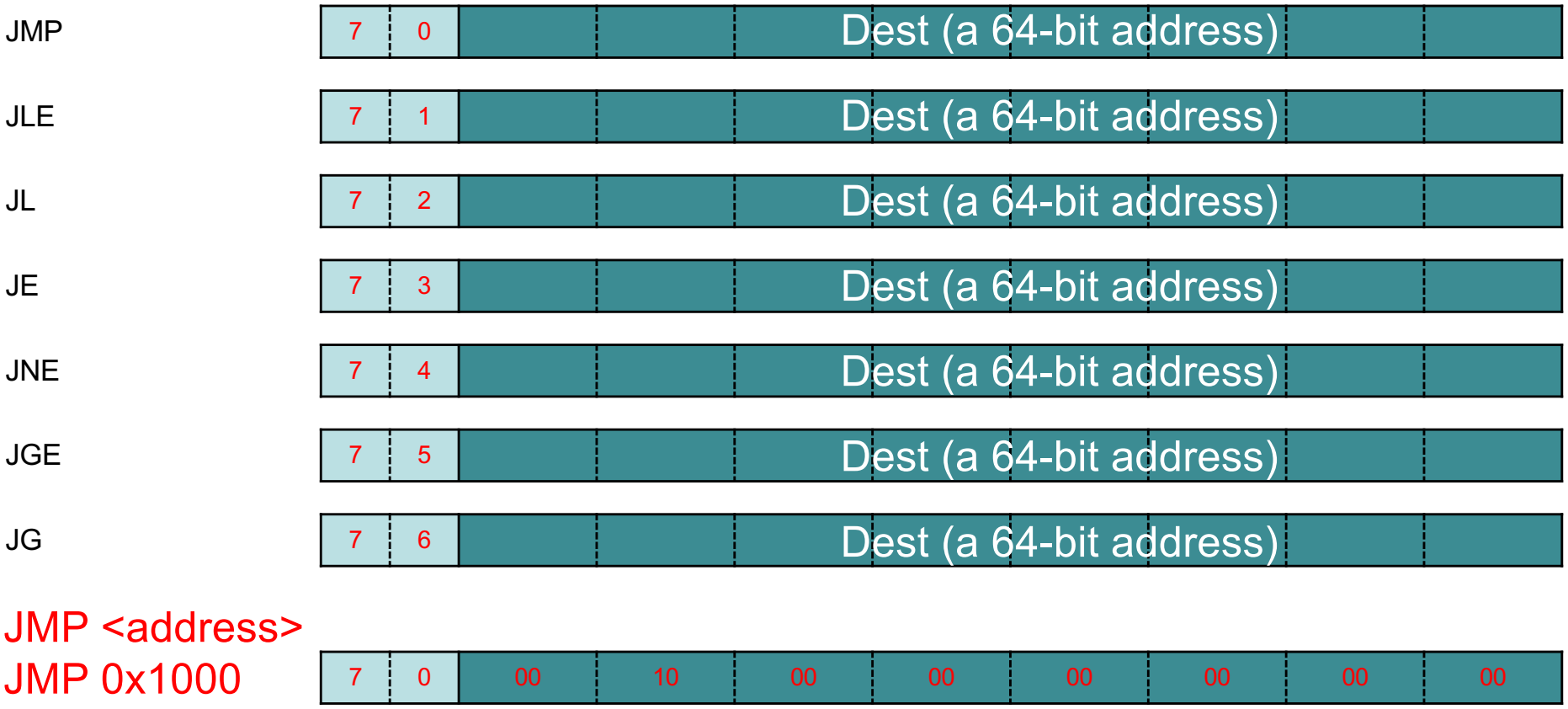
# Y86 Control Flow

- Topic
  - Using condition codes to direct execution
  - Conditional move instructions
- Learning Objectives
  - Write C conditional and looping constructs in y86.
  - Translate condition code configurations into actions for conditional JMP and MOVE instructions.
- Reading
  - 4.1.5

# The JMP instructions:

- Jump instructions: `jmp, jle, jl, je, jne, jg, jge`




JMP <address>
JMP 0x1000
JMP loop




PC <- Dest

# Conditional Jumps

- Jump based on the state of the condition codes:

| Condition | Test | Zero flag | | Sign Flag | |
|---|---|---|---|---|---|
| g | > 0 | 0 | and | 0 | !ZF & !SF |
| ge | >= 0 | | | 0 | !SF |
| e | == 0 | 1 | | | ZF |
| ne | != 0 | 0 | | | !ZF |
| le | <= 0 | 1 | or | 1 | ZF \| SF |
| l | < 0 | | | 1 | SF |

- Blanks mean that the flag can be either 0 or 1

# Code Example: Jump

```
# Using the conditional jump instructions
# to compare two numbers.
# Let's operate on our 'parameters'
# in registers rsi and rdi and our result
# in %rax
irmovq 0xff, %rsi
irmovq 0x1, %rdi

# rax = rsi > rdi ? 1 : 0
        irmovq 1, %rax  # initialize the
result
        rrmovq %rsi, %r10
        subq %rdi, %r10

# if %rdi was greater than %rsi, we want
# to return 1, which is what we already
# set %rax to, so we can go on to the next
# example
        jg next1
        irmovq 0, %rax  # Return 0
```

```
# rax = rsi  == rdi ? 1 : 0
next1:
        irmovq 1, %rax  # Initialize %rax
        rrmovq %rsi, %r10
        subq %rdi, %r10

# Once again, if the condition holds, we
# are done, because we initialized %rax to 1
        je next2
        irmovq 0, %rax  # Return 0

# rax = rsi >= 0
next2:
        irmovq 1, %rax  # Initialize to 1
        andq %r10, %r10

# if result should be 1, we're all set
        jge done        # Jump if true
        irmovq 0, %rax  # else, return 0
done:
        halt
```

# The Nifty Conditional Move

- Recall that we had a `fun` field for RRMOVQ

2-byte instruction


```
RRMOVQ %r8, %r9
```


- The CMOVxx instructions use the condition codes to provide conditional moves.

```
#Compare rsi and rdi
   irmovq 1, %rax
   rrmovq %rsi, %r10
   subq %rdi, %r10
   jxx L1
   irmovq 0, %rax
L1:
```

=

```
#Compare rsi and rdi
   irmovq 0, %rax
   irmovq 1, %r11
   rrmovq %rsi, %r10
   subq %rdi, %r10
   cmovxx %r11, %rax
```

# Code Example: CMOV

```
# Using the conditional move instructions
# to compare two numbers.
irmovq 0xff, %rsi
irmovq 0x1, %rdi
irmovq 0x1, %r11        # for return 1

# rax = rsi > rdi ? 1 : 0
        irmovq 0, %rax          # Initialize result to 0
        rrmovq %rsi, %r10
        subq %rdi, %rsi
        cmovg %r11, %rax        # Change result if condition is true

# rax = rsi  == rdi ? 1 : 0
        irmovq 0, %rax          # Initialize results to 0
        rrmovq %rsi, %r10
        subq %rdi, %r10
        cmove %r11, %rax        # Change result if condition is true

# rax = rsi >= 0
        irmovq 0, %rax          # Initialize results to 0
        andq %r10, %r10         # Sets flags based on value
# if result should be 1, we're all set
        cmovge %r11, %rax       # Change result if condition is true

        halt
```