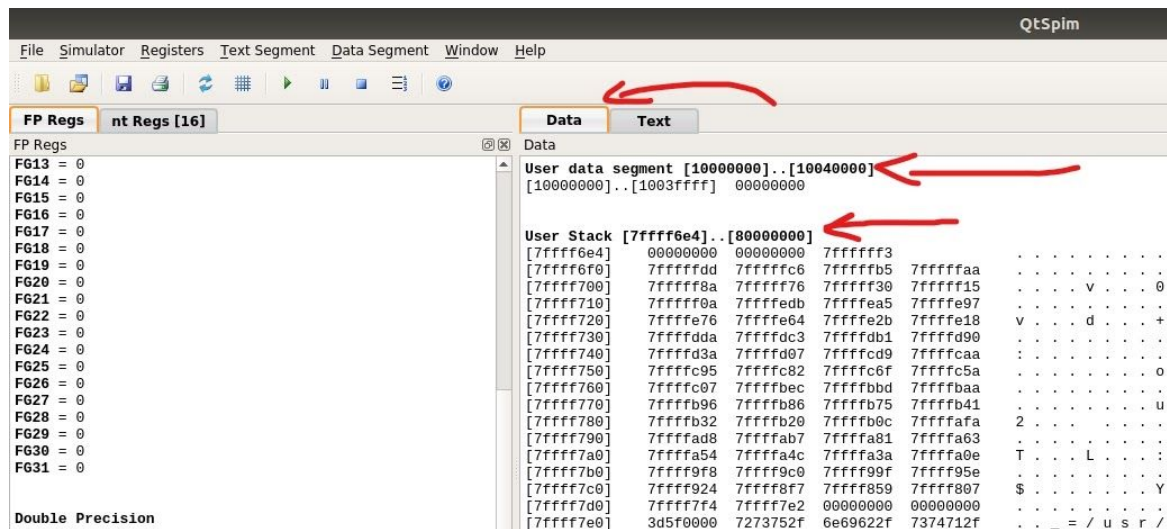


Práctica 1. Utilización del emulador QTSpm y representación de la información.

→ https://drive.google.com/file/d/1_7yaKfOIJgbVp-YHw2vy0eSqZjlNsyiN/view

a) Identificar en el emulador los siguientes elementos:

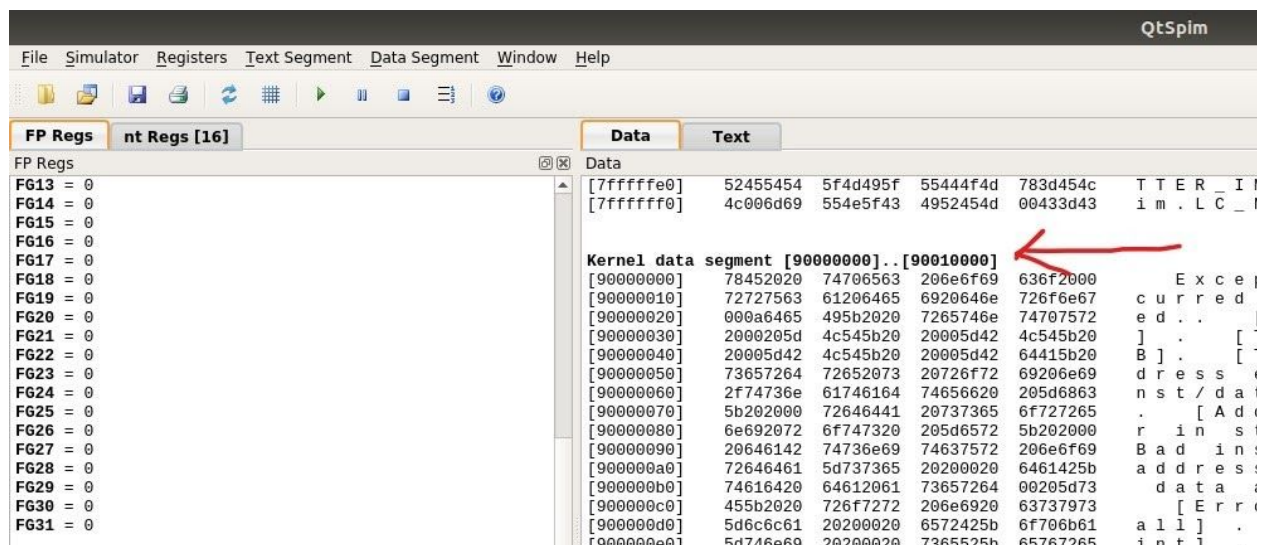
I. El segmento de Datos (Usuario, Kernel y Pila)



Segmento de Datos = Data

Usuario = User data segment

Pila = User Stack



Kernel = Kernel data segment

II. El segmento de Instrucciones (Usuario y Kernel)

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs nt Regs [16] Data Text

FP Regs

FP13 = 0
FP14 = 0
FP15 = 0
FP16 = 0
FP17 = 0
FP18 = 0
FP19 = 0
FP20 = 0
FP21 = 0
FP22 = 0
FP23 = 0
FP24 = 0
FP25 = 0
FP26 = 0
FP27 = 0
FP28 = 0
FP29 = 0
FP30 = 0

Double Precision

FP0 = 0
FP2 = 0
FP4 = 0
FP6 = 0
FP8 = 0
FP10 = 0
FP12 = 0
FP14 = 0
FP16 = 0
FP18 = 0
FP20 = 0
FP22 = 0
FP24 = 0
FP26 = 0
FP28 = 0
FP30 = 0

Text

[00400000] 8fa40000 lw \$4, 0(\$20) ; 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] 00041000 sll \$2, \$4, 2 ; 186: sll \$v0 \$a0 2
[00400010] 00c23021 addu \$6, \$6, \$2 ; 187: addu \$a2 \$a2 \$v0
[00400014] 0c000000 jal 0x00000000 [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)

User Text Segment [00400000]..[00440000]

[80000180] 0001d821 addu \$27, \$0, \$1 ; 90: move \$k1 \$at # Save \$at
[80000184] 3c019000 lui \$1, -28672 ; 92: sw \$v0 \$1 # Not re-entrant and we can't trust \$sp
[80000188] ac220200 sw \$2, 512(\$1) ; 93: sw \$a0 \$2 # But we need to use these registers
[8000018c] 3c019000 lui \$1, -28672
[80000190] ac240204 sw \$4, 516(\$1)
[80000194] 401a6800 mfc0 \$26, \$13 ; 95: mfc0 \$k0 \$13 # Cause register
[80000198] 001a2082 srl \$4, \$26, 2 ; 96: srl \$a0 \$k0 2 # Extract ExcCode Field
[8000019c] 3084001f andi \$4, \$4, 31 ; 97: andi \$a0 \$a0 0x1f
[800001a0] 34020004 ori \$2, \$0, 4 ; 101: li \$v0 4 # syscall 4 (print_str)
[800001a4] 3c049000 lui \$4, -28672 [_m1_] ; 102: la \$a0 _m1_
[800001a8] 0000000c syscall ; 103: syscall
[800001ac] 34020001 ori \$2, \$0, 1 ; 105: li \$v0 1 # syscall 1 (print_int)
[800001b0] 001a2082 srl \$4, \$26, 2 ; 106: srl \$a0 \$k0 2 # Extract ExcCode Field
[800001b4] 3084001f andi \$4, \$4, 31 ; 107: andi \$a0 \$a0 0x1f
[800001b8] 0000000c syscall ; 108: syscall
[800001bc] 34020004 ori \$2, \$0, 4 ; 110: li \$v0 4 # syscall 4 (print_str)
[800001c0] 3344003c andi \$4, \$26, 60 ; 111: andi \$a0 \$k0 0x3c
[800001c4] 3c019000 lui \$1, -28672 ; 112: lw \$a0 __excpc(\$a0)
[800001c8] 00240821 addu \$1, \$1, \$4
[800001cc] 8c240180 lw \$4, 384(\$1)
[800001d0] 00000000 nop ; 113: nop
[800001d4] 0000000c syscall ; 114: syscall
[800001d8] 34010018 ori \$1, \$0, 24 ; 116: bne \$k0 0x18 ok_pc # Bad PC exception requires special checks
[800001dc] 143a0008 bne \$1, \$26, 32 [ok_pc-0x800001dc]
[800001e0] 00000000 nop ; 117: nop
[800001e4] 40047000 mfc0 \$4, \$14 ; 119: mfc0 \$a0 \$14 # EPC
[800001e8] 30840003 andi \$4, \$4, 3 ; 120: andi \$a0 \$a0 0x3 # Is EPC word-aligned?
[800001ec] 10040004 beq \$0, \$4, 16 [ok_pc-0x800001ec]

Kernel Text Segment [80000000]..[80010000]

Memory and registers cleared

SPIM Version 9.1.20 of August 29, 2017
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

Usuario = User Text Segment

Kernel = Kernel Text Segment

III. El contenido de los Registros Enteros

QtSpim

File Simulator Registers Text Segment Data Segment Window Help

FP Regs nt Regs [16] Data Text

Int Regs [16]

PC = 0
EPC = 0
Cause = 0
BadAddr = 0
Status = 3000fff0
HI = 0
LO = 0
R0 [r0] = 0
R1 [at] = 0
R2 [v0] = 0
R3 [v1] = 0
R4 [a0] = 0
R5 [a1] = 0
R6 [a2] = 7fffffec
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] = 10000000
R29 [sp] = 7fffffec
R30 [ra] = 0

Text

[00400000] 8fa40000 lw \$4, 0(\$20) ; 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] 00041000 sll \$2, \$4, 2 ; 186: sll \$v0 \$a0 2
[00400010] 00c23021 addu \$6, \$6, \$2 ; 187: addu \$a2 \$a2 \$v0
[00400014] 0c000000 jal 0x00000000 [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)

User Text Segment [00400000]..[00440000]

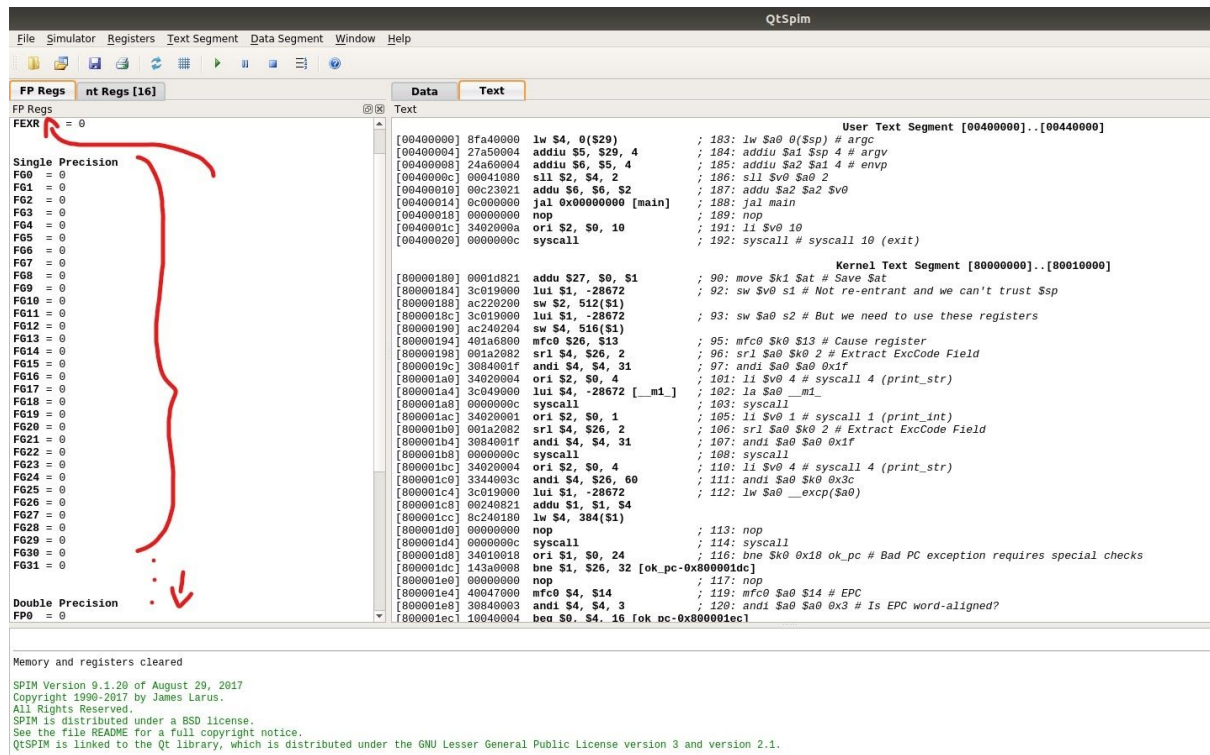
[80000180] 0001d821 addu \$27, \$0, \$1 ; 90: move \$k1 \$at # Save \$at
[80000184] 3c019000 lui \$1, -28672 ; 92: sw \$v0 \$1 # Not re-entrant and we can't trust \$sp
[80000188] ac220200 sw \$2, 512(\$1) ; 93: sw \$a0 \$2 # But we need to use these registers
[8000018c] 3c019000 lui \$1, -28672
[80000190] ac240204 sw \$4, 516(\$1)
[80000194] 401a6800 mfc0 \$26, \$13 ; 95: mfc0 \$k0 \$13 # Cause register
[80000198] 001a2082 srl \$4, \$26, 2 ; 96: srl \$a0 \$k0 2 # Extract ExcCode Field
[8000019c] 3084001f andi \$4, \$4, 31 ; 97: andi \$a0 \$a0 0x1f
[800001a0] 34020004 ori \$2, \$0, 4 ; 101: li \$v0 4 # syscall 4 (print_str)
[800001a4] 3c049000 lui \$4, -28672 [_m1_] ; 102: la \$a0 _m1_
[800001a8] 0000000c syscall ; 103: syscall
[800001ac] 34020001 ori \$2, \$0, 1 ; 105: li \$v0 1 # syscall 1 (print_int)
[800001b0] 001a2082 srl \$4, \$26, 2 ; 106: srl \$a0 \$k0 2 # Extract ExcCode Field
[800001b4] 3084001f andi \$4, \$4, 31 ; 107: andi \$a0 \$a0 0x1f
[800001b8] 0000000c syscall ; 108: syscall
[800001bc] 34020004 ori \$2, \$0, 4 ; 110: li \$v0 4 # syscall 4 (print_str)
[800001c0] 3344003c andi \$4, \$26, 60 ; 111: andi \$a0 \$k0 0x3c
[800001c4] 3c019000 lui \$1, -28672 ; 112: lw \$a0 __excpc(\$a0)
[800001c8] 00240821 addu \$1, \$1, \$4
[800001cc] 8c240180 lw \$4, 384(\$1)
[800001d0] 00000000 nop ; 113: nop
[800001d4] 0000000c syscall ; 114: syscall
[800001d8] 34010018 ori \$1, \$0, 24 ; 116: bne \$k0 0x18 ok_pc # Bad PC exception requires special checks
[800001dc] 143a0008 bne \$1, \$26, 32 [ok_pc-0x800001dc]
[800001e0] 00000000 nop ; 117: nop
[800001e4] 40047000 mfc0 \$4, \$14 ; 119: mfc0 \$a0 \$14 # EPC
[800001e8] 30840003 andi \$4, \$4, 3 ; 120: andi \$a0 \$a0 0x3 # Is EPC word-aligned?
[800001ec] 10040004 beq \$0, \$4, 16 [ok_pc-0x800001ec]

Kernel Text Segment [80000000]..[80010000]

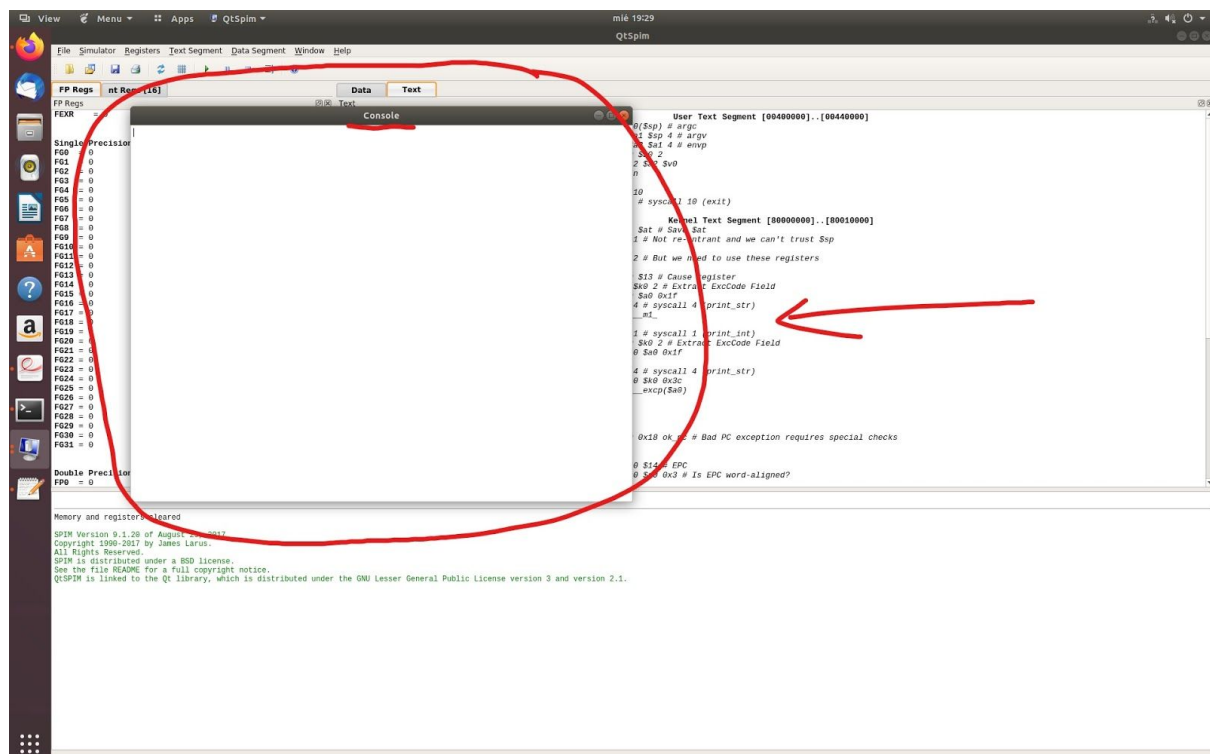
Memory and registers cleared

SPIM Version 9.1.20 of August 29, 2017
Copyright 1990-2017 by James Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
QtSPIM is linked to the Qt library, which is distributed under the GNU Lesser General Public License version 3 and version 2.1.

IV. El contenido de los Registros en Punto Flotante.

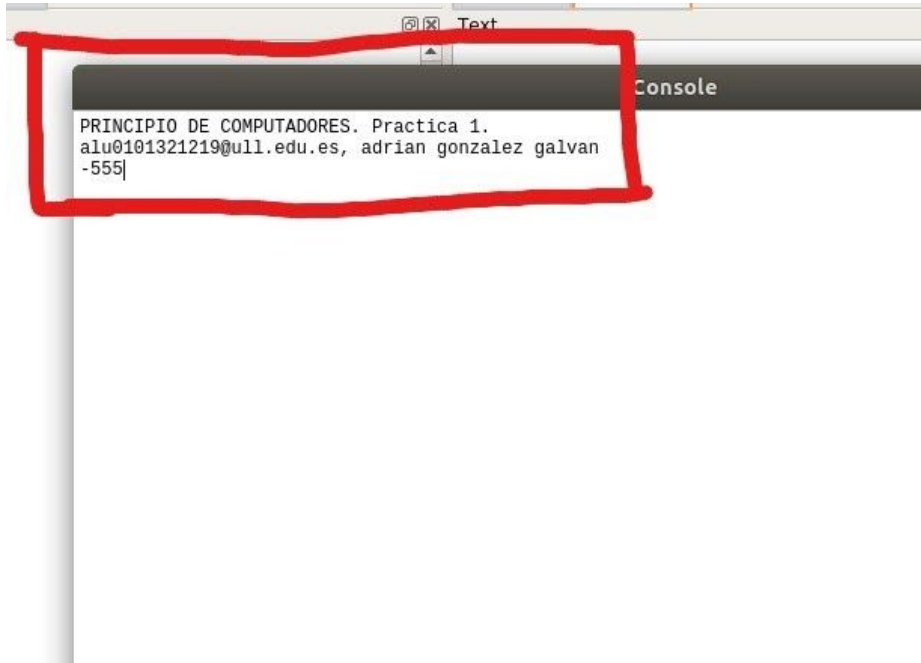


V. La consola del sistema



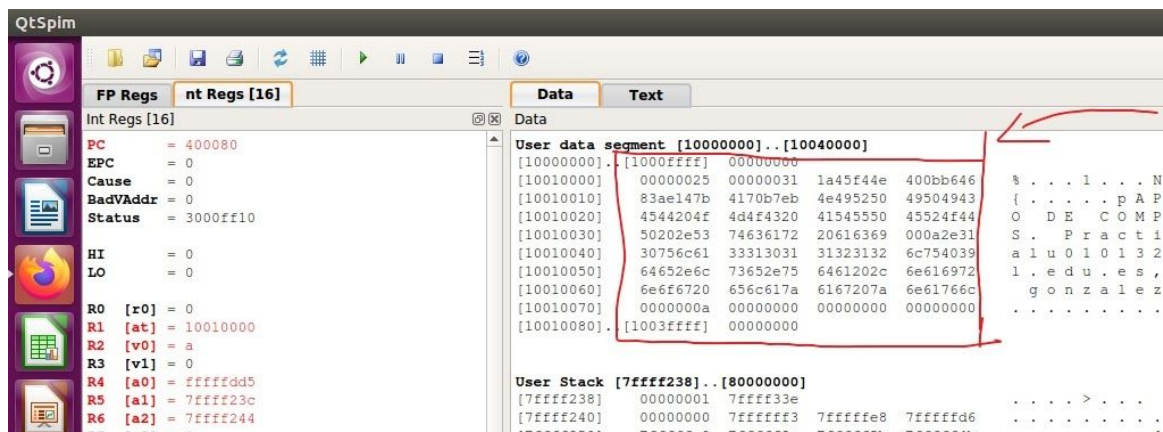
- b) Edita con un editor de textos plano (vi, vim, gedit, kate o el que prefieras) el fichero practica1.s y sustituye la cadena "alu9999999999@ull.edu.es, nombre

apellido1 apellido2\n" con tu dirección de correo, nombre y tus apellidos y graba el fichero. A continuación carga el programa en QtSpim y ejecuta el programa de una sola vez. Saca un pantallazo de la consola y marca mediante un cuadro rojo la impresión de tus datos.

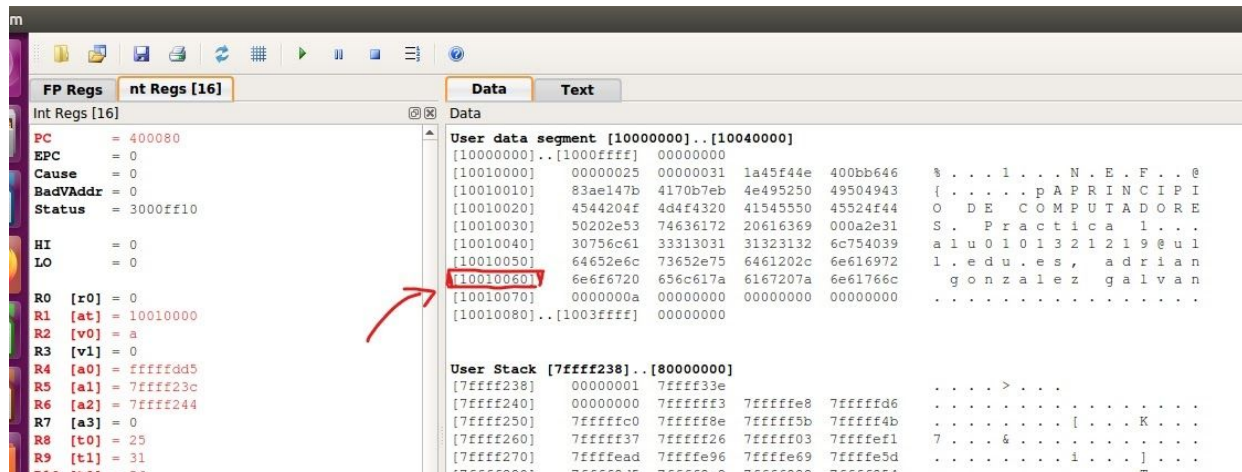


- c) Explora el segmento de datos. (NOTA: Al sacar los pantallazos debes incluir las direcciones y los valores del User Data Segment completo, ya que los valores y las direcciones dependerán del nombre de cada alumno y serán necesarios para poder realizar la corrección).

Comprueba que el segmento de datos está representado en Hexadecimal (en el menú Data Segment marca el checkbox correspondiente a Hex). Con las indicaciones que te dé tu profesor en la práctica responde a las siguientes preguntas:



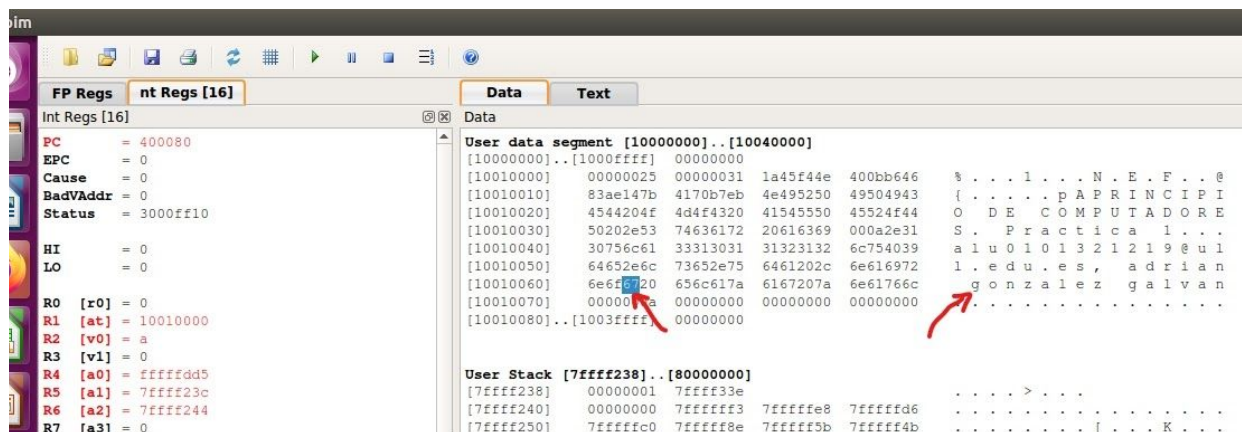
I. ¿Qué dirección de memoria (expresa la dirección en hexadecimal) ocupa el primer carácter de tu primer apellido (p.ej: en "alu999999999@ull.edu.es, Carlos Martin Galan\n" el carácter en cuestión es “M”)?



→ 0x10010061

→ Ya que “0x10010060” corresponde al carácter “espacio”.

II. ¿Qué carácter es y qué representación tiene en hexadecimal? Saca un pantallazo de User data Segment y marca con un cuadro rojo el byte correspondiente a ese carácter. El carácter es la “g” que como se puede apreciar en la imagen y según la tabla ascii se representa en hexadecimal como “67”:



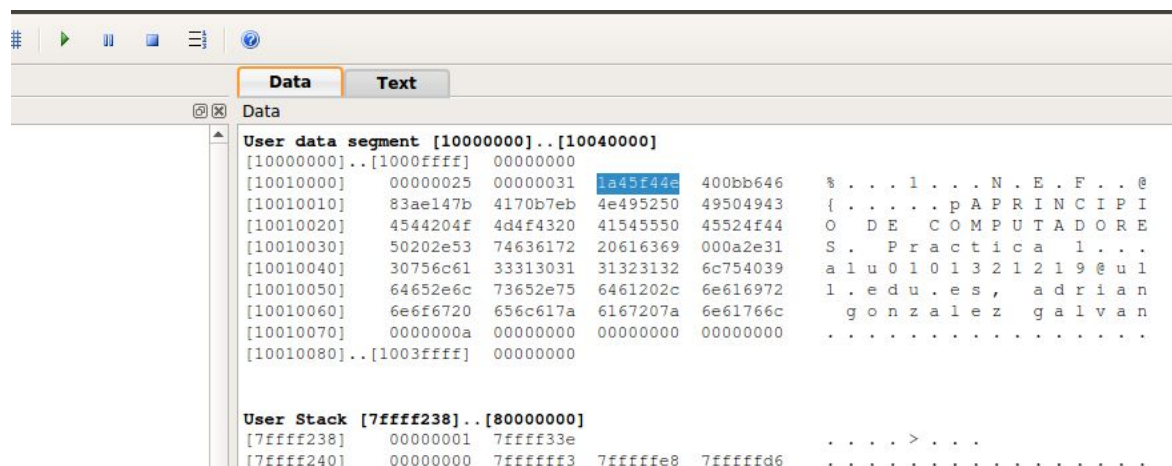
III. Recuerda que estás en hexadecimal. Busca en el segmento de datos de qtspim el número que se encuentra en la dirección etiquetada como num3. Saca un pantallazo y marca con un recuadro en rojo la palabra correspondiente.

```
# la directiva .data informa al compilador del comienzo de la definicion de datos

.data

.word 37
.word 49
.word 0x1A45F44E
.float 2.183
.double 17530552.23
.asciiz "PRINCIPIO DE COMPUTADORES. Practica 1.\n"
.asciiz "alu0101321219@ull.edu.es, adrian gonzalez galvan\n"
```

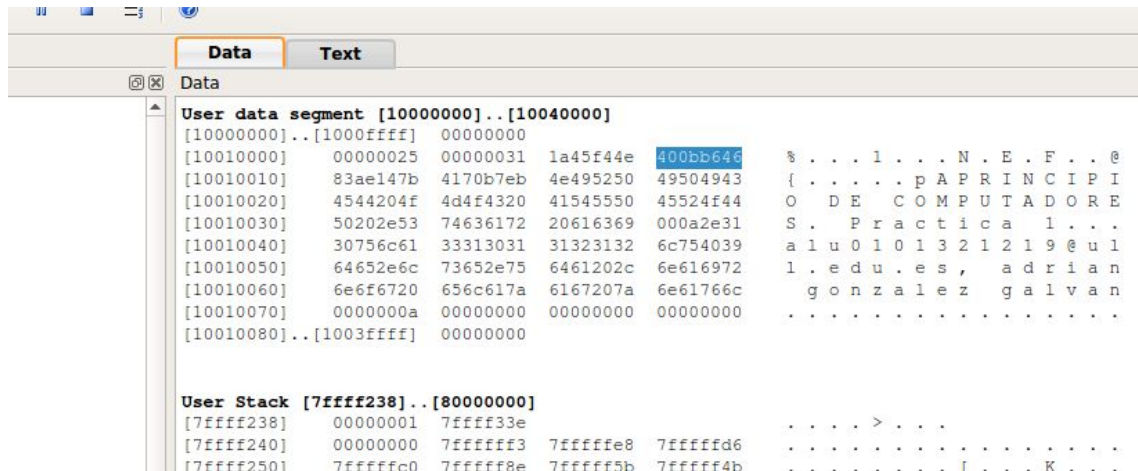
Vemos que la dirección etiquetada como num3 es el número en hexadecimal: “0x1A45F44E”



Este, como se ve, se encuentra en el segmento de datos como lo he marcado como “1a45f44e”.

IV. Convierte el número 2,183 a formato IEEE-754 para 32 bits (usa los apuntes del profesor o utiliza una calculadora online). Busca ahora este número en el segmento de datos, saca un pantallazo y márcalo con un cuadro en rojo.

2,183 = 0x400BB646, en el Qtspim lo podemos ubicar tal que:

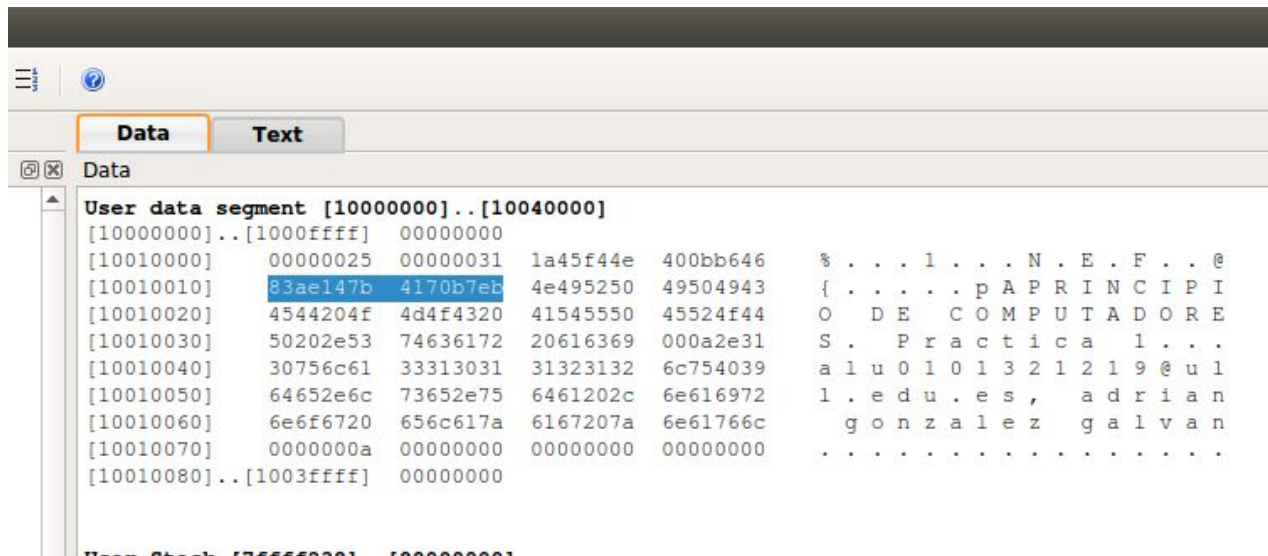


V. ¿En qué dirección empieza el número 2,183? expresa la dirección en hexadecimal.

- Si contamos vemos que está en la posición byte número 12
- $0x10000000 = "268435456"$ decimal
- $268435456 + 12 \text{ posiciones} = 268.435.468$
- $268.435.468 = 0x1000000c$ es su posición de dirección puesta en hexadecimal.

VI. Convierte el número 17530552.23 a formato IEEE-754 para 64 bits (usa los apuntes del profesor o utiliza una calculadora online). Busca ahora este número en el segmento de datos, saca un pantallazo y márcalo con un cuadro en rojo.

→ $17530552.23 = 0x4170B7EB83AE147B$



Como se puede observar el orden de los bytes del número “0x4170B7EB83AE147B” se encuentra ordenado de forma diferente. Esto se debe a que Qtspim reordena los bytes de 4 en 4 (dividiendo en 2 grupos de 4 bytes nuestro número) y por tanto no distingue un número tan grande como uno solo.

VII. ¿En qué dirección empieza el número 17530552.23? expresa la dirección en hexadecimal.

→ Si contamos vemos que está en la posición byte número 7.

→ $0x10010010 = “268501008”$ decimal

→ $268501008 + 7 \text{ posiciones} = 268501014$

→ $268501014 = 0x10010016$ es su posición de dirección puesta en hexadecimal.

- d) Reinicia la máquina y vuelve a cargar el programa en el QtSpim. Visualiza el banco de registros enteros y flotantes en hexadecimal. Menú Registers opción Hex). Recuerda, que las instrucciones de tu programa pueden ser convertidas en una o más instrucciones en QtSpim, por lo que tendrás que buscar la instrucción original de tu programa en los comentarios de la parte derecha.

I. Ejecuta paso a paso el programa hasta que hayas encontrado la instrucción `add $t2,$t0,$t1`. Una vez se haya ejecutado saca un pantallazo del banco de registros enteros y pon un cuadro rojo sobre el registro `$t2`. ¿Qué valor contiene? ¿sabrías expresarlo en decimal?

→ El registro `$t2` contiene el valor en hexadecimal “0x56” que es la suma de lo que contiene el registro `$t0` más lo que contiene el registro `$t1`.

→ $0x56 = 86$ en decimal.

FP Regs	Int Regs [16]	Data	Text
PC	= 400058		
EPC	= 0		
Cause	= 0		
BadVAddr	= 0		
Status	= 3000ff10		
HI	= 0		
LO	= 0		
R0 [r0]	= 0		
R1 [at]	= 10010000		
R2 [v0]	= 4		
R3 [v1]	= 0		
R4 [a0]	= 10010040		
R5 [a1]	= 7ffff23c		
R6 [a2]	= 7ffff244		
R7 [a3]	= 0		
R8 [t0]	= 25		
R9 [t1]	= 24		
R10 [t2]	= 56		
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		

Address	Hex	Assembly	Comment
00400000	8fa40000	lw \$4, 0(\$29)	; 18
00400004	27a50004	addiu \$5, \$29, 4	; 18
00400008	24a60004	addiu \$6, \$5, 4	; 18
0040000c	00041080	sll \$2, \$4, 2	; 18
00400010	00c23021	addu \$6, \$6, \$2	; 18
00400014	0c100009	jal 0x00400024 [main]	; 18
00400018	00000000	nop	; 18
0040001c	3402000a	ori \$2, \$0, 10	; 19
00400020	0000000c	syscall	; 19
00400024	3c011001	lui \$1, 4097 [titulo]	; 12
00400028	34240018	ori \$4, \$1, 24 [titulo]	; 13
0040002c	34020004	ori \$2, \$0, 4	; 13
00400030	0000000c	syscall	; 14
00400034	3c011001	lui \$1, 4097 [alumno]	; 17
00400038	34240040	ori \$4, \$1, 64 [alumno]	; 18
0040003c	34020004	ori \$2, \$0, 4	; 18
00400040	0000000c	syscall	; 19
00400044	3c011001	lui \$1, 4097 [num1]	; 21
00400048	8c280000	lw \$8, 0(\$1) [num1]	; 22
0040004c	3c011001	lui \$1, 4097 [num2]	; 22
00400050	8c290004	lw \$9, 4(\$1) [num2]	; 23
00400054	01095020	add \$10, \$8, \$9	; 23
00400058	014b6020	add \$t2, \$t0, \$t1	; 25
0040005c	340e022b	ori \$14, \$0, 555	; 29
00400060	11c00004	beq \$14, \$0, 16 [fin_buclewhi]	; 30
00400064	216bffff	addi \$11, \$11, -1	; 31
00400068	21ceffff	addi \$14, \$14, -1	; 32
0040006c	0401fffd	bgez \$0 -12 [buclewhile-0x004]	; 33
00400070	000b2021	addu \$4, \$0, \$11	; 39

II. Cuando hayas terminado de ejecutar esta instrucción, modifica a mano el valor del registro \$t3 (pulsas con el botón derecho del ratón sobre el registro correspondiente en el banco de registro y selecciona “Change Register Contents”, allí puedes seleccionar el formato y el valor). Deberás introducir un valor 10000 en formato decimal. Una vez lo hayas hecho saca un pantallazo y marca con un cuadro en rojo el registro correspondiente.

The screenshot shows the MIPS simulator interface. On the left, the 'Int Regs [16]' window displays the state of the integer registers. Register R11 [t3] is highlighted with a red box and contains the value 2710. The other registers are mostly zero, except for R1 [at] which is 10010000 and R5 [a1] which is 7ffff23c. On the right, the 'Text' window shows the assembly code for the 'User Text Segment'. The code includes instructions for loading values into registers, performing arithmetic operations, and branching. The instruction at address 00400058, 'add \$t2, \$t0, \$t1', is highlighted in blue.

FP Regs	Int Regs [16]	Data	Text
PC = 400058	PC = 400058		
EPC = 0	EPC = 0		
Cause = 0	Cause = 0		
BadVAddr = 0	BadVAddr = 0		
Status = 3000fff10	Status = 3000fff10		
HI = 0	HI = 0		
LO = 0	LO = 0		
R0 [r0] = 0	R0 [r0] = 0		
R1 [at] = 10010000	R1 [at] = 10010000		
R2 [v0] = 4	R2 [v0] = 4		
R3 [v1] = 0	R3 [v1] = 0		
R4 [a0] = 10010040	R4 [a0] = 10010040		
R5 [a1] = 7ffff23c	R5 [a1] = 7ffff23c		
R6 [a2] = 7ffff244	R6 [a2] = 7ffff244		
R7 [a3] = 0	R7 [a3] = 0		
R8 [t0] = 25	R8 [t0] = 25		
R9 [t1] = 31	R9 [t1] = 31		
	R11 [t3] = 2710		
R13 [t5] = 0	R13 [t5] = 0		
R14 [t6] = 0	R14 [t6] = 0		
R15 [t7] = 0	R15 [t7] = 0		
R16 [s0] = 0	R16 [s0] = 0		
R17 [s1] = 0	R17 [s1] = 0		
R18 [s2] = 0	R18 [s2] = 0		
R19 [s3] = 0	R19 [s3] = 0		
R20 [s4] = 0	R20 [s4] = 0		
R21 [s5] = 0	R21 [s5] = 0		

```

[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4 # arg1
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4 # en
[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
[00400024] 3c011001 lui $1, 4097 [titulo] ; 12: la $a0,titulo
[00400028] 34240018 ori $4, $1, 24 [titulo] ;
[0040002c] 34020004 ori $2, $0, 4 ; 13: li $v0,4
[00400030] 0000000c syscall ; 14: syscall
[00400034] 3c011001 lui $1, 4097 [alumno] ; 17: la $a0,alumno
[00400038] 34240040 ori $4, $1, 64 [alumno] ;
[0040003c] 34020004 ori $2, $0, 4 ; 18: li $v0,4
[00400040] 0000000c syscall ; 19: syscall
[00400044] 3c011001 lui $1, 4097 [num1] ; 21: lw $t0,num1 # carga e
[00400048] 8c280000 lw $8, 0($1) [num1] ;
[0040004c] 3c011001 lui $1, 4097 [num2] ; 22: lw $t1,num2 # carga e
[00400050] 8c290004 lw $9, 4($1) [num2] ;
[00400054] 01095020 add $10, $8, $9 ; 23: add $t2,$t0,$t1 # rea
[00400058] 014b6020 add $12, $10, $11 ; 25: add $t4,$t2,$t3 # r
[0040005c] 340e022b ori $14, $0, 555 ; 29: li $t6,555
[00400060] 11c00004 beq $14, $0, 16 [fin_buclewhile-0x00400060]
[00400064] 216bffff addi $11, $11, -1 ; 31: addi $t3,-1
[00400068] 21ceffff addi $14, $14, -1 ; 32: addi $t6,-1
[0040006c] 0401ffff bgez $0 -12 [buclewhile-0x0040006c]
[00400070] 000b2021 addu $4, $0, $11 ; 39: move $a0,$t3
[00400074] 34020001 ori $2, $0, 1 ; 40: li $v0,1

```

→ “10.000” decimal = 0x2710

III. A continuación sigue ejecutando paso a paso hasta terminar de ejecutar la instrucción `add $t4,$t2,$t3`. ¿Qué valor tiene el registro `$t4` en hexadecimal? ¿y en decimal?

The screenshot displays a MIPS simulator interface. On the left, the 'Int Regs [16]' window shows the state of the integer registers. Register `R12 [t4]` is highlighted with a red box and contains the value `2766`. A red arrow points to this register. The right window shows the 'Text' (code) section with the following instructions:

```
[00400000] 8fa40000 lw $4, 0($29) ; 183: lw $a0 0($29)
[00400004] 27a50004 addiu $5, $29, 4 ; 184: addiu $a1, $29, 4
[00400008] 24a60004 addiu $6, $5, 4 ; 185: addiu $a2, $5, 4
[0040000c] 00041080 sll $2, $4, 2 ; 186: sll $v0, $4, 2
[00400010] 00c23021 addu $6, $6, $2 ; 187: addu $a2, $6, $2
[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 #
[00400024] 3c011001 lui $1, 4097 [titulo] ; 12: la $a0, titulo
[00400028] 34240018 ori $4, $1, 24 [titulo] ; 13: li $v0, 4
[0040002c] 34020004 ori $2, $0, 4 ; 14: syscall
[00400030] 0000000c syscall ; 14: syscall
[00400034] 3c011001 lui $1, 4097 [alumno] ; 17: la $a0, alumno
[00400038] 34240040 ori $4, $1, 64 [alumno] ; 18: li $v0, 4
[0040003c] 34020004 ori $2, $0, 4 ; 19: syscall
[00400040] 0000000c syscall ; 19: syscall
[00400044] 3c011001 lui $1, 4097 [num1] ; 21: lw $t0, num1
[00400048] 8c280000 lw $8, 0($1) [num1] ; 22: lw $t1, num2
[0040004c] 3c011001 lui $1, 4097 [num2] ; 22: lw $t1, num2
[00400050] 8c290004 lw $9, 4($1) [num2] ; 23: add $t2, $t0
[00400054] 01095020 add $10, $8, $9 ; 25: add $t4, $t2, $t3
[00400058] 014b6020 add $12, $10, $11 ; 29: li $t6, 555
[0040005c] 340e022b ori $14, $0, 555 ; 29: li $t6, 555
[00400060] 11c00004 beq $14, $0, 16 [fin_buclewhile-0x00400060]
```

El registro `$t4` tiene un valor de `0x2766` = “10086” en decimal.

IV. A continuación establece un punto de ruptura “breakpoint” sobre la instrucción `move $a0,$t3` (sobre la instrucción correspondiente, pulsa en el botón derecho del ratón y selecciona “Set Breakpoint”). Después ejecuta todo el código (no paso a paso) y observarás que la ejecución se para en esta instrucción saltándose el bucle que hemos puesto. En este punto. ¿Qué valor tiene \$t3 (expresado en hexadecimal y también en decimal)? ¿y qué valor tiene \$t6?

The screenshot shows a debugger interface with two main panes. The left pane, titled 'Int Regs [16]', displays the values of various registers. The right pane, titled 'Text', shows the assembly code being executed.

In the 'Int Regs' pane, the following registers are highlighted with red boxes and red arrows:

- R11 [t3] = 24e5**: This register holds the value 24e5 in hexadecimal.
- R14 [t6] = 0**: This register holds the value 0 in hexadecimal.

The 'Text' pane shows the assembly code. The instruction `addu $4, $0, $t1` at address `000b2021` is highlighted in blue, corresponding to the `move $a0,$t3` instruction mentioned in the text. The code also includes a loop structure with `beq` and `bgez` instructions.

\$t3 tiene un valor de 0x24e5= “9445” decimal y \$t6 tiene un valor de 0.