

# Introduction to 8086 Assembly

## Lecture 7

Multiplication and Division

# Multiplication commands: mul and imul



K. N. Toosi  
University of Technology

mul source (source: register/memory)



# Unsigned Integer Multiplication (mul)

- `mul src` (*src: register/memory*)
  - `src: 8 bits`      `ax ← al * src`
  - `src: 16 bits`    `dx:ax ← ax * src`
  - `src: 32 bits`    `edx:eax ← eax * src`
  - `src: 64 bits`    `rdx:rax ← rax * src` (**x64 only**)

# Unsigned Integer multiplication (mul)



K. N. Toosi  
University of Technology

mul src8



8 bit

mul src16



16 bit

mul src32



32 bit

mul src64



64 bit (x64 only)

# Example

- `mul bl`
- `mul bx`
- `mul ebx`
- `mul rbx (x64 only)`



# Example



K. N. Toosi  
University of Technology

```
11:  db  0xFF, 0x1A, 0x11, 0xE2  
      db  0x2A, 0x82, 0x1F, 0x74  
  
mul  [11]
```



# Example

```
11:  db  0xFF, 0x1A, 0x11, 0xE2
      db  0x2A, 0x82, 0x1F, 0x74
```

~~mul [11]~~

mul byte	[11]	; 8 bit	$AX = AL * [I1]$
mul word	[11]	; 16 bit	$DX:AX = AX * [I1]$
mul dword	[11]	; 32 bit	$EDX:EAX = EAX * [I1]$
mul qword	[11]	; 64 bit	$RDX:RAX = RAX * [I1]$

# Signed Integer Multiplication (imul)



K. N. Toosi  
University of Technology

- `imul src` (`src: register/memory`)
  - `src: 8 bits`      `ax <- al * src`
  - `src: 16 bits`    `dx:ax <- ax * src`
  - `src: 32 bits`    `edx:eax <- eax * src`
  - `src: 64 bits`    `rdx:rax <- rax * src` (x64 only)



# Signed Integer multiplication (mul)



K. N. Toosi  
University of Technology

`imul src8`



8 bit

`imul src16`



16 bit

`imul src32`



32 bit

`imul src64`



64 bit (x64 only)

# Question



K. N. Toosi  
University of Technology

Why not have **add** and **iadd** just like  
**mul** and **imul**?



# Other forms of imul

- `imul src`
- `imul dest, src`
- `imul dest, src1, src2`

`dest = dest * src`

`dest = src1 * src2`



# Other forms of imul

- **imul** src                      **src:** reg/mem
- **imul** dest, src                **dest:** reg    **src:** reg/mem/immed
- **imul** dest, src1, src2        **dest:** reg    **src1:** reg/mem    **src2:** immed

# Practice: Factorial



**K. N. Toosi**  
University of Technology

Write a program reading an integer and printing its factorial

- assume that the answer fits in 32 bits

# Practice: Factorial



K. N. Toosi  
University of Technology

fact.asm

```
call read_int
mov ecx, eax

mov eax, 1
loop1:
mul ecx

loop loop1

call print_int
call print_nl
```

# Practice: Factorial



K. N. Toosi  
University of Technology

Write a program reading an integer and printing its factorial

- print an error message if the answer is out of range

# Practice: Factorial



```
segment .data                                     fact2.asm
msg:      db "out of range!", 10, 0

segment .text
:
call read_int
mov  ecx, eax
mov  eax, 1
l1:
mul  ecx

cmp  edx, 0
jne  errlbl

loop l1
```

```
call print_int                                     fact2.asm (cont.)
call print_nl
jmp  endl

errlbl:
mov  eax, msg
call print_string

endl:
```



# Division

- `div source` (source: register/memory)
- `idiv source` (source: register/memory)





# Integer division (div, idiv)

- `div src8,`
- `idiv src8` (`src8: 8 bits`)
  - `al`  $\leftarrow$  `ax` / `src8` (quotient)
  - `ah`  $\leftarrow$  `ax` % `src8` (remainder)

## Example:

- `div bh`
- `div byte [11]`



# Integer division (div, idiv)

- `div src16,`
- `idiv src16` (`src16: 16 bits`)
  - `ax` ← `dx:ax / src16` (quotient)
  - `dx` ← `dx:ax % src16` (remainder)

## Example:

- `div cx`
- `div word [a]`



# Integer division (div, idiv)

- `div src32,`
- `idiv src32` (`src32: 32 bits`)
  - `eax` ← `edx:eax / src32` (`quotient`)
  - `edx` ← `edx:eax % src32` (`remainder`)

## Example:

- `div esi`
- `div dword [num1]`



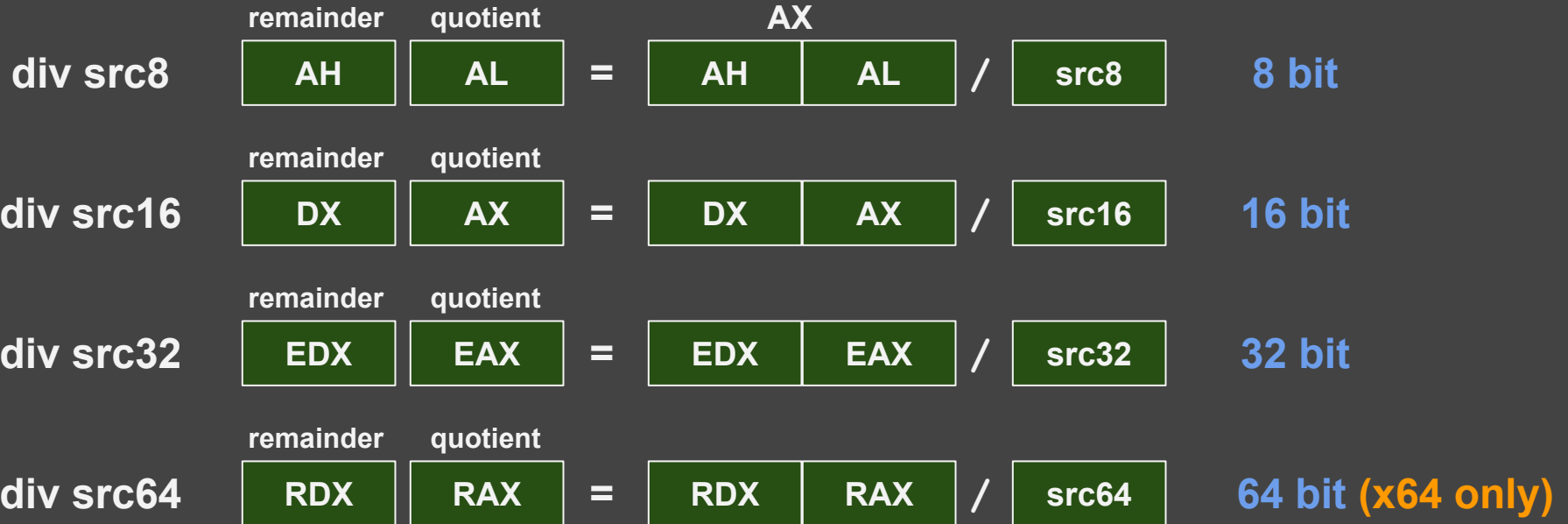
# Integer division (div, idiv)

- `div src64`,
- `idiv src64` (`src64`: 64 bits, x64 only)
  - `rax` ← `rdx:rax / src64` (quotient)
  - `rdx` ← `rdx:rax % src64` (remainder)

## Example:

- `div rdi`
- `div qword [sum]`

# Integer Division



# Further reading



**K. N. Toosi**  
University of Technology

- [https://www.tutorialspoint.com/assembly\\_programming/assembly\\_arithmetic\\_instructions.htm](https://www.tutorialspoint.com/assembly_programming/assembly_arithmetic_instructions.htm)
- [https://en.wikibooks.org/wiki/X86\\_Assembly/Arithmetic](https://en.wikibooks.org/wiki/X86_Assembly/Arithmetic)
- [https://www.csie.ntu.edu.tw/~acpang/course/asm\\_2004/slides/chapt\\_07\\_PartII\\_Solve.pdf](https://www.csie.ntu.edu.tw/~acpang/course/asm_2004/slides/chapt_07_PartII_Solve.pdf)

# Errors can happen in division



K. N. Toosi  
University of Technology

```
mov eax, 0
mov edx, 1 ; edx:eax=2^32

mov ecx, 1
div ecx
```



# Errors can happen in division



K. N. Toosi  
University of Technology

```
mov eax, 0
mov edx, 1 ; edx:eax=2^32

mov ecx, 1
div ecx
```

```
b.nasihatkon@kntu:lecture7$ ./run.sh divoverflow
./run.sh: line 5: 23877 Floating point exception(core dumped) ./$1
```

Usually dividend and divisor are  
of the same size!



K. N. Toosi  
University of Technology

Unsigned:

```
mov edx, 0  
div esi
```

Usually dividend and divisor are of the same size!



K. N. Toosi  
University of Technology

Unsigned:

```
mov edx, 0  
div esi
```

Signed:

```
CDQ  
idiv ebx
```

# Remember: Extending bit size - signed



K. N. Toosi  
University of Technology

- `AX ← AL`      **CBW**      (convert Byte to Word)
- `EAX ← AX`      **CWDE**      (convert Word to double word extended)
- `RAX ← EAX`      **CDQE**      (convert Double to Quad extended, **x64**)
  
- `DX:AX ← AX`      **CWD**      (convert Word to Double word)
- `EDX:EAX ← EAX`      **CDQ**      (convert Double word to Quad word)
- `RDX:RAX ← RAX`      **CQO**      (convert Quad word to Oct Word, **x64**)

# Practice: Prime Numbers



K. N. Toosi  
University of Technology

Write a program reading an integer and printing if it is prime

- assume that input is larger than 1

# Practice: Prime Numbers



segment .data

prime.asm

```
prime_msg:    db "Prime!", 10, 0
notprime_msg: db "Not prime!", 10, 0
```

segment .text

:

```
call read_int
mov ebx, eax
```

```
mov ecx, 2
```

startloop:

```
cmp ecx, ebx
jge endloop
```

```
mov eax, ebx
mov edx, 0
div ecx
cmp edx, 0
```

```
je notprime_lbl
```

```
inc ecx
jmp startloop
```

endloop:

```
mov eax, prime_msg
call print_string
jmp endl
```

notprime\_lbl:

```
mov eax, notprime_msg
call print_string
```

endl:

prime.asm (cont.)

# Code on the right also correct?



startloop:

```
    cmp ecx, ebx
    jge endloop

    mov eax, ebx
    mov edx, 0
    div ecx
    cmp edx, 0
    je  notprime_lbl

    inc ecx
    jmp startloop
```

endloop:

```
    mov eax, prime_msg
    call print_string
    jmp  endl
```

notprime\_lbl:

```
    mov eax, notprime_msg
    call print_string
```

endl:

startloop:

```
    cmp ecx, eax
    jge endloop

    mov eax, ebx
    mov edx, 0
    div ecx
    cmp edx, 0
    je  notprime_lbl

    inc ecx
    jmp startloop
```

endloop:

```
    mov eax, prime_msg
    call print_string
    jmp  endl
```

notprime\_lbl:

```
    mov eax, notprime_msg
    call print_string
```

endl: