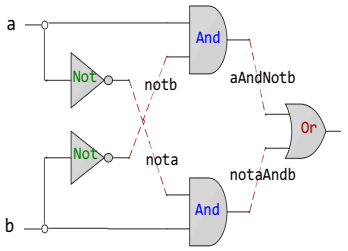
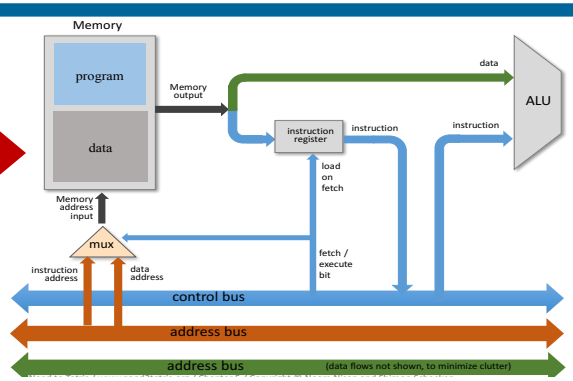
Computer Organization & Assembly Language Programming




```
CHIP Xor {
  IN a, b;
  OUT out;
  PARTS:
  Not(in=a, out=nota);
  Not(in=b, out=notb);
  And(a=nota, b=b, out=w1);
  And(a=a, b=notb, out=w2);
  Or(a=w1, b=w2, out=out);
}
```




```
@R1
D=M
@temp
M=D
```

```
0000000000000001
1111110000010000
0000000000010000
1110001100001000
```

Lecture # 36

Arithmetic Instructions - II

```
#include<stdio.h>
#include<stdlib.h>
int main(){
  printf("Learning is fun with Arif\n");
  exit(0);
}
```

```
global main
SECTION .data
   msg: db "Learning is fun with Arif", 0Ah, 0h
   len_msg: equ $ - msg
SECTION .text
   main:
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,len_msg
      syscall
      mov rax,60
      mov rdi,0
      syscall
```

```
0:  b8 01 00 00 00
5:  bf 01 00 00 00
a:  48 be 00 00 00 00 00
11: 00 00 00
14: ba 1b 00 00 00
19: 0f 05
1b: b8 3c 00 00 00
20: bf 00 00 00 00
25: 0f 05
```

For resources visit my personal website:
https://www.arifbutt.me
and course bitbucket repository:
https://bitbucket.org/arifpucit/coal-repo

Instructor: Muhammad Arif Butt, Ph.D.

Today's Agenda

- Recap: x86-64 Registers, Tool Chain & Instructions
- Arithmetic Instructions
  - Unsigned Multiplication (***mul.nasm***)
  - Signed Multiplication (***imul.nasm***)
  - Unsigned Division (***div.nasm***)
  - Signed Division (***idiv.nasm***)

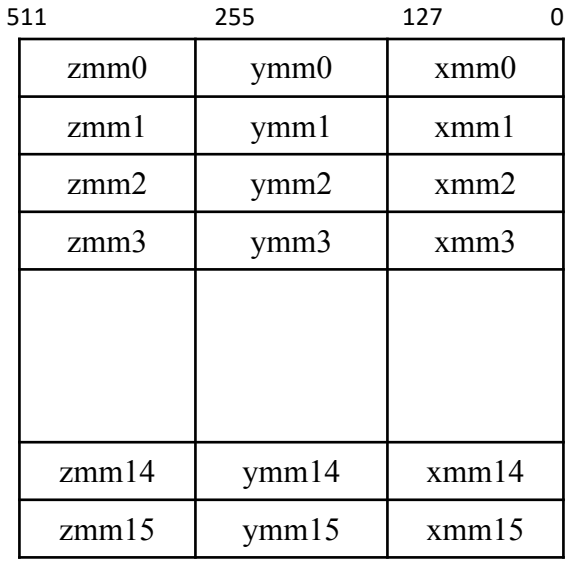
Recap

Review: x86-64 Register Set

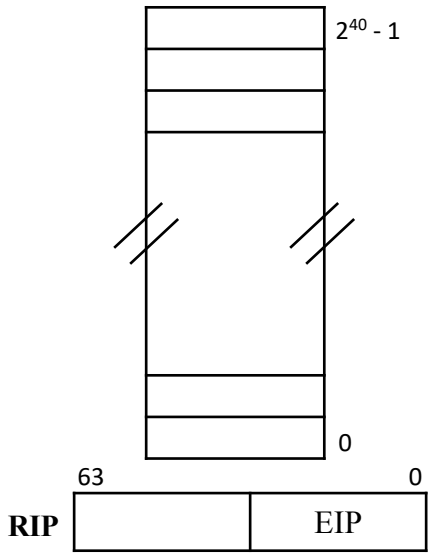
General Purpose Registers

64-bit register	Lowest 32-bits	Lowest 16-bits	Lowest 8-bits
r0/rax	eax	ax	al
r1/rbx	ebx	bx	bl
r2/rcx	ecx	cx	cl
r3/rdx	edx	dx	dl
r4/rsi	esi	si	sil
r5/rdi	edi	di	dil
r6/rbp	ebp	bp	bpl
r7/rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r8d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

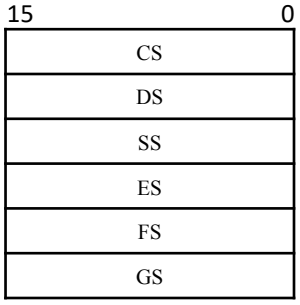
SSE Media Registers



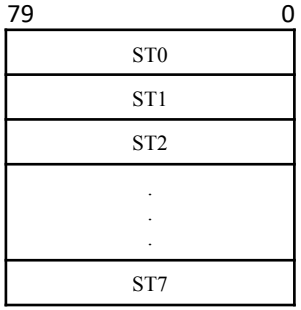
Memory

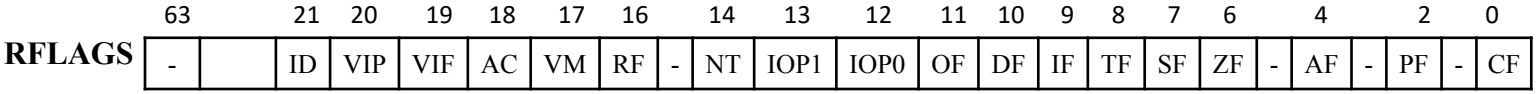


Segment Registers



FP Registers





Review: x86-64 Tool Chain

first.nasm

```
; COAL Video Lecture: 30
;  Programmer: Arif Butt
;  first.nasm
SECTION .data
   msg db "Learning…", 0xA
   EXIT_STATUS equ 54
SECTION .bss
;nothing here
SECTION .text
   global _start
   _start:
;display a message on screen
      mov rax,1
      mov rdi,1
      mov rsi,msg
      mov rdx,26
      syscall
;exit the program
      mov rax,60
      mov rdi, EXIT_STATUS
      syscall
```

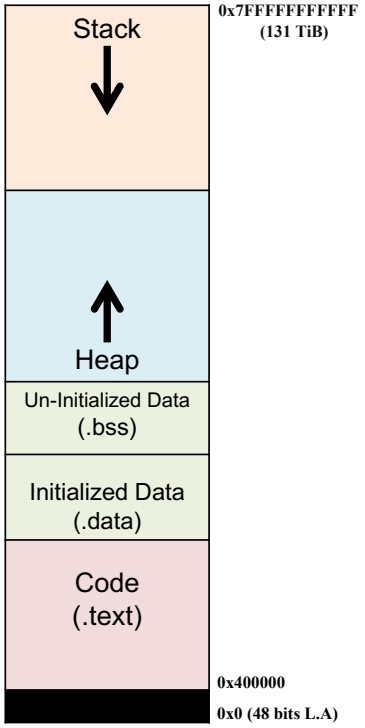
Assemble → **first.o**

```
10001000
01000001
1000101001001001
0101011000011111
0001010011110000
10001000
01001101
10001000
01001001
1000101001001000
0101011000011000
0001010010010001
10001010
01001011
```

Link → **myexe**

```
1000101001001001
0101011000011111
0001010011110000
10001000
01001101
10001000
10001000
01000001
0101011000011111
0001010011110000
10001000
1000101001001000
0001010010010001
10001010
01001011
0001010011110000
10001000
01001101
10001000
```

Load & Execute




```
$ nasm –felf64 first.nasm
$ ld first.o –o myexe
$./myexe
Learning is fun with Arif
```

Review: Categories of x86-64 Instructions

Category	Description	Examples
Data Transfer	Move from source to destination	`mov, movzx, movsx, lea, lds, lss, xchg, push, pop, pusha, popa, pushf, popf`
Arithmetic	Arithmetic on integer	`add, addc, sub, subb, mul, imul, div, idiv, neg, inc, dec, cmp`
Bit Manipulation	Logical & bit shifting operations	`and, or, not, xor, test, shl/sal, shr, sar, ror, rol, rcr, rcl`

Control Transfer	Conditional and unconditional jumps, and procedure calls	`jmp`
		`jcc(jz,jnz,jg,jge,jl,jle,jc,jnc,...)`
		`call, ret`

String	Move, compare, input and output	`movsb, movsw, lodsb, lodsw, stosb, stosw, rep, repz, repe, repnz, repne`
Floating Point	Arithmetic	`fld, fst, fstp, fadd, fsub, fmul, fdiv`
Conversion	Data type conversions	`cbw, cwd, cdq, xlat`
Input Output	For input and output	`in, out`
Miscellaneous	Manipulate individual flags	`clc, stc, cld, std, sti`

Multiplication Instructions

Multiplying Two 8 bit Numbers

In Decimal

```
    2 5 3   (Multiplicand)
x   2 5 0   (Multiplier)
    0 0 0
  1 2 6 5 x
  5 0 6 x x
  6 3 2 5 0   (Product)
```

In Binary

```
      1 1 1 1 1 1 0 1   (Multiplicand)
x     1 1 1 1 1 0 1 0   (Multiplier)
      0 0 0 0 0 0 0 0
    1 1 1 1 1 1 0 1 x
  0 0 0 0 0 0 0 0 x x
1 1 1 1 1 1 0 1 x x x
  1 1 1 1 1 1 0 1 x x x x
    1 1 1 1 1 1 0 1 x x x x x
      1 1 1 1 1 1 0 1 x x x x x x
1 1 1 0 1 1 1 1 0 0 0 1 0 0 1 0   (Product)
```

Unsigned Multiplication

`MUL multiplier`

Operand Size	Multiplier (Operand)	Multiplicand (Implicit)	Destination
Byte	r/m8	AL	AX = AL * r/m8
Word	r/m16	AX	DX:AX = AX * r/m16
Doubleword	r/m32	EAX	EDX:EAX = EAX * r/m32
Quadword	r/m64	RAX	RDX:RAX = RAX * r/m64

- CF and OF are both set if the product extends into the higher register AH, DX, or EDX respectively

Multiplicand		Multiplier		High-order 8-bits	Low-order 8-bits
AL	X	8-bit Multiplier	=	AH	AL
AX	X	16-bit Multiplier	=	DX	AX
EAX	X	32-bit Multiplier	=	EDX	EAX
RAX	X	64-bit Multiplier	=	RDX	RAX

Unsigned Multiplication (cont…)

MUL multiplier

Operand Size	Multiplier (Operand)	Multiplicand (Implicit)	Destination
Byte	r/m8	AL	AX = AL * r/m8
Word	r/m16	AX	DX:AX = AX * r/m16
Doubleword	r/m32	EAX	EDX:EAX = EAX * r/m32
Quadword	r/m64	RAX	RDX:RAX = RAX * r/m64

- CF and OF are both set if the product extends into the higher register AH, DX, or EDX respectively

Multiplying two 8-bit Numbers

```
MOV al, 4h   ;multiplicand
MOV bl, 10h  ;multiplier
MUL bl       ;AX = 0040h CF=OF=0
```

```
MOV al, 4h   ;multiplicand
var db 10h   ;multiplier
MUL byte[var] ;AX = 0040h CF=OF=0
```

Multiplying two 16-bit Numbers

```
MOV ax, 600h ;multiplicand
MOV bx, 100h ;multiplier
MUL bx       ;DX:AX = 0006 0000h
```

```
MOV ax, 600h ;multiplicand
var dw 100h  ;multiplier
MUL word[var] ;DX:AX = 0006 0000h
```

Question: What will be hexa-decimal values of edx and eax, and the carry flag after the execution of following code:

```
MOV eax, 00128765h    MOV ecx 1000h    MUL ecx
```

Assembling & Executing x86_64 Program

Signed Multiplication

IMUL multiplier

Operand Size	Multiplier (Operand)	Multiplicand (Implicit)	Destination
Byte	r/m8	AL	AX = AL * r/m8
Word	r/m16	AX	DX:AX = AX * r/m16
Doubleword	r/m32	EAX	EDX:EAX = EAX * r/m32
Quadword	r/m64	RAX	RDX:RAX = RAX * r/m64

- CF and OF are both set if the product extends into the higher register AH, DX, or EDX respectively

8-bit Multiplication

```
MOV al, -4d ;multiplicand
MOV bl,100d ;multiplier
IMUL bl     ;AX = FE70 (CF=OF=1)
```

16-bit Multiplication

```
MOV ax, -5d ;multiplicand
MOV bx, 5d  ;multiplier
IMUL bx     ;DX:AX=FFFF FFE7(CF=OF=0)
```

32-bit Multiplication

```
MOV eax, 4823424 ;multiplicand
MOV ebx, -423    ;multiplier
IMUL ebx         ;EDX:EAX = FFFFFFFF 86635D80h (CF=OF=0)
```

Multi-Operand IMUL Instruction

Two Operand Variant

- **Format:** `IMUL dest, source`
- **Operation:** Destination = Source * Destination
- **Operands:** Destination operand must be a register
  Source operand can be an imm/reg/mem

Example

```
MOV r8, 4823424
MOV r9, -423
IMUL r9, r8
;r9= r8 * r9
;r9= FFFFFFFF86635D80h
;CF = OF = 0
```

Three Operand Variant

- **Format:** `IMUL dest, src1, src2`
- **Operation:** Destination = src1 * src2
- **Operands:** Destination operand must be a register
  Source1 operand can be an reg/mem
  Source2 operand must be an immediate

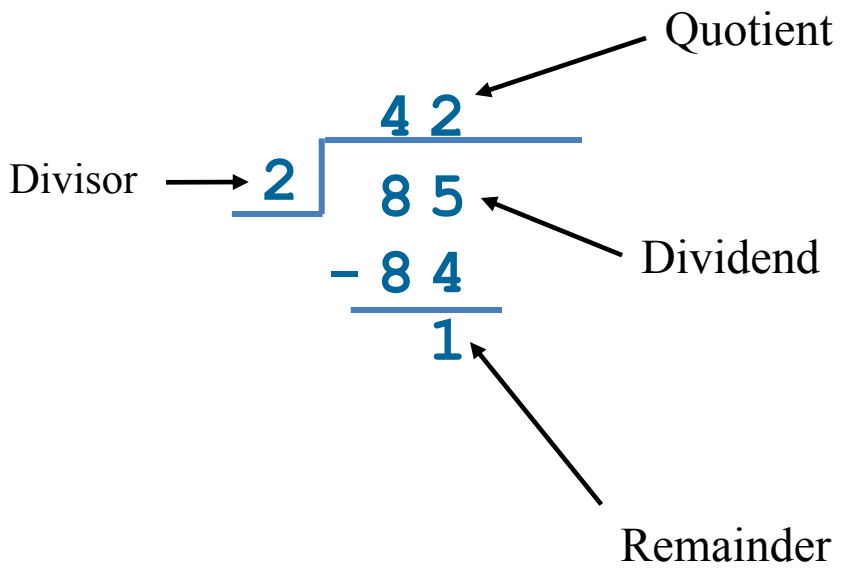
Example

```
MOV r8, 4823424
IMUL r9, r8, -423
;r9= r8 * -423
;r9= FFFFFFFF86635D80h
;CF = OF = 0
```

Assembling & Executing x86_64 Program

Division Instructions

Dividing Two 8 bit Numbers



Unsigned Division

DIV divisor

Operand Size	Dividend	Divisor	Quotient	Remainder	Maximum Quotient
Word/byte	AX	r/m8	AL	AH	$2^8 - 1$
Doubleword/word	DX:AX	r/m16	AX	DX	$2^{16} - 1$
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	$2^{32} - 1$
128-bit/64-bit	RDX:RAX	r/m64	RAX	RDX	$2^{64} - 1$
Status Flags are undefined for division instruction					

16-bits Dividend		8-bits Divisor		Quotient	Remainder
AX	/	8-bit Source	=	AL	AH

32-bits Dividend		16-bits Divisor		Quotient	Remainder
DX:AX	/	16-bit Source	=	AX	DX

64-bits Dividend		32-bits Divisor		Quotient	Remainder
EDX:EAX	/	32-bit Source	=	EAX	EDX

128-bits Dividend		64-bits Divisor		Quotient	Remainder
RDX:RAX	/	64-bit Source	=	RAX	RDX

Unsigned Division (cont…)

DIV divisor

Operand Size	Dividend	Divisor	Quotient	Remainder	Maximum Quotient
Word/byte	AX	r/m8	AL	AH	$2^8 - 1$
Doubleword/word	DX:AX	r/m16	AX	DX	$2^{16} - 1$
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	$2^{32} - 1$
128-bit/64-bit	RDX:RAX	r/m64	RAX	RDX	$2^{64} - 1$
Status Flags are undefined for division instruction					

16-bit / 8-bit Division

```
MOV ax, 85  ; dividend
MOV bl, 2   ; divisor
DIV bl      ; AL=42h, AH=01h
```

32-bit / 16-bit Division

```
MOV dx, 0h      ;dividend (high)
MOV ax, 8005h   ;dividend (low)
MOV cx, 100h    ;divisor
DIV cx          ; AX= 0008h, DX=0005h
```

64-bit / 32-bit Division

```
MOV edx, 0h     ;dividend (high)
MOV eax, 8005h  ;dividend (low)
MOV ecx, 100h   ;divisor
DIV ecx         ;EAX= 0008h, EDX=0005h
```

Divide Overflow Error

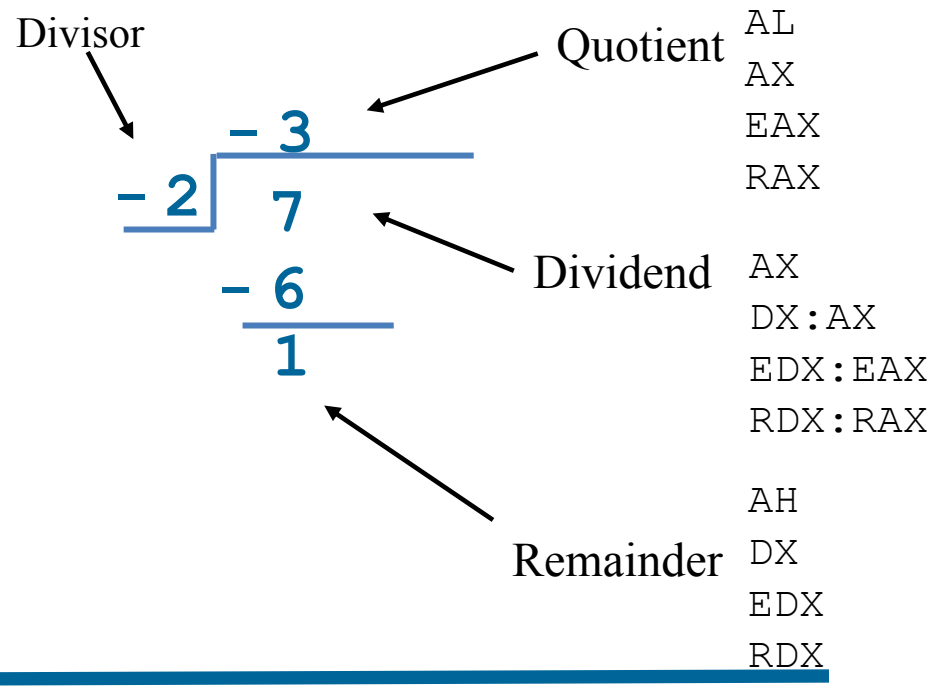
```
MOV dx, 0060h  ;dividend (high)
MOV ax, 0000h  ;dividend (low)
MOV cx, 10h    ;divisor
DIV cx         ;Quotient 60000h
                cannot fit in AX
```

Assembling & Executing x86_64 Program

Signed Division

IDIV divisor					
Operand Size	**Dividend**	**Divisor**	**Quotient**	**Remainder**	**Quotient Range**
Word/byte	AX	r/m8	AL	AH	-2^8 to $+2^8 - 1$
Doubleword/word	DX:AX	r/m16	AX	DX	-2^{16} to $+2^{16} - 1$
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	-2^{31} to $+2^{31} - 1$
128-bit/64-bit	RDX:RAX	r/m64	RAX	RDX	-2^{63} to $+2^{63} - 1$
Status Flags are undefined for division instruction					

```
MOV ax, 0007h ; dividend
MOV bl, feh   ; divisor (-2)
IDIV bl ; AL=fdh or -3, AH=01h
```



Sign Extension Instructions

CBW: Convert byte to word instruction extends the sign bit of AL into the AH register

```
MOV al, d0h        ; -8
CBW                ; AH:AL = FF D0h
```

CWD: Convert word to double word instruction extends the sign bit of AX into the DX register

```
MOV ax, FFD0h      ; -8
CWD                ; DX:AX = FFFF  FFD0h
```

CDQ: Convert double word to quad word instruction extends the sign bit of EAX into the EDX register

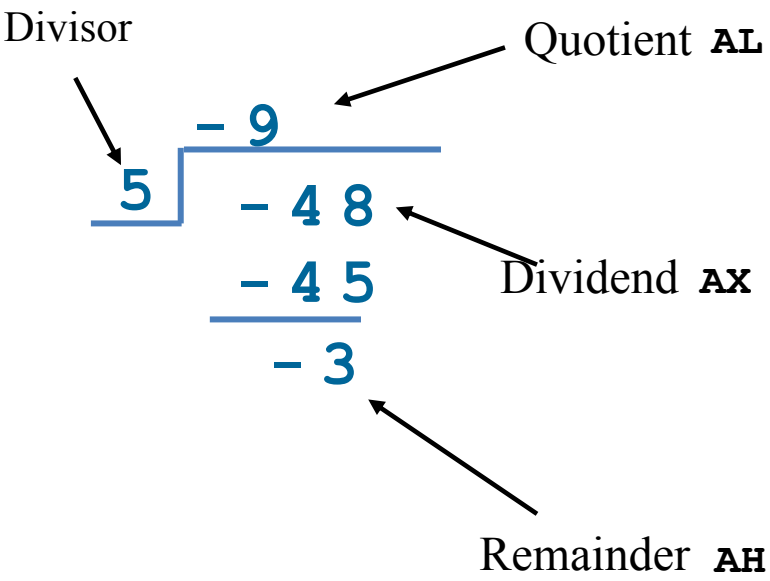
```
MOV eax, FFFFFFD0h   ; -8
CDQ                  ; EDX:EAX = FFFFFFFF  FFFFFFD0h
```

Signed Division (cont…)

IDIV divisor					
Operand Size	**Dividend**	**Divisor**	**Quotient**	**Remainder**	**Quotient Range**
Word/byte	AX	r/m8	AL	AH	-2^8 to $+2^8 - 1$
Doubleword/word	DX:AX	r/m16	AX	DX	-2^{16} to $+2^{16} - 1$
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	-2^{31} to $+2^{31} - 1$
128-bit/64-bit	RDX:RAX	r/m64	RAX	RDX	-2^{63} to $+2^{63} - 1$
Status Flags are undefined for division instruction					

16-bit / 8-bit Division

```
MOV al, -48 ; dividend
CBW ; extend sign bit of al into ah
MOV bl, 5   ; divisor
IDIV bl ; AL=f7h or -9, AH=fdh or -3
```

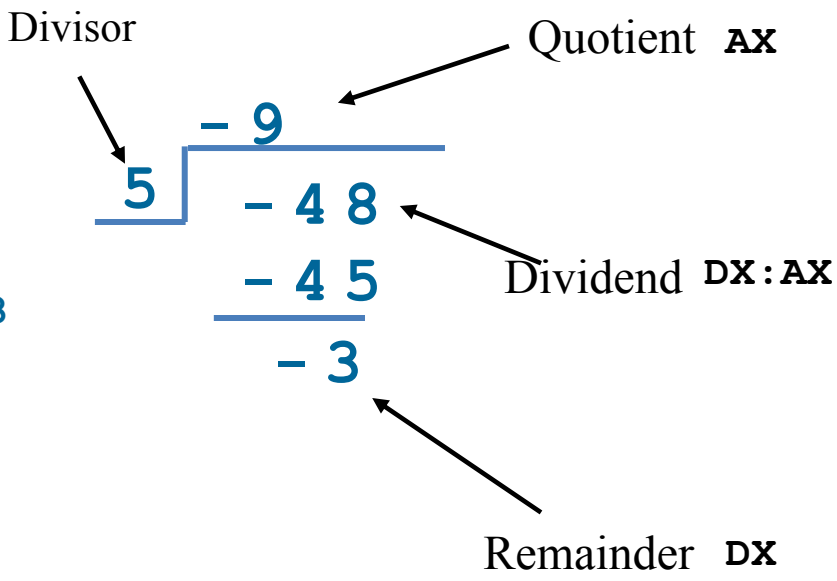


Signed Division (cont…)

IDIV divisor					
Operand Size	**Dividend**	**Divisor**	**Quotient**	**Remainder**	**Quotient Range**
Word/byte	AX	r/m8	AL	AH	-2^8 to $+2^8 - 1$
Doubleword/word	DX:AX	r/m16	AX	DX	-2^{16} to $+2^{16} - 1$
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	-2^{31} to $+2^{31} - 1$
128-bit/64-bit	RDX:RAX	r/m64	RAX	RDX	-2^{63} to $+2^{63} - 1$
Status Flags are undefined for division instruction					

32-bit / 16-bit Division

```
MOV ax, -48 ; dividend
CWD  ;extend sign bit of ax into dx
MOV bx, 5   ; divisor
IDIV bx ; AX=fff7h or -9, DX=fffdh -3
```

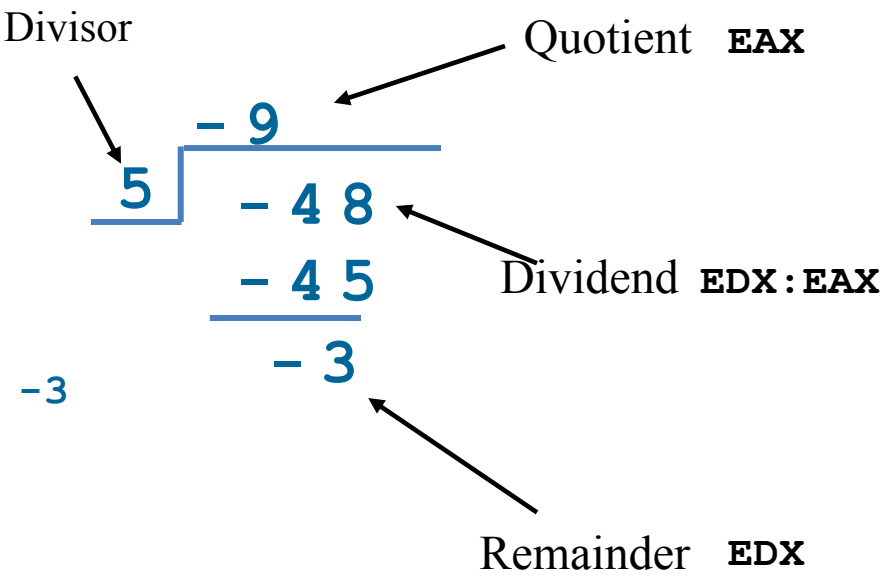


Signed Division (cont…)

IDIV divisor					
Operand Size	**Dividend**	**Divisor**	**Quotient**	**Remainder**	**Quotient Range**
Word/byte	AX	r/m8	AL	AH	-2^{8} to $+2^{8} - 1$
Doubleword/word	DX:AX	r/m16	AX	DX	-2^{16} to $+2^{16} - 1$
Quadword/doubleword	EDX:EAX	r/m32	EAX	EDX	-2^{31} to $+2^{31} - 1$
128-bit/64-bit	RDX:RAX	r/m64	RAX	RDX	-2^{63} to $+2^{63} - 1$
Status Flags are undefined for division instruction					

64-bit / 32-bit Division

```
MOV eax, -48  ; dividend
CDQ  ; extend sign bit of eax into edx
MOV ebx, 5   ; divisor
IDIV ebx
   ; EAX=fffffff7h or -9, EDX=fffffffdh -3
```



Assembling & Executing x86_64 Program

Things To Do

Coming to office hours does NOT mean you are academically week!