Slide 20:

Why use movl for the following?

```
        movl    $0, %eax                # result = 0 ; also set higher order bytes of rax to 0
#the immediate operand $0 in the instruction movl $0, %rax is 4 bytes long
#the immediate operand $0 in the instruction movq $0, %rax is 8 bytes long
.L2:                                    # loop:
  movq    %rdi, %rdx
  andl    $1, %edx                      #  t = x & 0x1 also higher order 4 bytes of %rdx is still zero
  addq    %rdx, %rax      #  result += t
  shrq    %rdi            #  x >>= 1
  jne     .L2             #  if (x) goto loop
  rep; ret
```

Refer to Intel instruction table from the textbook on the moodle

Note the absence of an explicit instruction to zero-extend a 4-byte source value to an 8-byte destination in Figure 3.5. Such an instruction would logically be named `movzlq`, but this instruction does not exist. Instead, this type of data movement can be implemented using a `movl` instruction having a register as the destination. This technique takes advantage of the property that an instruction generating a 4-byte value with a register as the destination will fill the upper 4 bytes with zeros. Otherwise, for 64-bit destinations, moving with sign extension is supported for all three source types, and moving with zero extension is supported for the two smaller source types.