Guía de trabajo – NASM

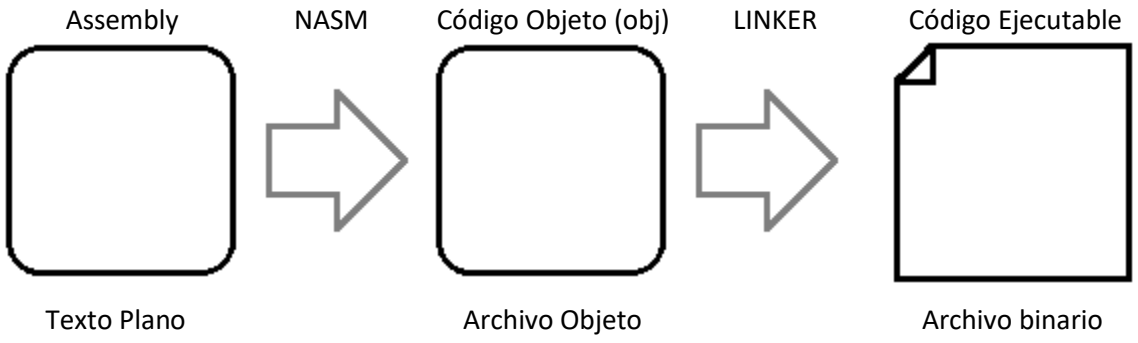
Introducción

El **Netwide Assembler** o **NASM**, es un ensamblador libre para la plataforma Intel x86. Puede ser usado para escribir programas tanto de 16 bits como de 32 bits (IA-32). En el NASM, si se usan las bibliotecas correctas, los programas de 32 bits se pueden escribir de una manera tal para que sean portables entre cualquier sistema operativo x86 de 32 bits.

En adelante, utilizaremos los archivos asm provistos por separado: ejemplo1.asm y ejemplo2.asm.

El NASM

El NASM produce principalmente código objeto, que por lo general no son ejecutables por sí mismos. La única excepción a esto es el binario plano (.COM) que es inherentemente limitado en el uso moderno. Para traducir los archivos objeto a programas ejecutables, se debe usar un enlazador apropiado, por ejemplo, la utilidad "LINK" del Visual Studio de Windows, o el LD para sistemas similares a UNIX (como GNU/Linux).



El código fuente se estructura en secciones:

- Inicio: sección de líneas para definir funciones externas con `extern`. Las entradas `global` son el punto de inicio del programa (se ponen ambas líneas para Windows/Linux). Según para que sistema operativo se compile, hay palabras reservadas que no se reconocen entre uno y otro, como por ejemplo la etiqueta que define dónde inicia el programa (en Windows es `_start` y en Linux es `main`). En estos casos la etiqueta que no se reconoce, se ignora. De modo que todo el programa se inicia con las líneas:

```
global main
global _start
extern printf
extern scanf
extern exit
extern gets
```

- 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 usaran en el programa. Por ejemplo, la especificación de formatos.

```
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) dividirla en etiquetas, que son puntos relevantes del programa. Estos puntos los crea el programador para ordenar el código de manera que puede saltar la ejecución del programa a dicho punto, una suerte de "funciones". Para poder utilizar las etiquetas como funciones, es necesario colocar la instrucción `ret` al final para seguir ejecutando la instrucción siguiente al `jmp` que llamo a la función predecesora. 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, el programa iniciará por aquella llamada _start o main. Por lo tanto, debe haber una función con dicho nombre para poder iniciar el programa. Para mejorar la compatibilidad entre sistemas operativos, utilizar ambas etiquetas como se ve a continuación:

```
_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
```

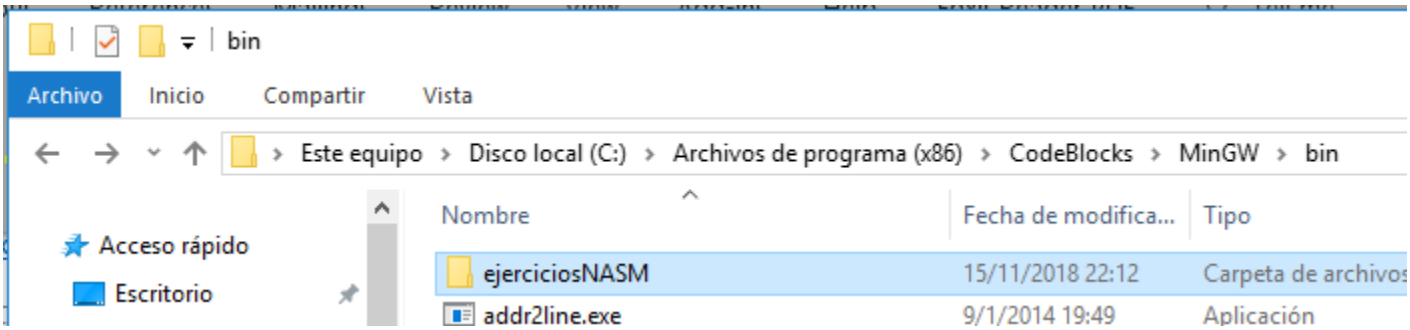
Como se ve, desde esta etiqueta se llama a otra y así sucesivamente hasta terminar la ejecución del programa. Observar que en medio del programa "principal" hay saltos a etiquetas que son "funciones", como ser `leerCadena`, y saltos a etiquetas que solo referencian una línea de código, como ser `seguir`. Este segundo uso es el verdadero modo de uso de las etiquetas: referenciar puntos relevantes en el código para saltar a ellos cuando es necesario. Dicho esto, se ve que código de arriba describe un ciclo `while` referenciando saltos con etiquetas.

Como compilar código NASM

El modo de compilación varía según la plataforma en la que se esté trabajando.

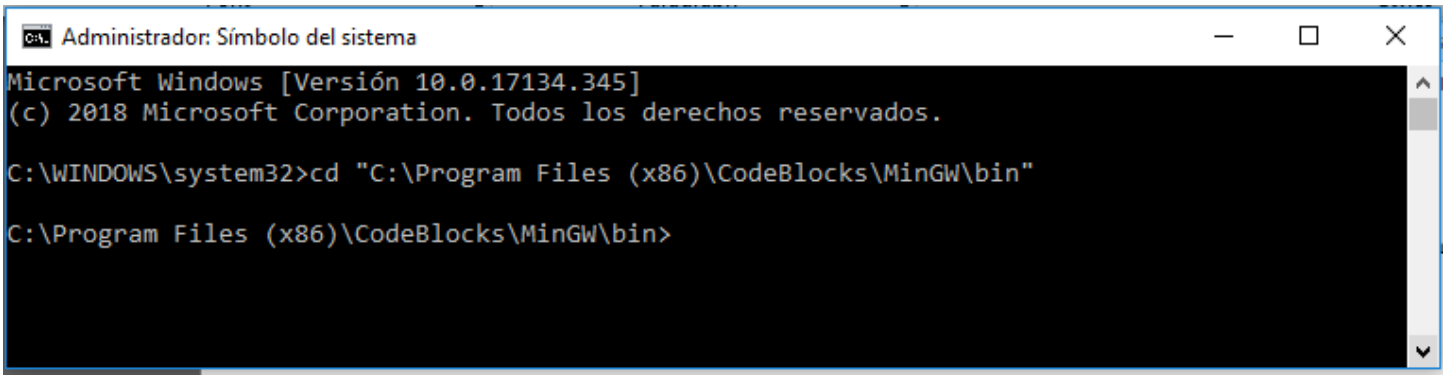
Primero hay que descargar NASM del sitio web oficial, la opción recomendada en Windows es nasm-2.14-installer-x86.exe. Ejecutar el instalador con permisos de Administrador dejando todas las opciones por defecto (instalara en la carpeta `C:\Program Files (x86)\NASM`). Además, necesitaremos instalar un linker. Por cuestiones prácticas, usaremos el GCC provisto por el CodeBlocks (se puede descargar desde aquí).

Una vez instalado Codeblocks, crea una carpeta llamada `ejerciciosNASM` en la carpeta `C:\Program Files (x86)\CodeBlocks\MinGW\bin` como se ve a continuación:



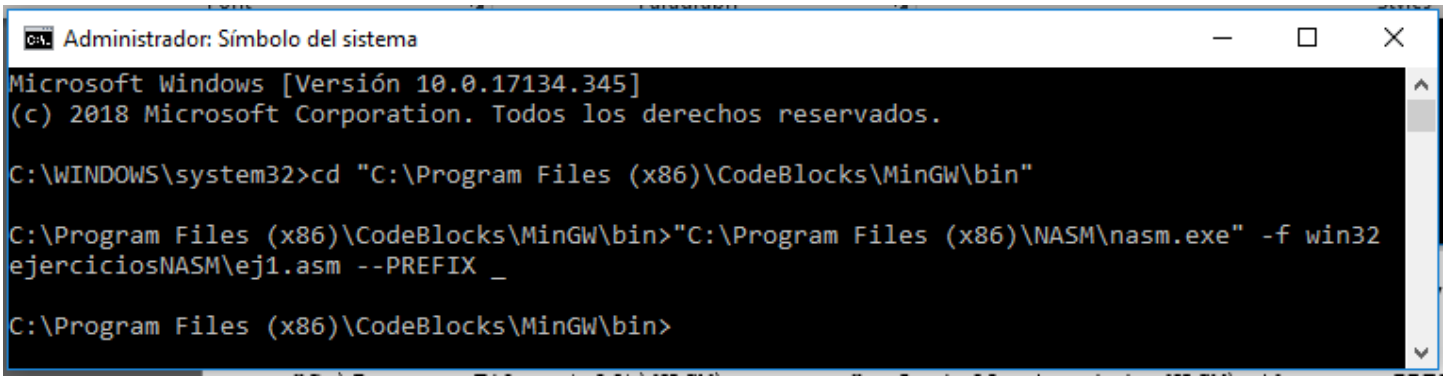
Coloca aquí el archivo asm, en nuestro ejemplo el ej1.asm.

Luego inicia una línea de comandos (cmd.exe) como administrador y cambia a la carpeta `C:\Program Files (x86)\CodeBlocks\MinGW\bin` corriendo el comando `cd "C:\Program Files (x86)\CodeBlocks\MinGW\bin"`

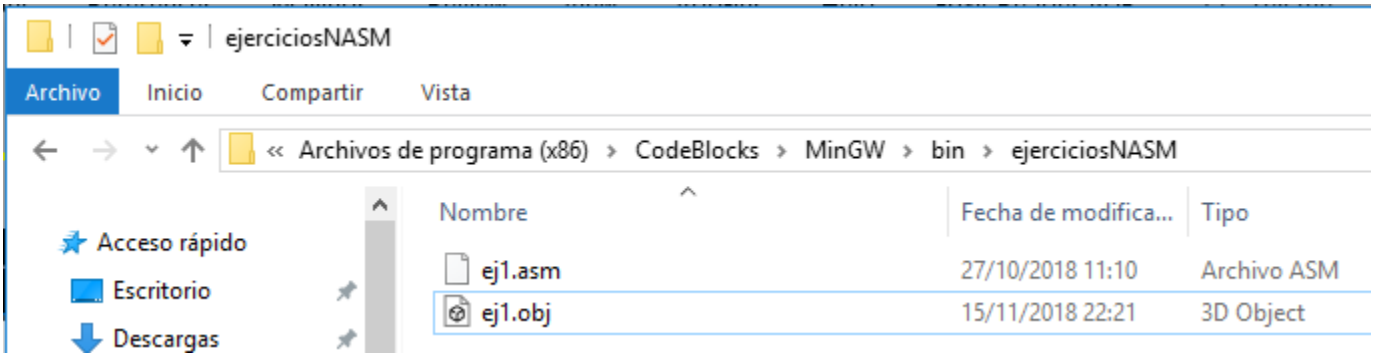


Dado que el ejecutable del NASM esta en su carpeta de instalación, para compilar nuestro ej1.asm hay que correr el comando:

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

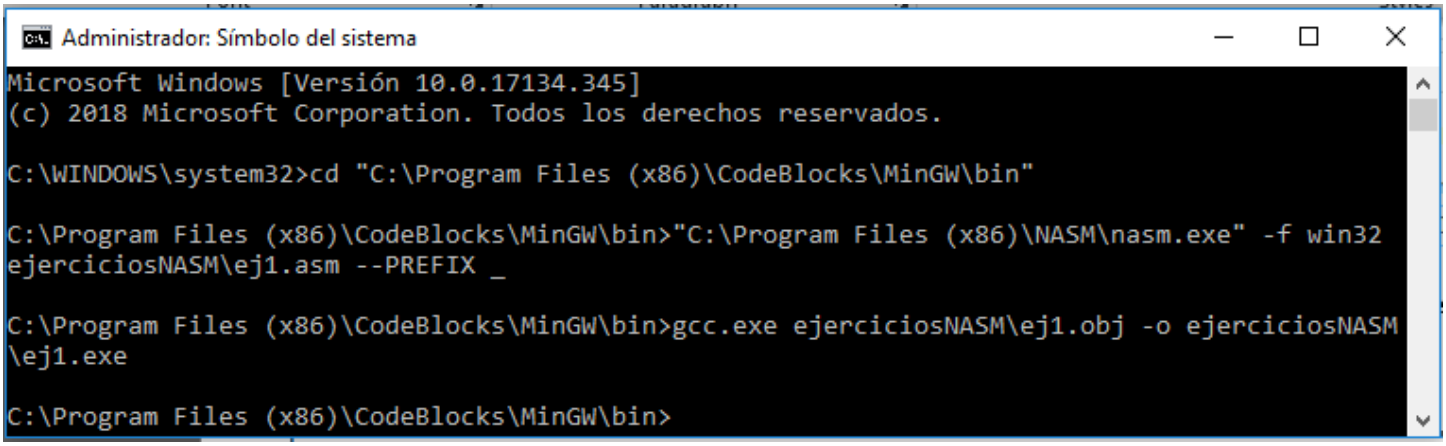


Con lo que vemos en la carpeta `ejerciciosNASM` que se creó el archivo .obj:

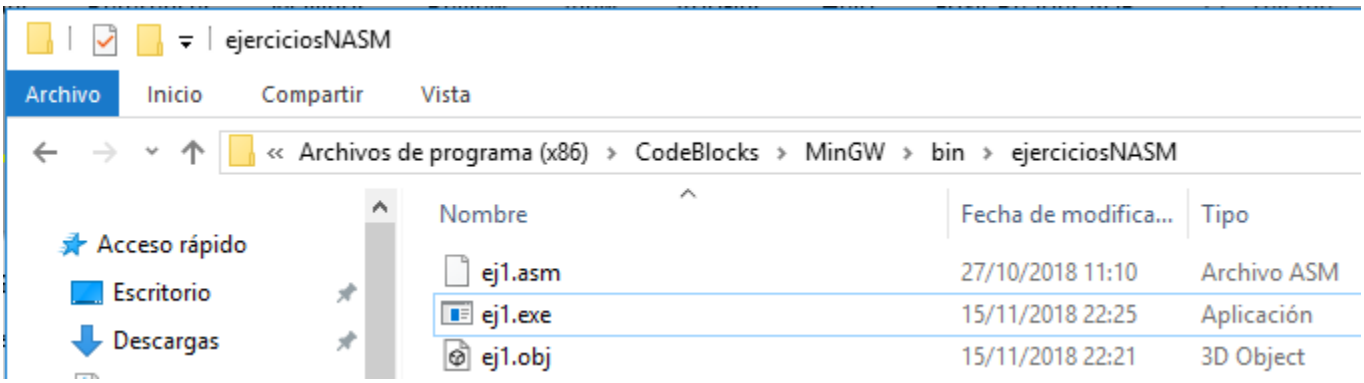


Luego sigue conectar/linkear las bibliotecas al archivo objeto creado. Para ello utilizaremos el linker del Codeblocks, que es el GCC, ejecutando el comando:

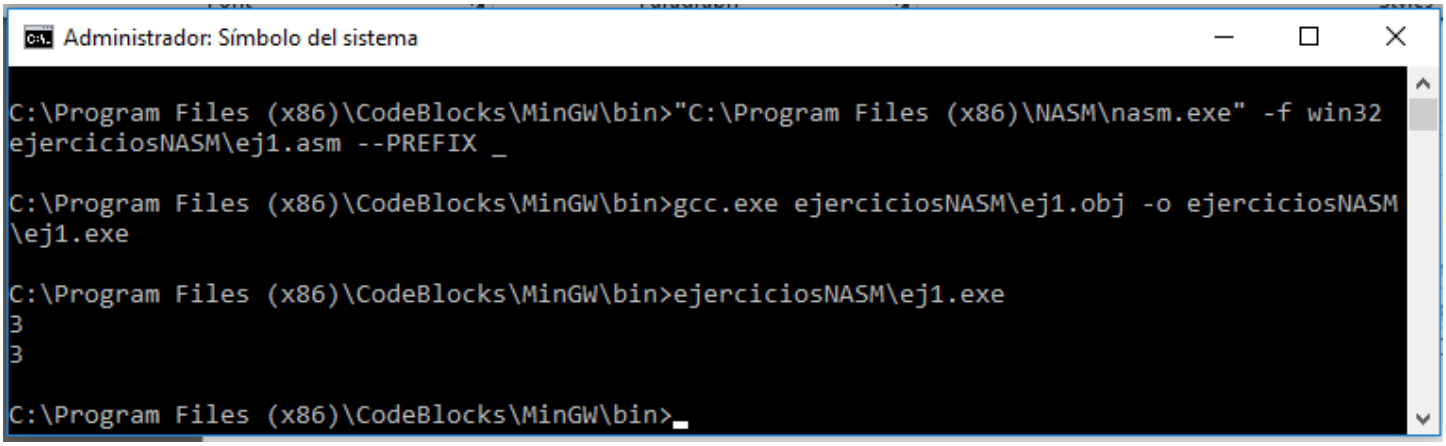
```
gcc.exe ejerciciosNASM\ej1.obj -o ejerciciosNASM\ej1.exe
```



Y con esto tenemos nuestro ejecutable:



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



Si se prefiere utilizar el Visual Studio, los comandos a utilizar son:

1) `nasm -f win32 ej3.asm --PREFIX _`

2) `link /out:ej3.exe ej3.obj libcmt.lib`

3) `ej3`

Y en GNU/Linux:

1) `nasm -f elf ej3.asm`

2) `ld -s -o ej3 ej3.o -lc -I /lib/ld-linux.so.2`

3) `./ej3`

En GNU/Linux de 64 bits (Previamente, en Ubuntu, hay que ejecutar: sudo apt-get install libc6-dev-i386):

1) `nasm -f elf ej3.asm`

2) `ld -m elf_i386 -s -o ej3 ej3.o -lc -I /lib/ld-linux.so.2`

3) `./ej3`

Notar que los comandos cambian según:

- la carpeta en la que estemos parados en la línea de comandos.
- la carpeta en la que se encuentran los archivos ejecutables y fuente (asm y obj).

A continuación, analizar los archivos `ejemplo1.asm` y `ejemplo2.asm` que se adjuntan al inicio de este documento y tras analizarlos, resolver los ejercicios de la sección siguiente.

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 bisiesto.
4. Se ingresan un entero N y, a continuación, N números enteros. La computadora muestra el promedio de los números pares ingresados y la suma de los impares.
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 horario.
8. Se ingresa una matriz de NxN componentes enteras. La computadora muestra la matriz transpuesta.

IMPORTANTE: estos ejercicios se deben presentar al momento de rendir el final.