Assembler de Intel

Objetivos de la clase

- ✔ Leer y escribir datos en memoria
- ✔ Realizar los primeros programas en ASM

Assembler - Sintáxis

Existen varias sintaxis para ASM, las más conocidas son:

- Intel
- AT & T

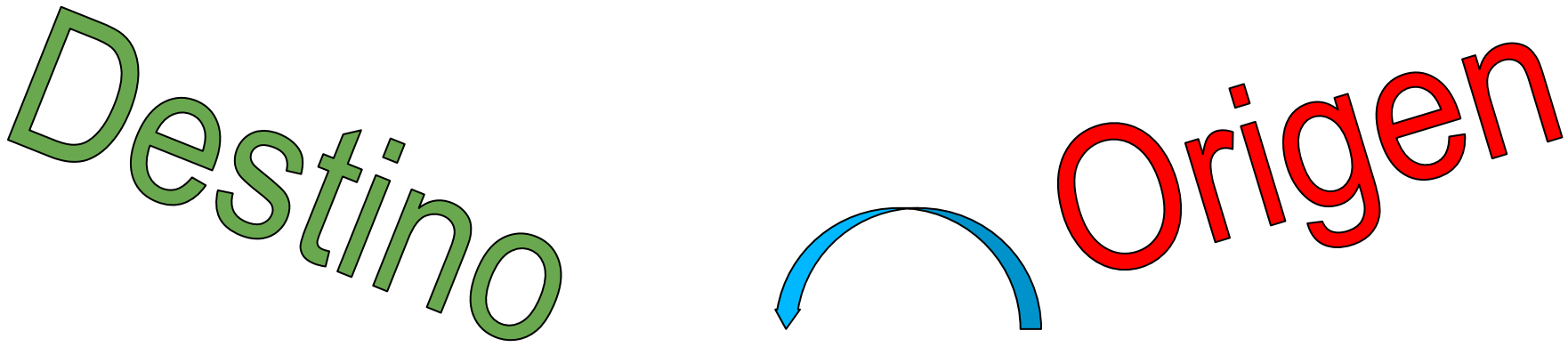
Veamos dos sentencias equivalentes con las dos sintaxis:

mov EAX, 1 ***(sintaxis Intel)***

movl $1, %eax ***(sintaxis AT&T)***

Tener en cuenta que el gcc por default genera salidas en sintaxis AT&T.

Assembler - Sintaxis



mov EAX, 1

Instrucciones

Flujo de bytes que interpretados por el procesador que realizan una acción

Instrucción: **add eax, 0x1**

Instrucción	contenido binario en mem.	contenido hexa en mem.
add eax, 0x1	1000 0011 1100 0000 0000 0001	83 c0 01

Registros de Intel

Ejemplos de uso con ASM:

```
mov     ah, 23

mov     bl, 99h

mov     ax, 1234h

mov     eax,12345678h

mov     rax,12345678ABCDEF00h
```

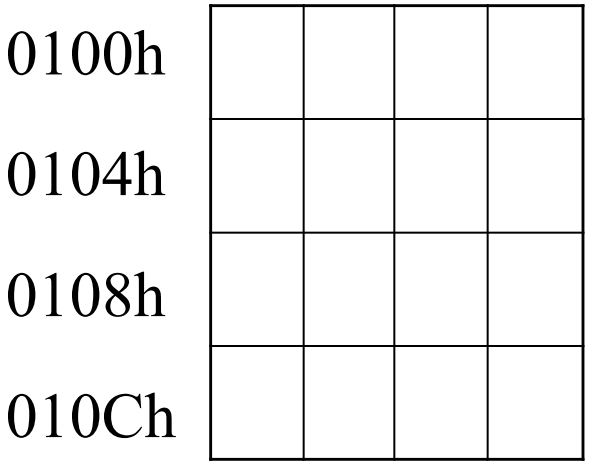
Lectura de memoria

```
mov     ax, [100h]

mov     ebx,[102h]

mov     cl,[109h]

mov     ax,[bx]
```

0100h	D0	12	00	11
0104h	00	3A	07	FF
0108h	32	B8	C0	C1
010Ch	74	7E	E2	AE

Escritura en memoria

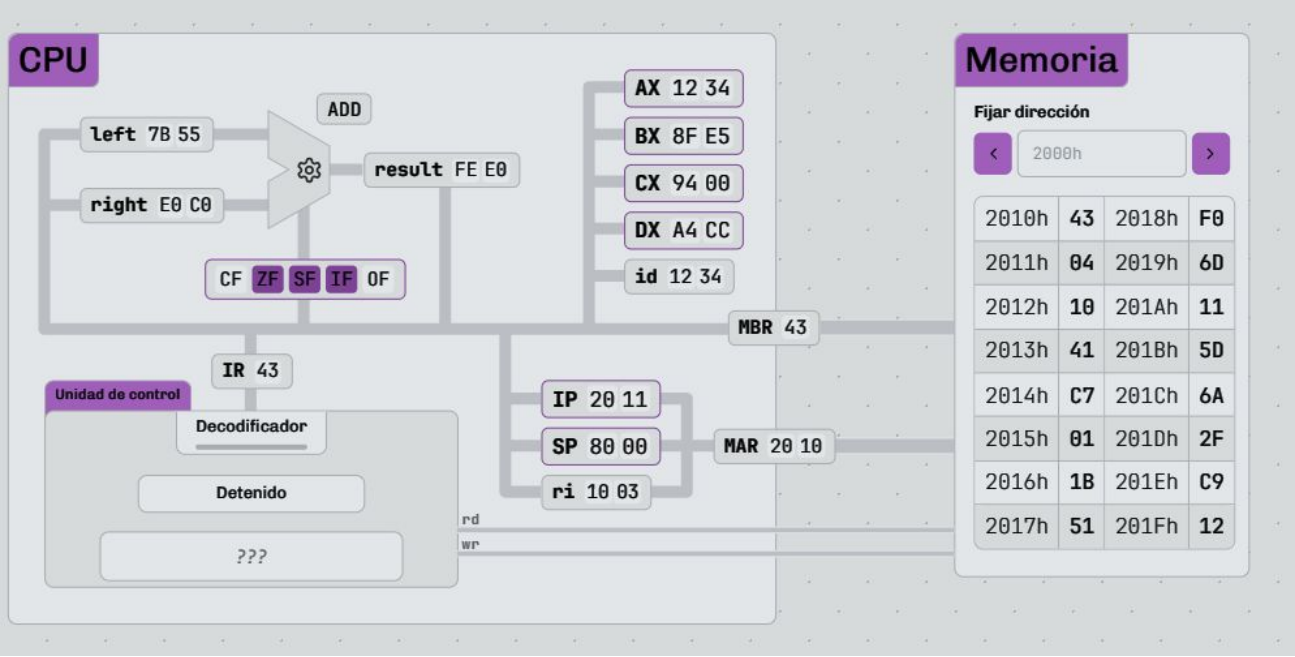
```
Mov     [102h],eax

Mov     [104h],bl

Mov     [108], rcx
```



Veamos en el simulador (VonSim)

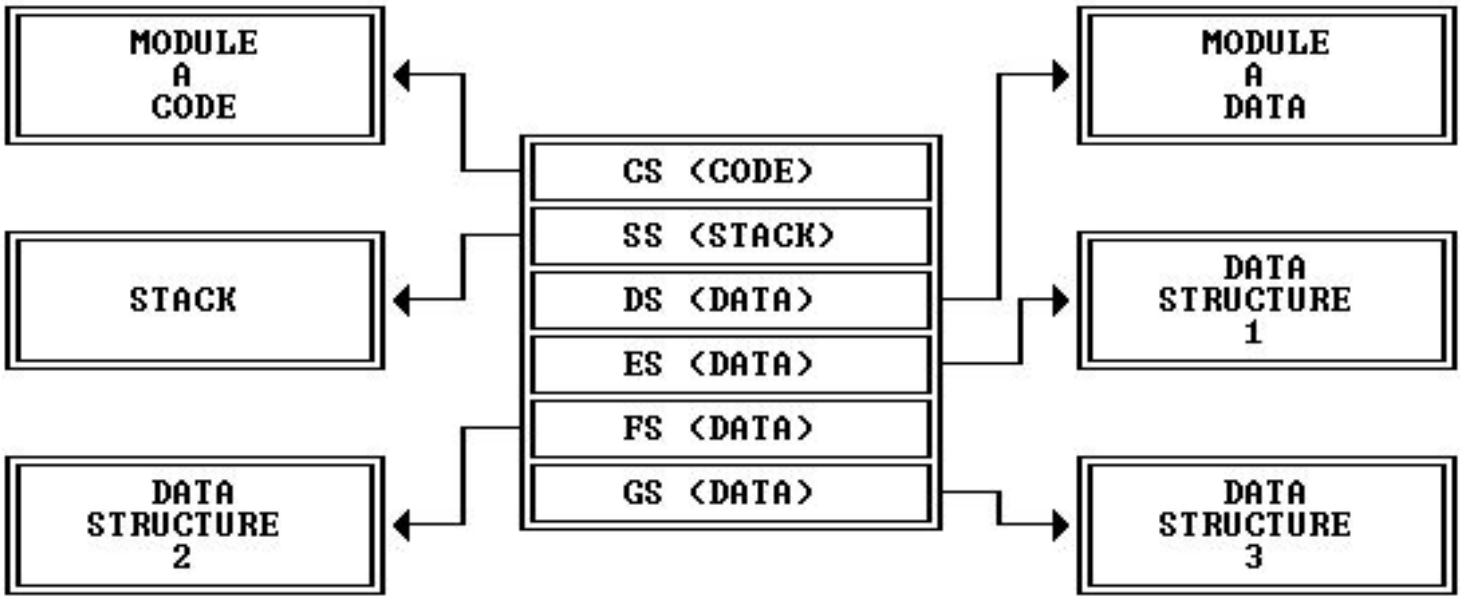
```
ORG 2000h

START:
        mov CL, 00H
        mov [1000h],CL
        mov AX,1234h
        mov [1002h],ax
        mov BX,[1004h]

END
```

Código



Segmentación de memoria en 80386

Figure 2-6. Use of Memory Segmentation



Recordar pares de punteros para acceder a memoria

Modos de direccionamiento

Como en otros procesadores tendremos la sintaxis general será:

Instrucción destino , fuente

Direccionamiento inmediato

```
mov  ax, 0ffffh
```

Direccionamiento de registro

```
mov     edx,eax
mov     ah,al
```

Modos de direccionamiento

Direccionamiento directo o absoluto

```
mov     ax, [57D1h]

mov     ebx, es:[42c9h]
```

Direccionamiento indirecto

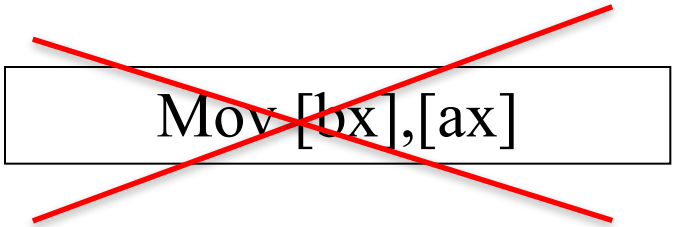
```
mov     cx, [bp]

mov     es:[di],ax
```

Direccionamiento con índice o indexado

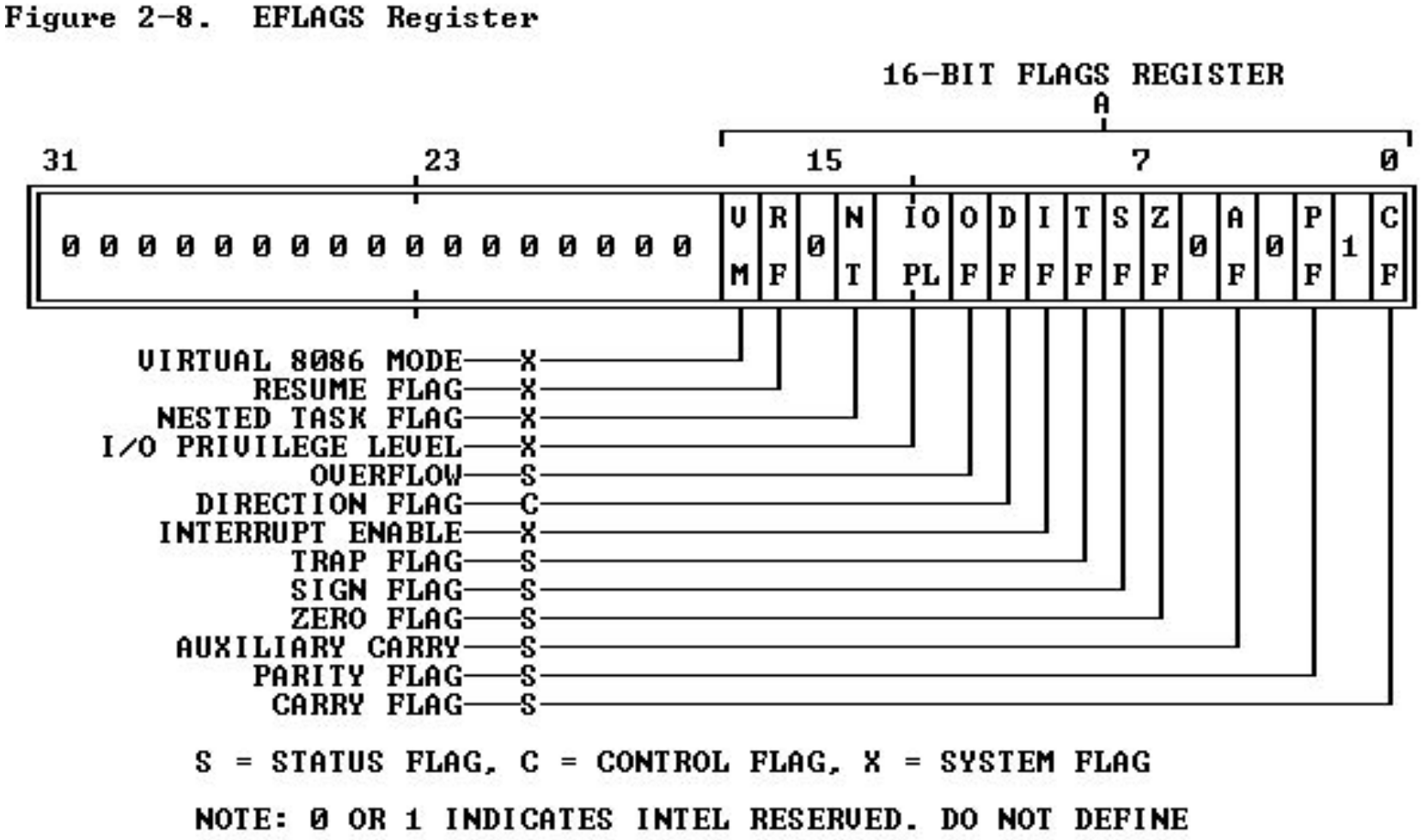
```
mov     cx, [bp+4]

mov     es:[di+8],ax
```

Recordatorio

NO existe el movimiento de datos de memoria a memoria en una sola instrucción:



Registro de Flags



Assembler – Ejemplo 1 – teoej1.asm

```
section .text
        global _start

_start:

        mov     dx,0FFh
        mov     bx,20h
        add     dx,bx
        push    dx
        push    4
        pop     cx

Ciclo:
        inc     bx
        dec     cx
        jnz     Ciclo

        mov     eax,parametros

        mov     AH,[parametros]
        mov     BL,[parametros+1]
        add     ah,bl
        mov     [salida],ah

        ret     0

section .data
parametros      db    11h,12h,13h
salida          db    0
```

Assembler – Ejemplo 1

Para compilar en Linux 64 bits:

nasm –f elf64 teoej1.asm -o teoej1.o

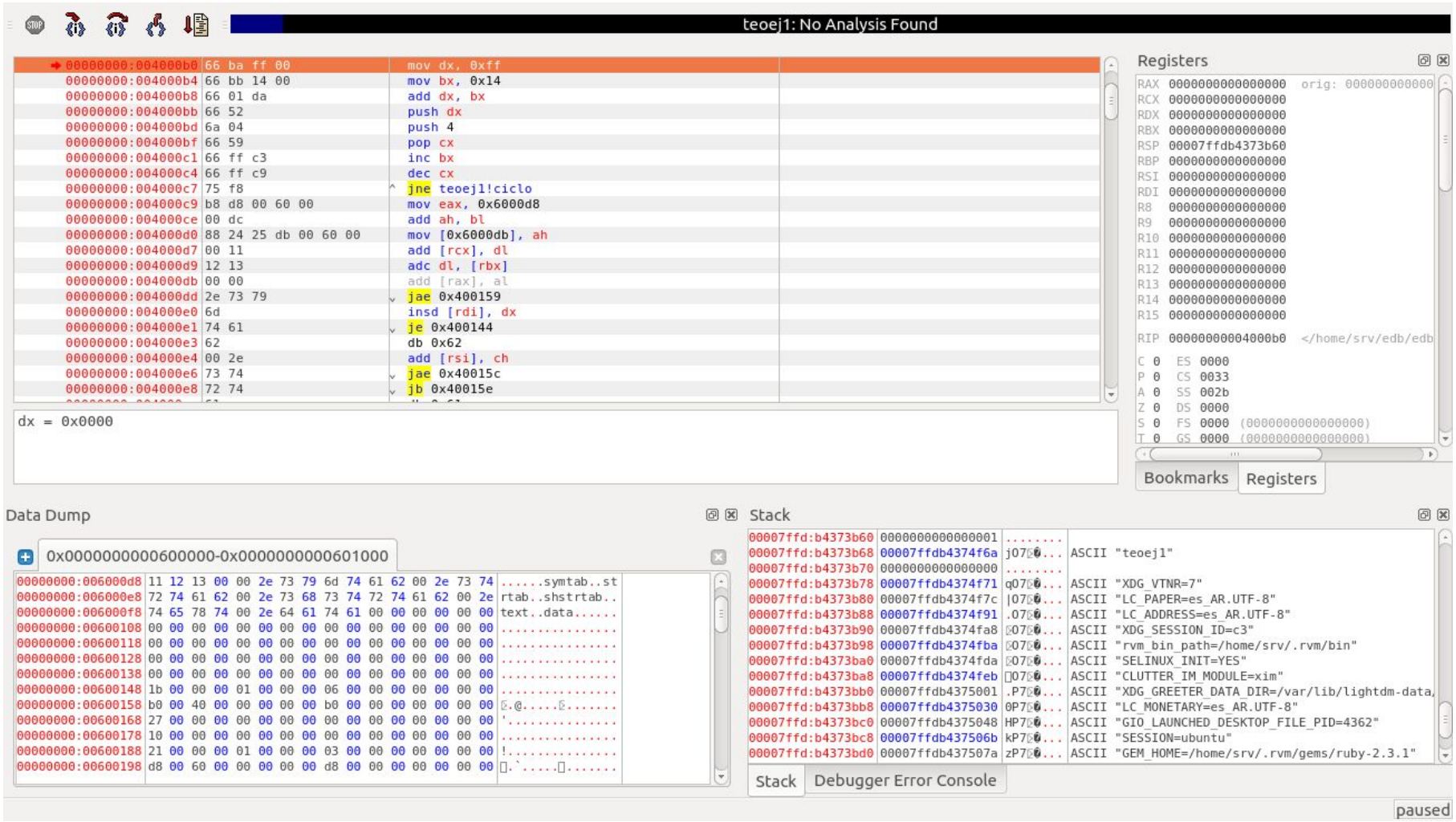
Para linkeditar:

ld teoej1.o -o teoej11

Genera el archivo ejecutable **teoej1** como salida

Al abrirlo con Evans Debugger.....

Assembler – Ejemplo 1 - DBG



Assembler – Ejemplo 1 - Codigo

```
00000000:004000b0 66 ba ff 00              mov dx, 0xff
00000000:004000b4 66 bb 14 00              mov bx, 0x14
00000000:004000b8 66 01 da                 add dx, bx
00000000:004000bb 66 52                    push dx
00000000:004000bd 6a 04                    push 4
00000000:004000bf 66 59                    pop cx
00000000:004000c1 66 ff c3                 inc bx
00000000:004000c4 66 ff c9                 dec cx
00000000:004000c7 75 f8                  ^ jne teoej1!ciclo
00000000:004000c9 b8 d8 00 60 00           mov eax, 0x6000d8
00000000:004000ce 00 dc                    add ah, bl
00000000:004000d0 88 24 25 db 00 60 00     mov [0x6000db], ah
00000000:004000d7 00 11                    add [rcx], dl
00000000:004000d9 12 13                    adc dl, [rbx]
00000000:004000db 00 00                    add [rax], al
00000000:004000dd 2e 73 79               v jae 0x400159
00000000:004000e0 6d                       insd [rdi], dx
00000000:004000e1 74 61                  v je 0x400144
00000000:004000e3 62                       db 0x62
00000000:004000e4 00 2e                    add [rsi], ch
00000000:004000e6 73 74                  v jae 0x40015c
00000000:004000e8 72 74                  v jb 0x40015e
```

```
dx = 0x0000
```

Assembler – Ejemplo 1 - Datos

Data Dump

0x0000000000600000-0x0000000000601000

```
00000000:006000d8 11 12 13 00 00 2e 73 79 6d 74 61 62 00 2e 73 74 ......symtab..st
00000000:006000e8 72 74 61 62 00 2e 73 68 73 74 72 74 61 62 00 2e rtab..shstrtab..
00000000:006000f8 74 65 78 74 00 2e 64 61 74 61 00 00 00 00 00 00 text..data......
00000000:00600108 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000000:00600118 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000000:00600128 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000000:00600138 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000000:00600148 1b 00 00 00 01 00 00 00 06 00 00 00 00 00 00 00 ................
00000000:00600158 b0 00 40 00 00 00 00 00 b0 00 00 00 00 00 00 00 □.@.....□.......
00000000:00600168 27 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 '...............
00000000:00600178 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00000000:00600188 21 00 00 00 01 00 00 00 03 00 00 00 00 00 00 00 !...............
00000000:00600198 d8 00 60 00 00 00 00 00 d8 00 00 00 00 00 00 00 □.`.....□.......
```

Assembler – Ej 1 – TP 2

```
section .text
GLOBAL _start

_start:
        mov ecx, cadena             ; Puntero a la cadena
        mov edx, longitud           ; Largo de la cadena
        mov ebx, 1                  ; FileDescriptor (STDOUT)
        mov eax, 4                  ; ID del Syscall WRITE
        int 80h                     ; Ejecución de la llamada

        mov eax, 1                  ; ID del Syscall EXIT
        mov ebx, 0                  ; Valor de Retorno
        int 80h                     ; Ejecución de la llamada


section .data
cadena db "Hola Mundo!!", 10        ;"Hola Mundo!!\n"
longitud equ $-cadena

section .bss
        placeholder resb 10
```

Assembler – Linkeditar con gcc

Cuando veamos ASM + C

Podemos usar el GCC para linkeditar:

Pero la función **start** se debe llamar **main:**

```
section .text
GLOBAL main

main:
        mov ecx, cadena         ; Puntero a la cadena
        mov edx, longitud       ; Largo de la cadena
        .........
```

Assembler – Linkeditar con gcc

Para compilar: *nasm –f elf64 teoej1forc.asm -o teoej1forc.o*
Para linkeditar : *gcc teoej1forc.o -o teoej1forc*

Los archivos ejecutables tienen distintos tamaños.

¿Por que?

```
-rwxrwxr-x 1 srv srv 8572 ago 13 10:06 teoej1forc
-rw-rw-r-- 1 srv srv  928 ago 13 09:58 teoej1forc.o
-rw-rw-r-- 1 srv srv  268 ago 13 09:58 teoej1forc.asm
-rw-rw-r-- 1 srv srv  928 ago 12 18:49 teoej1.o
-rwxrwxr-x 1 srv srv 1003 ago 12 18:18 teoej1
-rw-rw-r-- 1 srv srv  272 ago 12 18:17 teoej1.asm
```

Assembler – Linkeditar con gcc

Al abrir el ejecutable **teoej1forc** con el debugger vemos que tiene otro código al comienzo. ¿Donde está mi código ASM?

```
00007f15:68faf2d0 48 89 e7                mov rdi, rsp
00007f15:68faf2d3 e8 68 37 00 00          call ld-2.19.so!_dl_start
00007f15:68faf2d8 49 89 c4                mov r12, rax
00007f15:68faf2db 8b 05 17 1b 22 00       mov eax, [rel 0x7f15691d0df8]
00007f15:68faf2e1 5a                      pop rdx
00007f15:68faf2e2 48 8d 24 c4             lea rsp, [rsp+rax*8]
00007f15:68faf2e6 29 c2                   sub edx, eax
00007f15:68faf2e8 52                      push rdx
00007f15:68faf2e9 48 89 d6                mov rsi, rdx
00007f15:68faf2ec 49 89 e5                mov r13, rsp
00007f15:68faf2ef 48 83 e4 f0             and rsp, 0xfffffffffffffff0
00007f15:68faf2f3 48 8b 3d 66 1d 22 00    mov rdi, [rel 0x7f15691d1060]
00007f15:68faf2fa 49 8d 4c d5 10          lea rcx, [r13+rdx*8+0x10]
00007f15:68faf2ff 49 8d 55 08             lea rdx, [r13+8]
00007f15:68faf303 31 ed                   xor ebp, ebp
00007f15:68faf305 e8 76 ee 00 00          call ld-2.19.so!_dl_init_internal
00007f15:68faf30a 48 8d 15 1f f2 00 00    lea rdx, [rel 0x7f1568fbe530]
00007f15:68faf311 4c 89 ec                mov rsp, r13
00007f15:68faf314 41 ff e4                jmp r12
00007f15:68faf317 66 0f 1f 84 00 00 00 0… nop word [rax+rax]
00007f15:68faf320 48 8d 05 d9 2c 22 00    lea rax, [rel 0x7f15691d2000]
00007f15:68faf327 c3                      ret
```