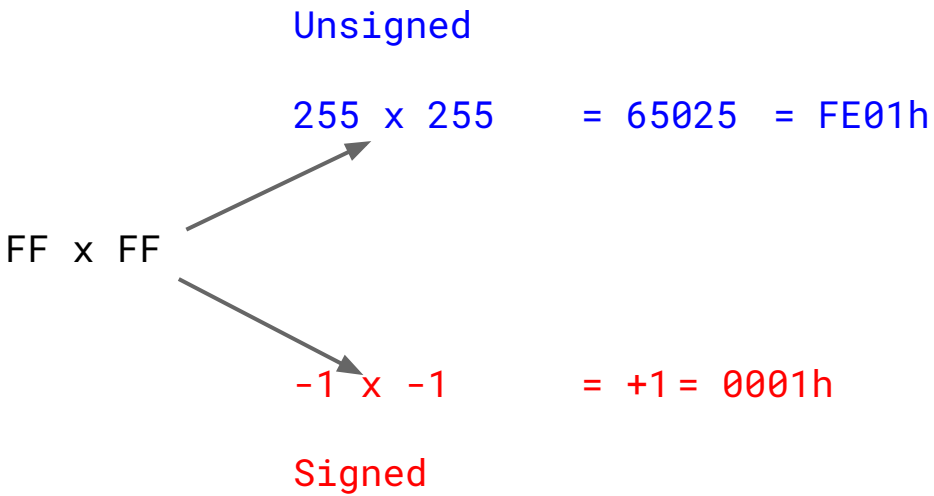
Multiply & divide

CSC 236

Multiply & divide

- Different from addition/subtraction
  - Separate instructions required for signed/unsigned



Multiply & divide

- Different from addition/subtraction
  - Separate instructions required for signed/unsigned
    - User must select instruction
  - Data sizes grow (not merely overflow)

```
        9 x         9 =     81
       99 x        99 =   9801
      999 x       999 = 998001
     …
d digits x d digits = 2d digits
d digits x e digits = d+e digits

    byte x      byte = word
    word x      word = double word
```

Multiply & divide

- Different from addition/subtraction
  - Separate instructions required for signed/unsigned
    - User must select instruction
  - Data sizes grow (not merely overflow)
  - Divide is distinct operation
    - Divide is not “multiply by reciprocal”

Instructions

- Unsigned: `mul <operand>`
- Signed: `imul <operand>`

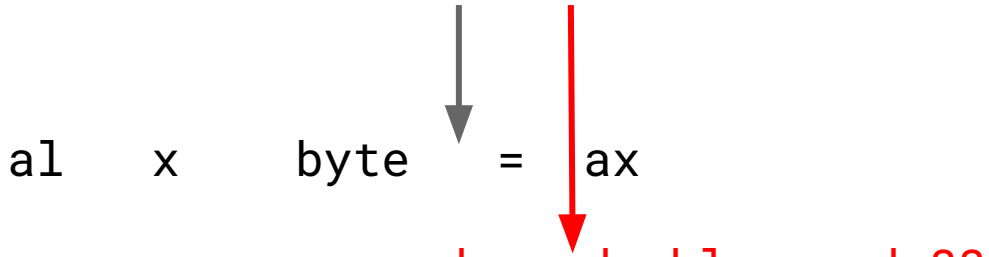
- Only one operand
  - Multiply requires two
  - One is implicit
- Operand
  - Byte or word
  - Register or memory

Instructions

- Unsigned: `mul <operand>`
- Signed: `imul <operand>`

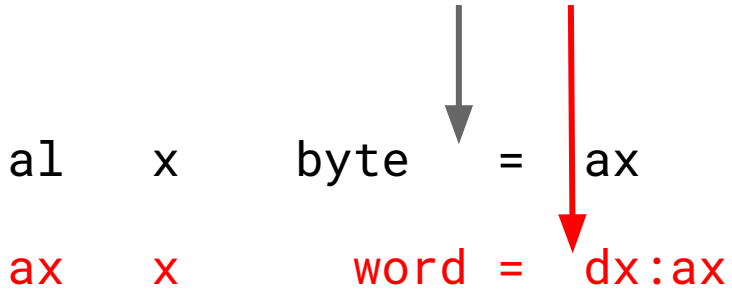
Instructions

- Unsigned: `mul <operand>`
- Signed: `imul <operand>`



Instructions

- Unsigned: `mul <operand>`
- Signed: `imul <operand>`

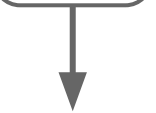


Two cases

- `al x byte = ax`
- `ax x word = dx:ax`

Two cases

- `al x byte = ax`
- `ax x word = dx:ax`



register or memory

cannot use immediate

Two cases

- `al x byte = ax`
- `ax x word = dx:ax`

- Overflow is not possible
- CF indicate result size
  - 0 ⇒ significant part same size as inputs
  - 1 ⇒ significant part larger than inputs

- `02 x 05 = 00 0A`
  - `CF = 0`
- `0A x 64 = 03 E8`
  - `CF = 1`

- **Useful for *chained* multiplies**

Example 1

- Multiply unsigned bytes
  - 10 x 100

```
u10    db  10    ;0A
u100   db 100    ;64
```

```
mov    al,[u10]   ;ax = -- 0A
mul    [u100]     ;ax = 03 E8
                  ;CF = 1
```

Example 2

- Multiply signed words
  - +2 x -1

```
p2     dw  2     ;00 02
neg1   dw -1     ;FF FF
```

```
mov    ax,[p2]   ;dx:ax = ---- 00 02
imul   [neg1]    ;dx:ax = FFFF FFFE
                 ;CF = ?
                 ;CF = 0
```

Division

- Grammar school
  - Integers only
  - No fractions
- Divide
  - 10 / 3
  - Answer: 3 remainder 1

Division

- Division in high-level language
  - Quotient — `div /` : `10 / 3 = 3`
  - Remainder — `mod %` : `10 % 3 = 1`

  $$\frac{\text{dividend}}{\text{divisor}}$$

- Quotient: how many times can one subtract divisor from dividend
- Remainder: the amount left over (after above)

Example

- Quotient: how many times can one subtract divisor from dividend
- Remainder: the amount left over (after these subtractions)

$$\begin{array}{r} 10 \\ \hline 3 \end{array}$$

Can subtract 3 from 10 3 times with 1 left over

Quotient = 3
Remainder = 1

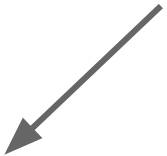
Instructions

- Unsigned: `div <operand>`
- Signed: `idiv <operand>`

- Implicit operand
- Byte or word
- Register or memory

Instructions

- Unsigned: `div <operand>`
- Signed: `idiv <operand>`



```
 ax
----     =  ah al
byte        rem quot
```

Instructions

- Unsigned: `div <operand>`
- Signed: `idiv <operand>`

```
 ax                          dx:ax
----     =   ah al          -----     =   dx ax
byte          rem quot       word         rem quot
```

Overflow

- Unlike multiplication
- Can have overflow
  - If quotient > byte : `1000/1 > byte`
  - If quotient > word: `1000000/1 > word`

Example 3

- unsigned_word / byte
- 10 / 3

```
mov   ax,10     ;ax = 00 0A
mov   bl,3      ;bx = -- 03
div   bl        ;ax = 01 03
```

Remainder (ah) = 1

Quotient (al) = 3

Notes

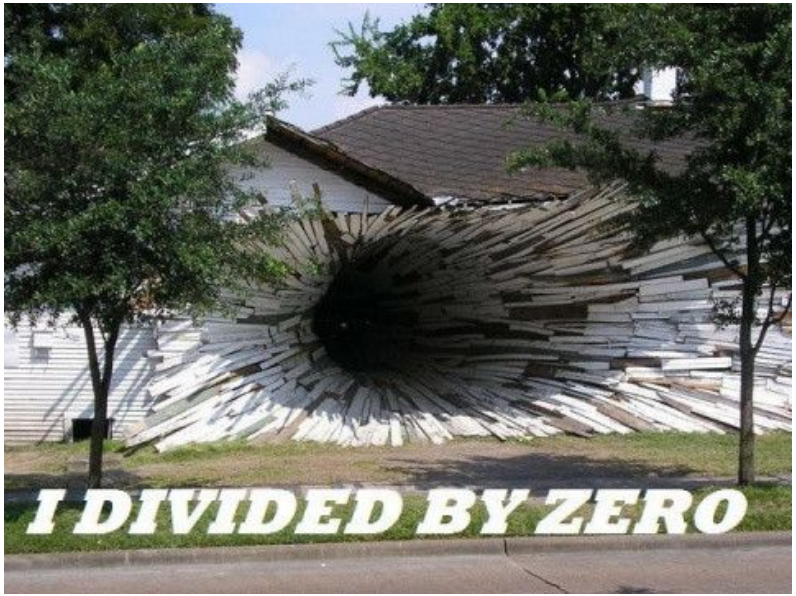
- Overflow if
  - Quotient is too big
    - alternatively : divisor too small
  - Divisor is zero

Notes

- Overflow if
  - Quotient is too big
    - alternatively : divisor too small
  - Divisor is zero

Generates divide overflow interrupt

Program is terminated



Notes

- Overflow if
  - Quotient is too big
    - alternatively : divisor too small
  - Divisor is zero

Generates divide overflow interrupt

Program is terminated

Not exactly: interrupt can be caught by *interrupt handler*

Notes

- Overflow if
  - Quotient is too big
    - alternatively : divisor too small
  - Divisor is zero
- Status flags undefined after divide
- Cannot divide by immediate
  - ~~`div 7`~~

Signs

Quotient & remainder are signed numbers

```
+10    +10    -10    -10
 + 3    - 3    + 3    - 3
```

- Quotient
  - + for like signs
  - - for unlike signs
- Remainder
  - Sign of dividend

Signs

Quotient & remainder are signed numbers

- Quotient
  - + for like signs
  - - for unlike signs
- Remainder
  - Sign of dividend

```
  +10     +10     -10     -10
+ 3     - 3     + 3     - 3
```

Q =

R =

Signs

Quotient & remainder are signed numbers

- Quotient
  - + for like signs
  - - for unlike signs
- Remainder
  - Sign of dividend

```
   +10     +10     -10     -10
   + 3     - 3     + 3     - 3

Q = +3                      +3

R =
```

Signs

Quotient & remainder are signed numbers

- Quotient
  - + for like signs
  - - for unlike signs
- Remainder
  - Sign of dividend

	+10	+10	-10	-10
	+ 3	- 3	+ 3	- 3
Q =	+3	-3	-3	+3
R =				

Signs

Quotient & remainder are signed numbers

- Quotient
  - + for like signs
  - - for unlike signs
- Remainder
  - Sign of dividend

	+10	+10	-10	-10
	+ 3	- 3	+ 3	- 3
Q =	+3	-3	-3	+3
R =	+1	+1	-1	-1

- **Quotient: how many times can one subtract divisor from dividend**
- **Remainder: the amount left over (after these subtractions)**

Signs

Quotient & remainder are signed numbers

- Quotient
  - + for like signs
  - - for unlike signs
- Remainder
  - Sign of dividend*

	+10	+10	-10	-10
	+ 3	- 3	+ 3	- 3
Q =	+3	-3	-3	+3
R =	+1	+1	-1	-1

*** Exception: remainder of zero technically isn't negative, even if the dividend is.**

Example 4

- signed_long_word / word
- +10 / -3

```
mov  ax,10     ;dx:ax = ---- 000A
cwd            ;dx:ax = 0000 000A
mov  bx,-3     ;bx    =      FFFD
idiv bx        ;dx:ax = 0001 FFFD
```

Remainder (dx) = +1

Quotient (ax) = -3

Overflow

- Addition/subtraction
  - Detect overflow after operation
- Multiply
  - Cannot overflow
- Divide
  - ??

Overflow

- Addition/subtraction
  - Detect overflow after operation
- Multiply
  - Cannot overflow
- Divide
  - Predict overflow

Prediction

ax	04E7	00 B3
bl	07	
ax	04E7	01 39
bl	04	
ax	04E7	02 73
bl	02	

- Which overflows?
  - If high byte (red) is not zero
- Can this be determined before divide?
  - Yes

Legend

$04E7_{16} = 1255_{10}$
1255 div 7 = 179
1255 div 4 = 313
1255 div 2 = 627

Prediction

ax	04E7	00	B3
bl	07		
ax	04E7	01	B3
bl	04		
ax	04E7	02	B3
bl	02		

- Which overflows?
  - If high byte (red) is not zero
- Can this be determined before divide?
- Compare
  - High-byte of dividend and
  - Divisor

`04 < 07` ⇒ no overflow

Prediction

```
ax      04E7        00 B3
bl      07

ax      04E7        01 B3
bl      04

ax      04E7        02 B3
bl      02
```

- Which overflows?
  - If high byte (red) is not zero
- Can this be determined before divide?
- Compare
  - High-byte of dividend and
  - Divisor

`04 < 07` ⇒ no overflow

`04 = 04` ⇒ overflow

Prediction

```
ax      04E7        00 B3
bl      07

ax      04E7        01 B3
bl      04

ax      04E7        02 B3
bl      02
```

- Which overflows?
  - If high byte (red) is not zero
- Can this be determined before divide?
- Compare
  - High-byte of dividend and
  - Divisor

04 < 07

Rule: There will be overflow iff high-part-of-dividend ≥ divisor

02 ⇒ overflow

Example with overflow (pre) check

- Calculate quotient and remainder
  - (a*b)/c
  - All unsigned bytes

```
mov  al,[a]   ;al=a
mul  [b]      ;ax=a*b
```

Example with overflow (pre) check

- Calculate quotient and remainder
  - (a*b)/c
  - All unsigned bytes

```
mov  al,[a]   ;al=a
mul  [b]      ;ax=a*b
```

```
 ax   →    ah al
[c]          [c]
```

ah must be less than [c]

Example with overflow (pre) check

- Calculate quotient and remainder
  - (a*b)/c
  - All unsigned bytes

```
mov  al,[a]    ;al=a
mul  [b]       ;ax=a*b
cmp  ah,[c]    ;hi-byte ? divisor
jae  cannot    ;do not divide
div  [c]       ;ah=rem, al=quot
mov  [quot],al
mov  [rem],ah
```

Plan ahead

```
mov  al,[a] ; ax = -- a
imul [a]    ; ax = (a)*a      = a^2
imul [a]    ; ax = (a*a)*a    = a^3
imul [a]    ; ax = (a*a*a)*a  = a^4
```

Is this OK?

Calculate a^4
where a is signed byte
result is signed word

Plan ahead

```
mov  al,[a] ; ax = -- a
imul [a]    ; ax = (a)*a     = a^2
imul [a]    ; ax = (a*a)*a   = a^3
imul [a]    ; ax = (a*a*a)*a = a^4
```

Is this OK?

```
mov  al,[a] ; ax = -- a
imul [a]    ; ax = (a)*a     = a^2
imul [a]    ; ax = (a*a)*a   = a^3
```

If a*a > byte this fails (overflows)

Plan ahead

```
mov  al,[a] ; ax = -- a
imul [a]    ; ax = (a)*a     = a^2
imul [a]    ; ax = (a*a)*a   = a^3
imul [a]    ; ax = (a*a*a)*a = a^4
```

```
mov  al,[a] ; ax = -- a
imul al     ; ax = a*a  = a^2
imul ax     ; ax = a^2*a^2 = a^4
jc   error  ; cf=0 ⇒ a^4 in ax
```

Simpler and safer

Generated code

C source code

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;
```

Generated code

C source code

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;

   (20*10)/3
     (200)/3
```

Generated code

C source code

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;
```

66

```
   (20*10)/3
     (200)/3
```

Generated code

C source code

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;

q1= a * (b / c);
```

66

Generated code

C source code

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;

q1= a * (b / c);

    20*(10/3)
    20*(3)
```

66

Generated code

C source code

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;

q1= a * (b / c);

    20*(10/3)
    20*(3)
```

66

60

Generated code 2

C source code

```
unsigned char a, b, c, q1;
a=100; b=10; c=3;

q1= (a * b) / c;
```

Generated code 2

C source code

```
unsigned char a, b, c, q1;
a=100; b=10; c=3;

q1= (a * b) / c;

  (100*10)/3 = 333
```

Generated code 2

C source code

```
unsigned char a, b, c, q1;
a=100; b=10; c=3;

q1= (a * b) / c;

  (100*10)/3 = 333
```

77

Generated code 2

C source code

```
unsigned char a, b, c, q1;
a=100; b=10; c=3;

q1= (a * b) / c;
```

77

```
  (100*10)/3 = 333
             = 01 4Dh
```

Generated code 2

C source code

```
unsigned char a, b, c, q1;
a=100; b=10; c=3;

q1= (a * b) / c;
```

```
(100*10)/3 = 333
           = 01 4Dh
```

77

```
4Dh = 4(16) + 13
    = 64 + 13
    = 77
```

Generated code 3

C source code

```
unsigned char a2, a3, a4, a=20;

a2=a * a;
```

Generated code 3

C source code

```
unsigned char a2, a3, a4, a=20;

a2=a * a;
```

144

```
  20 * 20 = 400
```

Generated code 3

C source code

```
unsigned char a2, a3, a4, a=20;

a2=a * a;
```

144

20 * 20 = 400

$400_{10} = 0190_{16}$

$90_{16} = 2^7 + 2^4 = 128 + 16$

Generated code 3

C source code

```
unsigned char a2, a3, a4, a=20;

a2=a * a;
a3=a * a * a;
```

144

64

$8000_{10} = 20_{10} * 20_{10} * 20_{10}$

$= 1F\underline{40}_{16}$

$40_{16} = 64_{10}$

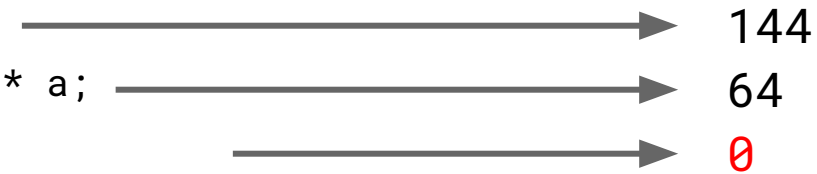
Generated code 3

C source code

$160000_{10} = 027100_{16}$

```
unsigned char a2, a3, a4, a=20;

a2=a * a;
a3=a * a * a;
a4=a * a * a * a;
```



Generated code 4

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;
q1= a * (b / c);

a=100;
q1= (a * b) / c;

unsigned char a2, a3, a4; a=20;

a2=a * a;
a3=a * a * a;
a4=a * a * a * a;
```

- C source
  - 9 multiplies
  - 3 divides
- How many did compiler emit?

Generated code 4

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;
q1= a * (b / c);

a=100;
q1= (a * b) / c;

unsigned char a2, a3, a4; a=20;

a2=a * a;
a3=a * a * a;
a4=a * a * a * a;
```

- C source
  - 9 multiplies
  - 3 divides
- How many did compiler emit?
  - > 12
  - < 12

Generated code 4

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;
q1= a * (b / c);

a=100;
q1= (a * b) / c;

unsigned char a2, a3, a4; a=20;

a2=a * a;
a3=a * a * a;
a4=a * a * a * a;
```

- C source
  - 9 multiplies
  - 3 divides
- How many did compiler emit?
  - > 12
  - < 12
- It emitted
  - Zero (0) multiplies
  - Zero (0) divides
- ???

Generated code 4

```
unsigned char a, b, c, q1;
a=20; b=10; c=3;

q1= (a * b) / c;
q1= a * (b / c);

a=100;
q1= (a * b) / c;

unsigned char a2, a3, a4; a=20;

a2=a * a;
a3=a * a * a;
a4=a * a * a * a;
```

- C source
  - 9 multiplies
  - 3 divides
- How many did compiler emit?
  - > 12
  - < 12
- It emitted
  - Zero (0) multiplies
  - Zero (0) divides
- Compiler
  - Knows all values at compile time
  - Creates result itself
  - `q1=(a*b)/c` ⇒ `mov [q1], 66`

Test 1

Cover through lecture 10

- Number systems
- Basic architecture
- 8086 architecture
- Assembler instructions
- Declaring data
- Flow control and looping
- File I/O

Test 1 — Open book

- You may use
  - Any printed material from the course Moodle page.
  - Anything printed material you created (eg, notes, cheatsheets, etc)
  - Specifically, you can use
    - Old tests
    - Homeworks
    - Your own class notes
    - Class slides
    - Class text
- You may not use a calculator, phone, computing device

Test 1 — Open book

- Most effective way to prepare
  - Take the old tests -- which are on the web site
- Test covers
  - Lectures up to and including lecture 10
  - Class notes up to and including chapter 8
- 35-40 multiple choice questions
  - ~15 number systems
  - ~5 architecture
  - ~15 8086 assembler instructions, data declarations, flow control, IO