COMPUTER FUNDAMENTALS
Practice 7
Data representation and introduction to SPIM

Second name	First name	DNI
Solution	Solution	
Solution	Solution	

Objective of the laboratory Session

Information's representation

The objective of this lab session is to reinforce the knowledge representation of data, both integer and real, as characters.

To achieve this goal, we will use the PCspim. In this session will work the two's complement operations, the actual representation and the representation of characters. Please, pay special attention to the log window and the data segment.

Introduction to the PCSpim simulator:

Start and configuration: The PCSpim simulator is executed from the direct link located in the programs menu.

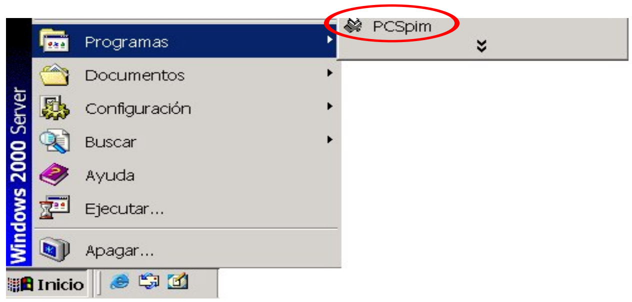


Figure 1. Execution of the simlulator.

When the simulator is executed by the first time, the following pop-up window is shown:

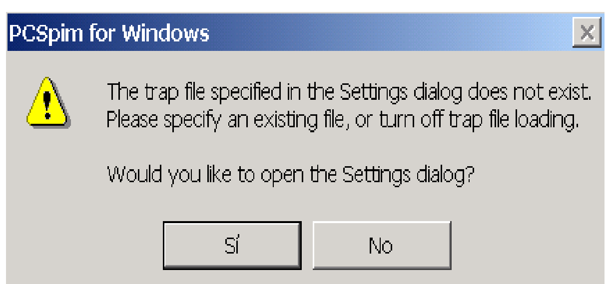


Figure 2. Pop-up window which appeas the first time the simulator is executed.

Click over the “Si” button.

The configuration pop-up window will be shown. Select the options shown below.

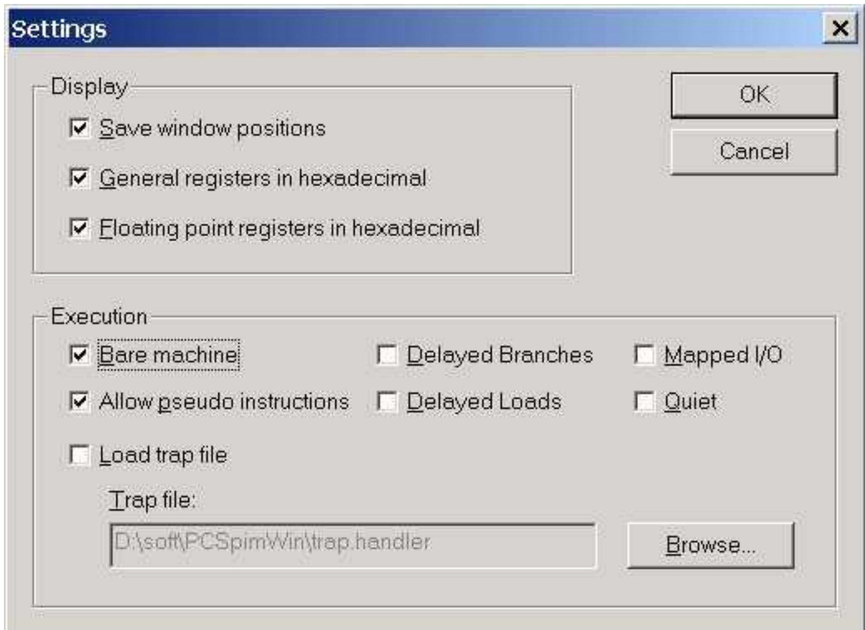


Figure 3. Recommended configuration.

Once the simulator is configured click over the “OK” button.

You should see the simulator shown below:

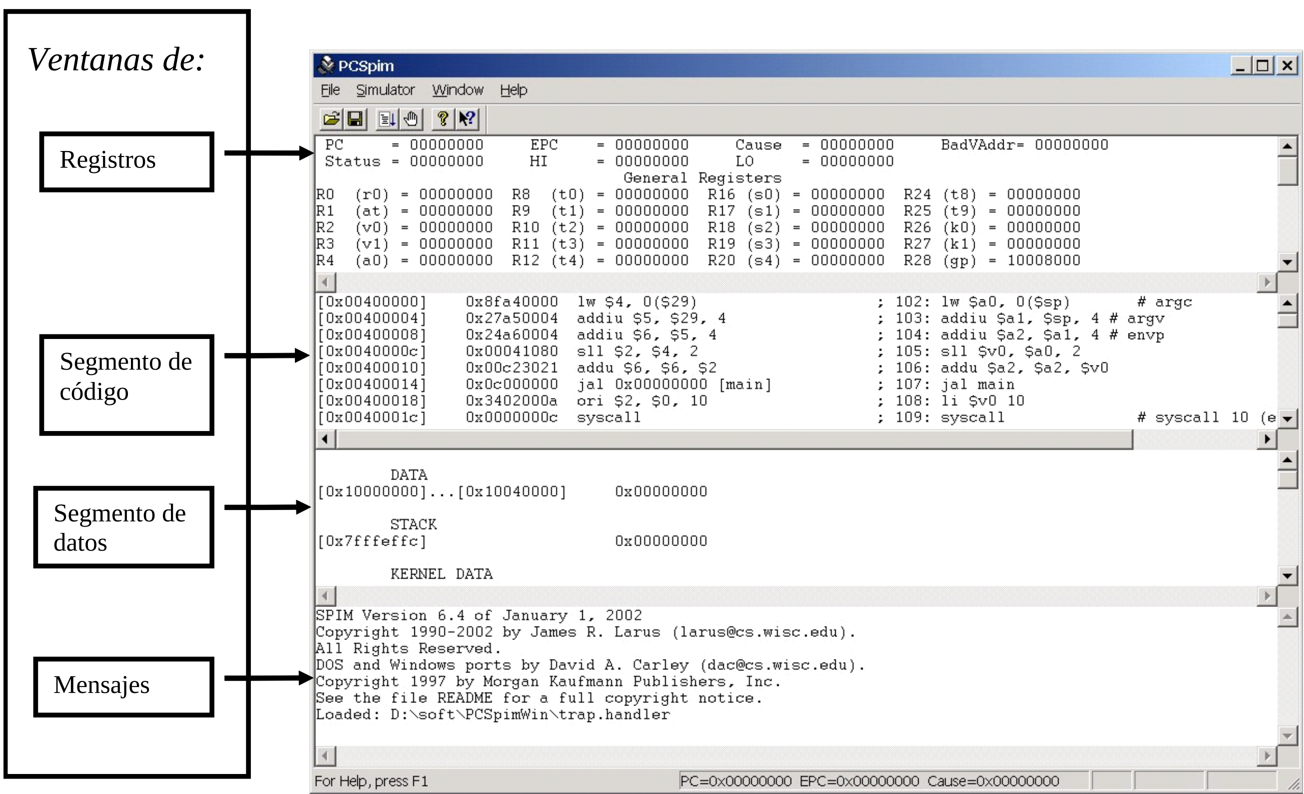


Figure 4. PCSpim Simulator.

Registers Window: Show the values stored on each one of the SPIM registers.

Code segment window: There are used 4 columns to show:

1. Address of each one of the instructions of the loaded program.
2. The coding of each one of the instructions of the loaded program. The coding is written in hexadecimal format.
3. The mnemotecnical codes of each instruction.
4. The instructions corresponding to the loaded program as they appear in the source code.

Data Segment data: Memory's content is shown on that window. Memory's content is display showing ranges of address directions and the content of such ranges. In other words, the following line:

[0x10000000]   0x12345678 0x22222222 0x3333333   0x44444444

Must be interpreted as:

5. The content of the 0x10000000 to 0x1000000C memory addresses are shown.
6. Memory address 0x10000000 has the hexadecimal value 0x12345678 (32 bits)
   - The content value can be interpreted as Bytes. Using the *little endian*: format, the values of each byte are:

     [0x10000000] 0x78
     [0x10000001] 0x56
     [0x10000002] 0x34
     [0x10000003] 0x12
7. Memory address 0x10000004 has the hexadecimal value 0x22222222 (32 bits)
8. Memory address 0x10000008 has the hexadecimal value 0x33333333 (32 bits)
9. Memory address 0x1000000C has the hexadecimal value 0x44444444 (32 bits)

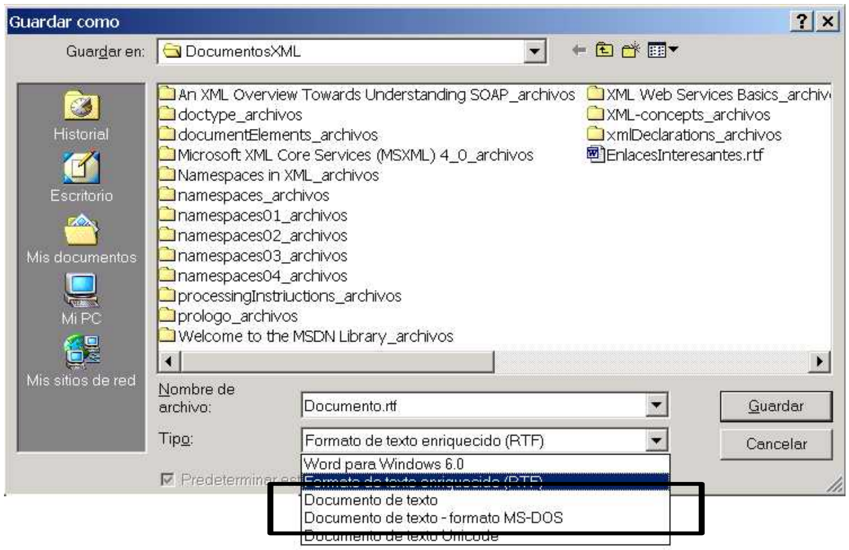
Program execution

The steps that you have to follow:

1. Write a program using MIPS's assembler language directives and instructions. The program should be stored in a file using a text editor (wordpad). File must be written in a file with format MS-DOS ASCII.
2. Load the program into the simulator (menu option File|Open)
3. Execute the program step by step (Key f10). Notice the evolution of the register values.

Important notes about program writing:

- The file must be written using ASCII format. If you use the program WORDPAD you have to store the file as a text document or as a text document in MS-DOS format



- You should change the file extension .txt by .s o .asm
- Remember that the predefined label __start have to be written using two underscores

Data directives

Data directives are indications for the assembler program (program to translate from assembler language to machine code). Such indications mare used when the machine code is generated. Data directives DO NOT GENERATES machine code. They do not generate real instructions. The meaning of the most used data directives are shown below:

Directive	Meaning
.data <addr>	Data is stored in data segment from the address indicated by the addr tag.
.byte $b_1,..,b_n$	*n* integer values are stored in successive bytes
.half $h_1,..,h_n$	*n* integer values are stored in successive addresses. Each value is stored using 16 bits.
.word $w_1,..,w_n$	*n* integer values are stored in successive addresses. Each value is stored using 32 bits.
.float $f_1,..,f_n$	*n* real values are stored in successive addresses. Each value is stored using 32 bits.
.ascii str	Stores each one of the characters of the string in memory. Each character is stored using one byte.
.asciiz str	Stores each one of the characters of the string in memory. Each character is stored using one byte. Finally it is stored an extra byte with the value 0x00 (NULL) after the last character.

Integers are stored using two's complement. Real Numbers are stored using the standard IEEE754 (single precision) data representation format. Characters are stored using 8 bits.

Data Alignment in Memory

The alignment of data in memory is a requirement of many processors. This alignment of data means that when the processor accesses a data of a certain type (byte, half or word), it will read the data according to its particular characteristics. For example, when the processor wants to read a word data type, the direction needs to be a multiple of four, as the words occupy four bytes. Instead, the addresses that refer to half data type is necessary that are multiples of two, while the byte type do not have any restrictions.

To maintain the relationship between the data type and address, the assembler program, loading the program and process the directives, will leave enough space between consecutive data.

Consider the following code and answer the following questions:

```
.data 0x10000000
.byte 1,-1,2
.half -2, 3
.byte 4
.word  -4
.byte 5
.half 6
```

1. **At home**. Fill the following table with the contents that have the memory to load and process the directives.

	Content (Decimal y hexadecimal)			
Adress (HEX)	31 ··· 24	23 ··· 16	15 ··· 8	7 ··· 0
0x1000000	---	0x01	0xff	0x02
0x1000004	0xff	0x03	0x00	0xfe
0x1000008	---	---	---	0x04
0x100000C	0xff	0xff	0xff	0xfc
0x1000010	0x00	0x06	---	0x05
0x1000014	---	---	---	---

Note: --- means that content is unknown.

2. **In the lab**, write a file and load it into PCspim. Fill the table below, this time copying the data as displayed in the window of the data segment

	Content (Decimal y hexadecimal)			
Adress (HEX)	31 ··· 24	23 ··· 16	15 ··· 8	7 ··· 0
0x1000000	0x00	0x01	0xff	0x02
0x1000004	0xff	0x03	0x00	0xfe
0x1000008	0x00	0x00	0x00	0x04
0x100000C	0xff	0xff	0xff	0xfc
0x1000010	0x00	0x06	0x00	0x05
0x1000014	0x00	0x00	0x00	0x00

3. Mark the appropriate sentence:
   - I did not prepare table at home.
   - The two tables do not match (the difference is that PCSPIM starts at zero all contents of the memory addresses)
   - The two tables are equal.
   - I do not understand we are talking about.

Real numbers representation

Real numbers are represented in most computers using the standard IEEE754 of single precision. In this format, the following criteria are used to represent a real number:

- The format is:

1 bit	8 bits	23 bits
Sign	Exponent	Mantissa

- The sign bit of zero means that the number represented is positive. And a one, to say that is negative.
- The exponent is represented in excess to Z, with Z = 127, This means that if the exponent of the number we want to represent is 3, the stored value is actually 130.
- The mantissa is required to be standardized as 1.xxx, and stored using the technique of leading implicit bit. That is, the first 1 is not stored.

Enter the code using an editor, save the program in a file. Load the file in PCspim, and answer the following questions:

```
.data 0x10000000
.float 1.0, -1.0, 3.23, -3.23
```

4. **(0.5 points)** Fill the following table with the contents of the memory displayed in the window of the data segment.

	Content (Decimal y hexadecimal)			
Address (HEX)	31 ··· 24	23 ··· 16	15 ··· 8	7 ··· 0
0x1000000	**0x3f**	**0x80**	**0x00**	**0x00**
0x1000004	**0xbf**	**0x80**	**0x00**	**0x00**
0x1000008	**0x40**	**0x4e**	**0xb8**	**0x52**
0x100000C	**0xc0**	**0x4e**	**0xb8**	**0x52**
0x1000010	**---**	**---**	**---**	**---**

4. Indicate the fields of sign, exponent and mantissa (binary three) of the four real values defined in the code above:

	Sign	Exponent	Mantissa
1,00	**0**	**01111111**	**00000000000000000000000**
-1,00	**1**	**01111111**	**00000000000000000000000**
3,23	**0**	**10000000**	**10011101011100001010010**
-3.23	**1**	**10000000**	**10011101011100001010010**

Can check the results in http://www.zator.com/Cpp/E2_2_4a1.htm/

Operations with integer Represented in 2's Complement

Enter the code in an editor. Load the file in PCspim and answer the following questions by observing the log window:

```
            .text 0x00400000
            .globl __start
__start:
            addi $2, $0, 1
            addi $3, $0, -2
            add  $4, $2, $3
            .end #end of program
```

5. Fill the following table with the content displayed in the log window after loading the program before running it.

	Value in hexadecimal (with all bits)	Decimal value
$2	**0x00000000**	**0**
$3	**0x00000000**	**0**
$4	**0x00000000**	**0**

6 Using the option GO from the PCSpim simulator menu execute the program. When the whole program had been executed fill the following table with the register's values.

	Value in hexadecimal (with all bits)	Decimal value
$2	**0x00000001**	**1**
$3	**0xfffffffe**	**-2**
$4	**0xffffffff**	**-1**

7 What is the function that performs the line addi $2, $0, 1?

```
Adds the number (immediate data) 1 to the contents of the register
zero (which is always set to 0) and stores the result in the register
two (named as $2)
```

8 What is the function that performs the line addi $3, $0, -2?

```
Adds the number (immediate data) -2 to the contents of the register
zero (which is always set to 0) and stores the result in the register
three (named as $3)
```

9 **(0.5 points)** What is the function that performs the line add $ 4, $ 2, $ 3?

```
Adds the content of the register three (named as $3) to the contents
of the register two (named as $2) and stores the result in the
register four (named as $4)
```

10 (0.5 points) the result of the line add $ 4, $ 2, $ 3 is correct?

```
Yes, because the result of the operation +1 + (-2) is -1, and the
representation of -1 in two's complement is 0xffffffff (using 32 bits
to store the result)
```

Enter the code using an editor, store it in a file. Load the file in PCspim and answer the following questions by observing the log window:

```
        .text 0x00400000
        .globl __start
__start:
        lui $2, 0x7fff
        lui $3, 0x7fff
        add $4,$2,$3
        .end #end of program
```

11. Fulfill the following table with the content displayed in the log window after loading the program before running it.

	Value in hexadecimal	Decimal value
$2	**0x00000000**	**0**
$3	**0x00000000**	**0**
$4	**0x00000000**	**0**

12. Now execute the program using the menu option *SIMULATOR GO*. After the execution, fill the following table with the new contents of the registers.

	Value in hexadecimal (with all bits)	Decimal Value
$2	**0x7fff0000**	**2147418112**
$3	**0x7fff0000**	**2147418112**
$4	**0x00000000**	**0**

13. What is the function that performs the line lui $ 2, 0x7FFF?

Load the immediate data (0x7fff) into the two upper bytes of the destination register ($2)

14. Is it right to the result of the line add $ 4, $ 2, $ 3?

Not, because the result can’t be represented into the available bytes of the registry (four bytes, thirty three bits) represented in two’s complement.

Input the following code

Enter the code using an editor, store it in a file. Load the file in PCspim and answer the following questions by observing the log window:

```
                .text 0x00400000
                .globl __start
    __start:
                addi $2, 10000
                mult $2, $2
                mflo $3
                mult $2, $3
                .end #end of program
```

Execute the program step by step until the execution of the mult instruction (mult $2, $2)

15. What is content of the registers HI and LO?

	Hexadecimal value	Decimal value
HI	0x00000000	0
LO	0x05f5e100	100000000

Execute the program step by step until the end instruction

16. What is content of the registers HI and LO?

	Hexadecimal value	Decimal value
HI	0x000000e8	232
LO	0xd4a51000	-727379968

16. What does the mflo instruction make?

MFLO means "move from LO" and loads into the specified registry the value of LO.

17. Justify the obtained final result, indicating in decimal the operation made.

_______10000_________ X_____1000_____X___100000000___=__1000000000000___

Character representation

Enter the code using an editor, save the program in a file. Load the file in PCspim and answer the following questions:

```
.data 0x10000000
.asciiz "el profe"
.byte 16
.ascii "el profe"
.byte 16
```

11 Fill the following table with the contents of the memory displayed in the window of the data segment.

	Content (Decimal and hexadecimal)			
Address (HEX)	31 ··· 24	23 ··· 16	15 ··· 8	7 ··· 0
0x1000000	**0x70**	**0x20**	**0x6c**	**0x65**
0x1000004	**0x65**	**0x66**	**0x6f**	**0x72**
0x1000008	**0x6c**	**0x65**	**0x10**	**0x00**
0x100000C	**0x6f**	**0x72**	**0x70**	**0x20**
0x1000010	**0x00**	**0x10**	**0x65**	**0x66**

12 In what memory addresses are the letters "p"?

Address in hexadecimal:
The letter 'p' is codified in ascii as "0x70" (in hexadecimal). So that the letter p is located at the addresses: 0x10000003 and 0x1000000D

13 ¿What is the difference between the directive .asciiz and the directive .ascii?

The directive asciiz completes the string with the 0x00 character (also known as NULL character)

Standard ASCII table

	0x00	0x10	0x20	0x30	0x40	0x50	0x60	0x70
+0	*NUL*	*DLE*	*SP*	0	@	P	`	p
+1	*SOH*	*DC1*	!	1	A	Q	a	q
+2	*STX*	*DC2*	"	2	B	R	b	r
+3	*ETX*	*DC3*	#	3	C	S	c	s
+4	*EOT*	*DC4*	$	4	D	T	d	t
+5	*ENQ*	*NAK*	%	5	E	U	e	u
+6	*ACK*	*SYN*	&	6	F	V	f	v
+7	*BEL*	*ETB*	'	7	G	W	g	w
+8	*BS*	*CAN*	(	8	H	X	h	x
+9	*HT*	*EM*	)	9	I	Y	i	y
+A	*LF*	*SUB*	*	:	J	Z	j	z
+B	*VT*	*ESC*	+	;	K	[	k	{
+C	*FF*	*FS*	,	<	L	\	l	|
+D	*CR*	*GS*	-	=	M	]	m	}
+E	*S0*	*RS*	.	>	N	^	n	~
+F	*S1*	*US*	/	?	O	_	o	*DEL*