Practice: Fill In The Blank

slido

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
```

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
```

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.

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