PRÁCTICA 3: OPERACIONES ARITMÉTICAS Y LÓGICAS

ÍNDICE

OP_ARIT_LOGGDB.TXT:

+focus cmd mostrar pantalla de abajo
Focus set to cmd window.
+layout regs mostrar contenido de los registros
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+p $eax imprimir por pantalla en decimal el valor del registro eax
$1 = -241
+p /t $eax imprimir por pantalla en binario el valor del registro eax
$2 = 11111111111111111111111100001111
+p /t $ebx imprimir por pantalla en binario el valor del registro ebx
$3 = 11110001
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+exit salir del programa

SALTOSGDB.TXT:

+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+watch $eflags mostrar contenido anterior y actual de los eflags cada vez que se modifican
Watchpoint 2: $eflags
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF ZF IF]
New value = [IF]
main () at saltos.s:42
+layout regs mostrar contenido de los registros
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [IF]
New value = [CF PF AF SF IF]
main () at saltos.s:43
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [CF PF AF SF IF]
New value = [CF PF AF IF OF]
main () at saltos.s:53
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [CF PF AF IF OF]
New value = [AF SF IF OF]
main () at saltos.s:54
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [AF SF IF OF]
New value = [PF ZF IF]
main () at saltos.s:56
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF ZF IF]
New value = [CF AF SF IF]
main () at saltos.s:58
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [CF AF SF IF]
New value = [AF SF IF]
main () at saltos.s:59
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [AF SF IF]
New value = [PF AF SF IF OF]
main () at saltos.s:67
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+p /t $eax mostrar en binario el contenido del registro eax
$1 = 1010101111111111
+p /t $ax mostrar en binario el contenido del registro ax
$2 = 1010101111111111
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF AF SF IF OF]
New value = [IF]
main () at saltos.s:71
+focus cmd mostrar pantalla de abajo
Focus set to cmd window.
+p /t $eax imprimir en pantalla el contenido del registro eax en binario
$3 = 1010101111111111
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [IF]
New value = [PF SF IF]
main () at saltos.s:72
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF SF IF]
New value = [PF ZF IF]
main () at saltos.s:78
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF ZF IF]
New value = [PF IF]
main () at saltos.s:82
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF IF]
New value = [CF AF SF IF]
main () at saltos.s:84
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [CF AF SF IF]
New value = [PF ZF IF]
main () at saltos.s:88
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF ZF IF]
New value = [PF IF]
main () at saltos.s:92
+n avanzar a la siguiente instrucción
salto4 () at saltos.s:94
+n

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF IF]
New value = [CF AF SF IF]
salto4 () at saltos.s:96
+n avanzar a la siguiente instrucción
salto5 () at saltos.s:98
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [CF AF SF IF]
New value = [PF AF IF OF]
salto5 () at saltos.s:104
+n avanzar a la siguiente instrucción
salto6 () at saltos.s:106
+n avanzar a la siguiente instrucción

Watchpoint 2: $eflags una vez que se ha introducido watch $eflags, cada vez que se modifiquen se mostrará por pantalla su anterior y actual valor

Old value = [PF AF IF OF]
New value = [PF ZF IF]
salto6 () at saltos.s:107
+n avanzar a la siguiente instrucción
salto7 () at saltos.s:113
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
+n avanzar a la siguiente instrucción
[Inferior 1 (process 6666) exited normally]
+exit salir del programa

MÓDULO OP_ARIT_LOG.S:

• Indicar cómo asociar el valor de los sumandos a las macros OPE1 y OPE2.

Para asociar el valor de los sumandos a las macros, se necesita declarar las macros en l a zona superior del archivo op_arit_log.s. Además, como se puede observar en la imagen, los valores de los sumandos son 5 y 10. Por lo tanto, el código sería de la siguiente forma:

```
## add: suma
mov $5,%eax
mov $10,%ebx
add %ebx,%eax
```

```
## MACROS
.equ SYS_EXIT,  1
.equ SUCCESS,   0
.equ N,     5
.equ OPE1,      5
.equ OPE2,      10
```

```
## add: suma
mov $OPE1,%eax
mov $OPE2,%ebx
add %ebx,%eax
```

• Sin cambiar el valor de los operandos:

Indicar el valor de la resta en la instrucción 1:

```
## sub: resta
mov $5,%eax
mov $10,%ebx
sub %ebx,%eax
```

Como se puede observar en la imagen, el resultado de la resta (5 – 10 = -5) quedará guardado en el registro destino, que en este caso es el registro eax. Por lo tanto, en el gdb la instrucción será la siguiente:

```
(gdb) p $eax
$2 = 5
(gdb) n
(gdb) p $eax
$3 = -5
(gdb)
```

Tras cargar el valor de 5 en el registro eax, se realiza la instrucción sub correspondiendo a la resta.

Indicar el valor de la multiplicación en la instrucción 2:

```
## imul: multiplicación entera "con signo": AX<- BL*AL
movb $-3,%bl
movb $5,%al
imulb %bl
```

En la instrucción imul, el operando destino es el registro al. Anteriormente, es necesario cargar en los registros al y en este caso bl, los valores 5 y -3, respectivamente. El resultado de la operación será -3*5 = -15.

```
(gdb) p $al
$5 = 5
(gdb) n
(gdb) p $al
$6 = -15
(gdb)
```

Indicar el valor de la división en la instrucción 3:

```
## idiv: división "con signo" .   (AL=Cociente, AH=Resto) <- AX/(byte en registro o memoria)
movw $5,%ax             #dividendo
movb $3,%bl             #divisor
idivb %bl           # 5/3 = 1*3 + 2
```

```
(gdb) p $al
$10 = 5
(gdb) p $ax
$11 = 5
(gdb) n
(gdb) n
(gdb) p $al
$12 = 1
(gdb) p $ah
$13 = 2
(gdb)
```

En la instrucción idiv (división entera), el operando destino es el registro al. Anteriormente, es necesario cargar en los registros al (en este caso utiliza el registro ax que contiene 2 bytes de tamaño, uno de ellos siendo el registro a su vez al (1 byte) y en este caso bl, los valores 5 y 3, respectivamente. El resultado de la operación será 5/3 = 1, que estará guardado en el registro al. El resto de la división entera se guardará en el registro ah (5/3 = 1*3 + 2 (resto)).

Indicar el valor de la división en la instrucción 4:

```
movw $(N+1),%ax
imulw %bx               #imulw Op ; Op=word ; DX:AX<- AX*Op
movw $2,%bx
## El resultado queda en AX y el resto DX=0
idivw %bx               #idivw Op ; Op=word ; AX<-(DX:AX)/Op ; DX:=Resto
```

Como se observa, se realiza la división entera con el tamaño de los operandos word (2 bytes). Carga el valor de N + 1 = 5 + 1 = 6 en el registro ax de 2 bytes (dividendo) y el valor de 2 en el registro bx de dos bytes. Aunque, posteriormente realiza la multiplicación entera con el registro bx cargado anteriormente con el valor de N = 5 (ax = ax*bx = 6 *5 = 30). El nuevo dividendo tomará el valor de 30 por lo tanto. El destino es el registro ax (30/2 = 15) y el resto se guardará en dx (30/2 = 15 + 0 (resto)).

```
(gdb) p $ax
$15 = 30
(gdb) n
(gdb) p $ax
$16 = 15
(gdb) p $dx
$17 = 0
(gdb)
```

Indicar el valor de las operaciones lógicas en la instrucción 5:

```
## OPERACIONES LOGICAS

mov $0xFFFF1F, %eax
mov $0x0000F1, %ebx
not %eax        # inversión
and %ebx,%eax   # producto lógico
or  %ebx,%eax   # suma lógica
```

En primer lugar, se carga en memoria los valores en hexadecimal de 0xFFFF1F y 0x0000F1 en los registros eax y ebx, respectivamente.

```
(gdb) p /s $eax
$20 = 16776991
(gdb) p /x $eax
$21 = 0xffff1f
(gdb) n
(gdb) p /x $ebx
$22 = 0xf1
(gdb) n
(gdb) p $eax
$23 = -16776992
(gdb) p /x $eax
$24 = 0xff0000e0
(gdb)
```

En segundo lugar, se realiza la instrucción not (negación), que en este caso, cambia de signo el valor del registro, como se puede observar en la imagen. La primera vez que se muestra el valor de eax, le siguen 2 bytes a la izquierda que corresponden a dos ceros (positivo). Sin embargo, tras realizar la operación de negación, el valor de eax pasa a ser negativo ya que empieza por f.

Operador	Algebra	C
NOT	¬ ~	~
OR	∨	|
AND	∧	&

x	y	z=x∨y	z=x∧y
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

A continuación, se encuentran las tablas de las operaciones lógicas, para entender el resultado de las operaciones not, or y and.

```
(gdb) p /t $eax
$30 = 11111111000000000000000011100000
(gdb) p /t $ebx
$31 = 11110001
(gdb) n
(gdb) p /t $eax
$32 = 11100000
(gdb)
```

En primer lugar, se muestran por pantalla los valores de los registros eax y ebx en binario antes de realizar la operación and. El operando destino es eax.
Cabe destacar que cuando se muestra el tamaño de ebx, falta de añadir los bits con el valor de 0 representando el signo del número hasta completar los 32 bits del registro (4 bytes). En este caso el nuevo valor de eax en binario tomará por bit el valor de 0 siempre y cuando en dicha posición no tenga un 1 tanto eax como ebx.

Sin embargo, como se muestra abajo, en el caso de la operación ocurre todo lo contrario, mientras el bit en dicha posición sea distinto de 0 tanto en eax como en ebx, se tomará el valor de 1. En caso contrario, el valor en dicha posición será cero.

```
(gdb) p /t $eax
$32 = 11100000
(gdb) p /t $ebx
$33 = 11110001
(gdb) n
(gdb) p /t $eax
$34 = 11110001
(gdb)
```

Continuamos comentando el resto de operaciones lógicas:

```
## Complemento a 2 mediante operación lógica not()+1
mov %ebx,%eax
not %eax
inc %eax
```

En primer lugar, cargamos en eax el contenido de ebx. Realizamos la operación de la negación en eax explicada anteriormente. Y para finalizar, incrementamos (inc) en 1 el valor de eax como se puede observar en el último bit en binario.

```
(gdb) p /t $eax
$37 = 11110001
(gdb) n
(gdb) p /t $eax
$38 = 11111111111111111111111100001110
(gdb) n
(gdb) p /t $eax
$39 = 11111111111111111111111100001111
(gdb)
```

```
## Desplazamiento de bits
shr $4,%eax          #desplazamiento lógico: bits a introduccir -> 0..  (tira por la derecha y mete 0 por la izda) igual haciendo print no se ve los 0 de la izda por convención
sar $4,%eax          #desplazamiento aritmético: bits a introducir -> extensión del signo (tira por la derecha y mete bit de signo por la izda)
```

Como se observa en los comentarios, tras la primera operación se introducen ceros por la izquierda, desplazando los últimos 4 bits en este caso de la derecha. Se introducen ceros al ser un desplazamiento lógico.

```
(gdb) p /t $eax
$39 = 11111111111111111111111100001111
(gdb) n
(gdb) p /t $eax
$40 = 11111111111111111111111110000
```

En el siguiente caso, se introducen 1 por la izquierda debido al bit de signo (número negativo), aunque si hubiera sido un número positivo, el resultado del desplazamiento lógico y aritmético habría sido el mismo.

```
(gdb) p /t $eax
$40 = 11111111111111111111111110000
(gdb) n
(gdb) p /t $eax
$41 = 11111111111111111111111
(gdb)
```

MÓDULO SALTOS.S:

• Registro de Flags

Editar, compilar y ejecutar el siguiente bloque de instrucciones para indicar el contenido registro EAX y el estado de los flags CF, ZF, SF, PF, OF después de la ejecución de cada instrucción:

```
mov  $0xFFFFFFFF,%eax
shr  $1,%eax
add  %eax,%eax
testb $0xFF,%eax
    cmpl $0xFFFFFFFF,%eax
```

En primer lugar, se modifica el tamaño de test, ya que se esta comparando con 1 byte cuando el registro eax es de 4 bytes. El programa quedaría de la siguiente forma:

```
##Programa: flags.s

## MACROS
        .equ SYS_EXIT, 1
        .equ SUCCESS, 0
## VARIABLES LOCALES
        .data
## INSTRUCCIONES
        .global main
        .text
main:
## RESET
        xor %eax,%eax
        xor %ebx,%ebx

        mov $0xFFFFFFFF, %eax
        shr $1, %eax
        add %eax, %eax
        test $0xFF, %al
        cmp $0xFFFFFFFF, %eax  ##cmp = cmpl
## SALIDA
        mov $SYS_EXIT, %eax
        mov $SUCCESS, %ebx
        int $0x80
        .end
```

mov $0xFFFFFFFF, %eax: (Al no realizar ninguna operación, los flags no se modifican)

```
(gdb) p /x $eax
$1 = 0x0
(gdb) n
(gdb) p /x $eax
$2 = 0xffffffff
(gdb)
```

Se muestra por pantalla el registro en hexadecimal (/x). Como al comenzar el programa hemos hecho un reset de los registros, el primer valor será 0x0, aunque al ejecutar la instrucción mov, se observa como se cargá el valor 0xFFFFFFFF.

shr $1, %eax:

```
(gdb) p /x $eax
$2 = 0xffffffff
(gdb) p /t $eax
$3 = 11111111111111111111111111111111
(gdb) n
(gdb) p /t $eax
$4 = 11111111111111111111111111111111
(gdb) p /x $eax
$5 = 0x7fffffff
(gdb)
```

Antes de ejecutar la instrucción, se muestra el valor del registro eax tanto en binario como en hexadecimal. A partir de la respectiva instrucción, se realiza un desplazamiento lógico de un único bit a la derecha, introduciendo el bit 0 por la izquierda. Cuando se muestra el contenido en binario, solamente se cuentan 31 dígitos (anteriormente se muestran 32, todo unos) debido a que el bit 0 de la izquierda no se enseña. Además al pasar de binario a hexadecimal, los 4 bits 0111 son igual a un 7 en hexadecimal, como además se observa.

```
Watchpoint 2: $eflags

Old value = [ PF ZF IF ]
New value = [ CF PF AF IF OF ]
main () at flags.s:18
(gdb)
```

Respecto a los flags, al reiniciar los registros a 0 con la operación xor y un mismo registro tanto como destino y fuente, se activan el parity flag (el cero se considera par) y el zero flag. A través del comando watch $eflags, cada vez que varíen éstos, se mostrará por pantalla tanto su anterior valor como el actual.
Al realizar el desplazamiento lógico, se desactiva el zero flag ya que el contenido registro es distinto de 0. Sigue activado el parity flag debido a que el Least Significant Byte (LSB) 0xFF contiene un número par de unos. El carry flag se activa debido al último bit salido. Además se muestra overflow.

add %eax, %eax:

```
Watchpoint 2: $eflags

Old value = [ CF PF AF IF OF ]
New value = [ AF SF IF OF ]
main () at flags.s:19
(gdb)
```

En este caso se da overflow porque una suma de dos números positivos, no puede dar un número negativo. Ésto sucede debido a que se suman dos operandos de 32 bits.

```
(gdb) p /x $eax
$2 = 0xfffffffe
(gdb)
```

test $0xFF, %eax:

```
(gdb) p /t $al
$1 = 11111110
(gdb) n

Watchpoint 2: $eflags

Old value = [ AF SF IF OF ]
New value = [ SF IF ]
main () at flags.s:20
(gdb) p /t $al
$2 = 11111110
(gdb)
```

La instrucción test equivale a una operación lógica AND en la que no cambia el valor del registro “destino”, y únicamente se modifican los eflags. Por lo tanto, el contenido de al no varía. Como se ha explicado anteriormente el funcionamiento de dicha operación, solamente se muestra el resultado:

Operación: 0xFF = 11111111 AND 11111110 = 11111110

Por lo tanto, no se activa el parity flag al haber un número impar de unos. Únicamente se mantiene el SF debido a que es un número negativo (1…).

cmp $0xFFFFFFFF, %eax:

```
(gdb) p /x $eax
$3 = 0xfffffffe
(gdb) n

Watchpoint 2: $eflags

Old value = [ SF IF ]
New value = [ CF PF AF SF IF ]
main () at flags.s:22
(gdb) p /x $eax
$4 = 0xfffffffe
(gdb)
```

La operación cmp equivale a sub (resta) en la que no cambia el contenido del destino y sólo se modifican los eflags.
Operación: 0xFFFFFFFE – 0xFFFFFFFF = 0xFFFFFFFF
Se activa el carry flag por la llevada final. Además hay un número par de unos, por lo tanto, se activa el parity flag.
Finalmente, al ser un número negativo implica que SF = 1.

- Saltos

Editar, compilar y ejecutar el siguiente bloque de instrucciones para indicar el estado de los flags CF, ZF, SF, PF, OF antes de la ejecución de la instrucción de salto e indicar si se produce o no el salto.

```
     mov $0x00AA, %ax
     mov $0xFF00, %bx
     cmp %bx,%ax
     ja  salto1
     jg  salto2
salto1: mov $0xFF,%ebx
salto2: mov $1,%eax
     int $0x80
```

```
## MACROS
        .equ SYS_EXIT, 1
        .equ SUCCESS, 0
## VARIABLES LOCALES
        .data
## INSTRUCCIONES
        .global main
        .text
main:
## RESET
        xor %eax,%eax
        xor %ebx,%ebx

        mov $0x00AA, %ax
        mov $0xFF00, %bx
        cmp %bx, %ax
        ja salto1
        jg salto2
salto1:
        mov $0xFF, %ebx
salto2:
        mov $1, %eax
## SALIDA
        int $0x80
        .end
```

mov $0x00AA, %ax:

```
(gdb) p /x $eax
$1 = 0x0
(gdb) n
(gdb)  p /x $eax
$2 = 0xaa
(gdb)
```

Tras hacer reset del registro con la operación xor, el contenido del registro es 0. A continuación, se carga dicho valor en el registro ax. Por lo tanto, el valor de los eflags no cambia.

mov $0xFF00, %bx:

```
(gdb) p /x $bx
$3 = 0x0
(gdb) n
(gdb) p /x $bx
$4 = 0xff00
```

Tras hacer reset del registro con la operación xor, el contenido del registro es 0. A continuación, se carga dicho valor en el registro bx. Por lo tanto, el valor de los eflags no cambia.

cmp %bx, %ax:

```
(gdb) p /x $ax
$5 = 0xaa
(gdb) p /x $bx
$6 = 0xff00
(gdb) n

Watchpoint 2: $eflags

Old value = [ PF ZF IF ]
New value = [ CF PF IF ]
main () at saltos2.s:17
(gdb)
```

Como se ha explicado anteriormente:
OPERACIÓN: AX – BX = 0x00AA – 0xFF00 = 0X01AA (número positivo)

Por lo tanto, se activa el carry flag por la llevada final. El resultado es distinto de cero asi que se desactiva el zero flag. Se mantiene el parity flag debido a que el LSB = 0xAA = 11001100 tiene un número par de unos (4).

ja salto1:

No se cumple la condición de saltar si es superior, y por lo tanto no se accede a la etiqueta salto1.

```
13
14          mov $0x00AA, %ax
15          mov $0xFF00, %bx
16          cmp %bx, %ax
17          ja salto1
18          jg salto2
19  salto1:
20          mov $0xFF, %ebx
21  salto2:
22          mov $1, %eax
23  ## SALIDA
24          int $0x80
```

jg salto2:

Como se cumple la condición de que el contenido de ax sea mayor que el de bl, se salta a la etiqueta salto 2 para realizar la instrucción mov $1,%eax.

```
13
14          mov $0x00AA, %ax
15          mov $0xFF00, %bx
16          cmp %bx, %ax
17          ja salto1
18          jg salto2
19  salto1:
20          mov $0xFF, %ebx
21  salto2:
22          mov $1, %eax
23  ## SALIDA
24          int $0x80
```

int $0x80:

Finalmente se llama al sistema operativo para que tome el control del programa y finalice.

COMPILACIÓN:

```
gcc -m32 -g -o op_arit_log op_arit_log.s

gcc -m32 -g -o saltos saltos.s
```

Cabe destacar el uso de -m32 para usar una máquina de 32 bits, y -g para cargar la tabla de símbolos. Además no se ha añadio el -nostartfiles porque en este caso se utiliza main.

CONCLUSIÓN:

Durante ésta práctica, se ha hecho uso de distintas operaciones tanto aritméticas como lógicas para ir observando como varían los valores de los registros dependiendo del tamaño de ellos. Además, se ha querido incidir en el uso de los saltos a través de etiquetas con secuencias alternativas y en el uso de los eflags para conocer como varían dependiendo de la operación y el dato que se ejecute.