# MÓDULO ASSEMBLER

## Repaso de conceptos Assembler

Autores:
Silvana Lis Gallo
Alejandro Héctor Gonzalez
Junio 2021

Este trabajo tiene licencia CC BY-NC 2.5 AR

# RESUMEN

En esta clase se repasan los conceptos de programación en bajo nivel, assembler del 8008, se revisa el set de instrucciones y la forma de declaración de variables. Se presentan ejercicios de repaso de pasajes de parámetros por valor y por referencia usando registros y pila.

# PALABRAS CLAVE

programación en bajo nivel, assembler del 8008, pasaje de parámetros, declaración de variables, tipos de datos, registros, modos de direccionamiento

# El Simulador

Registros **AX, BX, CX y DX**: uso general, 16 bits de longitud, se pueden dividir en 2 partes de 8 bits cada uno. Ejemplo: AX en AH y AL.

Registro **IP** (Instrucción Pointer) contiene la dirección de memoria de la próxima instrucción a ser ejecutada.

Registro **SP** (Stack Pointer) contiene la dirección de memoria del tope de la pila.

Registro de flags: muestra el estado de las banderas o flags luego de cada operación.

- Bandera de cero: identificada por la letra **Z**.
- Bandera de overflow: identificada por la letra **O.**
- Bandera de carry/borrow: identificada por la letra **C**.
- Bandera de signo del número: identificada por la letra **S**.

# Variables

nombre_variable especificador_tipo valor_inicial

| Especificador | Tipo | Tamaño |
|---|---|---|
| DB | Byte | 8 bits |
| DW | Word | 16 bits |

**Var1 DB 10**
**Var2 DW 0A000h**

Podemos observar que los valores numéricos se interpretan en decimal, a menos que terminen con una letra ‘h’, que en cuyo caso se interpretarán como valores en hexadecimal. Además, como los números deben comenzar con un dígito decimal, en el caso del A000h, se **antepone un cero para evitar que se la confunda con una variable que se pueda llamar A000h.**

# Modos de Direccionamiento

| | |
|---|---|
| **INMEDIATO**<br>**MOV AX, 1000h** | El operando contiene la información sobre la que hay que operar. (útil para inicializar registros) |
| **DIRECTO DE MEMORIA O ABSOLUTO**<br>**MOV BL, var_byte** | La instrucción contiene la dirección de memoria exacta donde se encuentra el operando. El operando se encuentra en memoria. |
| **DIRECTO DE REGISTRO**<br>**MOV BX, AX** | El operando se encuentra contenido en un registro |
| **INDIRECTO CON REGISTRO**<br>**MOV AX, [BX]** | La instrucción contiene una dirección que se emplea para leer en memoria una dirección intermedia que será la verdadera dirección del objeto buscado. El operando se encuentra en memoria. |
| **INDIRECTO CON DESPLAZAMIENTO**<br>**MOV AX, 20h+[BX]** | Similar al anterior al que se le agrega un desplazamiento para obtener la dirección final donde se encuentra el operando. |

# OFFSET

```
ORG 1000h
var_otro DW  0ABCDh
ORG 2000h
MOV BX, OFFSET  var_otro
HLT
END
```

Se mueve a BX OFFSET var_otro . Esto indica que se debe cargar en BX la dirección de var_otro **y NO** el contenido de dicha variable.

Solamente uso OFFSET para obtener direcciones de memoria de variables declaradas con anterioridad.

# Instrucción de Comparación

**CMP** es esencialmente equivalente en funcionamiento a **SUB** pero el resultado de la resta no se almacena en ninguna parte, por lo que ninguno de los operandos se ve. **Si se modifican los flags los cuales puedo usar para saber cosas.**

flag Z= 1 (el resultado de la resta fue cero) -> Numeros iguales

flag S= 1 (signo)  con operandos en CA2 -> El segundo operando mayor que el primero

# Instrucciones de Salto

Las instrucciones de **salto** permiten realizar saltos alterando el flujo de control a lo largo de la ejecución de un programa. Estas instrucciones tienen un operando que indica la dirección que se le asignará al registro IP.

Existen dos tipos:

- **Saltos incondicionales:** Se producen siempre que se ejecuta la instrucción
- **Saltos condicionales:** Dependen de alguna condición para que se produzca ,o no, dicho salto.

**Condicionales**

**JMP** dirección ; Salta siempre (**Incondicional**)
**JZ** dirección ; Salta si el flag Z=1
**JNZ** dirección ; Salta si el flag Z=0
**JS** dirección ; Salta si el flag S=1
**JNS** dirección ; Salta si el flag S=0
**JC** dirección ; Salta si el flag C=1
**JNC** dirección ; Salta si el flag C=0
**JO** dirección ; Salta si el flag O=1
**JNO** dirección ; Salta si el flag O=0

Z= Cero
O= Overflow
C= Carry/Borrow
S= Signo

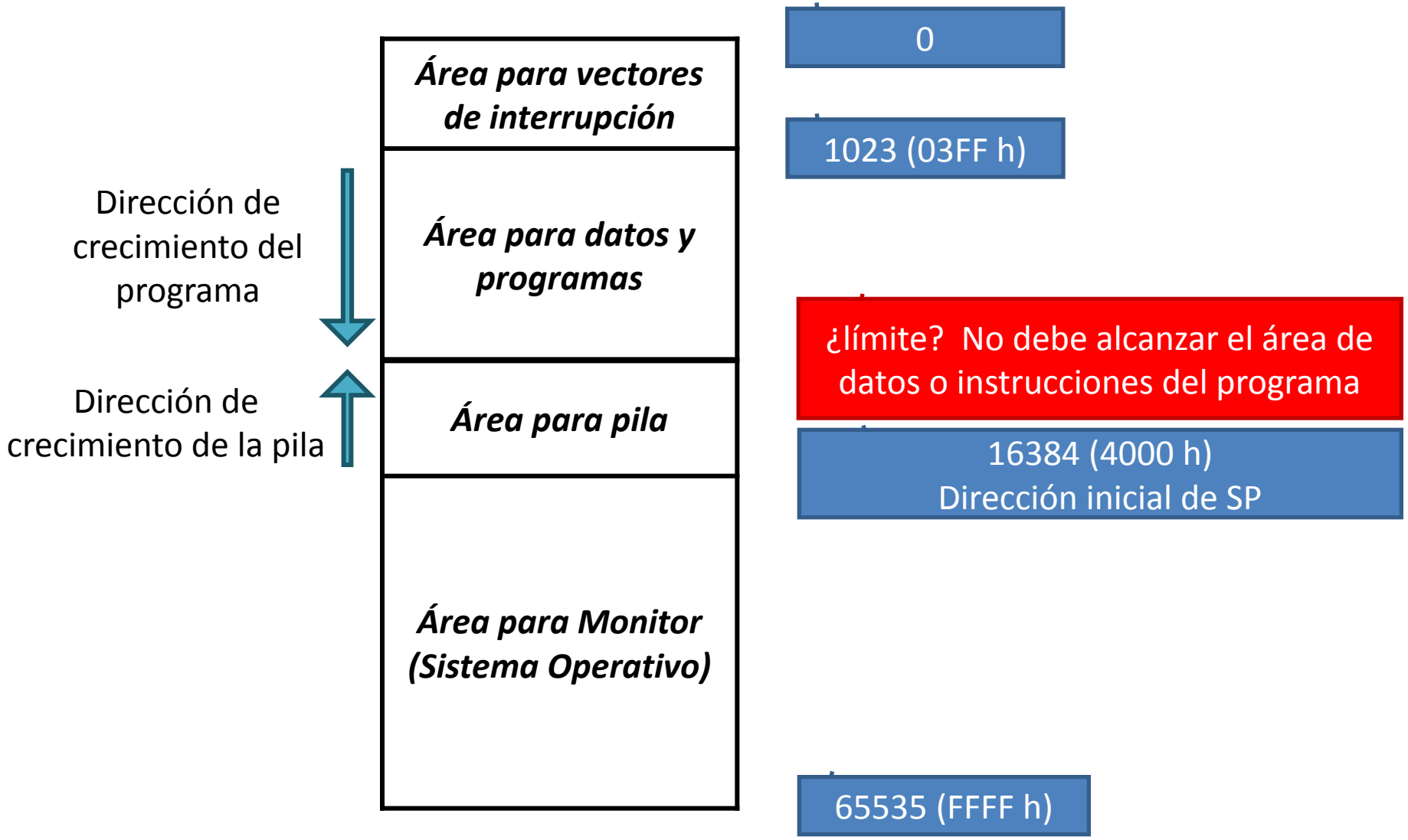
# Programación II

## Subrutinas y Parámetros

# Pila del VONSIM- 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 - Memoria



# Pila – Operaciones

- Operaciones de pila (seudo-código):
  - Push {registro 16 bits}:
    - SP □ SP – 2
    - [SP] □ {Reg. 16 bits}
  - Pop {registro de 16 bits}:
    - {Reg. 16 bits} □ [SP]
    - SP □ SP + 2
- Ejemplos:
  - PUSH AX,  POP BX
  - PUSH CL,  POP MiVar

No se puede, no compila!!!!

# 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.
- Limita la posibilidad de cometer errores.
- Permite independencia de variables si se utilizan parámetros.

# Subrutinas - Instrucciones

- Como invocar a una subrutina (seudo-código):
  - CALL {Dir. Subrutina}:
    - SP □ SP – 2
    - [SP] □ {Dir. Retorno = instr. Sig. al CALL}
    - IP □ {Dir. Subrutina}

**Pone en la pila la dirección de retorno**

**Asigna dir. rutina a IP**

- Como regresar de una subrutina (seudo-código):
  - RET:
    - IP □ [SP]
    - SP □ SP + 2

**Asigna dirección de retorno a IP**

**Desapila**

# 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.

| | Valor | Referencia |
|---|---|---|
| Registro | X | X |
| Pila | X | X |

# 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 (3FFCH)
           ADD BX, 2   ; para acceder al parámetro (3FFEH)
           ADD AX, [BX]; accede al valor de MiVar
           …
           RET
```

| | |
|---|---|
| **3FFCH** | **dir. Retorno** |
| **3FFEH** | **Valor de MiVar** |
| **4000H** | **--** |

**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 un 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 (3FFCH)
             ADD BX, 2   ; para acceder al parámetro (3FFEC)
             MOV BX, [BX]; recupera dir de MiVar
             ADD AX, [BX];  utiliza valor de MiVar
             …
             RET
```

| | |
|---|---|
| **3FFCH** | **dir. Retorno** |
| **3FFEH** | **Dir. de MiVar** |
| **4000H** | **--** |

# Subrutinas - Reutilización

- Para poder reutilizar nuestras rutinas en diferentes programas necesitaremos que **NO** alteren el contexto más allá de sus parámetros.

- La pila debe quedar en el mismo estado previo al **CALL** de la misma.

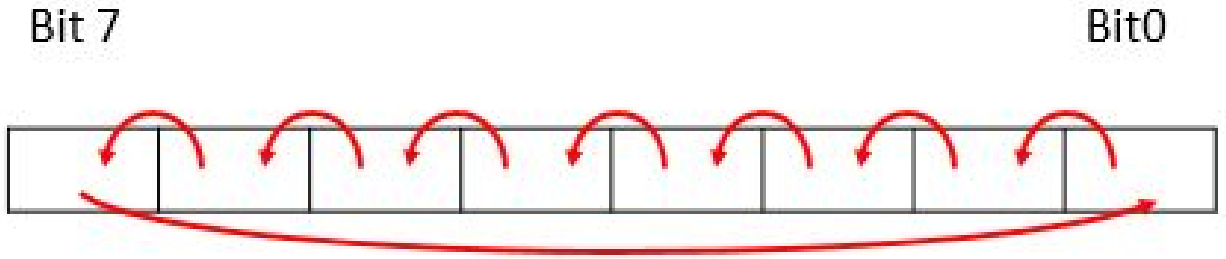
- Los registros del procesador deben conservar los valores previos al llamado de la subrutina.

- Solo se podrán ver modificados los valores de retorno de la subrutina (Parámetros por referencia o retorno en registro).

# Subrutinas - Ejercicio

Escriba una subrutina **reutilizable** para rotar hacia la izquierda un byte:

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




- Ejemplos:

10001000 □ 00010001 00011100 □ 00111000

- La rutina debe recibir:
  - El byte a rotar por referencia vía pila.
  - La cantidad de rotaciones a realizar por valor vía registro.

# Subrutinas - Ejercicio (cont.)

Escriba un programa que **usando la subrutina** rote a la izquierda X veces los elementos de la tabla:

- ¿Qué tipo de datos tiene la tabla? ¿y el valor X?
- ¿Cómo recorremos la tabla?

```
ORG 1000H ; DATOS
TABLA DB 1,2,3,4,5,6
X DB 2

ORG 3000H ; SUBRUTINA
ROTAR:
ADD  ,  ; SUMA PARA MULTIPLICAR X 2
ADC ,0 ; SUMA PARA EL POSIBLE CARRY
FINR: RET

ORG 2000H ; PROGRAMA
CALL ROTAR
FIN: HLT
END
```

# Subrutinas - Ejercicio (cont.)

Subrutina

1) Pasaje de parámetros
2) ¿Qué hace?
3) Vuelta al programa // Reutilización del módulo

Programa

1) Envío de parámetros
2) ¿Qué hace? // ¿Iteración?
3) Llamado a la subrutina
4) Adicionales