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

- 1. Índice
- 2. Objetivos
- 3. Introducción
- 4. C
- 5. Ensamblador
- 6. Ejemplos
- 6.1 hola
- 6.2 make
- 6.3 Ejemplo en C
- 6.4 Ejemplo en C++

- 6.5 Ejemplo en 32 bits
- 6.6 Ejemplo en 64 bits
- 6.7 ASM + C
- 6.8 Optimización
- 7. Herramientas en línea
- 7.1 Compiler Explorer
- 7.2 Cutter
- 7.3 Decompiler Explorer
- 8. Enlaces

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 80x86 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	сх	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	rıod	r10W	rıob
r11	r11d	r11W	r11b
r12	r12d	r12W	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Arquitectura 80x86: registros completos

ZMM0	YMM0 XMM0	ZMM1	YMM1	XMM1	ST(0)	MM0	ST(1) MM	1	ALAHAXEA	X RAX	R8W R8D	R8 8121 R121	R12DR12	CR0	CR4		
ZMM2	YMM2 XMM2	ZMM3	YMM3	ХММЗ	ST(2)	MM2	ST(3) MM	13	вценВХЕЕ	X RBX	ne R9W R9D	R9 813	MR13DR13	CR1	CR5		
ZMM4	YMM4 XMM4	ZMM5	YMM5	XMM5	ST(4)	MM4	ST(5) MM	15	CL[CH]CXEC	RCX	8100 R10W R10D	R10 100 R141	R140R14	CR2	CR6		
ZMM6	YMM6 XMM6	ZMM7	YMM7	XMM7	ST(6)	MM6	ST(7) MM	17	DLDHDXED	XRDX	****R11WR11D	R11 100 R150	MR15DR15	CR3	CR7		
ZMM8	YMM8 XMM8	ZMM9	YMM9	XMM9					BPLBPEBF	RBP	DIL DI EDI F	RDI 🗾 IP	EIP RIP	CR3	CR8		
ZMM10	YMM10 XMM10	ZMM11	YMM11	XMM11	CW	FP_IP	FP_DP FP_	CS	SIL SI ES	RSI	SPU SPESP	SP		MSW	CR9		
ZMM12	YMM12 XMM12	ZMM13	YMM13	XMM13	SW										CR10		
ZMM14	YMM14 XMM14	ZMM15	YMM15	XMM15	TW		8-bit regist	er (32-bit	register	80-bit	register	256-bit	register	CR11		
ZMM16 ZMI	M17 ZMM18 ZMM19	ZMM20 ZM	M21 ZMM22	2 ZMM23	FP_DS		10-bit legit	ster	04-01	register	120-01	t register	512-010	register	CR12		
ZMM24 ZMI	M25 ZMM26 ZMM27	ZMM28 ZM	M29 ZMM30	ZMM31	FP_OPC	FP_DP	FP_IP	CS	SS	DS	GDTR	IDTR	DR0	DR6	CR13		
								ES	FS	GS	TR	LDTR	DR1	DR7	CR14		
											TLAGS ITLAG	RELAGS	DR2	DR8	CR15	MXCSF	ł
													DR3	DR9			
													DR4	DR10	DR12	DR14	
													DR5	DB11	DR13	DB15	

eflags register



Paso de parámetros a funciones en 64 bits

8rax	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

minimo1.c

int main() {}

minimo2.c

int main() { return 0; }

minimo3.c

#include <stdlib.h>

```
int main() { exit(0); }
```

Trasteando el programa mínimo en C

- Ocompilar: gcc minimol.c -o minimol Qué he hecho? file ./minimo1 Oué contiene? nm /minimo1 • Ejecutar: ./minimo1 Oué devuelve? ./minimo1; echo \$? O Desensamblar: objdump -d minimo1 O Ver llamadas al sistema: strace ./minimo1 O Ver llamadas de biblioteca: ltrace ./minimo1 Oué 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

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

.data

- msg: .string "ihola, 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		
	sysca	ll		#	llamada	а	write
	ret						
exit:	mov	\$60,	%rax	#	exit		
	xor	%rdi,	%rdi	#	0		
	sysca	ll		#	llamada	а	exit
	ret						
_start:	call	write		#	llamada	а	función
	call	exit		#	llamada	а	función

Ensamblador desde 0: ejemplo básico hola.s

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

22

17

¿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
 - o gcc -nostdlib -no-pie hola.s -o hola
- \odot opción c: que lo haga alguien por mi \longrightarrow 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

#include <stdio.h>

```
int main()
{
    printf("ihola, 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"?

#include <iostream>

```
int main()
{
    std::cout << "ihola, mundo!" << std::endl;
}</pre>
```

- ◎ ¿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	<pre># longitud</pre>
	int	\$0x80	<pre># llamada a write</pre>
	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	
	sysca	II		#	llamada	a write
	ret					
exit:	mov	\$60,	%rax	#	exit	
	xor	%rdi,	%rdi	#	0	
	sysca	II		#	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. ¿Dónde están mis datos?

⊙ ¿Sabes C? \iff ¿Has usado la función printf()?

```
int main()
                               int i = 0 \times 12345678;
                               char *formato = "i = %i =
{
                               int i = 0 \times 12345678;
    printf("i = %i =
    \rightarrow 0x%08x\n", i, i);
                               int main()
    return 0:
                               {
                                    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 .globl _start	# printf en otro sitio # función principal					
at ant .	avab i	# opilo i					
_start:	push 1	# apita i					
	push \$f	# apila f					
	mov \$0, %eax	<pre># n de registros vectoriales</pre>					
	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.
- 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...
o sin optimización: g++ -00 sum.cc -o sum
o con optimización: g++ -03 sum.cc -o sum

Optimización: función main() de sum.cc

pu mo mo mo jm mo ad ad CM jl mo po re

sin optimización (gcc -00)

4005b6:	55						
4005b7:	48	89	e5				
4005ba:	c7	45	fc	00	00	00	00
4005c1:	c7	45	f8	00	00	00	00
4005c8:	eb	0a					
4005ca:	8b	45	f8				
4005cd:	01	45	fc				
4005d0:	83	45	f8	01			
4005d4:	83	7d	f8	09			
4005d8:	7e	fØ					
4005da:	8b	45	fc				
4005dd:	5d						
4005de:	с3						

sh	%rbp
v	%rsp,%rbp
vl	\$0x0 ,-0x4(%rbp)
vl	\$0x0 ,-0x8(%rbp)
р	4005d4 <main+0x1e></main+0x1e>
v	-0 <mark>x8(%rbp),%</mark> eax
d	<pre>%eax,-0x4(%rbp)</pre>
dl	\$0x1 ,-0x8(%rbp)
pl	\$0x9 ,-0x8(%rbp)
е	4005ca <main+0x14></main+0x14>
v	-0x4(%rbp),%eax
р	%rbp
tq	

con optimización (gcc -03)

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

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

C++ source #1 × A - □ + - v , [®] , ★ C++						
A• B +• ν β κ C++	▼ x86-64 gcc 10.2 ▼					
		-17 -				
<pre>1 template <class t=""> 2 concept bool Addable = 3 requires (T t) { t + t; }; 4 5 int main() 6 { 7 int x = 1, y = 2; 8 Addable z = x + y; 9 return z; 10 }</class></pre>	A ★ ★★ ▼ ★ ■ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★ ★	NA.				
	C ¹					
	x86-64 gcc 10.2 (Editor #1, Compiler #2) C++ × x86-64 gcc 10.2 c c -fconcepts -std=c++17 -0 :					
	A - ☆- ▼- ■- +- /- 1 main: 2 movl \$3, %eax 3 ret					

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



30

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

C++ sour	ce #1 🗙	\Box \times	x86-64 gc	cc 10.2 (Editor #1, Compiler #2) C++ 🗙	\Box \times
A- 8	ι + - ν β κ C++	-		x86-64 gcc 10.2 • or emarch=knl -03	-
1 i 2 {	nt main()	(<u>82</u>)	A• \$	× T+ E+ ++ /+	in and
3 4 5	<pre>int s = 0; for (int i = 0; i < 1000000; ++i)</pre>		1 2 3	main: vmovdqa32 <u>.LC0(</u> %rip), %zmm0 xorl %eax, %eax vnxor %xmm1 %xmm1	
7 }	recurr s,		5	vmovdqa32 .LC1(%rip), %zmm3	
x86-64 gc	c 10.2 (Editor #1, Compiler #1) C++ 🗙	$\square \times$	6	.L2:	
	x86-64 gcc 10.2 🔻 📀 Compiler option	IS 🔻	7 8	addl \$1, %eax vmovdqa32 %zmm0, %zmm2	
A- \$	F T E + × ✓·		9 10	cmpl \$62500, %eax vpaddd %zmm3, %zmm0, %zmm0	
1	main:	En .	11	vpaddd %zmm2, %zmm1, %zmm1	
2	pushq %rbp	and a second	12	jne <u>.L2</u>	
3	movq %rsp, %rbp		13	vmovdqa %ymm1, %ymm0	
4	movl \$0, -4(%rbp)		14	vextract164x4 \$0x1, %zmm1, %ymm1	
5	movl \$0, -8(%rbp)		15	vpaddd %ymm1, %ymm0, %ymm1	
6	.L3:		16	vmovdqa %xmm1, %xmm0	
7	cmpl \$999999, -8(%rbp)		1/	Vextract1128 \$0x1, %ymm1, %xmm1	
8	Jg <u>.L2</u>		18	vpaddd %xmmi, %xmm0, %xmm0	
9	movl -8(%rbp), %eax		19	vpsrtaq \$8, %xmm0, %xmm1	
10	addl %eax, -4(%rbp)		20	vpaddd %xmm1, %xmm0, %xmm0	
11	addl \$1, -8(%rbp)		21	vpsrtaq \$4, %xmm0, %xmm1	
12	jmp <u>.L3</u>		22	vpaudu %xmm1, %xmm0, %Xmm0	
13	LZ:		23	villovu zoklimio, zecak	
14	movt -4(%rbp), %eax		24		
15	popq %rbp		25	LLUU.	
16	ret		26	. tong e	

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



Decompiler Explorer: https://dogbolt.org

			Deo	compiler Explo	orer			What is t	this?
Upload File Your file must be less than 2MB in size. Uploaded binaries <u>are retained</u> .						Samples Or check out one of these samples we've provided:			
		Examinar	hola			Select a Sample			~
anc Ghi Ghi Bina 4.1.57 1 2 4 5 6 7 8 9 10 11 12 13 14 11 12 13 14 15 16 16	gr idra yze ryNi '47 (7 { we e / / } int64 { r void { s / / }	<pre>inja C /558ffe9) _start()nor rite(); xit(); * no return */ _t write() eturn syscall(exit()noret yscall(sys_exi * no return */</pre>	sys_wri surn t {0x3c	<pre>Ø BinaryNinja Hex-Rays RetDec te {1}, 1, &msg, tam); }, 0);</pre>		Boomerang RecStudio rev.ng		☐ dewolf ☐ Reko ☐ Snowman	

Enlaces de interés

Manuales:

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

Programación:

- o Programming from the ground up
- O Linux Assembly

Chuletas:

- Ohuleta del 8086
- Ochuleta del GDB