Introducción a la programación para ingeniería de computadores

Práctica 1: Entorno de desarrollo GNU

Gustavo Romero López

Updated: 27 de septiembre de 2024

Arquitectura y Tecnología de Computadores

Índice

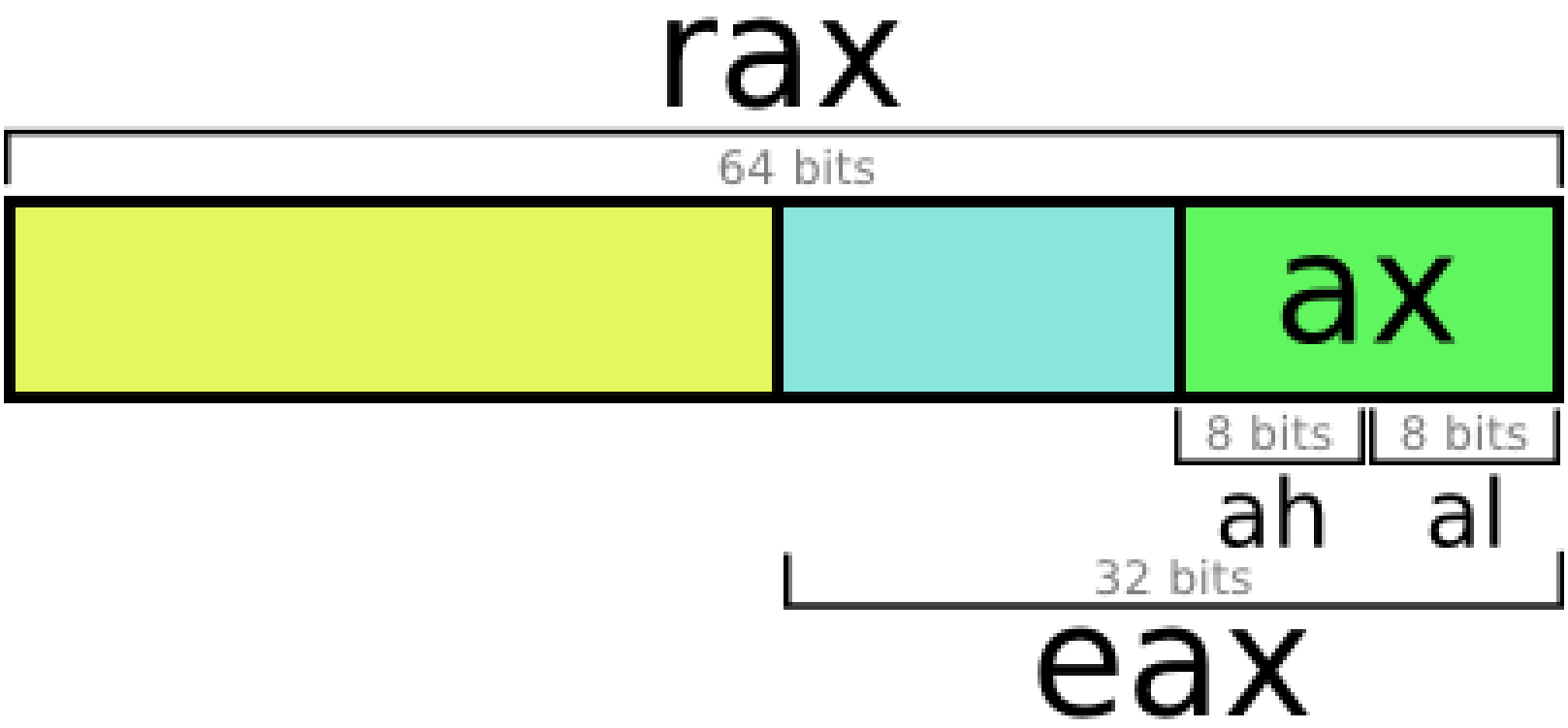
Objetivos

- Nociones de ensamblador 80x86 de 64 bits.
- Linux es tu amigo: si no sabes algo pregunta... **man**.
- Hoy aprenderemos varias cosas:
  - El esqueleto de un programa básico en ensamblador.
  - Como aprender de un maestro: el compilador **gcc**.
  - Herramientas clásicas del entorno de programación UNIX:
    - **make**: hará el trabajo sucio y rutinario por nosotros.
    - **as**: el ensamblador.
    - **ld**: el enlazador.
    - **gcc**: el compilador.
    - **nm**: lista los símbolos de un fichero.
    - **objdump**: el desensamblador.
  - Herramienta web: Compiler Explorer.
  - Ingeniería inversa: Cutter / Decompiler Explorer.

Ensamblador 80x86

- Los **8ox86** son una familia de procesadores.
- El más utilizado junto a los procesadores **ARM**.
- En estas prácticas vamos a centrarnos en su **lenguaje ensamblador** (inglés).
- El lenguaje ensamblador es el más básico, tras el binario, con el que podemos escribir programas utilizando las **instrucciones** que entiende el procesador.
- Cualquier estructura de un lenguajes de alto nivel pueden crearse mediante instrucciones muy sencillas.
- Normalmente es utilizado para poder acceder a partes que los lenguajes de alto nivel nos ocultan, complican o hacen de forma inconveniente.

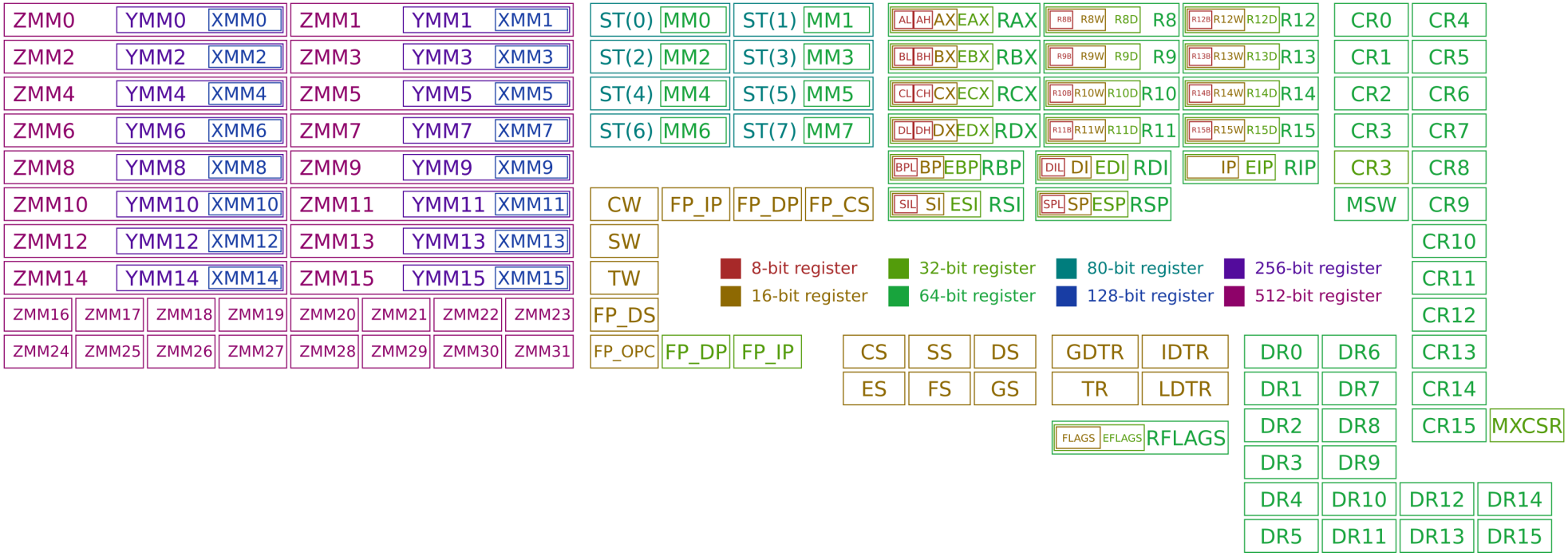
Arquitectura 80x86: el registro A



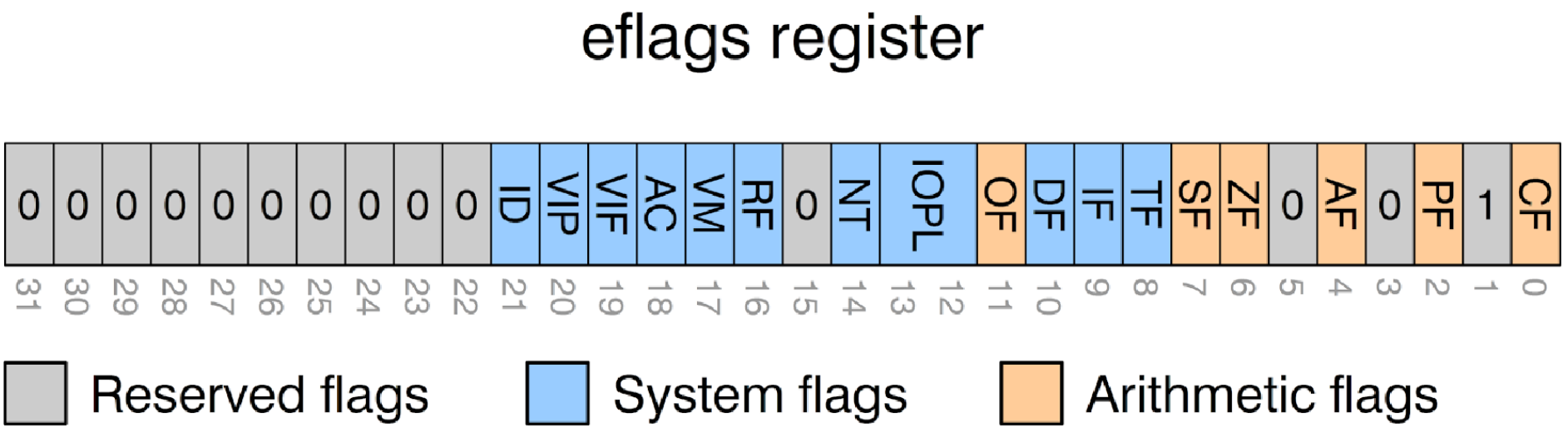
Arquitectura 80x86: registros de propósito general

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Arquitectura 80x86: registros completos



Arquitectura 80x86: banderas



TF: Trap
IF: Interrupt
DF: Direction

CF: Carry
PF: Parity
AF: Adjust
ZF: Zero
SF: Sign
OF: Overflow

Paso de parámetros a funciones en 64 bits

%rax	Return value	%r8	Argument #5
%rbx	Callee saved	%r9	Argument #6
%rcx	Argument #4	%r10	Caller saved
%rdx	Argument #3	%r11	Caller Saved
%rsi	Argument #2	%r12	Callee saved
%rdi	Argument #1	%r13	Callee saved
%rsp	Stack pointer	%r14	Callee saved
%rbp	Callee saved	%r15	Callee saved

Arquitectura 80x86 en Linux: llamadas al sistema

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	sys_read	unsigned int fd	char *buf	size_t count			
1	sys_write	unsigned int fd	const char *buf	size_t count			
2	sys_open	const char *filename	int flags	int mode			
3	sys_close	unsigned int fd					
4	sys_stat	const char *filename	struct stat *statbuf				
5	sys_fstat	unsigned int fd	struct stat *statbuf				
6	sys_lstat	fconst char *filename	struct stat *statbuf				
7	sys_poll	struct poll_fd *ufds	unsigned int nfds	long timeout_msecs			
8	sys_lseek	unsigned int fd	off_t offset	unsigned int origin			
9	sys_mmap	unsigned long addr	unsigned long len	unsigned long prot	unsigned long flags	unsigned long fd	unsigned long off

Programa mínimo en C...

minimo1.c

```c
int main() {}
```

minimo2.c

```c
int main() { return 0; }
```

minimo3.c

```c
#include <stdlib.h>
int main() { exit(0); }
```

Trasteando el programa mínimo en C

- Compilar: `gcc minimo1.c -o minimo1`
- ¿Qué he hecho? `file ./minimo1`
- ¿Qué contiene? `nm ./minimo1`
- Ejecutar: `./minimo1`
- ¿Qué devuelve? `./minimo1; echo $?`
- Desensamblar: `objdump -d minimo1`
- Ver llamadas al sistema: `strace ./minimo1`
- Ver llamadas de biblioteca: `ltrace ./minimo1`
- ¿Qué bibliotecas usa? `ldd minimo1`

```
linux-vdso.so.1 (0x00007ffe2ddbc000)
libc.so.6 => /lib64/libc.so.6 (0x00007fbc5043a000)
/lib64/ld-linux-x86-64.so.2 (0x0000558dbe5aa000)
```

- Examinar biblioteca: `objdump -d /lib64/libc.so.6`

Ensamblador desde 0: secciones básicas de un programa

```
.data

.text
```

Ensamblador desde 0: punto de entrada

Si no necesita la biblioteca de C puede evitarla así:

- La etiqueta `_start` identifica el programa principal.
- Ensamble/Compile con la opción `-nostartfiles`.

```
.text
_start: .globl _start
```

Si requiere de la biblioteca de C...

- La etiqueta `main` identifica el programa principal.
- Alinee correctamente la pila al inicio de `main`.

```
.text
main: .globl main
```

Ensamblador desde 0: datos

```
.data
msg:    .string "¡hola, mundo!\n"
tam:    .quad . - msg
```

Ensamblador desde 0: código

```
write:  mov   $1,   %rax  # write
        mov   $1,   %rdi  # stdout
        mov   $msg, %rsi  # texto
        mov   tam,  %rdx  # tamaño
        syscall           # llamada a write
        ret

exit:   mov   $60,  %rax  # exit
        xor   %rdi, %rdi  # 0
        syscall           # llamada a exit
        ret

_start: call  write       # llamada a función
        call  exit        # llamada a función
```

Ensamblador desde 0: ejemplo básico hola.s

```
1  .data
2  msg:    .string "¡hola, mundo!\n"
3  tam:    .quad . - msg
4
5  .text
6          .global _start
7
8  write:  mov   $1,   %rax  # write
9          mov   $1,   %rdi  # stdout
10         mov   $msg, %rsi  # texto
11         mov   tam,  %rdx  # tamaño
12         syscall           # llamada a write
13         ret
14
15 exit:   mov   $60,  %rax  # exit
16         xor   %rdi, %rdi  # 0
17         syscall           # llamada a exit
18         ret
19
20 _start: call  write       # llamada a función
21         call  exit        # llamada a función
22
```

¿Cómo hacer ejecutable mi programa?

¿Cómo hacer ejecutable el código anterior?

- opción a: ensamblar + enlazar
  - `as hola.s -o hola.o`
  - `ld hola.o -o hola`
- opción b: compilar = ensamblar + enlazar
  - `gcc -nostdlib -no-pie hola.s -o hola`
- opción c: que lo haga alguien por mi ⟶ `make`
  - `makefile`: fichero con definiciones, objetivos y recetas.

Ejercicios:

1. Cree un ejecutable a partir de hola.s.
2. Use `file` para ver el tipo de cada fichero.
3. Descargue el fichero `makefile`, pruébelo e intente hacer alguna modificación.
4. Examine el código ensamblador con `objdump -d hola`.

```
all: att

att: $(ATT)

clean:
    -rm -fv $(ATT) $(EXE) *~

exe: $(EXE)

.PHONY: all att clean exe
```

Ejemplo en C: hola-c.c

```c
#include <stdio.h>

int main()
{
    printf("¡hola, mundo!\n");
    return 0;
}
```

- ¿Qué hace gcc con mi programa?
- La única forma de saberlo es desensamblarlo:
  - Sintaxis AT&T: `objdump -d hola-c`
  - Sintaxis Intel: `objdump -d hola-c -M intel`

Ejercicios:

5. ¿Cómo se imprime el mensaje “hola mundo”?

Ejemplo en C++: hola-c++.cc

```cpp
#include <iostream>

int main()
{
    std::cout << "¡hola, mundo!" << std::endl;
}
```

- ¿Qué hace g++ con mi programa?
- La única forma de saberlo es desensamblarlo:
  - Sintaxis AT&T: `objdump -C -d hola-c++`
  - Sintaxis Intel: `objdump -C -d hola-c++ -M intel`

Ejercicios:

6. ¿Qué hace ahora diferente la función `main()` respecto a C?

Depuración: hola32.s

ejemplo de 32 bits

```
write:  movl  $4, %eax    # write
        movl  $1, %ebx    # salida estándar
        movl  $msg, %ecx  # cadena
        movl  tam, %edx   # longitud
        int   $0x80       # llamada a write
        ret               # retorno

exit:   movl  $1, %eax    # exit
        xorl  %ebx, %ebx  # 0
        int   $0x80       # llamada a exit
```

Puede ser necesario instalar algún paquete especial...

- fedora: `sudo dnf -y install glibc-devel.i686`
- ubuntu: `sudo apt-get install -y gcc-multilib`

Ejercicios:

7. Si siente curiosidad mire hola32p.s. Cabe destacar código de 32 bits, uso de "*little endian*", llamada a subrutina, uso de la pila y codificación de caracteres.

Depuración: hola64.s

ejemplo de 64 bits

```
write:  mov   $1,   %rax  # write
        mov   $1,   %rdi  # stdout
        mov   $msg, %rsi  # texto
        mov   tam,  %rdx  # tamaño
        syscall           # llamada a write
        ret

exit:   mov   $60,  %rax  # exit
        xor   %rdi, %rdi  # 0
        syscall           # llamada a exit
        ret
```

Ejercicios:

8. Compare hola64.s con hola64p.s. Sobre este podemos destacar: código de 64 bits, llamada a subrutina, uso de la pila y codificación de caracteres.

- ¿Sabes C? $\Longleftrightarrow$ ¿Has usado la función `printf()`?

```c
int main()
{
    int i = 0x12345678;
    printf("i = %i =
    ↪  0x%08x\n", i, i);
    return 0;
}
```

```c
int i = 0x12345678;
char *formato = "i = %i =
↪  0x%08x\n";

int main()
{
    printf(formato, i, i);
    return 0;
}
```

Ejercicios:

9. ¿En qué se parecen y en qué se diferencian printf-c-1.c y printf-c-2.c? `nm`, `objdump` y `kdiff3` serán muy útiles...

Mezclando lenguajes: ensamblador y C (32 bits) printf32.s

```
        .extern printf      # printf en otro sitio
        .globl _start       # función principal

_start: push i              # apila i
        push $f             # apila f
        mov  $0, %eax       # n de registros vectoriales
        call printf         # llamada a printf
        add  $8, %esp       # restaura pila

        movl  $1, %eax      # exit
        xorl  %ebx, %ebx    # 0
        int   $0x80         # llamada a exit
```

Ejercicios:

10. Descargue y compile `printf32.s`.
11. Modifique `printf32.s` para que finalice mediante la función `exit()` de C (`man 3 exit`). Solución: `printf32e.s`.

Mezclando lenguajes: ensamblador y C (64 bits) printf64.s

```
.text
        .globl _start

_start: mov $f, %rdi        # formato
        mov i, %rsi         # i
        xor %rax, %rax      # n de registros vectoriales
        call  printf        # llamada a función

        xor %rdi, %rdi      # valor de retorno
        call exit           # llamada a función
```

Ejercicios:

12. Descargue y compile `printf64.s`.
13. Busque las diferencias entre `printf32.s` y `printf64.s`.

Optimización: sum.cc

```
int main()
{
  int sum = 0;

  for (int i = 0; i < 10; ++i)
    sum += i;

  return sum;
}
```

Ejercicios:

14. ¿Cómo implementa gcc los bucles `for`?
15. Observe el código de la función `main()` al compilarlo...
    - sin optimización: `g++ -O0 sum.cc -o sum`
    - con optimización: `g++ -O3 sum.cc -o sum`

Optimización: función `main()` de sum.cc

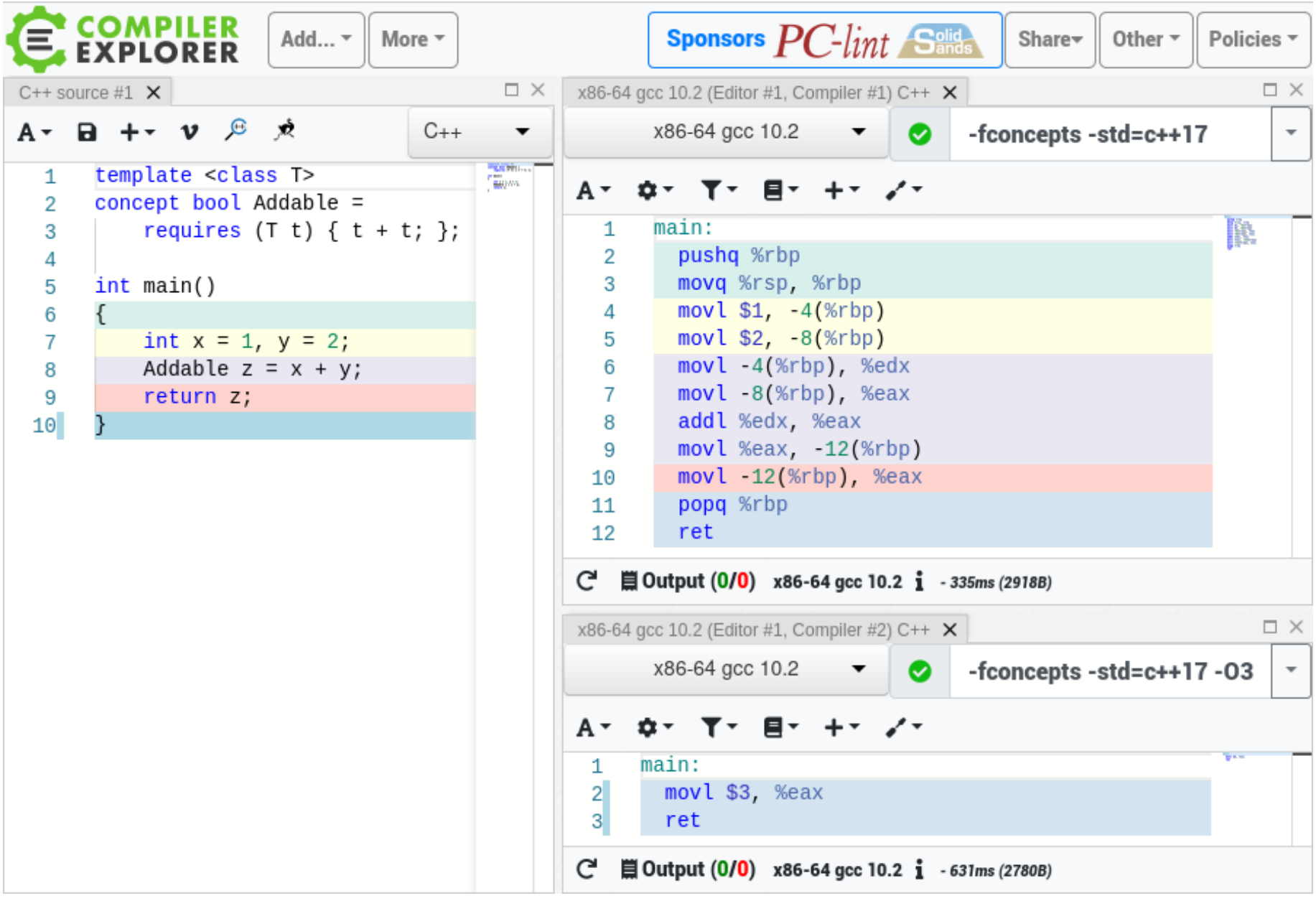
sin optimización (`gcc -O0`)

```
4005b6:    55                          push   %rbp
4005b7:    48 89 e5                    mov    %rsp,%rbp
4005ba:    c7 45 fc 00 00 00 00        movl   $0x0,-0x4(%rbp)
4005c1:    c7 45 f8 00 00 00 00        movl   $0x0,-0x8(%rbp)
4005c8:    eb 0a                       jmp    4005d4 <main+0x1e>
4005ca:    8b 45 f8                    mov    -0x8(%rbp),%eax
4005cd:    01 45 fc                    add    %eax,-0x4(%rbp)
4005d0:    83 45 f8 01                 addl   $0x1,-0x8(%rbp)
4005d4:    83 7d f8 09                 cmpl   $0x9,-0x8(%rbp)
4005d8:    7e f0                       jle    4005ca <main+0x14>
4005da:    8b 45 fc                    mov    -0x4(%rbp),%eax
4005dd:    5d                          pop    %rbp
4005de:    c3                          retq
```

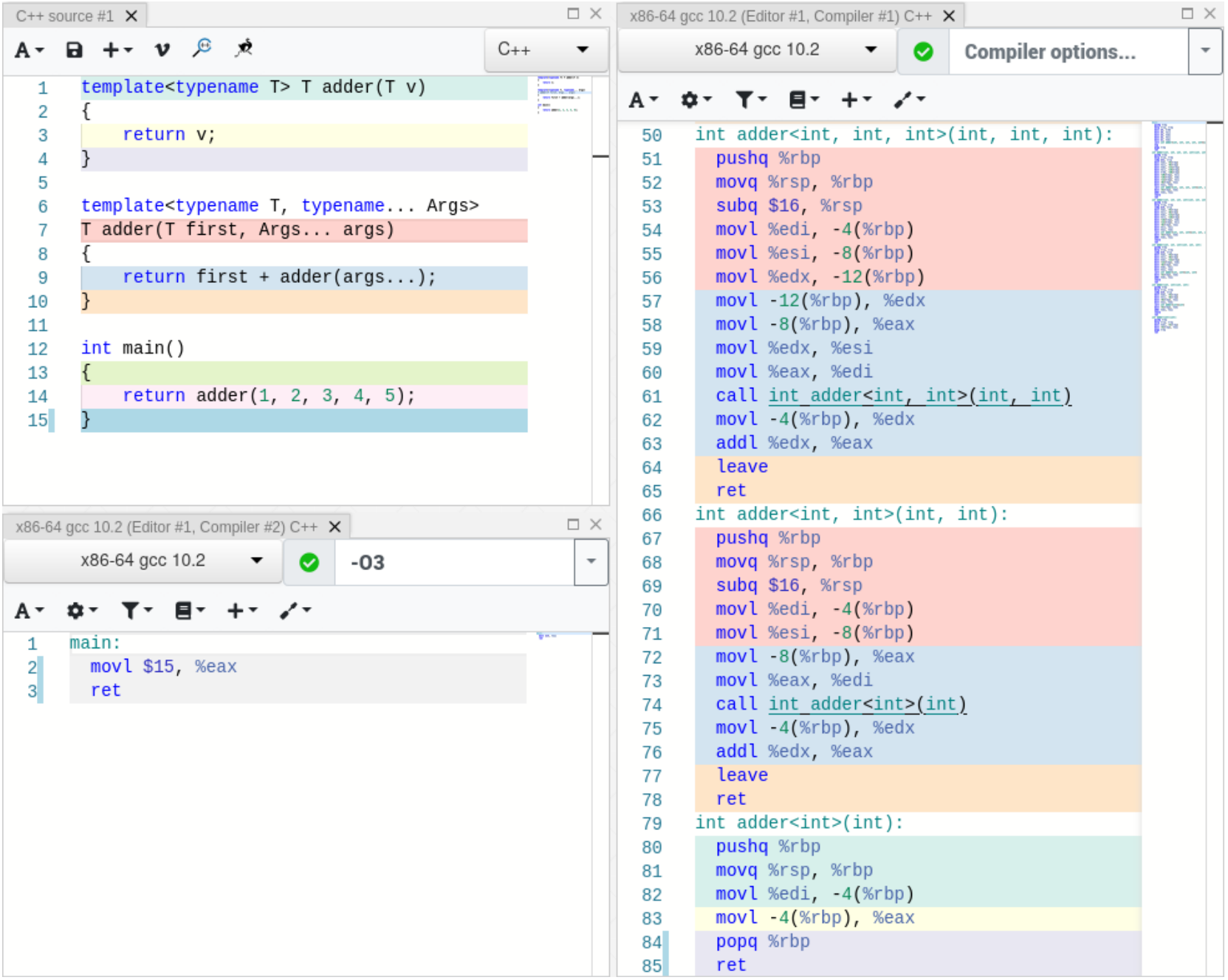
con optimización (`gcc -O3`)

```
4004c0:    b8 2d 00 00 00              mov    $0x2d,%eax
4004c5:    c3                          retq
```

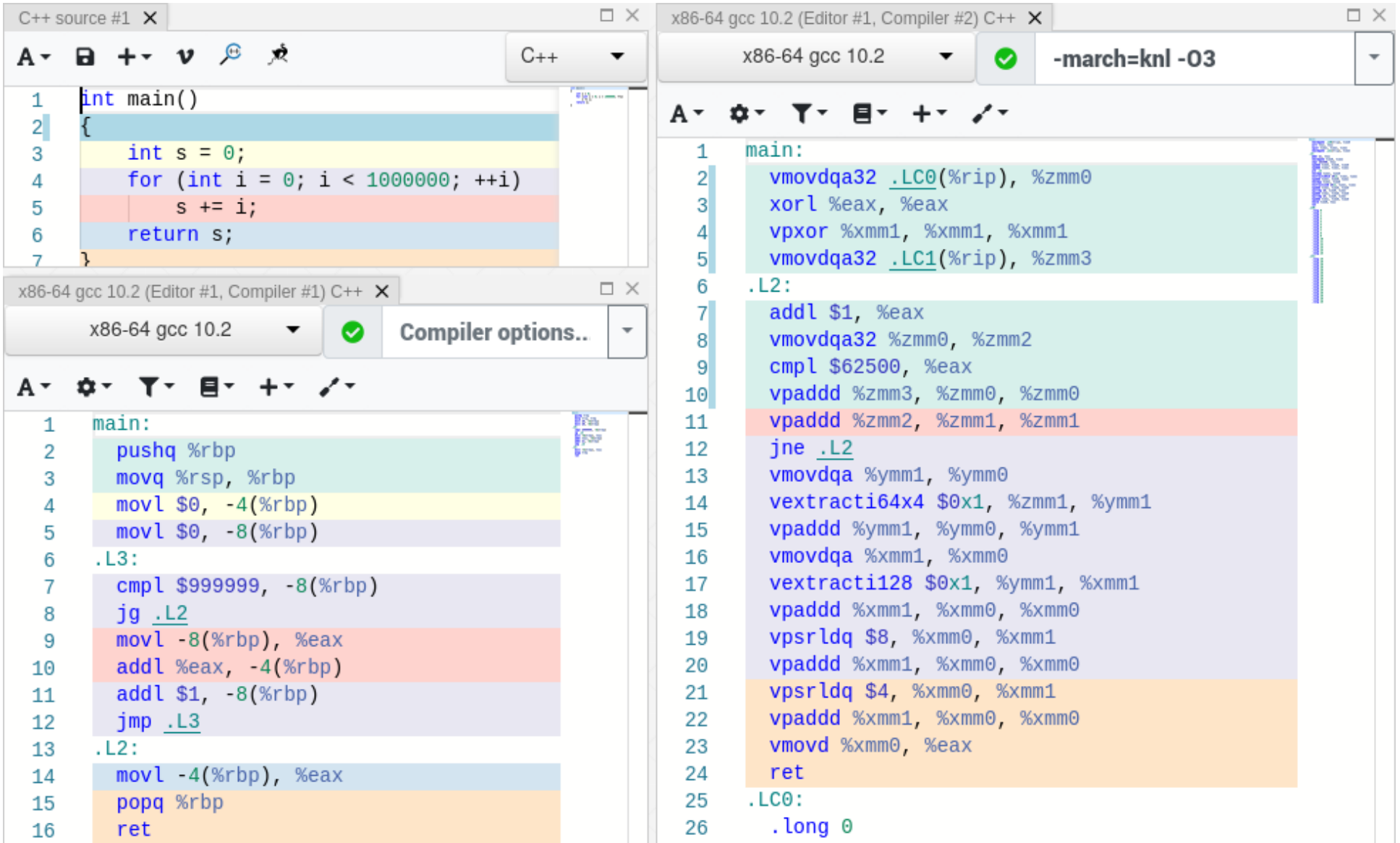
Compiler Explorer: https://godbolt.org/z/9bT7sb



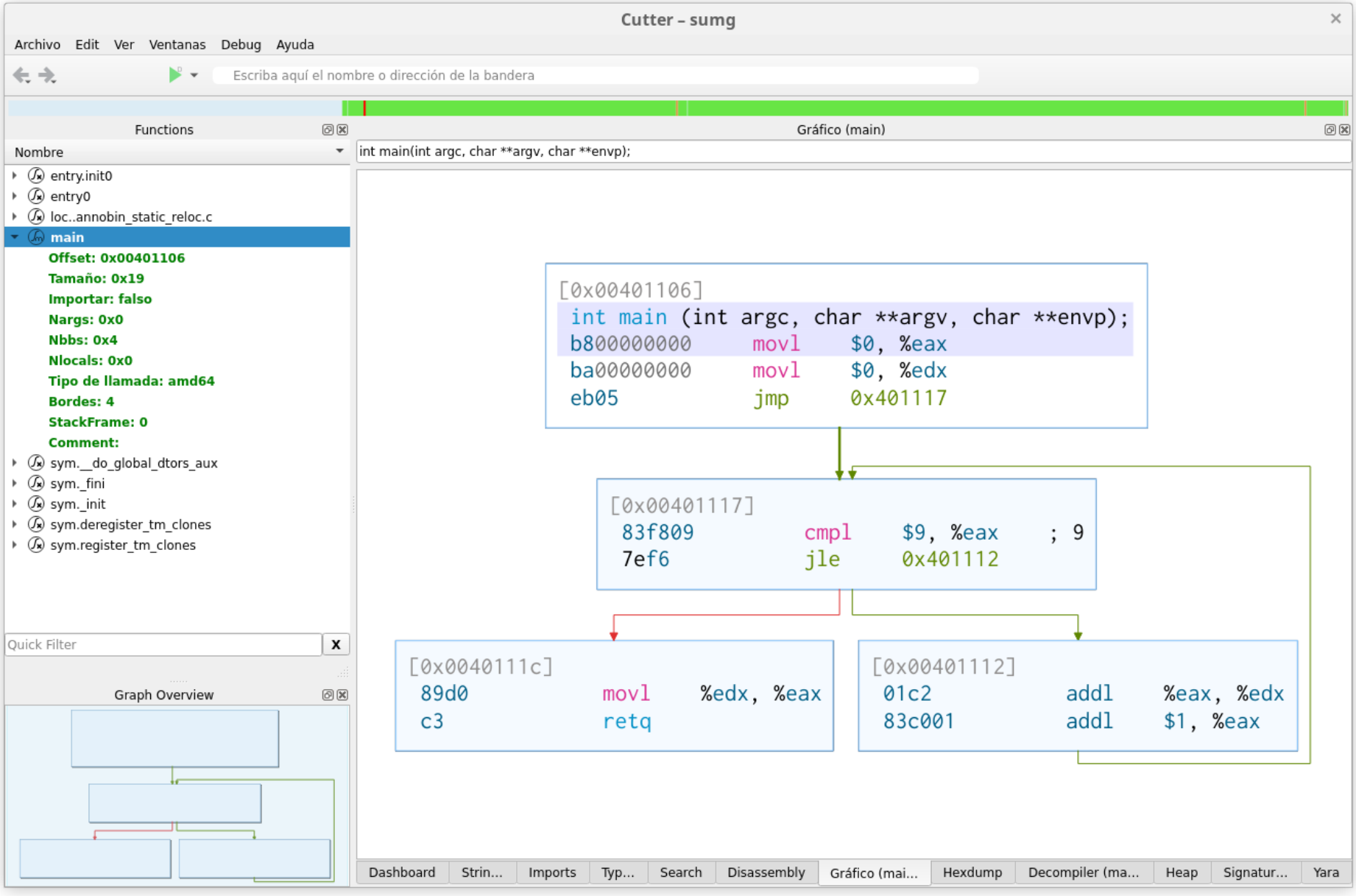
Compiler Explorer: https://godbolt.org/z/ahhqs9



Compiler Explorer: https://godbolt.org/z/1hWeWM



Cutter: https://github.com/rizinorg/cutter



Decompiler Explorer: https://dogbolt.org

Decompiler Explorer

What is this?

Upload File

Your file must be **less than 2MB** in size.
Uploaded binaries are retained.

Examinar... hola

Samples

Or check out one of these samples we've provided:

Select a Sample...

- [ ] angr
- [ ] Ghidra
- [ ] Relyze
- [x] BinaryNinja
- [ ] Hex-Rays
- [ ] RetDec
- [ ] Boomerang
- [ ] RecStudio
- [ ] rev.ng
- [ ] dewolf
- [ ] Reko
- [ ] Snowman

BinaryNinja ↻

4.1.5747 (7558ffe9)

```
1  void _start() __noreturn
2  {
3      write();
4      exit();
5      /* no return */
6  }
7
8  int64_t write()
9  {
10     return syscall(sys_write {1}, 1, &msg, tam);
11 }
12
13 void exit() __noreturn
14 {
15     syscall(sys_exit {0x3c}, 0);
16     /* no return */
17 }
```

Enlaces de interés

Manuales:

- Hardware:
  - AMD
  - Intel
- Software:
  - AS
  - NASM

Programación:

- Programming from the ground up
- Linux Assembly

Chuletas:

- Chuleta del 8086
- Chuleta del GDB