

# Régimen de cursada

Arquitectura de computadoras

# Turno 1

## Aula 3

# Aprobación de la cursada

- La cursada se divide en 2 partes con evaluaciones independientes
- MSX88 (parte 1, requiere 51% de asistencia):
  - El Parcial 1 evaluará los Trabajos Prácticos 1, 2 y 3.
  - Requiere 6 asistencias de 10
  - La primer fecha de Evaluación se toma al finalizar la Práctica 3.
  - 1 fecha de examen y 1 fecha de recuperatorio.
- Winmips (parte 2 , requiere 51% de asistencia):
  - El Parcial 2 evaluará los Trabajos Prácticos 4, 5 y 6.
  - Requiere 6 asistencias de 11 (51%)
  - La primer fecha de Evaluación se toma al finalizar la Práctica 6.
  - 1 fecha de examen y 1 fecha de recuperatorio.
- Fecha global de recuperatorio para rendir:
  - Solo la parte 1.
  - Solo la parte 2.
  - Ambas partes.

# Cronograma

## Cronograma de actividades - Año 2018 válido para todos los turnos

Exp P1	Dis P1	Res P1	Exp P2	Dis P2	Res P2	Exp P3	Dis P3	Res P3	Repaso	Parcial 1	Muestra P-1/ Exp1 P4
24-ago	28-ago	31-ago	04-set	07-set	14-set	18-set	25-set	28-set	02-oct	05-oct	09-oct

Exp2 P4	Recup 1 Parcial 1	Dis P4	Res P4	Exp P5	Dis P5	Res P5	Exp P6	Dist P6	Res P6	Repaso	Parcial 2
12-oct	16-oct	19-oct	23-oct	26-oct	30-oct	02-nov	06-nov	09-nov	13-nov	16-nov	23-nov

Muestra P-2 / Consulta	Recup 1 Parcial 2	Muestra / Consulta	Consulta	Recup 2 P-1 y P-2
27-nov	30-nov	04-dic	07-dic	11-dic

Evaluación Corta de Teoría
Miércoles 10 oct

Evaluación Teórica de Promoción
Miércoles 28 nov

**Exp:** explicación de práctica      **Dis:** distribución y discusión de resoluciones      **Res:** entrega de resultados

**Prácticas con MSX88:** **P1** – Subrutinas y Pasaje de Parámetros. **P2** – Interrupciones. **P3** – Entrada / Salida.  
**con WinMIPS64:** **P4** – Segmentación de cauce en RISC. **P5** – Pto Fte y pasaje.parám en RISC **P6** – E/S mapeada en memoria.

# Página de Arquitectura

- Toda la información y recursos necesarios de toda la cátedra esta disponible en:

<http://weblidi.info.unlp.edu.ar/catedras/arquitecturaP2003>

- Para acceder las diapositivas:

**<https://tinyurl.com/yc7k42m8>**

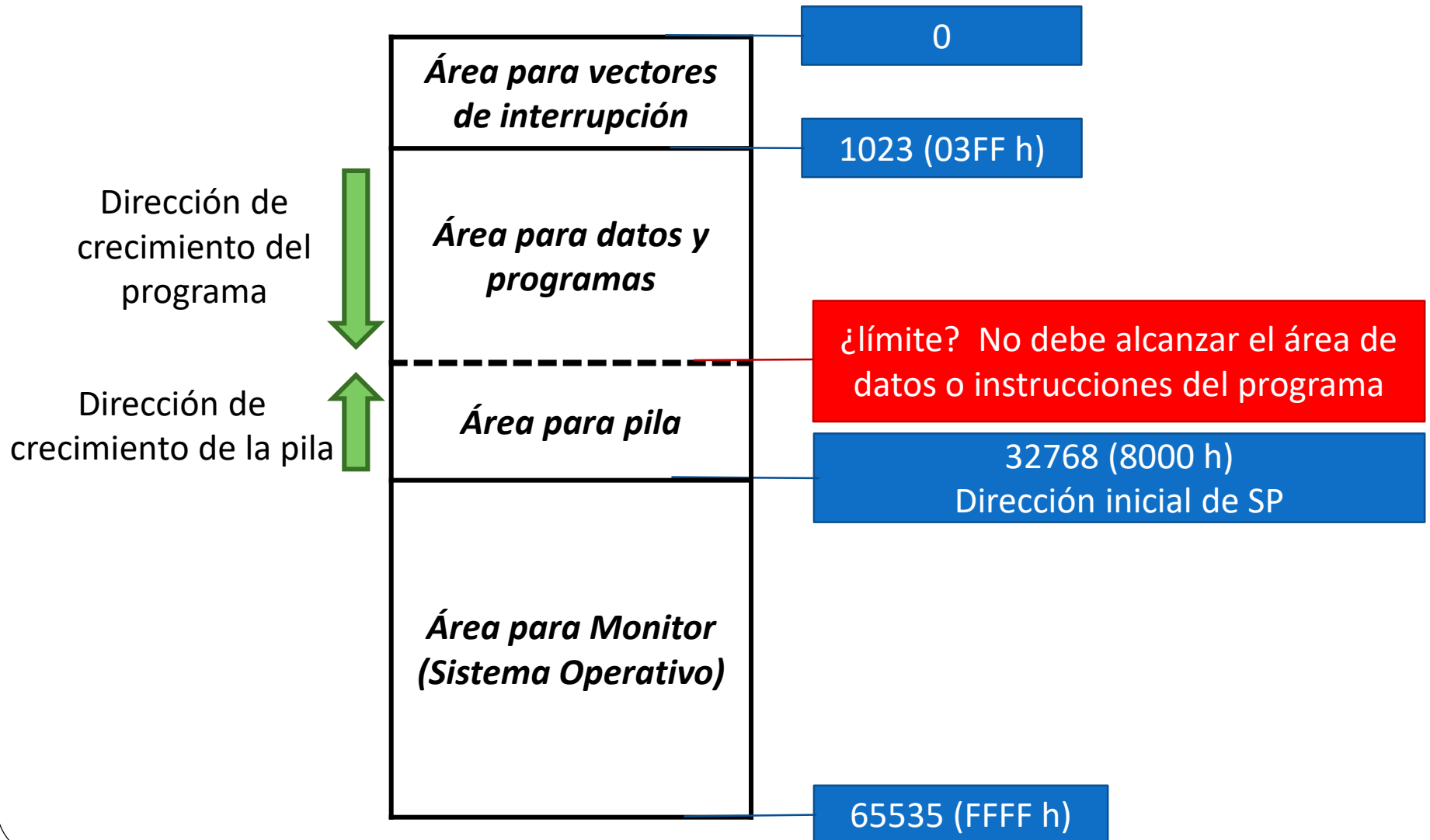
# Subrutinas y pasaje de parámetros

Arquitectura de computadoras

# Pila del MSX 88 - Características

- Cuando nos referimos a “pila”, nos referimos a un segmento de memoria donde se implementa una estructura LIFO (Last Input First Output), donde el último elemento en entrar es el primero en salir.
- La pila se implementa sobre la misma memoria que la del programa.
- Los elementos apilados tienen 16 bits (**ni mas, ni menos**).
- Cuenta con un registro de 16 bits para implementar la pila. Se denomina SP (Stack Pointer o Puntero de Pila) y apunta siempre al último elemento apilado.
- Tiene 2 operaciones: Push (Apilar) y Pop (Desapilar)

# Pila del MSX 88 - Memoria





# Pila del MSX 88 – Operaciones

- Operaciones de pila (pseudo-código):

- Push {registro 16 bits}:

- $SP \leftarrow SP - 2$
    - $[SP] \leftarrow \{\text{Reg. 16 bits}\}$

- Pop {registro de 16 bits}:

- $\{\text{Reg. 16 bits}\} \leftarrow [SP]$
    - $SP \leftarrow SP + 2$

- Ejemplos:

- PUSH AX, POP BX

- ~~• PUSH CL, POP MiVar~~

No se puede, no compila!!!!

# Direccionamiento indirecto

- Permite realizar operaciones utilizando un registro donde se encuentra la dirección del operando real. Es indirecto porque la dirección del operando no se encuentra en la instrucción.
- Permite trabajar con “punteros” a datos.
- Solo el registro BX puede utilizarse para direccionamiento indirecto.
- Ejemplo (AX = 10, BX = 1000, [1000] = 2):

• ADD AX, [BX] —————  $AX \leftarrow AX + [BX] \rightarrow AX = 10 + 2 = 12$

• ADD AX, BX —————  $AX \leftarrow AX + BX \rightarrow AX = 10 + 1000 = 1010$

# Direccionamiento indirecto

- La directiva OFFSET permite obtener la dirección de memoria de una variable

Dirección	Contenido	Variable
1000H	5	Num1 (baja)
1001H	0	Num1 (alta)

- Ejemplo:

- MOV AX, 2
  - MOV BX, OFFSET Num1
  - ADD [BX], AX
- AX  $\leftarrow$  2
- BX  $\leftarrow$  1000H
- [BX]  $\leftarrow$  [BX] + AX  $\rightarrow$  [BX] = 5+2

¿Que diferencia hay con ADD Num1, AX ?

Hacen lo mismo, pero [BX] no depende de ninguna variable

# Subrutinas - Ventajas

- Evitan la repetición de código.
- Facilitan la reutilización de código.
- Permiten modularizar las soluciones de los programas.
- Facilitan la lectura y comprensión del programa porque ocultan los detalles de implementación.
- Limitan la posibilidad de cometer errores.
- Permite independencia de variables si se utilizan parámetros.

# Subrutinas - Instrucciones

- Como invocar a una subrutina (pseudo-código):

- CALL {Dir. Subrutina}:

- $SP \leftarrow SP - 2$

- $[SP] \leftarrow \{\text{Dir. Retorno} = \text{instr. Sig. al CALL}\}$

- $IP \leftarrow \{\text{Dir. Subrutina}\}$

**Pone en la pila  
la dirección  
de retorno  
Asigna dir. rutina a IP**

- Como regresar de una subrutina (pseudo-código):

- RET:

- $IP \leftarrow [SP]$

- $SP \leftarrow SP + 2$

**Asigna dirección de retorno a IP  
Desapila**

# Subrutinas - Ejemplo

- Ejemplo de subrutina para sumar:

```
ORG 4000H
```

```
MI_SUMA:    ADD BX, AX
```

```
    RET
```

```
OTRA_RUTINA: . . . . .
```

```
MEM  ORG 2000H ; prog principal
```

```
2000h MOV AX, 5
```

```
2003h MOV BX, 10
```

```
2006h CALL MI_SUMA
```

```
2009h ... ; esta es la dirección de retorno
```

Al ejecutar CALL se asigna:

IP = 4000h

[SP] = 2009h



# Subrutinas - Parámetros

- Los parámetros nos permiten intercambiar datos con subrutinas.
- Ofrecen independencia entre las variables del programa y las subrutinas. En otras palabras las subrutinas no quedan ligadas a variables del programa.
- Existen dos clasificaciones (superpuestas) del pasaje de parámetros:
  - Por valor y por referencia.
  - Por registro o por pila.

# Subrutinas - Parámetros

- Por Registro y Valor:
  - Tamaño: 8 o 16 bits
  - Pasaje: cargar registros, llamar subrutina.
  - Uso: usar directamente desde la subrutina los registros para acceder a los valores.
  - Ejemplo:

MOV AL, MiVar1

CALL Subrutina

...

Subrutina: ADD AL, 1

...

RET



# Subrutinas - Parámetros

- Por Registro y Referencia:
  - Tamaño: solo 16 bits (tamaño de una dirección de memoria)
  - Pasaje: cargar registros usando OFFSET, llamar subrutina.
  - Uso: desde la subrutina direccionar indirectamente con BX para acceder a los valores.
  - Ejemplo:

```
MOV AX, OFFSET MiVar1
CALL Subrutina
...
Subrutina: MOV BX, AX
OR [BX], CX; cambia la celda apuntada por BX
...
RET
```

# Subrutinas - Parámetros

- Por Pila y Valor
  - Tamaño: siempre 16 bits
  - Pasaje: cargar registros, apilar, llamar a subrutina, desapilar registros.
  - Uso: recuperar los valores de la pila con BX, para acceder a los valores
- Importante: hay que tener cuidado al manipular la pila desde la subrutina. Si no se pone atención en el orden en que se apilan los parámetros, la dirección de retorno de la subrutina podría perderse.

# Subrutinas - Parámetros

- Ejemplo de parámetro por Pila y Valor

MOV AX, MiVar

PUSH AX ; solo 1 parámetro

CALL SubSuma2

POP AX

...

ORG 3000H

SubSuma2: MOV BX, SP ; recupera dirección de pila (7FFCH)

ADD BX, 2 ; para acceder al parámetro (7FFEh)

ADD AX, [BX]; accede al valor de MiVar

...

RET

7FFCH	dir. Retorno
7FFEh	Valor de MiVar
8000H	--

Valores SP

# Subrutinas - Parámetros

- Por Pila y Referencia:
  - Tamaño: siempre 16 bits
  - Pasaje: cargar registros usando OFFSET, apilar, llamar a subrutina, desapilar registros.
  - Uso: recuperar los valores de la pila con BX, y luego volver a usar BX indirectamente para acceder al dato. **Recordar que lo que hay en la pila es una referencia al valor y no el valor del parámetro. Hay 2 niveles de indirección: uno para acceder al valor de la pila y otro adicional para acceder al valor del parámetro**

# Subrutinas - Parámetros

- Ejemplo de parámetro por Pila y Referencia

MOV AX, **OFFSET** MiVar

PUSH AX ; solo 1 parámetro

CALL Subrutina

POP AX

...

ORG 3000H

Subrutina: MOV BX, SP ; recupera dirección de pila (7FFCH)

ADD BX, 2 ; para acceder al parámetro (7FFEC)

**MOV BX, [BX]**; recupera dir de MiVar

ADD AX, [BX]; utiliza valor de MiVar

...

RET

7FFCH	dir. Retorno
7FFEh	Dir. de MiVar
8000H	--

# Parámetros – Ejemplos de práctica

Escribir un programa que calcule el producto entre dos números sin signo almacenados en la memoria del microprocesador:

- 1.2) Llamando a una subrutina MUL para efectuar la operación, pasando los parámetros por valor desde el programa principal a través de registros
- 1.3) Llamando a una subrutina MUL, pasando los parámetros por referencia desde el programa principal a través de registros.

# Parámetros – Ejemplo 1.2

```
1.  ORG 1000H ; Datos
2.  NUM1 DB 5H
3.  NUM2 DB 3H
```

```
4.  ; Instrucciones
5.  ; Subrutina MUL
6.  ; entrada AL y CL
7.  ; resultado en DX
```

```
8.  ORG 3000H
9.  MUL: MOV DX, 0
10.      CMP AL, 0
11.      JZ FIN
12.      CMP CL, 0
13.      JZ FIN
14.      MOV AH, 0
15. LAZO: ADD DX, AX
16.      DEC CL
17.      JNZ LAZO
18. FIN: RET
```

```
19. ORG 2000H ; Programa
20. MOV AL, NUM1
21. MOV CL, NUM2
22. CALL MUL
23. HLT
24. END
```

- La multiplicación se resuelve haciendo sumas.
- Se usa 16 bits (DX) porque el resultado puede no entrar en 8 bits (ej:  $255 \times 255 = 65535$ ).
- Líneas 10 a 13 están por si AL o CL son cero (el resultado es 0. La línea 9 DX asigna ese valor).
- Línea 14 es para que AX tenga un valor válido en 16 bits. Como no hay instrucciones que sumen registros de 16 bits con 8 bits.
- Para pensar: ¿que pasaría si las líneas 11 y 13 se reemplazan por RET?

# Parámetros – Ejemplo 1.3

```
1.  ; Mem. de datos
2.  ORG 1000H
3.  NUM1 DW 5H
4.  NUM2 DW 3H
5.  ; NUM1 y NUM2 > 0
6.  ORG 3000H ; Subrutina MUL
7.  MUL : MOV DX, 0          ; resultado de 16 bits en BX
8.  LAZO: MOV BX, AX         ; recupera direccion de NUM1 (1000H)
9.      ADD DX, [BX]        ; recupera valor de NUM1 (5H) y suma
10.     PUSH DX             a DX
11.     MOV BX, CX          ; guarda resultado parcial en pila
12.     MOV DX, [BX]        ; recupera direccion de NUM2 (1002H)
13.     DEC DX              ; recupera valor de NUM2 (3H, 2H, 1H)
14.     MOV [BX], DX        ; decrementa valor recuperado de NUM2
15.     POP DX              ; asigna NUM2 (queda modificado en
16.     JNZ LAZO            memoria)
17.     RET                ; recupera resultado desde la pila
                             ; flag Z=0 vuelve a L8. Nota Z se
                             ; modifico en L13. L14 y L15 no
```

```
18. ORG 2000H ; Programa
19. MOV AX, OFFSET NUM1
20. MOV CX, OFFSET NUM2
21. CALL MUL
22. HLT
23. END
```



# Parámetros – Ejemplos de práctica

## Ejercicio 2:

Escribir un programa que calcule el producto entre dos números sin signo almacenados en la memoria del microprocesador llamando a una subrutina MUL, pero en este caso pasando los parámetros por valor y por referencia a través de la pila.

1. ORG 1000H; Mem. de datos  
2. NUM1 DW 5H  
3. NUM2 DW 3H  
4. RES DW ?

5. ORG 3000H ; Subrutina MUL  
6. MUL: PUSH BX  
7.       MOV BX, SP  
8.       PUSH CX  
9.       PUSH AX  
10.      PUSH DX  
11.      ADD BX, 6  
12.      MOV CX, [BX]  
13.      ADD BX, 2  
14.      MOV AX, [BX]  
15. SUMA: ADD DX, AX  
16.      DEC CX  
17.      JNZ SUMA  
18.      SUB BX, 4  
19.      MOV AX, [BX]  
20.      MOV BX, AX

21. MOV [BX], DX  
22. POP DX  
23. POP AX  
24. POP CX  
25. POP BX  
26. RET

27. ORG 2000H ; Programa  
28. MOV AX, NUM1  
29. PUSH AX  
30. MOV AX, NUM2  
31. PUSH AX  
32. MOV AX, OFFSET RES  
33. PUSH AX  
34. MOV DX, 0  
35. CALL MUL  
36. POP AX  
37. POP AX  
38. POP AX  
39. HLT  
40. END

```

1.  ORG 1000H; Mem. de datos
2.  NUM1 DW 5H           ; comienza en 1000H y termina en 1001H
3.  NUM2 DW 3H           ; comienza en 1002H y termina en 1003H
4.  RES DW ?             ; comienza en 1004H y termina en 1005H

```

...

```

27. ORG 2000H ; Programa
28. MOV AX, NUM1           ; asigna valor de NUM1 (5H) a AX
29. PUSH AX                ; apila en dir 7FFEh
30. MOV AX, NUM2           ; asigna valor de NUM2 (3H) a AX
31. PUSH AX                ; apila en dir 7FFCh
32. MOV AX, OFFSET RES     ; recupera direccion de RES (1004H)
33. PUSH AX                ; apila en dir 7FFAh
34. MOV DX, 0              ; resultado inicial de multiplicacion en DX
35. CALL MUL               ; llama rutina, apila dir. retorno en 7FF8H
36. POP AX                 ; desapila la misma cantidad que apilo
37. POP AX
38. POP AX
39. HLT
40. END

```

Estado de pila antes  
de llamado a  
subrutina

7FFAh	Dir RES (1004H)
7FFCh	NUM2 (0003H)
7FFEh	NUM1 (0005H)
8000H	--

5.	ORG 3000H ; Subrutina MUL	
6.	MUL: PUSH BX	; apila BX en 7FF6H, preserva valor anterior
7.	MOV BX, SP	; asigna SP (7FF6H) en BX (BX apunta a pila)
8.	PUSH CX	; apila CX, AX, DX para preservar valores de quien llama a la subrutina. Es una buena practica de programacion. SP queda en 7FF0H
9.	PUSH AX	
10.	PUSH DX	
11.	ADD BX, 6	; desplaza 3 lugares en pila BX=7FFCH (NUM2)
12.	MOV CX, [BX]	; CX queda con el valor de NUM2
13.	ADD BX, 2	; desplaza 1 lugar en pila BX=7FFEh (NUM1)
14.	MOV AX, [BX]	; AX queda con el valor de NUM1
15.	SUMA: ADD DX, AX	; acumula valor de NUM1 en el resultado
16.	DEC CX	; decrementa veces restantes a sumar (NUM2)
17.	JNZ SUMA	; salta si CX (valor de NUM2) no llego a 0
18.	SUB BX, 4	; desplaza 2 lugares en pila BX=7FFA(dir RES)
19.	MOV AX, [BX]	; asigna dir. de RES: [BX] ⇔ [7FFAH] ⇔ 1004H
20.	MOV BX, AX	; asigna dir de RES a BX (1004H)
21.	MOV [BX], DX	; guarda resultado. [BX] ⇔ [1004H] ⇔ DX ⇔ 15
22.	POP DX	
23.	POP AX	
24.	POP CX	
25.	POP BX	
26.	RET	

Estado de pila luego de  
linea 10

Estado de pila al ingresar a la  
subrutina

7FF0H	Valor de DX
7FF2H	Valor de AX
7FF4H	Valor de CX
7FF6H	Valor de BX
7FF8H	Dir Retorno (L36)
7FFAH	Dir RES (1004H)
7FFCH	NUM2 (0003H)
7FFEh	NUM1 (0005H)
8000H	--

# Ejercicios de práctica – Suma 32 bits

Sumar dos números de 32 bits:

- MSX88 suma 8 o 16 bits
- Hay que agrupar 2 bloques de 16 bits para formar un número de 32 bits:

- NUM1\_L DW 1234H
- NUM1\_H DW 0000H

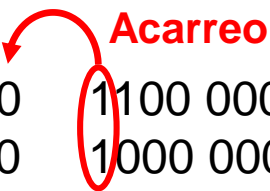
**Dos partes. Puede ponerse la alta o la baja primero, es lo mismo. Debe tenerse en cuenta esto al momento de realizar la suma.**

**Notar que los 2 valores quedan consecutivos. Un puntero al primero permite acceder al segundo**

# Ejercicios de práctica – Suma 32 bits

Sumar dos números de 32 bits:

- Para obtener el resultado deben sumarse 16 bits de los 2 operandos y luego los 16 bits restantes.
- Hay que tener en cuenta que a los 16 bits más significativos hay que sumarle el acarreo de los 16 menos significativos.



	0000 0000 0000 0000	1100 0000 0000 0000	⇔	49152
+	0000 0000 0000 0000	1000 0000 0000 0000	⇔	32768
<hr/>				
	0000 0000 0000 0001	0100 0000 0000 0000	⇔	81920

# Ejercicios de práctica - Suma 32 bits

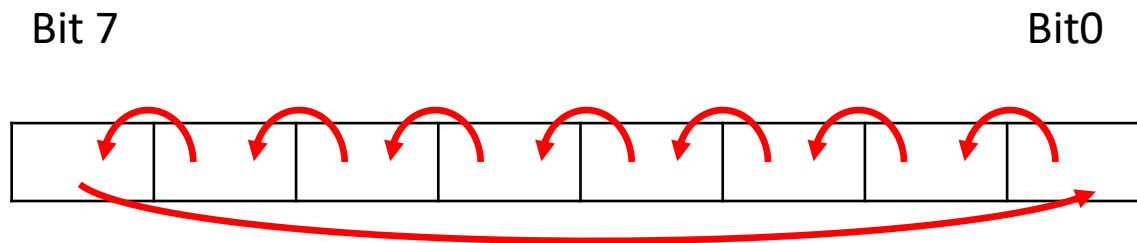
Sumar dos números de 32 bits:

- ¿Cómo hacer el acarreo? Dos maneras:
  - Opción 1:
    - Sumar los 16 bits altos
    - Sumar los 16 bits bajos (si hay acarreo, flag C=1)
    - Validar la condición C=1 e incrementar los 16 bits altos
  - Opción 2:
    - Sumar los 16 bits bajos
    - Sumar los 16 bits altos y sumar el carry: hay que usar la instrucción ADC (**A**dd with **C**arry).
    - Esta opción puede complicarse si la suma de la parte baja queda separada por varias instrucciones de la suma de la parte alta. En este caso se usa PUSHF (apila los flags, incluido C) para salvar el acarreo y POPF para recuperarlo, antes de hacer ADC

# Ejercicios de práctica - Rotación

Rotar a izquierda un byte:

- Rotar a la izquierda significa que todos los bits se desplazan a izquierda y que el más significativo se copie en el bit menos significativo:



- Ejemplos:
  - 10001000 → 00010001
  - 00011100 → 00111000



# Ejercicios de práctica - Rotación

Rotar a izquierda un byte:

- MSX88 no tiene instrucciones para rotar y ni desplazar
- ¿Cómo hacemos? Analicemos que significa desplazar a izquierda varias veces en binario y decimal:
  - 00000011 → 00000110 → 00001100 → 00011000 → 00110000
  - 3 → 6 → 12 → 24 → 48
- En cada desplazamiento el número se duplica, es decir multiplica por 2 (pero MSX88 no tiene instrucciones de multiplicación):
- $2 * A$  puede escribirse como una suma:  $A + A$
- Se utiliza la instrucción ADD. Ej: ADD AL, AL

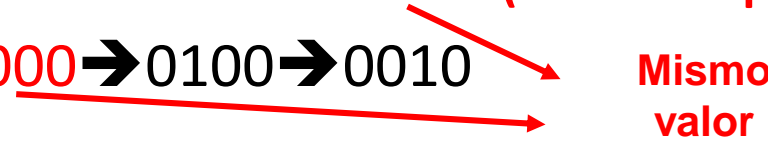
# Ejercicios de práctica - Rotación

Rotar a izquierda un byte:

- Es importante no perder el bit más significativo
- Al desplazar a izquierda el bit mas significativo queda en el acarreo
- ¿Como se ubica el bit más significativo en el menos significativo? Hay 2 maneras:
  - Puede usarse la ADC con 0: sumar al valor desplazado el valor cero y el del carry.
  - Puede utilizarse instrucción de salto verificando C=1 y ajustarlo solo en este caso. El ajuste se puede hacer con una instrucción OR con 1 o ADD con 1

# Ejercicios de práctica - Rotación

Rotar a derecha un byte:

- Para rotar a derecha se puede utilizar la rotación a izquierda.
- Ejemplo para 4 bits:
  - Rotar a izquierda:  $0001 \rightarrow 0010 \rightarrow 0100 \rightarrow 1000$  (3 = 4-1 despl.)
  - Rotar a derecha :  $0001 \rightarrow 1000 \rightarrow 0100 \rightarrow 0010$   **Mismo valor**
- Dados n (4, 8, 16, 32, etc.) bits de un valor a rotar:
  - rotar 1 lugar a derecha es equivalente a rotar n-1 lugares a izquierda

# Ejercicios de práctica - Cadena

Escribir subrutina que cuente los caracteres de una cadena terminada en cero (00H)

- Terminada en cero significa que después del último carácter legible hay un cero. Cero significa código ASCII 0 y no "0" que tiene el código ASCII 48 decimal o 30H
- Cuando hablamos de cadena, el pasaje de parámetros se realiza por referencia, pasando el puntero al primer carácter.
- El recorrido implica avanzar hasta llegar al cero.
- La declaración:
  - NUM DB "3210", 0 es una secuencia de 5 bytes con los valores hexadecimales 33h, 32h, 31h, 30h, 00h

# Ejercicios de práctica - Cadena

Escribir subrutina que determine si un carácter es vocal

- Hay dos alternativas de solución:
  - La simple, fácil, poco escalable y poco digna de un buen programador: comparar con instrucciones vocal por vocal con el carácter a probar:

```
CMP AL, "A"
```

```
JZ DEVOLVER_VERDADERO
```

```
CMP AL, "a"
```

```
JZ DEVOLVER_VERDADERO
```

```
CMP AL, "E"
```

```
JZ DEVOLVER_VERDADERO
```

```
...
```

# Ejercicios de práctica - Cadena

Escribir subrutina que determine si un carácter es vocal

- La no tan simple, no tan fácil, escalable y digna de un buen programador: usar un “arreglo” de vocales para comparar con el carácter a probar

```
VOCAL DB "AaEeliOoUu"
```

```
MOV BX, OFFSET VOCAL
```

```
MOV CL, 10 ; cantidad en el arreglo también se puede usar 00h al final
```

```
...
```

```
VOLVER: CMP AL, [BX]
```

```
JZ DEVOLVER_VERDADERO
```

```
INC BX ; apunta a próxima vocal
```

```
DEC CL; actualiza cantidad vocales restantes
```

```
JNZ VOLVER ; quedan vocales para testear?
```

```
...
```

# Ejercicios de práctica - Cadena

Escribir subrutina que determine si un carácter es vocal

- Cuando hablamos de escalabilidad en este ejemplo, hacemos referencia en la dificultad de aplicar la solución a problemas más grandes.
- Reflexionar en como debería cambiar ambas implementaciones si se requiere una subrutina que determine si un carácter es consonante:
  - Para la primera aproximación de solución hay que escribir mucho código
  - Para la segunda aproximación, solo hay que actualizar el “arreglo” y la cantidad de elementos que posee (en caso de no haber usado la terminación ASCII cero)