Integer Arithmetic

Computer Organization
&
Assembly Language Programming

Dr Adnan Gutub

aagutub 'at' uqu.edu.sa

[Adapted from slides of Dr. Kip Irvine: Assembly Language for Intel-Based Computers]
Most Slides contents have been arranged by Dr Muhamed Mudawar & Dr Aiman El-Maleh from Computer Engineering Dept. at KFUPM

Outline

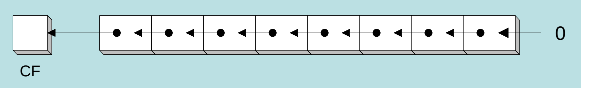
- ❖ Shift and Rotate Instructions
- ❖ Shift and Rotate Applications
- ❖ Multiplication and Division Instructions
- ❖ Translating Arithmetic Expressions
- ❖ Decimal String to Number Conversions

dịch trái logic

SHL Instruction

❖ SHL is the Shift Left instruction

- ✧ Performs a logical left shift on the destination operand
- ✧ Fills the lowest bit with zero
- ✧ The last bit shifted out from the left becomes the Carry Flag



CF mang bit vừa mới bị đẩy ra

❖ Operand types for SHL:

```
SHL reg,imm8
SHL mem,imm8
SHL reg,CL
SHL mem,CL
```

The shift count is either:

8-bit immediate ***imm8***, or

stored in register ***CL***

Only least sig. 5 bits used

Fast Multiplication

Shifting left 1 bit multiplies a number by 2

dl=5
dl<<1

```
mov dl,5
shl dl,1
```

Before:	0 0 0 0 0 1 0 1	= 5
After:	0 0 0 0 1 0 1 0	= 10

Shifting left *n* bits multiplies the operand by 2^n

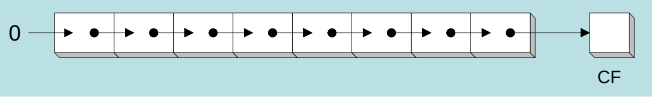
For example, $5 * 2^2 = 20$

```
mov dl,5   ; DL = 00000101b
shl dl,2   ; DL = 00010100b = 20, CF = 0
```

dịch phải logic

SHR Instruction

- SHR is the Shift Right instruction
  - Performs a logical right shift on the destination operand
  - The highest bit position is filled with a zero
  - The last bit shifted out from the right becomes the Carry Flag
  - SHR uses the same instruction format as SHL

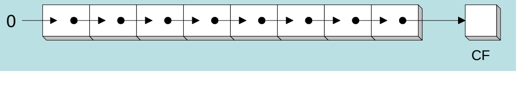


- Shifting right n bits divides the operand by 2^n

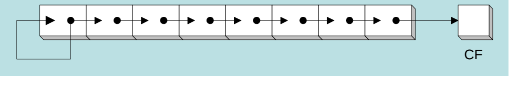
```
mov dl,80   ; DL = 01010000b
shr dl,1    ; DL = 00101000b = 40, CF = 0
shr dl,2    ; DL = 00001010b = 10, CF = 0
```


Logical versus Arithmetic Shifts

- Logical Shift
  - Fills the newly created bit position with zero



- Arithmetic Shift
  - Fills the newly created bit position with a copy of the sign bit
  - Applies only to Shift Arithmetic Right (SAR)



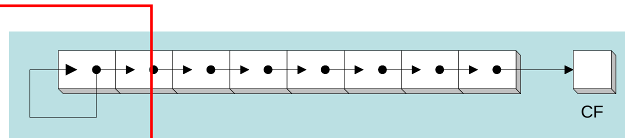
dịch trái/phải toán học

SAL and SAR Instructions

- SAL: Shift Arithmetic Left is identical to SHL

SALvà SHL là như nhau

- SAR: Shift Arithmetic Right
  - Performs a right arithmetic shift on the destination operand



- SAR preserves the number's sign SAR bảo toàn dấu

```
mov dl,-80    ; DL = 10110000b
sar dl,1      ; DL = 11011000b = -40, CF = 0
sar dl,2      ; DL = 11110110b = -10, CF = 0
```


Your Turn . . .

Indicate the value of AL and CF after each shift

```
mov al,6Bh        ; al = 01101011b
shr al,1          ; al = 00110101b = 35h, CF = 1
shl al,3          ; al = 10101000b = A8h, CF = 1
mov al,8Ch        ; al = 10001100b
sar al,1          ; al = 11000110b = C6h, CF = 0
sar al,3          ; al = 11111000b = F8h, CF = 1
```

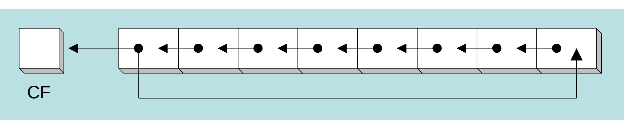
Effect of Shift Instructions on Flags

- The CF is the last bit shifted
- The OF is defined for single bit shift only
  - It is 1 if the sign bit changes
- The ZF, SF and PF are affected according to the result
- The AF is unaffected

xoay trái thông qua CF

ROL Instruction

- ROL is the Rotate Left instruction
  - Rotates each bit to the left, according to the count operand
  - Highest bit is copied into the Carry Flag and into the Lowest Bit
- No bits are lost




```
mov al,11110000b
rol al,1          ; AL = 11100001b, CF = 1
mov dl,3Fh        ; DL = 00111111b
rol dl,4          ; DL = 11110011b = F3h, CF = 1
```

dịch phải thông qua CF

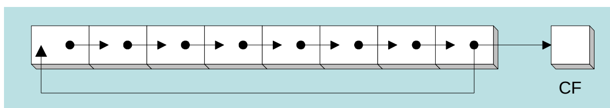
ROR Instruction

❖ ROR is the Rotate Right instruction

✧ Rotates each bit to the right, according to the count operand

✧ Lowest bit is copied into the Carry flag and into the highest bit

❖ No bits are lost



3F=0011 1111
F3=1111 0011

```
mov al,11110000b
ror al,1              ; AL = 01111000b, CF = 0
mov dl,3Fh            ; DL = 00111111b
ror dl,4              ; DL = F3h, CF = 1
```


xoay trái có dùng CF

RCL Instruction

thực hiện trước

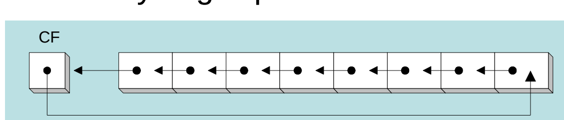
❖ RCL is the Rotate Carry Left instruction

✧ Rotates each bit to the left, according to the count operand

✧ Copies the Carry flag to the least significant bit

✧ Copies the most significant bit to the Carry flag

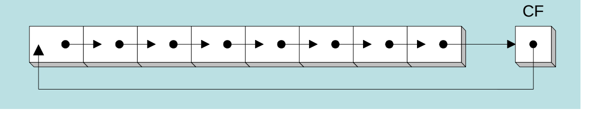
❖ As if the carry flag is part of the destination operand




```
clc               ; clear carry, CF = 0
mov bl,88h        ; BL = 10001000b
rcl bl,1          ; CF = 1, BL = 00010000b
rcl bl,2          ; CF = 0, BL = 01000010b
```

RCR Instruction

- RCR is the Rotate Carry Right instruction
  - Rotates each bit to the right, according to the count operand
  - Copies the Carry flag to the most significant bit
  - Copies the least significant bit to the Carry flag
- As if the carry flag is part of the destination operand




```
stc              ; set carry, CF = 1
mov  ah,11h      ; AH = 00010001b
rcr  ah,1        ; CF = 1, AH = 10001000b
rcr  ah,3        ; CF = 0, AH = 00110001b
```


Effect of Rotate Instructions on Flags

- The CF is the last bit shifted
- The OF is defined for single bit rotates only
  - It is 1 if the sign bit changes
- The ZF, SF, PF and AF are unaffected

SHLD Instruction

- SHLD is the Shift Left Double instruction
- Syntax: **SHLD** ***destination, source, count***
  - Shifts a *destination* operand a given *count* of bits to the left
- The rightmost bits of *destination* are filled by the leftmost bits of the *source* operand
- The *source* operand is not modified
- Operand types:

```
SHLD reg/mem16, reg16, imm8/CL
SHLD reg/mem32, reg32, imm8/CL
```

lấy cái bit tận cùng bên trái của source vào các bit tận cùng bên phải của des

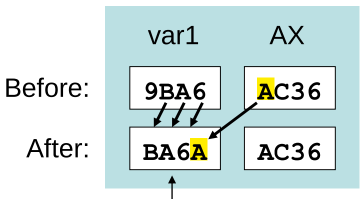
SHLD Example

Shift variable **var1** 4 bits to the left

Replace the lowest 4 bits of **var1** with the high 4 bits of AX

```
.data
var1 WORD 9BA6h
.code
mov  ax, 0AC36h
shld var1, ax, 4
```

destination source count



destination A=1010

Only the *destination* is modified, not the *source*

SHRD Instruction

- SHRD is the Shift Right Double instruction
- Syntax: **SHRD *destination, source, count***
  - Shifts a *destination* operand a given *count* of bits to the right
- The leftmost bits of *destination* are filled by the rightmost bits of the *source* operand
- The *source* operand is not modified
- Operand types:

```
SHRD reg/mem16, reg16, imm8/CL
SHRD reg/mem32, reg32, imm8/CL
```

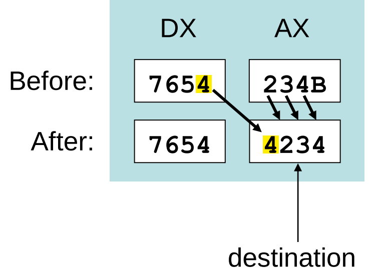

SHRD Example

Shift AX 4 bits to the right

Replace the highest 4 bits of AX with the low 4 bits of DX

```
mov  ax,234Bh
mov  dx,7654h
shrd ax, dx, 4
```

destination source count



4=0100

Only the *destination* is modified, not the *source*

Your Turn . . .

Indicate the values (in hex) of each destination operand

```
mov   ax,7C36h
mov   dx,9FA6h
shld  dx,ax,4      ; DX = FA67h
shrd  ax,dx,8      ; AX = 677Ch
```


Next . . .

- ❖ Shift and Rotate Instructions
- ❖ Shift and Rotate Applications
- ❖ Multiplication and Division Instructions
- ❖ Translating Arithmetic Expressions
- ❖ Decimal String to Number Conversions

Shifting Bits within an Array

- ❖ Sometimes, we need to shift all bits within an array
  - ✧ Example: moving a bitmapped image from one screen to another
- ❖ Task: shift an array of bytes 1 bit right

```
.data
    ArraySize  EQU 100
    array BYTE ArraySize DUP(9Bh)
.code
    mov ecx, ArraySize
    mov esi, 0
    clc                       ; clear carry flag
L1:
    rcr array[esi], 1         ; propagate the carry flag
    inc esi                   ; does not modify carry
    loop L1                   ; does not modify carry
```

	[0]	[1]	[2]		[99]
array before	9B	9B	9B	. . .	9B
array after	4D	CD	CD	. . .	CD

Binary Multiplication

- ❖ You know that SHL performs multiplication efficiently
  - ✧ When the multiplier is a power of 2
- ❖ You can factor any binary number into powers of 2
  - ✧ Example: multiply EAX by 36
    - ▪ Factor 36 into (4 + 32) and use distributive property of multiplication
  - ✧ EAX * 36 = EAX * (4 + 32) = EAX * 4 + EAX * 32

```
mov ebx, eax        ; EBX = number
shl eax, 2          ; EAX = number * 4
shl ebx, 5          ; EBX = number * 32
add eax, ebx        ; EAX = number * 36
```

Your Turn . . .

Multiply EAX by 26, using shifting and addition instructions

Hint: 26 = 2 + 8 + 16

```
mov   ebx, eax               ; EBX = number
shl   eax, 1                 ; EAX = number * 2
shl   ebx, 3                 ; EBX = number * 8
add   eax, ebx               ; EAX = number * 10
shl   ebx, 1                 ; EBX = number * 16
add   eax, ebx               ; EAX = number * 26
```

Multiply EAX by 31, Hint: 31 = 32 – 1

```
mov   ebx, eax               ; EBX = number
shl   eax, 5                 ; EAX = number * 32
sub   eax, ebx               ; EAX = number * 31
```


Convert Number to Binary String

Task: Convert Number in EAX to an ASCII Binary String

Receives: EAX = Number
ESI = Address of binary string

Returns: String is filled with binary characters '0' and '1'

```
ConvToBinStr PROC USES ecx esi
    mov   ecx,32
L1: rol   eax,1
    mov   BYTE PTR [esi],'0'
    jnc   L2
    mov   BYTE PTR [esi],'1'
L2: inc   esi
    loop  L1
    mov   BYTE PTR [esi], 0
    ret
ConvToBinStr ENDP
```

Rotate left most significant bit of EAX into the Carry flag; If CF = 0, append a '0' character to a string; otherwise, append a '1'; Repeat in a loop 32 times for all bits of EAX.

Convert Number to Hex String

Task: Convert EAX to a Hexadecimal String pointed by ESI

Receives: EAX = Number, ESI= Address of hex string

Returns: String pointed by ESI is filled with hex characters '0' to 'F'

```
ConvToHexStr PROC  USES ebx ecx esi
    mov   ecx, 8              ; 8 iterations, why?
L1: rol   eax, 4              ; rotate upper 4 bits
    mov   ebx, eax
    and   ebx, 0Fh            ; keep only lower 4 bits
    mov   bl,  HexChar[ebx]   ; convert to a hex char
    mov   [esi], bl           ; store hex char in string
    inc   esi
    loop L1                   ; loop 8 times
    mov   BYTE PTR [esi], 0   ; append a null byte
    ret
HexChar BYTE "0123456789ABCDEF"
ConvToHexStr ENDP
```


Isolating a Bit String

- MS-DOS date packs the year, month, & day into 16 bits
  - Year is relative to 1980

	DH		DL	
	0 0 1 0 0 1 1	0	0 1 1 0 1 0 1 0	
Field:	Year	Month		Day
Bit numbers:	9-15	5-8		0-4

In this example:

Day = 10
Month = 3
Year = 1980 + 19
Date = March 10, 1999

Isolate the Month field:

```
mov ax,dx             ; Assume DX = 16-bit MS-DOS date
shr ax,5              ; shift right 5 bits
and al,00001111b      ; clear bits 4-7
mov month,al          ; save in month variable
```

Next . . .

- Shift and Rotate Instructions
- Shift and Rotate Applications
- Multiplication and Division Instructions
- Translating Arithmetic Expressions
- Decimal String to Number Conversions

nhân số không dấu

MUL Instruction

- The MUL instruction is used for unsigned multiplication
- Multiplies 8-, 16-, or 32-bit operand by AL, AX, or EAX
- The instruction formats are:

```
MUL r/m8      ; AX      = AL  * r/m8
MUL r/m16     ; DX:AX   = AX  * r/m16
MUL r/m32     ; EDX:EAX = EAX * r/m32
```

Multiplicand	Multiplier	Product
AL	*r/m8*	AX
AX	*r/m16*	DX:AX
EAX	*r/m32*	EDX:EAX

MUL Examples

Example 1: Multiply 16-bit var1 (2000h) * var2 (100h)

nếu viết thêm dòng này thì chỉ nhân AX*VAR3

DX=0020 (khác 0); AX=0000

```
.data
var1 WORD 2000h
var2 WORD 100h
.code
mov ax,var1
mul var2     ; DX:AX = 00200000h, CF = OF = 1
```

mul var3 ; DX:AX <--- AX*VAR3

The Carry and Overflow flags are set if upper half of the product is non-zero

Example 2: Multiply EAX (12345h) * EBX (1000h) 32bit

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

Solution

DX = 0012h, AX = 3400h, CF = 1
vì DX!=0

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
```

Solution

EDX = 00000012h,

EAX = 87650000h, CF = OF = 1

nhân số có dấu

IMUL Instruction

- The IMUL instruction is used for signed multiplication
  - Preserves the sign of the product by sign-extending it
- One-Operand formats, as in MUL

```
IMUL r/m8        ; AX       = AL  * r/m8
IMUL r/m16       ; DX:AX    = AX  * r/m16
IMUL r/m32       ; EDX:EAX  = EAX * r/m32
```

- Two-Operand formats:

```
IMUL r16, r16/m16/imm8/imm16
IMUL r32, r32/m32/imm8/imm32
```

- Three-Operand formats:

```
IMUL r16, r16/m16, imm8/imm16
IMUL r32, r32/m32, imm8/imm32
```

The Carry and Overflow flags are set if the upper half of the product is not a sign extension of the lower half

IMUL Examples

48=01001000

- Multiply AL = 48 by BL = 4

hệ 10

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

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

- Your Turn: What will be DX, AX and OF ?

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

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

Two and Three Operand Formats

```
.data
wval  SWORD -4
dval  SDWORD 4
.code
mov   ax, -16
mov   bx, 2
imul  bx, ax                ; BX  = BX * AX     = -32
imul  bx, 2                 ; BX  = BX * 2      = -64
imul  bx, wval              ; BX  = BX * wval   = 256
imul  bx, 5000              ; OF  = CF = 1
mov   edx,-16
imul  edx,dval              ; EDX = EDX * dval = -64
imul  bx, wval,-16          ; BX  = wval * -16 =  64
imul  ebx,dval,-16          ; EBX = dval * -16 = -64
imul  eax,ebx,2000000000    ; OF  = CF = 1
```


DIV Instruction

- The DIV instruction is used for unsigned division
- A single operand (divisor) is supplied
  - Divisor is an 8-bit, 16-bit, or 32-bit register or memory
  - Dividend is implicit and is either AX, DX:AX, or EDX:EAX
- The instruction formats are:

```
DIV r/m8
DIV r/m16
DIV r/m32
```

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

DIV Examples

Divide AX = 8003h by CX = 100h

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

Your turn: what will be the hexadecimal values of DX and AX after the following instructions execute?

```
mov dx,0087h
mov ax,6023h
mov bx,100h
div bx
```

Solution: DX = 0023h, AX = 8760h

Divide Overflow

- Divide Overflow occurs when …
  - Quotient cannot fit into the destination operand, or when
  - Dividing by Zero
- Divide Overflow causes a CPU interrupt
  - The current program halts and an error dialog box is produced
- Example of a Divide Overflow

(DX_AX):BX

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

16bit

```
Divide overflow:
Quotient = 87600h
Cannot fit in AX
```

00876002:10
=87600
(5*4=20bit)

Signed Integer Division

- Signed integers must be sign-extended before division
  - Fill high byte, word, or double-word with a copy of the sign bit
- CBW, CWD, and CDQ instructions
  - Provide important sign-extension operations before division
  - CBW: Convert Byte to Word, sign-extends AL into AH 1byte->2bytes
  - CWD: Convert Word to Double, sign-extends AX into DX 2bytes->4bytes
  - CDQ: Convert Double to Quad, sign-extends EAX into EDX 4bytes->8bytes
- Example:

```
mov ax, 0FE9Bh          ; AX = -357
cwd                     ; DX:AX = FFFFFF9Bh
```

= - 357

cặp thanh ghi DX:AX

chia số có dấu

IDIV Instruction

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

```
IDIV r/m8
IDIV r/m16
IDIV r/m32
```

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

- Example: divide eax (-503) by ebx (10)

```
mov   eax, -503
cdq
mov   ebx, 10
idiv ebx         ; EAX = -50, EDX = -3
```

All status flags are undefined after executing DIV and IDIV

-48/-5=9
(-48)-9*(-50)= -3
|-3|<|-5|

IDIV Examples

Example: Divide DX:AX (-48) by BX (-5)

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

Example: Divide EDX:EAX (48) by EBX (-5)

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


Next . . .

- ❖ Shift and Rotate Instructions
- ❖ Shift and Rotate Applications
- ❖ Multiplication and Division Instructions
- ❖ Translating Arithmetic Expressions
- ❖ Decimal String to Number Conversions

Translating Arithmetic Expressions

- ❖ Some good reasons to translate arithmetic expressions
  - ✧ Learn how compilers do it
  - ✧ Test your understanding of MUL, IMUL, DIV, and IDIV
  - ✧ Check for Carry and Overflow flags
- ❖ Two Types of Arithmetic Expressions
  - ✧ Unsigned arithmetic expressions
    - Unsigned variables and values are used only
    - Use MUL and DIV for unsigned multiplication and division
  - ✧ Signed arithmetic expressions
    - Signed variables and values
    - Use IMUL and IDIV for signed multiplication and division

Unsigned Arithmetic Expressions

- ❖ Example: **var4 = (var1 + var2) * var3**
- ❖ All variables are 32-bit unsigned integers
- ❖ Translation:

```
    mov   eax, var1
    add   eax, var2       ; EAX = var1 + var2
    jc    tooBig          ; check for carry
    mul   var3            ; EAX = EAX * var3
    jc    tooBig          ; check for carry
    mov   var4, eax       ; save result
    jmp   next
tooBig:
    . . .                 ; display error message
next:
```

Signed Arithmetic Expressions

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

```
mov  eax, var1
neg  eax
imul var2               ; signed multiplication
jo   tooBig             ; check for overflow
add  eax, var3
jo   tooBig             ; check for overflow
mov  var4, eax          ; save result
```

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

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


Your Turn . . .

Translate: **var5 = (var1 * -var2)/(var3 – var4)**

Assume signed 32-bit integers

```
mov  eax, var1
mov  edx, var2
neg  edx
imul edx                ; EDX:EAX = product
mov  ecx, var3
sub  ecx, var4
idiv ecx                ; EAX = quotient
mov  var5, eax
```

Next . . .

- ❖ Shift and Rotate Instructions
- ❖ Shift and Rotate Applications
- ❖ Multiplication and Division Instructions
- ❖ Translating Arithmetic Expressions
- ❖ Decimal String to Number Conversions

Convert Decimal String to Number

Task: Convert decimal string pointed by ESI to a number

Receives: ESI = address of decimal string

Returns: EAX = number in binary format

Algorithm:

Start by initializing EAX to 0

For each decimal character in string (example: "1083")

Move one decimal character of string into EDX

Convert EDX to digit (0 to 9): **`EDX = EDX – '0'`**

Compute: **`EAX = EAX * 10 + EDX`**

Repeat until end of string (NULL char)

Convert Decimal String - cont'd

```
; Assumes: String should contain only decimal chars
;          String should not be empty
;          Procedure does not detect invalid input
;          Procedure does not skip leading spaces

ConvDecStr PROC USES edx esi
    mov   eax, 0               ; Initialize EAX
L1: imul  eax, 10              ; EAX = EAX * 10
    movzx edx, BYTE PTR [esi]  ; EDX = '0' to '9'
    sub   edx, '0'             ; EDX =  0  to  9
    add   eax, edx             ; EAX = EAX*10 + EDX
    inc   esi                  ; point at next char
    cmp   BYTE PTR [esi],0     ; NULL byte?
    jne   L1
    ret                        ; return
ConvDecStr ENDP
```


Convert Number to Decimal String

Task: Convert Number in EAX to a Decimal String

Receives: EAX = Number, ESI = String Address

Returns: String is filled with decimal characters '0' to '9'

Algorithm: Divide EAX by 10 (Example: EAX = 1083)

```
mov EBX, 10   ; divisor    = EBX = 10
mov EDX, 0    ; dividend   = EDX:EAX
div EBX       ; EDX (rem) = 3, EAX = 108
add dl, '0'   ; DL = '3'
```

Repeat division until EAX becomes 0

Remainder chars are computed backwards: '3', '8', '0', '1'

Store characters in reverse order in string pointed by ESI

Convert to Decimal String - cont'd

```
ConvToDecStr PROC
    pushad                    ; save all since most are used
    mov  ecx, 0               ; Used to count decimal digits
    mov  ebx, 10              ; divisor = 10
L1: mov  edx, 0               ; dividend = EDX:EAX
    div  ebx                  ; EDX = remainder = 0 to 9
    add  dl, '0'              ; convert DL to '0' to '9'
    push dx                   ; save decimal character
    inc  ecx                  ; and count it
    cmp  eax, 0
    jnz  L1                   ; loop back if EAX != 0
L2: pop  dx                   ; pop in reverse order
    mov  [esi], dl            ; store decimal char in string
    inc  esi
    loop L2
    mov  BYTE PTR [esi], 0    ; Terminate with a NULL char
    popad                     ; restore all registers
    ret                       ; return
ConvToDecStr ENDP
```


Summary

- Shift and rotate instructions
  - Provide finer control over bits than high-level languages
  - Can shift and rotate more than one bit left or right
  - SHL, SHR, SAR, SHLD, SHRD, ROL, ROR, RCL, RCR
  - Shifting left by n bits is a multiplication by 2^n
  - Shifting right does integer division (use SAR to preserve sign)
- MUL, IMUL, DIV, and IDIV instructions
  - Provide signed and unsigned multiplication and division
  - One operand format: one of the operands is always implicit
  - Two and three operand formats for IMUL instruction only
  - CBW, CDQ, CWD: extend AL, AX, and EAX for signed division