Lab #2 – Branch statements and arrays in MIPS assembly

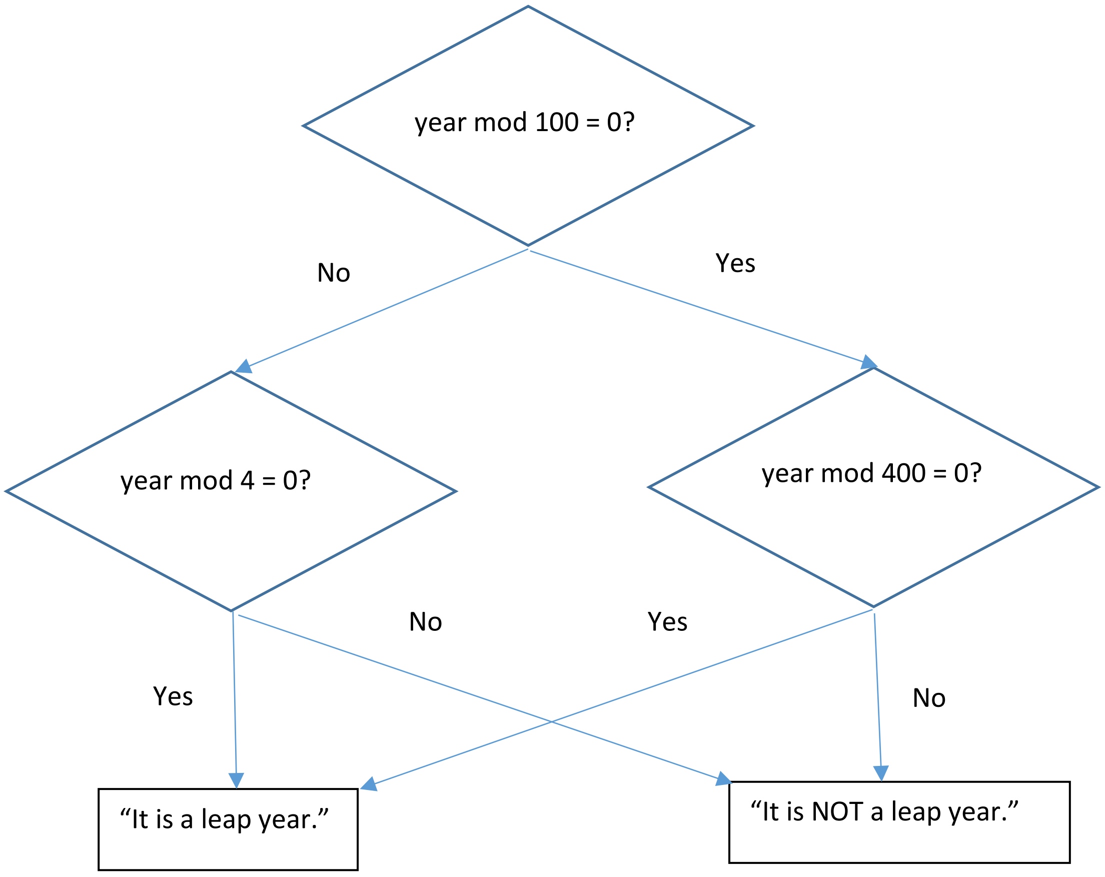
This lab features three assembly programs for you to modify. The first one focuses on "if-then" situations, the second is about loops, and the third one practices one-dimensional arrays.

First, log in to the computer as usual. You may want to insert a USB drive to organize all of your lab work. The software we will use to compose and run MIPS assembly programs is called MARS (MIPS assembler and runtime simulator). You can download it from the class Web site or from the original URL (https://courses.missouristate.edu/KenVollmar/mars/download.htm). On the class Web site you should see a folder called lab2 containing files you will need for today's lab. It would be a good idea for you to create a lab2 folder for yourself and do all of today's work in there. Have fun!

Part 1: Making comparisons

Double-click on the MARS icon to launch it. Let it take up the entire screen. From the File menu, select Open. Navigate the file hierarchy to select leap.s. This is an interactive program to determine leap years. At the moment, the program is incomplete. It only performs the basic I/O, but no calculations. You need to complete the implementation.

The following flow chart shows the modern "Gregorian" way to determine leap years.



Based on the intended logic of the program, write the necessary calculation and branch instructions. You are free to arrange the code as you see fit. Recent example programs we saw in class might give you a good idea for how your solution should be structured.

Note: In the MIPS language, the instruction to check remainders is called "rem", not "mod". For example, rem \$t1, \$t2, \$t3 will compute (\$t2 mod \$t3) and place the result in \$t1. If this result equals zero, then you can conclude that \$t2 is evenly divisible by \$t3.

Once you have typed in your program, you will need to assemble, and then run the program. Assembling is analogous to compiling – you will be warned of any syntax errors that you need to correct. The command to assemble is Run → Assemble. To run, the command is Run → Go.

Each time you run the program, you need to issue the command Run → Reset before running the program again. This will bring the simulator's registers back to their initial state.

Because there are four possible execution paths through the above flow chart, you should test your program with four different year input values. Which ones did you try?

______________ , ______________ , ______________ , ______________

Please have your program checked by the instructor or lab aide before continuing. √

Part 2: Using loops

Once we understand how to write branch and jump instructions, it is rather straightforward to write loops. All that we need to do is branch or jump *backwards* in the code in order to repeat something.

1. Open the program 1to10.s. This is the same program we saw in class, which prints the numbers 1 through 10 on the screen. Assemble and run it to verify it works correctly.

2. We can modify the program to make it more useful. Copy this program into a new file called range.s. Enhance this program so that instead of always printing 1-10, it does the following:
   a. Asks the user to enter two integers.
   b. Prints all integers in the range from the first integer up to the second. But if the first number is greater than the second, it will print nothing and quit.

   For example, if the user enters 4 and 7, then the program should print

   ```
   4
   5
   ```

6
7

and then quit.

3. Finally, how would you change your program so that it will count backwards in case the first number is bigger than the second? In other words, the program should count from the first number to the second number no matter which direction it happens to be. (4, 7) gives 4, 5, 6, 7 and (7, 4) gives 7, 6, 5, 4. Give it a try. √

Part 3: Max and min of an array

The array is everyone's favorite data structure. Today we experiment how to access array elements in the MIPS assembly language. Recall that when we have an array of integers, the instructions we commonly use to access individual elements are

- lw – load word, for copying a number from an array cell to a register
- sw – store word, for copying a number from a register into the array

In this section, we perform a useful array operation: finding the maximum and minimum elements.

1. The program max.s finds the maximum element in an array. Open this program in MARS. Assemble and run it, and observe what happens. You should be able to recognize the loop in the source code, and the comparison on each iteration.

2. Copy max.s into a new file called maxmin.s, which you will enhance. Open maxmin.s in MARS.

3. The program should output "The largest number in the array is 48". It would be nice if we also had a period at the end of the sentence. Add the code to accomplish this.

4. Now, a more interesting modification. Suppose that we were not just interested in the value of the largest element, but where in the array this largest element resides. It turns out that the value 48 is at position 5 (which is the 6th element) in the array. If this were a HLL program, we'd say array[5] is 48.

   Change the program so that it will also keep track of the index value where the largest entry exists. This may entail using another register, such as $s6. And at the end of the program, print this index out as well.

5. If we can find the max, then we also know how to find the min. Adapt the max program so that it also performs the analogous process of finding the smallest element of the array, and

also its position within the array. You can test your program by changing how the element values are initialized, such as turning –39 into positive 39 so that the smallest number will now be –23. √

After all three of your programs have been checked, you're done! ☺ Close the MARS application, and log out. Don't forget to take your USB drive with you.

Learning a new language is a lot of information to digest. Please don't hesitate to ask me questions if you ever have difficulty figuring out what to do.