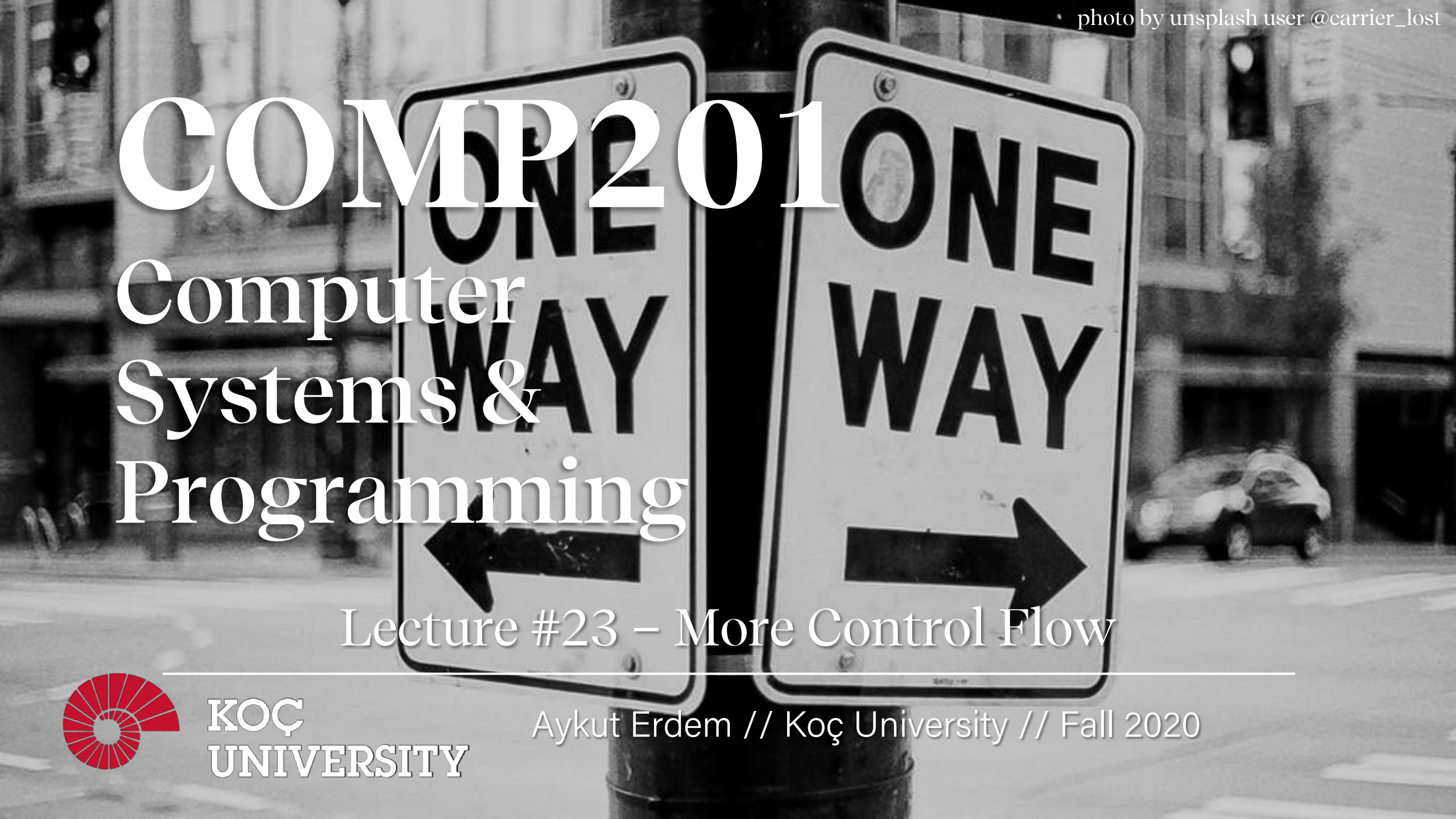

photo by unsplash user @carrier_lost

COMP201

Computer Systems & Programming

Lecture #23 – More Control Flow

KOÇ UNIVERSITY

Aykut Erdem // Koç University // Fall 2020

Good news, everyone!

- Midterm exam will be held on December 5, 2020 (Saturday) at 11:45am
- Check out the midterm guide for more information.

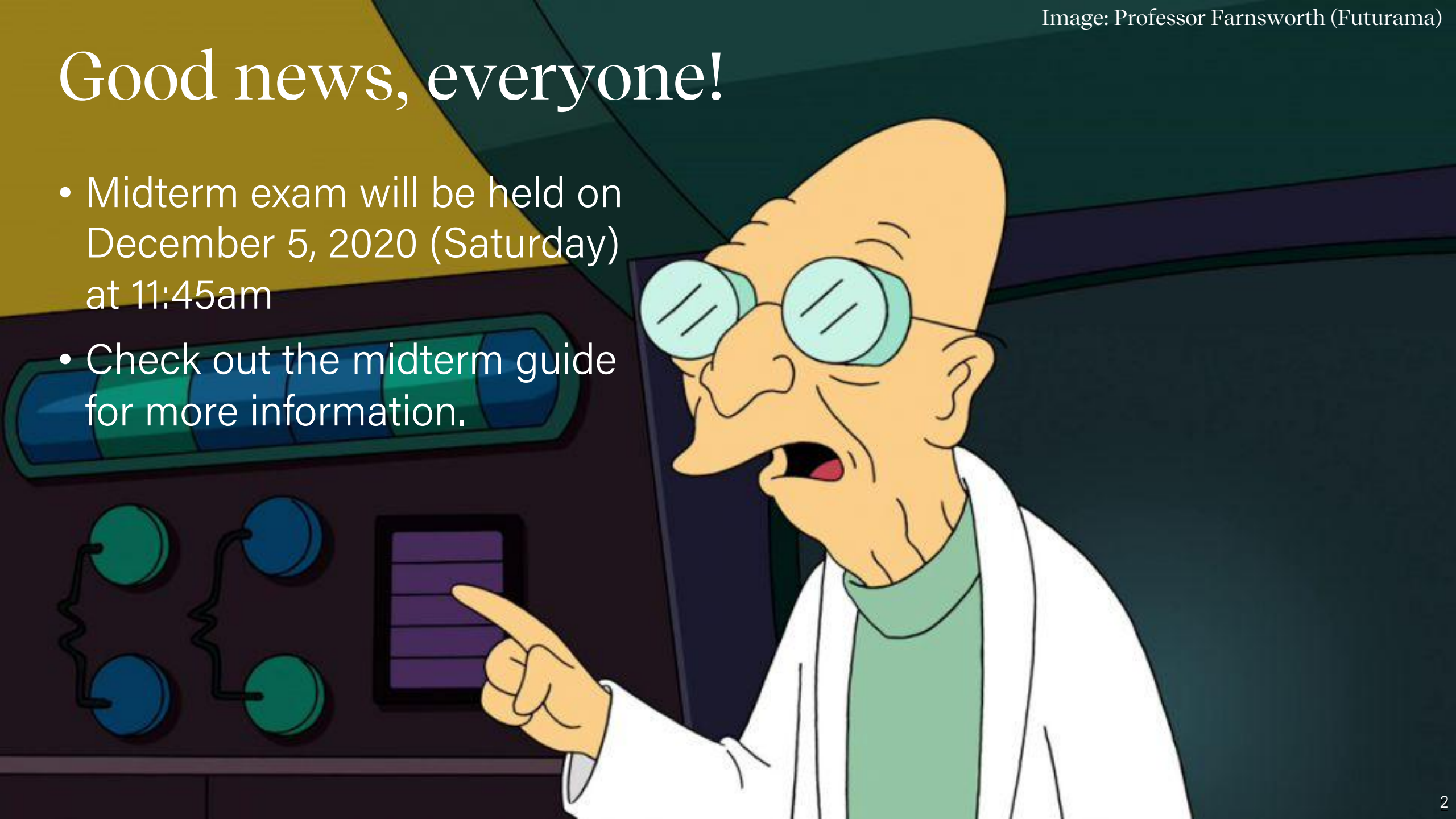

Image: Professor Farnsworth (Futurama)

Recap

- Control Flow Mechanics
  - Condition Codes
  - Assembly Instructions
- If statements

Plan for Today

- If statements (cont'd.)
- Loops
- Other Instructions That Depend On Condition Codes

Disclaimer: Slides for this lecture were borrowed from

—Nick Troccoli's Stanford CS107 class

Lecture Plan

- If statements (cont'd.)
- Loops
- Other Instructions That Depend On Condition Codes

Practice: Fill In The Blank

If-Else In C

```
if (arg > 3) {
    ret = 10;
} else {
    ret = 0;
}

ret++;
```

If-Else In Assembly pseudocode

```
Test
Jump to else-body if test fails
If-body
Jump to past else-body
Else-body
Past else body
```

Practice: Fill In The Blank

slido Event code: 73165

If-Else In C

```
if ( _________ ) {
    _________;
} else {
    _________;
}
_____;
```

```
400552 <+0>:  cmp  $0x3,%edi
400555 <+3>:  jle  0x40055e <if_else+12>
400557 <+5>:  mov  $0xa,%eax
40055c <+10>: jmp  0x400563 <if_else+17>
40055e <+12>: mov  $0x0,%eax
400563 <+17>: add  $0x1,%eax
```

If-Else In Assembly pseudocode

```
Test
Jump to else-body if test fails
If-body
Jump to past else-body
Else-body
Past else body
```

Practice: Fill In The Blank

slido Event code: 73165

If-Else In C

```
if ( arg > 3    ) {
    ret = 10;
} else {
    ret = 0;
}
ret++;
```

```
400552 <+0>:  cmp  $0x3,%edi
400555 <+3>:  jle  0x40055e <if_else+12>
400557 <+5>:  mov  $0xa,%eax
40055c <+10>: jmp  0x400563 <if_else+17>
40055e <+12>: mov  $0x0,%eax
400563 <+17>: add  $0x1,%eax
```

If-Else In Assembly pseudocode

```
Test
Jump to else-body if test fails
If-body
Jump to past else-body
Else-body
Past else body
```

Lecture Plan

- If statements (cont'd.)
- Loops
  - While loops
  - For loops
- Other Instructions That Depend On Condition Codes

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:    mov    $0x0,%eax
0x0000000000400575 <+5>:    jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:    add    $0x1,%eax
0x000000000040057a <+10>:   cmp    $0x63,%eax
0x000000000040057d <+13>:   jle    0x400577 <loop+7>
0x000000000040057f <+15>:   repz retq
```

Set `%eax` (`i`) to 0.

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

Jump to another instruction.

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

Compare `%eax` (`i`) to 0x63 (99) by calculating `%eax` – 0x63. This is 0 – 99 = -99, so it sets the Sign Flag to 1.

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

`jle` means “jump if less than or equal”. This jumps if **`%eax`** <= 0x63. The flags indicate this is true, so we jump.

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

Add 1 to %eax (i).

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

Compare `%eax` (`i`) to 0x63 (99) by calculating `%eax` – 0x63. This is 1 – 99 = -98, so it sets the Sign Flag to 1.

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

`jle` means "jump if less than or equal". This jumps if **`%eax`** <= 0x63. The flags indicate this is true, so we jump.

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

We continue in this pattern until we do not make this conditional jump. When will that be?

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

We will stop looping when this comparison says that **%eax** – 0x63 > 0!

Loops and Control Flow

```
void loop() {
    int i = 0;
    while (i < 100) {
        i++;
    }
}
```

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

Then, we return from the function.

Common While Loop Construction

C

```
while (test) {
      body
}
```

Assembly

```
Jump to test
Body
Test
Jump to body if success
```

From Previous Slide:

```
0x0000000000400570 <+0>:     mov    $0x0,%eax
0x0000000000400575 <+5>:     jmp    0x40057a <loop+10>
0x0000000000400577 <+7>:     add    $0x1,%eax
0x000000000040057a <+10>:    cmp    $0x63,%eax
0x000000000040057d <+13>:    jle    0x400577 <loop+7>
0x000000000040057f <+15>:    repz retq
```

Lecture Plan

- Loops
  - While loops
  - For loops
- Other Instructions That Depend On Condition Codes

Common While Loop Construction

C For loop

```
for (init; test; update) {
      body
}
```

C Equivalent While Loop

```
init
while(test) {
     body
     update
}
```

Assembly pseudocode

```
Init
Jump to test
Body
Update
Test
Jump to body if success
```

For loops and while loops are treated (essentially) the same when compiled down to assembly.

Back to Our First Assembly

```
int sum_array(int arr[], int nelems) {
    int sum = 0;
    for (int i = 0; i < nelems; i++) {
        sum += arr[i];
    }
    return sum;
}
```

1. Which register is C code's `sum`?
2. Which register is C code's `i`?
3. Which assembly instruction is C code's `sum += arr[i]`?
4. What are the **`cmp`** and **`jl`** instructions doing?
   (**`jl`**: jump less; signed <)

```
0000000004005b6 <sum_array>:
  4005b6:       mov    $0x0,%edx
  4005bb<+5>:   mov    $0x0,%eax
  4005c0<+10>:  jmp    4005cb <sum_array+21>
  4005c2<+12>:  movslq %edx,%rcx
  4005c5<+15>:  add    (%rdi,%rcx,4),%eax
  4005c8<+18>:  add    $0x1,%edx
  4005cb<+21>:  cmp    %esi,%edx
  4005cd<+23>:  jl     4005c2 <sum_array+12>
  4005cf<+25>:  repz retq
```

Lecture Plan

- Loops
- Other Instructions That Depend On Condition Codes

Condition Code-Dependent Instructions

There are three common instruction types that use condition codes:

- **`jmp`** instructions conditionally jump to a different next instruction
- **`set`** instructions conditionally set a byte to 0 or 1
- new versions of **`mov`** instructions conditionally move data

`set`: Read condition codes

`set` instructions conditionally set a byte to 0 or 1.

- Reads current state of flags
- Destination is a single-byte register (e.g., `%al`) or single-byte memory location
- Does not perturb other bytes of register
- Typically followed by `movzbl` to zero those bytes

```
int small(int x) {
    return x < 16;
}
```

```
    cmp $0xf,%edi
    setle %al
    movzbl %al, %eax
    retq
```

`set`: Read condition codes

Instruction	Synonym	Set Condition (1 if true, 0 if false)
`sete D`	`setz`	Equal / zero
`setne D`	`setnz`	Not equal / not zero
`sets D`		Negative
`setns D`		Nonnegative
`setg D`	`setnle`	Greater (signed >)
`setge D`	`setnl`	Greater or equal (signed >=)
`setl D`	`setnge`	Less (signed <)
`setle D`	`setng`	Less or equal (signed <=)
`seta D`	`setnbe`	Above (unsigned >)
`setae D`	`setnb`	Above or equal (unsigned >=)
`setb D`	`setnae`	Below (unsigned <)
`setbe D`	`setna`	Below or equal (unsigned <=)

`cmov`: Conditional move

`cmovx src,dst` conditionally moves data in `src` to data in `dst`.

- Mov `src` to `dst` if condition `x` holds; no change otherwise
- `src` is memory address/register, `dst` is register
- May be more efficient than branch (i.e., jump)
- Often seen with C ternary operator: `result = test ? then: else;`

```
int max(int x, int y) {
    return x > y ? x : y;
}
```

```
cmp     %edi,%esi
mov     %edi, %eax
cmovge  %esi, %eax
retq
```

Ternary Operator

The ternary operator is a shorthand for using if/else to evaluate to a value.

`condition ? expressionIfTrue : expressionIfFalse`

```
int x;
if (argc > 1) {
    x = 50;
} else {
    x = 0;
}

// equivalent to
int x = argc > 1 ? 50 : 0;
```

cmov: Conditional move

Instruction	Synonym	Move Condition
`cmove S,R`	`cmovz`	Equal / zero (ZF = 1)
`cmovne S,R`	`cmovnz`	Not equal / not zero (ZF = 0)
`cmovs S,R`		Negative (SF = 1)
`cmovns S,R`		Nonnegative (SF = 0)
`cmovg S,R`	`cmovnle`	Greater (signed >) (SF = 0 and SF = OF)
`cmovge S,R`	`cmovnl`	Greater or equal (signed >=) (SF = OF)
`cmovl S,R`	`cmovnge`	Less (signed <) (SF != OF)
`cmovle S,R`	`cmovng`	Less or equal (signed <=) (ZF = 1 or SF! = OF)
`cmova S,R`	`cmovnbe`	Above (unsigned >) (CF = 0 and ZF = 0)
`cmovae S,R`	`cmovnb`	Above or equal (unsigned >=) (CF = 0)
`cmovb S,R`	`cmovnae`	Below (unsigned <) (CF = 1)
`cmovbe S,R`	`cmovna`	Below or equal (unsigned <=) (CF = 1 or ZF = 1)

Practice: Conditional Move

```
int signed_division(int x) {
    return x / 4;
}
```

```
signed_division:
  leal 3(%rdi), %eax
  testl %edi, %edi
  cmovns %edi, %eax
  sarl $2, %eax
  ret
```

(See Sec. 2.3.7)

Put x + 3 into %eax (add appropriate bias, 2^2-1)

To see whether x is negative, zero, or positive

If x is positive, put x into %eax

Divide %eax by 4

Extra Practice

Practice: Fill In The Blank

Note: .L2/.L3 are "labels" that make jumps easier to read.

C Code

```
long loop(long a, long b) {
    long result = _______;
    while (_________) {
      result = _____________;
      a = _________;
    }
    return result;
}
```

Common while loop construction:

Jump to test
Body
Test
Jump to body if success

What does this assembly code translate to?

```
// a in %rdi, b in %rsi
loop:
    movl $1, %eax
    jmp .L2
.L3
    leaq (%rdi,%rsi), %rdx
    imulq %rdx, %rax
    addq $1, %rdi
.L2
    cmpq %rsi, %rdi
    jl .L3
rep; ret
```

Practice: Fill In The Blank

Note: .L2/.L3 are "labels" that make jumps easier to read.

C Code

```
long loop(long a, long b) {
    long result = ____1____;
    while (__ a < b __) {
      result = result*(a+b);
      a = __ a + 1 __;
    }
    return result;
}
```

Common while loop construction:

Jump to test
Body
Test
Jump to body if success

What does this assembly code translate to?

```
// a in %rdi, b in %rsi
loop:
    movl $1, %eax
    jmp .L2
.L3
    leaq (%rdi,%rsi), %rdx
    imulq %rdx, %rax
    addq $1, %rdi
.L2
    cmpq %rsi, %rdi
    jl .L3
rep; ret
```

Practice: “Escape Room”

```
escapeRoom:
  leal (%rdi,%rdi), %eax
  cmpl $5, %eax
  jg .L3
  cmpl $1, %edi
  jne .L4
  movl $1, %eax
  ret
.L3:
  movl $1, %eax
  ret
.L4:
  movl $0, %eax
  ret
```

What must be passed to the `escapeRoom` function such that it returns true (1) and not false (0)?

Practice: “Escape Room”

```
escapeRoom:
  leal (%rdi,%rdi), %eax
  cmpl $5, %eax
  jg .L3
  cmpl $1, %edi
  jne .L4
  movl $1, %eax
  ret
.L3:
  movl $1, %eax
  ret
.L4:
  movl $0, %eax
  ret
```

What must be passed to the `escapeRoom` function such that it returns true (1) and not false (0)?

First param > 2 or == 1.

Recap

- Assembly Execution and `%rip`
- Control Flow Mechanics
  - Condition Codes
  - Assembly Instructions
- If statements
- Loops
  - While loops
  - For loops
- Other Instructions That Depend On Condition Codes

Next time: Function calls in assembly