Career: Engineering in Computer Science
Teacher: Juan Manuel Tirado Martín
Subject: Computer Structure

November 11, 2012

MIPS assembler and introduction to PCSpim

Adrián María Mora Carreto NIA: 100291775
Ramses Joel Salas Machado NIA: 100292149

MIPS assembler and introduction to PCSpim

Index

Exercise 2

a) Analyze the code and indicate what it is doing. Indicate what is the printed value after running the execution

What we have in this code is that given an array and its size we add two by two values and then print them.

In the .data we declare an array where the numbers are stored in one word each, n that is also another number and the space in order to print numbers separately.

Now, in the .text we begin to execute the program, firstly with the main method. First of all, it loads in every temporal register a value: in $t5 stores n and in $t7 2, in the rest 0 is loaded. Then it divides n and 2 and loads the quotient into $s0 and the remainder into $s1. The quotient is the number of values that we have at the end of the addition. Then we evaluate if $s1 is zero: if it is zero, it goes to label2, if not we add to $s0 1.

Now we are in label2 and we modify the value of $v0 with 9 and $t7 with 4, in order to multiply the quotient of the previous division with $t7 and allocate a memory with the number of bytes in $a0 (that is where the result of the multiplication is stored) and the memory address of the allocation is in $v0. Then we copy this memory address into $s1 and $s2 and we load the value of array($t1) (that is now the first value of the array) into $t4.

When the code enters in label1 it first checks if $t0 and $t5 are greater or equal, if it is true then it jumps to label3. If not it goes through this code: first, it loads the value of array($t1) (that is the data in the position $t1 on the array) into $t4, then adds $t3 and $t4 storing the result in $t3. Then, the value of $t3 is stored in the direction previously copied into $s1. Now, we add to $t4 and $t0 4 and 1 respectively and we check if now $t0 is equal to $t5, as previously. If not, it continues with the previous code but at the end it sets $t3 to 0 and adds $s1 4 and jump to the beginning of label1.
What the code do now is to add two values from the array and store them into memory, then add 4 to the memory address (in order to store the number and not overwrite it) and start again to add the two values.

When the code jump into label3, the register $t0 is set into zero and then goes into label4.

In label4, firstly it checks if $t0 and $s0 are equal, and if it is true the program goes to the end. If not, the program prints all the results of the addition.

When the end is reached, it jumps to $ra that is the exit of the program.

b) Indicate what are registers $t0 and $t1 used for. What is the difference between them? What is the reason for using both of them?

$t0 is used to count the number of additions that are done, because for each value it is done one addition, even if $t3 is equal to 0. $t1 in this case is used to get the position of the number in the array, and it is added 4 instead of 1 because we store a number in a word, that is 4 bytes.

c) Given the code below. Explain what the code does and why it is used.

```
li $v0, 9
li $t7, 4
mul $a0, $t7, $s0
syscall
move $s1, $v0
```

What we do in this part of the code is allocate some bytes in order to store in the memory the addition of two values of the array, and put the memory address of the allocation in $v0, so then we copy its value into $s1.

d) Complete the value of the registers before the execution of the next instruction in the table below:

```
bgeu $t0, $s0, end
```

ITERATION / REGISTER	1	2	3	4	5	6
R8 ($t0)	0	1	2	3	4	5
R16 ($s0)	5	5	5	5	5	5
R18($s2)	10040000	10040004	10040008	1004000C	10040010	10040014
PC	004000d4	004000d4	004000d4	004000d4	004000d4	004000d4

Description of the implemented methods

Exercise 3

In this exercise we declare in the .data section a string. We create a function called *strlen* which will return the length of the declared string. Basically the program check char by char until it find the null character (‘ \0’ 0 value in hexadecimal) adding 1 to a counter each time we pick a character different from null.

Exercise 4

In this exercise we also declare in the .data section a string. We implement a function called *toLower* to change a string from uppercase to lowercase. The function checks character by character if it is an uppercase letter, if it is then we proceed to change it to its homonymous lowercase until we reach the null symbol of the string.

Exercise 5

In this exercise we declare and initialize in the .data section an array, its correspondent number of elements and a string with a bar in order to separate the results when we print. At the beginning we take the first value of the array and we start to compare it with the rest of the values in the array storing the greatest in each one of the iteration. At the same time, we use a counter to know the position of each value in the array. Finally, we print the greatest value and its position in the console.

Exercise 6

In this exercise we declare and initialize in the .data section two strings, one to be search in the other. We implement the function *isInside* in order to check if the second string is inside the first, to do this we take the first character in the string to be search and look for this character in the other string, if we find it we compare the character of both strings until we reach the end of the string to be search; the program will end if we reach the null symbol in the that we are searching returning -1 or if we both string match returning 1.

Exercise 7

In this exercise we declare and initialize in the .data section a string where we want to substitute “%i” with a number declared in a array. We impement a subMain method were we initialize the string and the array of numbers and we load the first character of the string. Then in the forCheck method we search in the string if we have found ‘%’ or the end of the string. If the conditions are not fulfilled, then we print the character and go to the next one. If the end of the string is reached, then the program finish. If ‘%’ is found then we check if the next character is ‘i’: if this is true, then we print the number corresponding to the number of times that “%i” have appeared; if this is false, then the “%” is printed and continues searching in the string

Done tests and results

In each exercise we have made at least 3 tests in order to ensure the correct operation of them. The most difficult problem for us was Exercise 7 in where we have to do a lot of tests because of the problems of the implementations of the code. Now, we can see that for every exercise we have developed a general coding where most of the cases are taking into account without encountering any problem.

Conclusion

This practice has been very useful in order to understand PCSpim simulator works and how MIPS architecture is structured. After this assignment we are able to read basic assembly codes and programing in this language using the PCSpim simulator to test our codes.