

¿cómo hacer saltos condicionales?

Con instrucciones que inspeccionan rflags. Banderas:

Bit	Abr	Descripción	Categoría
0	CF	Carry flag	Status
1	1	Reserved	
2	PF	Parity flag	Status
3	0	Reserved	
4	AF	Adjust flag	Status
5	0	Reserved	
6	ZF	Zero flag	Status
7	SF	Sign flag	Status
8	TF	Trap flag (single step)	Control
9	IF	Interrupt enable flag	Control
10	DF	Direction flag	Control
11	OF	Overflow flag	Status

Más banderas

Bit:	Abr.	Descripción	
12-13	IOPL	I/O privilege level (286+ only), always 1 on 8086 and 186	System
14	NT	Nested task flag (286+ only), always 1 on 8086 and 186	System
15	0	Reserved, always 1 on 8086 and 186, always 0 on later models	

----- EFLAGS -----

16	RF	Resume flag (386+ only)	System
17	VM	Virtual 8086 mode flag (386+ only)	System
18	AC	Alignment check (486SX+ only)	System
19	VIF	Virtual interrupt flag (Pentium+)	System
20	VIP	Virtual interrupt pending (Pentium+)	System
21	ID	Able to use CUID instruction (Pentium+)	System
22-63	0	Reserved	

Manipulando las banderas

- Lo más usual: **indirectamente** (la mayoría de las operación aritméticas o lógicas reflejan su resultado en rflags).
- Instrucciones para manipular bits: `setc`, **`std`**, `setz`, `seto`...
- Copiar de/hacia **ah**: `lahf` y `sahf` y de/hacia la pila: `pushfq` y `popq`
- `cmpS o,d` hace la resta $d - o$ (igual que `subS o,d`), actualizando rflags pero sin cambiar d
- `testS o,d` hace $d \text{ AND } o$ (igual que `andS o,d`), actualizando rflags pero sin cambiar d

Instrucciones que usan el CF (carry)

`adcS o,d`: add with Carry ($d=o+CF$)

`rc1S o,d`: rotate left with carry

`rcrS o,d`: rotate right with carry

Volviendo a los saltos condicionales

Algo sencillo:

- Iterar hasta que algo sea cero (ver ejemplo en apunte, pág 19.)

¿cómo se trata signed/unsigned?

- Para la suma y la resta hay una sola versión: add y sub* ¿cómo tratar los casos negativos?

Con las banderas de rflags: ej.:

```
movb $1,    %al  
adddb $0x0f,%al
```

0	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

0	0	0	0	1	1	1	1
---	---	---	---	---	---	---	---

0	0	0	1	0	0	0	0
---	---	---	---	---	---	---	---

Un caso más complicado

		Signed	Unsigned								
movb \$0x6f,%al	<table><tr><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	0	1	1	0	1	1	1	1	111	111
0	1	1	0	1	1	1	1				
addb \$0x80,%al	<table><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	0	0	0	0	0	0	-128	128
1	0	0	0	0	0	0	0				
<hr/>											
	<table><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr></table>	1	1	1	0	1	1	1	1	-17	239
1	1	1	0	1	1	1	1				

OF = 0, CF = 0, SF = 1

Un caso con acarreo

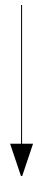
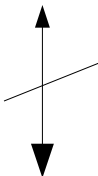
		Signed	Unsigned								
movb \$0xb0,%al	<table><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	1	0	0	0	0	0	0	-64	192
1	1	0	0	0	0	0	0				
addb \$-32,%al	<table><tr><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	1	1	0	0	0	0	0	-32	224
1	1	1	0	0	0	0	0				
	<div><table><tr><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table></div>	1	1	0	1	0	0	0	0	-96	160
1	1	0	1	0	0	0	0				
			+ 256								
			<hr/>								
			416								

1


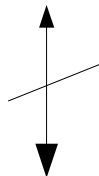
↓

OF = 0, CF = 1, SF = 1

Un caso más feo

		Signed	Unsigned								
movb \$-64,%a1	<table><tr><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	1	0	0	0	0	0	0	-64	192
1	1	0	0	0	0	0	0				
addb \$-96,%a1	<table><tr><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	1	0	0	0	0	0	-96	160
1	0	1	0	0	0	0	0				
<hr/>											
	<table><tr><td>1</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	1	1	0	0	0	0	96	96
1	0	1	1	0	0	0	0				
			+ 256								
OF = 1, CF = 1, SF = 0		-160	<hr/> 352								

Última combinación

		Signed	Unsigned								
movb \$64,%a1	<table><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	0	0	0	0	0	64	64
0	1	0	0	0	0	0	0				
addb \$64,%a1	<table><tr><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	0	1	0	0	0	0	0	0	64	64
0	1	0	0	0	0	0	0				
<hr/>											
	<table><tr><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr></table>	1	0	0	0	0	0	0	0	-128	128
1	0	0	0	0	0	0	0				
											
OF = 1, CF = 0, SF = 1		128									

Entonces ¿qué usar?

<http://www.unixwiz.net/techtips/x86-jumps.html>

Productos y cocientes

Unsigned: mul y div

- mul: destinos: AX, DX:AX, EDX:EAX, RDX:RAX
- div: orígenes: los mismos destinos que el mul.
Resultados:
 - En la parte alta: resto
 - En la parte baja: cociente
- Ej.: `mulw %bx # bx * ax`, resultado en dx:ax
- Ej.: `divw %bx # dx:ax / bx`, div en ax, resto en dx

Productos y cocientes

Signed: imul e idiv

- Similares a mul y div, pero además permiten otras combinaciones (¡¡y hasta 3 operandos!!)