Guía de trabajo – NASM

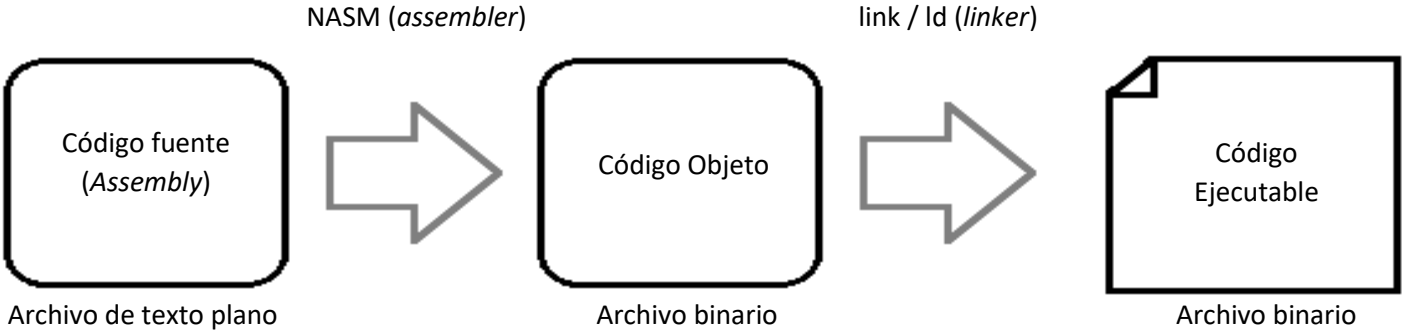
Introducción

El **Netwide Assembler** o **NASM** es un ensamblador libre para la plataforma Intel x86. Puede ser usado para escribir programas en *assembly* de 16 bits, de 32 bits (IA-32) o de 64 bits (x86-64).

En esta guía utilizaremos ***assembly* de 32 bits**.

El NASM produce principalmente código objeto que, por lo general, no es directamente ejecutable. Para producir programas ejecutables, se debe usar un enlazador apropiado como, por ejemplo, el comando `ld` disponible en sistemas derivados de UNIX (como **GNU/Linux** o **macOS**).

En **Windows**, la mayoría de los IDE para **C** (como Code::Blocks, Qt Creator, etc.) pueden instalarse con MinGW, que incluye el enlazador `ld`. Si se opta por esta alternativa, en lugar de utilizar directamente el enlazador, se llamará al *front-end* `gcc`, y será este el encargado de llamar a `ld`. Otra posibilidad es usar la utilidad `link` de Visual Studio.



Estructura básica del código fuente en *assembly*

El código fuente se estructura de la siguiente manera:

- **Al inicio:** Las entradas `global` declaran etiquetas que marcarán el punto de arranque del programa. Según el enlazador que se vaya a utilizar, se requerirá `_start` o `_main`. Es recomendable colocar ambas, ya que las etiquetas no utilizadas se ignoran. Después, se declaran con `extern` las funciones importadas. De este modo, todos los programas se inician con las líneas:

`ld`	`link (Visual Studio)`	`Para ambos (*)`

`global _start`
 	
`extern printf`
`extern scanf`
`extern exit`
`extern gets`
	`global _main`
	
	`extern _printf`
	`extern _scanf`
	`extern _exit`
	`extern _gets`
		`global _start`
		`global main`
		
		`extern printf`
		`extern scanf`
		`extern exit`
		`extern gets`

(*) `ld` ignorará `main` y `link` funcionará solo si NASM se usa con `--PREFIX _`

- **section .bss:** sección de declaración de variables. En esta sección se declara cada variable a utilizar. Por ejemplo:

```
numero:
      resd    1              ; 1 dword (4 bytes)

cadena:
      resb    0x0100         ; 256 bytes

caracter:
      resb    1              ; 1 byte (dato)
      resb    3              ; 3 bytes (relleno)
```

- **section .data:** sección de declaración de constantes que se usarán en el programa. Por ejemplo, la especificación de formatos para `scanf` y `printf`.

```
fmtInt:
      db    "%d", 0          ; FORMATO PARA NUMEROS ENTEROS

fmtString:
      db    "%s", 0          ; FORMATO PARA CADENAS

fmtChar:
      db    "%c", 0          ; FORMATO PARA CARACTERES

fmtLF:
      db    0xA, 0           ; SALTO DE LINEA (LF)
```

- **section .text:** sección de instrucciones. Esta sección se puede (y se recomienda) dividir con etiquetas, que marcarán puntos relevantes del programa. Estos puntos los crea el programador para ordenar el código de manera que la ejecución del programa pueda saltar a dichos puntos. Para poder utilizar las etiquetas como funciones invocables mediante `call`, es necesario colocar la instrucción `ret` al final del conjunto de instrucciones que constituyen la función, para seguir con la instrucción que sucede a la que efectuó la llamada. Por ejemplo:

```
mostrarNumero:               ; RUTINA PARA MOSTRAR UN NUMERO ENTERO USANDO PRINTF
      push dword [numero]
      push fmtInt
      call printf
      add esp, 8
      ret
```

NOTA: el carácter ';' marca el comentario.

Dado que en esta sección habrá funciones definidas una debajo de la otra, la ejecución del programa se iniciará por la instrucción etiquetada con `_start` o `main`.

```
_start:
main:                              ; PUNTO DE INICIO DEL PROGRAMA
      call leerCadena
      mov edi, 0
      mov eax, 0
seguir:
      mov al, [edi + cadena]
      cmp al, 0
      je finPrograma
      cmp al, 97
      jb dejar
      cmp al, 122
      ja dejar
      sub al, 32
dejar:
      mov [caracter], al
      call mostrarCaracter
      inc edi
      jmp seguir
finPrograma:
      call mostrarSaltoDeLinea
      jmp salirDelPrograma
```

Observar que en este último código hay llamados a etiquetas que indican funciones, como ser `leerCadena`, y saltos a etiquetas que solo indican una instrucción, como ser `seguir`. Dicho esto, se ve que el código de arriba describe un ciclo, indicando con etiquetas el destino de los saltos.

Cómo transformar el *assembly* en un archivo ejecutable para poder correrlo

El modo de trabajo varía según el sistema operativo que se esté utilizando.

En GNU/Linux, una vez instalado NASM, deberán ejecutarse los siguientes pasos:

En **GNU/Linux de 32 bits**:

1) `nasm -f elf` **`ejemplo.asm`**

2) `ld -s -o` **`ejemplo ejemplo.o`** `-lc -I /lib/ld-linux.so.2`

3) `./`**`ejemplo`**

En **GNU/Linux de 64 bits** (en Ubuntu, previamente, hay que ejecutar: `sudo apt-getinstall libc6-dev-i386`):

1) `nasm -f elf` **`ejemplo.asm`**

2) `ld -m elf_i386 -s -o` **`ejemplo ejemplo.o`** `-lc -I /lib/ld-linux.so.2`

3) `./`**`ejemplo`**

En **Windows**, primero hay que descargar NASM del sitio web oficial y ejecutar el instalador con permisos de Administrador, dejando todas las opciones por defecto. Además, debemos disponer de un *linker*.

Si se prefiere usar la utilidad `link` disponible en la consola del **Visual Studio**, los comandos a emplear son:

1) `nasm -f win32 ejemplo.asm --PREFIX _`

2) `link /out:ejemplo.exe ejemplo.obj libcmt.lib`

3) `ejemplo`

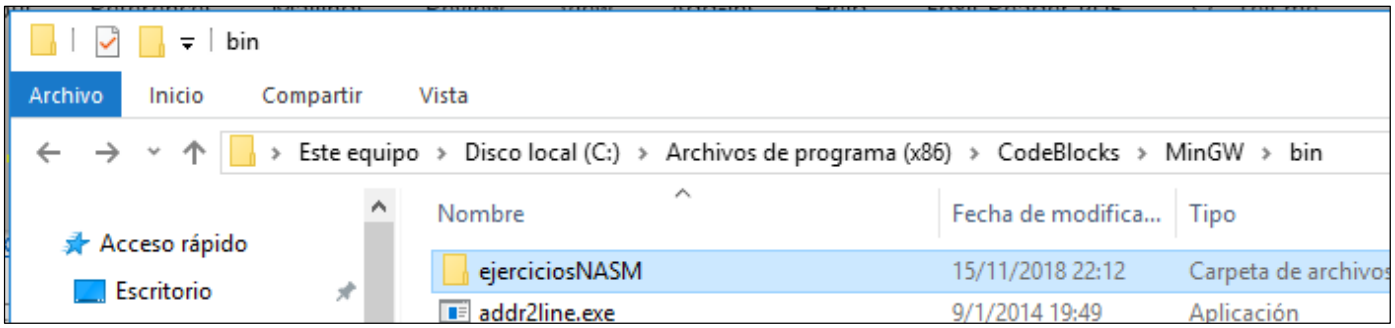
Notar que los comandos deberán adaptarse según la carpeta en la que estemos parados en la línea de comandos y la carpeta en la que se encuentran los archivos (asm y obj).

Cuando no se dispone del Visual Studio (o cuando la versión instalada no incluye una consola donde esté disponible la utilidad `link`), lo recomendable es utilizar **MinGW** (en su versión de 32 bits), pues ocupa mucho menos espacio que el Visual Studio. Existen varias maneras de obtener este software:

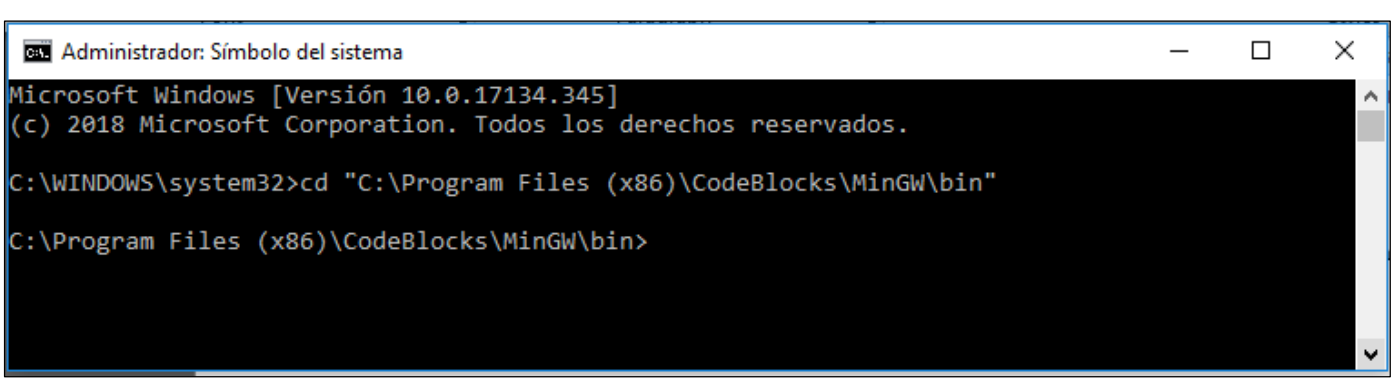
a) Descargando un MinGW antiguo, no instalable, que solo requiera copiar los archivos, por ejemplo de aquí.
b) Descargando de aquí un MinGW-w64 actual y, al instalarlo, configurarlo para i686.
c) Utilizando el MinGW que ya está instalado en Code::Blocks, Qt Creator u otro IDE (de 32 bits).

MinGW incluye diversas herramientas, entre ellas `gcc` (un *front-end* para C que llama al *linker* `ld`).

Para comenzar a utilizar este software en conjunto con **NASM**, crear una carpeta llamada `ejerciciosNASM` en la carpeta `bin` de **MinGW** (por ejemplo, en `C:\Program Files (x86)\CodeBlocks\MinGW\bin`), como se ve a continuación:

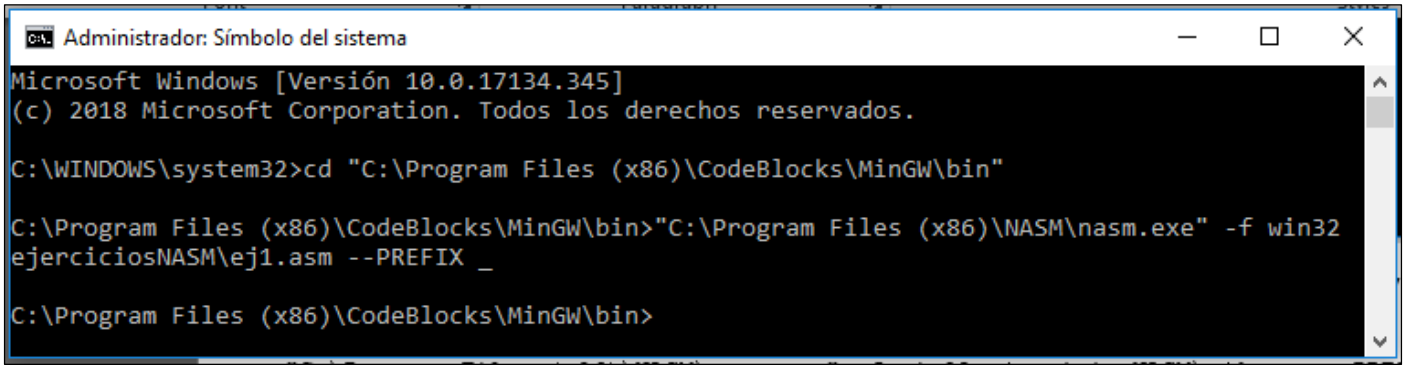


Colocar aquí el archivo asm, en nuestro ejemplo `ej1.asm`. Iniciar una línea de comandos (cmd.exe) como administrador y pasarse a la carpeta `C:\Program Files (x86)\CodeBlocks\MinGW\bin` ejecutando el comando `cd "C:\Program Files (x86)\CodeBlocks\MinGW\bin"`

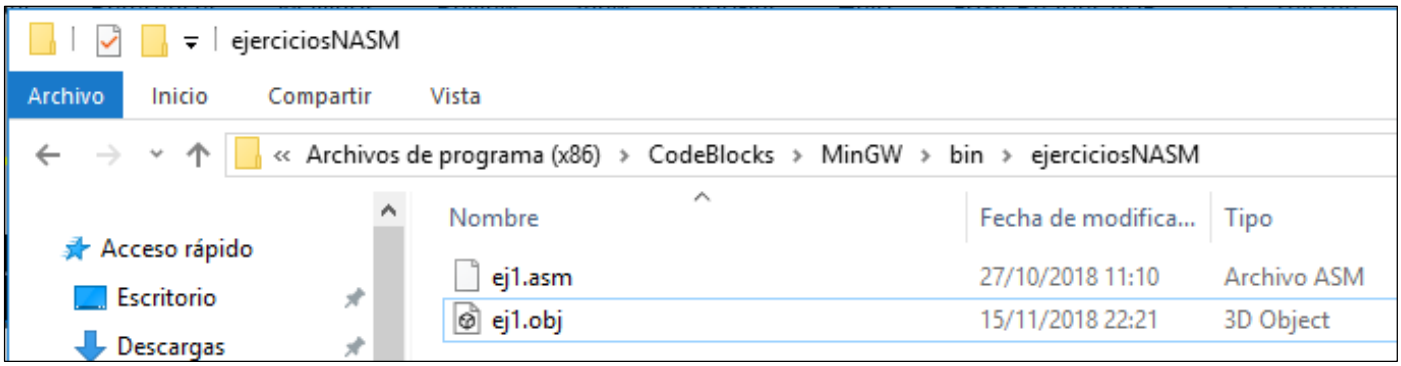


Dado que el ejecutable del NASM está en su carpeta de instalación, para ensamblar nuestro **`ej1.asm`** hay que ejecutar el comando:

```
"C:\Program Files (x86)\NASM\nasm.exe" -f win32 ejerciciosNASM\ej1.asm --PREFIX _
```

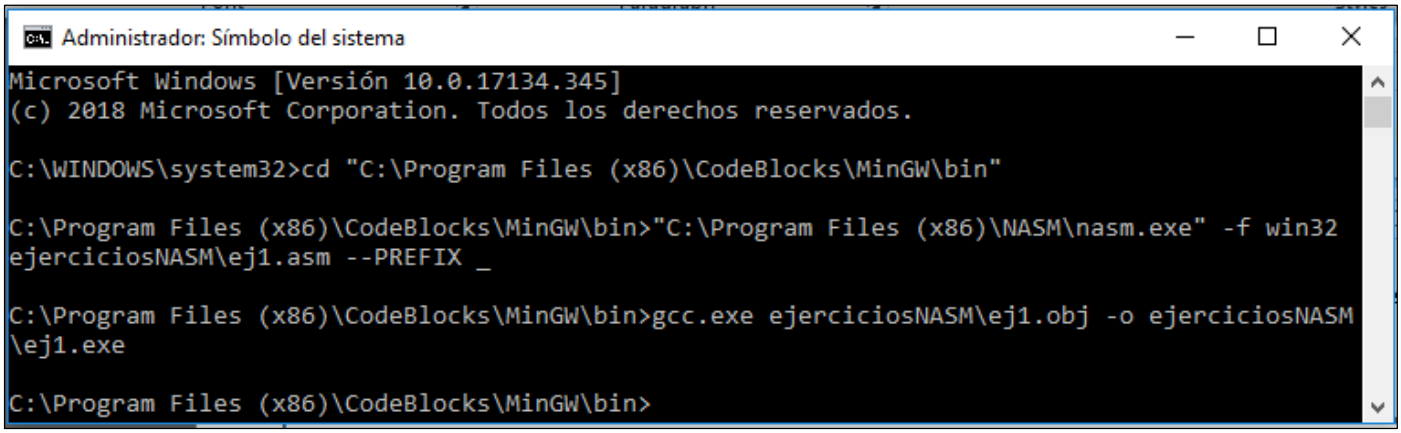


Vemos, en la carpeta `ejerciciosNASM`, que se creó el archivo **`ej1.obj`**:

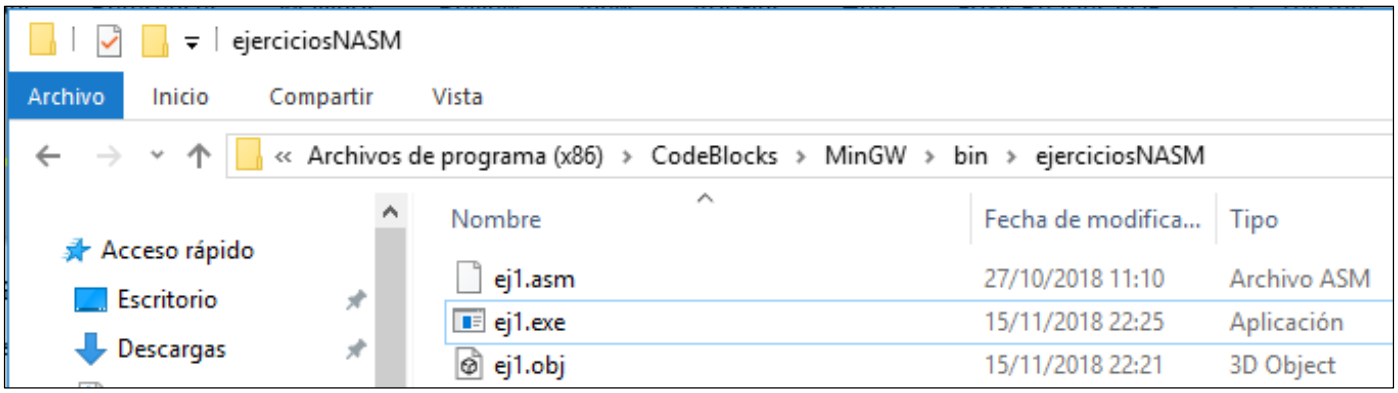


Luego sigue enlazar las funciones de C al archivo objeto creado. Para ello utilizaremos `gcc` (que llamará a `ld`):

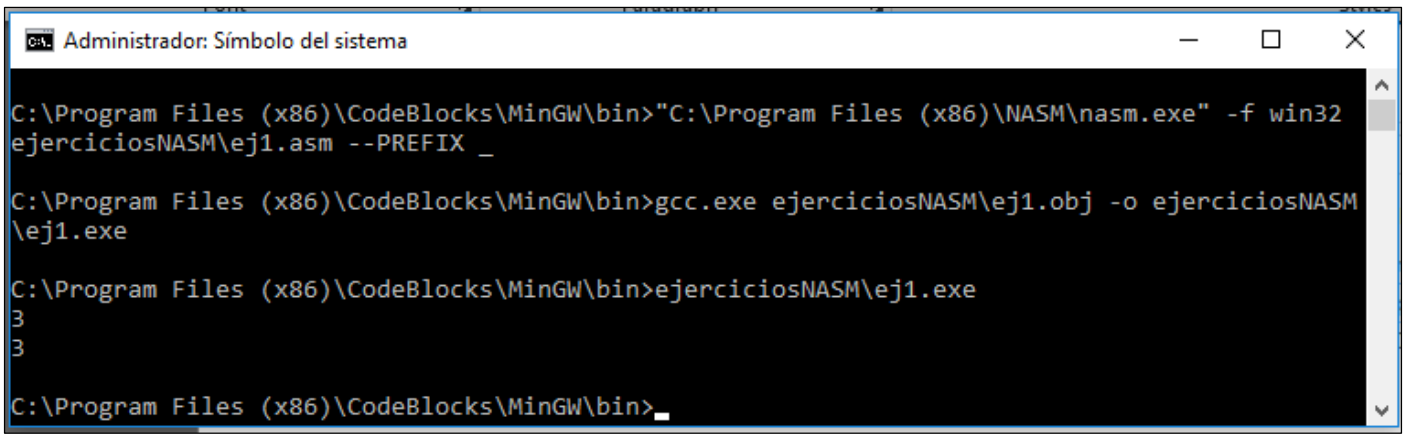
```
gcc.exe ejerciciosNASM\ej1.obj -o ejerciciosNASM\ej1.exe
```



Y con esto tenemos nuestro ejecutable **`ej1.exe`**:



IMPORTANTE: para poder ver lo que corren estos ejecutables, deben lanzarlos por la línea de comandos; como se ve a continuación.



Ahora, recomendamos estudiar detalladamente los archivos `ejemplo1.asm` y `ejemplo2.asm` que se indican a continuación y, tras analizarlos, resolver los ocho ejercicios de la guía.

ejemplo1.asm

```
   global main             ; ETIQUETAS QUE MARCAN EL PUNTO DE INICIO DE LA EJECUCION
   global _start

   extern printf           ;
   extern scanf            ; FUNCIONES DE C (IMPORTADAS)
   extern exit             ;
   extern gets             ; GETS ES PELIGROSA. SOLO USARLA EN EJERCICIOS BASICOS!!!


section .bss                ; SECCION DE LAS VARIABLES

numero:
   resd    1               ; 1 dword (4 bytes)

cadena:
   resb    0x0100          ; 256 bytes

caracter:
   resb    1               ; 1 byte (dato)
   resb    3               ; 3 bytes (relleno)


section .data               ; SECCION DE LAS CONSTANTES

fmtInt:
   db    "%d", 0           ; FORMATO PARA NUMEROS ENTEROS

fmtString:
   db    "%s", 0           ; FORMATO PARA CADENAS

fmtChar:
   db    "%c", 0           ; FORMATO PARA CARACTERES

fmtLF:
   db    0xA, 0            ; SALTO DE LINEA (LF)


section .text               ; SECCION DE LAS INSTRUCCIONES

leerCadena:                 ; RUTINA PARA LEER UNA CADENA USANDO GETS
   push cadena
   call gets
   add esp, 4
   ret

leerNumero:                 ; RUTINA PARA LEER UN NUMERO ENTERO USANDO SCANF
   push numero
   push fmtInt
   call scanf
   add esp, 8
   ret
```

ejemplo1.asm (cont.)

```
mostrarCadena:              ; RUTINA PARA MOSTRAR UNA CADENA USANDO PRINTF
   push cadena
   push fmtString
   call printf
   add esp, 8
   ret

mostrarNumero:              ; RUTINA PARA MOSTRAR UN NUMERO ENTERO USANDO PRINTF
   push dword [numero]
   push fmtInt
   call printf
   add esp, 8
   ret

mostrarCaracter:            ; RUTINA PARA MOSTRAR UN CARACTER USANDO PRINTF
   push dword [caracter]
   push fmtChar
   call printf
   add esp, 8
   ret

mostrarSaltoDeLinea:        ; RUTINA PARA MOSTRAR UN SALTO DE LINEA USANDO PRINTF
   push fmtLF
   call printf
   add esp, 4
   ret

salirDelPrograma:           ; PUNTO DE SALIDA DEL PROGRAMA USANDO EXIT
   push 0
   call exit

_start:
main:                       ; PUNTO DE INICIO DEL PROGRAMA
   call leerCadena
   mov edi,0
   mov eax, 0
seguir:
   mov al,[edi + cadena]
   cmp al,0
   je finPrograma
   cmp al, 97
   jb dejar
   cmp al, 122
   ja dejar
   sub al, 32
dejar:
   mov [caracter], al
   call mostrarCaracter
   inc edi
   jmp seguir
finPrograma:
   call mostrarSaltoDeLinea
   jmp salirDelPrograma
```

ejemplo2.asm

```
   global main             ; ETIQUETAS QUE MARCAN EL PUNTO DE INICIO DE LA EJECUCION
   global _start

   extern printf           ;
   extern scanf            ; FUNCIONES DE C (IMPORTADAS)
   extern exit             ;
   extern gets             ; GETS ES PELIGROSA. SOLO USARLA EN EJERCICIOS BASICOS!!!


section .bss                ; SECCION DE LAS VARIABLES

numero:
   resd    1               ; 1 dword (4 bytes)

cadena:
   resb    0x0100          ; 256 bytes

caracter:
   resb    1               ; 1 byte (dato)
   resb    3               ; 3 bytes (relleno)

matriz:
   resd    100             ;  matriz como maximo de 10x10

n:
   resd    1               ;  lado de la matriz (cuadrada)

f:
   resd    1               ; fila

c:
   resd    1               ; columna


section .data               ; SECCION DE LAS CONSTANTES

fmtInt:
   db    "%d", 0           ; FORMATO PARA NUMEROS ENTEROS

fmtString:
   db    "%s", 0           ; FORMATO PARA CADENAS

fmtChar:
   db    "%c", 0           ; FORMATO PARA CARACTERES

fmtLF:
   db    0xA, 0            ; SALTO DE LINEA (LF)

nStr:
   db    "N: ", 0          ; Cadena "N: "
```

ejemplo2.asm (cont.)

```
filaStr:
   db    "Fila:", 0         ;  Cadena "Fila:"

columnaStr:
   db    " Columna:", 0     ;  Cadena "Columna:"


section .text               ; SECCION DE LAS INSTRUCCIONES

leerCadena:                 ; RUTINA PARA LEER UNA CADENA USANDO GETS
   push cadena
   call gets
   add esp, 4
   ret

leerNumero:                 ; RUTINA PARA LEER UN NUMERO ENTERO USANDO SCANF
   push numero
   push fmtInt
   call scanf
   add esp, 8
   ret

mostrarCadena:              ; RUTINA PARA MOSTRAR UNA CADENA USANDO PRINTF
   push cadena
   push fmtString
   call printf
   add esp, 8
   ret

mostrarNumero:              ; RUTINA PARA MOSTRAR UN NUMERO ENTERO USANDO PRINTF
   push dword [numero]
   push fmtInt
   call printf
   add esp, 8
   ret

mostrarCaracter:            ; RUTINA PARA MOSTRAR UN CARACTER USANDO PRINTF
   push dword [caracter]
   push fmtChar
   call printf
   add esp, 8
   ret

mostrarSaltoDeLinea:        ; RUTINA PARA MOSTRAR UN SALTO DE LINEA USANDO PRINTF
   push fmtLF
   call printf
   add esp, 4
   ret

salirDelPrograma:           ; PUNTO DE SALIDA DEL PROGRAMA USANDO EXIT
   push 0
   call exit
```

ejemplo2.asm (cont.)

```
_start:
main:                          ; PUNTO DE INICIO DEL PROGRAMA
   mov esi, 0
   mov ebx, 0
copiaAcadena1:
   mov al, [ebx+nStr]
   mov [ebx+cadena], al
   inc ebx
   cmp al, 0
   jne copiaAcadena1
   call mostrarCadena
   call leerNumero

   mov eax, [numero]
   cmp eax, 0
   jg seguir1
   jmp main
seguir1:
   cmp eax, 11
   jl seguir2
   jmp main
seguir2:
   mov [n], eax

   mov [f], dword 0
proximoF:
   mov [c], dword 0
proximoC:
   mov ebx, 0
copiaAcadena2:
   mov al, [ebx+filaStr]
   mov [ebx+cadena], al
   inc ebx
   cmp al, 0
   jne copiaAcadena2
   call mostrarCadena

   mov eax, [f]
   mov [numero], eax
   call mostrarNumero

   mov ebx, 0
copiaAcadena3:
   mov al, [ebx+columnaStr]
   mov [ebx+cadena], al
   inc ebx
   cmp al, 0
   jne copiaAcadena3
   call mostrarCadena
```

ejemplo2.asm (cont.)

```
    mov eax, [c]
    mov [numero], eax
    call mostrarNumero

    mov eax, 32
    mov [caracter], eax
    call mostrarCaracter
    call mostrarCaracter
    call leerNumero
    mov eax, [numero]
    mov [esi+matriz], eax
    add esi, 4

    inc dword [c]
    mov eax, [c]
    cmp eax, [n]
    jb proximoC

    inc dword [f]
    mov eax, [f]
    cmp eax, [n]
    jb proximoF

    call mostrarSaltoDeLinea

    mov edi, 0
    mov esi, matriz
cicloDiagonal:
    mov eax, [esi]
    mov [numero], eax
    call mostrarNumero

    mov eax, 32
    mov [caracter], eax
    call mostrarCaracter

    add esi, [n]
    add esi, [n]
    add esi, [n]
    add esi, [n]
    add esi, 4
    inc edi
    cmp edi, [n]
    jl cicloDiagonal

    jmp salirDelPrograma
```

Ejercicios

1. Dado un entero N, la computadora lo muestra descompuesto en sus factores primos. Ej: 132 = 2 × 2 × 3 × 11

2. Se ingresa una cadena. La computadora muestra las subcadenas formadas por las posiciones pares e impares de la cadena. Ej: FAISANSACRO : ASNAR FIASCO

3. Se ingresa un año. La computadora indica si es, o no, bisiesto.

4. Se ingresan un entero N y, a continuación, N números enteros. La computadora muestra el promedio de los números impares ingresados y la suma de los pares.

5. Se ingresan 100 caracteres. La computadora los muestra ordenados sin repeticiones.

6. Se ingresa N. La computadora muestra los primeros N términos de la Secuencia de Connell.

7. Se ingresa una matriz de NxM componentes. La computadora la muestra girada 90° en sentido antihorario.

8. Se ingresa una matriz de NxN componentes enteras. La computadora muestra la matriz transpuesta.