Fundamentos de Ordenadores

Colección de problemas adicionales

Tema 6. Gestión de subrutinas

Manel Guerrero

Problema 1

Para invocar a la subrutina printf() haremos exactamente lo mismo que si invocáramos a una subrutina escrita por nosotros:

```
8<-[t7_ex_printf.s]-------------
.data
s: .asciz "Hello World!\n"
.text
.globl main
main:
 pushl $s
    # Passar param (punter a s)
 call printf # Cridar subrutina
 addl $4, %esp
    # Eliminar param de la pila

 movl $1, %eax
 movl $0, %ebx
 int $0x80
------------------------------>8
$gcc -o t7_ex_printf t7_ex_printf.s
$ ./t7_ex_printf
Hello World!
$
```

Es importante que utilicemos “.asciz” en lugar de “.ascii” (que equivale a usar “.ascii” pero acabando la cadena de caracteres con el carácter cero ('\0')).

Este problema enseña a invocar una subrutina que ya está implementada sin tener que saber como implementar subrutinas. Además, enseña que si un parámetro es un vector en realidad se pasa la posición de memoria donde ese vector empieza.

a) Escribe un programa que dado una secuencia de enteros acabada en cero (por ejemplo “21, 15, 34, 11, 6, 50, 32, 80, 10, 0”) imprima por pantalla los números de la secuencia.

```
$ ./t7_printf
El numero es: 21
El numero es: 15
El numero es: 34
El numero es: 11
```

```
El numero es: 6
El numero es: 50
El numero es: 32
El numero es: 80
El numero es: 10
$
```

b) Haz otra versión del programa donde lo que escriba por pantalla sea:

```
$ ./t7_printf2
i[0] = 21
i[1] = 15
i[2] = 34
i[3] = 11
i[4] = 6
i[5] = 50
i[6] = 32
i[7] = 80
i[8] = 10
$
```

Solución:

```
8<-[t7_printf.s]---------------
.data
i: .int 21, 15, 34, 11, 6, 50, 32, 80, 10, 0 # Acabat en 0
s: .asciz "El numero es: %d\n" # .asciz es com .ascii pero acabat
en '\0'
.text
.globl main
main:
 movl $0, %ecx
loop:
 pushl %ecx         # 1 Salvar eax,ecx,edx
 pushl i(,%ecx,4)   # 2 Pasar params
 pushl $s      # 2 Pasar params
 call printf        # 3 Llamar subrutina
               # 12 Resultado en eax
 addl $8, %esp      # 13 Eliminar params
 popl %ecx          # 14 Restaurar eax,ecx,edx
 incl %ecx
 cmpl $0, i(,%ecx,4)
 jne loop

 movl $1, %eax
 movl $0, %ebx
 int $0x80
------------------------------>8

8<-[t7_printf2.s]--------------
```

```
.data
i: .int 21, 15, 34, 11, 6, 50, 32, 80, 10, 0 # Acabat en 0
s: .asciz "i[%d] = %d\n" # .asciz es com .ascii pero acabat en
'\0'
.text
.globl main
main:
 movl $0, %ecx
loop:
 pushl %ecx         # 1 Salvar eax,ecx,edx
 pushl i(,%ecx,4)   # 2 Pasar params (comencant per l'ultim)
 pushl %ecx         # 2 Pasar params
 pushl $s      # 2 Pasar params
 call printf        # 3 Llamar subrutina
               # 12 Resultado en eax
 addl $12, %esp          # 13 Eliminar params
 popl %ecx          # 14 Restaurar eax,ecx,edx
 incl %ecx
 cmpl $0, i(,%ecx,4)
 jne loop

 movl $1, %eax
 movl $0, %ebx
 int $0x80
------------------------------>8
```

Problema 2

Podemos invocar scanf() de la misma manera que invocamos printf(). Pero recuerda que en scanf se pasan los punteros a las variables a leer (ejemplo: “scanf(“%d”, &num);”). En ensamblador tendréis que pasar la posición de memoria de las variables a leer.

Este problema sirve para repasar como se pasa por parámetro punteros.

Escribe un programa que pida al usuario que introduzca un número y muestre por pantalla el número leído y que ha devuelto la función scanf().

```
$ ./t7_scanf
Introduex un núm
123
El numero introduit és: 123. Scanf ha retornat: 1
$ ./t7_scanf
Introduex un núm
No vull
El numero introduit és: 0. Scanf ha retornat: 0
$
```

Solución:

```
8<-[t7_scanf.s]-----------------
```

```
.data
s: .asciz "Introduex un núm\n"
is: .asciz "%d"
os: .asciz "El numero introduit és: %d. Scanf ha retornat: %d\n"
.bss
.comm i,4,4
.comm r,4,4
.text
.globl main
main:
 pushl $s # Passar param #1 (punter a s)
 call printf # Cridar subrutina
 addl $4, %esp # Eliminar params de pila
#printf("Introduex un núm\n");

 pushl $i        # 2 Passar param #1 (punter a i)
 pushl $is   # 2 Passar param #2 (punter a is)
 call scanf  # Cridar subrutina
 movl %eax,r # El retorn de scanf està a eax
 addl $8, %esp # Eliminar params de la pila
#scanf("%d");

 pushl r     # Passar params
 pushl i     # Passar params
 pushl $os   # Passar params
 call printf # Cridar subrutina
 addl $8, %esp # Eliminar params de la pila
#printf("El numero introduit és: %d. Scanf ha retornat: %d\n", i,
r);

 movl $1, %eax
 movl $0, %ebx
 int $0x80
------------------------------>8
```

Problema 3

Escribir un programa que dada una secuencia de caracteres acabada en '\0' ('\0' es el carácter cuyo valor numérico es cero, y se suele usar para indicar el final de una secuencia de caracteres), la imprima por pantalla y la copie con todos los caracteres en minúsculas a un vector y imprima este vector por pantalla. Ignorad los caracteres con acentos.

Este problema sirve para aprender cómo implementar una subrutina simple.

Si la secuencia es "Demasiadas Mayusculas PARA TAN pOcAs BalAs.\n\0" el programa debería retornar:

```
$ ./t7_minuscules_print_2
Demasiadas Mayusculas PARA TAN pOcAs BalAs.
demasiadas mayusculas para tan pocas balas.
$
```

Solución:

```
8<-[t7_minuscules_print_2.s]----
MIDA = 100
.data
is: .asciz "Demasiadas Mayusculas PARA TAN pOcAs BalAs.\n"
.bss
.comm os,MIDA,1
.text
.global main
main:
 pushl $is
 call printf
 addl $4, %esp

 movl $0, %ecx
for:
 # is(%ecx)==$0
 cmpb $0, is(%ecx)
 je endfor
 pushl %ecx
 movsbl is(%ecx), %edx
 pushl %edx
 call minuscula
 addl $4, %esp
 popl %ecx
 movb %al, os(%ecx)
 incl %ecx
 jmp for
endfor:
 movb $0, os(%ecx)

 pushl $os
 call printf
 addl $4, %esp

 movl $0, %ebx
 movl $1, %eax
 int $0x80

minuscula:
 pushl %ebp
 movl %esp, %ebp

 movb 8(%ebp), %al
 # %al < $'A'
 cmpb $'A', %al
 jl minendif
 # %al > $'Z'
 cmpb $'Z', %al
 jg minendif
```

```
 addb $'a'-'A', %al

minendif:
 popl %ebp
 ret
------------------------------>8
```

Problema 4

El valor que contiene el registro EBX en el momento de llamar a la interrupción 0x80 es el valor que el programa devuelve al sistema operativo (de hecho la parte contenida en el registro BL). Y se puede consultar, en Linux, si en la misma consola donde hemos ejecutado el programa, escribimos "echo $?".

Dada la siguiente función en C (que implementa la multiplicación a base de sumas):

```
8<------------------------------
int multiplica(int a, int b) {
     int m=0;
     while(b!=0) {
          m=m+a;
          b--;
     }
     return m;
}
------------------------------>8
```

Escribe un programa en ensamblador que calcule 2*3 + 5*2 utilizando una subrutina que haga lo mismo que la función en C "multiplica()". Aunque no sería necesario, haz que en la versión en ensamblador la variable "m" sea una variable local en lugar de utilizar un registro. Y haz que el programa mueva el resultado de 2*3 + 5*2 a EBX antes de finalizar el programa. Si la variable 'r' contiene 2*3 + 5*2, escribiríamos:

```
8<------------------------------
     movl r, %ebx
     movl $1, %eax            #exit (%ebx is returned)
     int $0x80
------------------------------>8
```

Este problema enseña un primer ejemplo de subrutina con una variable local.

Ejemplo de ejecución:

```
$ ./multiplica; echo $?
16
$
```

Solución:

```
8<-[multiplica.s]--------------
```

```
# Inspirado en power.s del libro "Programming from the ground up"
# Pero power.s era chapucero por que en 'imull' op2 tiene que ser
# registro y todo el rato estaba moviendo de reg a local var.
#
# 2*3+5*2=16
.text
.global main
main:
 pushl $3
 pushl $2
 call multiplica
 addl $8, %esp
 movl %eax, %ebx
 pushl $2
 pushl $5
 call multiplica
 addl $8, %esp
 addl %eax, %ebx
 movl $1, %eax
 int $0x80

multiplica:
 pushl %ebp
 movl %esp, %ebp
 subl $4, %esp
 movl 8(%ebp), %eax
 movl 12(%ebp), %ecx
 movl $0, -4(%ebp)
multiplica_loop:
 #ecx==0
 cmpl $0, %ecx
 je multiplica_endloop
 addl %eax, -4(%ebp)
 decl %ecx
 jmp multiplica_loop
multiplica_endloop:
 movl -4(%ebp), %eax
 movl %ebp, %esp
 popl %ebp
 ret
------------------------------>8
```

Problema 5

Escribe un programa que, dado un vector de enteros acabado en cero, recorra el vector y por cada elemento del vector escriba por pantalla el valor y si es un valor positivo o negativo.

Este problema sirve para seguir practicando subrutinas y como precalentamiento para el siguiente problema.

Ejemplo de ejecución con el vector -7, 7, -1, 32, 45, 0:

```
$ ./t7_signe
El valor -7 és negatiu
El valor 7 és positiu
El valor -1 és negatiu
El valor 32 és positiu
El valor 45 és positiu
$
```

Solución:

```
8<-[t7_signe.s]-----------------
MIDA = 100
.data
v: .int -7, 7, -1, 32, 45, 0 # Acabat en 0
sp: .asciz "El valor %d és positiu\n"
sn: .asciz "El valor %d és negatiu\n"
.text
.global main
main:
 movl $0, %ecx
for:
 cmpl $0, v(,%ecx,4)
 je endfor
 pushl %ecx
 pushl v(,%ecx,4)
 call signe
 cmp $1, %eax
 jne else
 pushl v(,%ecx,4)
 pushl $sp
 call printf
 addl $8, %esp
 jmp endif
else:
 pushl v(,%ecx,4)
 pushl $sn
 call printf
 addl $8, %esp
endif:
 addl $4, %esp
 popl %ecx
 incl %ecx
 jmp for
endfor:

 movl $0, %ebx
 movl $1, %eax
 int $0x80

signe:
```

```
 pushl %ebp
 movl %esp, %ebp
 movl 8(%ebp), %eax
 cmpl $0, %eax
 jge signe_else
 movl $0, %eax
 jmp signe_endif
signe_else:
 movl $1, %eax
signe_endif:
 popl %ebp
 ret
------------------------------>8
```

Problema 6

Escribe un programa que, dado varios vectores de enteros llame para cada uno de ellos una función que como primer parámetro tenga el número de elementos del vector y como segundo un puntero al vector, y que dicha función recorra el vector y escriba por pantalla cuantos ceros, números positivos y números negativos hay en el vector.

Este problema enseña cómo trabajar con subrutinas con varias variables locales.

Ejemplo de 3 vectores junto con las variables enteras que indican la longitud de los vectores:

```
n1: .long 8
v1: .long 7,9,2,3,-4,-5,-6,0
n2: .long 6
v2: .long 1,2,3,-4,-5,-6
n3: .long 1
v3: .long 1
```

Ejemplo de ejecución con dichos vectores:

```
$ ./t7_signes
7, 9, 2, 3, -4, -5, -6, 0,
Zeros:1. Positius:4. Negatius:3.
1, 2, 3, -4, -5, -6,
Zeros:0. Positius:3. Negatius:3.
1,
Zeros:0. Positius:1. Negatius:0.
$
```

Solución:

```
8<-[t7_signes.s]---------------
.data
s: .asciz "\nZeros:%d. Positius:%d. Negatius:%d.\n"
sd: .asciz "%d, "
n1: .long 8
```

```
v1: .long 7,9,2,3,-4,-5,-6,0
n2: .long 6
v2: .long 1,2,3,-4,-5,-6
n3: .long 1
v3: .long 1
.text
.global main
main:
 push $v1
 push n1
 call signes
 add $8, %esp

 push $v2
 push n2
 call signes
 add $8, %esp

 push $v3
 push n3
 call signes
 add $8, %esp

 movl $0, %ebx
 movl $1, %eax
 int $0x80

signes:
 pushl %ebp
 movl %esp, %ebp
 subl $12, %esp

 movl $0, -4(%ebp)
 movl $0, -8(%ebp)
 movl $0, -12(%ebp)
 movl $0, %ecx
 movl 12(%ebp), %ebx
signes_for:
 cmpl 8(%ebp), %ecx
 jge signes_endfor # %ecx >= 8(%ebp)

 pushl %ebx
 pushl %ecx
 pushl (%ebx,%ecx,4)
 pushl $sd
 call printf
 addl $8, %esp
 popl %ecx
 popl %ebx

 cmpl $0, (%ebx,%ecx,4)
# je signes_zero # == 0
```

```
 jg signes_posi # > 0
 jl signes_nega # < 0
signes_zero:
 incl -4(%ebp)
 jmp signes_forinc
signes_posi:
 incl -8(%ebp)
 jmp signes_forinc
signes_nega:
 incl -12(%ebp)
# jmp signes_forinc
signes_forinc:
 incl %ecx
 jmp signes_for
signes_endfor:

 pushl %ebx
 pushl %ecx
 pushl -12(%ebp)
 pushl -8(%ebp)
 pushl -4(%ebp)
 pushl $s
 call printf
 addl $16, %esp
 popl %ecx
 popl %ebx

 addl $12, %esp
 popl %ebp
 ret
------------------------------>8
```

Problema 7

Escribe un programa que, dado varios vectores de caracteres (declarados como “.asciz”) llame para cada uno de ellos una función que recorra el vector y escriba por pantalla cuantas 'a's , 'b's, 'c's, ... , y 'z's hay en el vector.

Este problema enseña cómo acceder a posiciones de vectores que son pasados como parámetro a una función o que son una variable local de una función y que, por tanto, deben ser accedidos mediante direccionamientos de memoria con registro base en lugar de con desplazamiento. Es decir, utilizando expresiones del tipo “OFFSET(%ebp,%ecx,4)” o “(%ebx,%ecx,4)” en lugar de “v(,%ecx,4)”.

Ejemplo de vectores de caracteres:

```
s1: .asciz "hola, muchas letras veo yo!"
s2: .asciz "murcielago"
s3: .asciz "9aaaaaaaaas"
```

Ejemplo de ejecución con dichos vectores:

```
$ ./t7_cuento_minusculas
Cuento las apariciones de letras minúsculas.

hola, muchas letras veo yo!
a b c d e f g h i j k l m n o p q r s t u v w x y z
3 0 1 0 2 0 0 2 0 0 0 2 1 0 3 0 0 1 2 1 1 1 0 0 1 0

murcielago
a b c d e f g h i j k l m n o p q r s t u v w x y z
1 0 1 0 1 0 1 0 1 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0

9aaaaaaaaas
a b c d e f g h i j k l m n o p q r s t u v w x y z
9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0

$
```

Solución 1 (con “(%ebx,%ecx,4)”):

```
8<-[t7_cuento_minusculas.s]-----
NUM_LETRAS = 26
SIZE_LETRAS = +4*NUM_LETRAS
OFFSET_LETRAS = -4*NUM_LETRAS
.data
s_intro: .asciz "Cuento las apariciones de letras minúsculas.\n\n"
s1: .asciz "hola, muchas letras veo yo!"
s2: .asciz "murcielago"
s3: .asciz "9aaaaaaaaas"
s: .asciz "%d "
sc: .asciz "%c"
sr: .asciz "\n\n"
s_letras: .asciz "\na b c d e f g h i j k l m n o p q r s t u v w
x y z\n"
.text
.global main
main:
 pushl $s_intro
 call printf
 addl $4, %esp

 pushl $s1
 call letras
 addl $4, %esp

 pushl $s2
 call letras
 addl $4, %esp

 pushl $s3
 call letras
```

```
 addl $4, %esp

 movl $0, %ebx
 movl $1, %eax
 int $0x80

letras:
 pushl %ebp
 movl %esp, %ebp
 subl $SIZE_LETRAS, %esp
 pushl %esi

 movl $0, %ecx
 movl %ebp, %ebx
 addl $OFFSET_LETRAS, %ebx
letras_for0:
 cmpl $NUM_LETRAS, %ecx
 jge letras_endfor0 # %ecx>=$NUM_LETRAS
 movl $0, (%ebx,%ecx,4)
 incl %ecx
 jmp letras_for0
letras_endfor0:

 movl 8(%ebp), %esi
 movl $0, %ecx
letras_for1:
 cmpb $0, (%esi,%ecx)
 je letras_endfor1 # $0==(%esi,%ecx)

 pushl %ecx
 pushl (%esi,%ecx)
 pushl $sc
 call printf
 addl $8, %esp
 popl %ecx

 # passar de 'a' a 0, de 'b' a 1, ..., 'z' a 25
 movl $0, %eax
 movb (%esi,%ecx), %al
 # begin controlar que passa si (%al<'a' || %al>'z')
 cmpb $'a', %al
 jl letras_incfor1 # %al<'a'
 cmpb $'z', %al
 jg letras_incfor1 # %al>'z'
 # end controlar
 subb $'a', %al

 incl (%ebx,%eax,4)

letras_incfor1:
 incl %ecx
 jmp letras_for1
```

```
letras_endfor1:

 pushl %ecx
 pushl $s_letras
 call printf
 addl $4, %esp
 popl %ecx

 movl $0, %ecx
letras_for2:
 cmpl $NUM_LETRAS, %ecx
 jge letras_endfor2 # %ecx>=$NUM_LETRAS

 pushl %ecx
 pushl (%ebx,%ecx,4)
 pushl $s
 call printf
 addl $8, %esp
 popl %ecx

 incl %ecx
 jmp letras_for2
letras_endfor2:

 pushl %ecx
 pushl $sr
 call printf
 addl $4, %esp
 popl %ecx

 popl %esi
 movl %ebp, %esp
 popl %ebp
 ret
------------------------------>8
```

Solución alternativa (con “OFFSET_LETRAS(%ebp,%ecx,4)”):

```
8<-[t7_cuento_minusculas_b.s]---
NUM_LETRAS = 26
SIZE_LETRAS = +4*NUM_LETRAS
OFFSET_LETRAS = -4*NUM_LETRAS
.data
s_intro: .asciz "Cuento las apariciones de letras minúsculas.\n\n"
s1: .asciz "hola, muchas letras veo yo!"
s2: .asciz "murcielago"
s3: .asciz "9aaaaaaaaas"
s: .asciz "%d "
sc: .asciz "%c"
sr: .asciz "\n\n"
```

```
s_letras: .asciz "\na b c d e f g h i j k l m n o p q r s t u v w
x y z\n"
.text
.global main
main:
 pushl $s_intro
 call printf
 addl $4, %esp

 pushl $s1
 call letras
 addl $4, %esp

 pushl $s2
 call letras
 addl $4, %esp

 pushl $s3
 call letras
 addl $4, %esp

 movl $0, %ebx
 movl $1, %eax
 int $0x80

letras:
 pushl %ebp
 movl %esp, %ebp
 subl $SIZE_LETRAS, %esp
 pushl %esi

 movl $0, %ecx
letras_for0:
 cmpl $NUM_LETRAS, %ecx
 jge letras_endfor0 # %ecx>=$NUM_LETRAS
 movl $0, OFFSET_LETRAS(%ebp,%ecx,4)
 incl %ecx
 jmp letras_for0
letras_endfor0:

 movl 8(%ebp), %esi
 movl $0, %ecx
letras_for1:
 cmpb $0, (%esi,%ecx)
 je letras_endfor1 # $0==(%esi,%ecx)

 pushl %ecx
 pushl (%esi,%ecx)
 pushl $sc
 call printf
 addl $8, %esp
 popl %ecx
```

```
 # passar de 'a' a 0, de 'b' a 1, ..., 'z' a 25
 movl $0, %eax
 movb (%esi,%ecx), %al
 # begin controlar que passa si (%al<'a' || %al>'z')
 cmpb $'a', %al
 jl letras_incfor1 # %al<'a'
 cmpb $'z', %al
 jg letras_incfor1 # %al>'z'
 # end controlar
 subb $'a', %al

 incl OFFSET_LETRAS(%ebp,%eax,4)

letras_incfor1:
 incl %ecx
 jmp letras_for1
letras_endfor1:

 pushl %ecx
 pushl $s_letras
 call printf
 addl $4, %esp
 popl %ecx

 movl $0, %ecx
letras_for2:
 cmpl $NUM_LETRAS, %ecx
 jge letras_endfor2 # %ecx>=$NUM_LETRAS

 pushl %ecx
 pushl OFFSET_LETRAS(%ebp,%ecx,4)
 pushl $s
 call printf
 addl $8, %esp
 popl %ecx

 incl %ecx
 jmp letras_for2
letras_endfor2:

 pushl %ecx
 pushl $sr
 call printf
 addl $4, %esp
 popl %ecx

 popl %esi
 movl %ebp, %esp
 popl %ebp
 ret
------------------------------>8
```

Problema 8

Escribe un programa que rellene un vector 'v' de enteros de 20 posiciones con la secuencia de Fibonacci. Sabiendo que:

- v[0]=1; v[1]=1; Para (i>=2) v[i]=v[i-1]+v[i-2].
- En ensamblador se puede acceder a v[i-1] así: “v(%esi,%ecx,4)” (si antes hemos hecho “movl $-4, %esi”).
- Del mismo modo se puede acceder a v[i-2] así: “v(%edi,%ecx,4)” (si antes hemos hecho “movl $-8, %edi”).

Este problema repasa los modos de direccionamiento más complejos utilizándolos para acceder a posiciones de vectores 'n' posiciones más o menos que el valor de un registro índice. Es decir, utilizando expresiones del tipo “v(%esi,%ecx,4)” donde '%esi' vale 4*n.

```
$ ./t7_grande_finale_2
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
$
```

Solución:

```
8<-[t7_grande_finale_2.s]-------
SIZE = 20
.data
s: .asciz "%d\n"
.bss
.comm v,4*SIZE,4
.text
.global main
main:
```

```
 # fibonacci[0]=1
 movl $0, %ecx
 movl $1, v(, %ecx,4)
 # fibonacci[1]=1
 movl $1, %ecx
 movl $1, v(, %ecx,4)
 # fibonacci[2..SIZE]
 movl $2, %ecx
 movl $-4, %esi
 movl $-8, %edi
loop:
 # %ecx>=$SIZE
 cmpl $SIZE, %ecx
 jge endloop

 movl v(%esi,%ecx,4), %edx
 addl v(%edi,%ecx,4), %edx
 movl %edx, v(,%ecx,4)

 incl %ecx
 jmp loop
endloop:

 # printf the array
 movl $0, %ecx
loop_print:
 cmpl $SIZE, %ecx
 jge endloop_print
 pushl %ecx
 pushl v(,%ecx,4)
 pushl $s
 call printf
 addl $8, %esp
 popl %ecx
 incl %ecx
 jmp loop_print
endloop_print:

 # the end
 movl $0, %ebx
 movl $1, %eax
 int $0x80
------------------------------>8
```

Problema 9

De nuevo, escribe un programa que rellene un vector 'v' de enteros de 20 posiciones con la secuencia de Fibonacci. Pero esta vez con una subrutina 'fibonacci' a la que se le pasa como parámetros la dirección de memoria del vector 'v' y la posición 'i' del vector a calcular (asumiendo que 'i>=2' y que las posiciones anteriores ya han sido calculadas), calcula el valor de Fibonacci(i) y lo mueve a 'v[i]'. Para ello podremos usar modos de direccionamiento donde utilicemos los cuatro elementos.

Este problema sirve para repasar algunos de los modos de direccionamiento más complejos.

```
$ ./t7_grande_finale
1
1
2
3
5
8
13
21
34
55
89
144
233
377
610
987
1597
2584
4181
6765
$
```

Solución:

```
8<-[t7_grande_finale.s]---------
SIZE = 20
.data
s: .asciz "%d\n"
.bss
.comm v,4*SIZE,4
.text
.global main
main:
 # fibonacci[0]=1
 movl $0, %ecx
 movl $1, v(, %ecx,4)
 # fibonacci[1]=1
 movl $1, %ecx
 movl $1, v(, %ecx,4)
 # fibonacci[2..SIZE]
 movl $2, %ecx
loop:
 # %ecx>=$SIZE
 cmpl $SIZE, %ecx
 jge endloop
```

```
 push %ecx
 push $v
 call fibonacci
 addl $4, %esp
 popl %ecx

 incl %ecx
 jmp loop
endloop:

 # printf the array
 movl $0, %ecx
loop_print:
 cmpl $SIZE, %ecx
 jge endloop_print
 pushl %ecx
 pushl v(,%ecx,4)
 pushl $s
 call printf
 addl $8, %esp
 popl %ecx
 incl %ecx
 jmp loop_print
endloop_print:

 # the end
 movl $0, %ebx
 movl $1, %eax
 int $0x80

fibonacci:
 pushl %ebp
 movl %esp, %ebp
 pushl %ebx

 movl 12(%ebp), %ecx
 movl 8(%ebp), %ebx

 movl -4(%ebx,%ecx,4), %edx
 addl -8(%ebx,%ecx,4), %edx
 movl %edx, (%ebx,%ecx,4)

 popl %ebx
 popl %ebp
 ret
------------------------------>8
```

Problema 10

Escribe un programa que muestre por pantalla la tabla de multiplicar del siete.

```
$ ./t7_la_taula_del_7
7 x 00 = 00
7 x 01 = 07
7 x 02 = 14
7 x 03 = 21
7 x 04 = 28
7 x 05 = 35
7 x 06 = 42
7 x 07 = 49
7 x 08 = 56
7 x 09 = 63
7 x 10 = 70
$
```

Solución:

```
8<-[t7_la_taula_del_7.s]--------
.data
s: .asciz "%d x %02d = %02d\n"
.text
.global main
main:
 movl $7, %ebx
 movl $0, %ecx
for:
 cmp $10, %ecx
 jg endfor # %ecx > $10
 movl %ecx, %eax
 imull %ebx, %eax
 pushl %eax
 pushl %ecx
 pushl %eax
 pushl %ecx
 pushl %ebx
 pushl $s
 call printf
 addl $16, %esp
 popl %ecx
 popl %eax
 incl %ecx
 jmp for
endfor:
 movl $0, %ebx
 movl $1, %eax
 int $0x80

# #include<stdio.h>
# main() {
#     int i=7, j;
#     for(j=0;j<=10;j++) {
```

```
#         printf("%d x %02d = %02d\n", i, j, i*j);
#     printf("\n");
# }
------------------------------>8
```

Problema 11

Haz un programa que escriba por pantalla las tablas de multiplicar del 0 al 10.

```
$ ./t7_taules_mul
0 x 00 = 00
0 x 01 = 00
0 x 02 = 00
0 x 03 = 00
0 x 04 = 00
0 x 05 = 00
0 x 06 = 00
0 x 07 = 00
0 x 08 = 00
0 x 09 = 00
0 x 10 = 00
1 x 00 = 00
1 x 01 = 01
1 x 02 = 02
1 x 03 = 03
[...]
9 x 08 = 72
9 x 09 = 81
9 x 10 = 90
10 x 00 = 00
10 x 01 = 10
10 x 02 = 20
10 x 03 = 30
10 x 04 = 40
10 x 05 = 50
10 x 06 = 60
10 x 07 = 70
10 x 08 = 80
10 x 09 = 90
10 x 10 = 100
$
```

Solución:

```
8<-[t7_taules_mul.s]------------
.data
s: .asciz "%d x %02d = %02d\n"
.text
.global main
main:
```

```
 movl $0, %ebx
for_ebx:
 cmp $10, %ebx
 jg endfor_ebx # %ebx > $10
 movl $0, %ecx
for_ecx:
 cmp $10, %ecx
 jg endfor_ecx # %ecx > $10
 movl %ecx, %eax
 imull %ebx, %eax
 pushl %eax
 pushl %ecx
 pushl %eax
 pushl %ecx
 pushl %ebx
 pushl $s
 call printf
 addl $16, %esp
 popl %ecx
 popl %eax
 incl %ecx
 jmp for_ecx
endfor_ecx:
 incl %ebx
 jmp for_ebx
endfor_ebx:
 movl $0, %ebx
 movl $1, %eax
 int $0x80

# #include<stdio.h>
# main() {
#     int i, j;
#     for(i=0;i<=10;i++) {
#         for(j=0;j<=10;j++) {
#             printf("%d x %02d = %02d\n", i, j, i*j);
#         }
#         printf("\n");
#     }
# }
------------------------------>8
```

Problema 12

Haz un programa que escriba por pantalla las tablas de multiplicar del 0 al 11 en tres columnas.

```
$ ./t7_taules_mul_3cols
0 x 00 = 00    1 x 00 = 00    2 x 00 = 00
0 x 01 = 00    1 x 01 = 01    2 x 01 = 02
0 x 02 = 00    1 x 02 = 02    2 x 02 = 04
0 x 03 = 00    1 x 03 = 03    2 x 03 = 06
```

```
0 x 04 = 00    1 x 04 = 04    2 x 04 = 08
0 x 05 = 00    1 x 05 = 05    2 x 05 = 10
0 x 06 = 00    1 x 06 = 06    2 x 06 = 12
0 x 07 = 00    1 x 07 = 07    2 x 07 = 14
0 x 08 = 00    1 x 08 = 08    2 x 08 = 16
0 x 09 = 00    1 x 09 = 09    2 x 09 = 18
0 x 10 = 00    1 x 10 = 10    2 x 10 = 20

3 x 00 = 00    4 x 00 = 00    5 x 00 = 00
3 x 01 = 03    4 x 01 = 04    5 x 01 = 05
3 x 02 = 06    4 x 02 = 08    5 x 02 = 10
3 x 03 = 09    4 x 03 = 12    5 x 03 = 15
3 x 04 = 12    4 x 04 = 16    5 x 04 = 20
3 x 05 = 15    4 x 05 = 20    5 x 05 = 25
3 x 06 = 18    4 x 06 = 24    5 x 06 = 30
3 x 07 = 21    4 x 07 = 28    5 x 07 = 35
3 x 08 = 24    4 x 08 = 32    5 x 08 = 40
3 x 09 = 27    4 x 09 = 36    5 x 09 = 45
3 x 10 = 30    4 x 10 = 40    5 x 10 = 50

6 x 00 = 00    7 x 00 = 00    8 x 00 = 00
6 x 01 = 06    7 x 01 = 07    8 x 01 = 08
6 x 02 = 12    7 x 02 = 14    8 x 02 = 16
6 x 03 = 18    7 x 03 = 21    8 x 03 = 24
6 x 04 = 24    7 x 04 = 28    8 x 04 = 32
6 x 05 = 30    7 x 05 = 35    8 x 05 = 40
6 x 06 = 36    7 x 06 = 42    8 x 06 = 48
6 x 07 = 42    7 x 07 = 49    8 x 07 = 56
6 x 08 = 48    7 x 08 = 56    8 x 08 = 64
6 x 09 = 54    7 x 09 = 63    8 x 09 = 72
6 x 10 = 60    7 x 10 = 70    8 x 10 = 80

9 x 00 = 00    10 x 00 = 00   11 x 00 = 00
9 x 01 = 09    10 x 01 = 10   11 x 01 = 11
9 x 02 = 18    10 x 02 = 20   11 x 02 = 22
9 x 03 = 27    10 x 03 = 30   11 x 03 = 33
9 x 04 = 36    10 x 04 = 40   11 x 04 = 44
9 x 05 = 45    10 x 05 = 50   11 x 05 = 55
9 x 06 = 54    10 x 06 = 60   11 x 06 = 66
9 x 07 = 63    10 x 07 = 70   11 x 07 = 77
9 x 08 = 72    10 x 08 = 80   11 x 08 = 88
9 x 09 = 81    10 x 09 = 90   11 x 09 = 99
9 x 10 = 90    10 x 10 = 100  11 x 10 = 110

$
```

Solución:

```
8<-[t7_taules_mul_3cols.s]------
.data
```

```
s: .asciz "%d x %02d = %02d\t%d x %02d = %02d\t%d x %02d = %02d\n"
sr: .asciz "\n"
.text
.global main
main:
 movl $0, %ebx
for_ebx:
 cmp $10, %ebx
 jg endfor_ebx # %ebx > $10
 movl $0, %ecx
for_ecx:
 cmp $10, %ecx
 jg endfor_ecx # %ecx > $10
 addl $2, %ebx
 movl %ecx, %eax
 imull %ebx, %eax
 pushl %eax
 pushl %ecx
 pushl %eax
 pushl %ecx
 pushl %ebx
 decl %ebx
 movl %ecx, %eax
 imull %ebx, %eax
 pushl %eax
 pushl %ecx
 pushl %ebx
 decl %ebx
 movl %ecx, %eax
 imull %ebx, %eax
 pushl %eax
 pushl %ecx
 pushl %ebx
 pushl $s
 call printf
 addl $40, %esp
 popl %ecx
 popl %eax
 incl %ecx
 jmp for_ecx
endfor_ecx:
 pushl $sr
 call printf
 addl $4, %esp
 addl $3, %ebx
 jmp for_ebx
endfor_ebx:
 movl $0, %ebx
 movl $1, %eax
 int $0x80

##include<stdio.h>
```

```
#main() {
#    int i, j;
#    for(i=1;i<=9;i=i+3) {
#        for(j=1;j<=10;j++) {
#            printf("%d x %02d = %02d\t%d x %02d = %02d\t%d x %02d
= %02d\n",
#             i, j, i*j, i+1, j, (i+1)*j, i+2, j, (i+2)*j);
#        }
#        printf("\n");
#    }
#}
------------------------------>8
```