

¿Cual es el concepto de palabra que maneja Intel? Dd significa double word. Cada palabra tiene 4 bytes, 32 bits. Así como específicas dd, puedes poner db (data byte). Ahí siempre pones cuanta memoria reservas para el campo. Los que tienen dos puntos son etiquetas.

Caso de estudio: Intel

Isa: registros

Hay varios tipos de registros. El registro A se utiliza para instrucciones aritmeticas y logicas. El registro B es el base y se suele utilizar para direccionamiento de operandos. El registro C se utiliza para aritmetica o como contador. El D se suele utilizar cuando necesitas una dupla de registros, porque con uno no alcanza.

Para los registros generales, cada registro tiene 16 bits y se subdivide en dos subregistros. Cada subregistro tiene 8 bits. Puedes acceder al registro general como a los subregistros.

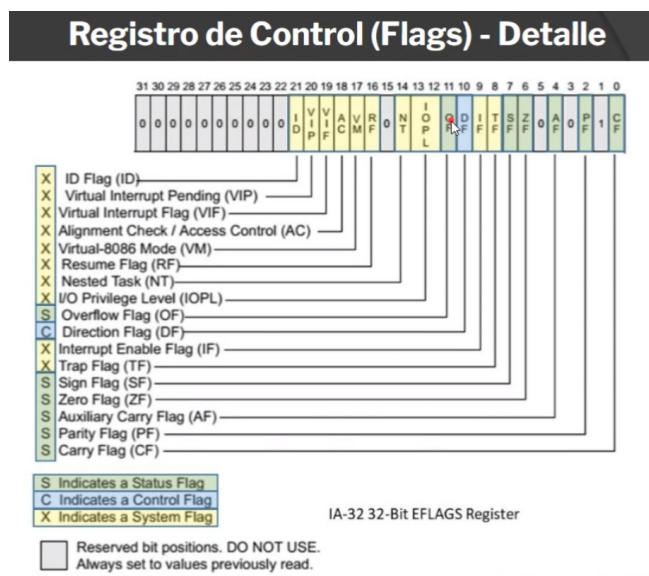
Pueden estar extendidos a 32 bits, pero aun se puede acceder a los de 16. También puedes tener más generales, con registros de 64 bits. Se puede trabajar con cualquiera de ellos.

Cuando se va al registro general, se pueden subdividir en mitades, cuartos y octavos (32, 16 y 8 bits).

Después tenés los registros índices, que se suelen usar para operaciones de manejos de cadenas, para apuntar al operando “origen”. Vos tenés partes bajas y altas, las más bajas son las mas chicas. También puedes acceder al destino.

El registro de la memoria Stack (pila) tenés las direcciones ordenadas mayor a menor, teniendo la menor en el tope. Tienes un puntero a la base de la pila y al tope. A mayor tope, menor es la dirección.

El registro de instrucción y control contiene la dirección de la próxima instrucción a ejecutar. Los flags asociados a cada registro me indica el estado del mismo que me sirve para tomar decisiones según su valor. De todos los bits que tiene, enciendes o apagas un registro según lo que haya pasado.



Direccionamiento

Se trata de que forma accedes a los datos a la hora de operar. Si está implícito, en el mismo código de operación se encuentra el dato. Por ejemplo si se usa CBW, se da por sentado que es el que está en el registro A.

Otro modo de operación es a partir de acceder a registros, donde uno explicita que registro será.

Cuando se usa un operando inmediato, este se aloja / se define en la misma instrucción.

Un operador directo es cuando a partir de corchetes y el nombre de una variable accedo a un valor,

- **Directo:** El dato está en memoria referenciado por el nombre de un campo

Ej. `MOV RAX, [VARIABLE]`

Uno indirecto:

- **Registro Indirecto:** El dato está en memoria apuntado por un registro base o índice.

Ej. `MOV EAX, [EBX]`

Relativo:

- **Registro Relativo:** El dato está en memoria apuntado por un registro base o índice más un desplazamiento.

Ej. `MOV RAX, [RBX+4]`
`MOV RAX, [VECTOR+RBX]`

Siempre sumo un numero o variable

Base + índice:

- **Base + Índice:** El dato está en memoria apuntado por un registro base más un registro índice.

Ej. `MOV [RBX+RDI], CL`

Sumo dos registros.

- **Base Relativo + Índice:** El dato está en memoria apuntado por un registro base más un registro índice más un desplazamiento.

Ej. `MOV RAX, [RBX+RDI+4]`
`MOV RAX, [VECTOR+RBX+RDI]`

Memoria y tipos de dato

podes guardar ASCII, numero entero o decimal. La celda de memoria tiene 1 byte y la palabra en intel es de 2 bytes. Esto difiere de una infraestructura a otra. Podes tener dobles o cuádruples palabras.

Endiannes

Es el método aplicado para almacenar datos mayores a un byte en una computadora respecto a la dirección que se le asigna a cada uno de ellos en la memoria.

Existen 2 métodos:

- **Big-Endian:** determina que el orden en la memoria coincide con el orden lógico del dato.

“el dato final en la mayor dirección”

Ej. IBM Mainframe

- **Little-Endian :** es a la inversa, el dato inicial para la lógica se coloca en la mayor dirección y el dato final en la menor.

“el dato final en la menor dirección”

Intel trabaja en Little-Endian.

Endiannes - Ejemplos

- **Caso 1:** Definición de un área de memoria con contenido inicial definido en formato carácter

```
msg      db      'HOLA'
```

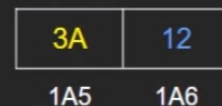


Aquí la posición de memoria que toma cada caracter recibido es la que por intuición uno asume, o sea, la letra 'H' en la dirección menor

Endiannes - Ejemplos

- **Caso 2:** Definición de un área de memoria con contenido inicial definido en formato numérico

```
num      dw 4666 ; es 123A en he
```



```
num2     dd 12345678h
```



Aquí es donde se observa la ubicación de los bytes con el método Little-Endian, el byte menos significativo se ubica en la dirección de memoria menor que el byte más significativo

Cuando se le agrega la h, el número está en base 16, por eso se guarda directamente como está escrito en el segundo ejemplo.

Endiannes - Ejemplos

- **Caso 3:** Se ejecuta una copia de memoria a registro

```
mov    AX, [msg]
```

```
mov    BX, [num1]
```

```
mov    ECX, [msg]
```

```
mov    EDX, [num2]
```

RAX						4F	48
RBX						12	3A
RCX					41	4C	4F
RCX					12	34	56

48	4F	4C	41
1A1	1A2	1A3	1A4
3A	12		
1A5	1A6		
48	4F	4C	41
1A1	1A2	1A3	1A4
78	56	34	12
1A7	1A8	1A9	1AA

La parte alta del registro contiene el byte de orden superior de memoria, y la parte baja del registro contiene el byte de orden inferior.

Moves según el tamaño del registro destino. Si vos tenés en la memoria little endian, se copia “al derecho” en el registro.

Ensamblador NASM

Estos son temas asociados a la herramienta que ensambla, no a la arquitectura.

Directivas al ensamblador / Pseudo-instrucciones

Son instrucciones para que el ensamblador tome alguna acción durante el proceso de ensamblado. No se traducen a código máquina.

section	indica el comienzo de un segmento
global	indica que una etiqueta declarada en el programa es visible para un programa externo
extern	indica que una etiqueta usada en el programa pertenece a un programa externo (donde habrá sido declarada global)
db, dw, dd, dq, dt	sirven para definir áreas de memoria (variables) con contenido inicial
resb, resw, resd, resq, rest	sirven para definir áreas de memoria (variables) sin contenido inicial
times	repite una definición la cantidad de veces que se indica
%macro %endmacro	indican el inicio y final de un bloque para definir una macro
%include	permite incluir el contenido de un archivo

Esto sirve para decirle que es cada cosa al propio ensamblador y así sepa que hacer con eso.

Componentes minimos:

Estructura de un programa

```
global main

section .data
;variables con contenido inicial

section .bss ;block starting symbol
;variables sin contenido inicial

section .text
;instrucciones
main:

ret
```

Definición y reserva de campos en memoria

- Con contenido inicial (en `section .data`)

<code>db</code>	define byte (1 byte)
<code>dw</code>	define word (2 bytes)
<code>dd</code>	define double (4 bytes)
<code>dq</code>	define quad (8 bytes)
<code>dt</code>	define ten (10 bytes)

- Sin contenido inicial (en `section .bss`)

<code>resb</code>	reserve byte (1 byte)
<code>resw</code>	reserve word (2 bytes)
<code>resd</code>	reserve double (4 bytes)
<code>resq</code>	reserve quad (8 bytes)
<code>rest</code>	reserve ten (10 bytes)

Definición y reserva de campos en memoria

● Ejemplos (2/4)

Definición	Bytes reservados	Contenido memoria
decimal1 db 11	1	0B
decimal2 dw -11	2	F5 FF
decimal3 dd 12345	4	39 30 00 00
decimal4 dq -1	8	FF FF FF FF FF FF FF FF
hexa1 db -0Bh	1	F5
hexa2 dw 0Ch	2	0C 00
hexa3 dd FFFFh	4	FF FF 00 00
hexa4 dq 96B43Fh	8	3F B4 96 00 00 00 00 00

En la memoria se guarda en base 16 y en little endian. Recordar que si tiene h, le estas pasando literalmente en base 16.

Definición y reserva de campos en memoria

● Ejemplos (3/4)

Definición	Bytes reservados	Contenido memoria
octal1 db 13o	1	0B
octal2 dw 71o	2	39 00
octal3 dd 10o	4	08 00 00 00
binario1 db 1011b	1	0B
binario2 dw 1011b	2	0B 00
binario3 dd -1000b	4	F8 FF FF FF
truncado db 2571 ;0A0Bh	1	0B
noTruncado dw 2571 ;0A0Bh	2	0B 0A

Definición y reserva de campos en memoria

● Ejemplos (4/4)

Definición		Bytes reservados	Contenido memoria
letra	db 'A'	1	41
letra2	dw 'A'	2	41 20
letra3	db 'a'	1	61
cadena	db 'hola'	4	68 6F 6C 61
cadena2	dw 'ola'	4	6F 6C 61 20
numero	db '12'	2	31 32
vector1	times 3 db 'A'	3	41 41 41
vector2	times 3 db 'A',0	6	41 00 41 00 41 00
registro	times 0 db 'A'	0	n/a

Para las cadenas soles usar una medida de “db”.

Macros e inclusión de archivos

Macros

Son secuencias de instrucciones asignadas a un nombre que pueden usarse en cualquier parte del programa.

La sintaxis es:

```
%macro macro_name number_of_params  
<macro body>  
%endmacro
```

Macros

Sin parámetros

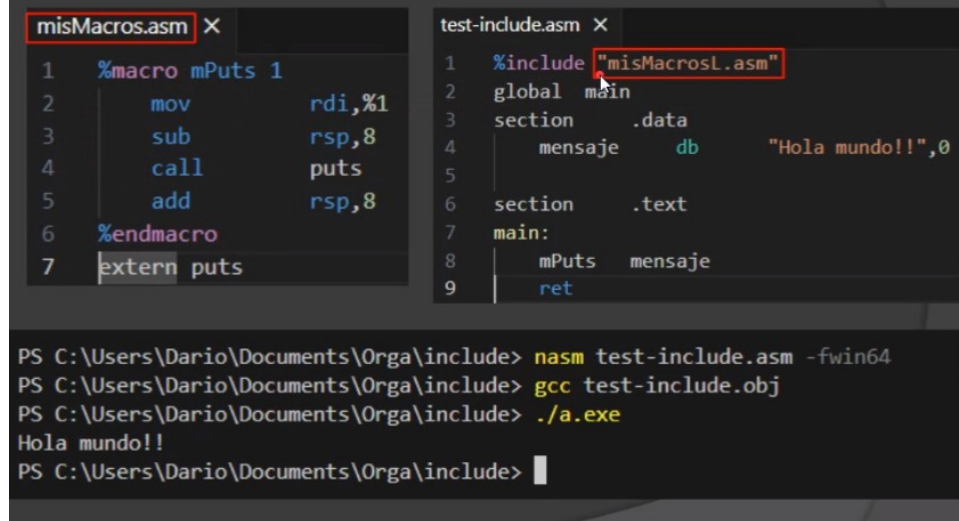
```
1 %macro mPuts 0  
2     sub     rsp,8  
3     call    puts  
4     add     rsp,8  
5 %endmacro  
6 global main  
7 extern puts  
8  
9 section     .data  
10     mensaje     db     "Hola mundo!",0  
11     mensaje2     db     "Chau!",0  
12  
13 section     .text  
14 main:  
15     mov     rdi,mensaje  
16     mPuts  
17     mov     rdi,mensaje2  
18     mPuts  
19     ret
```

Con 1 parámetro

```
1 %macro mPuts 1  
2     mov     rdi,%1  
3     sub     rsp,8  
4     call    puts  
5     add     rsp,8  
6 %endmacro  
7 global main  
8 extern puts  
9  
10 section     .data  
11     mensaje     db     "Hola mundo!",0  
12     mensaje2     db     "Chau!",0  
13  
14 section     .text  
15 main:  
16     mPuts     mensaje  
17     mPuts     mensaje2  
18     ret
```

No funciona como las funciones porque en memoria ocupa como si estuviera copiado dos veces. El ensamblador lo que hace es escribirlo por mí.

Inclusión de archivos



```
misMacros.asm X
1  %macro mPuts 1
2      mov     rdi,%1
3      sub     rsp,8
4      call    puts
5      add     rsp,8
6  %endmacro
7  extern puts

test-include.asm X
1  %include "misMacrosL.asm"
2  global main
3  section .data
4      mensaje db "Hola mundo!!",0
5
6  section .text
7  main:
8      mPuts mensaje
9      ret

PS C:\Users\Dario\Documents\Orga\include> nasm test-include.asm -fwin64
PS C:\Users\Dario\Documents\Orga\include> gcc test-include.obj
PS C:\Users\Dario\Documents\Orga\include> ./a.exe
Hola mundo!!
PS C:\Users\Dario\Documents\Orga\include> 
```

Trabajando en grupo, si todos usamos las mismas macros usamos esto.

Instrucciones

Instrucciones - Transferencia y Copia

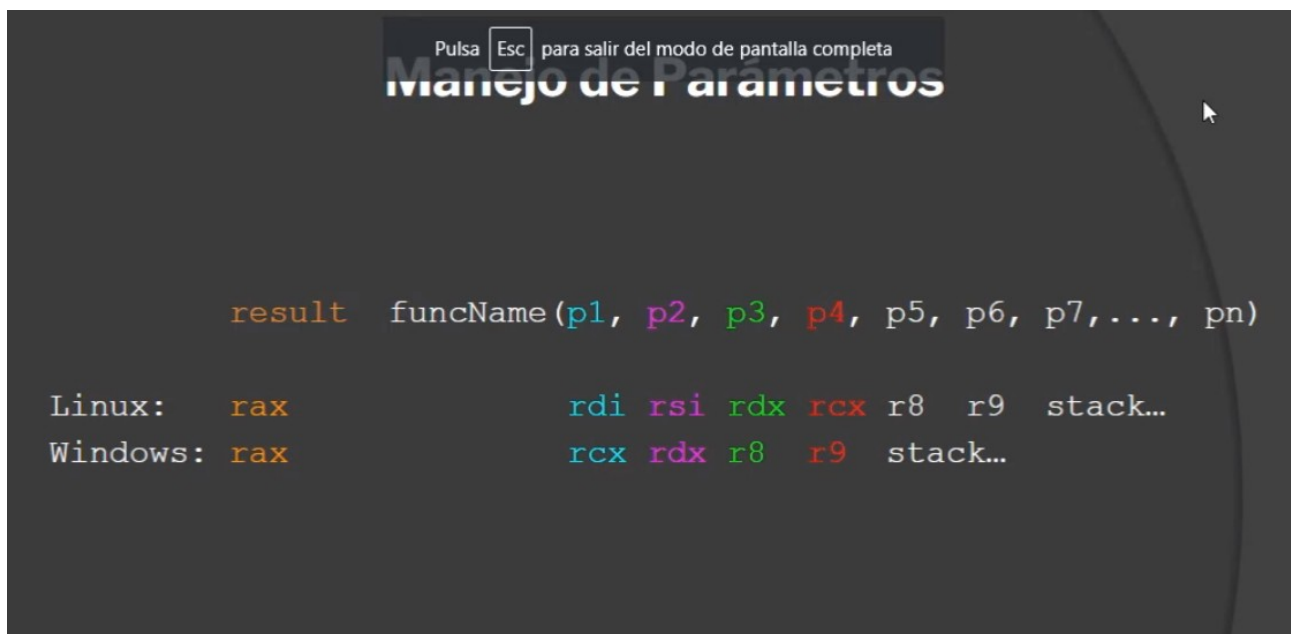
MOV op1,op2

Copia el valor del 2do operando en el primer operando.

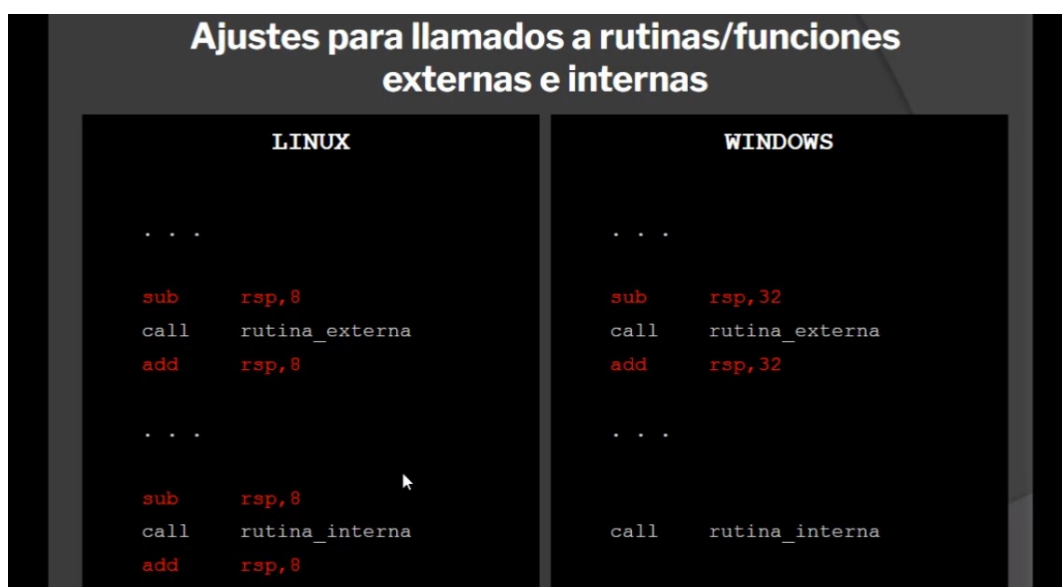
Combinaciones	Ejemplos en NASM
MOV <reg>,<reg>	MOV AH,BL MOV AX,BX MOV ECX, EAX MOV RDX,RCX
MOV <reg>,<mem>	MOV CH,[VARIABLE_8] (*) MOV CX, [VARIABLE_16] (*) MOV ECX, [VARIABLE_32] (*) MOV RDX, [VARIABLE_64] (*)
MOV <reg>,<inm>	MOV DL,7o MOV CX,2450h MOV EAX,0h MOV RDX,28h

MOV op1,op2

Combinaciones	Ejemplos en NASM
MOV <mem>,<reg>	MOV [VARIABLE_8],AH (*) MOV [VARIABLE_16],AX (*) MOV [VARIABLE_32],EAX (*) MOV [VARIABLE_64],RAX (*) MOV VARIABLE_64,RAX
MOV <long><mem>,<inm>	MOV byte [VARIABLE_8],2Ah MOV word [VARIABLE_16],777o MOV dword [VARIABLE_32],1234 MOV qword [VARIABLE_64],1234 MOV [VARIABLE_64],4321



los parametros se cargan antes en un call.



Salida por Pantalla

Función puts

Imprime un string hasta que encuentra un 0 (cero binario). Agrega el caracter de fin de línea a la salida

```
int puts(const char *str)
```

```
;LINUX
global main
extern puts

section .data
|   cadena db "Hola",0
section .text
main:
|   mov     rdi,cadena
|   sub     rsp,8
|   call    puts
|   add     rsp,8
```

```
;WINDOWS
global main
extern puts

section .data
|   cadena db "Hola",0
section .text
main:
|   mov     rcx,cadena
|   sub     rsp,32
|   call    puts
|   add     rsp,32
```

Ver como funciona cada registro

```
holamundo.asm X
1  global main
2  extern puts
3
4  section .data
5      mensaje db "Hola mundo",0
6
7  section .text
8  main:
9      mov rdi,mensaje
10     sub rsp,8
11     call puts
12     add rsp,8
13
14     ret
```

Instrucciones para ensamblar:


```
dario@dario-HP-EliteBook-8440p: ~/Orga/Intel/Linux
File Edit View Search Terminal Help
dario@dario-HP-EliteBook-8440p:~/Orga/Intel/Linux$ nasm holam.asm -f elf64
dario@dario-HP-EliteBook-8440p:~/Orga/Intel/Linux$ gcc holam.o -o hola.out -no-pie
dario@dario-HP-EliteBook-8440p:~/Orga/Intel/Linux$ ./hola.out
Hola mundo!
dario@dario-HP-EliteBook-8440p:~/Orga/Intel/Linux$
```

como lo puso el profe:

```
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ nasm holamundo.asm -f elf64
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ ls
holamundo.asm holamundo.o
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ gcc holamundo.o -no-pie
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ ls
a.out holamundo.asm holamundo.o
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ gcc holamundo.o -no-pie -o holamundo
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ ls
a.out holamundo holamundo.asm holamundo.o
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ ./holamundo
Hola mundo
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$
```

como uso debugger:

```
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ nasm holamundo.asm -f elf64 -g -F dwarf -l
holamundo.lst
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ ls
a.out holamundo holamundo.asm holamundo.lst holamundo.o
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ gcc holamundo.o -no-pie -o holamundo
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ ./holamundo
Hola mundo
dario@DESKTOP-1KL5D7K:/mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1$ gdb holamundo
GNU gdb (Ubuntu 12.1-0ubuntu1~22.04) 12.1
Copyright (C) 2022 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:
<https://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...
(gdb)
```

interface:

```
rax 0x401130 4196704 rbx 0x0 0 rcx 0x401010 4210200
rdx 0x7fffffff0000 140737488347288 rdi 0x26ffffff0000 140737488347272 rsi 0x400020 4210736
rip 0x7ffffffc3900 140737353012384 r10 0x7ffffffc3900 140737353090056 r12 0x7fffffffa8f0 140737353781008
r12 0x7ffffffc3900 140737488347272 r13 0x401130 4196704 r14 0x7ffffffde600 140737353999968
r15 0x7ffffffc3900 140737354125376 r16 0x401130 4196704 r15 0x7ffffffde600 140737353999968
cs 0x33 51 fs 0x20 0 gs 0x0 0
es 0x0 0 fs 0x0 0 gs 0x0 0

-holamundo.asm
0
1 section text
2
3 main:
4 mov rdi,mensaje
5
6
7
8
9
10
11 call puts
12 add rsp,8
13
14 ret
15
16
17
18

[1] Thread Thread 0x7ffff7db7 in: main
(gdb) b
Breakpoint 1 at 0x401130: file holamundo.asm, line 9.
(gdb) r
Starting program: /mnt/c/Users/Dario/Documents/Orga/Intel/2024C1/clase1/holamundo
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib64/libthread_db.so.1".

Breakpoint 1, main () at holamundo.asm:9
(gdb) n
main () at holamundo.asm:11
(gdb)
```

