Chapter Overview

• **Shift and Rotate Instructions**
• Shift and Rotate Applications
• Multiplication and Division Instructions
• Extended Addition and Subtraction
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:

  \[
  \begin{align*}
  &\text{SHL } \text{reg}, \text{imm8} \\
  &\text{SHL } \text{mem}, \text{imm8} \\
  &\text{SHL } \text{reg}, \text{CL} \\
  &\text{SHL } \text{mem}, \text{CL}
  \end{align*}
  \]

  (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, \(5 \times 2^2 = 20\)

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

; DL = 40
; 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:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Hex Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mov al, 6Bh</td>
<td></td>
</tr>
<tr>
<td>shr al, 1</td>
<td>a. 35h</td>
</tr>
<tr>
<td>shl al, 3</td>
<td>b. A8h</td>
</tr>
<tr>
<td>mov al, 8Ch</td>
<td></td>
</tr>
<tr>
<td>sar al, 1</td>
<td>c. C6h</td>
</tr>
<tr>
<td>sar al, 3</td>
<td>d. F8h</td>
</tr>
</tbody>
</table>
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

\[
\begin{align*}
\text{mov al, } & \quad \text{11110000}_b \\
\text{ror al, 1} & \quad \text{; AL = 01111000}_b \\
\text{mov dl, } & \quad \text{3Fh} \\
\text{ror dl, 4} & \quad \text{; DL = F3h}
\end{align*}
\]
Your turn . . .

Indicate the hexadecimal value of AL after each rotation:

\[
\begin{array}{ll}
\text{mov al, } 6Bh & \\
\text{ror al, 1} & \text{a. } B5h \\
\text{rol al, 3} & \text{b. } ADh
\end{array}
\]
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:

\[
\begin{array}{l}
\text{stc} \\
\text{mov al, 6Bh} \\
\text{rcr al, 1} \quad \text{a. B5h} \\
\text{rcl al, 3} \quad \text{b. AEh}
\end{array}
\]
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:
  \[ \text{SHLD } \text{destination}, \text{source}, \text{count} \]
• Operand types:

  | SHLD reg16/32, reg16/32, imm8/CL |
  | SHLD mem16/32, reg16/32, imm8/CL  |
SHLD Example

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

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

Before:

\begin{tabular}{c|c}
\texttt{wval} & \texttt{AX} \\
9BA6 & AC36 \\
\end{tabular}

After:

\begin{tabular}{c|c}
\texttt{wval} & \texttt{AX} \\
BA6A & AC36 \\
\end{tabular}
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:
  \[
  \text{SHRD~} \text{destination, source, count}
  \]
- Operand types:
  
<table>
<thead>
<tr>
<th>Operation</th>
<th>Syntax</th>
</tr>
</thead>
</table>

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
```

Before:

<table>
<thead>
<tr>
<th>DX</th>
<th>AX</th>
</tr>
</thead>
<tbody>
<tr>
<td>7654</td>
<td>234B</td>
</tr>
</tbody>
</table>

After:

<table>
<thead>
<tr>
<th>DX</th>
<th>AX</th>
</tr>
</thead>
<tbody>
<tr>
<td>7654</td>
<td>4234</td>
</tr>
</tbody>
</table>
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
Shift and Rotate Applications

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

• To shift an extended integer by dividing it into an array of bytes, words, or double-words first. The common way to store the number using little-endian order.

• The following shifts an array of 3 double-words 1 bit to the right. (With little-endian, the low-order value is at the lowest address.
  – E.g.,; extended integer 111122223333444455556666h

```plaintext
.data
ArraySize = 3
array DWORD ArraySize 55556666h, 33334444h, 11112222h
.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
```

```
66665555  44443333  22221111
```

low dword [esi]  [esi+4]  [esi+8]  High dword
### 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:

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

```assembly
mov eax, 123
mov ebx, eax
shr eax, 5 ; mult by $2^5$
shr 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.

```assembly
.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:

```
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

How to isolate the Month field?

```
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
```
What's Next

• 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

- 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

<table>
<thead>
<tr>
<th>Multiplicand</th>
<th>Multiplier</th>
<th>Product</th>
</tr>
</thead>
<tbody>
<tr>
<td>AL</td>
<td>r/m8</td>
<td>AX</td>
</tr>
<tr>
<td>AX</td>
<td>r/m16</td>
<td>DX:AX</td>
</tr>
<tr>
<td>EAX</td>
<td>r/m32</td>
<td>EDX:EAX</td>
</tr>
</tbody>
</table>

The multiplier and multiplicand are the same size, and the product is twice their size.
MUL Examples

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

```assembly
.data
val1 WORD 2000h
val2 WORD 100h
.code
mov ax, val1
mul val2 ; DX:AX = 00200000h, CF=1
          ; CF=1 because DX is not zero
```

The Carry flag indicates whether or not the upper half of the product contains significant digits. You can use it to decide whether to ignore the upper half of the product.

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

```assembly
mov eax, 12345h
mov ebx, 1000h
mul ebx ; EDX:EAX = 0000000012345000h, CF=0
          ; CF=0 because EDX is 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.
IMUL Examples

Multiply 4,823,424 * −423:

\[
\begin{align*}
\text{mov eax,} & \ 4823424 \\
\text{mov ebx,} & \ -423 \\
\text{imul ebx} & \quad ; \ 	ext{EDX:EAX} = \text{FFFFFF86635D80h, OF}=0
\end{align*}
\]

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

<table>
<thead>
<tr>
<th>Dividend</th>
<th>Divisor</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>AX</td>
<td>r/m8</td>
<td>AL</td>
<td>AH</td>
</tr>
<tr>
<td>DX:AX</td>
<td>r/m16</td>
<td>AX</td>
<td>DX</td>
</tr>
<tr>
<td>EDX:EAX</td>
<td>r/m32</td>
<td>EAX</td>
<td>EDX</td>
</tr>
</tbody>
</table>

Default Operands:
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:

\[
\begin{align*}
\text{mov} & \text{ dx, 0087h} \\
\text{mov} & \text{ ax, 6002h} \\
\text{mov} & \text{ bx, 10h} \\
\text{div} & \text{ bx}
\end{align*}
\]

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:

```assembly
mov eax, 0FFFFFFF9Bh ; (-101)
cdq ; EDX:EAX = FFFFFFFFFFFFFF9Bh
```

Your copy of the book may have an error on page 243: 9Bh equals –101 rather than –65.
IDIV Instruction

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

Example: 8-bit division of \(-48\) by 5

\[
\begin{align*}
\text{mov} &\quad \text{al}, -48 \\
\text{cbw} &\quad \text{; extend AL into AH} \\
\text{mov} &\quad \text{bl}, 5 \\
\text{idiv} &\quad \text{bl} \quad \text{; AL = -9, AH = -3}
\end{align*}
\]
IDIV Examples

Example: 16-bit division of \(-48\) by 5

\begin{verbatim}
    mov ax, -48
    cwd ; extend AX into DX
    mov bx, 5
    idiv bx ; AX = -9, DX = -3
\end{verbatim}

Example: 32-bit division of \(-48\) by 5

\begin{verbatim}
    mov eax, -48
    cdq ; extend EAX into EDX
    mov ebx, 5
    idiv ebx ; EAX = -9, EDX = -3
\end{verbatim}
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: \( \text{var4} = (\text{var1} + \text{var2}) \times \text{var3} \)

; Assume unsigned operands
mov  eax, var1
add  eax, var2      ; EAX = var1 + var2
mul  var3           ; EAX = EAX \times var3
jc    TooBig        ; check for carry
mov  var4, eax      ; save product
Signed Arithmetic Expressions

Example: $eax = (-\text{var1} \times \text{var2}) + \text{var3}$

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

Example: $\text{var4} = (\text{var1} \times 5) / (\text{var2} - 3)$

```
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: \( \text{var4} = (\text{var1} \times -5) / (-\text{var2} \mod \text{var3}); \)

```
mov  eax, var2                  ; begin right side
neg  eax
cdq
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:

\[
eax = (ebx \times 20) / ecx
\]

```assembly
mov eax, 20
imul ebx
idiv ecx
```
Your turn . . .

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

\[
eax = (ecx \times edx) / eax
\]

```
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:

$$\text{var3} = (\text{var1} \times -\text{var2}) / (\text{var3} - \text{ebx})$$

```assembly
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
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:

\[
\begin{align*}
\text{mov} & \quad \text{edx, 0} \\
\text{mov} & \quad \text{eax, 0FFFFFFFFh} \\
\text{add} & \quad \text{eax, 0FFFFFFFFh} \\
\text{adc} & \quad \text{edx, 0} \\
\end{align*}
\]

; EDX:EAX = 00000001FFFFFFFFh
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
```
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)