Computer Organization & Assembly Languages

Integer Arithmetic

Pu-Jen Cheng

Adapted from the slides prepared by Kip Irvine for the book, Assembly Language for Intel-Based Computers, 5th Ed.

Chapter Overview

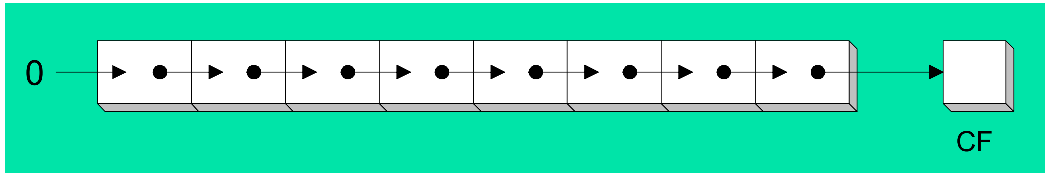
- **Shift and Rotate Instructions**
- Shift and Rotate Applications
- Multiplication and Division Instructions
- Extended Addition and Subtraction
- ASCII and Unpacked Decimal Arithmetic
- Packed Decimal Arithmetic

Shift and Rotate Instructions

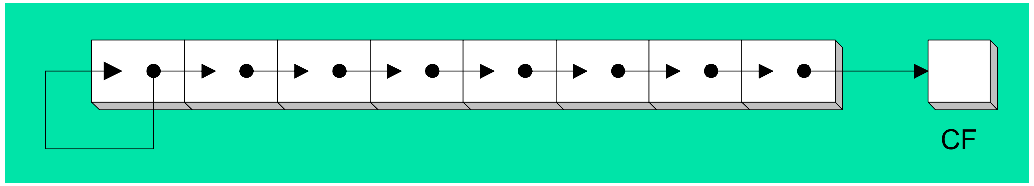
- Logical vs Arithmetic Shifts
- SHL Instruction
- SHR Instruction
- SAL and SAR Instructions
- ROL Instruction
- ROR Instruction
- RCL and RCR Instructions
- SHLD/SHRD Instructions

Logical vs Arithmetic Shifts

- A logical shift fills the newly created bit position with zero:

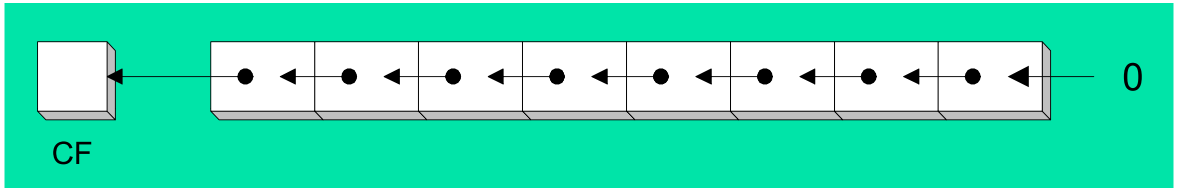


- An arithmetic shift fills the newly created bit position with a copy of the number's sign bit:



SHL Instruction

- The SHL (shift left) instruction performs a logical left shift on the destination operand, filling the lowest bit with 0.



- Operand types for SHL: SHL *destination,count*

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

(Same for all shift and rotate instructions)

Fast Multiplication

Shifting left 1 bit multiplies a number by 2

```
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

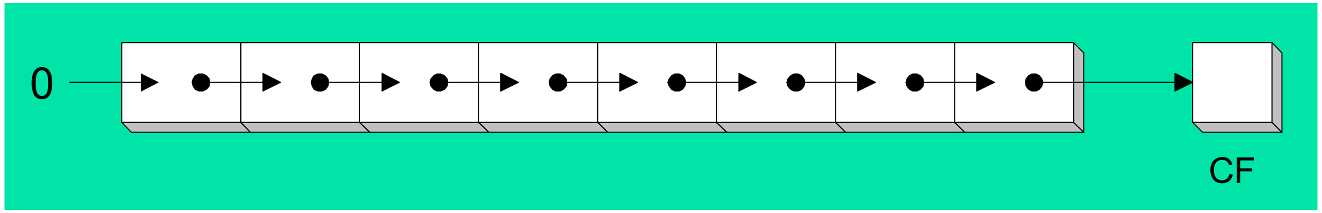
Shifting left *n* bits multiplies the operand by 2^n

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

```
mov dl,5
shl dl,2                    ; DL = 20
```

SHR Instruction

- The SHR (shift right) instruction performs a logical right shift on the destination operand. The highest bit position is filled with a zero.

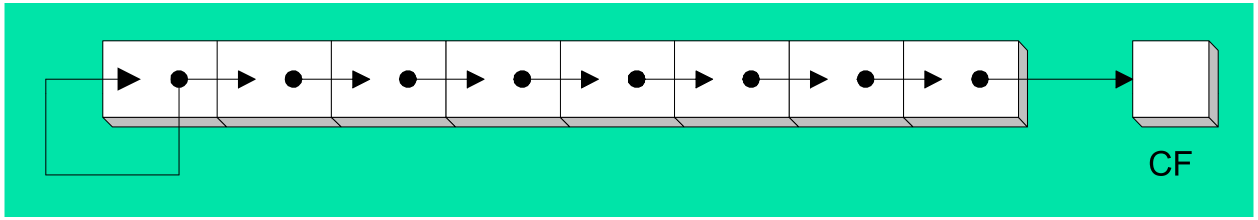


Shifting right *n* bits divides the operand by 2^n

```
mov dl,80
shr dl,1                              ; DL = 40
shr dl,2                              ; DL = 10
```

SAL and SAR Instructions

- SAL (shift arithmetic left) is identical to SHL.
- SAR (shift arithmetic right) performs a right arithmetic shift on the destination operand.



An arithmetic shift preserves the number's sign.

```
mov dl,-80
sar dl,1                        ; DL = -40
sar dl,2                        ; DL = -10
```

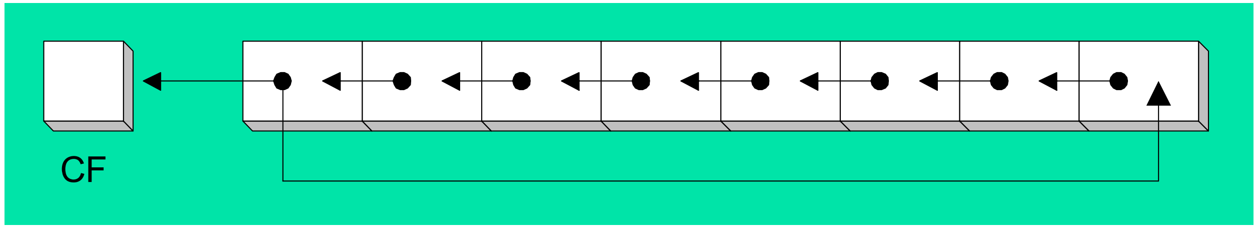
Your turn . . .

Indicate the hexadecimal value of AL after each shift:

```
mov al,6Bh                    ; 01101011
shr al,1                      a.  35h
shl al,3                      b.  A8h
mov al,8Ch                    ; 10001100
sar al,1                      c.  C6h
sar al,3                      d.  F8h
```

ROL Instruction

- ROL (rotate) shifts each bit to the left
- The highest bit is copied into both the Carry flag and into the lowest bit
- No bits are lost

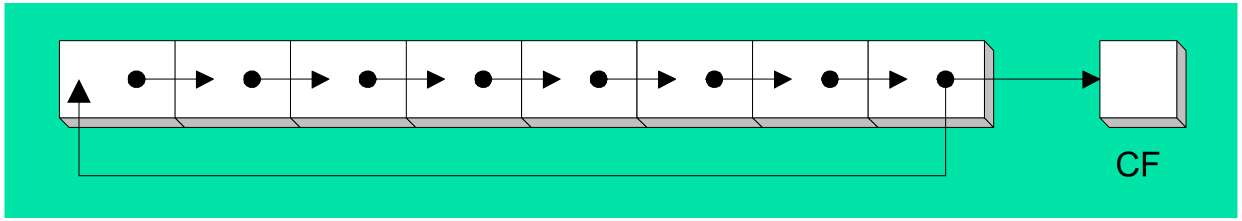



```
mov al,11110000b
rol al,1                        ; AL = 11100001b

mov dl,3Fh
rol dl,4                        ; DL = F3h
```

ROR Instruction

- ROR (rotate right) shifts each bit to the right
- The lowest bit is copied into both the Carry flag and into the highest bit
- No bits are lost




```
mov al,11110000b
ror al,1                    ; AL = 01111000b

mov dl,3Fh
ror dl,4                    ; DL = F3h
```

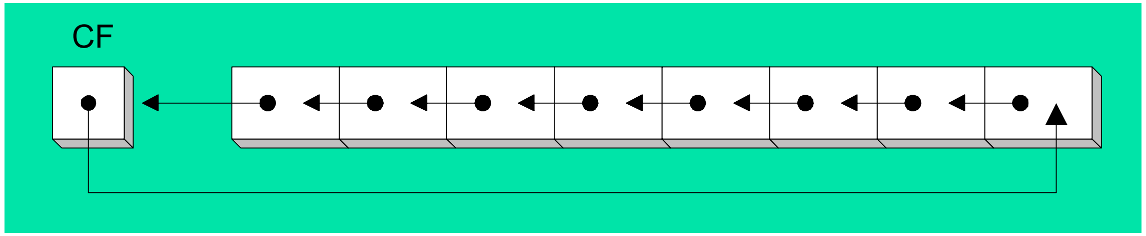
Your turn . . .

Indicate the hexadecimal value of AL after each rotation:

```
mov al,6Bh                      ; 01101011
ror al,1                       a. B5h
rol al,3                       b. ADh
```

RCL Instruction

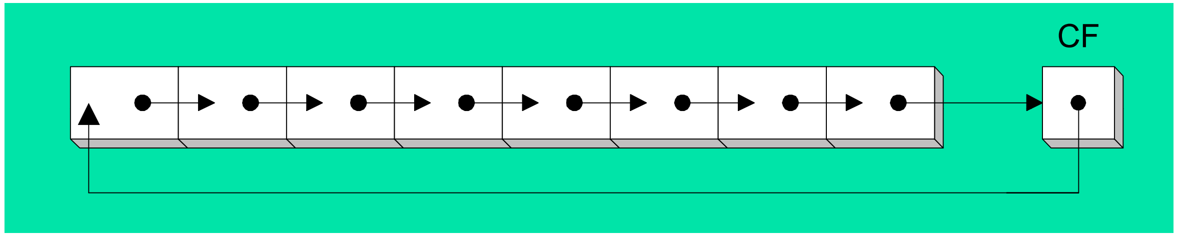
- RCL (rotate carry left) shifts each bit to the left
- Copies the Carry flag to the least significant bit
- Copies the most significant bit to the Carry flag




```
clc                          ; CF = 0
mov bl,88h                   ; CF,BL = 0 10001000b
rcl bl,1                     ; CF,BL = 1 00010000b
rcl bl,1                     ; CF,BL = 0 00100001b
```

RCR Instruction

- RCR (rotate carry right) shifts each bit to the right
- Copies the Carry flag to the most significant bit
- Copies the least significant bit to the Carry flag




```
stc                       ; CF = 1
mov ah,10h                ; CF,AH = 1 00010000b
rcr ah,1                  ; CF,AH = 0 10001000b
```

Your turn . . .

Indicate the hexadecimal value of AL after each rotation:

```
stc
mov al,6Bh
rcr al,1                          a.  B5h
rcl al,3                          b.  AEh
```

SHLD Instruction

- Shifts a destination operand a given number of bits to the left
- The bit positions opened up by the shift are filled by the most significant bits of the source operand
- The source operand is not affected
- Syntax:

  SHLD *destination, source, count*

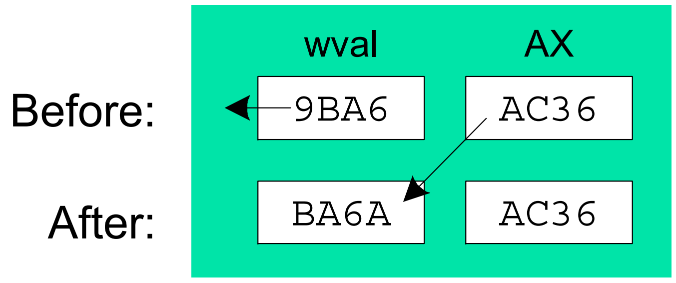
- Operand types:

```
SHLD reg16/32, reg16/32, imm8/CL
SHLD mem16/32, reg16/32, imm8/CL
```

SHLD Example

Shift wval 4 bits to the left and replace its lowest 4 bits with the high 4 bits of AX:

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



SHRD Instruction

- Shifts a destination operand a given number of bits to the right
- The bit positions opened up by the shift are filled by the least significant bits of the source operand
- The source operand is not affected
- Syntax:

    SHRD *destination, source, count*

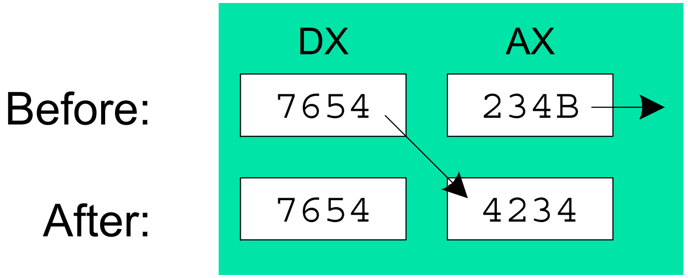
- Operand types:

```
SHRD reg16/32, reg16/32, imm8/CL
SHRD mem16/32, reg16/32, imm8/CL
```

SHRD Example

Shift AX 4 bits to the right and replace its highest 4 bits with the low 4 bits of DX:

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



Your turn . . .

Indicate the hexadecimal values of each destination operand:

```
mov  ax,7C36h
mov  dx,9FA6h
shld dx,ax,4                ; DX = FA67h
shrd dx,ax,8                ; DX = 36FAh
```

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

Shift and Rotate Applications

- Shifting Multiple Doublewords
- Binary Multiplication
- Displaying Binary Bits
- Isolating a Bit String

Shifting Multiple Doublewords

- Programs sometimes need to shift all bits within an array, as one might when moving a bitmapped graphic image from one screen location to another.
- The following shifts an array of 3 doublewords 1 bit to the right

```
.data
ArraySize = 3
array DWORD ArraySize DUP(99999999h)       ; 1001 1001...
.code
mov esi,0
shr array[esi + 8],1          ; high dword
rcr array[esi + 4],1          ; middle dword, include Carry
rcr array[esi],1              ; low dword, include Carry
```

Binary Multiplication

- We already know that SHL performs unsigned multiplication efficiently when the multiplier is a power of 2.
- You can factor any binary number into powers of 2.
  - For example, to multiply EAX * 36, factor 36 into 32 + 4 and use the distributive property of multiplication to carry out the operation:

```
EAX * 36
= EAX * (32 + 4)
= (EAX * 32)+(EAX * 4)
```

```
mov eax,123
mov ebx,eax
shl eax,5            ; mult by 2^5
shl ebx,2            ; mult by 2^2
add eax,ebx
```

Your turn . . .

Multiply AX by 26, using shifting and addition instructions. *Hint:* 26 = 16 + 8 + 2.

```
mov ax,2                        ; test value

mov dx,ax
shl dx,4                        ; AX * 16
push dx                         ; save for later
mov dx,ax
shl dx,3                        ; AX * 8
shl ax,1                        ; AX * 2
add ax,dx                       ; AX * 10
pop dx                          ; recall AX * 16
add ax,dx                       ; AX * 26
```

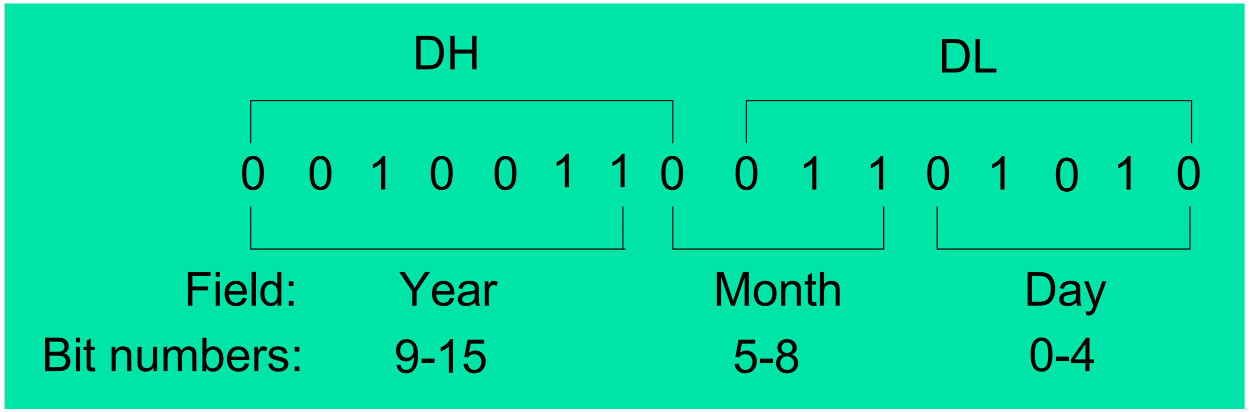
Displaying Binary Bits

Algorithm: Shift MSB into the Carry flag; If CF = 1, append a "1" character to a string; otherwise, append a "0" character. Repeat in a loop, 32 times.

```
.data
buffer BYTE 32 DUP(0),0
.code
    mov ecx,32
    mov esi,OFFSET buffer
L1: shl eax,1
    mov BYTE PTR [esi],'0'
    jnc L2
    mov BYTE PTR [esi],'1'
L2: inc esi
    loop L1
```

Isolating a Bit String

- The MS-DOS file date field packs the year, month, and day into 16 bits:



Isolating a Bit String (cont.)

```
mov al,dl          ; make a copy of DL
and al,00011111b   ; clear bits 5-7
mov day,al         ; save in day variable
```

```
mov ax,dx          ; make a copy of DX
shr ax,5           ; shift right 5 bits
and al,00001111b   ; clear bits 4-7
mov month,al       ; save in month variable
```

```
mov al,dh          ; make a copy of DH
shr al,1           ; shift right 1 bit
mov ah,0           ; clear AH to 0
add ax,1980        ; year is relative to 1980
mov year,ax        ; save in year
```

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

- The 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

Implied operands:

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

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          ; 110000
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.

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 *r/m8*

DIV *r/m16*

DIV *r/m32*

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
```

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 . . .

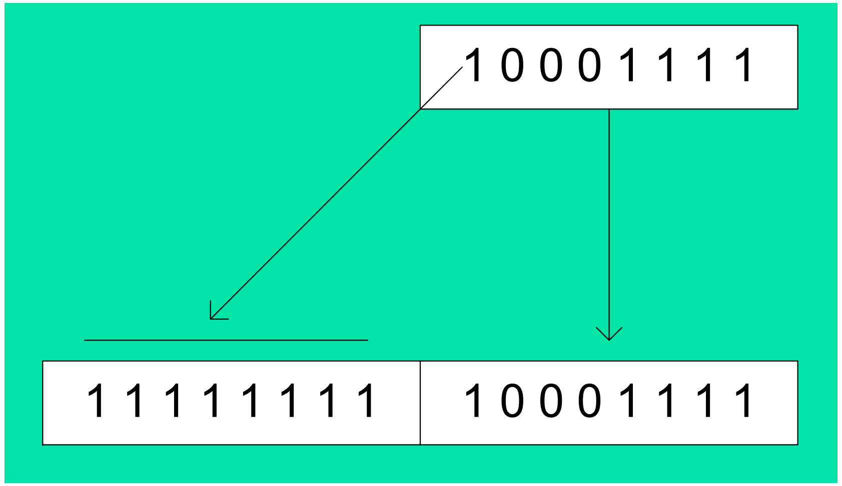
What will be the hexadecimal values of DX and AX after the following instructions execute? Or, if divide overflow occurs, you can indicate that as your answer:

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

Divide Overflow

Signed Integer Division

- 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:

```
mov eax,0FFFFFF9Bh  ; (-101)
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                     ; 256
idiv bx
```

DX = FFFFh (–1), AX = FFFEh (–2)

Divide Overflow

- *Divide overflow* happens when the quotient is too large to fit into the destination.

```
mov ax, 1000h
mov bl, 10h
div bl
```

It causes a CPU interrupt and halts the program. (divided by zero cause similar results)

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
```

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

Extended Addition and Subtraction

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

Extended Precision Addition

- Adding two operands that are longer than the computer's word 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: 00000000FFFFFFFFh
  - 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
```

What's Next

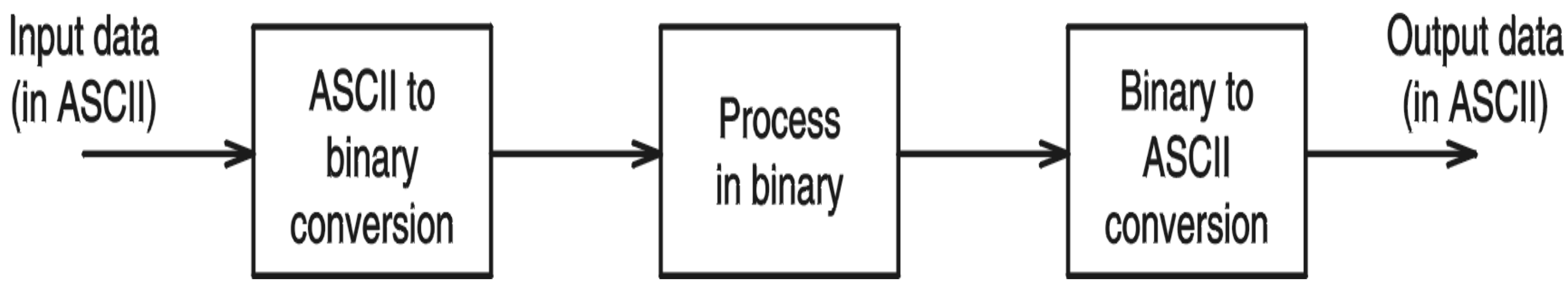
- Shift and Rotate Instructions
- Shift and Rotate Applications
- Multiplication and Division Instructions
- Extended Addition and Subtraction
- **ASCII and UnPacked Decimal Arithmetic**
- Packed Decimal Arithmetic

ASCII and Packed Decimal Arithmetic

- Binary Coded Decimal
- ASCII Decimal
- AAA Instruction
- AAS Instruction
- AAM Instruction
- AAD Instruction
- Packed Decimal Integers
- DAA Instruction
- DAS Instruction

Representation of Numbers

- Numbers are in ASCII form
  - when received from keyboard
  - when sending to the display
- Binary form is efficient to process numbers internally
- Requires conversion between these two number representations



Advantages of ASCII Arithmetic

- Avoid conversion overheads between two formats
- Avoid danger of the round-off errors that occur with floating-point numbers

Representations of Decimal Numbers

- ASCII representation
- BCD representation
  - Unpacked BCD
  - Packed BCD

ASCII Decimal

- A number using ASCII Decimal representation stores a single ASCII digit in each byte
  - For example, 5,678 is stored as the following sequence of hexadecimal bytes:

35	36	37	38

Binary-Coded Decimal

- Binary-coded decimal (BCD) integers use 4 binary bits to represent each decimal digit
- A number using unpacked BCD representation stores a decimal digit in the lower four bits of each byte
  - For example, 5,678 is stored as the following sequence of hexadecimal bytes:

05	06	07	08

AAA Instruction

- The AAA (ASCII adjust after addition) instruction adjusts the binary result of an ADD or ADC instruction. It makes the result in AL consistent with ASCII decimal representation.
  - The Carry value, if any ends up in AH
- Example: Add '8' and '2'

```
mov ah,0
mov al,'8'              ; AX = 0038h
add al,'2'              ; AX = 006Ah
aaa                     ; AX = 0100h (adjust result)
or  ax,3030h            ; AX = 3130h = '10'
```

Processing ASCII Numbers

ASCII addition

Example 1	Example 2
`34H = 00110100B`	`36H = 00110110B`
`35H = 00110101B`	`37H = 00110111B`
`69H = 01101001B`	`6DH = 01101101B`
Should be 09H	Should be 13H
Ignore 6	Ignore 6 and add 9 to D

- The AAA instruction performs these adjustments to the byte in AL register

AAS Instruction

- The AAS (ASCII adjust after subtraction) instruction adjusts the binary result of an SUB or SBB instruction. It makes the result in AL consistent with ASCII decimal representation.
  - It places the Carry value, if any, in AH
- Example: Subtract '9' from '8'

```
mov ah,0
mov al,'8'          ; AX = 0038h
sub al,'9'          ; AX = 00FFh
aas                 ; AX = FF09h, CF=1
pushf               ; save CF
or al,30h           ; AL = '9`
popf                ; restore CF
```

$$\begin{array}{r} 68 \\ -29 \\ \hline 39 \end{array}$$

AAM Instruction

- The AAM (ASCII adjust after multiplication) instruction adjusts the binary result of a MUL instruction. The multiplication must have been performed on unpacked BCD numbers.

```
mov bl,05h              ; first operand
mov al,06h              ; second operand
mul bl                  ; AX = 001Eh
aam                     ; AX = 0300h
```

AAD Instruction

- The AAD (ASCII adjust before division) instruction adjusts the unpacked BCD dividend in AX before a division operation

```
.data
quotient  BYTE ?
remainder BYTE ?
.code
mov ax,0307h                    ; dividend
aad                             ; AX = 0025h
mov bl,5                        ; divisor
div bl                          ; AX = 0207h
mov quotient,al
mov remainder,ah
```

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**

Packed Decimal Arithmetic

- Packed decimal integers store two decimal digits per byte
  - For example, 12,345,678 can be stored as the following sequence of hexadecimal bytes:

12	34	56	78

Packed decimal is also known as packed BCD.

Good for financial values – extended precision possible, without rounding errors.

DAA Instruction

- The DAA (decimal adjust after addition) instruction converts the binary result of an ADD or ADC operation to packed decimal format.
  - The value to be adjusted must be in AL
  - If the lower digit is adjusted, the Auxiliary Carry flag is set.
  - If the upper digit is adjusted, the Carry flag is set.

DAA Logic

```
If (AL(lo) > 9) or (AuxCarry = 1)
    AL = AL + 6

If (AL(hi) > 9) or Carry = 1
  AL = AL + 60h
```

DAA Examples

- Example: calculate BCD 35 + 48

```
mov al,35h
add al,48h              ; AL = 7Dh
daa                     ; AL = 83h, CF = 0
```

- Example: calculate BCD 35 + 65

```
mov al,35h
add al,65h              ; AL = 9Ah
daa                     ; AL = 00h, CF = 1
```

- Example: calculate BCD 69 + 29

```
mov al,69h
add al,29h              ; AL = 92h
daa                     ; AL = 98h, CF = 0
```

DAS Instruction

- The DAS (decimal adjust after subtraction) instruction converts the binary result of a SUB or SBB operation to packed decimal format.
- The value must be in AL

DAS Logic

```
If (AL(lo) > 9) OR (AuxCarry = 1)
   AL = AL - 6;

If (AL(hi) > 9) or (Carry = 1)
   AL = AL - 60h;
```

DAS Examples (1 of 2)

- Example: subtract BCD 48 – 35

```
mov al,48h
sub al,35h              ; AL = 13h
das                     ; AL = 13h CF = 0
```

- Example: subtract BCD 62 – 35

```
mov al,62h
sub al,35h              ; AL = 2Dh, CF = 0
das                     ; AL = 27h, CF = 0
```

- Example: subtract BCD 32 – 29

```
mov al,32h
sub al,29h              ; AL = 09h, CF = 0
daa                     ; AL = 03h, CF = 0
```

DAS Examples (2 of 2)

- Example: subtract BCD 32 – 39

```
mov al,32h
sub al,39h                ; AL = F9h, CF = 1
das                       ; AL = 93h, CF = 1
```

```
Steps:
AL = F9h
ACF = 1, so subtract 6 from F9h
AL = F3h
F > 9, so subtract 60h from F3h
AL = 93h, CF = 1
```

```
  432
 -139
 ----
  293
```

Summary

- Shift and rotate instructions are some of the best tools of assembly language
  - finer control than in high-level languages
  - SHL, SHR, SAR, ROL, ROR, RCL, RCR
- MUL and DIV – integer operations
  - close relatives of SHL and SHR
  - CBW, CDQ, CWD: preparation for division
- Extended precision arithmetic: ADC, SBB
- ASCII decimal operations (AAA, AAS, AAM, AAD)
- Packed decimal operations (DAA, DAS)