MANUAL DE ENSAMBLADOR

Contenido

Introducción a ASM

***Lenguaje ensamblador* (**o código de ensamblaje**)** : lenguaje de programación de *bajo nivel* que se basa en una notación simbólica basada en la representación de operaciones mediante mnemónicos (pequeña secuencia de letras que representa el concepto).

Ensamblador : Programa traductor que se encarga de traducir los programas escritos en código de ensamblaje a lenguaje máquina (código binario).

Ensamblar : Cuando las instrucciones de ensamblador se traducen a lenguaje binario.

DOS BOX : máquina virtual para correr programas .asm

Interrupción (comunicación con el sistema operativo): la puesta que se usa para comunicarme con los sistemas operativos, para usar archivos y otros recursos que tiene la computadora.

Pila: almacena datos, apilando datos, lo último que entra es lo primero que sale. (ej: pila de platos) LIFO last in fist out

¿Por qué usar lenguaje ensamblador para programar?

Eficiencia: La gente programa en ensamblador para ser más eficiente. Objetivo: que sea rápido y gaste menos memoria.

Herramientas para programar en ASM

TASM.exe (este es el ensamblador) V 3.2

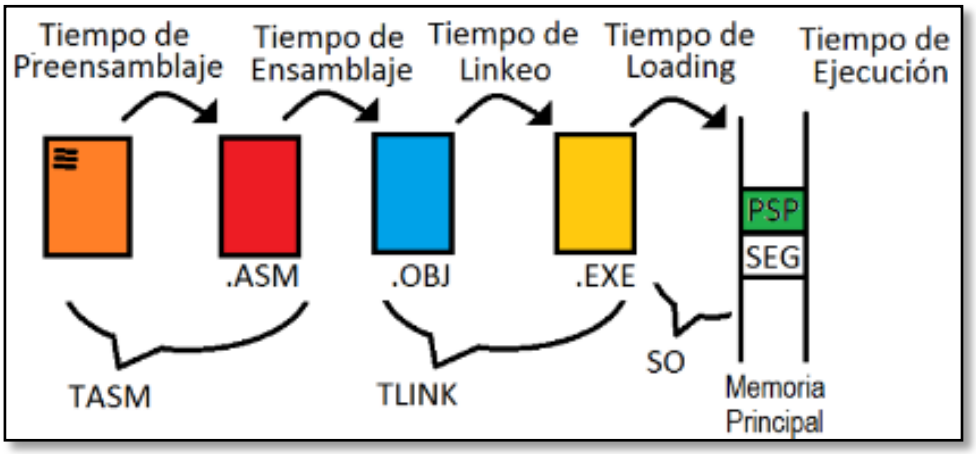
TLINK.exe (este es linker (enlazador)) V 3.01

TD.exe (este es el debugger (depurador))

DOSBox (Máquina virtual)

Tiempos de trabajo

- Tiempo de preensamblaje
- Tiempo de edición
  - Con editor de su preferencia archivo.ASM (texto plano).
  - El procesador solo habla un lenguaje: código máquina.
- Tiempo de ensamblaje:
  - **TASM** archivo.obj (código objeto del programa)
  - Instrucciones en binario, pero sin datos suficientes para ser ejecutadas
- Tiempo de linkeo:
  - **TLINK** archivo.exe (código ejecutable)
  - Instrucciones para ejecutar el programa, son diferentes para cada sistema operativo aunque el código máquina sea el mismo.
  - Son compatibles usualmente si tiene el mismo procesador.
  - Crea código para rutinas que existen, sino existe lanza un error
- Tiempo de carga:
  - Sistema operativo (SO) sube el programa a memoria.
  - La parte del SO que hace ese trabajo es el LOADER (código en memoria queda listo para ejecutarse)
- Tiempo de ejecución:
  - El ciclo de fetch hace su trabajo para cada instrucción.



Instalación del DOSBox

1. Descargar DOSBox

Link : www.dosbox.com

2. Montar una unidad

comando mount: Se necesita montar la carpeta donde se va a trabajar.

Ej:

:> mount k: c:\carpeta

donde **k** es la letra de drive que se usará para trabajar y **c:\carpeta** la carpeta que se crea en la máquina real donde se copian los archivos necesarios para trabajar.

Z: \> MOUNT k : c: \carpeta1\carpeta2 * depende de la cantidad de carpetas/niveles

Z: \> MOUNT T : c: \carpeta1\carpeta2\carpeta3

Z: \> T:

T: \> cd carpeta4

T: \ carpeta4>\ (nombre del archivo .asm)

3. Ejecutar el programa .asm

* Guardar el archivo .asm en el disco duro (c:)

(después del paso 2):

```
T:\(carpeta)>\tasm archivo.asm
T:\(carpeta)>\tlink archivo.asm
T:\(carpeta)> archivo.asm                    * Invoca al loader y crea el PSP
```

Fundamentos de programación en ASM

Detalles

- Los nombres en ensamblador deben ser ÚNICOS en todo el programa.
- Se pueden nombrar : Variables, Segmentos, Etiquetas, Procedimientos, Estructuras, Campos de Estructuras, etc.
- Nombres de máximo **32 caracteres** mezclando letras, dígitos y el underscore(_). Tienen que empezar con una letra
- Las instrucciones pueden tener **0**, **1** o **2** operandos.
- Operandos:
    - operandos fuente: dan información o datos
    - operandos destino: es donde el programador escribe información o datos. SIEMPRE a la izquierda
    - mov ax, 5
- A veces un operando puede ser fuente y destino a la vez. (Add ax, bx ; ax = ax + bx)
- Cada operando tiene su modo de direccionamiento.
- El modo de direccionamiento es la forma en que la UC consigue el dato.
- x86 tiene 9 modos diferentes de direccionamiento.
- Los **comentarios** : de línea y se colocan con un ;
- Los **corchetes** en un operando representan acceso a memoria. **[**bx**]**

Instrucciones

4 familias :

>> aritméticas : su operación se hace mediante la ALU (and, or, add, sub, mul, imul, etc.)
>> transferencia: mover datos de un lado a otro (mov, xlat, xchg, etc.)
>> E/S: comunicarse con otros positivos mediante puertos (in,out)
>> control (jmp, jz , jc, jnc)

- Simples
- Solo una instrucción por línea
- Partes:
  - **Etiqueta**: nombre a las instrucciones, para poder referenciarlas o llamarlas. Se le pone solo a las que se van a referenciar después.
  - n-mónico: Pequeña secuencia de letras que representa el concepto de lo que hace la instrucción. (div) = divide
  - operandos (destino y/o fuente)
  - comentarios

  Ejemplo:

  xxx: add ax , bx ; ax = ax + bx

- Instrucciones de uso frecuente:
  - **Call**: llama a una rutina y luego regresa.
  - **Cmp:** compara dos datos (por medio de restas) Modifica dos banderas SF y ZF(el cero es positivo)
  - **Int**: llamar a una rutina que atiende una interrupción.
  - **Jz** o **Je**, **Jl** … : saltos condicionales se usan para tomar decisiones, como un if

Tabla instrucciones

Instrucción	Función	Instrucción	Función	Instrucción	Función
jmp	Salto	**push**	Ingresa en la pila	**dec**	Resta 1
loop	Salto condicional para incrementar ciclos. (cx = contador)	**pop**	Saca de la pila	**inc**	Suma 1
lea	Calcula la dirección efectiva	**ret**	Return	**sub**	Restar
mov	Move	**xchg**	Intercambio de datos	**div**	Dividir
nop	“no operation”	**add**	Sumar	**mul**	Multiplicar
and	Op de bits	**or**	Op de bits	**xor**	Op de bits
not	Op de bits	**shl**	Corre bits a la izquierda	**shr**	Corre bits a la derecha

Tabla saltos condicionales (uso de negativos)

Salto	Uso	Salto	Uso
JG	> greater	**JNG**	<= not greater
JL	< less	**JNL**	>= not less
JGE	>= great equal	**JNGE**	< not great equal
JLE	<= less equal	**JNLE**	> not les equal
JE	= equal	**JNE**	¡= / <> / >< not equal

Tabla saltos condicionales (uso de naturales)

Salto	Uso	Salto	Uso
JA	above	**JNB**	not below
JB	below	**JNAE**	not above equal
JAE	above equal	**JNBE**	not below equal
JBE	below equal	**JNA**	not above

Registros

Los registros son muy rápidos, es mejor trabajar con estos que con variables, porque las variables están en la memoria. Son como la unidad fundamental de trabajo.

- Forma más veloz de almacenamiento.
- Nombre y un tamaño determinado.
- Se pueden dividir en dos clases:
    - registros **visibles**: el programador los puede usar, y
    - registros **no visibles** : *hacen un trabajo importante en el procesador pero el programador no los puede usar directamente.* (IP o IR, almacena la dirección actual en ejecución o MAR o MBR, asesar memoria principal)

- Todos los registros que terminan en X miden 16 bits.
- Se dividen en H (high) y L(low).
- Usualmente se usa la parte baja, para 8 bits, si se ocupa de 16, se le pone un 0 a la parte alta. Si se pone en la parte alta se puede estar moviendo.
- Registro **BX** es el único con el que se puede ir a memoria

Registro	Función	Registro	Función
SI	Para accesar a memoria	**DS**	datos
DI	Para accesar a memoria	**SS**	Dirección del segmento de Pila
ES	Asociado al DI. Se puede usar las operaciones con cadenas de caracteres	**CS**	Dirección del segmento de código
OF	overflow, desbordamiento de bits	**SF**	signo, los saltos condicionales la utilizan bastante. De la última comparación que se hizo. Se enciende cuando la última operación realizada da positivo
DF	Designa la dirección hacia la izquierda o hacia la derecha para mover o comparar cadenas de caracteres.	**ZF**	Indica el resultado de una operación aritmética. (0= resultado diferente de cero y 1=resultado igual a cero).
IF	interrupción, se indica si se permite realizar interrupciones	**AF**	acarreo
TF	trampa, pasito a pasito. Los programas depuradores la activan	**CF**	acarreo
AX	Para operaciones que implican entrada/salida y la mayor parte de la aritmética.	**CX**	Contener un valor para controlar el número de veces que un ciclo se repite o un valor para corrimiento de bits, hacia la derecha o hacia la izquierda
BX	Es el único registro de propósitos generales. Cálculos.	**DX**	Operaciones de entrada/salida requieren su uso, y las operaciones de multiplicación y división con cifras grandes suponen al DX y al AX trabajando juntos.
SP	Tope de la pila, asociado al SS	**BP**	Facilita la referencia de parámetros

Registros visibles

>> **de trabajo**:

Cada uno de los que termina en X es de 16 bits.

Cada uno de los dos bytes (8 bits) que lo conforman se puede accesar de forma independiente. Al byte más alto se le accesa con una **H de high** y al más bajo con una **L de low**. Ej: AX se divide en el AH y AL.

* En procesadores a partir de 32 bits existe una extensión de cada registro de trabajo que se llama con una E al inicio. Por ejemplo EAX.

AX: Acumulador

CX: Counter

BX: Base

DX: Data

>> **de segmento:**

Almacenan la dirección de los segmentos actuales del programa.

Son de 16 bits

CS: Almacena la dirección del segmento de código (code segment)

DS: Almacena la dirección del segmento de datos (data segment)

SS: Almacena la dirección del segmento de pila (stack segment)

ES: Extra segment (procesadores de 32 bits o más: FS y GS)

>> **índice:**

Se utilizan como registros índices para accesar la memoria. Algo similar a usar vectores en alto nivel.

SI: Source Index DI: Destiny Index

[si] **ds:[si]**

byte ptr word ptr dword ptr

>> **de pila**:

Trabajan con el segmento de pila y son de 16 bits.

Sirven para indexar.

La pila crece al contrario de las direcciones de memoria y almacena datos de 16 bits (trabaja en words)

SP: Stack Pointer = Es el que hace el trabajo de tener el tope de la pila.

BP: Base Pointer = A veces es necesario apuntar a la mitad de la pila.

>> **banderas:**

Es un registro de 16 bits de las cuales hay 9 banderas del hardware y el resto se dejan para el Sistema Operativo.

Los 9 comunes son: ZF SF CF AF PF OF IF DF TF

Stc (enciende)
Clc (apaga)
Cmc (complementa, inverso)

Jc ("jump if carry")
Jnc ("jump if not carry")

Memoria

- La memoria se divide en segmentos traslapados de tamaño de 64 kb. (16 bytes entre segmentos = un párrafo de distancia)
- Como se tiene esta división una dirección debe constar de dos partes: **segment:offset**
- 16 bits para direccionar cada uno: 34E5(número de segmento):67FF (número de desplazamiento)
- Los segmentos comienzan cada párrafo de distancia.
- La memoria es byte direccionable.
- **Dirección normalizada:** Tiene el desplazamiento más bajo y es la que se resalta dentro de las múltiples direcciones equivalentes.
- **Direcciones equivalentes :** Llevan a la misma celda de memoria.
- **Dirección normalizada** : está más cerca al principio del segmento: S' = S + D div 16 D' = D mod 16

Si se avanza en memoria la dirección aumenta, si retrocedo en la memoria la dirección disminuye

Ejemplos del cálculo de la dirección normalizada:

34DE:A10C	B3F5:1010	BE5F:03A2
+A10:	+101	+03A
3EEE:000C	B4F6:0000	BE99:0002

Puntero

***P segmento: desplazamiento**

es:[si] voy a accesar al segmento y el desplazamiento está en el si

byte ptr *(el cuadrito es un byte, si fuera word, el cuadrito sería word)* es:[si] *voy a acceder al segmento y el desplazamiento está en el si*

- [di] asume ds y el ss
- X asume ds
- ss:[sp] ss:[bp]
- ds:[di]
- bp y sp que trabajan con el ss

Modos de Direccionamiento

Es la forma en la cual la unidad de control del procesador consigue un operando para trabajar.

En los x86 hay 9 modos de direccionamiento:

- **Inherente**: es el que no se dice (tácito) cli ej: *multiplicación y división*
- **Inmediato**: Literales almacenadas en el código máquina de la instrucción hexadecimal, binario u octal. [10h , 1010b, 12o, 10 (decimal no necesita letra)]
- **Registro**: Se almacena el dato directamente en un registro del procesador.

Modos de direccionamiento de memoria:

- **Directo**: La UC tiene la dirección de forma inmediata y solo debe accesar la memoria para extraer el dato. Por ejemplo: las variables. (*dirección de memoria directa*)

>> indirectos: hay que conseguir/calcular la dirección para poder accesarla

- **Indexado**: se almacena un registro índice que contiene la dirección de memoria que se desea accesar: add [si], **5** *los **[]** son un apuntador a memoria*

Solo se puede indexar con tres registros que trabajan con el DS por default: **Si, Di, Bx** y uno que trabaja con el SS por default: **Bp**

- **Basado**: Usa un índice desde un punto base. Por ejemplo: add X[si], 1 (*sale a partir del inicio de la variable x)*
- **Indexado-basado**: Mezcla un registro base con uno índice para calcular la dirección de memoria: [bx+si] *(la base no es fija, en matrices sería como recorrer las diferentes filas.)*
- **Indexado-basado-relativo**: Suma un desplazamiento adicional: [bx+si+54h]

- **Relativo** (para saltos condicionales) no almacena la dirección exacta sino que dependen de un punto. Lo que se tiene que mover se guarda en un byte y por eso se pueden salir de rango. Dependen de donde estén

Bifurcaciones

(División del flujo de ejecución de un programa. Una instrucción determina si la siguiente se ejecuta o no.)

Tipos:

- **JMPS** : saltos

No se espera retorno
PC = CS:IP guarda la dirección
Hay:

- Condicional / Incondicional (ej : je / jmp)
- Directo / Indirecto
- Intrasegment (near) / Intersegment (far)

- **PROCS** : procedimientos

Se utilizan para implementar rutinas. Regresan a dónde fueron llamados.
Ej:

```
mov ax, 5
call rutina ; llamamos a la rutina, que en el bx deja el resultado
add bx, cx
```

parámetros:

Alto nivel:
- Valor
- Referencia
- Nombre

Bajo nivel:
- registros
- variables de data segment
- por la pila

➢ **INTS** : Interrupciones

Estado de la máquina es: CS:IP y FLAGS.
Se usan para llamar a rutinas del SO.
Comunes: INT 21H, INT 06H…
Enlace de interrupciones: http://spike.scu.edu.au/~barry/interrupts.html

```
mov ah, 09h  ; función 09h
lea dx, Rot1   ; imprime el un rotulo hasta que encuentre el carácter '$'
int 21h
```

```
mov ax, 4C00h ; función para cerrar el programa
int 21h
```

```
mov dl, 20h      ; Función para imprimir un caracter
mov ah, 02h      ; el carácter se pone en el ah (en este caso es un espacio ' ')
int 21h
```

```
mov ah, 3Eh ; Función para cerrar un archivo
mov bx, handle
int 21h
```

```
mov bx, handle
mov ah, 3Fh  ; Función para leer de un archivo
mov cx, 1      ; En este caso, lee solo un byte (un carcater) (en el cx se pone la cantidad de bytes a leer)
lea dx, buffer
int 21h
```

```
mov ah, 40h   ; Función para escribir en un archivo
mov cx, 1      ; En este ejemplo, escribe solo 1 carácter (en el cx se pone la cantidad de bytes a escribir)
lea dx, buffer
mov bx, handle
int 21h
jc error         ; Si ocurre un error activa la bandera CF (en este caso, salta a la etiqueta 'error' si algo falla)
```

Declaración de datos

Los datos se declaran en el segmento de datos.
Si la cantidad de datos se pasan más de 65000 bytes, se puede crear otro segmento de datos.

Estructura:

```
Datos Segment
  nombre tam inicialización
  nombre tam inicialización
  nombre tam inicialización
  nombre tam inicialización
  nombre tam inicialización
  nombre tam inicialización
Datos EndS
```

Nombre válido en ASM (único) y opcional

(Directiva que indica la cantidad de bytes a reservar)
db = define byte
dw = define word
dd = define double word
dq = define quad word
dt = define ten

(lo que va a contener)
-1 complemento a la base
45h hexadecimal
555o octal
10101b binario
'a' caracter ASCII
? no me interesa

Ejemplos:

1.

A db 73h ; 0 bytes (*desplazamiento 0)*
B dw -2 ; 1 byte FFFEh (*desplazamiento depende del tamaño de la primera variable. Lea dx, X: busca la dirección de X y la guarda en el dx (más completa) // mov dx, offset X (más rápida)*)
C db 'A' ; 3 bytes
Total: 4 bytes

2.

x db 1,2,3,4,5,6,7,8,9,0 (*varias declaraciones, 10 variables distintas y solo la primera tiene nombre, solo la x reserva 1 byte*)
es equivalente a declarar:

```
x db 1
  db 2
  db 3
  db 4
  db 5
  db 6
  db 7
  db 8
  db 9
  db 0
```

3.

casa db 'c','a','s','a' ; *"string"*
casa db "casa",'$' ; (esto solo sirve en TASM)

casa db "casa", 0 ; Like C

casa db 4,"casa" ; Like Pascal ; 4 = tamaño de la cadena

Segmentos de programación

- **Datos**: acá se almacenan las variables del programa y todo dato que se necesite.
- **Código**: acá están las instrucciones del programa (rutinas o código suelto)
- **Pila**: es un segmento que ayuda a que el programa funcione. Almacena direcciones de retorno, parámetros de rutinas, etc.

Programas Ejemplo

Machote : Partes del programa

```
; partes básicas de un programa.

datos segment
  X db ?
  Base dw 10
datos endS

pila segment stack 'stack'
   dw 256 dup(?)
pila endS

codigo segment
      Assume CS:codigo,DS:datos,SS:pila


main:
     mov ax, pila
     mov ss, ax

     mov ax, datos
     mov ds, ax


    ; programa

     mov ax, 4C00h

     int 21h

codigo ends
```

SEGMENTO DE DATOS

SEGMENTO DE PILA

PROTOCOLO DE INICIO

SEGMENTO DE CÓDIGO

Sudáfrica

Programa : Hola Mundo

```
datos segment
  rotulo db "Hola Mundo!$"  ; el dólar es para que la rutina 09h sepa donde terminar
datos endS

pila segment stack 'stack'
   dw 256 dup(?)
pila endS

codigo segment
   assume cs: codigo, ds: datos, ss : pila ; dice quién es quien

main:
  mov ax, pila
  mov ss, ax
  mov ax, datos
  mov ds, ax

  mov ah, 09h ; rutina recibe una dirección y despliega hasta que encuentre el $, termina ahí.
  lea dx, rotulo
  int 21h, 09h

  ; mov ah, 4Ch 4Chexadecimal
  ; xor al, al    ; mov al, 0  código estándar para “todo está bien”

  mov ax ,4C00h
  int 21h
codigo ends
end main
```

Programa: Eco

```
; Este programa de ASM lo que hace es repetir un eco de lo que se le diga en la linea de comandos.
datos segment
   rotulo db 128 dup ( ? ) ;  define bytes
   N dw 4         ; cantidad de repeticiones   define word
datos ends

pila segment stack 'stack'
   dw 256 dup (?)
pila ends

codigo segment
   assume  cs:codigo, ds:datos, ss:pila

inicio: mov ax, ds
        mov es, ax        ; es apunta al inicio del PSP
        mov ax, datos
        mov ds, ax
        mov ax, pila
        mov ss, ax

        mov si, 80h               ; en el 80h está el tamaño
        mov cl, byte ptr es:[si]  ; registro índice para acceso a la memoria ; los registros de x de parten en l y h
        xor ch, ch
        xor di, di ; punteros

ciclo:   inc si                   ; se brinca el 80h
        mov al, byte ptr es:[si]
        mov byte ptr rotulo[di], al
        inc di                    ; con este avanzamos en el rotulo
        loop ciclo

        mov byte ptr rotulo[di], 0Dh    ; 13
        inc di
        mov byte ptr rotulo[di], 0Ah    ; 10 ptr puntero a un solo byte, si no lo pongo lanza una advertencia
        inc di
        mov byte ptr rotulo[di],'$'
 ; despliega el rótulo
        mov cx, N
        mov ah, 09h
        lea dx, rotulo
ciclo2: int 21h
        loop ciclo2
        mov ax, 4C00h
        int 21h
codigo ends
 end inicio
```

Rutina para imprimir un número

Uso de instrucciones: **push, pop, xor, mov, div, inc, cmp, jne, add, int 21h, loop, ret**

Uso de registros : **ax, bx, cx, dx** Etiquetas: en color verde

```
printAX proc near
; imprime a la salida estándar un número que supone estar en el AX
; supone que es un número positivo y natural en 16 bits.
; lo imprime en la base que indica la variable Base del Data Segment.

; vamos a salvar en la pila los registros que se van a modificar.
    push AX
    push BX
    push CX
    push DX

    xor cx, cx
    mov bx, Base
ciclo1PAX: xor dx, dx
    div bx
    push dx
    inc cx
    cmp ax, 0
    jne ciclo1PAX
    mov ah, 02h
ciclo2PAX: pop DX     ; Se restauran los
    add dl, 30h
    int 21h
    loop ciclo2PAX

; con pop se restauran los registros que salvé
    pop DX
    pop CX
    pop BX
    pop AX
    ret
printAX endP
```

Rutina para imprimir un cambio de línea

Uso de instrucciones: **push, pop, mov, int, ret**

Uso de registros : **ax, dx**

```
camLin Proc
   push ax
   push dx
   mov dl, 13          ; Carácter ASCII (retorno de carro)
   mov ah, 02h         ; Función 02h
   int 21h
   mov dl, 10          ; Carácter ASCII (salto de línea)
   int 21h
   pop dx
   pop ax
   ret
camLin EndP
```

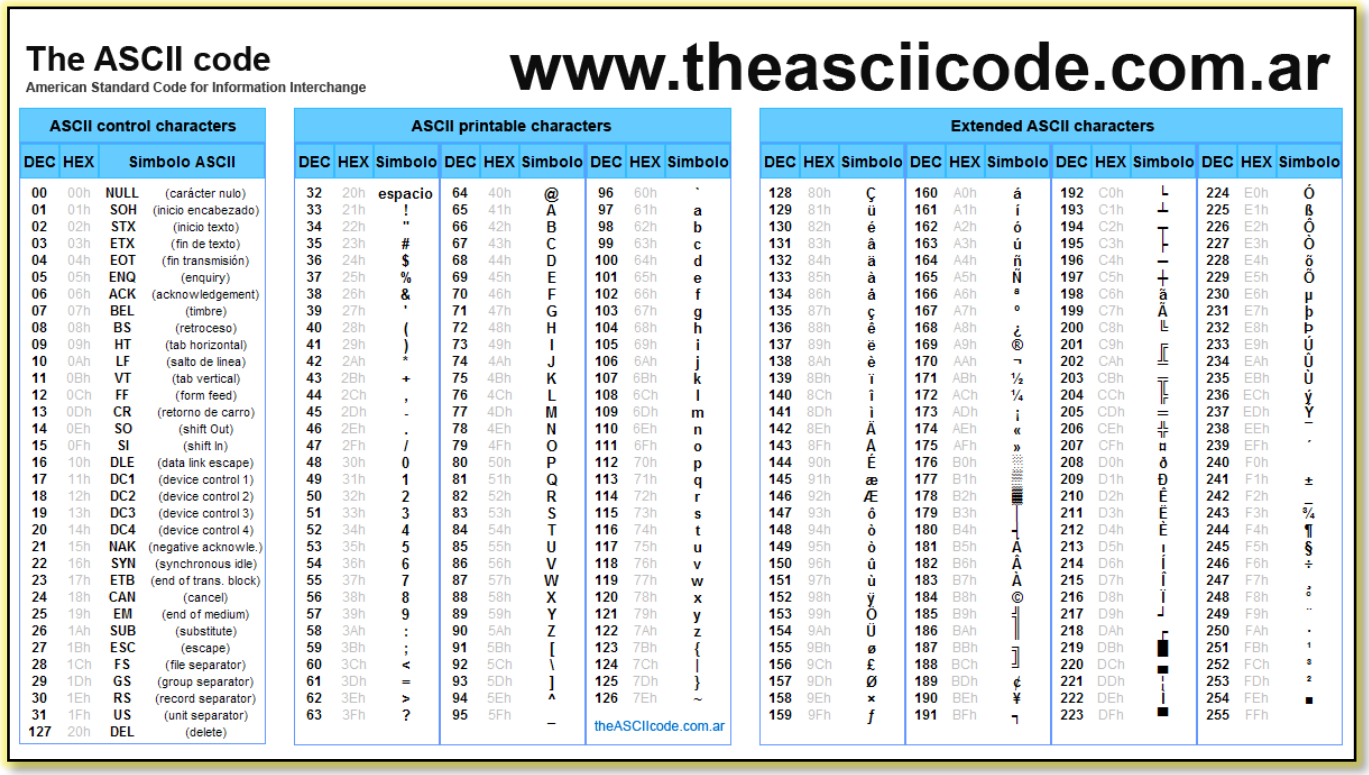
Rutina para imprimir un espacio en blanco

Uso de instrucciones: **push, pop, mov, int, ret**

Uso de registros : **ax, dx**

```
space Proc
   push ax
   push dx
   mov dl, 20h
   mov ah, 02h ; Carácter ASCII (salto de espacio)
   int 21h
   pop dx
   pop ax
   ret
space EndP
```

Tabla ASCII (imagen)



The ASCII code

American Standard Code for Information Interchange

www.theasciicode.com.ar

ASCII control characters

DEC	HEX	Simbolo ASCII	
00	00h	NULL	(carácter nulo)
01	01h	SOH	(inicio encabezado)
02	02h	STX	(inicio texto)
03	03h	ETX	(fin de texto)
04	04h	EOT	(fin transmisión)
05	05h	ENQ	(enquiry)
06	06h	ACK	(acknowledgement)
07	07h	BEL	(timbre)
08	08h	BS	(retroceso)
09	09h	HT	(tab horizontal)
10	0Ah	LF	(salto de linea)
11	0Bh	VT	(tab vertical)
12	0Ch	FF	(form feed)
13	0Dh	CR	(retorno de carro)
14	0Eh	SO	(shift Out)
15	0Fh	SI	(shift In)
16	10h	DLE	(data link escape)
17	11h	DC1	(device control 1)
18	12h	DC2	(device control 2)
19	13h	DC3	(device control 3)
20	14h	DC4	(device control 4)
21	15h	NAK	(negative acknowle.)
22	16h	SYN	(synchronous idle)
23	17h	ETB	(end of trans. block)
24	18h	CAN	(cancel)
25	19h	EM	(end of medium)
26	1Ah	SUB	(substitute)
27	1Bh	ESC	(escape)
28	1Ch	FS	(file separator)
29	1Dh	GS	(group separator)
30	1Eh	RS	(record separator)
31	1Fh	US	(unit separator)
127	20h	DEL	(delete)

ASCII printable characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	|
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

theASCIIcode.com.ar

Extended ASCII characters

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	└	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	┴	225	E1h	ß
130	82h	é	162	A2h	ó	194	C2h	┬	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	├	227	E3h	Ò
132	84h	ä	164	A4h	ñ	196	C4h	─	228	E4h	õ
133	85h	à	165	A5h	Ñ	197	C5h	┼	229	E5h	Õ
134	86h	å	166	A6h	ª	198	C6h	ã	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	Ã	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	╚	232	E8h	Þ
137	89h	ë	169	A9h	®	201	C9h	╔	233	E9h	Ú
138	8Ah	è	170	AAh	¬	202	CAh	╩	234	EAh	Û
139	8Bh	ï	171	ABh	½	203	CBh	╦	235	EBh	Ù
140	8Ch	î	172	ACh	¼	204	CCh	╠	236	ECh	ý
141	8Dh	ì	173	ADh	¡	205	CDh	═	237	EDh	Ý
142	8Eh	Ä	174	AEh	«	206	CEh	╬	238	EEh	¯
143	8Fh	Å	175	AFh	»	207	CFh	¤	239	EFh	´
144	90h	É	176	B0h	░	208	D0h	ð	240	F0h	
145	91h	æ	177	B1h	▒	209	D1h	Ð	241	F1h	±
146	92h	Æ	178	B2h	▓	210	D2h	Ê	242	F2h	‗
147	93h	ô	179	B3h	│	211	D3h	Ë	243	F3h	¾
148	94h	ö	180	B4h	┤	212	D4h	È	244	F4h	¶
149	95h	ò	181	B5h	Á	213	D5h	ı	245	F5h	§
150	96h	û	182	B6h	Â	214	D6h	Í	246	F6h	÷
151	97h	ù	183	B7h	À	215	D7h	Î	247	F7h	¸
152	98h	ÿ	184	B8h	©	216	D8h	Ï	248	F8h	°
153	99h	Ö	185	B9h	╣	217	D9h	┘	249	F9h	¨
154	9Ah	Ü	186	BAh	║	218	DAh	┌	250	FAh	·
155	9Bh	ø	187	BBh	╗	219	DBh	█	251	FBh	¹
156	9Ch	£	188	BCh	╝	220	DCh	▄	252	FCh	³
157	9Dh	Ø	189	BDh	¢	221	DDh	¦	253	FDh	²
158	9Eh	×	190	BEh	¥	222	DEh	▌	254	FEh	■
159	9Fh	ƒ	191	BFh	┐	223	DFh	▀	255	FFh	