# Arquitectura de Computadoras

**Clase 1**

# 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

Notas de Clase 1

# Temas de clase

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

# Concepto de programa

* Antes se tenían sistemas cableados

Secuencia de

funciones

aritmético/lógicas

Datos

Resultados

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

# Concepto de programa (2)

•

Ahora

Códigos de instrucción

Intérprete de

instrucción

Datos

Funciones

aritmético/lógicas

Resultados

Señales de control

• 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

Memoria

PC

IR

MAR

MBR

E/S AR

E/S BR

CPU

Módulo de E/S

IR = Registro de instrucción

Instrucción

Instrucción

Instrucción

Datos

Datos

Datos

Datos

PC = Contador de programa

Registros

MAR = Registro de dirección de memoria

MBR = Registro de buffer de memoria

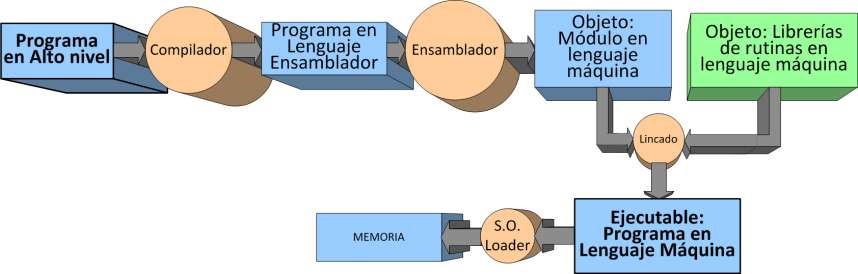
E/S AR = Registro de dirección de E / S

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

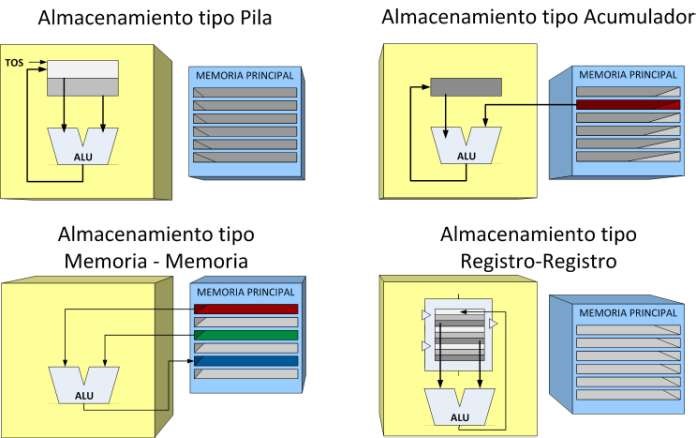
**¿Dónde se almacenan operandos?**

* Memoria principal
* o memoria virtual o en memoria cache

* Registro de la CPU

* Dispositivo de E/S

# Alternativas de almacenamiento



# 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

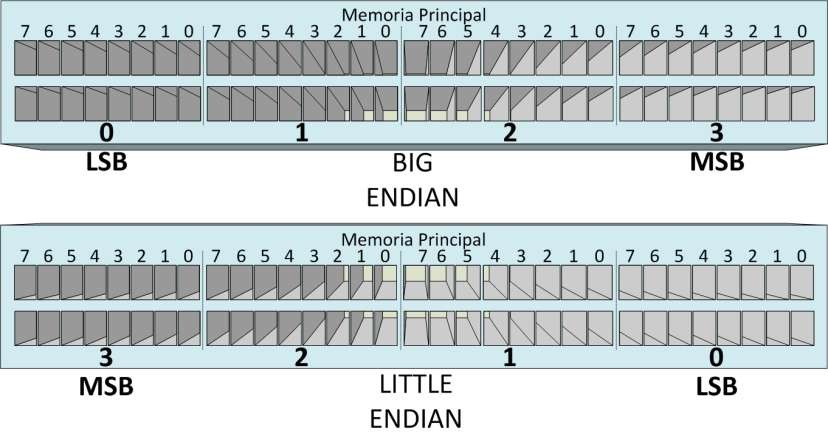
Supongamos una memoria direccionable de a byte

• ¿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)

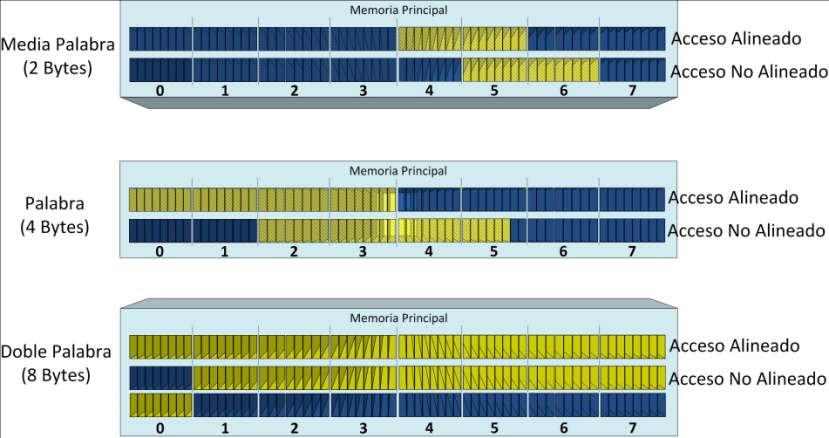
|  |  |  |
| --- | --- | --- |
| Dir. de byte | Forma 1 | Forma 2 |
| 00 | 98 | 32 |
| 01 | 76 | 54 |
| 02 | 54 | 76 |
| 03 | 32 | 98 |
| ¿cuál forma uso? | Big endian: el byte más significativo en la dirección con valor numérico más bajo | Little endian: el byte menos significativo en la dirección con 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**stract, **Mul**tiply y **Div**ide

* Números enteros sin/con signo.
* ¿Números en punto flotante?
* Pueden incluirse otras operaciones ...
* **Inc**rement o **Dec**rement (en 1 el operando)
* **Neg**ate: cambia el signo del operando (Ca2).
* **Abs**olute: 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)

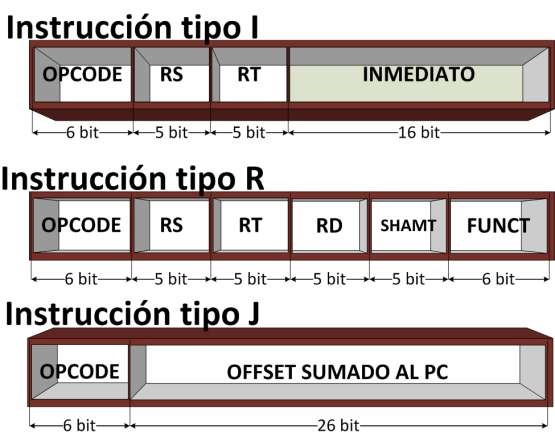
# Control de flujo

Modifican el valor contenido en el registro PC

* Salto Incondicional
* **JMP** equis ; saltar a la posicion ‘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

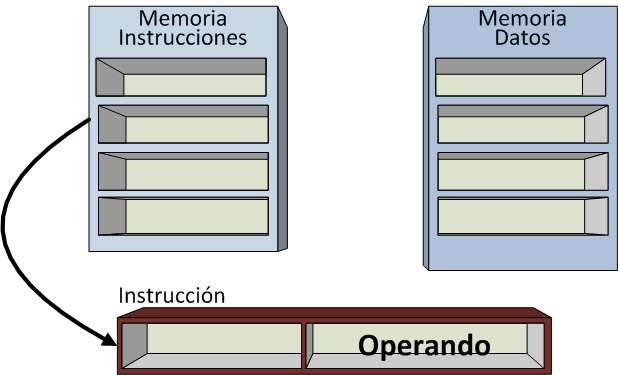
# Formatos de instrucción



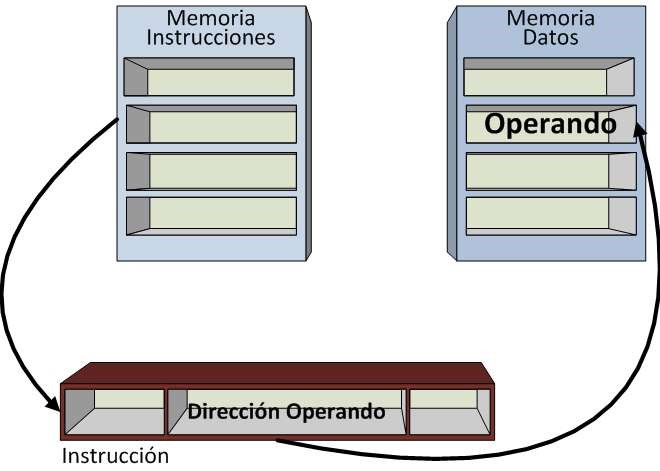
# Modos de direccionamiento

* 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)

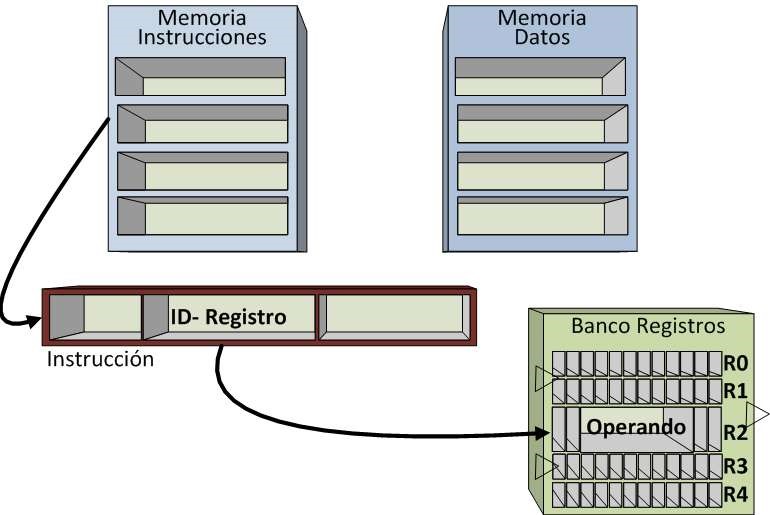
# MDD Inmediato



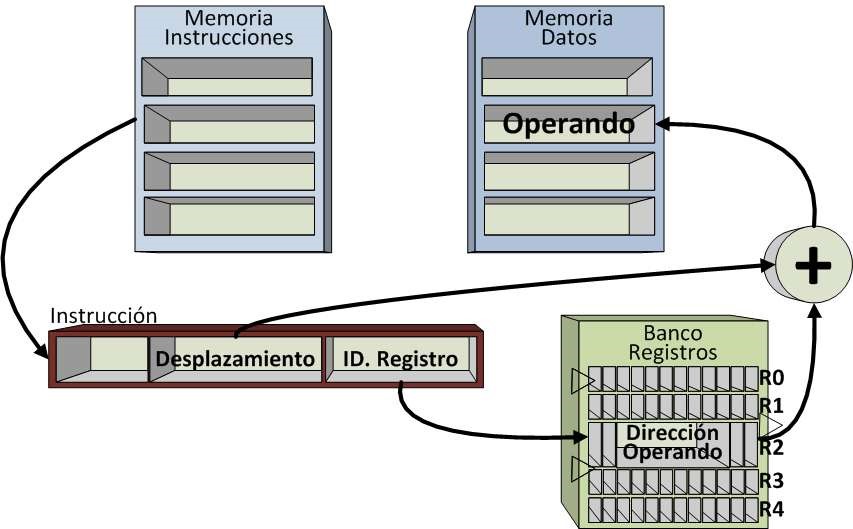
# MDD Directo o Absoluto (de memoria)



# MDD Directo de Registro



## MDD Indirecto con desplazamiento



# instrucción básico

* Dos pasos:
* Captación
* Ejecución

7

INICIO

Captar la

siguiente

instrucción

Ejecutar la

instrucción

PARADA

**Ciclo de captación Ciclo de ejecución**

3

# 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 lleva a cabo la acción requerida

# 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

Captación de

instrucción

Cálculo de la

dirección de

instrucción

Decodifica

-

ción de la

operación de la

instrucción

Cálculo de la

dirección de

operando

Operación

con datos

Cálculo de la

dirección de

operando

Almacena

-

miento del

operando

Captación

de

operando

Varios

operandos

Varios

resultados

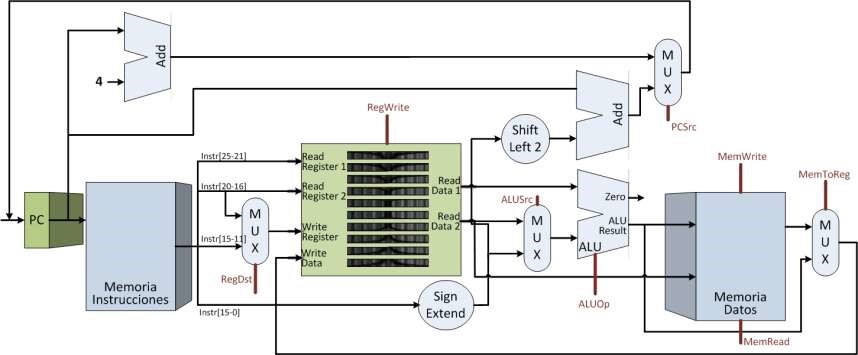
Instrucción completada,

captar siguiente instrucción

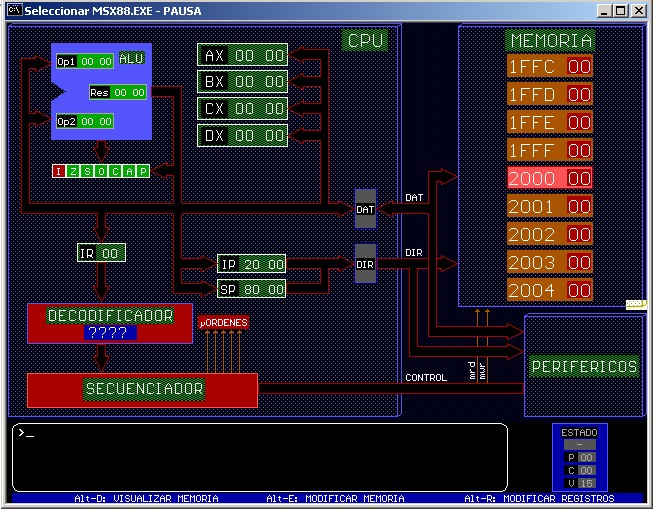
Cadena o vector

de datos

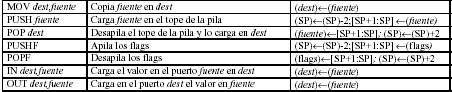
# Ruta de Datos



# Simulador MSX88



# MSX88: inst. de transferencia

1

2

2

2

3

4

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

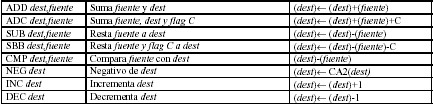
*mem* puede ser una etiqueta (dir.directo) o [BX] (dir.indirecto).

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

# Inst. aritméticas y lógicas

1



1

1

1

1 5 5

5

1

1

1

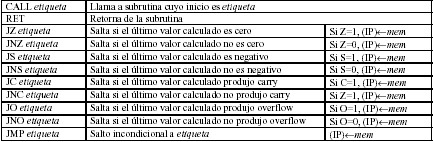
5

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 memoria (dir.indirecto).

# Inst. transf. de control

6

6

6 6 6

6

6 6 6

6

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

Apilar Desapilar

SP x86

2

1

2

1

* Mover dato
* Modificar SP

2

2

1

1

PUSH POP

## Ejemplo en Assembly y máquina

Dir. Codigo maquina Linea Codigo en lenguaje ensamble

1 ORG 2000H

2000 BB 00 30 2 MOV BX, 3000H

2003 8B 07 3 MOV AX, [BX]

1. 50 4 PUSH AX
2. BB 02 30 5 MOV BX, 3002H

2009 8B 0F 6 MOV CX, [BX]

200B 51 7 PUSH CX 200C 58 8 POP AX 200D 59 9 POP CX

200E F4 10 HLT

11 ORG 3000H

3000 55 33 44 22 12 datos DB 55h, 33h, 44h, 22h

13 END

S I M B O L O S:

Nombre: Tipo: Valor:

datos Byte 3000h No usado

# Definición del procedimiento

Nombre Proc

... …

Cuerpo del procedimiento

…

Ret

Nombre Endp

# Llamada al procedimiento

En programa principal

…

Push Parametro 1 Push Parametro 2

Call Nombre

… …

# Ejemplo con subrutina

|  |  |  |
| --- | --- | --- |
|  | **ORG 1000H** |  |
| subrutin: | NEG AX |  |
|  | RET |  |
|  | **ORG 2000H** |  |
|  | MOV BX, 0 |  |
|  | MOV AX, dato |  |
| dato | PUSH AX  CALL subrutin  POP BX  HLT  **ORG 3000H**  DB 55H  **END** | Analizar la pila y los valores finales de AX y BX |

Listado Fuente: subrut14.LST

Programa Fuente en: subrut14.ASM

Fecha: Tue Aug 19 14:46:48 2014

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dir. Codigo maquina Linea | | Codigo en lenguaje ensamble | | |
|  | 1 |  | ORG | 1000H |
| 1000 F7 D8 | 2 | subrutin: | NEG | AX |
| 1002 C3 | 3 |  | RET |  |
|  | 4 |  |  |  |
|  | 5 |  | ORG | 2000H |
| 2000 BB 00 00 | 6 |  | MOV | BX, 0 |
| 2003 8B 06 00 30 | 7 |  | MOV | AX, dato |
| 2007 50 | 8 |  | PUSH | AX |
| 2008 E8 00 10 | 9 |  | CALL | subrutin |
| 200B 5B | 10 |  | POP | BX |
| 200C F4 | 11 |  | HLT |  |
|  | 12 |  | ORG | 3000H |
| 3000 55 | 13 | dato | DB | 55H |
| S I M B O L O S: | 14 |  | END |  |
| Nombre: Tipo: |  | Valor: |  |  |
| subrutin Label |  | 1000h |  |  |
| dato Byte |  | 3000h |  |  |

# 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

Parámetro 1

Parámetro 2

Dirección de

retorno 1

Parámetro 1

Parámetro 2

Dirección de

retorno 1

Viejo BP

Variables locales 1

SP

BP

SP 1

BP 1

# 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.

Posición de SP luego de salvar por ej. DI

Parámetro 1

Parámetro 2

Dirección de

retorno

Viejo BP

Variables locales

Salvar DI

2

bytes

2

bytes

2

bytes

4

bytes

2

bytes

2

bytes

Espacio para datos locales

Salvar viejo BP y hacer BP=SP

SP al entrar al procedimiento

SP luego de pasar parámetro 2 y antes de llamar al procedimiento

SP luego de pasar parámetro 1 SP antes de pasar parámetros

## 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 ]

Parámetro 1

Parámetro 2

Dirección de retorno

Viejo BP

Variables locales

Salvar DI

BP

BP + 8

SP

2

bytes

4

bytes

2

bytes

# 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 Con parámetro de retorno retorno

Parámetro

Dirección de

retorno

Viejo BP

Parámetro

Dirección de retorno

de parámetro

Dirección de

retorno

Viejo BP

BP

BP+2

BP+6

BP

BP+2

BP+8

BP+6

B

2

B

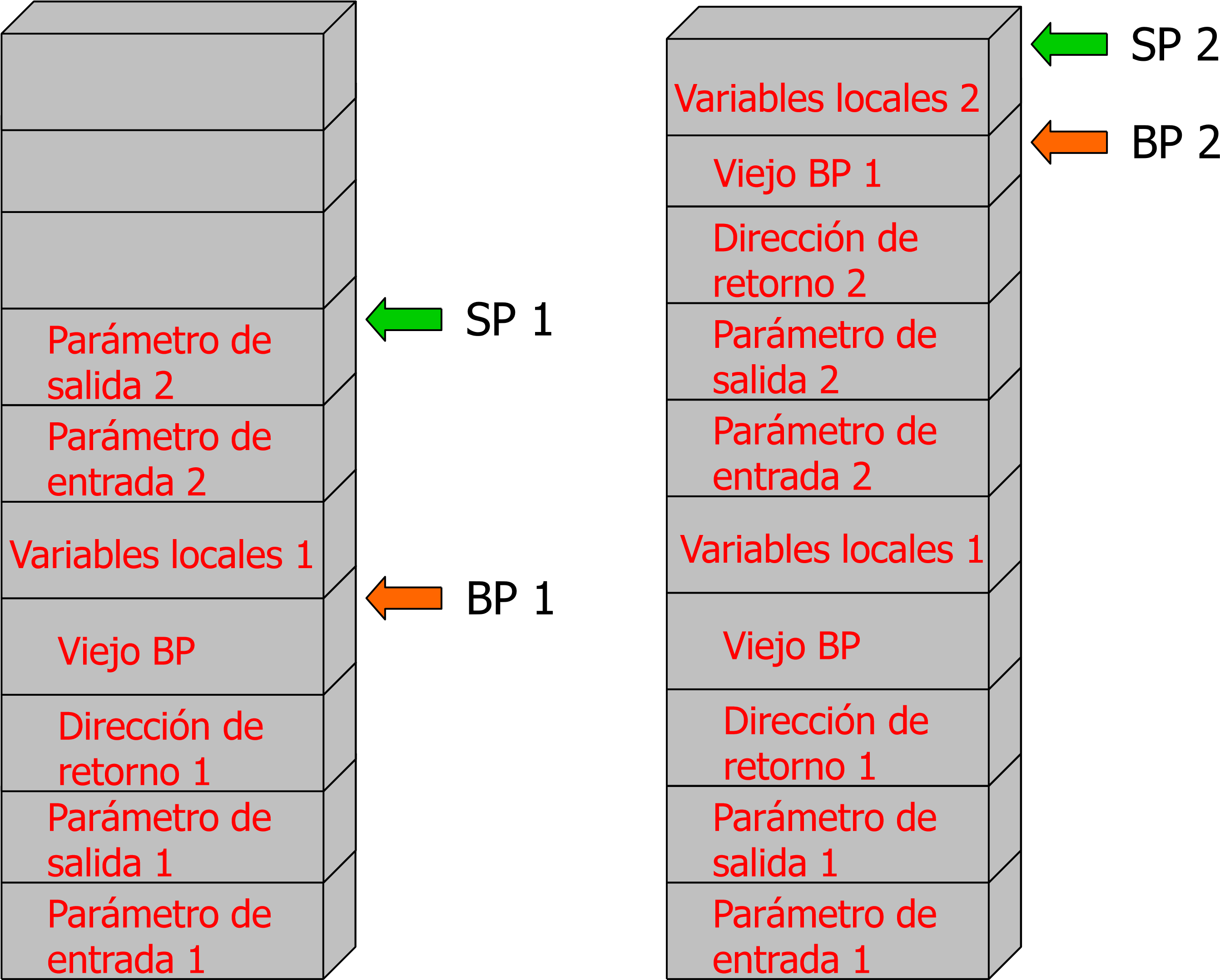
4

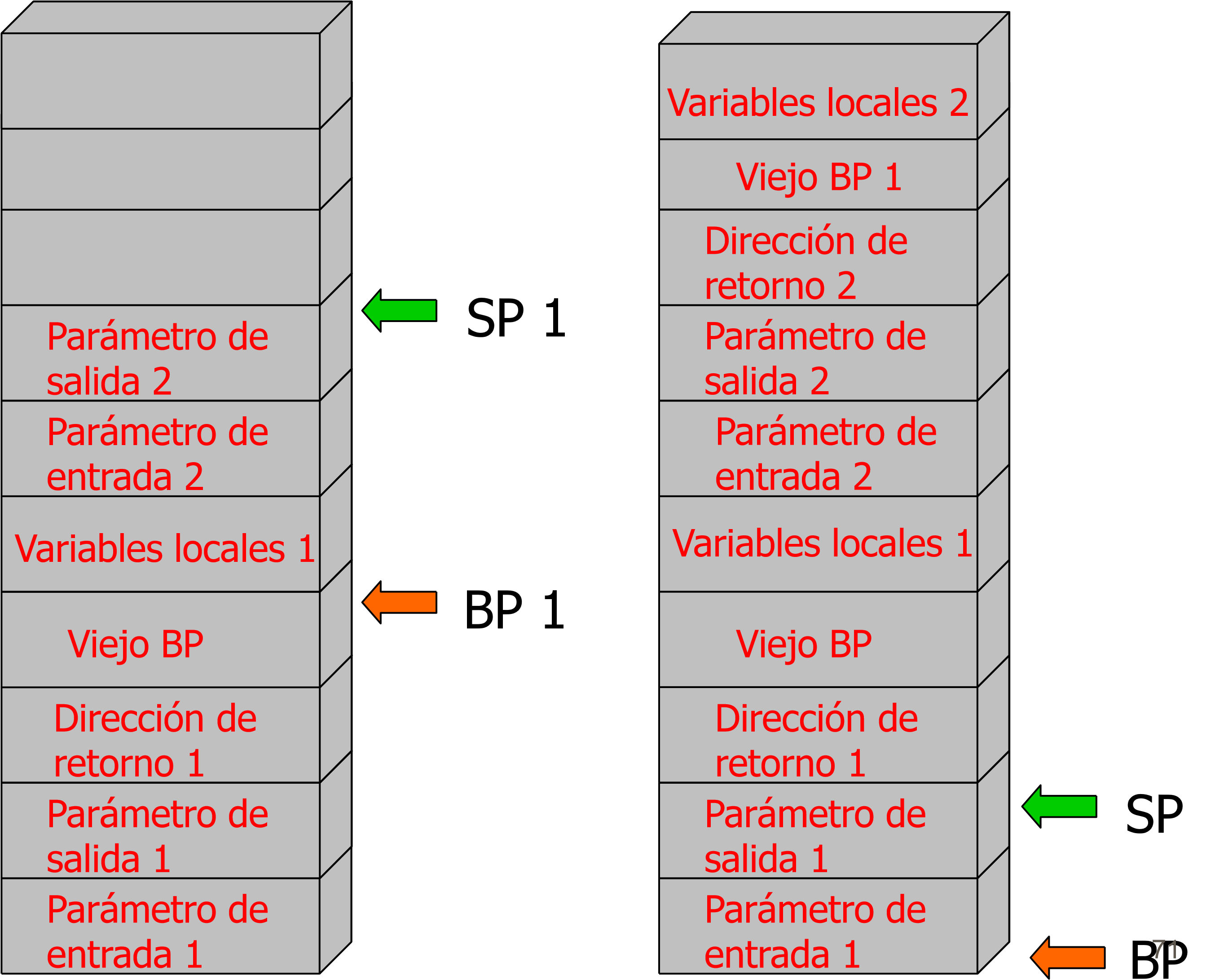
B

2

# Anidamiento de subrutinas

|  |  |  |  |
| --- | --- | --- | --- |
| **ORG 1000H** rutina1: NEG AX  PUSH AX  CALL rutina2  POP AX  RET  **ORG 1020H**  rutina2: INC AX  RET |  | PPIO:          dato: | **ORG 2000H**  MOV BX, 0  MOV AX, dato  PUSH AX  CALL rutina1  POP BX  HLT  **ORG 3000H**  **DB** 55H |
|  |  |  | **END** |





# Para el simulador

* Declaración del procedimiento nombre: instrucción

. .

* En lugar de BP se usa BX

# Ejemplo para simulador MSX88

|  |  |  |
| --- | --- | --- |
| **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 MSX88

Link de interés: www.williamstallings.com