MIPS control flow instructions:
Jumps, Branches, and Loops

Pick up handout, bring back on Friday
Quick Policy Reminder

Penalty for involvement in Plagiarism

- All parties involved will receive a 0 on that assignment or exam and their final course grade reduced by one letter grade (e.g., A->B, B->C, etc.). A second offense will result in a failing grade for the class.
Today’s lecture

- **Control Flow**
  - Programmatically updating the program counter (PC)

- **Jumps**
  - Unconditional control flow
  - How is it implemented?

- **Branches**
  - Loops
  - If/then/else
  - How implemented?
Sequential lines of code are executed by “incrementing” the Program Counter

0x00400004 → mul $14, $13, $20
      addi $14, $14, 4
      sub $15, $14, $15
      xor ← $12, $15, $8

Where is instruction XOR located?

a) 0x00400007  b) 0x00400008
   c) 0x00400010  d) 0x00400016
We use **control flow** in high level languages to implement conditionals and loops

Repetition via Loops
```c
for (int i = 0 ; i < N ; i ++) {
    sum += i;
}
```

Selective execution via Conditionals
```c
if (x < 0) {
    x = -x;
}
```

How do we implement these in MIPS assembly?
An **unconditional jump** always transfers control (like a goto statement in C)

- Use a “label” to tell where in the code to jump to:

  ```
  j target_label
  ```

- Example:

  ```
  Loop:    j Loop
  ```

- What does this code do? **Infinite**
Jumps use the J-type encoding

Where do the other 6 bits come from?
- Last two bits are always 00, because PC is word aligned
- 4 most significant bits come from existing PC value.

\[ \text{jump } PC[31:28] = PC[31:28] \]
Example encoding: The infinite loop

Loop: j Loop

PC = 0x00400024: j 0x00400024

0x00400024 0000 0000 0100 0000 0000 0000 0010 0100
Jump instructions can only move within 1 of 16 regions

- A 26-bit address field lets you jump to any address from 0 to $2^{28}$.
- your Lab solutions had better be smaller than 256MB
Implement Jump

What should wr_enable be?
- a) 0
- b) 1
- c) don’t care
Branches provide **conditional** control flow

```
beq rs, rt, target_label
```

- **Branch if EQual (BEQ):**
  - If $(R[rs] == R[rt])$, then branch to $target_label$
  - Otherwise execute next instruction $(PC+4)$

```
bne rs, rt, target_label
```

- **Branch if Not Equal (BNE):**
  - Branch when $(R[rs] != R[rt])$
Implement the C code in MIPS assembly

```c
int sum = 0;
int i = 0;
do {
    sum += i;
    i ++;
} while (i != 10)
```

Assembly:
```
add $10, $0, 10
add $2, $0, $0 # sum = 0
add $3, $0, 0 # i = 0
do:
    add $2, $2, $3
    add $3, $3, 1 # i++
    bne $3, $10, do
```

A) eq
B) ne
Implement the C code in MIPS assembly

```c
int sum = 0;
for (int i = 0; i != x; i++) {
    sum += i;
}
```

A) eq
B) ne

Assembly:
```
add $2, $0, $0
add $3, $0, $0
loop: b eq $3, $4, done
      add $2, $2, $3
      add $3, $3, 1
      j loop
done:
```
Implement the C code in MIPS assembly

if (x == 0) {
    x = 1;
}

Assembly:

```
bne $2,$0, skip
add $2,$0, 1  # x = 1
```

```
skip:
```

Hint: Sometimes it’s easier to invert the original condition.
Change “continue if x < 0” to “skip if x >= 0”.

A) eq
B) ne
The address in branch is an *offset* from PC+4 to the target address.

```
beq $1, $0, L
add $1, $3, $0
add $2, $3, $3
j Somewhere
```

```
L: add $2, $3, $3
```

What value should be stored in the address of the *beq* instruction?

- a) 1
- b) 2
- c) 3
- d) 4