CS/COE 0447

Control Flow, Conditions, and Loops. Oh my.

wilkie (with content borrowed from:

Jarrett Billingsley

Dr. Bruce Childers)

Introduction to Conditions

- Programs don't do much if they are simply a list of instructions.
- We want to introduce **interactivity** to our code.
- Our programs should make **decisions**.
  - If this happens, do this. Otherwise, do that.

- The possible decisions make up the *potential* **control flow** of the program.
  - When there is no possible route to a piece of code in your program, that is called **dead code**. Spoooooky!
    - This would be a great Halloween costume, right?

Control Flow

Two roads diverged in a wood, and I—
I took the one less traveled by,
And that has made all the difference.

— Robert Frost

Control Flow

- **Control flow** is the order that your instructions run in
  - What kinds of control flow statements do you know of?
  - What about functions?
- WELLLLLLLLLLLLLLLLLLLLLLLLLLLL
- In asm, the only thing you get for free is that instructions run in order.
- **You're responsible for coming up with everything else.**
  - If you screw up your control flow, the CPU doesn't care.
  - You'll just have a broken, malfunctioning program.
    - And it'll be half an hour before the lab is due
      - And you'll be sad
        - This is like 90% of the bugs I've seen in previous semesters.

How to Approach Writing Asm (Reprise)

- First and foremost: WRITE PSEUDOCODE!

```
if(x == w - 1) {

} else {

}
```

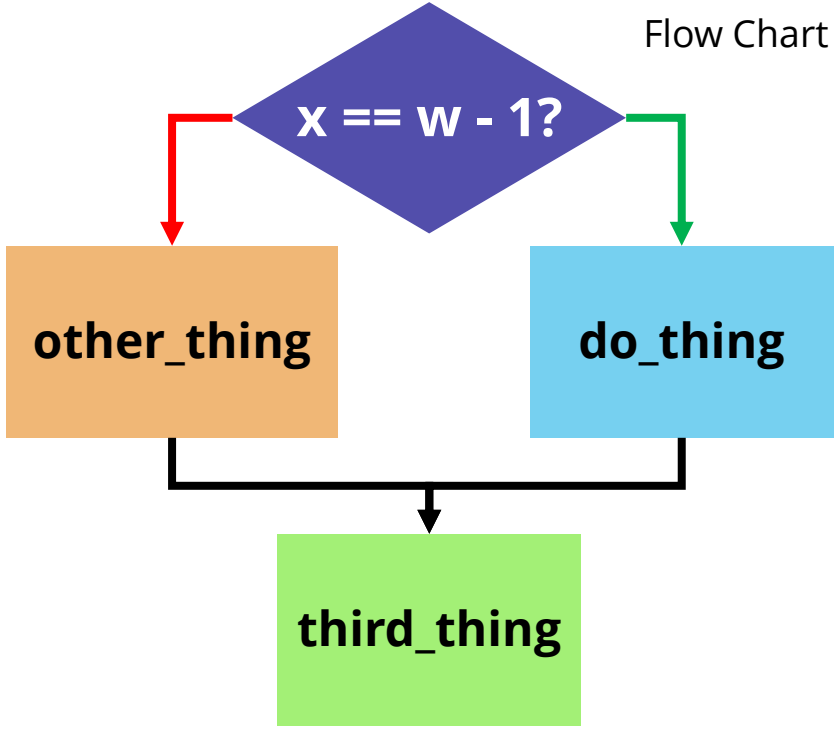
IGNORE THE CRAP INSIDE

TRANSLATE THE CONTROL FLOW TO ASM *FIRSTTTTTTTTT*

The Building Blocks

- A **basic block** is a chunk of code that has no control flow in it.
- Control flow statements separate basic blocks:

```
if(x == w - 1) {
    do_thing()
} else {
    other_thing()
}
third_thing()
```



thinking about this is REAL HELPFUL

To sum up:

- The way control flow works in asm is **you make basic blocks**
  - You gotta name (label) them
- Then, you use special instructions to choose where to go.
  - “Which basic block runs next?"
  - *That's* the question you need to ask yourself for this stuff.
  - We will look at these instructions next.
- *Use pseudo-code (with comments) to keep track of control flow.*
- *Or: use a flow-chart (pen + paper method) to design your program.*

Loops

These come in many forms: infinite, finite, and fruit.

Introducing Branches

- All control flow is done with **branches** and **jumps**.
  - These are instructions which say "go somewhere else“
- For example…

```
main_loop:
    # clear screen
    # draw one thing
    # sleep
    # draw another thing
    # etc
    b main_loop
```

this is an infinite loop, which is sometimes useful but not too interesting

b stands for "branch" – go somewhere else

MIPS ISA: The conditional branch

- **conditional** branch instructions do one of two things:
  - if the condition is met, we go to the label
  - otherwise, **nothing happens,** and we go to the **next instruction**

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 }`

above, **a** must be a register, but **b** can be a register or immediate

The Simple Conditional Loop

- something like this…

what are the basic blocks?

which conditional branch?

```
while(s2 < 10)
{
  // stuff!!
}
// more stuff
```

```
loop_top:
blt s2, 10, stuff
b more_stuff
stuff:
  # stuff!!
b loop_top
more_stuff:
  # more stuff
```

if s2<10, where do we go?

if s2≥10, where do we go?

how to go back up to the top?

we **NEED THIS** – the CPU doesn't see/care about your labels!!

Conditionals: If and If-Else

If it is sunny outside, then we will have a picnic.
If it is not sunny outside, well, who knows.
Maybe we will. Maybe we won't.
— Professor Yasir Khalifa

Like mad-libs... but for code.

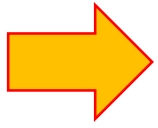
- I'll use these 'blocks' to represent the basic blocks
  - Cause they don't matter. (Focus on the control flow first!)

```
if(some condition) {
        block A
} else {
        block B
}
    block C
```

Simple Conditional Block (if)

- If there is no *else*, it's pretty simple.

```
if(s0 == 30) {
      block A
}
   block B
```



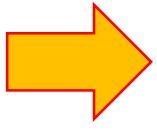
```
   beq s0, 30, blockA
   b blockB
blockA:
      block A
blockB:
      block B
```

coming up with unique names for all the control flow labels is kind of a chore

Simple Conditional Expanded (if-else)

- more blocks now…

```
if(s0 == 30) {
      block A
}
else {
      block B
}
   block C
```



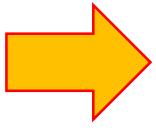
```
beq s0, 30, blockA
b blockB
blockA:
      block A
b blockC
blockB:
      block B
blockC:
      block C
```

we **NEED THIS** – the CPU doesn't see/care about your labels!!

Another Approach (negation)

- Right now, we are very literally transcribing our pseudo-code
  - That's because it is a **good** idea, and I recommend it, for now.
- However, we generally have a different approach.
  - Removes a label and some clutter and the expense of being *different.*
- We **negate** the condition in the assembly to skip over "blockA"
  - Thus we don't need that label. It's a preference. (I don't recommend it until some practice)

```
if(s0 == 30) {
        block A
}
else {
        block B
}
    block C
```



```
bne s0, 30, blockElse
        block A
b blockExit # skip the else
blockElse:
        block B
blockExit:
        block C
```

Complex Conditionals

If it is sunny outside AND we have free time, then we will have a picnic.

Consider This Code... (if X || Y, if X && Y)

```
if(dog_size < 10 || dog_name() == "Fluffy")
```

If dog_size is 3, **is dog_name() called?**

NO!

This is **short circuit evaluation.**

For || (logical OR), if the first condition is true, **the second one is skipped.** (cause there's no way for the result of the OR to be false.)

For && (logical AND), if the first condition is *false*, **the second one is skipped.**

Consider This Code… (if-else-if-else)

```
if(dog_size < 10)
    small();
else if(dog_size < 20)
    medium();
else if(dog_size < 30)
    large();
else
    enormous();
```

if dog_size is 3, **is this condition checked?**

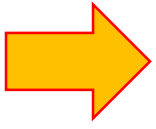
NO!

once a true condition is found, **no more conditions are checked.** after small(), it comes down here.

The && Logical And Operator

- block A is run if ***both*** **conditions are true.**
  - To think of it another way... it's *skipped* if either condition is *false.*

```
if(s0 == 30 && s1 > 1)
{
    block A
}
```

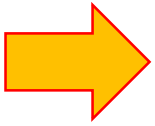


```
    bne s0, 30, skipA
    ble s1,  1, skipA
      block A
skipA:
```

The || Logical Or Operator

- We go to block A if ***either*** **condition is true.**

```
if(s0 == 30 || s1 > 1)
{
    block A
}
```



```
    beq s0, 30, blockA
    bgt s1,  1, blockA
    b skipA
blockA:
    block A
skipA:
```