Week 9

Chapter 7: Integer Arithmetic

What's Next

- Shift and Rotate Instructions
- Shift and Rotate Applications
- **Multiplication and Division Instructions**
- Extended Addition and Subtraction
- ASCII and Unpacked Decimal Arithmetic
- Packed Decimal Arithmetic

Multiplication and Division Instructions

- MUL Instruction
- IMUL Instruction
- DIV Instruction
- Signed Integer Division
- CBW, CWD, CDQ Instructions
- IDIV Instruction
- Implementing Arithmetic Expressions

MUL Instruction

- In 32-bit mode, MUL (unsigned multiply) instruction multiplies an 8-, 16-, or 32-bit operand by either AL, AX, or EAX.

- The instruction formats are:

```
MUL r/m8
MUL r/m16
MUL r/m32
```

Table 7-2 MUL Operands.

Multiplicand	Multiplier	Product
AL	reg/mem8	AX
AX	reg/mem16	DX:AX
EAX	reg/mem32	EDX:EAX

64-Bit MUL Instruction

- In 64-bit mode, MUL (unsigned multiply) instruction multiplies a 64-bit operand by RAX, producing a 128-bit product.

- The instruction formats are:

  `MUL r/m64`

Example:

```
mov rax,0FFFF0000FFFF0000h
mov rbx,2
mul rbx        ; RDX:RAX = 0000000000000001FFFE0001FFFE0000
```

MUL Examples

100h * 2000h, using 16-bit operands:

```
.data
val1 WORD 2000h
val2 WORD 100h
.code
mov ax,val1
mul val2        ; DX:AX = 00200000h, CF=1
```

> The Carry flag indicates whether or not the upper half of the product contains significant digits.

12345h * 1000h, using 32-bit operands:

```
mov eax,12345h
mov ebx,1000h
mul ebx         ; EDX:EAX = 0000000012345000h, CF=0
```

Your turn . . .

What will be the hexadecimal values of DX, AX, and the Carry flag after the following instructions execute?

```
mov ax,1234h
mov bx,100h
mul bx
```

DX = 0012h, AX = 3400h, CF = 1

Your turn . . .

What will be the hexadecimal values of EDX, EAX, and the Carry flag after the following instructions execute?

```
mov eax,00128765h
mov ecx,10000h
mul ecx
```

EDX = 00000012h, EAX = 87650000h, CF = 1

IMUL Instruction

- IMUL (signed integer multiply) multiplies an 8-, 16-, or 32-bit signed operand by either AL, AX, or EAX
- Preserves the sign of the product by sign-extending it into the upper half of the destination register

Example: multiply 48 * 4, using 8-bit operands:

```
mov  al,48
mov  bl,4
imul bl          ; AX = 00C0h, OF=1
```

OF=1 because AH is not a sign extension of AL.

Using IMUL in 64-Bit Mode

- You can use 64-bit operands. In the two-operand format, a 64-bit register or memory operand is multiplied against RDX
  - 128-bit product produced in RDX:RAX
- The three-operand format produces a 64-bit product in RAX

```
.data
multiplicand QWORD -16
.code
imul rax, multiplicand, 4 ; RAX = FFFFFFFFFFFFFFC0 (-64)
```

IMUL Examples

Multiply 4,823,424 * –423:

```
mov eax,4823424
mov ebx,-423
imul ebx           ; EDX:EAX = FFFFFFFF86635D80h, OF=0
```

OF=0 because EDX is a sign extension of EAX.

Your turn . . .

What will be the hexadecimal values of DX, AX, and the Carry flag after the following instructions execute?

```
mov ax,8760h
mov bx,100h
imul bx
```

DX = FF87h, AX = 6000h, OF = 1

DIV Instruction

- The DIV (unsigned divide) instruction performs 8-bit, 16-bit, and 32-bit division on unsigned integers
- A single operand is supplied (register or memory operand), which is assumed to be the divisor
- Instruction formats:

```
DIV reg/mem8
DIV reg/mem16
DIV reg/mem32
```

Default Operands:

Dividend	Divisor	Quotient	Remainder
AX	*r/m8*	AL	AH
DX:AX	*r/m16*	AX	DX
EDX:EAX	*r/m32*	EAX	EDX

DIV Examples

Divide 8003h by 100h, using 16-bit operands:

```
mov dx,0          ; clear dividend, high
mov ax,8003h      ; dividend, low
mov cx,100h       ; divisor
div cx            ; AX = 0080h, DX = 3
```

Same division, using 32-bit operands:

```
mov edx,0         ; clear dividend, high
mov eax,8003h     ; dividend, low
mov ecx,100h      ; divisor
div ecx           ; EAX = 00000080h, DX = 3
```

64-Bit DIV Example

Divide 000001080000000033300020h by 00010000h:

```
.data
dividend_hi QWORD 00000108h
dividend_lo QWORD 33300020h
divisor QWORD 00010000h

.code
mov rdx, dividend_hi
mov rax, dividend_lo
div divisor                    ; RAX = quotient
                               ; RDX = remainder
```

quotient: 0108000000003330h
remainder: 0000000000000020h

Your turn . . .

What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

```
mov dx,0087h
mov ax,6000h
mov bx,100h
div bx
```

DX = 0000h, AX = 8760h

Your turn . . .

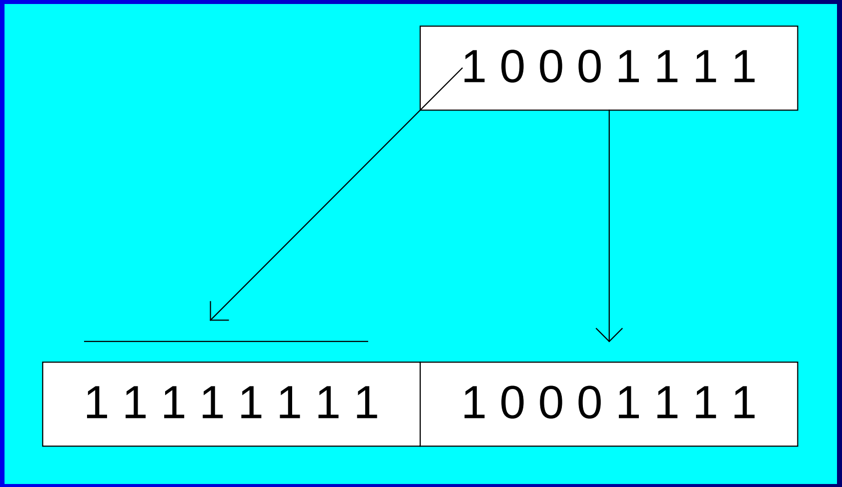
What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

```
mov dx,0087h
mov ax,6002h
mov bx,10h
div bx
```

Divide Overflow

Signed Integer Division (IDIV)

- Signed integers must be sign-extended before division takes place
  - fill high byte/word/doubleword with a copy of the low byte/word/doubleword's sign bit
- For example, the high byte contains a copy of the sign bit from the low byte:



CBW, CWD, CDQ Instructions

- The CBW, CWD, and CDQ instructions provide important sign-extension operations:
  - CBW (convert byte to word) extends AL into AH
  - CWD (convert word to doubleword) extends AX into DX
  - CDQ (convert doubleword to quadword) extends EAX into EDX
- Example:

```
.data
dwordVal SDWORD -101    ; FFFFFF9Bh
.code
mov eax,dwordVal
cdq                     ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

IDIV Instruction

- IDIV (signed divide) performs signed integer division
- Same syntax and operands as DIV instruction

Example: 8-bit division of –48 by 5

```
mov  al,-48
cbw                 ; extend AL into AH
mov  bl,5
idiv bl             ; AL = -9,  AH = -3
```

IDIV Examples

Example: 16-bit division of –48 by 5

```
mov  ax,-48
cwd                  ; extend AX into DX
mov  bx,5
idiv bx              ; AX = -9,  DX = -3
```

Example: 32-bit division of –48 by 5

```
mov  eax,-48
cdq                  ; extend EAX into EDX
mov  ebx,5
idiv ebx             ; EAX = -9,  EDX = -3
```

Your turn . . .

What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

```
mov  ax,0FDFFh            ; -513
cwd
mov  bx,100h
idiv bx
```

DX = FFFFh (–1), AX = FFFEh (–2)

Unsigned Arithmetic Expressions

- Some good reasons to learn how to implement integer expressions:
  - Learn how do compilers do it
  - Test your understanding of MUL, IMUL, DIV, IDIV
  - Check for overflow (Carry and Overflow flags)

Example: **var4 = (var1 + var2) * var3**

```
; Assume unsigned operands
mov  eax,var1
add  eax,var2          ; EAX = var1 + var2
mul  var3              ; EAX = EAX * var3
jc   TooBig            ; check for carry
mov  var4,eax          ; save product
```

Signed Arithmetic Expressions (1 of 2)

Example: **eax = (-var1 * var2) + var3**

```
mov  eax,var1
neg  eax
imul var2
jo   TooBig          ; check for overflow
add  eax,var3
jo   TooBig          ; check for overflow
```

Example: **var4 = (var1 * 5) / (var2 – 3)**

```
mov  eax,var1            ; left side
mov  ebx,5
imul ebx                 ; EDX:EAX = product
mov  ebx,var2            ; right side
sub  ebx,3
idiv ebx                 ; EAX = quotient
mov  var4,eax
```

Signed Arithmetic Expressions (2 of 2)

Example: **`var4 = (var1 * -5) / (-var2 % var3);`**

```
mov  eax,var2              ; begin right side
neg  eax
cdq                        ; sign-extend dividend
idiv var3                  ; EDX = remainder
mov  ebx,edx               ; EBX = right side
mov  eax,-5                ; begin left side
imul var1                  ; EDX:EAX = left side
idiv ebx                   ; final division
mov  var4,eax              ; quotient
```

Sometimes it's easiest to calculate the right-hand term of an expression first.

Your turn . . .

Implement the following expression using signed 32-bit integers:

```
eax = (ebx * 20) / ecx
```

```
mov eax,20
imul ebx
idiv ecx
```

Your turn . . .

Implement the following expression using signed 32-bit integers. Save and restore ECX and EDX:

```
eax = (ecx * edx) / eax
```

```
push  edx
push  eax                 ; EAX needed later
mov   eax,ecx
imul  edx                 ; left side: EDX:EAX
pop   ebx                 ; saved value of EAX
idiv  ebx                 ; EAX = quotient
pop   edx                 ; restore EDX, ECX
```

Your turn . . .

Implement the following expression using signed 32-bit integers. Do not modify any variables other than var3:

var3 = (var1 * -var2) / (var3 – ebx)

```
mov  eax,var1
mov  edx,var2
neg  edx
imul edx                 ; left side: EDX:EAX
mov  ecx,var3
sub  ecx,ebx
idiv ecx                 ; EAX = quotient
mov  var3,eax
```