Chapter 7

Integer Arithmetic

7.1 Introduction 193

7.2 Shift and Rotate Instructions 194
  7.2.1 Logical Shifts and Arithmetic Shifts 194
  7.2.2 SHL Instruction 195
  7.2.3 SHR Instruction 196
  7.2.4 SAL and SAR Instructions 196
  7.2.5 ROL Instruction 197
  7.2.6 ROR Instruction 198
  7.2.7 RCL and RCR Instructions 198
  7.2.8 Signed Overflow 199
  7.2.9 SHLD/SHRD Instructions 199
  7.2.10 Section Review 200

7.3 Shift and Rotate Applications 201
  7.3.1 Shifting Multiple Doublewords 201
  7.3.2 Binary Multiplication 202
  7.3.3 Displaying Binary Bits 202
  7.3.4 Isolating MS-DOS File Date Fields 203
  7.3.5 Section Review 203

7.4 Multiplication and Division Operations 204
  7.4.1 MUL Instruction 204
  7.4.2 IMUL Instruction 205
  7.4.3 Benchmarking Multiplication Operations 207
  7.4.4 DIV Instruction 208
  7.4.5 Signed Integer Division 209
  7.4.6 Implementing Arithmetic Expressions 211
  7.4.7 Section Review 212

7.5 Extended Addition and Subtraction 213
  7.5.1 ADC Instruction 213
  7.5.2 Extended Addition Example 213
  7.5.3 SBB Instruction 214
  7.5.4 Section Review 215
7.1 Introduction  
- Integer Arithmetic  
  - Shift and Rotate Instructions  
  - Multiplication and Division Operations  
  - Extended Addition and Subtraction

7.2 Shift and Rotate Instructions  
7.2.1 Logical Shifts and Arithmetic Shifts  
- A logical shift fills the newly created bit position with zero:

![Logical Shift Diagram]

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

![Arithmetic Shift Diagram]
7.2.2  SHL Instruction

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

- Operands for SHL:
  - SHL reg,imm8
  - SHL mem,imm8
  - SHL reg,CL
  - SHL mem,CL

- Fast Multiplication
  - Shifting left 1 bit multiplies a number by 2
    
    ```
    mov dl,5
    shl dl,1
    
    Before: 0000101 = 5
    After: 0001010 = 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
    ```

7.2.3  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
  ```
7.2.4  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

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

7.2.5  ROL Instruction

- ROL (rotate left) 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.

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

7.2.6  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.

```assembly
mov al, 11110000b
ror al, 1 ; AL = 01111000b
mov dl, 3Fh
ror dl, 4 ; DL = F3h
```
7.2.7 RCL and RCR Instructions

- **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
  ```
### 7.2.9 SHLD/SHRD Instructions

**SHLD (shift left double) 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:
  
  $\text{SHLD destination, source, count}$

- **Operand types:**
  
  $\text{SHLD reg16/32, reg16/32, imm8/CL}$
  $\text{SHLD mem16/32, reg16/32, imm8/CL}$

- 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 (shift right double) 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:
  
  $\text{SHRD destination, source, count}$

- **Operand types:**
  
  $\text{SHLD reg16/32, reg16/32, imm8/CL}$
  $\text{SHLD mem16/32, reg16/32, imm8/CL}$

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

  ```
  .code
  mov ax,234Bh
  mov dx,7654h
  shrd ax,dx,4
  ```
7.3 Shift and Rotate Applications

7.3.1 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.

```
.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
```

- The following shifts an array of 3 doublewords 1 bit to the right

```
mov eax,123
mov ebx,eax
shl eax,5 ; mult by 32 (2^5)
shl ebx,2 ; mult by 4 (2^2)
add eax,ebx
```

7.3.2 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)
```
7.4 Multiplication and Division Operations 204

7.4.1 MUL Instruction 204

- 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:

<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>

- 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
    ```
  - 12345h * 1000h, using 32-bit operands:
    ```
    mov eax,12345h
    mov ebx,1000h
    mul ebx   ; EDX:EAX = 0000000012345000h, CF=0
    ```

7.4.2 IMUL Instruction 205

- 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
- Examples:
  - Multiply 48 * 4, using 8-bit operands
    ```
    mov al,48   ; AL = 30h
    mov bl,4
    imul bl    ; AX = 00C0h, OF=1
    ```
    Note: OF=1 because AH is not a sign extension of AL
  - Multiply 4,823,424 * -423
    ```
    mov eax,4823424
    ```
mov ebx, -423
imul ebx

Note: OF=0 because EDX is a sign extension of EAX

7.4.4 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:

<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>

- 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
7.4.5 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:

![Sign Extension Diagram]

- 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,0FFFFFFF9Bh ; (-101)
cdq              ; EDX:EAX = FFFFFFFF000000000F9Bh
    ```

- IDIV Instruction
  - IDIV (signed divide) performs signed integer division
  - Same syntax and operands as DIV instruction
  - Example 1: 8-bit division of –48 by 5
    ```
mov al,-48
    cbw        ; extend AL into AH
    mov bl,5
    idiv bl      ; AL = -9,  AH = -3
    ```

  - Example 2: 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 3: 32-bit division of –48 by 5
    ```
mov eax,-48
    cdq        ; extend EAX into EDX
    mov ebx,5
    idiv ebx      ; EAX = -9,  EDX = -3
    ```
7.4.6 Implementing Arithmetic Expressions

- 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: \( \text{var4} = (\text{var1} + \text{var2}) \times \text{var3} \)
    
    ```
    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 1: \( \text{eax} = (-\text{var1} \times \text{var2}) + \text{var3} \)
    
    ```
    mov eax, var1
    neg eax
    imul var2
    jo TooBig ; check for overflow
    add eax, var3
    jo TooBig ; check for overflow
    ```
  - Example 2: \( \text{var4} = (\text{var1} \times 5) / (\text{var2} - 3) \)
    
    ```
    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
    ```
  - Example 3: \( \text{var4} = (\text{var1} \times -5) / (-\text{var2} \% \text{var3}) \)
    
    ```
    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
    ```
7.5 Extended Addition and Subtraction

7.5.1 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
  ```

- Example: 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
    ```

7.5.2 Extended Addition Example

- 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

```
TITLE Extended Addition Example (ExtAdd.asm)

; This program calculates the sum of two 64-bit integers.
; Chapter 7 example.
; Last update: 06/01/2006

INCLUDE Irvine32.inc

.data
op1 QWORD 0A2B2A40674981234h
op2 QWORD 08010870000234502h
sum DWORD 3 dup(0FFFFFFFFh) ; = 0000000122C32B0674BB5736

.code
main PROC

  mov  esi,OFFSET op1 ; first operand
  mov  edi,OFFSET op2 ; second operand
  mov  ebx,OFFSET sum ; sum operand
  mov  ecx,2 ; number of doublewords
```

Kuo-pao Yang
call Extended_Add

; Display the sum.
mov eax, sum + 8 ; display high-order dword
call WriteHex
mov eax, sum + 4 ; display middle dword
call WriteHex
mov eax, sum ; display low-order dword
call WriteHex
call Crlf
exit
main ENDP

;--------------------------------------------------------
Extended_Add PROC
;
; Calculates the sum of two extended integers stored
; as an array of doublewords.
; Receives: ESI and EDI point to the two integers,
; EBX points to a variable that will hold the sum, and
; ECX indicates the number of doublewords to be added.
; The sum must be one doubleword longer than the
; input operands.
;--------------------------------------------------------
pushad
clc ; clear the Carry flag
L1: mov eax,[esi] ; get the first integer
    adc eax,[edi] ; add the second integer
    pushfd ; save the Carry flag
    mov [ebx],eax ; store partial sum
    add esi,4 ; advance all 3 pointers
    add edi,4
    add ebx,4
    popfd ; restore the Carry flag
    loop L1 ; repeat the loop
mov dword ptr [ebx],0 ; clear high dword of sum
adc dword ptr [ebx],0 ; add any leftover carry
popad
ret
Extended_Add ENDP
END main
7.5.3  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
- Example: 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