

x86 Overview

Introducción

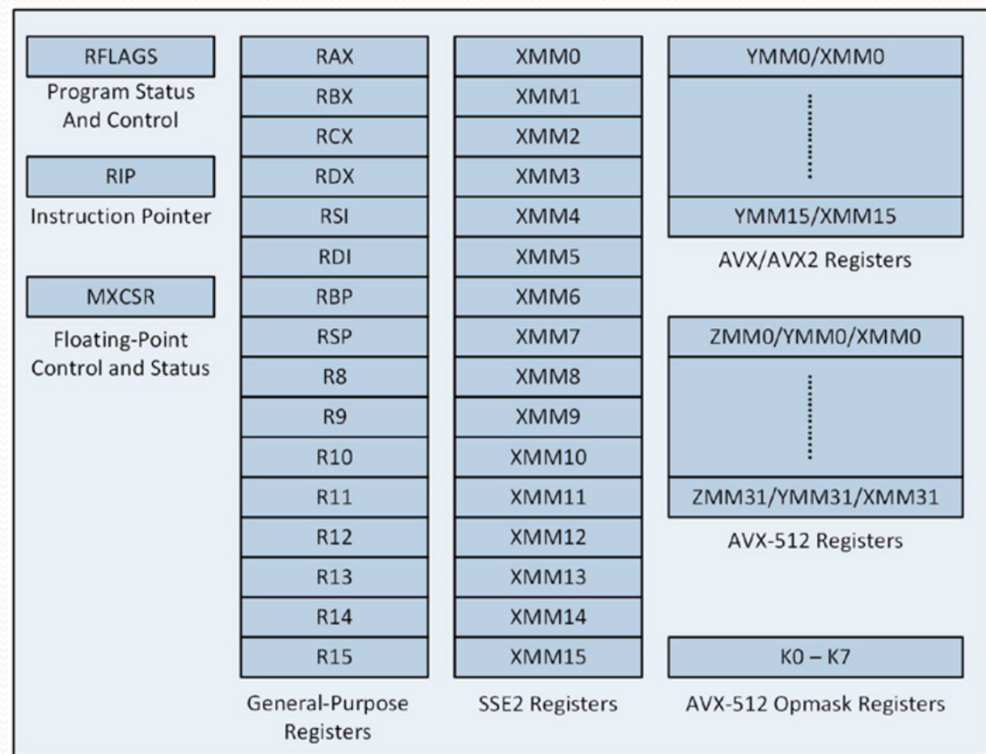
- Hay dos versiones de x86:
 1. IA-32 (Intel Architecture, 32 bits). Versión de 32 bits del ISA x86. También se le conoce como i386.
 2. x86-64. Versión de 64 bits del ISA x86. También se le conoce como x64, AMD64 o Intel 64.
- Primer procesador con x86: Intel 8086 (1978), un procesador de 16 bits.

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

63	31	0	15	7	0
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	

64-Bit and 32-Bit Registers

16-Bit and 8-Bit Registers

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 $\leftarrow$ eax + ebx en sintaxis Intel**
- Hay otras diferencias.

Sintaxis Intel vs Sintaxis AT&T

	Intel	AT&T
Orden de operandos	destino, fuente	fuentes, 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

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



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).

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

xor rax, rax

mov rcx, rax

mov qword [rsp - 56], 0

mov rax, qword [rsi - 4]

add eax, dword [edi]

; rsp $\leftarrow$ rsp + 20

; rax $\leftarrow$ rax ^ rax

; rcx $\leftarrow$ rax

; Mem[rsp - 56] $\leftarrow$ 0

; rax $\leftarrow$ Mem[rsi - 4]

; eax $\leftarrow$ eax + Mem[edi]

Addressing Modes

- Una instrucción x86-64 requiere hasta cuatro componentes para especificar la ubicación de un operando en la memoria.
- Los cuatro componentes son un offset, un registro base, un registro índice y un factor de escala.
- La dirección efectiva para un operando de memoria es:
- $\text{EffectiveAddress} = \text{BaseReg} + \text{IndexReg} * \text{ScaleFactor} + \text{Disp}$

Addressing Modes

- Inmediato:

mov rax, 10 ; rax $\leftarrow$ 10

- Directo:

mov eax, X ; eax $\leftarrow$ Mem[&X]

X dw 3

- Registro:

mov rax, rbx ; rax $\leftarrow$ rbx

- Indirecto con registro:

mov eax, [ebx] ; eax $\leftarrow$ Mem[ebx]

Addressing Modes

- 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 Modes

- Base con índice escalado:

`mov rax, [rbx + rcx * 4]` ; $\text{rax} \leftarrow \text{Mem}[\text{rbx} + \text{rcx} * 4]$

- Base con índice escalado y offset:

`mov rax, [rbx + rcx * 4 + 48]` ; $\text{rax} \leftarrow \text{Mem}[\text{rbx} + \text{rcx} * 4 + 48]$

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	<code>mov rax,42</code>	<code>rax = 42</code>
	<code>imul r12,-47</code>	<code>r12 *= -47</code>
	<code>shl r15,8</code>	<code>r15 <<= 8</code>
	<code>xor ecx,80000000h</code>	<code>ecx ^= 0x80000000</code>
	<code>sub r9b,14</code>	<code>r9b -= 14</code>
Register	<code>mov rax,rbx</code>	<code>rax = rbx</code>
	<code>add rbx,r10</code>	<code>rbx += r10</code>
	<code>mul rbx</code>	<code>rdx:rax = rax * rbx</code>
	<code>and r8w,0fff0h</code>	<code>r8w &= 0xfff0</code>
Memory	<code>mov rax,[r13]</code>	<code>rax = *r13</code>
	<code>or rcx,[rbx+rsi*8]</code>	<code>rcx = *(rbx+rsi*8)</code>
	<code>sub qword ptr [r8],17</code>	<code>*(long long*)r8 -= 17</code>
	<code>shl word ptr [r12],2</code>	<code>*(short*)r12 <<= 2</code>

Fuente: Mx86ALP, p. 11

x86-32 vs x86-64

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

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	Jump if equal		ZF = 1
JZ	Jump if zero		ZF = 1
JNE	Jump if not equal		ZF = 0
JNZ	Jump if not zero		ZF = 0
JB	Jump if below	unsigned	CF = 1
JNAE	Jump if not above or equal		
JC	Jump if carry		
JNB	Jump if not below	unsigned	CF = 0
JA	Jump if above or equal		
JNC	Jump if not carry		
JBE	Jump if below or equal	unsigned	CF = 1 or ZF = 1
JNA	Jump if not above		
JA	Jump if above	unsigned	CF = 0 and ZF = 0
JNBE	Jump if not below or equal		
JL	Jump if less	signed	SF $\neq$ OF
JNGE	Jump if not greater or equal		
JGE	Jump if greater or equal	signed	SF = OF
JNL	Jump if not less		
JLE	Jump if less or equal	signed	ZF = 1 or SF $\neq$ OF
JNG	Jump if not greater		
JG	Jump if greater	signed	ZF = 0 and SF = OF
JNLE	Jump if not less or equal		
JP	Jump if parity		PF = 1
JPE	Jump if parity even		
JNP	Jump if not parity		PF = 0
JPO	Jump if parity odd		
JCXZ	Jump if CX register is 0		CX = 0
JECXZ	Jump if ECX register is 0		ECX = 0

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

Ejemplo do while

```
do {  
    <instrucciones>  
    ecx--;  
} while (ecx != 0);
```

- x86:

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

Ejemplo for

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

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

Instrucción loop

- El mismo ejemplo del for:

```
mov ecx, 10
```

```
L1: inc[ebx]
```

```
add ebx, 4
```

```
loop L1
```

- La instrucción loop decrementa el registro `ecx` y brinca a la etiqueta si no es cero.

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 acceder 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 <code>ecx</code> sea cero
repne	Repite mientras no sea igual (hasta que <code>ecx</code> sea 0 o mientras <code>ZF</code> = 0)
repnz	Repite mientras no sea cero (hasta que <code>ecx</code> sea 0 o mientras <code>ZF</code> = 0)
repe	Repite mientras sea igual (hasta que <code>ecx</code> sea 0 o mientras <code>ZF</code> = 1)
repz	Repite mientras sea cero (hasta que <code>ecx</code> sea 0 o mientras <code>ZF</code> = 1)

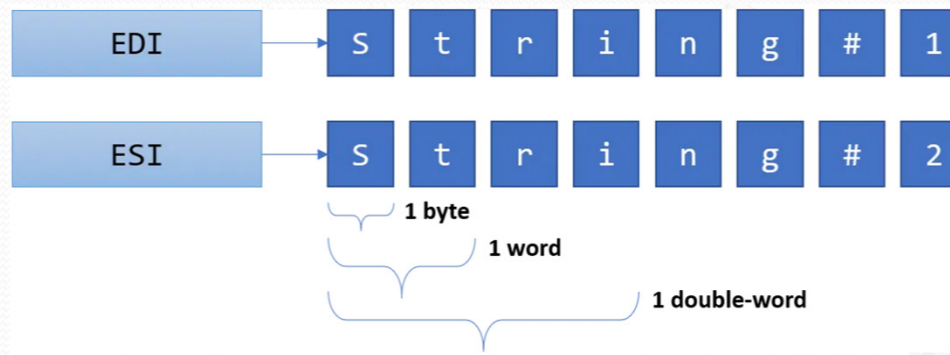
- **Nota:** `repne` y `repnz` son iguales; `repe` y `repz` son iguales.

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

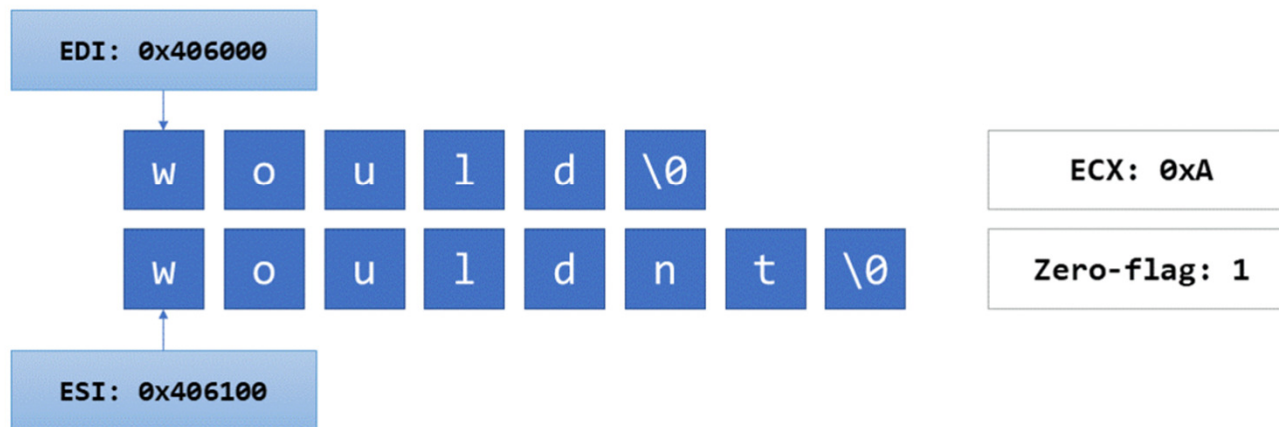
- Se utilizan para comparar dos strings completos.
- Ejemplo:

`mov ecx, 0ah`

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

Comparación de dos strings

REPE CMPSB



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).
- Algunos 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
```



Más información

Modern x86 Assembly Language Programming

Daniel Kusswurm

Apress, 2018