Week 3 Recitation

January 25th, 2024

Important Stuff

Lab 1 - Due 1/28

Lab 2 - Due 2/4

Lab 2: Input, Variables, & Control Flow

Lab 2: Input, Variables, & Control Flow

- For this lab, we are building a simple calculator using MIPS
- The calculator will have 7 operations:
  - Add
  - Subtract
  - Multiply
  - Divide
  - = (Displays a given input number)
  - c (clears the display)
  - q (quit)

The Java Equivalent

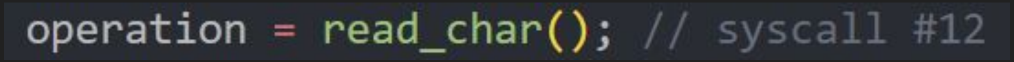
- https://canvas.pitt.edu/courses/241470/files/15429467?wrap=1
- Also inside lab 2

So, how do we do this in MIPS?

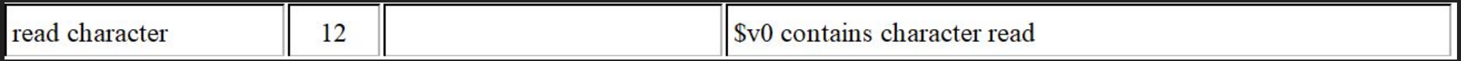
- According to the Java code, we need a scanner, a switch case, print statements…
- Thankfully, this can all be done in MIPS (fairly) easily!

Getting an Input w/ MIPS

- When a program requests an input, a system call is done to obtain the input
  - Even Scanners in Java (when compiled down to assembly) uses system calls to do this
- So, we just need to use the proper system call
- In this case, we can set v0 = 12 to get user input



```
operation = read_char(); // syscall #12
```



read character	12		$v0 contains character read

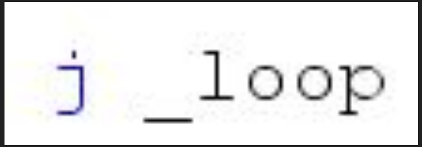
MIPS Execution

- We want our calculator to run indefinitely, until we use the quit command
- MIPS programs execute sequentially (line-by-line), so we need to find a way to move our execution around the program
  - Like a car with a brick on the gas pedal
- We can take advantage of this execution with control flow!

Control Flow

Control Flow: Jump Instructions

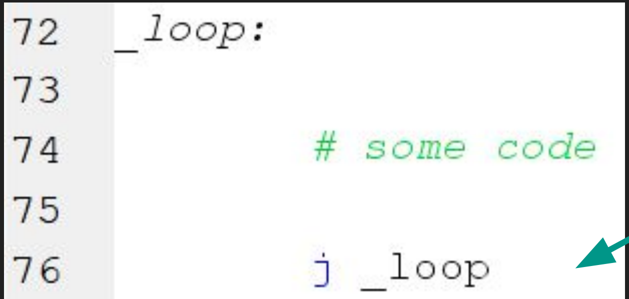
- We need to way to control the next line a MIPS instruction executes
- To change the order a MIPS program executes, we can use a *j (jump)* instruction
- Jumps move the current line of execution to whatever the destination of the jump is
  - But… what is _*loop*?



```
j _loop
```

Control Flow: Labels

- Labels allow us to name different sections of our program
- By jumping to different labels, we can execute different portions of our program at different times!
- In the last example, *_loop* is a label for the jump instruction to jump to
    - We use underscores to denotes these
    - Use this stylesheet from Prof. Billingsley !

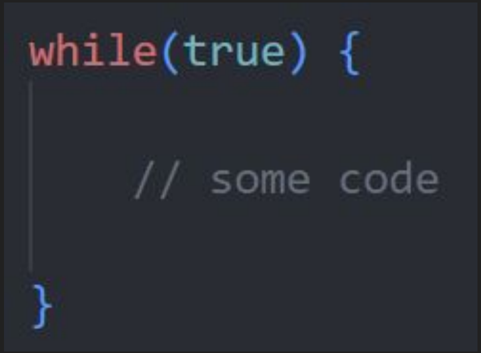


```
72  _loop:
73
74          # some code
75
76          j _loop
```

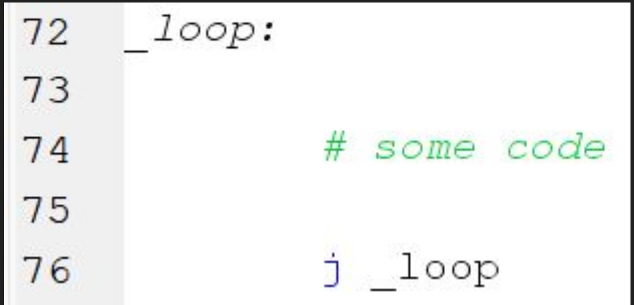
j _loop moves execution from line 76 back to line 72

Control Flow Example 1

- The following Java and MIPS code are essentially equivalent
- This jump allows us to have an infinite loop
  - Think of *j _loop* as equivalent to reaching the bottom *}* of the Java while loop

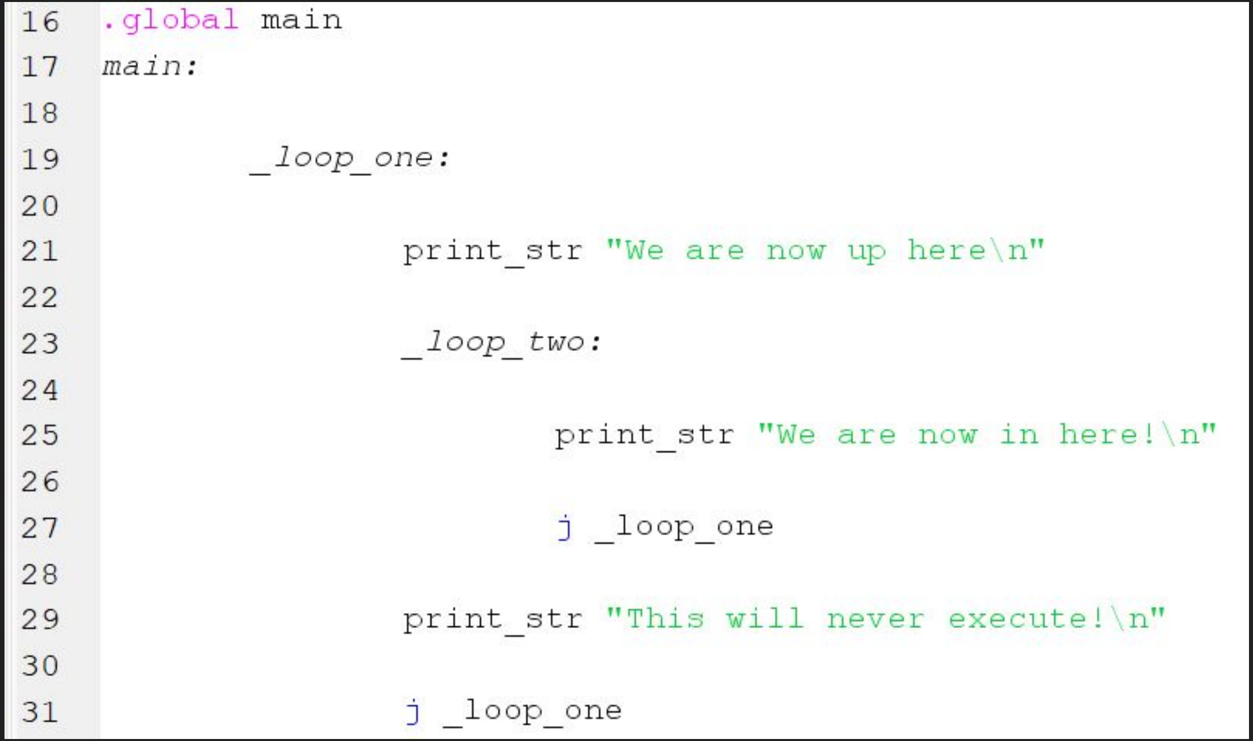


```
while(true) {

    // some code

}
```



```
_loop:

        # some code

        j _loop
```

Control Flow Example 2



```
16  .global main
17  main:
18
19          _loop_one:
20
21                  print_str "We are now up here\n"
22
23                  _loop_two:
24
25                          print_str "We are now in here!\n"
26
27                          j _loop_one
28
29                  print_str "This will never execute!\n"
30
31                  j _loop_one
```

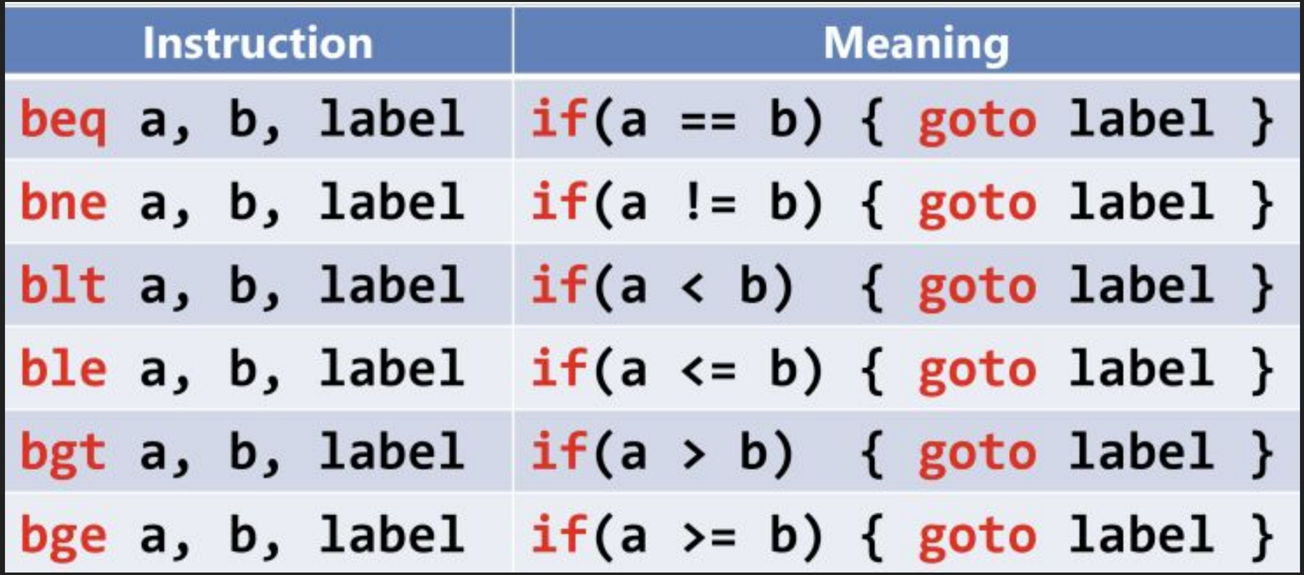
- In the following code, we have two labels
- Because of jump back to _loop_one inside of _loop_two, the print on line 29 never executes

Now, what about the switch statement?

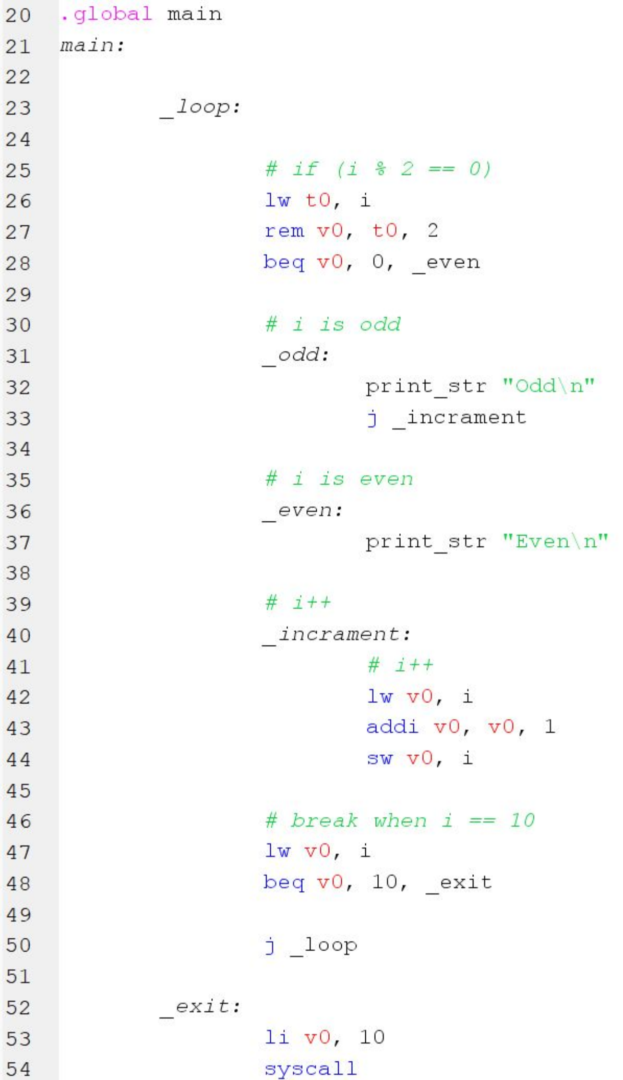
- With our jumps and our labels, we can start to see how a switch statement would work
- However, how we are missing one key component: comparisons
  - How will our calculator know if the user input is ‘q’ or ‘+’

Control Flow: MIPS Conditional Instructions

- Conditional instructions are jump instructions that only jump when a condition is met
- The ‘b’ in these stands for branch (“branch equal”, “branch greater than“, …)



Instruction	Meaning
beq a, b, label	if(a == b) { goto label }
bne a, b, label	if(a != b) { goto label }
blt a, b, label	if(a < b) { goto label }
ble a, b, label	if(a <= b) { goto label }
bgt a, b, label	if(a > b) { goto label }
bge a, b, label	if(a >= b) { goto label }



```
20 .global main
21 main:
22
23         _loop:
24
25                 # if (i % 2 == 0)
26                 lw t0, i
27                 rem v0, t0, 2
28                 beq v0, 0, _even
29
30                 # i is odd
31                 _odd:
32                         print_str "Odd\n"
33                         j _incrament
34
35                 # i is even
36                 _even:
37                         print_str "Even\n"
38
39                 # i++
40                 _incrament:
41                         # i++
42                         lw v0, i
43                         addi v0, v0, 1
44                         sw v0, i
45
46                 # break when i == 10
47                 lw v0, i
48                 beq v0, 10, _exit
49
50                 j _loop
51
52         _exit:
53                 li v0, 10
54                 syscall
```

Control Flow Example 3

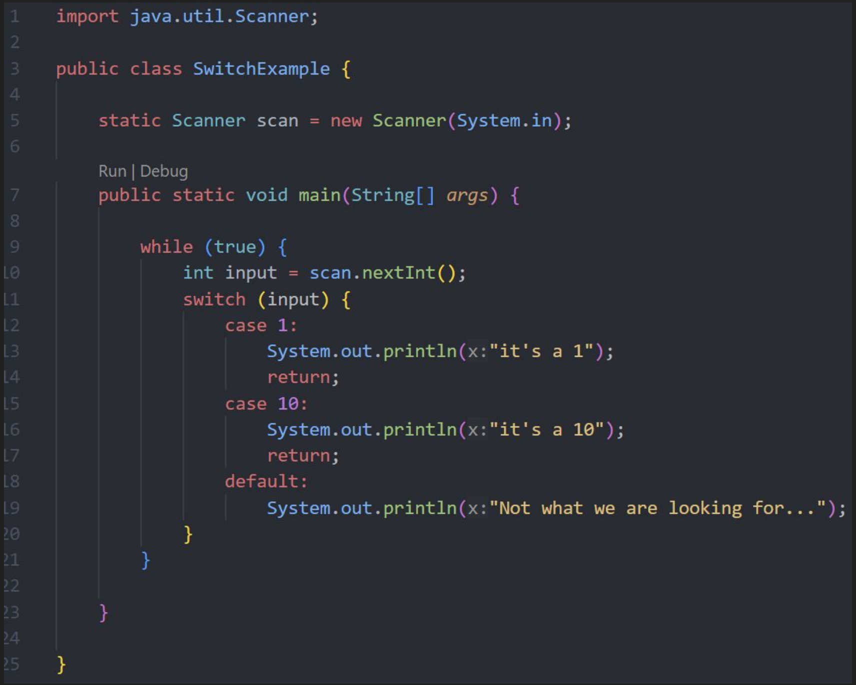
- This example is a simple loop, where while i < 10, we print if i is currently odd or even
- *rem v0, t0, 2* is equivalent to v0 = t0 % 2
- *beq v0, 0, _even* jumps our program to _even (line 36) if the v0 is equal to 0
    - Otherwise, we enter the _odd: label (though, this label is technically not needed because of the nature of this program!)

Putting it all together

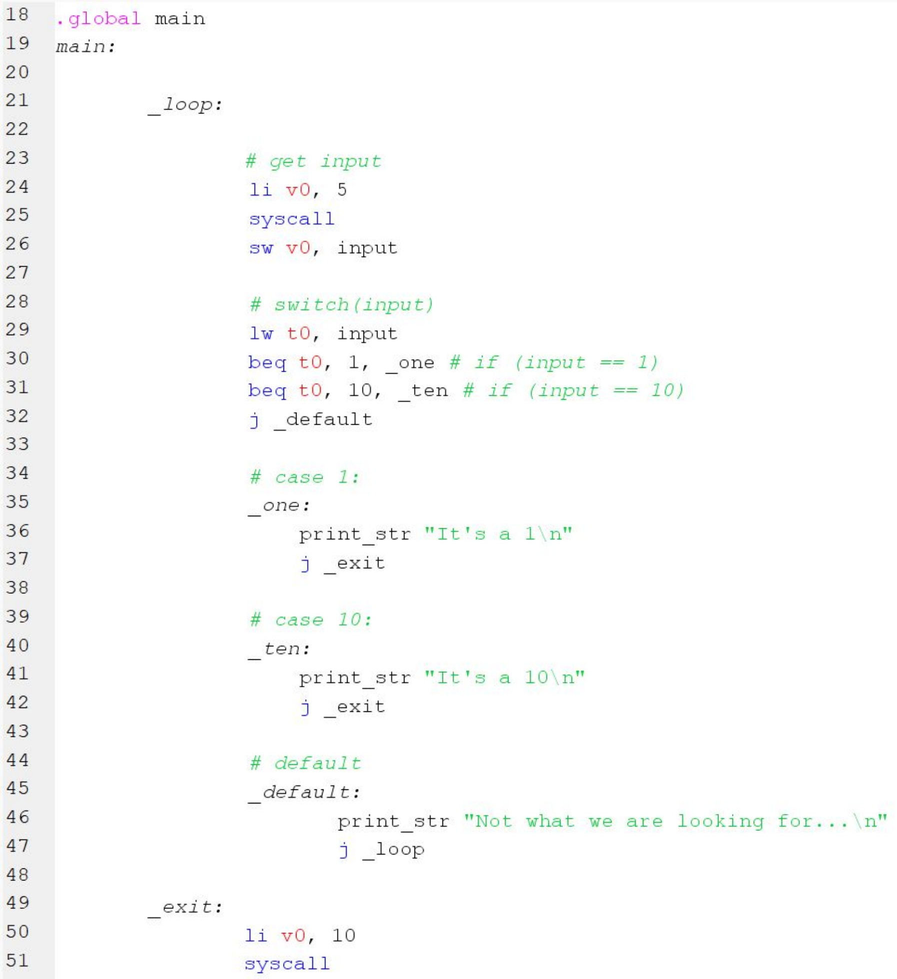
- Now we have all the pieces to make a switch statement in MIPS!
- We can check which case the input is with variables, then jump to the respective label!

Control Flow Example 4

- Java
- MIPS



```java
import java.util.Scanner;

public class SwitchExample {

    static Scanner scan = new Scanner(System.in);

    public static void main(String[] args) {

        while (true) {
            int input = scan.nextInt();
            switch (input) {
                case 1:
                    System.out.println(x:"it's a 1");
                    return;
                case 10:
                    System.out.println(x:"it's a 10");
                    return;
                default:
                    System.out.println(x:"Not what we are looking for...");
            }
        }

    }

}
```



```
.global main
main:

        _loop:

                # get input
                li v0, 5
                syscall
                sw v0, input

                # switch(input)
                lw t0, input
                beq t0, 1, _one # if (input == 1)
                beq t0, 10, _ten # if (input == 10)
                j _default

                # case 1:
                _one:
                    print_str "It's a 1\n"
                    j _exit

                # case 10:
                _ten:
                    print_str "It's a 10\n"
                    j _exit

                # default
                _default:
                       print_str "Not what we are looking for...\n"
                       j _loop

        _exit:
                li v0, 10
                syscall
```

Now it’s your turn!

- You can view the Java solution for the lab (found in the Lab 2 canvas assignment)
- Don’t forget to copy-paste the *print_str* macro provided in the lab!
- Please use the styling guidelines that can be found here
- Let me know if you have any questions!
- All example code is available on the GitHub