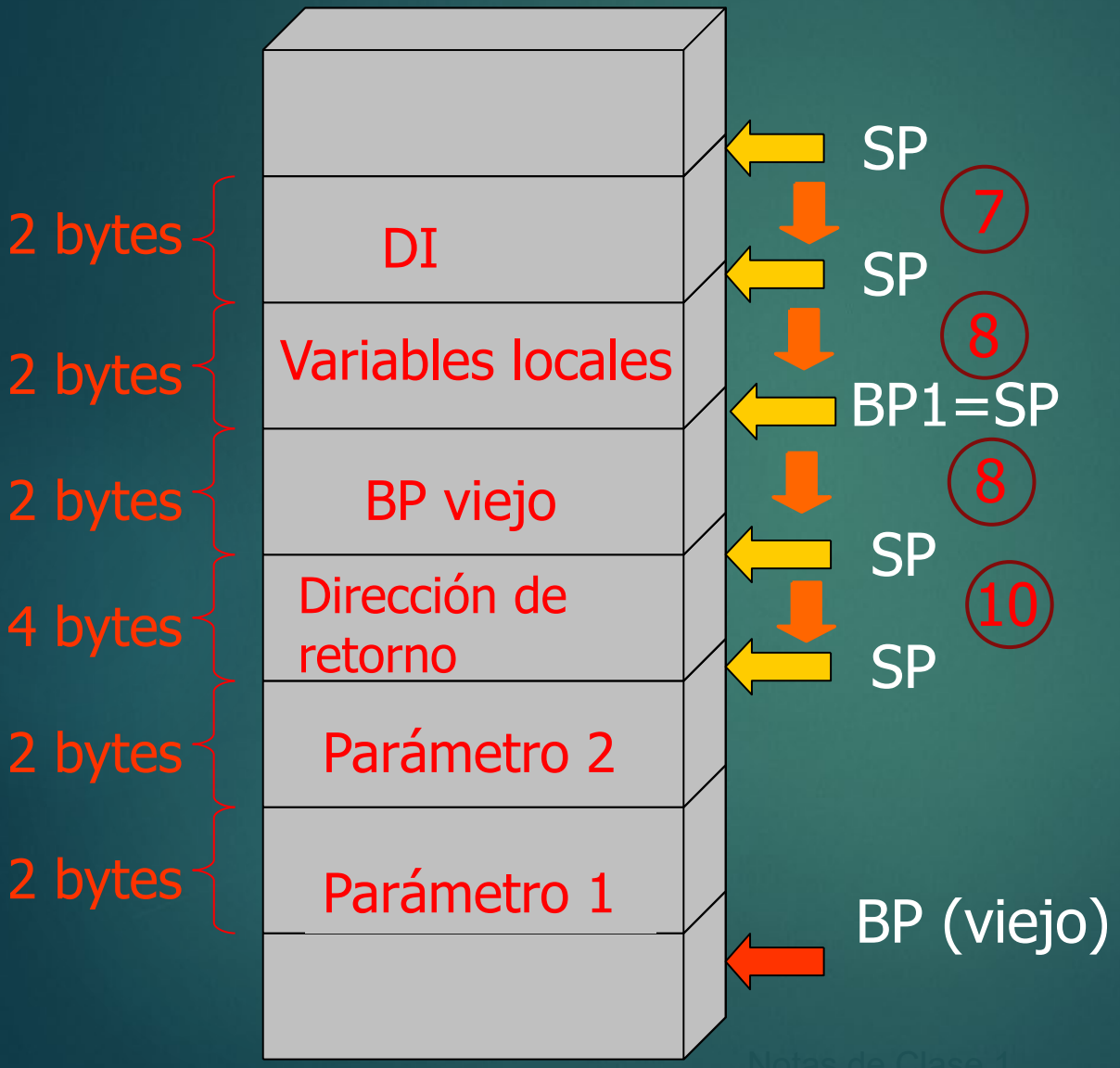
9. Retornar parámetros	….
	….

- En el paso 9 se actualizan los resultados obtenidos por la subrutina a ser devueltos al programa que la invoca.

10.- Retornar	*ret*

- El ultimo paso de la subrutina es la instrucción de retorno de subrutina Ret. De esta manera se le devuelve el control al programa principal (que invoca la subrutina) en la instrucción siguiente a la instrucción que invoca a la subrutina (CALL).

Detalles de uso de subrutinas



Pila y punteros pasos 7, 8, 9 y 10

7.- Restaurar DI

8.- Descartar espacio reservado para datos locales y reponer BP

9.- Pasar los resultados

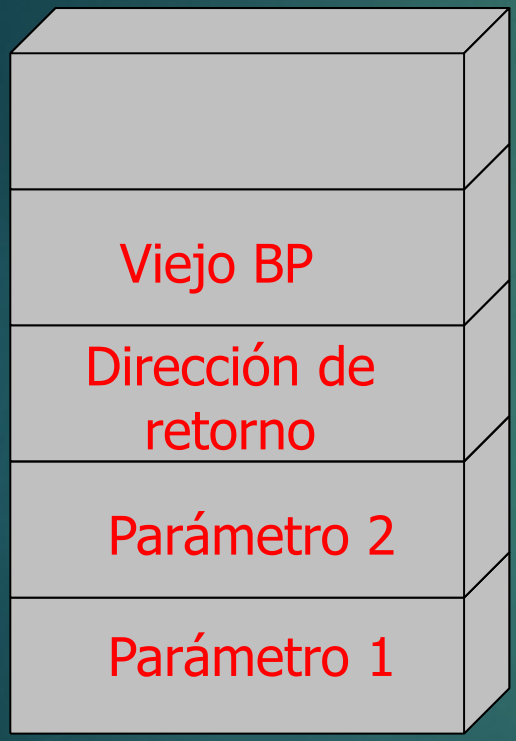
10.- Retornar

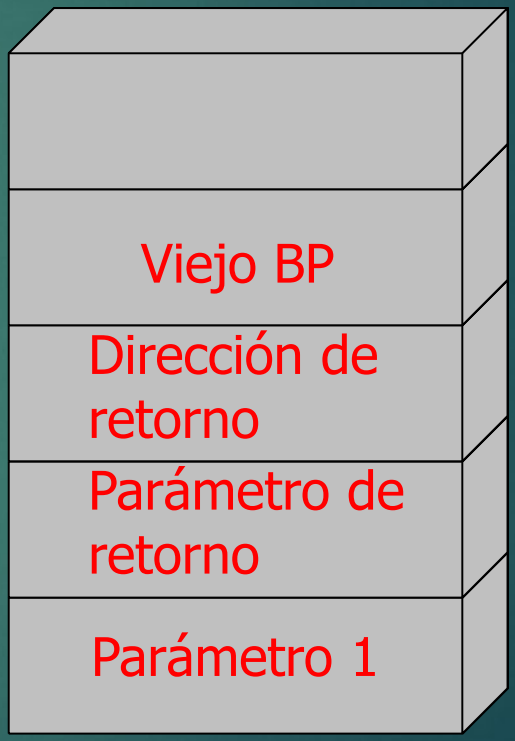
Detalles de uso de subrutinas

Estructura de la subrutina – Parámetros de retorno

- Las subrutinas realizan un trabajo determinado. Ese trabajo puede producir un resultado, que debe ser devuelto al programa que la invoca.
- Típicamente la subrutina devuelve 1 o más resultados.
- El resultado se retorna por las mismas posiciones de memoria donde la subrutina recibió los parámetros.
- En el ejemplo, el resultado debe guardarse en la dirección de memoria con un offset de 6 respecto del valor actual de BP, es decir: *BP+6*
- Notar que en BP+2 está la dirección de retorno que NO debe ser alterada de ninguna manera.

Detalles de uso de subrutinas





Pila sin parámetros de retorno

Pila con parámetros de retorno

Anidamiento de subrutinas

- Una subrutina puede invocar a otra subrutina. Esto se conoce como “anidamiento de subrutinas”.
- Cada subrutina tiene asociado su propio espacio de memoria en la Pila, tal como se vio anteriormente.
- La cantidad de anidamientos posibles de subrutinas depende de varios factores, pero básicamente del espacio de memoria para disponer de la estructura de datos de cada subrutina. Es por eso que la cantidad de anidamiento es muy grande (casi “infinita”)
- Dado que la información se guarda en una estructura tipo LIFO, cada vez que se termina una subrutina de un nivel de anidamiento dado, se retorna a la subrutina del nivel anterior. En otras palabras, cuando se termina una subrutina se devuelve el control a la subrutina que la invocó.

Anidamiento de subrutinas

- En el ejemplo que sigue se muestra un programa (principal) PPIO que comienza en la dirección 1000.
- El PPIO invoca una subrutina SUBR1 (nivel de anidamiento 1) que comienza en la dirección 3000H.
- La subrutina SUBR1 invoca a subrutina SUBR2 (nivel de anidamiento 2) que comienza en la dirección 4000H.
- Cuando termina SUBR2, de forma correcta, retorna a la tarea que la invocó, es decir ejecuta en SUBR1 la instrucción siguiente del CALL SUBR2 (es decir ejecutará POP BX).
- Cuando termina SUBR1, retorna a PPIO, a la instrucción siguiente del CALL SUBR1 (es decir ejecutará POP BX).

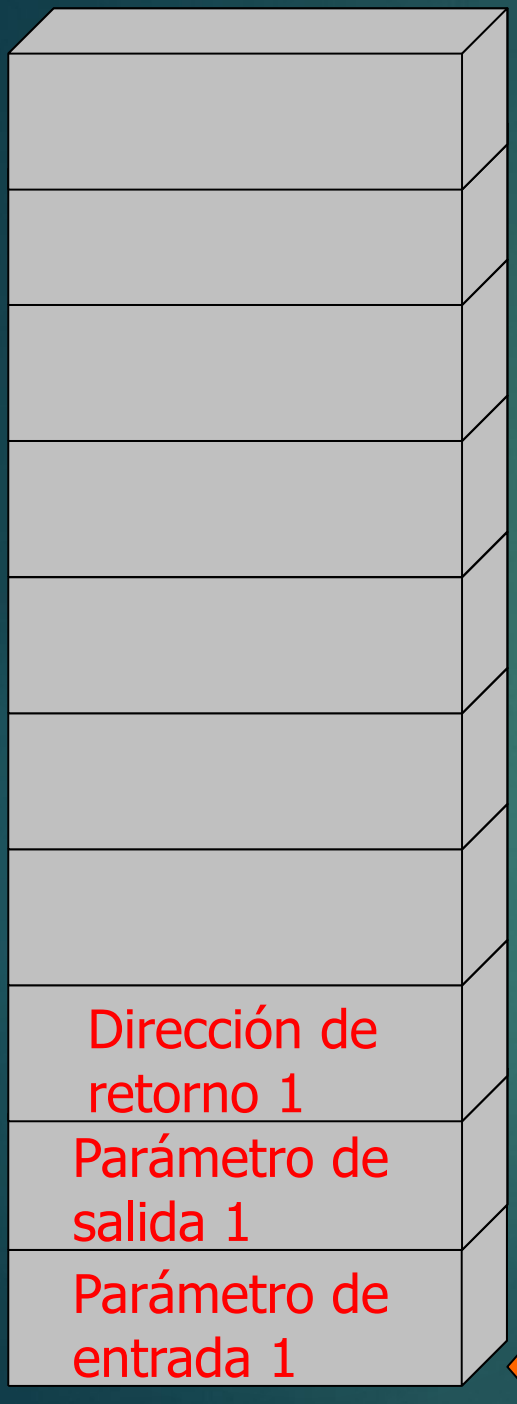
Anidamiento de subrutinas

```
        ORG 1000H
PPIO:   MOV   BX, 0
        MOV   AX, dato
        PUSH  AX
        CALL  SUBR1
        POP   BX
        HLT
        ORG 2000H
dato:   DB  55H,00H
        END
```

```
        ORG 3000H
SUBR1:  NEG   AX
        PUSH  AX
        CALL  SUBR2
        POP   BX
        RET
```

```
        ORG 4000H
SUBR2:  INC   AX
        RET
```

Anidamiento de subrutinas

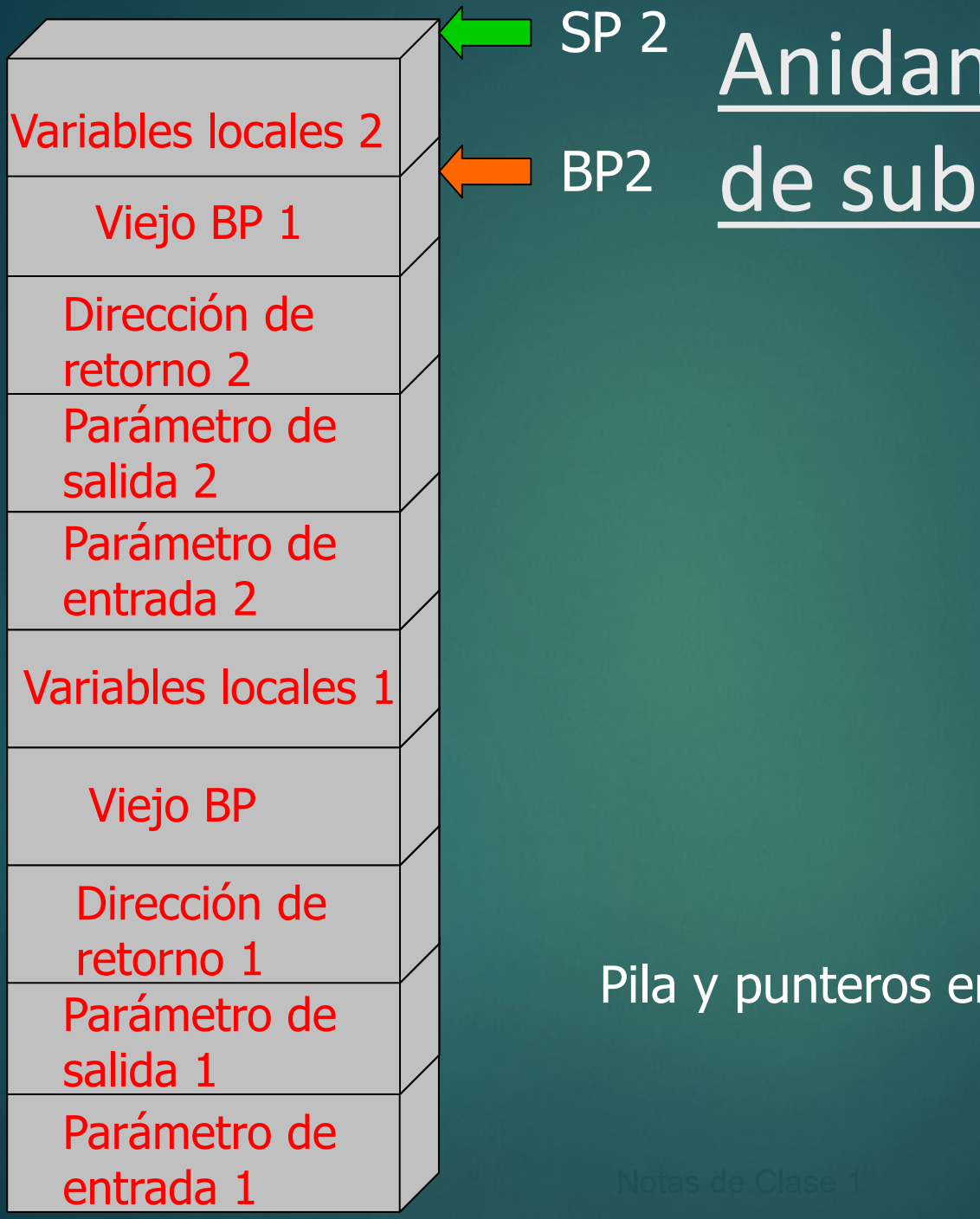


Pila y punteros al invocar la subrutina 1

Anidamiento de subrutinas

Pila y punteros al invocar la subrutina 2

Anidamiento de subrutinas



Pila y punteros en la subrutina 2

Ejemplo para simulador MSX88

- A continuación se muestra un programa de ejemplo para ser cargado y ejecutado en el simulador MSX88.
- El programa principal que invoca una subrutina empieza en la dirección 2000H.
- La subrutina a ser invocada empieza en la dirección 3000H.
- Como ejercicio, analizar la secuencia de acciones realizada en el programa principal y en la subrutina, y comparar con los pasos previstos en las estructuras de programa principal y subrutina vistos.
- Tener en cuenta que la decaración del procedimiento se hace directamente así: *nombre: instrucción*
- Además, dado que el MSX88 no dispone del registro BP, para el acceso a la estructura de datos de la Pila se usa el BX en su reemplazo.

Ejemplo para simulador MSX88

```
        ORG 1000H
NUM1    DW  5H
NUM2    DW  3H
RES     DW  ?
        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
```

```
        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
      POP  DX
      POP  AX
      POP  CX
      POP  BX
       RET
```

Bibliografía

- ***Organización y Arquitectura de Computadoras – Diseño para optimizar prestaciones***, Stallings W., Editorial Prentice Hall (5º edición).
- ***Organización de Computadoras,*** Tanenbaum A., Editorial Prentice Hall (4º edición).
- ***Arquitectura de Computadores - Un enfoque cuantitativo***, Hennessy & Patterson., Editorial Mc Graw Hill (1º edición).
- ***Diseño y evaluación de arquitecturas de computadoras***, Beltrán M. y Guzmán A., Editorial Prentice Hall (1º edición).

Referencias

- 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 MSX88