x86 Overview

Introducción

- Hay dos versiones de x86:

1. IA-32 (Intel Architecture, 32 bits). Versión de 32 bits del ISA x86. También se le conoce como i386.
2. x86-64. Versión de 64 bits del ISA x86. También se le conoce como x64, AMD64 o Intel 64.

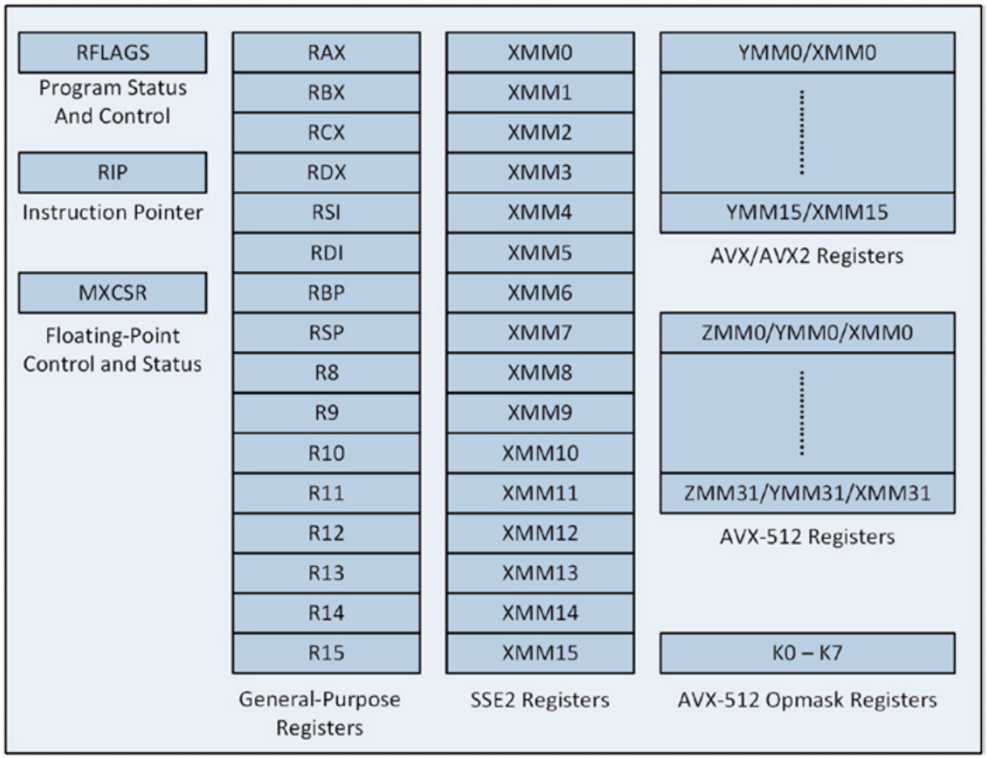
- Primer procesador con x86: Intel 8086 (1978), un procesador de 16 bits.

Tipos de datos

Data Type	Size (Bits)	Typical Use
Byte	8	Characters, small integers
Word	16	Characters, integers
Doubleword	32	Integers, single-precision floating-point
Quadword	64	Integers, double-precision floating-point
Double Quadword	128	Packed integers, packed floating-point

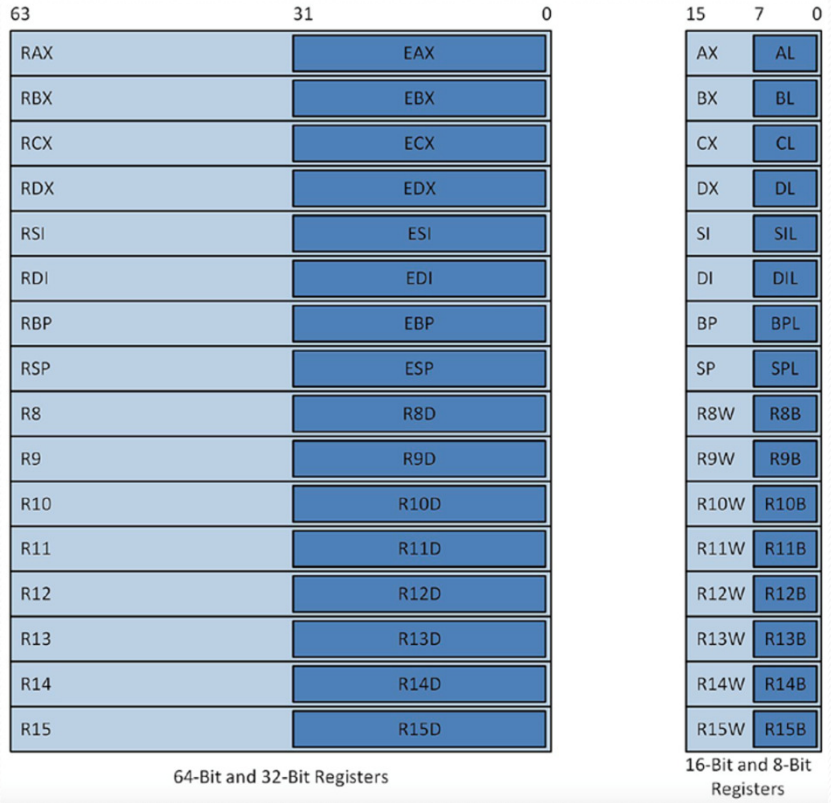
Fuente: Mx86ALP, p. 3

Registros



Fuente: Mx86ALP, p. 7

Registros de propósito general



Fuente: Mx86ALP, p. 8

Instrucciones

- Una instrucción puede tener 0, 1, 2 o 3 operandos.
- Los operandos se separan con comas.
- Para instrucciones con 2 operandos hay dos sintaxis:

  operando destino, fuente ; **sintaxis de Intel**

  operando fuente, destino ; **sintaxis de AT&T**
- Además, en algunos casos, el registro destino también es fuente.
- Por ejemplo:

  add eax, ebx ; **eax ← eax + ebx en sintaxis Intel**
- Hay otras diferencias.

Sintaxis Intel vs Sintaxis AT&T

	Intel	AT&T
Orden de operandos	destino, fuente	fuente, destino
Prefijo de registro	rbx	%rbx
Sufijo de tamaño	mov	movb, movw, movl, movq
Memoria	QWORD PTR [rbx]	(rbx)
Dirección efectiva	mov eax, [ebx + ecx*4 + offset]	movl offset(%ebx,%ecx,4), %eax
Prefijo de inmediato	80h	$0x80

Sintaxis Intel vs Sintaxis AT&T

- Sintaxis de Intel: FASM (Flat Assembler), MASM (Microsoft Macro Assembler), NASM (Netwide Macro Assembler).
- Sintaxis de AT&T: GAS (GNU Assembler).

Sufijos de tamaño de datos para AT&T

Sufijo	Tamaño	Descripción
b	8 bits	byte
w	16 bits	word
l	32 bits	long (double) word
q	64 bits	quad word

Tipos de instrucción

- Para instrucciones aritméticas, lógicas y de transferencia de datos.

Source/destination operand type	Second source operand
Register	Register
Register	Immediate
Register	Memory
Memory	Register
Memory	Immediate

Fuente: COD5, p. 153

Ejemplos

```
add rsp, 20                    ; rsp ← rsp + 20
xor rax, rax                   ; rax ← rax ^ rax
mov rcx, rax                   ; rcx ← rax
mov qword [rsp – 56], 0        ; Mem[rsp – 56] ← 0
mov rax, qword [rsi – 4]       ; rax ← Mem[rsi – 4]
add eax, dword [edi]           ; eax ← eax + Mem[edi]
```

Addressing Mode

- Una instrucción x86-64 requiere hasta cuatro componentes para especificar la ubicación de un operando en la memoria.
- Los cuatro componentes incluyen un offset, un registro base, un registro índice y un factor de escala.
- Usando estos componentes, el procesador calcula una dirección efectiva para un operando de memoria de la siguiente manera:
- EffectiveAddress = BaseReg + IndexReg * ScaleFactor + Disp

Addressing Mode

- Inmediato:

```
mov rax, 10        ; rax ← 10
```

- Directo:

```
mov eax, X         ; eax ← Mem[&X]
X dw 3
```

- Registro:

```
mov rax, rbx       ; rax ← rbx
```

- Indirecto con registro:

```
mov eax, [ebx]     ; eax ← Mem[ebx]
```

Addressing Mode

- Base y offset:

```
mov rax, [rbx + 8]          ; rax ← Mem[rbx + 8]
```

- Base con índice:

```
mov rax, [rbx + rcx]        ; rax ← Mem[rbx + rcx]
```

- Base con índice y offset:

```
mov rax, [rbx + rcx + 32]   ; rax ← Mem[rbx + rcx + 32]
```

- Índice escalado y offset:

```
mov rax, [rcx * 4 + 16]     ; rax ← Mem[rcx * 4 + 16]
```

Addressing Mode

- Base con índice escalado:

```
mov rax, [rbx + rcx * 4]          ; rax ← Mem[rbx + rcx * 4]
```

- Base con índice escalado y offset:

```
mov rax, [rbx + rcx * 4 + 48]     ; rax ← Mem[rbx + rcx * 4 + 48]
```

Ejemplos de instrucciones

Instruction	Function
je name	if equal(condition code) {EIP=name}; EIP-128 <= name < EIP+128
jmp name	EIP=name
call name	SP=SP-4; M[SP]=EIP+5; EIP=name;
movw EBX,[EDI+45]	EBX=M[EDI+45]
push ESI	SP=SP-4; M[SP]=ESI
pop EDI	EDI=M[SP]; SP=SP+4
add EAX,#6765	EAX= EAX+6765
test EDX,#42	Set condition code (flags) with EDX and 42
movsl	M[EDI]=M[ESI]; EDI=EDI+4; ESI=ESI+4

Fuente: COD5, p. 155

Ejemplos de instrucciones

Type	Example	Analogous C/C++ Statement
Immediate	`mov rax,42`	`rax = 42`
	`imul r12,-47`	`r12 *= -47`
	`shl r15,8`	`r15 <<= 8`
	`xor ecx,80000000h`	`ecx ^= 0x80000000`
	`sub r9b,14`	`r9b -= 14`
Register	`mov rax,rbx`	`rax = rbx`
	`add rbx,r10`	`rbx += r10`
	`mul rbx`	`rdx:rax = rax * rbx`
	`and r8w,0ff00h`	`r8w &= 0xff00`
Memory	`mov rax,[r13]`	`rax = *r13`
	`or rcx,[rbx+rsi*8]`	`rcx |= *(rbx+rsi*8)`
	`sub qword ptr [r8],17`	`*(long long*)r8 -= 17`
	`shl word ptr [r12],2`	`*(short*)r12 <<= 2`

Fuente: Mx86ALP, p. 11

x86-32 vs x86-64

8-Bit	16-Bit	32-Bit	64-Bit
`add al,bl`	`add ax,bx`	`add eax,ebx`	`add rax,rbx`
`cmp dl,[r15]`	`cmp dx,[r15]`	`cmp edx,[r15]`	`cmp rdx,[r15]`
`mul r10b`	`mul r10w`	`mul r10d`	`mul r10`
`or [r8+rdi],al`	`or [r8+rdi],ax`	`or [r8+rdi],eax`	`or [r8+rdi],rax`
`shl r9b,cl`	`shl r9w,cl`	`shl r9d,cl`	`shl r9,cl`

Fuente: Mx86ALP, p. 13

RFLAGS

- Contiene una serie de bits de estado (o banderas) que el procesador usa para indicar los resultados de una operación aritmética, lógica o de comparación.
- También contiene una serie de bits de control que utilizan los sistemas operativos.

RFLAGS

Bit Position	Name	Symbol	Use
0	Carry Flag	CF	Status
1	Reserved		1
2	Parity Flag	PF	Status
3	Reserved		0
4	Auxiliary Carry Flag	AF	Status
5	Reserved		0
6	Zero Flag	ZF	Status
7	Sign Flag	SF	Status
8	Trap Flag	TF	System
9	Interrupt Enable Flag	IF	System
10	Direction Flag	DF	Control
11	Overflow Flag	OF	Status
12	I/O Privilege Level Bit 0	IOPL	System
13	I/O Privilege Level Bit 1	IOPL	System
14	Nested Task	NT	System
15	Reserved		0

Fuente: Mx86ALP, p. 9

RFLAGS

Bit Position	Name	Symbol	Use
16	Resume Flag	RF	System
17	Virtual 8086 Mode	VM	System
18	Alignment Check	AC	System
19	Virtual Interrupt Flag	VIF	System
20	Virtual Interrupt Pending	VIP	System
21	ID Flag	ID	System
22 - 63	Reserved		0

Fuente: Mx86ALP, p. 10

Banderas de status

- CF (carry flag): 1 si la operación generó un carry o un borrow; 0 en otro caso.
- PF (parity flag): 1 si el resultado tiene paridad par (los 8 bits más bajos contienen un número par de unos); 0 si tiene paridad impar.
- ZF (zero flag): 1 si el resultado es cero; 0 en otro caso.
- SF (sign flag): 1 si el resultado es negativo; 0 si es positivo o cero.
- OF (overflow flag): 1 si la operación generó un overflow; 0 en otro caso.

Brincos condicionales

- x86 utiliza el estado de las banderas para los brincos condicionales.
- Las instrucciones aritméticas y lógicas cambian las banderas.
- Las instrucciones cmp y test también cambian las banderas.
- La instrucción test calcula el AND lógico bit a bit del primer operando y el segundo operando. El resultado cambia las banderas SF, ZF y PF.
- Ejemplo:

```
test eax, eax
jz Label
```

Brincos condicionales

- La instrucción cmp realiza resta sus operandos y cambia las banderas ZF y CF de acuerdo con el resultado.
- Ejemplo:

```
cmp eax, 1
je Loop
```

- Se puede hacer brincos con cualquier instrucción que cambie las banderas.

Lista

Instruction	Description	Signedness	Flags
JO	Jump if overflow		OF = 1
JNO	Jump if not overflow		OF = 0
JS	Jump if sign		SF = 1
JNS	Jump if not sign		SF = 0
JE JZ	Jump if equal Jump if zero		ZF = 1
JNE JNZ	Jump if not equal Jump if not zero		ZF = 0
JB JNAE JC	Jump if below Jump if not above or equal Jump if carry	unsigned	CF = 1
JNB JAE JNC	Jump if not below Jump if above or equal Jump if not carry	unsigned	CF = 0
JBE JNA	Jump if below or equal Jump if not above	unsigned	CF = 1 or ZF = 1
JA JNBE	Jump if above Jump if not below or equal	unsigned	CF = 0 and ZF = 0
JL JNGE	Jump if less Jump if not greater or equal	signed	SF <> OF
JGE JNL	Jump if greater or equal Jump if not less	signed	SF = OF
JLE JNG	Jump if less or equal Jump if not greater	signed	ZF = 1 or SF <> OF
JG JNLE	Jump if greater Jump if not less or equal	signed	ZF = 0 and SF = OF
JP JPE	Jump if parity Jump if parity even		PF = 1
JNP JPO	Jump if not parity Jump if parity odd		PF = 0
JCXZ JECXZ	Jump if CX register is 0 Jump if ECX register is 0		CX = 0 ECX = 0

Fuente: http://euler.mat.uson.mx/~havillam/ca/Common/x86/flags-jumps.pdf

Ejemplo

```
Label:
    <instrucciones>
    dec ecx, 1
    jnz Label
```

- Equivale a lo siguiente en C / Java:

```
do {
    <instrucciones>
    ecx--;
} while (ecx != 1);
```

Ejemplo

```
for (i = 0; i < 10; i++)
    a[i]++;
```

```
        mov eax, 0          ; eax = i = 0
L1:     cmp eax, 10         ; if (i < 10)
        jnl L2              ; jump to L2 if i >= 10
        inc [ebx]           ; Mem[ebx](= a[i])++
        add ebx, 4          ; ebx = &a[i+1]
        inc eax             ; eax++
        jmp L1              ; goto L1
L2:     ...
```

Instrucción loop

- Decrementa el registro ecx y brinca a la etiqueta si no es cero.
- El mismo ejemplo:

```
        mov ecx, 10
L1:     inc[ebx]
        add ebx, 4
        loop L1
```

Funciones

- Instrucciones:

```
call Foo        ; push return address on stack
                ; and go to Foo
ret             ; pop return address from stack
                ; and jump to it
```

- rbp se usa como un frame pointer para apuntar dentro del stack frame y accesar las variables locales.
- rsp es el apuntador al tope de la pila.

Funciones

- Instrucciones especiales:

```
push eax        ; esp -= 4, Mem[esp] = eax
pop  eax        ; eax = Mem[esp], esp += 4
```

Operaciones de strings

- Operan en cadenas (strings) de bytes.

Instrucción	Descripción
cmps	Compara strings
lods	Carga string
movs	Mueve string
scas	Scan string
stos	Almacena string

- Hay variantes para cadenas de bytes, words, dwords y quadwords.

Prefijos de repetición

- Repiten la instrucción de string según la condición.

Prefijo	Descripción
rep	Repite hasta que ecx sea cero
repne	Repite mientras no sea igual (hasta que ecx sea 0 o mientras ZF = 0)
repnz	Repite mientras no sea cero (hasta que ecx sea 0 o mientras ZF = 0)
repe	Repite mientras sea igual (hasta que ecx sea 0 o mientras ZF = 1)
repz	Repite mientras sea cero (hasta que ecx sea 0 o mientras ZF = 1)

- **Nota**: repne y repnz son iguales; repe y reps son iguales.

Instrucción cmps

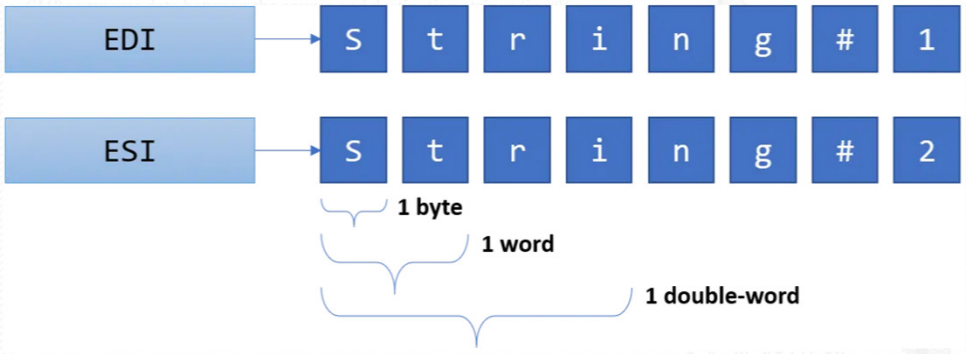
- Compara los datos apuntados por el registro (R|E)SI con los apuntados por (R|E)DI.
- El registro de banderas cambia de acuerdo a la comparación.
- (R|E)SI y (R|E)DI se incrementan si DF = 0 o se decrementan si DF = 1.
- Se pueden comparar bytes (cmpsb), palabras (cmpsw), palabras dobles (cmpsd) o palabras cuádruples (cmpsq).

Instrucción cmps

- Ejemplo:

```
cmpsb    ; los argumentos son implícitos
```

- Esto solo compara un byte.



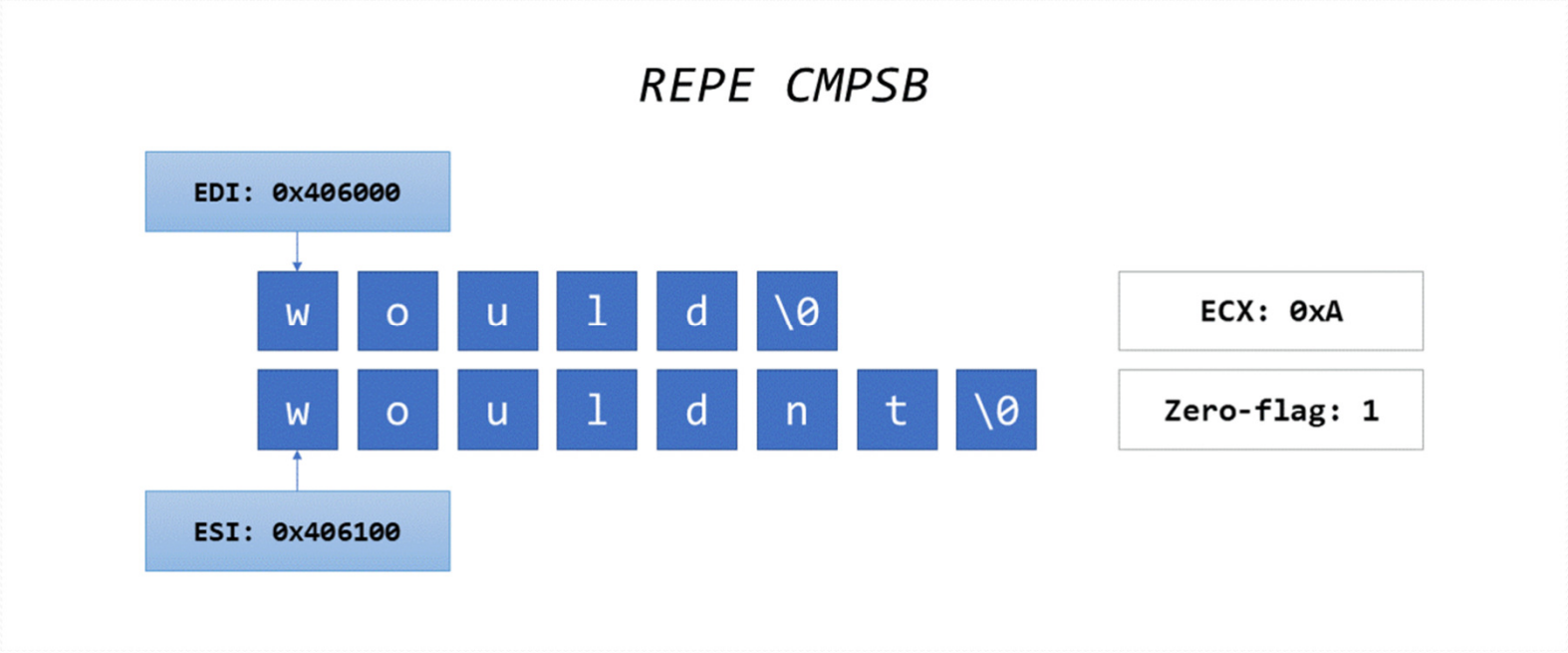
Fuente: https://medium.com/@ophirharpaz/a-summary-of-x86-string-instructions-87566a28c20c

Prefijos de repetición

- Se utilizan para comparar dos strings completos.
- Ejemplo:

```
mov ecx, 0ah
repe cmpsb      ; repite hasta que ecx sea 0 o mientras ZF = 1
```

Comparación de dos strings



Fuente: https://medium.com/@ophirharpaz/a-summary-of-x86-string-instructions-87566a28c20c

Instrucción movs

- Mueve datos de la dirección (r|e)si a (r|e)di.
- Si la bandera DF es 0, (r|e)si y (r|e)di se incrementan, si DF es 1 (r|e)si y (r|e)di se decrementan.
- Variantes: movsb (mueve un byte), movsw (mueve una palabra), movsd (mueve una dword), movsq (mueve una quadword).
- Algunas programas ensambladores aceptan especificar el tamaño explícitamente.

Instrucción movs

- Ejemplos:

  movs dword ptr [edi], dword ptr [esi]

  movs byte ptr [edi], byte ptr [esi]

- **Nota**: edi y esi se incrementan o decrementan según el valor de DF.

Instrucción movs

- Ejemplo: mover 4 enteros desde la dirección apuntada por esi a la dirección apuntada por edi.

```
mov ecx, 4
rep movs dword ptr [edi], dword ptr [esi]
```

- Otra opción es usar movsd que tiene sus argumentos implícitos:

```
mov ecx, 4
rep movsd
```

Ensambladores para x86

- FASM (Flat Assembler): Windows, Linux, Unix-like.
- GAS (GNU Assembler): Unix-like, Windows.
- MASM (Microsoft Macro Assembler): Windows.
- NASM (Netwide Assembler): Linux, macOS, Windows.

Hola mundo en macOS

```
; ------------------------------------------------------------------------------
; Runs on 64-bit macOS only.
;     nasm -fmacho64 hello.asm
;     ld -macosx_version_min 12.6.0 -
L/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/lib -lSystem -o hello hello.o
; ------------------------------------------------------------------------------
            global    _main
            section   .text
_main:      mov       rax, 0x02000004         ; system call for write
            mov       rdi, 1                  ; file handle 1 is stdout
            mov       rsi, message            ; address of string to output
            mov       rdx, 13                 ; number of bytes
            syscall                           ; invoke operating system to do the write
```

Hola mundo en macOS

```
                mov     rax, 0x02000001     ; system call for exit
                xor     rdi, rdi            ; exit code 0
                syscall                     ; invoke operating system to exit
                section .data
message:        db      "Hello, World", 10  ; note the newline at the end
```

Ejemplo C y ensamblador

```
// File: max3.c, runs on 64-bit macOS
//     gcc -o max3  max3.c max3.o
#include <stdio.h>
long maxofthree(long, long, long);
int main()
{
   printf("%ld\n", maxofthree(1, -4, -7));
   printf("%ld\n", maxofthree(2, -6, 1));
   printf("%ld\n", maxofthree(2, 3, 1));
   printf("%ld\n", maxofthree(-2, 4, 3));
   printf("%ld\n", maxofthree(2, -6, 5));
   printf("%ld\n", maxofthree(2, 4, 6));
   return 0;
}
```

Ejemplo C y ensamblador

```
; ------------------------------------------------------------------------
; File: max3.asm, returns the maximum value of its three 64-bit integer
;     nasm -fmacho64 max3.asm
; Parameters are passed in rdi, rsi, and rdx. The value is returned in rax.
; ------------------------------------------------------------------------
        global  _maxofthree
        section .text
_maxofthree:
        mov     rax, rdi        ; result (rax) initially holds x
        cmp     rax, rsi        ; is x less than y?
        cmovl   rax, rsi        ; if so, set result to y
        cmp     rax, rdx        ; is max(x, y) less than z?
        cmovl   rax, rdx        ; if so, set result to z
        ret                     ; the max will be in rax
```

Bibliografía

- Modern x86 Assembly Language Programming
- Daniel Kusswurm
- Apress, 2018

- Computer Organization and Design, 5th edition
- David A. Patterson, John L. Hennessy
- Morgan Kaufmann, 2014