Secciones, etiquetas y símbolos

Un programa en general se separa en secciones

- `.data`: Donde declarar variables globales inicializadas. (DB, DW, DD y DQ).
- `.rodata`: Donde declarar constantes globales inicializadas. (DB, DW, DD y DQ).
- `.bss`: Donde declarar variables globales no inicializadas. (RESB, RESW, RESD y RESQ).
- `.text`: Es donde se escribe el código.

Etiquetas y símbolos

- `global`: Modificador que define un símbolo que va a ser visto externamente.
- `_start`: Símbolo utilizando como punto de entrada de un programa.

Pseudoinstrucciones

Comandos e instrucciones para el ensamblador

- `DB`, `DW`, `DD`, `DQ`, `RESB`, `RESW`, `RESD` y `RESQ`.
- expresión `$`, se evalúa en la posición en memoria al principio de la línea que contiene la expresión.
- comando `EQU`, para definir constantes que después no quedan en el archivo objeto.
- comando `INCBIN`, incluye un binario en un archivo assembler.
- prefijo `TIMES`, repite una cantidad de veces la instrucción que le sigue.

Llamadas al sistema operativo (syscalls)

Utilizando la famosa `int 0x80` (en Linux) solicitamos al Sistema Operativo que haga algo por nosotros.

Su interfaz es:

1- El número de función que queremos en rax

2- Los parámetros en rbx, rcx, rdx, rsi, rdi y rbp; en ese orden

3- Llamamos a la interrupción del sistema operativo (int 0x80)

4- En general, la respuesta está en rax

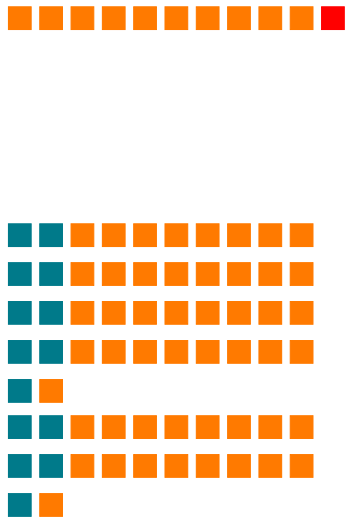
- Mostrar por pantalla (**sys_write**):
  Función **4**
  Parámetro 1: **¿donde?** (1 = stdout)
  Parámetro 2: **Dirección de memoria del mensaje**
  Parámetro 3: **Longitud del mensaje** (en bytes)

- Terminar programa (**exit**):
  Función **1**
  Parámetro 1: **código de retorno** (0 = sin error)

Hola Mundo... solución

```
section .data
  msg: DB 'Hola Mundo', 10
  largo EQU $ - msg

  global _start
section .text
  _start:
    mov rax, 4      ; funcion 4
    mov rbx, 1      ; stdout
    mov rcx, msg    ; mensaje
    mov rdx, largo  ; longitud
    int 0x80
    mov rax, 1      ; funcion 1
    mov rbx, 0      ; codigo
    int 0x80
```



Ensamblando y linkeando

Ensamblamos:

```
nasm -f elf64 -g -F DWARF holamundo.asm
```

Linkeamos:

```
ld -o holamundo holamundo.o
```

Ejecutamos:

```
./holamundo
```

GDB

Comandos Básicos

`r | run`	Ejecuta el programa hasta el primer break
`b | break FILE:LINE`	Breakpoint en la línea
`b | break FUNCTION`	Breakpoint en la función
`info breakpoints`	Muestra información sobre los breakpoints
`c | continue`	Continúa con la ejecución
`s | step`	Siguiente línea (Into)
`n | next`	Siguiente línea (Over)
`si | stepi`	Siguiente instrucción asm (Into)
`ni | nexti`	Siguiente instrucción asm (Over)
`x/Nuf ADDR`	Muestra los datos en memoria

- `N` = Cantidad (bytes)
- `u` = Unidad `b|h|w|g`
  - `b:byte, h:word, w:dword, g:qword`
- `f` = Formato `x|d|u|o|f|a`
  - x:hex, d:decimal, u:decimal sin signo, o:octal, f:float, a:direcciones, s:strings, i:inst.

GDB - Mostrar memoria

`x/Nuf ADDR`

N = Cantidad (bytes)
u = Unidad b|h|w|g
`b:byte, h:word, w:dword, g:qword`
f = Formato x|d|u|o|f|a
x:hex, d:decimal, u:decimal sin signo, o:octal, `f`:float,
a:direcciones, s:strings, `i`:instrucciones.

Ejemplos

- `x/3bx` addr : Tres bytes en hexadecimal.
- `x/5wd` addr : Cinco enteros de 32 bits con signo.
- `x/1ho` addr : Un número de 16 bits en representación octal.
- `x/10gf` addr : Diez doubles.
- `x/s` addr : Una string terminada en cero.

GDB

Configuración de GDB:

```
~/.gdbinit
```

Para usar sintaxis intel y guardar historial de comandos:

```
set disassembly-flavor intel
set history save
```

Correr GDB con argumentos:

```
gdb --args <ejecutable> <arg1> <arg2> ...
```