

Arquitectura de Computadoras



Curso 2020

Clase 1

Pasaje de parámetros

Temas de clase

- Programas
- Subrutinas
- Pasaje de parámetros
- Funcionamiento de la pila
- Push/Pop

Subrutinas

- ❑ Misma idea de los procedimientos en Pascal.
- ❑ También tiene una definición determinada. Encabezamiento y final. Cuerpo del procedimiento (instrucciones).
- ❑ Vamos a ver ésto para el simulador.

Definición de subrutinas (simulador)

ORG 3000H

Nombre: ...

...

...

Cuerpo del procedimiento

Como la etiqueta
de los saltos

Ret

- 3000H=Distinta a 2000H (programa principal). La subrutina no debe sobre escribir otras zonas del programa

Pasaje de parámetros a subrutinas

- ❑ Procedimientos (subrutinas), en general requieren datos de entrada y proveen datos de salidas.
- ❑ Parámetros son estos datos pasados a y desde el procedimiento.
- ❑ Estos parámetros pueden ser pasados de dos maneras :
 - Por valor
 - Por referencia

Parámetros por valor

- Es eso: se pasa el valor de una variable al procedimiento.
- Son sólo parámetros de entrada.
- Independientemente del uso de este valor por parte del procedimiento, éste no puede ser modificado.

Parámetros por referencia

- Aquí es pasada la dirección de la variable y no su valor.
- Es necesario cuando hay que modificar el valor del parámetro (retorno).
- En general menos eficiente que pasar por valor. Tenemos la dir y hay que acceder a memoria para buscar el dato.
- Pero más eficiente cuando hay que transferir un arreglo datos entre proc. Así se pasa la dir del arreglo.

¿Dónde se pasan los parámetros?

- Vía registros

- El número de registros es la principal limitación
- Es importante documentar que registros se usan

- Vía memoria

- Variables definidas en el programa (“globales”)
- Se usa un área definida de memoria (RAM).
- Difícil de estandarizar

¿Dónde se pasan los parámetros?

- Vía pila (stack)
 - Es el método más ampliamente usado.
 - La pila (ó stack) es una zona de memoria (RAM) destinada a este uso específico.
 - Lo usa la mayoría de los lenguajes de alto nivel.
 - Independiente de memoria y registros.
 - Hay que comprender bien como funciona porque la pila (stack) es usada por el usuario y por el sistema.
 - Aunque es más complicado, en general los registros son recursos muy preciados.

Funcionamiento de la pila

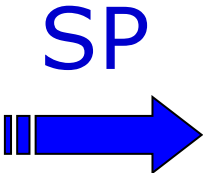
- Zona de memoria destinada a un uso específico
 - Uso del sistema: salva la dirección de retorno cuando es llamada una subrutina ó se produce una interrupción por hardware
 - Uso del usuario: pasaje de parámetros
- Cada vez que se ejecuta un programa, el so inicializa el reg SP apuntando a la pila
 - El simulador inicializa SP=8000H

Operaciones de apilar/desapilar

- PUSH : apila datos. El SP apunta al último dato almacenado, por lo tanto primero se decrementa y luego almacena el dato.
- POP : desapila datos. Desapila el dato y luego incrementa el SP. Operación inversa a la anterior.
- PUSH y POP apilan y desapilan datos de 16 bits (2 bytes).

MOV AX, 9234H
PUSH AX


Apilar



7FFBH	00H
7FFCH	00H
7FFDH	00H
7FFEH	00H
7FFFH	00H
8000H	00H

Antes de ejecutar el push

SP

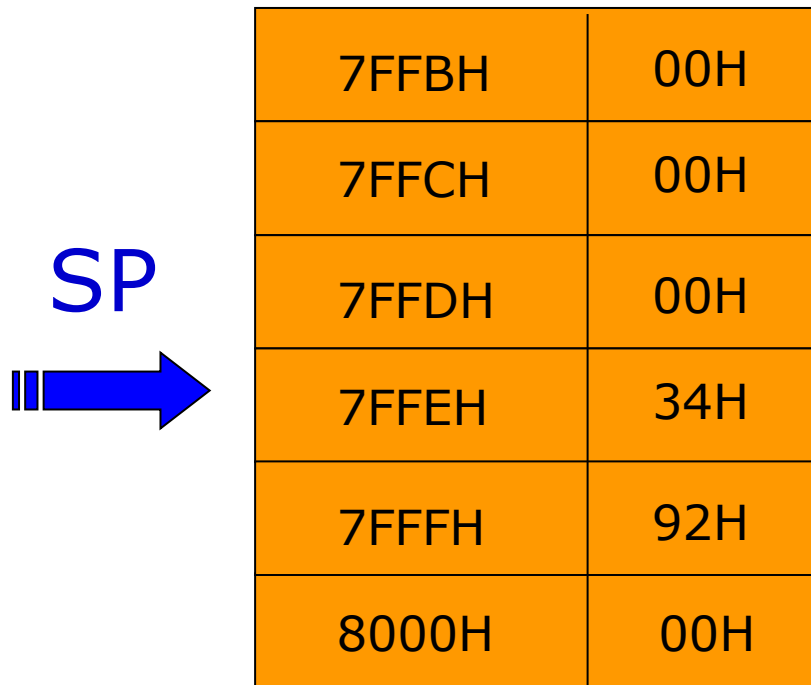


7FFBH	00H
7FFCH	00H
7FFDH	00H
7FFEH	34H
7FFFH	92H
8000H	00H

Después de ejecutar el push

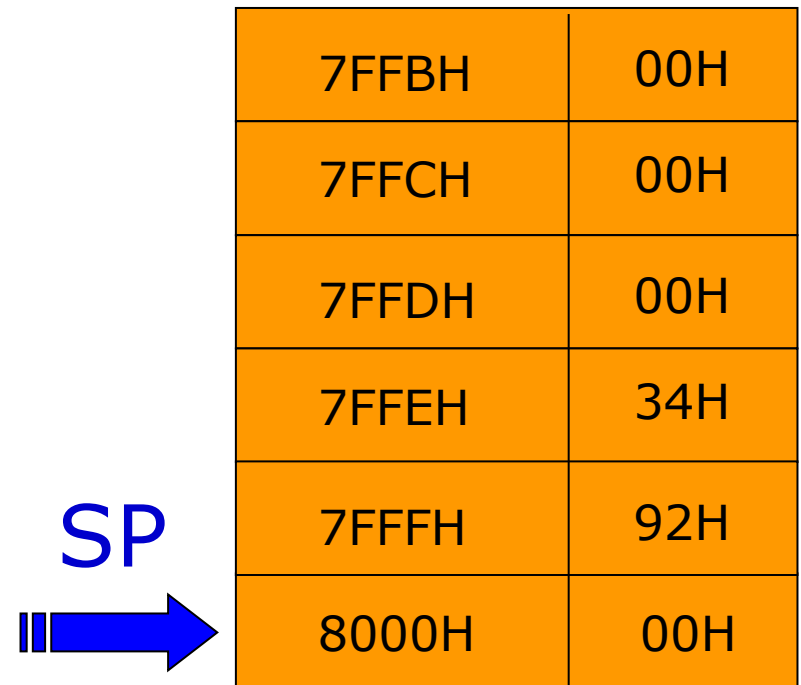
POP BX

Desapilar



Antes de ejecutar el pop

BX 00 00

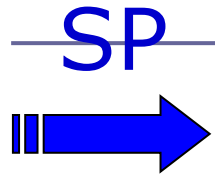


Después de ejecutar el pop

BX 92 34

PUSH AX
PUSH BX

AX	92	34
BX	76	81

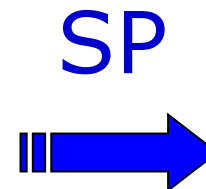


7FFBH	00H
7FFCH	81H
7FFDH	76H
7FFEH	34H
7FFFH	92H
8000H	00H

Después de ejecutar el push

POP CX
POP DX

AX	92	34
BX	76	81



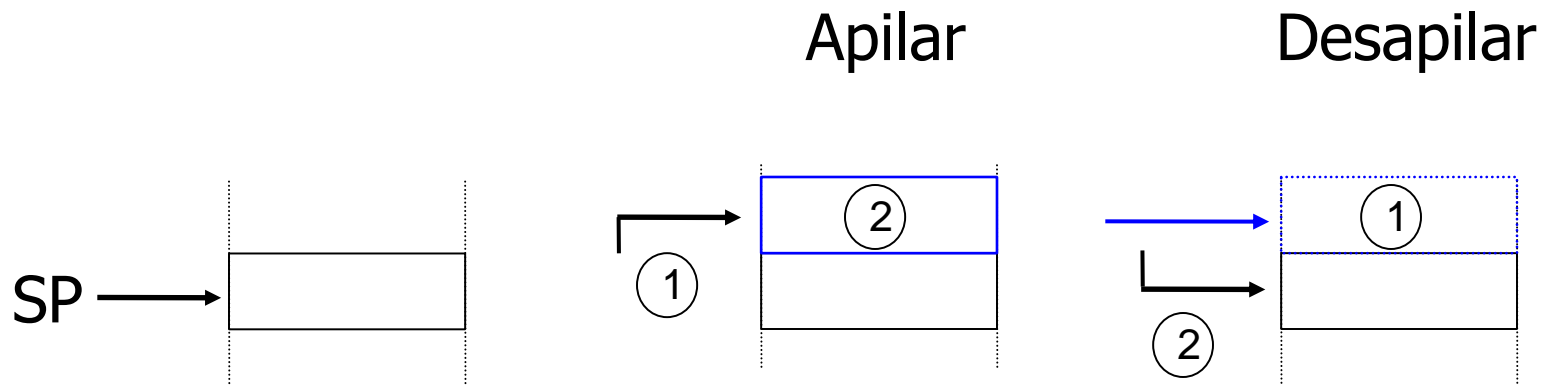
7FFBH	00H
7FFCH	81H
7FFDH	76H
7FFEH	34H
7FFFH	92H
8000H	00H

Después de ejecutar el pop

CX	76	81
DX	92	34

Resumiendo:

funcionamiento de la pila



- Mover dato
- Modificar SP
- Siempre 2 bytes (la figura muestra solo 1)

En HLL y por
valor

En x86 por valor
en registros

En x86 por valor
en la pila

Procedure Mio (I)

.....

.....

Begin

.....

.....

Mio (x);

Mio Proc

...El dato está en ax..

.....

Mio Enp

.....

dato dw

.....

mov ax, dato

call Mio

Mio Proc

...El dato está en la
pila.....

.....

Mio Enp

dato dw

.....

mov ax, dato

push ax

call Mio

En HLL y por
referencia

Procedure Mio (Var:I)

.....

.....

Begin

.....

.....

Mio (x);

En x86 por
referencia en
registros

Mio Proc

....La dir de dato en
.....ax.....

.....

Mio Enp

.....

dato dw

.....

mov ax, offset dato
call Mio

En x86 por
referencia y en la
pila

Mio Proc

...La dir de dato en la
pila.....

.....

Mio Enp

.....

dato dw

.....

mov ax,offset dato
push ax
call Mio

Llamada al procedimiento y pasaje de parámetros por pila

En programa principal

```
...  
Push Parámetro 1  
Push Parámetro 2  
Call  Nombre  
...  
...
```

Uso de registros con la pila (1)

- SP = Stack pointer (Puntero de pila).
- Apunta al último dato almacenado en la pila :
- Por el usuario con push y pop
- Por el sistema al salvar la dirección de retorno en el llamado a una subrutina ó cuando se produce una interrupción (la veremos más adelante).

Uso de registros con la pila (2)

- ❑ BP= Base pointer (Puntero base). Como SP no es un registro base ó índice, no puede ser usado para direccionar entre corchetes. No está permitido [SP].
- ❑ Para direccionar la pila se usa BP. Así si se ejecutan instrucciones push y pop se modifica sp; si hay una llamada a procedimiento también se modifica SP.
- ❑ BP está manejado por el usuario.

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

En el procedimiento.....

Paso 1: salvar el estado de bp (anterior)

Primero Proc

OPCIONAL

push bp

OPCIONAL

mov bp, sp

OPCIONAL

sub sp, 2

push ax

mov cx, [bp+número]

(instrucciones)

pop ax

add sp, 2

pop bp

ret

Paso 2 : copio sp. Uso bp para

Paso 3 : corre 2 lugares sp y esos 2 bytes pueden ser usados para "locales" de procedimiento

Porque estaba push ax

Porque push bp

Paso 7

Endp

Primero

Porque sub sp/2

Pasos.....dentro del procedimiento

- 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 fijo.

Pasos.....dentro del procedimiento

- 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.....dentro del procedimiento

- 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.



Si hay otra llamada a Procedimiento ó push, el movimiento de SP es hacia "arriba" y no sobrescribe los bytes de variables locales

SP y BP después del paso 3. Todo esto pasa dentro del procedimiento

Call

En el programa principal antes de llamar a la subrutina




SP y BP al entrar a Primero (proc)

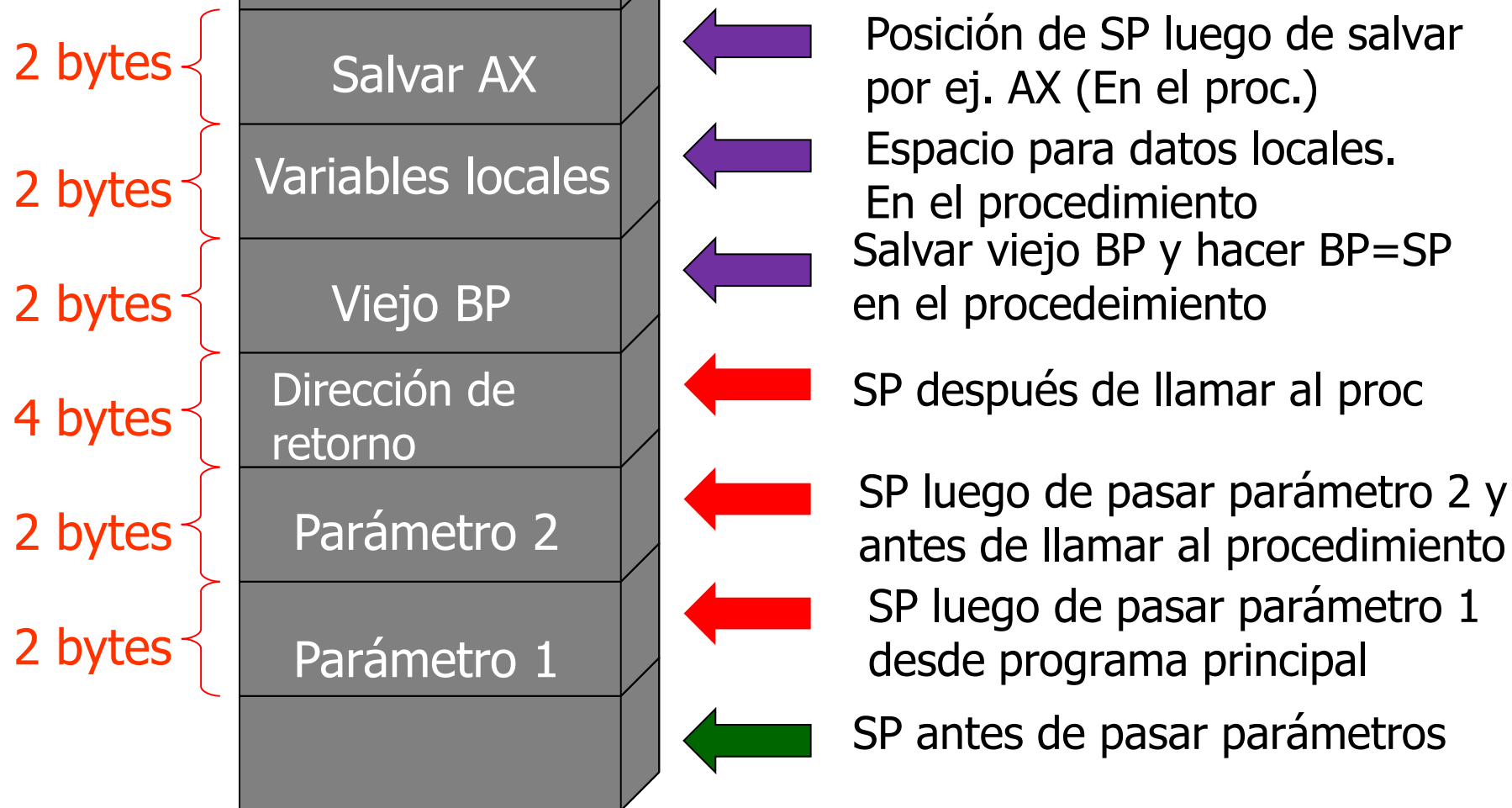
Pasos.....dentro del procedimiento

- Salvar otros registros
 - por ej. ax

push ax

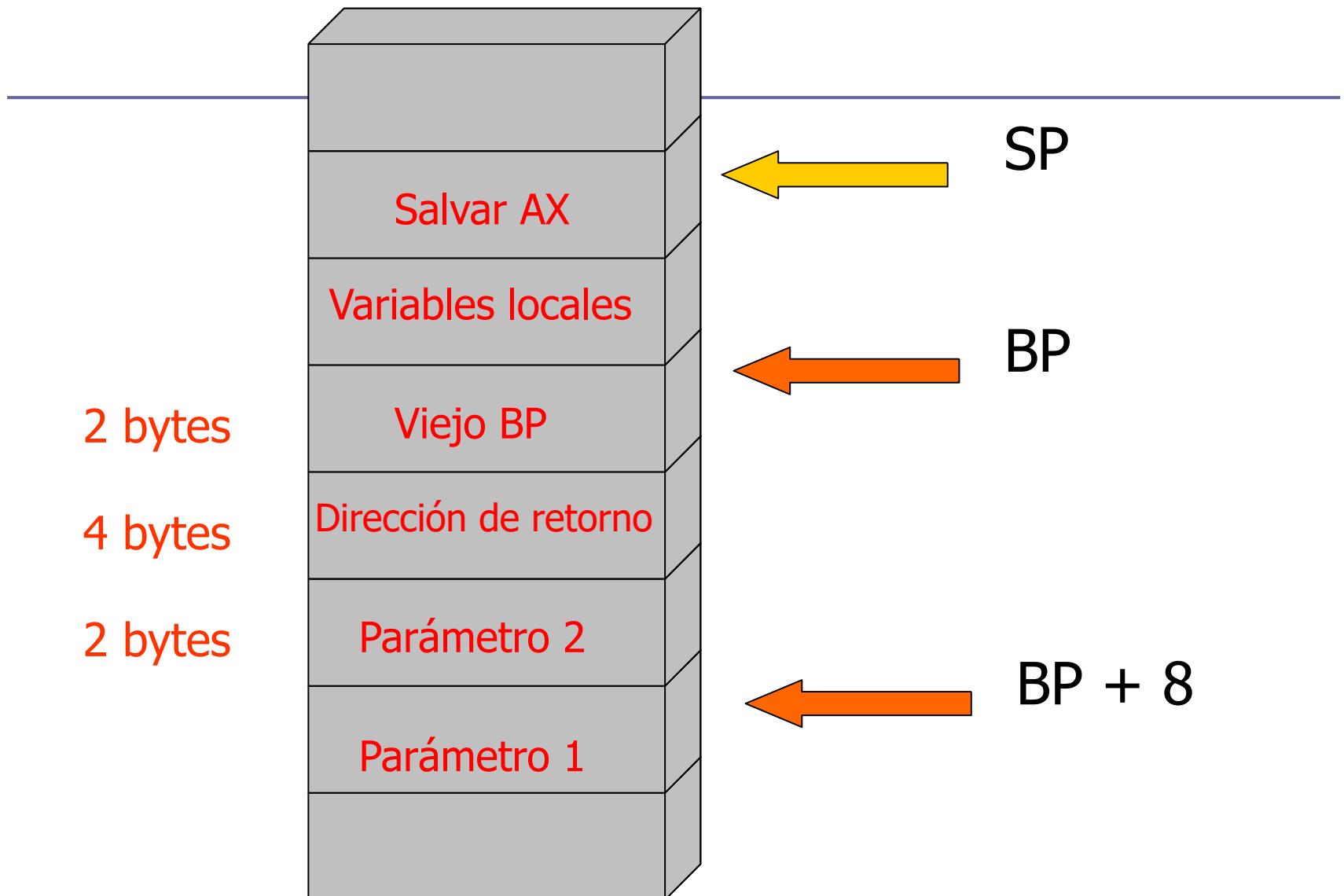
- Si el procedimiento no cambia el valor de los registros, éstos no necesitan ser salvados. Normalmente los registros son salvados después de establecer el puntero (frame pointer) y los datos locales.

-  = antes del programa principal
-  = en el programa principal
-  = en el procedimiento



Pasos.....dentro del procedimiento acceso a los parámetros

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



Salida del procedimiento

- 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

- En nuestro ej.

▪

▪

```
pop AX
add SP, 2
pop BP
ret
```

Anidamiento de subrutinas

Programa principal

.

.

Push Param1

Call Proce1

.

.

Proce1 Proc
Push BP1 😊
Push Param2
Call Proce2

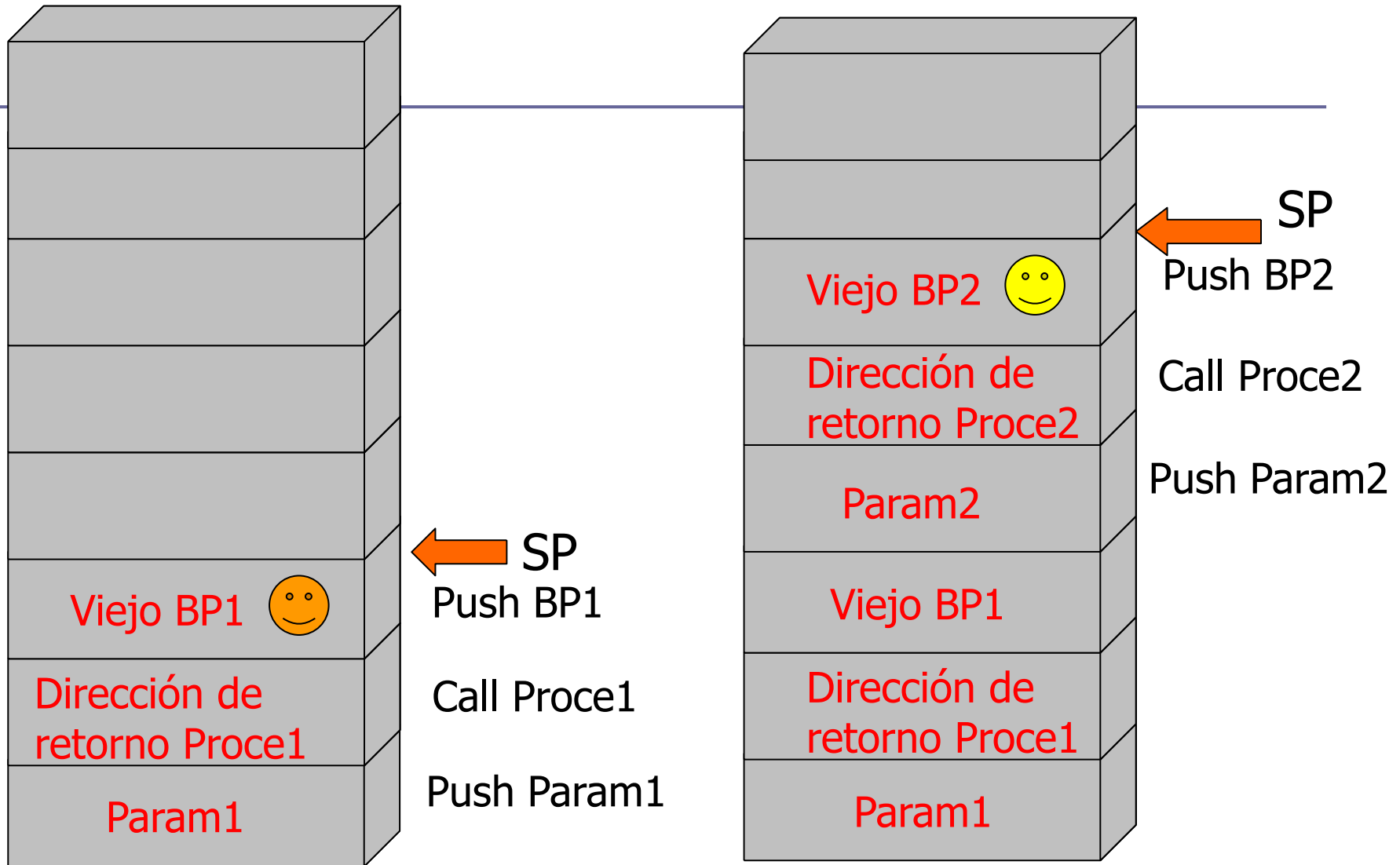
Pop Param2
Pop BP1
Ret

Proce1 Endp

Proce2 Proc
Push BP2 😊
Instrucciones
Pop BP2
Ret

Proce2 Endp

El simulador no tiene BP, hay que usar BX



Para el simulador

- Declaración del procedimiento

org 3000h

nombre: instrucción

.

.

ret

- En lugar de BP se usa BX
- Las direcciones son de 16 bits. La dirección de retorno salvada en la pila es de 2 bytes.

Para el simulador




- En lugar de `mov bp,sp` tenemos que hacer `mov bx,sp`
- En lugar de `mov cx, [bp + 8]` tenemos que hacer
- `add bx,8`
- `mov cx,[bx]`
- El `sp=8000h` arranca para cada programa

AX 63 56

DX 11 22

A ORG 1000H
DW 5234H
B DW 1122H

ORG 3000H
SUMA: ADD AX,DX 
RET

 ORG 2000H
 MOV AX, A
MOV DX, B
CALL SUMA 
(200B) HLT
END



7FF6H	00H
7FF7H	00H
7FF8H	00H
7FF9H	00H
7FFAH	00H
7FFBH	00H
7FFCH	00H
7FFDH	00H
7FFEH	0BH
7FFFH	20H
8000H	00H

AX 63 56

BX 10 02

DX 11 22

ORG 3000H
SUMA: MOV BX, AX
MOV AX, [BX]
MOV BX, DX
MOV DX, [BX]
ADD AX, DX
RET



ORG 2000H
MOV AX, OFFSET A
MOV DX, OFFSET B
CALL SUMA



(200B) HLT
END

ORG 1000H
A DW 5234H
B DW 1122H



7FF6H	00H
7FF7H	00H
7FF8H	00H
7FF9H	00H
7FFAH	00H
7FFBH	00H
7FFCH	00H
7FFDH	00H
7FFEH	0BH
7FFFH	20H
8000H	00H

AX 63 56

BX AB 89

DX 11 22

ORG 3000H

SUMA: PUSH BX
MOV BX, SP
ADD BX, 6
MOV AX, [BX]
SUB BX, 2
MOV DX, [BX]
ADD AX, DX
POP BX
RET



ORG 2000H

MOV AX, A
PUSH AX
MOV AX, B
PUSH AX
CALL SUMA

(200D) HLT
END



7FF6H	00H
7FF7H	00H
7FF8H	89H
7FF9H	ABH
7FFAH	0DH
7FFBH	20H
7FFCH	22H
7FFDH	11H
7FFEH	34H
7FFFH	52H
8000H	00H

