Topic 7
Lecture 7b
Integer Arithmetic

CSCI 150

Assembly Language / Machine Architecture

Prof. Dominick Atanasio

What's Next (2 of 5)

- Shift and Rotate Instructions
- Shift and Rotate Applications
- **Multiplication and Division Instructions**
- Extended Addition and Subtraction

Multiplication and Division Instructions

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

MUL Instruction

- 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:

```
section .data
val1:   dw 2000h
val2:   dw 100h
section .text
mov ax, [val1]
mul  word [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 . . . (6 of 16)

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 . . . (7 of 16)

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.

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.

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
```

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

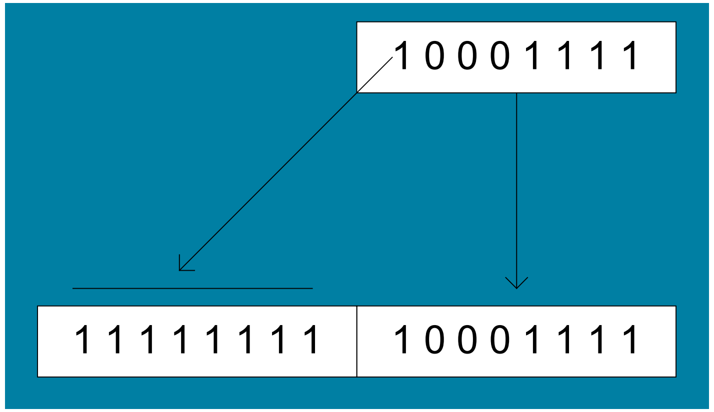
What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

```
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:



Your turn . . . (10 of 16)

Write the correct instructions for performing the following division operation of two signed 16-bit values in a program called idiv.asm

8544h / 32h

What is the quotient and remainder?

Answer: EAX = 0xfd8c   EDX = 0xffec

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:

```
section .data
    ddVal:   dd  -101              ; FFFFFF9Bh
section .text
    mov eax,  [ddVal]
    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
```

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   too_big               ; 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 dword [var2]
jo   too_big                ; check for overflow
add  eax, [var3]
jo   too_big                ; 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
```

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

```
mov  eax, [var2]          ; begin right side
neg  eax
cdq                       ; sign-extend dividend
idiv dword [var3]         ; EDX = remainder
mov  ebx, edx             ; EBX = right side
mov  eax, -5              ; begin left side
imul dword [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 . . . (12 of 16)

Implement the following expression using signed 32-bit integers:

eax = (ebx * 20) / ecx

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

eax = (ecx * edx) / eax

Your turn . . . (14 of 16)

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

var3 = (var1 * -var2) / (var3 – ebx)

What's Next (3 of 5)

- Shift and Rotate Instructions
- Shift and Rotate Applications
- Multiplication and Division Instructions
- **Extended Addition and Subtraction**

Extended Addition and Subtraction

- ADC Instruction
- Extended Precision Addition
- SBB Instruction
- Extended Precision Subtraction

The instructions in this section do not apply to 64-bit mode programming.

Extended Precision Addition

- Adding two operands that are longer than the a register size (32 bits).
  - Virtually no limit to the size of the operands
- The arithmetic must be performed in steps
  - The Carry value from each step is passed on to the next step.

ADC Instruction

- ADC (add with carry) instruction adds both a source operand and the contents of the Carry flag to a destination operand.
- Operands are binary values
  - Same syntax as ADD, SUB, etc.
- Example
  - Add two 32-bit integers (FFFFFFFFh + FFFFFFFFh), producing a 64-bit sum in EDX:EAX:

```
mov edx, 0
mov eax, 0FFFFFFFFh
add eax, 0FFFFFFFFh
adc edx, 0                    ;EDX:EAX = 00000001FFFFFFFEh
```

Extended Addition Example

- Task: Add 1 to EDX:EAX
  - Starting value of EDX:EAX: 00000000 FFFFFFFFh
  - Add the lower 32 bits first, setting the Carry flag.
  - Add the upper 32 bits, and include the Carry flag.

```
mov edx,0               ; set upper half
mov eax,0FFFFFFFFh      ; set lower half
add eax,1               ; add lower half
adc edx,0               ; add upper half
```

EDX:EAX = 00000001 00000000

SBB Instruction

- The SBB (subtract with borrow) instruction subtracts both a source operand and the value of the Carry flag from a destination operand.
- Operand syntax:
  - Same as for the ADC instruction

Extended Subtraction Example

- Task: Subtract 1 from EDX:EAX
  - Starting value of EDX:EAX: 0000000100000000h
  - Subtract the lower 32 bits first, setting the Carry flag.
  - Subtract the upper 32 bits, and include the Carry flag.

```
mov edx, 1          ; set upper half
mov eax, 0          ; set lower half
sub eax, 1          ; subtract lower half
sbb edx, 0          ; subtract upper half
```

EDX:EAX = 00000000 FFFFFFFF

49 20 68 6f 70 65 20 79 6f 75 20 61 72 65 20 65 6e 6a 6f 79 69 6e 67 20 74 68 65 20 63 6c 61 73 73