UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria
de Vilanova i la Geltrú

P1 - Informe Final: Subrutinas

Arquitectura de Computadors

Daniel Aagaard Pérez i Pau Martín Nadal

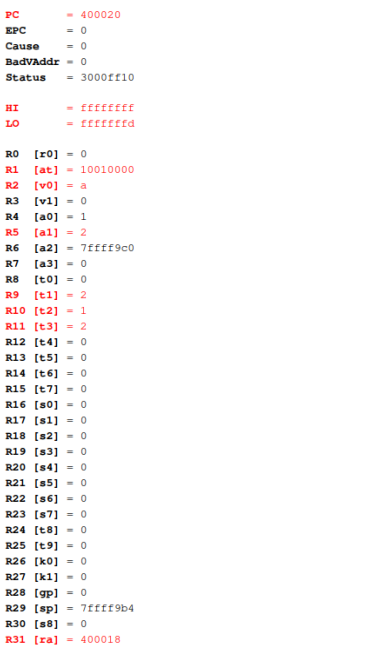
Eva Marin Tordera
I4511

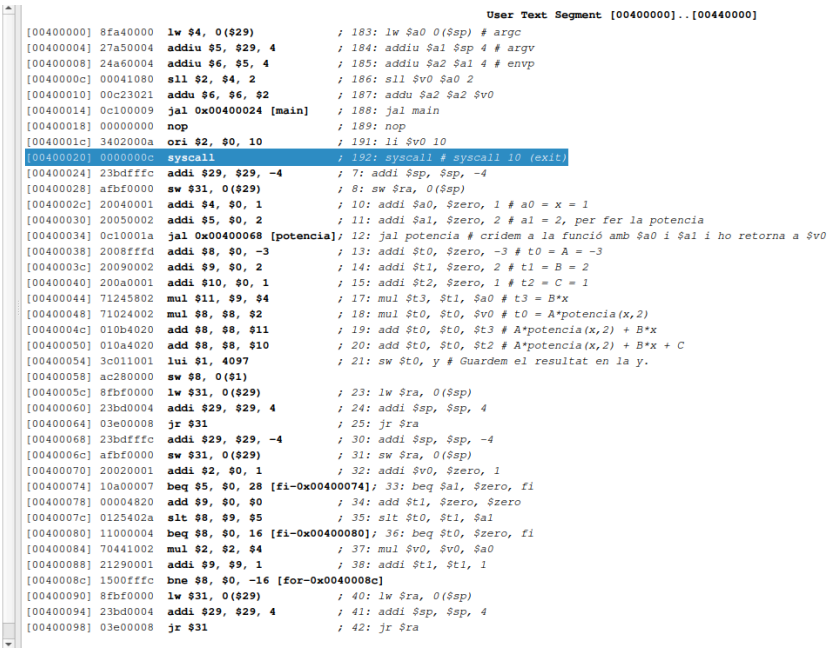
1. Traduce al ensamblador la rutina potencia(x,y) e impleméntala en el simulador junto con el programa main de la pregunta 2 del informe previo también en el ensamblador. Simúlalo paso a paso. NOTA: Aunque no sea una rutina que llame a otra ni tampoco recursiva seguiremos la metodología de al entrar en una subrutina guardar en la pila todos los registros $s0-s7 que vayamos a utilizar. Además si fuese una subrutina que llama a otra o a sí misma ha de guardar también en la pila la @ de retorno, $ra; así como los argumentos $a0 y $a1 si es necesario, por ejemplo en una recursiva.

1.1. Escribe aquí la rutina potencia(x, y) en ensamblador:

```
.data
        y: .word 0
.align 2
.text
.globl main
main:
        addi $sp, $sp, -4
        sw $ra, 0($sp)

        addi  $a0, $zero, 3   # a0 = x = 3
        addi $a1, $zero, 2    # a1 = 2, per fer la potencia
        jal potencia          # cridem a la funció amb $a0 i $a1 i retorna a $v0
        addi  $t0, $zero, 10  # t0 = A = 10
        addi  $t1, $zero, 6   # t1 = B = 6
        addi  $t2, $zero, -1  # t2 = C = -1

        mul $t3, $t1, $a0     # t3 = B*x
        mul $t0, $t0, $v0     # t0 = A*potencia(x,2)
        add $t0, $t0, $t3     # A*potencia(x,2) + B*x
        add $t0, $t0, $t2     # A*potencia(x,2) + B*x + C
        sw $t0, y             # Guardem el resultat en la y.

        lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra

.end main

potencia:
        addi $sp, $sp, -4
        sw $ra, 0($sp)
        addi $v0, $zero, 1
        beq $a1, $zero, fi
        add $t1, $zero, $zero
```

```
for:    slt $t0, $t1, $a1
        beq $t0, $zero, fi
        mul $v0, $v0, $a0
        addi $t1, $t1, 1
        bne $t0, $zero, for
fi:     lw $ra, 0($sp)
        addi $sp, $sp, 4
        jr $ra
```





1.2. Prueba con diferentes valores de A,B,C y X: A=10, B=6, C=-1, X=3, A=-3, B=2, C=1, X=1, A=10, B=-3, C=2 y X=0; cuando funcione avisa al profesor.

A=10, B=6, C=-1, X=3 –> 107 –> 0x6B

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff]  00000000
[10010000]    0000006b  00000000  00000000  00000000
[10010010]..[1003ffff]  00000000
```

A=-3, B=2, C=1, X=1 –> 0 –> 0x00

```
User data segment [10000000]..[10040000]
[10000000]..[1003ffff]  00000000
```

A=10, B=-3, C=2, X=0 –> 2 –> 0x02

```
User data segment [10000000]..[10040000]
[10000000]..[1000ffff]  00000000
[10010000]    00000002  00000000  00000000  00000000    . . . . . . . . . . . . . . . .
[10010010]..[1003ffff]  00000000
```

2. A partir del ejercicio 3 del trabajo previo, implementa el main y la subrutina en el simulador.

a) En el archivo ejercicio3.s debes completar la traducción. Simúlalo paso a paso.

```
.data
result: .word 0
.text
.globl main
main:
addi $sp, $sp, -4
sw $ra, 0($sp)
addi $a0, $zero, 2 # x = 2
addi $a1, $zero, 5 # y = 5
jal potencia_recursiva
sw $v0, result # guardem a memoria
lw $ra, 0($sp)
addi $sp, $sp, 4
jr $ra
.end main
potencia_recursiva:
addi $sp, $sp, -12 # fem espai dos arguments i l'adreça de retorn
sw $ra, 8($sp)
sw $a0, 4($sp)
sw $a1, 0($sp)
bne $a1, $zero, cond2 # if(y != 0) cond 2
addi $v0, $zero, 1 # v0 = 1
addi $sp, $sp, 12 # pop 3 items de la pila
jr $ra

cond2: addi $t0, $zero, 1
bne $t0, $a1, else
```

```
add $v0, $zero, $a0
addi $sp, $sp, 12 # pop 3 items de la pila
jr $ra
else:
srl $a1, $a1, 1 # y = y >> 1 // y = y/2
jal potencia_recursiva
lw $t0, 0($sp) # t0 = y
addi $sp, $sp, -4 # push $v0
sw $v0, 0($sp) # guardem $v0 a la pila
srl $a1, $t0, 1 # a1 = y/2
sub $a1, $t0, $a1 # a1 = y-(y/2)
jal potencia_recursiva
lw $t1, 0($sp) # t1 = potencia_recursiva(x,y/2)
addi $sp, $sp, 4
mul $v0, $t1, $v0 #v0=potencia_recursiva(x,y/2)*potencia_recursiva(x,y-(y/2))
lw $ra, 8($sp)
addi $sp, $sp, 12
jr $ra
```

```
PC        = 400034
EPC       = 0
Cause     = 0
BadVAddr  = 0
Status    = 3000ff10

HI        = 0
LO        = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 2
R5  [a1] = 5
R6  [a2] = 7ffff9c0
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff9b0
R30 [s8] = 0
R31 [ra] = 400018
```

```
                                                    User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)            ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4         ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4          ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2            ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2          ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]    ; 188: jal main
[00400018] 00000000  nop                      ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10           ; 191: li $v0 10
[00400020] 0000000c  syscall                  ; 192: syscall # syscall 10 (exit)
[00400024] 23bdfffc  addi $29, $29, -4        ; 6: addi $sp, $sp, -4
[00400028] afbf0000  sw $31, 0($29)           ; 7: sw $ra, 0($sp)
[0040002c] 20040002  addi $4, $0, 2           ; 8: addi $a0, $zero, 2 #x = 2
[00400030] 20050005  addi $5, $0, 5           ; 9: addi $a1, $zero, 5 #y = 5
[00400034] 0c100013  jal 0x0040004c [potencia_recursiva]
[00400038] 3c011001  lui $1, 4097             ; 11: sw $v0, result #guardem a memoria
[0040003c] ac220000  sw $2, 0($1)
[00400040] 8fbf0000  lw $31, 0($29)           ; 12: lw $ra, 0($sp)
[00400044] 23bd0004  addi $29, $29, 4         ; 13: addi $sp, $sp, 4
[00400048] 03e00008  jr $31                   ; 14: jr $ra
[0040004c] 23bdfff4  addi $29, $29, -12       ; 19: addi $sp, $sp, -12 # fem espai pels dos arguments i 1 adrea de retorn
[00400050] afbf0008  sw $31, 8($29)           ; 20: sw $ra, 8($sp)
[00400054] afa40004  sw $4, 4($29)            ; 21: sw $a0, 4($sp)
[00400058] afa50000  sw $5, 0($29)            ; 22: sw $a1, 0($sp)
[0040005c] 14a00004  bne $5, $0, 16 [cond2-0x0040005c]
[00400060] 20020001  addi $2, $0, 1           ; 25: addi $v0, $zero, 1 #v0 = 1
[00400064] 23bd000c  addi $29, $29, 12        ; 26: addi $sp, $sp, 12 #pop 3 items from stack
[00400068] 03e00008  jr $31                   ; 27: jr $ra
[0040006c] 20080001  addi $8, $0, 1           ; 29: addi $t0, $zero, 1
[00400070] 15050004  bne $8, $5, 16 [else-0x00400070]
[00400074] 00041020  add $2, $0, $4           ; 31: add $v0, $zero, $a0
[00400078] 23bd000c  addi $29, $29, 12        ; 32: addi $sp, $sp, 12 #pop 3 items form stack
[0040007c] 03e00008  jr $31                   ; 33: jr $ra
[00400080] 00052842  srl $5, $5, 1            ; 36: srl $a1, $a1, 1 # y = y >> 1 // y = y/2
[00400084] 0c100013  jal 0x0040004c [potencia_recursiva]
[00400088] 8fa80000  lw $8, 0($29)            ; 38: lw $t0, 0($sp) #t0 = y
[0040008c] 23bdfffc  addi $29, $29, -4        ; 39: addi $sp, $sp, -4 #push $v0
[00400090] afa20000  sw $2, 0($29)            ; 40: sw $v0, 0($sp) #save $v0 to the stack
[00400094] 00082842  srl $5, $8, 1            ; 41: srl $a1, $t0, 1 #a1 = y/2
[00400098] 01052822  sub $5, $8, $5           ; 42: sub $a1, $t0, $a1 #a1 = y-(y/2)
[0040009c] 0c100013  jal 0x0040004c [potencia_recursiva]
[004000a0] 8fa90000  lw $9, 0($29)            ; 44: lw $t1, 0($sp) #t1 = potencia_recursiva(x,y/2)
[004000a4] 23bd0004  addi $29, $29, 4         ; 45: addi $sp, $sp, 4
[004000a8] 71221002  mul $2, $9, $2           ; 46: mul $v0, $t1, $v0 #v0 = potencia_recursiva(x,y/2) *potencia_recursiva(x,y-(y/2))
```

```
PC       = 400054
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 0

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 2
R5  [a1] = 2
R6  [a2] = 7ffff9c0
R7  [a3] = 0
R8  [t0] = 1
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff998
R30 [s8] = 0
R31 [ra] = 400088
```

User Text Segment [00400000]..[00440000]

```
[00400000] 8fa40000  lw $4, 0($29)            ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4         ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4          ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2            ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2          ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]    ; 188: jal main
[00400018] 00000000  nop                      ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10           ; 191: li $v0 10
[00400020] 0000000c  syscall                  ; 192: syscall # syscall 10 (exit)
[00400024] 23bdfffc  addi $29, $29, -4        ; 6: addi $sp, $sp, -4
[00400028] afbf0000  sw $31, 0($29)           ; 7: sw $ra, 0($sp)
[0040002c] 20040002  addi $4, $0, 2           ; 8: addi $a0, $zero, 2 #x = 2
[00400030] 20050005  addi $5, $0, 5           ; 9: addi $a1, $zero, 5 #y = 5
[00400034] 0c100013  jal 0x0040004c [potencia_recursiva]
[00400038] 3c011001  lui $1, 4097             ; 11: sw $v0, result #guardem a memoria
[0040003c] ac220000  sw $2, 0($1)
[00400040] 8fbf0000  lw $31, 0($29)           ; 12: lw $ra, 0($sp)
[00400044] 23bd0004  addi $29, $29, 4         ; 13: addi $sp, $sp, 4
[00400048] 03e00008  jr $31                   ; 14: jr $ra
[0040004c] 23bdfff4  addi $29, $29, -12       ; 19: addi $sp, $sp, -12 # fem espai pels dos arguments i 1 adrea de retorn
[00400050] afbf0008  sw $31, 8($29)           ; 20: sw $ra, 8($sp)
[00400054] afa40004  sw $4, 4($29)            ; 21: sw $a0, 4($sp)
[00400058] afa50000  sw $5, 0($29)            ; 22: sw $a1, 0($sp)
[0040005c] 14a00004  bne $5, $0, 16 [cond2-0x0040005c]
[00400060] 20020001  addi $2, $0, 1           ; 25: addi $v0, $zero, 1 #v0 = 1
[00400064] 23bd000c  addi $29, $29, 12        ; 26: addi $sp, $sp, 12 #pop 3 items from stack
[00400068] 03e00008  jr $31                   ; 27: jr $ra
[0040006c] 20080001  addi $8, $0, 1           ; 29: addi $t0, $zero, 1
[00400070] 15050004  bne $8, $5, 16 [else-0x00400070]
[00400074] 00041020  add $2, $0, $4           ; 31: add $v0, $zero, $a0
[00400078] 23bd000c  addi $29, $29, 12        ; 32: addi $sp, $sp, 12 #pop 3 items form stack
[0040007c] 03e00008  jr $31                   ; 33: jr $ra
[00400080] 00052842  srl $5, $5, 1            ; 36: srl $a1, $a1, 1 # y = y >> 1 // y = y/2
[00400084] 0c100013  jal 0x0040004c [potencia_recursiva]
[00400088] 8fa80000  lw $8, 0($29)            ; 38: lw $t0, 0($sp) #t0 = y
[0040008c] 23bdfffc  addi $29, $29, -4        ; 39: addi $sp, $sp, -4 #push $v0
[00400090] afa20000  sw $2, 0($29)            ; 40: sw $v0, 0($sp) #save $v0 to the stack
[00400094] 00082842  srl $5, $8, 1            ; 41: srl $a1, $t0, 1 #a1 = y/2
[00400098] 01052822  sub $5, $8, $5           ; 42: sub $a1, $t0, $a1 #a1 = y-(y/2)
[0040009c] 0c100013  jal 0x0040004c [potencia_recursiva]
[004000a0] 8fa90000  lw $9, 0($29)            ; 44: lw $t1, 0($sp) #t1 = potencia_recursiva(x,y/2)
[004000a4] 23bd0004  addi $29, $29, 4         ; 45: addi $sp, $sp, 4
[004000a8] 71221002  mul $2, $9, $2           ; 46: mul $v0, $t1, $v0 #v0 = potencia_recursiva(x,y/2) *potencia_recursiva(x,y-(y/2))
```

```
PC       = 4000b4
EPC      = 0
Cause    = 0
BadVAddr = 0
Status   = 3000ff10

HI       = 0
LO       = 4

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 4
R3  [v1] = 0
R4  [a0] = 2
R5  [a1] = 1
R6  [a2] = 7ffff9c0
R7  [a3] = 0
R8  [t0] = 1
R9  [t1] = 2
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
R28 [gp] = 10008000
R29 [sp] = 7ffff9a4
R30 [s8] = 0
R31 [ra] = 400088
```

```
[00400008] 24a60004  addiu $6, $5, 4          ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2            ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2          ; 187: addu $a2 $a2 $v0
[00400014] 0c100009  jal 0x00400024 [main]    ; 188: jal main
[00400018] 00000000  nop                      ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10           ; 191: li $v0 10
[00400020] 0000000c  syscall                  ; 192: syscall # syscall 10 (exit)
[00400024] 23bdfffc  addi $29, $29, -4        ; 6: addi $sp, $sp, -4
[00400028] afbf0000  sw $31, 0($29)           ; 7: sw $ra, 0($sp)
[0040002c] 20040002  addi $4, $0, 2           ; 8: addi $a0, $zero, 2 #x = 2
[00400030] 20050005  addi $5, $0, 5           ; 9: addi $a1, $zero, 5 #y = 5
[00400034] 0c100013  jal 0x0040004c [potencia_recursiva]
[00400038] 3c011001  lui $1, 4097             ; 11: sw $v0, result #guardem a memoria
[0040003c] ac220000  sw $2, 0($1)
[00400040] 8fbf0000  lw $31, 0($29)           ; 12: lw $ra, 0($sp)
[00400044] 23bd0004  addi $29, $29, 4         ; 13: addi $sp, $sp, 4
[00400048] 03e00008  jr $31                   ; 14: jr $ra
[0040004c] 23bdfff4  addi $29, $29, -12       ; 19: addi $sp, $sp, -12 # fem espai pels dos arguments i 1 adrea de retorn
[00400050] afbf0008  sw $31, 8($29)           ; 20: sw $ra, 8($sp)
[00400054] afa40004  sw $4, 4($29)            ; 21: sw $a0, 4($sp)
[00400058] afa50000  sw $5, 0($29)            ; 22: sw $a1, 0($sp)
[0040005c] 14a00004  bne $5, $0, 16 [cond2-0x0040005c]
[00400060] 20020001  addi $2, $0, 1           ; 25: addi $v0, $zero, 1 #v0 = 1
[00400064] 23bd000c  addi $29, $29, 12        ; 26: addi $sp, $sp, 12 #pop 3 items from stack
[00400068] 03e00008  jr $31                   ; 27: jr $ra
[0040006c] 20080001  addi $8, $0, 1           ; 29: addi $t0, $zero, 1
[00400070] 15050004  bne $8, $5, 16 [else-0x00400070]
[00400074] 00041020  add $2, $0, $4           ; 31: add $v0, $zero, $a0
[00400078] 23bd000c  addi $29, $29, 12        ; 32: addi $sp, $sp, 12 #pop 3 items form stack
[0040007c] 03e00008  jr $31                   ; 33: jr $ra
[00400080] 00052842  srl $5, $5, 1            ; 36: srl $a1, $a1, 1 # y = y >> 1 // y = y/2
[00400084] 0c100013  jal 0x0040004c [potencia_recursiva]
[00400088] 8fa80000  lw $8, 0($29)            ; 38: lw $t0, 0($sp) #t0 = y
[0040008c] 23bdfffc  addi $29, $29, -4        ; 39: addi $sp, $sp, -4 #push $v0
[00400090] afa20000  sw $2, 0($29)            ; 40: sw $v0, 0($sp) #save $v0 to the stack
[00400094] 00082842  srl $5, $8, 1            ; 41: srl $a1, $t0, 1 #a1 = y/2
[00400098] 01052822  sub $5, $8, $5           ; 42: sub $a1, $t0, $a1 #a1 = y-(y/2)
[0040009c] 0c100013  jal 0x0040004c [potencia_recursiva]
[004000a0] 8fa90000  lw $9, 0($29)            ; 44: lw $t1, 0($sp) #t1 = potencia_recursiva(x,y/2)
[004000a4] 23bd0004  addi $29, $29, 4         ; 45: addi $sp, $sp, 4
[004000a8] 71221002  mul $2, $9, $2           ; 46: mul $v0, $t1, $v0 #v0 = potencia_recursiva(x,y/2) *potencia_recursiva(x,y-(y/2))
[004000ac] 8fbf0008  lw $31, 8($29)           ; 47: lw $ra, 8($sp)
[004000b0] 23bd000c  addi $29, $29, 12        ; 48: addi $sp, $sp, 12
[004000b4] 03e00008  jr $31                   ; 49: jr $ra
```

b) Si el programa main llama a la subrutina potencia_recursiva con unos valores iniciales de x=2 e y=5, como está en el enunciado en C, describe la ejecución dinámica del programa, ayúdate simulando paso a paso en el simulador. Avisa al profesor cuando funcione.

Primer ens guardem la direcció de retorn del main, perquè la necesitarem després de cridar a la funció de potencia_recursiva. Després fiquem els valors de x e y a $a0 i $a1 respectivament perquè els necessitem per a la funció potencia_recursiva.

Un cop dins de potencia_recursiva, primer fem espai per guardar-nos els dos arguments i la direcció de retorn de la funció, perquè els necessitarem i en ser una funció recursiva en tornar-la a cridar podem perdre aquests valors.

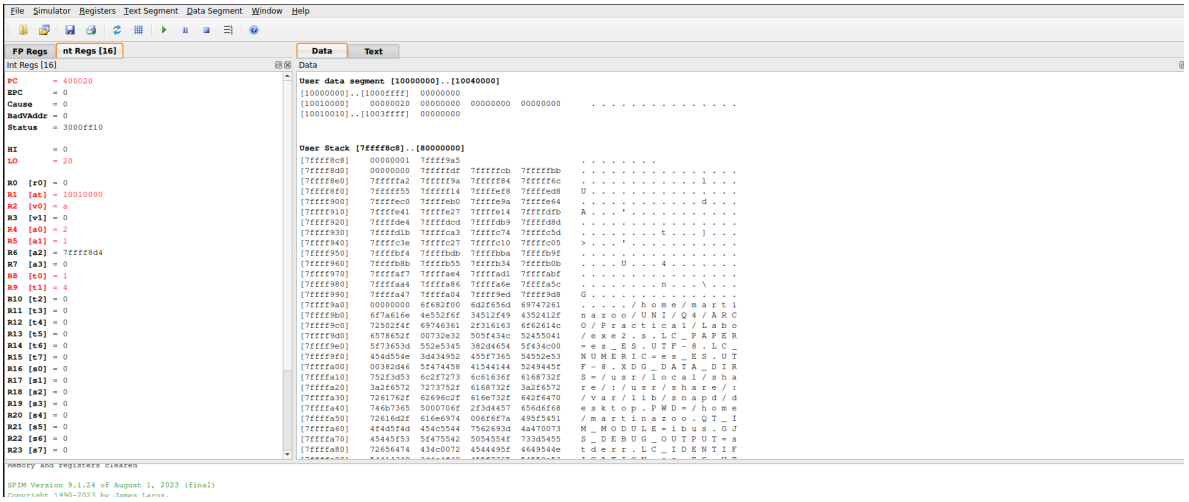
En aquest cas com la y = 5, llavors saltem directament al else que divideix la y entre 2 i tornem a cridar a la funció. Això ho fa dos cops i llavors accedeix quan la y == 1. Sumem el valor de la x a $v0 que serà on retornarem el resultat final de la funció.

Llavors tornem a agafar de la pila el valor inicial de y i afegim el resultat de la primera crida recursiva a la pila (potencia_recursiva(x,y/2)), perquè ho necessitarem després per calcular el resultat final.

Més tard, preparem les noves variables per a la crida de la funció, com que la y == 1 fem la suma del valor de la x a $v0 que serà on retornarem el resultat final de la funció.

Finalment, recuperem el resultat final de la primera crida recursiva de la pila i el multipliquem pel resultat aconseguit en la segona crida recursiva. Després d'això es faran totes les crides recursives que falten de fer-se.

Al final tornem a agafar la direcció de retorn que es troba a la pila i tornem al main. Guardem el resultat a la variable global resultat i tornem a agafar l'adreça de memòria del main de la pila.


File Simulator Registers Text Segment Data Segment Window Help
FP Regs
nt Regs [16]
Int Regs [16]
PC = 400020
EPC = 0
Cause = 0
BadVAddr = 0
Status = 3000ff10
HI = 0
LO = 20
R0 [r0] = 0
R1 [at] = 10010000
R2 [v0] = a
R3 [v1] = 0
R4 [a0] = 2
R5 [a1] = 1
R6 [a2] = 7ffff8d4
R7 [a3] = 0
R8 [t0] = 1
R9 [t1] = 4
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
Data
Text
Data
User data segment [10000000]..[10040000]
[10000000]..[1000ffff] 00000000
[10010000] 00000020 00000000 00000000 00000000 . . . . . .
[10010010]..[1003ffff] 00000000
User Stack [7fffff8c8]..[80000000]
[7fffff8c8] 00000001 7fffff9a5 . . . . . .
[7fffff8d0] 00000000 7fffffdf 7fffffcb 7fffffbb . . . . . .
[7fffff8e0] 7ffffa2 7fffff9a 7fffff84 7fffff6c . . . . . l . . .
[7fffff8f0] 7fffff55 7fffff14 7fffef8 7fffed8 U . . . . . .
[7ffff900] 7ffffec0 7ffffeb0 7ffffe9a 7ffffe64 . . . . . d . . .
[7ffff910] 7ffffe41 7ffffe27 7ffffe14 7ffffdfb A . . . ' . . . . . .
[7ffff920] 7ffffde4 7ffffdcd 7ffffdb9 7ffffd8d . . . . . .
[7ffff930] 7ffffd1b 7ffffca3 7ffffc74 7ffffc5d . . . . . t . . .] . . .
[7ffff940] 7ffffc3e 7ffffc27 7ffffc10 7ffffc05 > . . . ' . . . . . .
[7ffff950] 7ffffbf4 7ffffbdb 7ffffbba 7ffffb9f . . . . . .
[7ffff960] 7ffffb8b 7ffffb55 7ffffb34 7ffffb0b . . . . U . . . 4 . . . . . .
[7ffff970] 7ffffaf7 7ffffae4 7ffffad1 7ffffabf . . . . . .
[7ffff980] 7ffffaa4 7ffffa86 7ffffa6e 7ffffa5c . . . . . n . . . \ . . .
[7ffff990] 7ffffa47 7ffffa04 7ffff9ed 7ffff9d8 G . . . . . .
[7ffff9a0] 00000000 6f682f00 6d2f656d 69747261 . . . . . / h o m e / m a r t i
[7ffff9b0] 6f7a616e 4e552f6f 34512f49 4352412f n a z o o / U N I / Q 4 / A R C
[7ffff9c0] 72502f4f 69746361 2f316163 6f62614c O / P r a c t i c a l / L a b o
[7ffff9d0] 6578652f 00732e32 505f434c 52455041 / e x e 2 . s . L C _ P A P E R
[7ffff9e0] 5f73653d 552e5345 382d4654 5f434c00 = e s _ E S . U T F - 8 . L C _
[7ffff9f0] 454d554e 3d434952 455f7365 54552e53 N U M E R I C = e s _ E S . U T
[7ffffa00] 00382d46 5f474458 41544144 5249445f F - 8 . X D G _ D A T A _ D I R
[7ffffa10] 752f3d53 6c2f7273 6c61636f 6168732f S = / u s r / l o c a l / s h a
[7ffffa20] 3a2f6572 7273752f 6168732f 3a2f6572 r e / : / u s r / s h a r e / :
[7ffffa30] 7261762f 62696c2f 616e732f 642f6470 / v a r / l i b / s n a p d / d
[7ffffa40] 746b7365 5000706f 2f3d4457 656d6f68 e s k t o p . P W D = / h o m e
[7ffffa50] 72616d2f 616e6974 006f6f7a 495f5451 / m a r t i n a z o o . Q T _ I
[7ffffa60] 4f4d5f4d 454c5544 7562693d 4a470073 M _ M O D U L E = i b u s . G J
[7ffffa70] 45445f53 5f475542 5054554f 733d5455 S _ D E B U G _ O U T P U T = s
[7ffffa80] 72656474 434c0072 4544495f 4649544e t d e r r . L C _ I D E N T I F
memory and registers cleared
SPIM Version 9.1.24 of August 1, 2023 (final)
Copyright 1990-2023 by James Larus.