Hola Mundo en ASM

David Alejandro González Márquez

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Hola Mundo ...

Ejercicio

Escriba un programa en lenguaje ensamblador que imprima por pantalla:

```
Hola Mundo
```

Hola Mundo ...

Ejercicio

Escriba un programa en lenguaje ensamblador que imprima por pantalla:

```
Hola Mundo
```

¿Cómo?

Secciones, etiquetas y símbolos

Un programa en general se separa en secciones

Secciones, etiquetas y símbolos

Un programa en general se separa en secciones

- `.data`: Donde declarar variables globales inicializadas. (`DB`, `DW`, `DD` y `DQ`).

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`).

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`).

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.

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

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.

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

Pseudoinstrucciones

Comandos e instrucciones para el ensamblador

- DB, DW, DD, DQ, RESB, RESW, RESD y RESQ.

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.

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.

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.

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.

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

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)

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

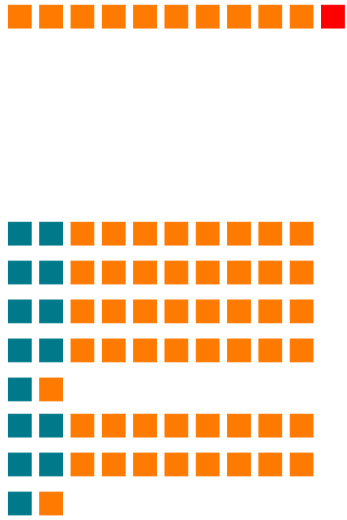
```
section .data
  msg: DB 'Hola Mundo', 10
  largo EQU $ - msg

  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
```

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
```

Ejecución

```
$ ./holamundo
```

Ejecución

```
$ ./holamundo
Hola Mundo
```

Ejecución

```
$ ./holamundo
Hola Mundo
```

¿Dondé quedo el código de error?

Ejecución

```
$ ./holamundo
Hola Mundo
```

¿Dondé quedo el código de error?

```
$ echo "$?"
```

Ejecución

```
$ ./holamundo
Hola Mundo
```

¿Dondé quedo el código de error?

```
$ echo "$?"
0
```

Ejecución

```
$ ./holamundo
Hola Mundo
```

¿Dondé quedo el código de error?

```
$ echo "$?"
0
```

¿Cómo reconozco un binario?

Ejecución

```
$ ./holamundo
Hola Mundo
```

¿Dondé quedo el código de error?

```
$ echo "$?"
0
```

¿Cómo reconozco un binario?

```
$ file holamundo
```

Ejecución

```
$ ./holamundo
Hola Mundo
```

¿Dondé quedo el código de error?

```
$ echo "$?"
0
```

¿Cómo reconozco un binario?

```
$ file holamundo
holamundo: ELF 64-bit LSB executable, x86-64, version 1 (SYSV),
 statically linked, with debug_info, not stripped
```

Ejecución

Ya sabemos que es un `elf`. Pero ¿Qué tiene dentro?

Ejecución

Ya sabemos que es un `elf`. Pero ¿Qué tiene dentro?

```
$ objdump -d holamundo
```

Ejecución

Ya sabemos que es un `elf`. Pero ¿Qué tiene dentro?

```
$ objdump -d holamundo

holamundo:     file format elf64-x86-64

Disassembly of section .text:

00000000004000b0 <_start>:
  4000b0: b8 04 00 00 00         mov    $0x4,%eax
  4000b5: bb 01 00 00 00         mov    $0x1,%ebx
  4000ba: 48 b9 d8 00 60 00 00   movabs $0x6000d8,%rcx
  4000c1: 00 00 00
  4000c4: ba 0b 00 00 00         mov    $0xb,%edx
  4000c9: cd 80                  int    $0x80
  4000cb: b8 01 00 00 00         mov    $0x1,%eax
  4000d0: bb 00 00 00 00         mov    $0x0,%ebx
  4000d5: cd 80                  int    $0x80
```

Ejecución

Y la sección de datos

Ejecución

Y la sección de datos

```
$ objdump -D holamundo
```

Ejecución

Y la sección de datos

```
$ objdump -D holamundo

...

Disassembly of section .data:

00000000006000d8 <msg>:
  6000d8: 48 6f                   rex.W outsl %ds:(%rsi),(%dx)
  6000da: 6c                      insb   (%dx),%es:(%rdi)
  6000db: 61                      (bad)
  6000dc: 20 4d 75                and    %cl,0x75(%rbp)
  6000df: 6e                      outsb  %ds:(%rsi),(%dx)
  6000e0: 64 6f                   outsl  %fs:(%rsi),(%dx)
  6000e2: 0a                      .byte 0xa

...
```

Ejecución

Y si quiero encontrar una cadena de texto

Ejecución

Y si quiero encontrar una cadena de texto

```
$ strings holamundo
```

Ejecución

Y si quiero encontrar una cadena de texto

```
$ strings holamundo

Hola Mundo
holamundo.asm
NASM 2.13.02
...
largo
...
.symtab
.strtab
...
.text
.data
.debug_aranges
.debug_pubnames
.debug_info
...
```

Ejecución

No confundir con los simbolos dentro de un binario.

Ejecución

No confundir con los simbolos dentro de un binario.

```
$ nm holamundo
```

Ejecución

No confundir con los simbolos dentro de un binario.

```
$ nm holamundo
```

```
00000000006000e3 D __bss_start
00000000006000e3 D _edata
00000000006000e8 D _end
000000000000000b a largo
00000000006000d8 d msg
00000000004000b0 T _start
```

Ejecución

No confundir con los simbolos dentro de un binario.

```
$ nm holamundo
```

```
00000000006000e3 D __bss_start
00000000006000e3 D _edata
00000000006000e8 D _end
000000000000000b a largo
00000000006000d8 d msg
00000000004000b0 T _start
```

¿Y donde se aprende como usar estos bonitos comandos?

Ejecución

No confundir con los simbolos dentro de un binario.

```
$ nm holamundo
```

```
00000000006000e3 D __bss_start
00000000006000e3 D _edata
00000000006000e8 D _end
000000000000000b a largo
00000000006000d8 d msg
00000000004000b0 T _start
```

¿Y donde se aprende como usar estos bonitos comandos?

En el manual de referencia.

Ejecución

No confundir con los simbolos dentro de un binario.

```
$ nm holamundo
```

```
00000000006000e3 D __bss_start
00000000006000e3 D _edata
00000000006000e8 D _end
000000000000000b a largo
00000000006000d8 d msg
00000000004000b0 T _start
```

¿Y donde se aprende como usar estos bonitos comandos?

En el manual de referencia.

¿Cómo accedo al manual?

Ejecución

No confundir con los simbolos dentro de un binario.

```
$ nm holamundo
```

```
00000000006000e3 D __bss_start
00000000006000e3 D _edata
00000000006000e8 D _end
000000000000000b a largo
00000000006000d8 d msg
00000000004000b0 T _start
```

¿Y donde se aprende como usar estos bonitos comandos?

En el manual de referencia.

¿Cómo accedo al manual?

Mediante el comando man

```
$ man comando
```

Ejecución

```
$ man nm
```

```
NM(1)                                   GNU Development Tools                                   NM(1)

NAME
       nm - list symbols from object files

SYNOPSIS
       nm [-A|-o|--print-file-name] [-a|--debug-syms]
          [-B|--format=bsd] [-C|--demangle[=style]]
          [-D|--dynamic] [-fformat|--format=format]
          [-g|--extern-only] [-h|--help]
          [-l|--line-numbers] [--inlines]
          [-n|-v|--numeric-sort]
          [-P|--portability] [-p|--no-sort]
          [-r|--reverse-sort] [-S|--print-size]
          [-s|--print-armap] [-t radix|--radix=radix]
          [-u|--undefined-only] [-V|--version]
          [-X 32_64] [--defined-only] [--no-demangle]
          [--plugin name] [--size-sort] [--special-syms]
          [--synthetic] [--with-symbol-versions] [--target=bfdname]
          [objfile...]

DESCRIPTION
       GNU nm lists the symbols from object files objfile....  If no object files are listed as
       arguments, nm assumes the file a.out.

       For each symbol, nm shows:

       ·   The symbol value, in the radix selected by options (see below), or hexadecimal by default.
```

Debugger

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.

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.

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.

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.

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.

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> ...
```

GDB

```
$ gdb holamundo
```

GDB

```
$ gdb holamundo

GNU gdb (Ubuntu 8.1-0ubuntu3.2) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from holamundo...done.
(gdb)
```

GDB

```
(gdb)
```

GDB

```
(gdb) b _start
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4      ; funcion 4
(gdb) n
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb)
Hola Mundo
13     mov rax, 1     ; funcion 1
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb)
Hola Mundo
13     mov rax, 1     ; funcion 1
(gdb)
14     mov rbx, 0     ; codigo
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb)
Hola Mundo
13     mov rax, 1     ; funcion 1
(gdb)
14     mov rbx, 0     ; codigo
(gdb)
15     int 0x80
(gdb)
```

GDB

```
(gdb) b _start
Breakpoint 1 at 0x4000b0: file holamundo.asm, line 8.
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb)
Hola Mundo
13     mov rax, 1     ; funcion 1
(gdb)
14     mov rbx, 0     ; codigo
(gdb)
15     int 0x80
(gdb)
[Inferior 1 (process 20626) exited normally]
(gdb)
```

GDB

```
(gdb)
```

GDB

```
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb)
```

GDB

```
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb) x/s $rcx
```

GDB

```
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb) x/s $rcx
0x6000d8 <msg>: "Hola Mundo\n,"
(gdb)
```

GDB

```
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb) x/s $rcx
0x6000d8 <msg>: "Hola Mundo\n,"
(gdb) info reg
```

GDB

```
(gdb) r
Starting program: /holamundo
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) n
9     mov rbx, 1     ; stdout
(gdb)
10     mov rcx, msg   ; mensaje
(gdb)
11     mov rdx, largo ; longitud
(gdb)
12     int 0x80
(gdb) x/s $rcx
0x6000d8 <msg>: "Hola Mundo\n,"
(gdb) info reg
rax            0x4 4
rbx            0x1 1
rcx            0x6000d8 6291672
rdx            0xb 11
...
rsp            0x7fffffffdbf0 0x7fffffffdbf0
...
rip            0x4000c9 0x4000c9 <_start+25>
eflags         0x202 [ IF ]
...
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb)
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) list
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) list
3   largo EQU $ - msg
4
5   global _start
6 section .text
7   _start:
8     mov rax, 4     ; funcion 4
9     mov rbx, 1     ; stdout
10    mov rcx, msg   ; mensaje
11    mov rdx, largo ; longitud
12    int 0x80
(gdb)
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8     mov rax, 4     ; funcion 4
(gdb) list
3   largo EQU $ - msg
4
5   global _start
6 section .text
7   _start:
8     mov rax, 4     ; funcion 4
9     mov rbx, 1     ; stdout
10    mov rcx, msg   ; mensaje
11    mov rdx, largo ; longitud
12    int 0x80
(gdb) list
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8       mov rax, 4      ; funcion 4
(gdb) list
3    largo EQU $ - msg
4
5    global _start
6  section .text
7    _start:
8       mov rax, 4      ; funcion 4
9       mov rbx, 1      ; stdout
10      mov rcx, msg    ; mensaje
11      mov rdx, largo  ; longitud
12      int 0x80
(gdb) list
13      mov rax, 1      ; funcion 1
14      mov rbx, 0      ; codigo
15      int 0x80
(gdb)
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8       mov rax, 4      ; funcion 4
(gdb)
```

holamundo.asm

```
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
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8       mov rax, 4      ; funcion 4
(gdb)  x/12i $rip
```

holamundo.asm

```
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
```

GDB

```
Breakpoint 1, _start () at holamundo.asm:8
8       mov rax, 4      ; funcion 4
(gdb) x/12i $rip
=> 0x4000b0 <_start>:      mov    $0x4,%eax
   0x4000b5 <_start+5>:    mov    $0x1,%ebx
   0x4000ba <_start+10>:   movabs $0x6000d8,%rcx
   0x4000c4 <_start+20>:   mov    $0xb,%edx
   0x4000c9 <_start+25>:   int    $0x80
   0x4000cb <_start+27>:   mov    $0x1,%eax
   0x4000d0 <_start+32>:   mov    $0x0,%ebx
   0x4000d5 <_start+37>:   int    $0x80
   0x4000d7: add    %cl,0x6f(%rax)
   0x4000da: insb   (%dx),%es:(%rdi)
   0x4000db: (bad)
   0x4000dc: and    %cl,0x75(%rbp)
(gdb)
```

holamundo.asm

```
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
```

Bibliografía: Fuentes y material adicional

- Convenciones de llamados a función en x86:
  `https://en.wikipedia.org/wiki/X86_calling_conventions`
- Notas sobre System V ABI:
  `https://wiki.osdev.org/System_V_ABI`
- Documentación de NASM:
  `https://nasm.us/doc/`
- Artículo sobre el flag `-pie`:
  `https://eklitzke.org/position-independent-executables`
- Documentación de System V ABI:
  `https://uclibc.org/docs/psABI-x86_64.pdf`
- Manuales de Intel:
  `https://software.intel.com/en-us/articles/intel-sdm`

¡Gracias!

Recuerden leer los comentarios al final de este video por aclaraciones o fe de erratas.