Programación a nivel máquina II: Control (contenido alternativo)

Estructura de Computadores
Semana 4

Bibliografía:

[BRY16] Cap.3 Computer Systems: A Programmer's Perspective 3 rd ed. Bryant, O'Hallaron. Pearson, 2016
Signatura ESIIT/C.1 BRY com

Transparencias del libro CS:APP, Cap.3

Introduction to Computer Systems: a Programmer's Perspective

Autores: Randal E. Bryant y David R. O'Hallaron

http://www.cs.cmu.edu/afs/cs/academic/class/15213-f15/www/schedule.html

Guía de trabajo autónomo (4h/s)

- **Lectura**
  - Control § 3.6, pp. 236-274
- **Ejercicios**
  - Probl. 3.13 – 3.14 § 3.6.2, pp.240, 241
  - Probl. 3.15 § 3.6.4, pp.245†
  - Probl. 3.16 – 3.18 § 3.6.5, pp.248 2 , 249
  - Probl. 3.19 – 3.21 § 3.6.6, pp.252‡ , 255_2
  - Probl. 3.22 – 3.29 § 3.6.7, pp.257, 258, 260, 262, 264, 267_2 , 268
  - Probl. 3.30 – 3.31 § 3.6.8, pp.272, 273

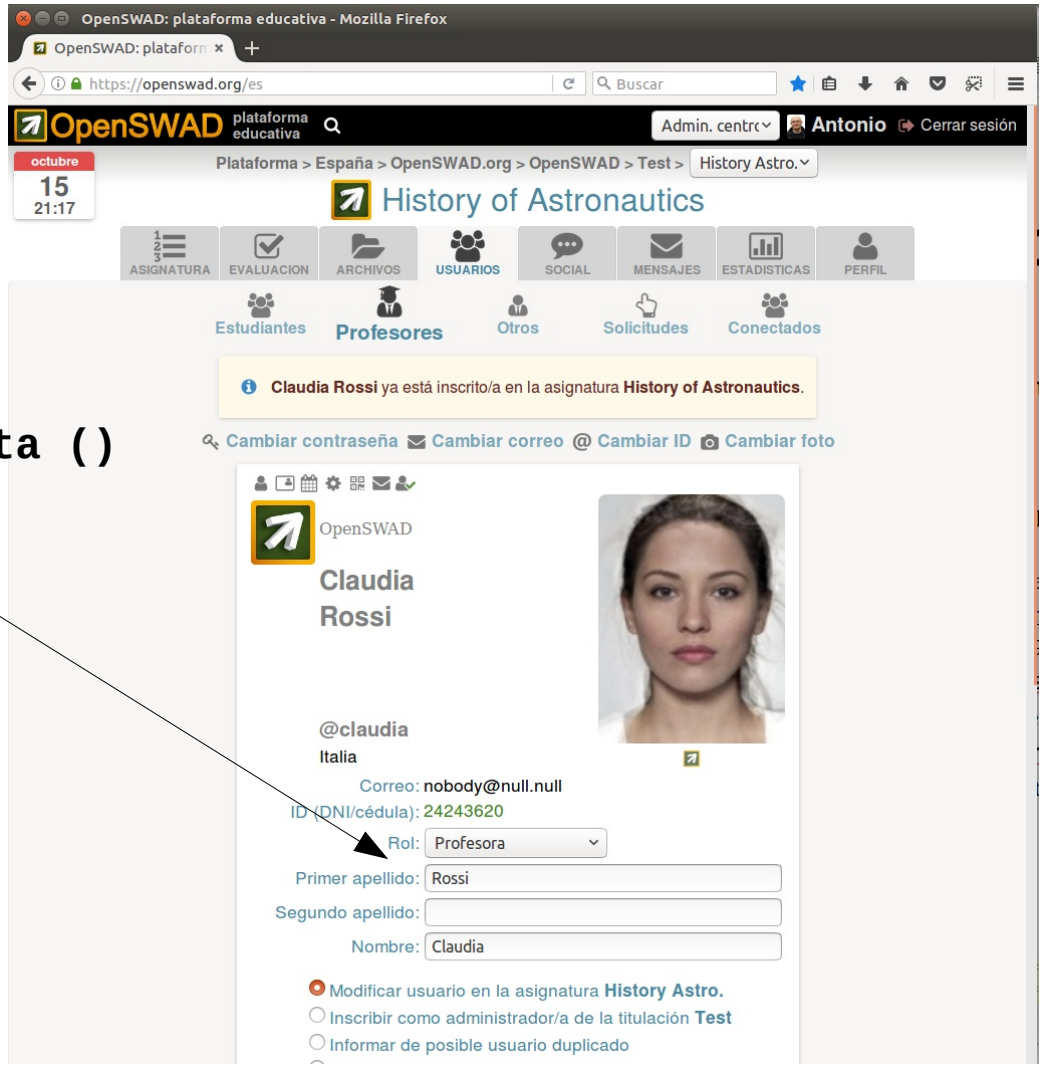
† direccionamiento relativo a contador de programa, “PC-relative”
‡ penalización por predicción saltos

Progr. a nivel máquina II: Control

- Partiendo de un código real
- Instrucciones de comparación
- Saltos y ajustes condicionales
  - Bits de estado
  - Sumas y restas sin signo/con signo
  - Volviendo a los saltos y ajustes
- Instrucciones de copia condicional
- Otras instrucciones condicionales
- Traducción de sentencias condicionales
- Traducción de bucles

Partiendo de un código real

- SWAD
  - Módulo:
    - `swad_user.c`
  - Función:
    - `Usr_IcanChangeOtherUsrData ()`



Código C

```
/*****************************************************************************/
/*************** Check if I can change another user's data *******************/
/*****************************************************************************/

bool Usr_ICanChangeOtherUsrData (const struct UsrData *UsrDat)
  {
   if (UsrDat->UsrCod == Gbl.Usrs.Me.UsrDat.UsrCod)    // It's me
      return true;

   /***** Check if I have permission to see another user's IDs *****/
   switch (Gbl.Usrs.Me.Role.Logged)
     {
      case Rol_TCH:
         /* Check 1: I can change data of users who do not exist in database */
         if (UsrDat->UsrCod <= 0) // User does not exist (creating a new user)
            return true;

         /* Check 2: I change data of users without password */
         if (!UsrDat->Password[0])    // User has no password (never logged)
            return true;

         return false;
      case Rol_DEG_ADM:
      case Rol_CTR_ADM:
      case Rol_INS_ADM:
      case Rol_SYS_ADM:
         return Usr_ICanEditOtherUsr (UsrDat);
      default:
         return false;
     }
  }
```

Código ensamblador

```
gcc -O2 -S swad_user.c -o swad_user.s
```

```
    .globl  Usr_IcanChangeOtherUsrData

Usr_ICanChangeOtherUsrData:
    movq    (%rdi), %rax
    cmpq    Gbl+84936(%rip), %rax
    je      .L556
    movl    Gbl+97440(%rip), %edx
    cmpl    $5, %edx
    je      .L552
    jb      .L555
    cmpl    $9, %edx
    ja      .L555
    jmp     Usr_ICanEditOtherUsr
.L555:
    xorl    %eax, %eax
    ret
.L552:
    testq   %rax, %rax
    jle     .L556
    cmpb    $0, 345(%rdi)
    sete    %al
    ret
.L556:
    movl    $1, %eax
    ret
```

Traducción (1)

1

C
```
bool Usr_ICanChangeOtherUsrData (const struct UsrData *UsrDat)
  {
   if (UsrDat->UsrCod == Gbl.Usrs.Me.UsrDat.UsrCod)    // It's me
      return true;
```

Ensamblador
```
    movq    (%rdi), %rax              # UsrDat->UsrCod
    cmpq    Gbl+84936(%rip), %rax     # Gbl.Usrs.Me.UsrDat.UsrCod
    je      .L556                     # ==

...

.L556:
    movl    $1, %eax                  # true
    ret
```

Traducción (2)

2

C

```
switch (Gbl.Usrs.Me.Role.Logged)
  {
   case Rol_TCH:
      /* Check 1: I can change data of users
         who do not exist in database */
      if (UsrDat->UsrCod <= 0)     // User does not exist
                                   // (creating a new user)
         return true;
      /* Check 2: I change data of users without password */
      if (!UsrDat->Password[0])    // User has no password
                                   // (never logged)
         return true;
      return false;
```

Ensamblador

```
    movl    Gbl+97440(%rip), %edx    # Gbl.Usrs.Me.Role.Logged
    cmpl    $5, %edx                 # Rol_TCH
    je      .L552                    # ==
...
.L552:
    testq   %rax, %rax               # UsrDat->UsrCod
    jle     .L556                    # <= 0
    cmpb    $0, 345(%rdi)            # UsrDat->Password[0]
    sete    %al                      # == 0 ==> true
    ret                              # != 0 ==> false
...
.L556:
    movl    $1, %eax                 # true
    ret
```

Traducción (3)

3

C

```
case Rol_DEG_ADM:
case Rol_CTR_ADM:
case Rol_INS_ADM:
case Rol_SYS_ADM:
   return Usr_ICanEditOtherUsr (UsrDat);
default:
   return false;
```

Ensamblador

```
    jb      .L555       # Gbl.Usrs.Me.Role.Logged < Rol_TCH
    cmpl    $9, %edx
    ja      .L555       # Gbl.Usrs.Me.Role.Logged > Rol_SYS_ADM
    jmp     Usr_ICanEditOtherUsr
...
.L555:
    xorl    %eax, %eax  # false
    ret
```

Instrucciones de comparación

- **`cmp b,a`**
  - realiza la resta a – b
    - sin almacenar el resultado
    - afectando a los indicadores de estado.
- **`test b,a`**
  - realiza la operación and bit a bit a and b
    - sin almacenar el resultado
    - afectando a los indicadores de estado.

Saltos

Instrucciones de salto condicional en comparaciones		
Condición	Números sin signo	Números con signo
=	`je`	
≠	`jne`	
<	`jb, jnae`	`jl, jnge`
≥	`jae, jnb`	`jge, jnl`
>	`ja, jnbe`	`jg, jnle`
≤	`jbe, jna`	`jle, jng`
Instrucción de salto incondicional		
saltar siempre	`jmp`	

Ajustes condicionales

Instrucciones de salto condicional en comparaciones		
Condición	Números sin signo	Números con signo
=	sete	
≠	setne	
<	setb, setnae	setl, setnge
≥	setae, setnb	setge, setnl
>	seta, setnbe	setg, setnle
≤	setbe, setna	setle, setng

Bits de estado

- **CF** (*Carry Flag*): 1 si resultado fuera de rango sin signo
  - Suma: es una copia del acarreo en el bit más significativo
  - Resta: es la negación del acarreo en el bit más significativo
- **PF** (*Parity Flag*): 1 si núm. unos en LSB de resultado es par
- **ZF** (*Zero Flag*): 1 si resultado = 0
- **SF** (*Sign flag*): 1 si resultado < 0 en operaciones con signo
  - Es una copia del bit más significativo del resultado
- **OF** (*Overflow flag*): 1 si resultado fuera de rango con signo

  - `mov` nunca afecta a los indicadores de estado
  - `inc` y `dec` no afectan a CF

Sumas y restas sin/con signo

- Números sin/con signo
  - Representación de los negativos
  - Complemento a dos
- Desbordamiento sin signo (CF)
- Desbordamiento con signo (OF)
  - Cálculo del overflow
- Circuito para suma y resta

Números sin signo y con signo

- n = 8 bits → 2^8 combinaciones binarias:

sin signo		con signo
1111 1111 (255) ... 1000 0000 (128)		
0111 1111 (127) ... 0000 0000 (0)		0111 1111 (127) ... 0000 0000 (0)
		1111 1111 (-1) ... 1000 0000 (-128)

Representación de los negativos

número x		representación f(x)
–1	→	1111 1111 (256 – 1 = 255)
–2	→	1111 1110 (256 – 2 = 254)
–3	→	1111 1101 (256 – 3 = 253)
		...
–126	→	1000 0010 (256 – 126 = 130)
–127	→	1000 0001 (256 – 127 = 129)
–128	→	1000 0000 (256 – 128 = 128)

- En general: $f(x) = 2^n + x = 2^n - |x|$

Complemento a dos

- Llamamos a la representación de los negativos "representación en complemento a 2" o C_2:
  - Para x < 0:

$$C_2(|x|) = 2^n - |x|$$

- Realmente usamos aritmética modular
  - Para cualquier x:

$$\mathbf{C_2(x) = (2^n - x) \bmod 2^n}$$

Complemento a dos

- $x \geq 0$:
  - $C_2(x) = (2^n - x) \bmod 2^n = -x$
  - Ejemplo con $n = 8$ bits:

    $C_2(5)$ $= (256 - 5) \bmod 256 = 251 \bmod 256 = 1111\ 1011_2 =$ **-5**
- $x < 0$:
  - $C_2(x) = (2^n - x) \bmod 2^n = (2^n - (-|x|)) \bmod 2^n = (2^n + |x|) \bmod 2^n = |x| = -x$
  - Ejemplo con $n = 8$ bits:

    $C_2(-5)$ $= (256 + 5) \bmod 256 = 261 \bmod 256 = 1\ 0000\ 0101_2 \bmod 256 = 0000\ 0101_2 =$ **5**

Complemento a dos

- Cálculo rápido 1:
  - $\mathbf{C_2(x)} = 2^n - x = 2^n - x + 1 - 1 = ((2^n - 1) - x) + 1 =$ $(111...111 - x) + 1 = \mathbf{C_1(x) + 1}$
  - Pasos para calcular C_2:
    1. **Comp. a 1 (cambiar ceros por unos y unos por ceros)**
    2. **Sumar 1 al resultado**
  - Ejemplo:
    - $C_2(5) = \overline{0000\ 0101}+1 = 1111\ 1010+1 = 1111\ 1011$ (–5)

Complemento a dos

- Cálculo rápido 2:
  - Sumar 1 = buscar 1er. cero comenzando por la dcha. cambiando unos por ceros y ese 1er. cero por un uno.
  - Pasos para calcular C_2:
    1. **Buscar el 1er. uno comenzando por la dcha., copiando todos los ceros y ese 1er. uno.**
    2. **Complementar resto de bits hasta la izda.**
  - Ejemplo:
    - $C_2(68) = C_2(0110\ 1000) \rightarrow \ldots 1000 \rightarrow 1001\ 1000$ (–68)

Complemento a dos

- ¿Por qué se usa la representación en complemento a dos?
  - Porque permite usar el mismo circuito aritmético para suma sin signo, suma con signo y la resta.
    - Para sumar $x + y$, con $y < 0$, el sumador realmente calcula la suma $x + C_2(|y|)$:
      - $x + C_2(|y|) = x + (2^n - |y|) \bmod 2^n = x + 2^n - (-y) \bmod 2^n = (x + 2^n + y) \bmod 2^n = x + y$
    - Para restar $x - y$ el sumador calcula $x + (-y) = x + C_2(y)$

Desbordamiento sin signo

- **Ejemplo 1** con *n* = 8 bits:

```
   01100000 (2^6+2^5 =   64+32 =  96)
+  01100000 (2^6+2^5 =   64+32 =  96)
0  11000000 (2^7+2^6 = 128+64 = 192)
```

- CF = 0 → no acarreo, 192 resultado correcto
- Signo y overflow (SF = 1, OF = 1) irrelevantes en oper. sin signo

Desbordamiento sin signo

- **Ejemplo 2** con n = 8 bits:

  11000000(2^7+2^6 = 128+64 = 192)

  \+ **11000000**(2^7+2^6 = 128+64 = 192)

  1 **10000000**(2^7 = 128)

- CF = 1 → acarreo, 128 resultado incorrecto
- Signo y overflow (SF = 1, OF = 0) irrelevantes en oper. sin signo

Desbordamiento con signo

- **Ejemplo 1** con n = 8 bits:

  01100000$(2^6+2^5 = 64+32 = 96)$

  \+ **01100000**$(2^6+2^5 = 64+32 = 96)$

  0 **11000000**(-64) $C_2(11000000)=00111111+1=01000000(2^6=64)$

- SF = 1 → resultado negativo
- OF = 1 → desbord., –64 resultado correcto
- Acarreo (CF = 0) irrelevante en oper. con signo

Desbordamiento con signo

- **Ejemplo 2** con n = 8 bits:

  11000000(−64) $C_2(11000000)=00111111+1=01000000(2^6=64)$

  \+ **11000000**(−64) $C_2(11000000)=00111111+1=01000000(2^6=64)$

  1 **10000000**(−128) $C_2(10000000)=01111111+1=10000000(2^7=128)$

- SF = 1 → resultado negativo
- OF = 0 → no desbord., –128 resultado correcto
- Acarreo (CF = 1) irrelevante en oper. con signo

Cálculo del overflow (suma)

- $r = x + y$; OF = $\overline{s}_x \overline{s}_y s_r + s_x s_y \overline{s}_r = c_n \hat{} c_{n-1}$

Caso	s_x	s_y	s_r	OF	Acarreo c_n (bit n–1 a n) y c_{n-1} (bit n–2 a n–1)	Comentarios
1	0	0	0	0	$c_n = 0$, $c_{n-1} = 0$	Sumamos dos positivos. Resultado positivo → no desbordamiento.
2	0	0	1	**1**	**$c_n = 0$, $c_{n-1} = 1$**	Sumamos dos positivos. Resultado (incorrecto) negativo → desbordamiento
3	0	1	0	0	$c_n = 1$, $c_{n-1} = 1$	Sumamos un número positivo y uno negativo. Resultado siempre < que el nº positivo, y ≥ que el nº negativo → no desbordamiento.
4	0	1	1	0	$c_n = 0$, $c_{n-1} = 0$	
5	1	0	0	0	$c_n = 1$, $c_{n-1} = 1$	
6	1	0	1	0	$c_n = 0$, $c_{n-1} = 0$	
7	1	1	0	**1**	**$c_n = 1$, $c_{n-1} = 0$**	Sumamos dos negativos y el resultado (incorrecto) es positivo → desbordamiento.
8	1	1	1	0	$c_n = 1$, $c_{n-1} = 1$	Sumamos dos negativos y el resultado es negativo → no desbordamiento.

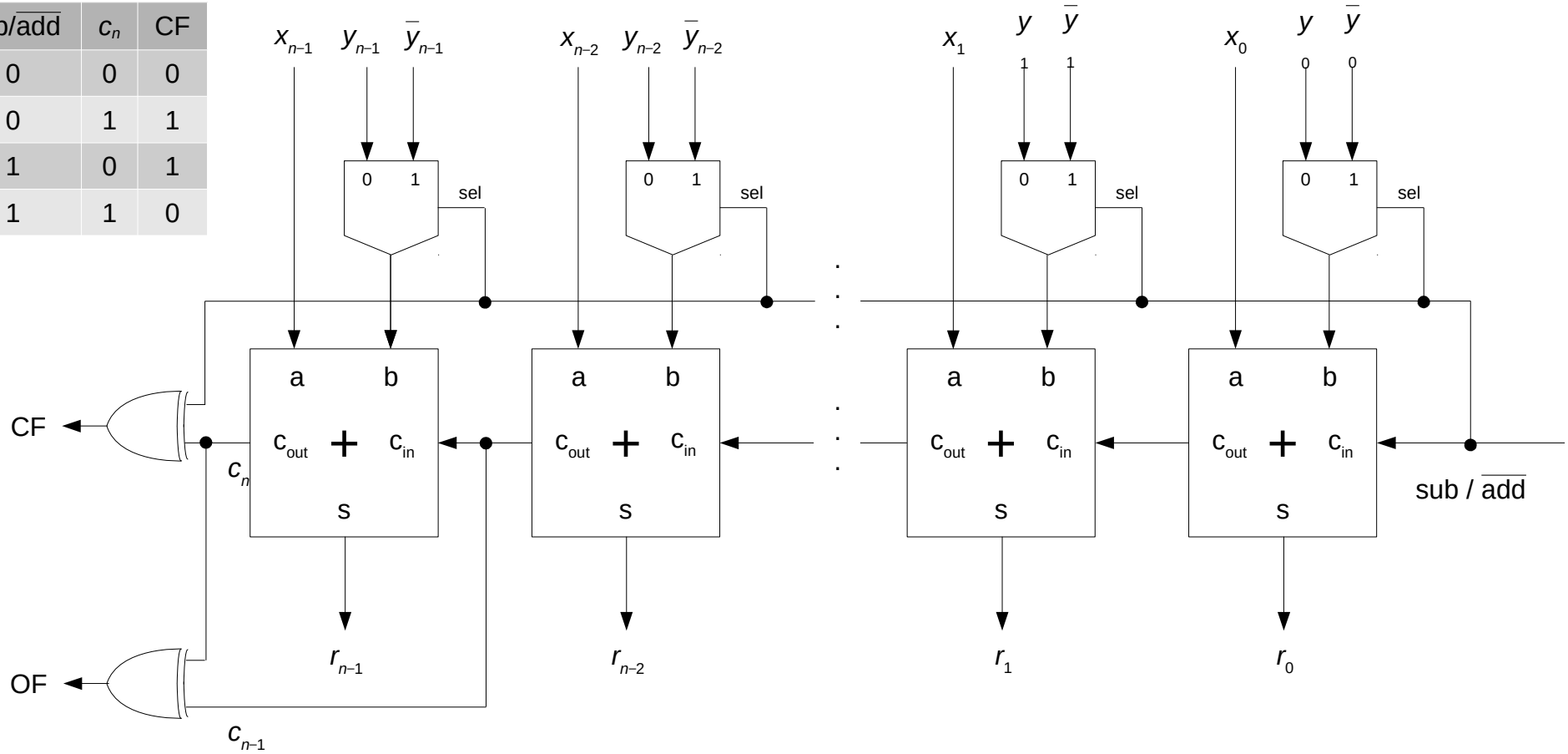
Cálculo del overflow (resta)

- $r = x - y = x + (-y)$; OF = $\overline{s}_x s_y s_r + s_x \overline{s}_y \overline{s}_r = c_n \hat{} c_{n-1}$

$x - y$				$x + (-y)$				OF	Acarreo c_n (bit n–1 a n) y c_{n-1} (bit n–2 a n–1)	Comentarios
Caso	s_x	s_y	s_r	Caso anterior	s_x	s_{-y}	s_r			
1	0	0	0	3	0	1	0	0	$c_n = 1$, $c_{n-1} = 1$	Positivo menos positivo ≈ sumar positivo y negativo. Resultado siempre < que el n.º positivo, y ≥ que el n.º negativo → no desbordamiento
2	0	0	1	4	0	1	1	0	$c_n = 0$, $c_{n-1} = 0$	
3	0	1	0	1	0	0	0	0	$c_n = 0$, $c_{n-1} = 0$	Positivo menos negativo ≈ sumar dos positivos. Resultado positivo → no desbordamiento
4	0	1	1	2	0	0	1	**1**	**$c_n = 0$, $c_{n-1} = 1$**	Positivo menos negativo ≈ sumar dos positivos. Resultado (incorrecto) negativo → desbordamiento
5	1	0	0	7	1	1	0	**1**	**$c_n = 1$, $c_{n-1} = 0$**	Negativo menos positivo ≈ sumar dos negativos. Resultado (incorrecto) positivo → desbordamiento
6	1	0	1	8	1	1	1	0	$c_n = 1$, $c_{n-1} = 1$	Negativo menos positivo ≈ sumar dos negativos. Resultado negativo → no desbordamiento
7	1	1	0	5	1	0	0	0	$c_n = 1$, $c_{n-1} = 1$	Negativo menos negativo ≈ sumar negativo y positivo. Resultado siempre < que el nº positivo, y ≥ que el nº negativo → no desbordamiento
8	1	1	1	6	1	0	1	0	$c_n = 0$, $c_{n-1} = 0$	

Circuito para suma y resta

	sub/$\overline{\text{add}}$	c_n	CF
add	0	0	0
	0	1	1
sub	1	0	1
	1	1	0



Volviendo a los saltos y ajustes

COND	Números sin signo		Números con signo				
	Instrucción	Condición a comprobar	Instrucción	Condición a comprobar			
=	**je** **sete**	A=B ↔ A–B=0 ↔ **ZF=1**	**je** **sete**	A=B ↔ A–B=0 ↔ **ZF=1**			
≠	**jne** **setne**	A≠B ↔ A–B≠0 ↔ **ZF=0**	**jne** **setne**	A≠B ↔ A–B≠0 ↔ **ZF=0**			
<	**jb=jnae** **setb=setnae**	A<B ↔ A–B → acarreo ↔ **CF=1**	**jl=jnge** **setl=setnge**	A<B ↔	SF=1 y OF=0 SF=0 y OF=1 (– – + = +)		↔ **SF≠OF**
≥	**jae=jnb** **setae=setnb**	A≥B ↔ A–B → no acarreo ↔ **CF=0**	**jge=jnl** **setge=setnl**	A≥B ↔	SF=0 y OF=0 SF=1 y OF=1 (+ – – = –)		↔ **SF=OF**
>	**ja=jnbe** **seta=setnbe**	A>B ↔	A≥B ↔ A–B → no acarreo ↔ **CF=0**	**jg=jnle** **setg=setnle**	A>B ↔	A≥B ↔	SF=0 y OF=0 SF=1 y OF=1 (+ – – = –)	↔ **SF=OF**
			y			**y**		
			A≠B ↔ A–B≠0 ↔ **ZF=0**			A≠B ↔ A–B≠0 ↔ **ZF=0**		
≤	**jbe=jna** **setbe=setna**	A≤B ↔	A<B ↔ A–B → acarreo ↔ **CF=1**	**jle=jng** **setle=setng**	A≤B ↔	A<B ↔	SF=1 o OF=0 SF=0 o OF=1 (– – + = +)	↔ **SF≠OF**
			o			**o**		
			A=B ↔ A–B=0 ↔ **ZF=1**			A=B ↔ A–B=0 ↔ **ZF=1**		

Volviendo a los saltos y ajustes

- Resumidamente:

COND	Números sin signo		Números con signo	
	Instrucción	Condición	Instrucción	Condición
=	`je=jz` `sete=setz`	**ZF**	`je=jz` `sete=setz`	**ZF**
≠	`jne=jnz` `setne=setnz`	**~ZF**	`jne=jnz` `setne=setnz`	**~ZF**
<	`jb=jnae=jc` `setb=setnae=setc`	**CF**	`jl=jnge` `setl=setnge`	**SF^OF**
≥	`jae=jnb=jnc` `setae=setnb=setnc`	**~CF**	`jge=jnl` `setge=setnl`	**~(SF^OF)**
>	`ja=jnbe` `seta=setnbe`	**~CF · ~ZF** **~(CF | ZF)**	`jg=jnle` `setg=setnle`	**~(SF^OF) · ~ZF** **~((SF^OF) | ZF)**
≤	`jbe=jna` `setbe=setna`	**CF | ZF**	`jle=jng` `setle=setng`	**(SF^OF) | ZF**

Otras instruc. de salto y ajuste

Condición	Instrucción	Descripción	Indicadores
Overflow	`jo` `seto`	jump/set if overflow	**OF**
No overflow	`jno` `setno`	jump/set if not overflow	**~OF**
< 0	`js` `sets`	jump/set if sign	**SF**
≥ 0	`jns` `setns`	jump/set if not sign	**~SF**
Paridad par	`jp=jpe` `setp=setpe`	jump/set if parity / jump/set if parity even	**PF** (8 bits menos signif. de resultado nº par de unos)
Paridad impar	`jnp=jpo` `setnp=setpo`	jump/set if not parity / jump/set if parity odd	**~PF** (8 bits menos signif. de resultado nº impar de unos)
Registro *CX = 0	`jcxz,jecxz,jrcxz`	jump if `cx`/`ecx`/`rcx` is zero	**~CX**, **~ECX**, **~RCX**

Instr. de copia (mov) condicional

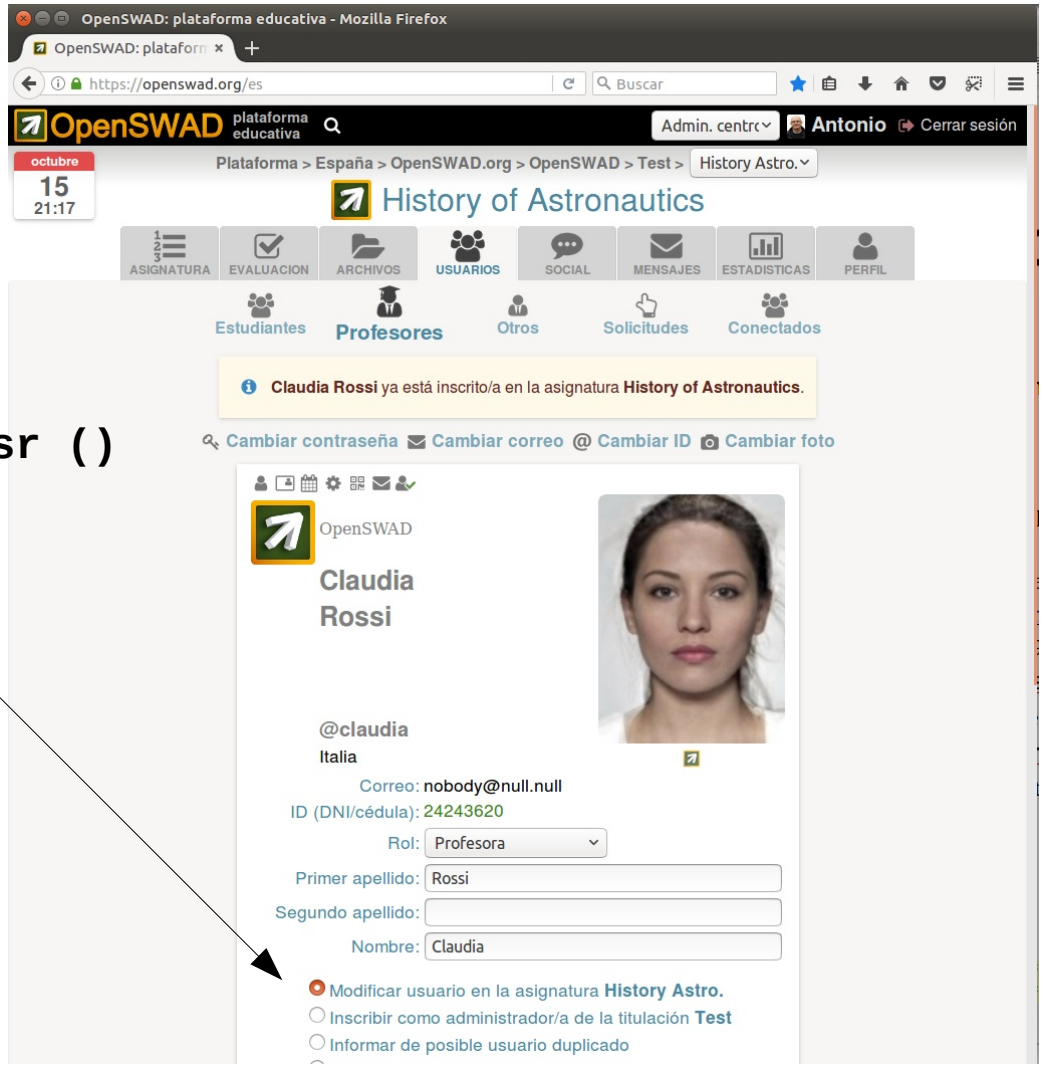
- Añadidas en 1995 (Pentium Pro)

COND	Números sin signo		Números con signo	
	Instrucción	Condición	Instrucción	Condición
=	`cmove=cmovz`	**ZF**	`cmove=cmovz`	**ZF**
≠	`cmovne=cmovnz`	**~ZF**	`cmovne=cmovnz`	**~ZF**
<	`cmovb=cmovnae=cmovc`	**CF**	`cmovl=cmovnge`	**SF^OF**
≥	`cmovae=cmovnb=cmovnc`	**~CF**	`cmovge=cmovnl`	**~(SF^OF)**
>	`cmova=cmovnbe`	**~CF · ~ZF**	`cmovg=cmovnle`	**~(SF^OF) · ~ZF**
≤	`cmovbe=cmovna`	**CF | ZF**	`cmovle=cmovng`	**(SF^OF) | ZF**

Condición	Instrucción	Indicadores
Overflow	`cmovo`	**OF**
No overflow	`cmovno`	**~OF**
< 0	`cmovs`	**SF**
≥ 0	`cmovns`	**~SF**
Paridad par	`cmovp=cmovpe`	**PF**
Paridad impar	`cmovnp=cmovpo`	**~PF**

Instr. de copia (mov) condicional

- SWAD
  - Módulo:
    - `swad_enrolment.c`
  - Función:
    - `Enr_PutActionsRegRemOneUsr ()`



Código C

```
bool Enr_PutActionsRegRemOneUsr (bool ItsMe)
  {
   ...

   /***** Register user in course / Modify user's data *****/
   if (Gbl.CurrentCrs.Crs.CrsCod > 0 &&
       Gbl.Usrs.Me.Role.Logged >= Rol_STD)
     {
      sprintf (Gbl.Alert.Txt,UsrBelongsToCrs ? (ItsMe ? Txt_Modify_me_in_the_course_X :
                                                        Txt_Modify_user_in_the_course_X) :
                                               (ItsMe ? Txt_Register_me_in_X :
                                                        Txt_Register_USER_in_the_course_X),
               Gbl.CurrentCrs.Crs.ShrtName);
      ...
```

Código ensamblador

```
gcc -O2 -S swad_enrolment.c -o swad_enrolment.s
```

```
    ...
    cmpq    $0, Gbl+142616(%rip)
    jle     .L596
    cmpl    $2, Gbl+97440(%rip)
    jbe     .L596
    testb   %r13b, %r13b
    jne     .L615
    testb   %r12b, %r12b
    movq    Txt_Register_me_in_X(%rip), %rcx
    cmove   Txt_Register_USER_in_the_course_X(%rip), %rcx
.L551:
    movl    $Gbl+142904, %r8d
    movl    $16384, %edx
    movl    $1, %esi
    movl    $Gbl+1988, %edi
    xorl    %eax, %eax
    ...
    call    __sprintf_chk
    ...
.L615:
    testb   %r12b, %r12b
    movq    Txt_Modify_me_in_the_course_X(%rip), %rcx
    cmove   Txt_Modify_user_in_the_course_X(%rip), %rcx
    jmp     .L551
    ...
```

Traducción (1, 2)

1

C

```
if (Gbl.CurrentCrs.Crs.CrsCod > 0 &&
    Gbl.Usrs.Me.Role.Logged >= Rol_STD)
```

Ensamblador

```
cmpq    $0, Gbl+142616(%rip) # Gbl.CurrentCrs.Crs.CrsCod
jle     .L596                # <= 0
cmpl    $2, Gbl+97440(%rip)  # Gbl.Usrs.Me.Role.Logged
jbe     .L596                # <= Rol_STD - 1
```

2

C

```
UsrBelongsToCrs ?
```

Ensamblador

```
testb   %r13b, %r13b         # UsrBelongsToCrs ?
jne     .L615                # false
```

Traducción (3, 4, 5)

3

C

```
(ItsMe ? Txt_Modify_me_in_the_course_X :
         Txt_Modify_user_in_the_course_X)
```

Ensamblador

```
.L615:
    testb   %r12b, %r12b                                  # ItsMe ?
    movq    Txt_Modify_me_in_the_course_X(%rip), %rcx      # true
    cmove   Txt_Modify_user_in_the_course_X(%rip), %rcx   # false
    jmp     .L551
```

4

C

```
(ItsMe ? Txt_Register_me_in_X :
         Txt_Register_USER_in_the_course_X)
```

Ensamblador

```
    testb   %r12b, %r12b                                  # ItsMe ?
    movq    Txt_Register_me_in_X(%rip), %rcx              # true
    cmove   Txt_Register_USER_in_the_course_X(%rip), %rcx # false
```

5

C

```
sprintf (Gbl.Alert.Txt,...,
         Gbl.CurrentCrs.Crs.ShrtName);
```

Ensamblador

```
.L551:
    movl    $Gbl+142904, %r8d    # Gbl.CurrentCrs.Crs.ShrtName
    movl    $16384, %edx         # strlen
    movl    $1, %esi             # flag (nivel de seguridad...)
    movl    $Gbl+1988, %edi      # Gbl.Alert.Txt
    xorl    %eax, %eax           # núm. regs. vectoriales usados
    ...                       # int __sprintf_chk(
    call    __sprintf_chk     #   char *str,int flag,size_t strlen,
                              #   const char *format,...);
```

Otras instruc. condicionales

- Instrucciones de **bucle**
  - rcx = rcx – 1
  - if rcx ≠ 0, saltar a etiqueta destino

Instrucción	Funcionamiento
`loop`	Decrementa el contador. Salta si el contador es ≠ 0
`loope=loopz`	Decrementa el contador. Salta si el contador es ≠ 0 y ZF = 1
`loopne=loopnz`	Decrementa el contador. Salta si el contador es ≠ 0 y ZF = 0

```
        mov n, %rcx
bucle: # cuerpo del bucle
        loop bucle
```

Otras instruc. condicionales

- Instrucciones de **comprobación de bits** individuales
  - bt n,x # copia el bit n de x en CF

Instrucción	Funcionamiento
`btc`	Comprueba un bit y lo complementa
`btr`	Comprueba un bit y lo pone a cero (*reset*)
`bts`	Comprueba un bit y lo pone a uno (*set*)

```
bt $4,%eax # ¿valor del bit 4?
```

Traducción de sentencias cond.

- Sentencia if then
- Sentencia if then else
  - Condición compuesta con AND
  - Condición compuesta con OR
- Operador condicional
- Sentencia switch
  - Árbol
  - Tabla

Sentencia if then

Sentencia en C	Estructura en ensamblador
if (*cond***)** *bloque-then*	*comprobar-cond* **jno-cond endif** *bloque-then* **endif:**

Sentencia if then

Código C	Código ensamblador

```c
static void if_then (int x,int y)
{
    if (x < y)
        puts ("then");

    puts ("the");
    puts ("end");
}
```

```
    .rodata
.LC0:   .string "then"
.LC1:   .string "the"
.LC2:   .string "end"
    .text
if_then:
    subq    $8, %rsp

    cmpl    %esi, %edi      # x : y ?
    jge     .L2             # x >= y

    movl    $.LC0, %edi     # then
    call    puts
.L2:
    movl    $.LC1, %edi     # the
    call    puts
    movl    $.LC2, %edi     # end
    call    puts

    addq    $8, %rsp
    ret
```

Sentencia if then else

Sentencia en C	Estructura en ensamblador

if (*cond***)**
  bloque-then
else
  bloque-else
	comprobar-cond
	***jno-cond* else**
	bloque-then
	jmp endif
	else:
	bloque-else
	endif:

Sentencia if then else

Código C	Código ensamblador

```c
static void if_then_else (int x,int y)
{
    if (x < y)
        puts ("then");
    else
        puts ("else");

    puts ("the");
    puts ("end");
}
```

```
    .rodata
.LC0:   .string "then"
.LC1:   .string "else"
.LC2:   .string "the"
.LC3:   .string "end"
    .text
if_then_else:
    subq    $8, %rsp

    cmpl    %esi, %edi  # x : y ?
    jge     .L2         # x >= y

    movl    $.LC0, %edi # then
    call    puts
    jmp     .L3
.L2:
    movl    $.LC1, %edi # else
    call    puts
.L3:
    movl    $.LC2, %edi # the
    call    puts
    movl    $.LC3, %edi # end
    call    puts

    addq    $8, %rsp
    ret
```

Condición compuesta con AND

- Atajo: si cond1 falsa → no comprobar cond2

Sentencia en C	Estructura en ensamblador

if (*cond1* **&&** *cond2***)**
    bloque-then
else
    bloque-else
	comprobar-cond1
	j*no-cond1* else
	comprobar-cond2
	j*no-cond2* else
	bloque-then
	jmp endif
	else:
	bloque-else
	endif:

Condición compuesta con AND

Código C	Código ensamblador

```c
static void
if_and_then_else (int x,int y,int z)
{
    if (x < y && y < z)
        puts ("then");
    else
        puts ("else");

    puts ("the");
    puts ("end");
}
```

```
    .rodata
.LC0:   .string "then"
.LC1:   .string "else"
.LC2:   .string "the"
.LC3:   .string "end"
    .text
if_and_then_else:
    subq    $8, %rsp

    cmpl    %esi, %edi   # x : y ?
    jge     .L2          # x >= y
    cmpl    %edx, %esi   # y : z ?
    jge     .L2          # y >= z

    movl    $.LC0, %edi  # then
    call    puts
    jmp     .L3
.L2:
    movl    $.LC1, %edi  # else
    call    puts
.L3:
    movl    $.LC2, %edi  # the
    call    puts
    movl    $.LC3, %edi  # end
    call    puts

    addq    $8, %rsp
    ret
```

Condición compuesta con OR

- Atajo: si cond1 verdad. → no comprobar cond2

Sentencia en C	Estructura en ensamblador

if (*cond1* **||** *cond2***)**
  bloque-then
else
  bloque-else
	comprobar-cond1
	j*cond1* then
	comprobar-cond2
	j*no-cond2* else
	then:
	  bloque-then
	  jmp endif
	else:
	  bloque-else
	endif:

Condición compuesta con OR

Código C	Código ensamblador

```
static void
if_or_then_else (int x,int y,int z)
{
    if (x < y || y < z)
        puts ("then");
    else
        puts ("else");

    puts ("the");
    puts ("end");
}
```

```
    .rodata
.LC0:   .string "then"
.LC1:   .string "else"
.LC2:   .string "the"
.LC3:   .string "end"
    .text
if_or_then_else:
    subq    $8, %rsp

    cmpl    %esi, %edi   # x : y ?
    jl      .L5          # x < y
    cmpl    %edx, %esi   # y : z ?
    jge     .L2          # y >= z
.L5:
    movl    $.LC0, %edi  # then
    call    puts
    jmp     .L4
.L2:
    movl    $.LC1, %edi  # else
    call    puts
.L4:
    movl    $.LC2, %edi  # the
    call    puts
    movl    $.LC3, %edi  # end
    call    puts

    addq    $8, %rsp
    ret
```

Operador condicional

- Se suele traducir aprovechando las instrucciones de copia condicional `cmov`*cc*

Sentencia en C	Estructura en ensamblador

`cond ? expresión-then : expresión-else`	`t = expresión-then`
	`result = expresión-else`
	`if (cond) result = t`

Código C	Código ensamblador

`int conditional_operator (int x,int y)`
`{`
`  return (x < y) ? y - x : x - y;`
`}`
	`  .globl conditional_operator`
	`conditional_operator:`
	`  movl  %esi, %edx # y`
	`  subl  %edi, %edx # t = y – x`
	`  movl  %edi, %eax # x`
	`  subl  %esi, %eax # result = x – y`
	`  cmpl  %esi, %edi # x : y ?`
	`  cmovl  %edx, %eax # if (x < y)`
	`  # result = t`
	`  ret`

Operador condicional

- Casos en los que no se debería usar `cmov`*`cc`*:
  - Cálculos costosos:

    ```
    val = Test(x) ? Hard1(x) : Hard2(x);
    ```

  - Cálculos arriesgados:

    ```
    val = p ? *p : 0;
    ```

  - Cálculos con efectos colaterales:

    ```
    val = x > 0 ? x*=7 : x+=3;
    ```

Sentencia switch: árbol

Código C

```
static void
switch_tree (int x) {
    switch (x) {
    case 100:
        puts ("100");
        break;
    case 200:
        puts ("200");
        break;
    case 300:
        puts ("300");
        break;
    case 400:
        puts ("400");
        break;
    case 500:
        puts ("500");
        break;
    case 600:
        puts ("600");
        break;
    case 700:
        puts ("700");
        break;
    default:
        puts ("default");
        break;
    }
}
```

Código ensamblador

```
    .rodata
.LC0:    .string  "100"
.LC1:    .string  "200"
.LC2:    .string  "300"
.LC3:    .string  "400"
.LC4:    .string  "500"
.LC5:    .string  "600"
.LC6:    .string  "700"
.LC7:    .string  "default"
    .text
switch_tree:
    subq $8, %rsp

    cmpl $400, %edi
    je   .L3
    cmpl $400, %edi
        # 2º cmpl quitado
        # con -O2
    jg   .L4
    cmpl $200, %edi
    je   .L5
    cmpl $300, %edi
    je   .L6
    cmpl $100, %edi
    jne  .L2
    jmp  .L7
.L4:
    cmpl $600, %edi
    je   .L8
    cmpl $700, %edi
    je   .L9
    cmpl $500, %edi
    jne  .L2
    jmp  .L10
.L7:    # 100
    movl $.LC0, %edi
    call puts
    jmp  .L1
.L5:    # 200
    movl $.LC1, %edi
    call puts
    jmp  .L1
.L6:    # 300
    movl $.LC2, %edi
    call puts
    jmp  .L1
.L3:    # 400
    movl $.LC3, %edi
    call puts
    jmp  .L1
.L10:   # 500
    movl $.LC4, %edi
    call puts
    jmp  .L1
.L8:    # 600
    movl $.LC5, %edi
    call puts
    jmp  .L1
.L9:    # 700
    movl $.LC6, %edi
    call puts
    jmp  .L1
.L2:    # 800
    movl $.LC7, %edi
    call puts
.L1:
    addq $8, %rsp
    ret
```

Sentencia switch: árbol

<table>
<tr><td colspan="10">Comparar x con 400</td></tr>
<tr><td>=</td><td colspan="9">≠</td></tr>
<tr><td rowspan="8"></td><td colspan="4"><</td><td colspan="4">></td></tr>
<tr><td colspan="4">Comparar x con 200</td><td colspan="4">.L4: Comparar x con 600</td></tr>
<tr><td>=</td><td colspan="3">≠</td><td>=</td><td colspan="3">≠</td></tr>
<tr><td rowspan="5"></td><td colspan="3">Comparar x con 300</td><td rowspan="5"></td><td colspan="3">Comparar x con 700</td></tr>
<tr><td>=</td><td colspan="2">≠</td><td>=</td><td colspan="2">≠</td></tr>
<tr><td rowspan="3"></td><td colspan="2">Comp. x con 100</td><td rowspan="3"></td><td colspan="2">Comp. x con 500</td></tr>
<tr><td>=</td><td>≠</td><td>=</td><td>≠</td></tr>
<tr><td></td><td></td><td></td><td></td></tr>
<tr><td>.L3
400</td><td>.L5
200</td><td>.L6
300</td><td>.L7
100</td><td>.L2
default</td><td>.L8
600</td><td>.L9
700</td><td>.L10
500</td><td>.L2
default</td></tr>
</table>

Sentencia switch: tabla de saltos

Código C

```
static void
switch_table (int x)
{
    switch (x)
    {
    case 0:
        puts ("0");
        /* no break */
    case 1:
        puts ("1");
        break;
    case 2:
    case 3:
        puts ("2, 3");
        break;
    case 4:
        puts ("4");
        break;
    default:
        puts (
        "x<0 || x>4");
        break;
    }
}
```

Código ensamblador

```
    .rodata
.LC0:   .string "0"
.LC1:   .string "1"
.LC2:   .string "2, 3"
.LC3:   .string "4"
.LC4:   .string "x<0 || x>4"

    .text
switch_table:
    subq    $8, %rsp

    cmpl    $4, %edi# x : 4 ?
    ja      .L2      # x > 4 || x < 0
    movl    %edi, %edi   # rdi = 0:edi
    jmp     *.L4(,%rdi,8)

    .rodata
.L4:
    .quad   .L3
    .quad   .L5
    .quad   .L6
    .quad   .L6
    .quad   .L7
```

```
    .text
.L3:    # 0
    movl    $.LC0, %edi
    call    puts
.L5:    # 1
    movl    $.LC1, %edi
    call    puts
    jmp     .L1
.L6:    # 2, 3
    movl    $.LC2, %edi
    call    puts
    jmp     .L1
.L7:    # 4
    movl    $.LC3, %edi
    call    puts
    jmp     .L1
.L2:    # x > 4 || x < 0
    movl    $.LC4, %edi
    call    puts
.L1:
    addq    $8, %rsp
    ret
```

Traducción de bucles

- Bucle do while
- Bucle while
- Bucle for

Bucle do while

Código C	Versión usando goto

`do`
`  Body`
`While (Test);`
	`loop:`
	`  Body`
	`  if (Test)`
	`  goto loop;`

Bucle do while

Código C

```
int pcount_do (unsigned x) {
   int result = 0;

   do {
      result += x & 1;
      x >>= 1;
   } while (x);

   return result;
}
```

Versión usando goto

```
int pcount_do (unsigned x) {
    int result = 0;

loop:
    result += x & 1;
    x >>= 1;
    if (x)
        goto loop;

    return result;
}
```

Código ensamblador

```
pcount_do:
   movl   $0, %eax      # result = 0
.L2:
   movl   %edi, %edx    # x
   andl   $1, %edx      # x & 1
   addl   %edx, %eax    # result += x & 1
   shrl   %edi          # x >>= 1
   jne    .L2           # if (x != 0) goto .L2
   ret
```

Bucle while

Código C	Versión usando do while	Versión usando goto

`while (Test)`
`  Body`
	`if (Test)`
	`  do`
	`  Body`
	`  while (Test);`
		`  if (!Test)`
		`  goto done;`
		`loop:`
		`  Body`
		`  if (Test)`
		`  goto loop;`
		`done:`

Bucle while

Código C

```
int pcount_while (unsigned x)
{
    int result = 0;

    while (x) {
        result += x & 1;
        x >>= 1;
    }

    return result;
}
```

Versión usando do while

```
int pcount_while (unsigned x)
{
    int result = 0;

    if (x)
        do {
            result += x & 1;
            x >>= 1;
        }
        while (x);

    return result;
}
```

Versión usando goto

```
int pcount_while (unsigned x)
{
    int result = 0;

    if (!x)
        goto done;
loop:
    result += x & 1;
    x >>= 1;
    if (x)
        goto loop;
done:
    return result;
}
```

Código ensamblador

```
pcount_while:
    movl    $0, %eax      # result = 0
    testl   %edi, %edi    # ¿x == 0?
    je      .L2
.L3:
    movl    %edi, %edx    # x
    andl    $1, %edx      # x & 1
    addl    %edx, %eax    # result += x & 1
    shrl    %edi          # x >>= 1
    jne .L3               # if (x != 0) goto .L3
.L2:
    ret
```

Bucle for

Código C	Versión usando while

`for (Init; Test; Update)`
`  Body`
	`Init;`
	`while (Test) {`
	`  Body`
	`  Update;`
	`}`

Versión usando do while	**Versión usando goto**

`Init;`
`if (Test)`
`  do {`
`  Body`
`  Update;`
`  } while (Test);`
	`  Init;`
	`  if (!Test)`
	`  goto done;`
	`loop:`
	`  Body`
	`  Update;`
	`  if (Test)`
	`  goto loop;`
	`done:`

Bucle for

Código C	Versión usando while

```c
#define WSIZE (8 * sizeof (int))

int pcount_for (unsigned x) {
    int i;
    int result = 0;
    unsigned mask;

    for (i = 0; i < WSIZE; i++) {
        mask = 1 << i;
        result += (x & mask) != 0;
    }

    return result;
}
```

```c
#define WSIZE (8 * sizeof (int))

int pcount_for (unsigned x) {
    int i;
    int result = 0;
    unsigned mask;

    i = 0;
    while (i < WSIZE) {
        mask = 1 << i;
        result += (x & mask) != 0;
        i++;
    }

    return result;
}
```

Bucle for

Versión usando do while

```
#define WSIZE (8 * sizeof (int))

int pcount_for (unsigned x) {
    int i;
    int result = 0;
    unsigned mask;

    i = 0;
    if (i < WSIZE)
        do {
            mask = 1 << i;
            result += (x & mask) != 0;
            i++;
        }
        while (i < WSIZE);

    return result;
}
```

Versión usando goto

```
#define WSIZE (8 * sizeof (int))

int pcount_for (unsigned x) {
    int i;
    int result = 0;
    unsigned mask;

    i = 0;
    if (!(i < WSIZE))
        goto done;
loop:
    mask = 1 << i;
    result += (x & mask) != 0;
    i++;
    if (i < WSIZE)
        goto loop;
done:
    return result;
}
```

Bucle for

Código ensamblador

```
pcount_for:
    movl    $0, %eax    # result = 0
    movl    $0, %ecx    # i = 0
    movl    $1, %esi    # 1
.L2:
    movl    %esi, %edx  # 1
    sall    %cl, %edx   # mask = 1 << i
    testl   %edi, %edx  # x & mask
    setne   %dl         # (x & mask) != 0
    movzbl  %dl, %edx
    addl    %edx, %eax  # result += (x & mask) != 0
    addl    $1, %ecx    # i++
    cmpl    $32, %ecx   # ¿i == 32?
    jne     .L2         # if (i == 32) goto .L2
    ret
```

Resumen

- Instrucciones de comparación
- Saltos y ajustes condicionales
  - Bits de estado
  - Sumas y restas sin signo/con signo
- Instrucciones de copia condicional
- Otras instrucciones condicionales
- Traducción de sentencias condicionales
- Traducción de bucles