# Chapter 7
# Integer Arithmetic

## 7.1 Chapter Overview

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



## 7.2 Shift and Rotate Instructions

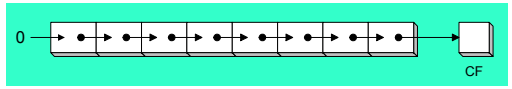
| Instructions | 동작 | 구분 |
|---|---|---|
| SHL dst, count | shift left | logical shift |
| SHR dst, count | shift right | |
| SAL dst, count | shift arithmetic left | arithmetic shift |
| SAR dst, count | shift arithmetic right | |
| ROL dst, count | rotate left | rotate |
| ROR dst, count | rotate right | |
| RCL dst, count | rotate carry left | rotate with carry |
| RCR dst, count | rotate carry right | |
| SHLD dst, src, count | double-precision shift left | double-precision shift |
| SHRD dst, src, count | double-precision shift right | |

- dst는 r/m
- count는 imm8 또는 CL

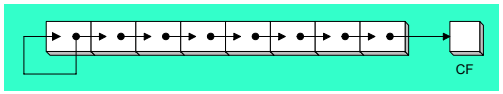


## Logical vs Arithmetic Shifts

- Logical shift vs. Arithmetic shift
  - logical shift = unsigned shift



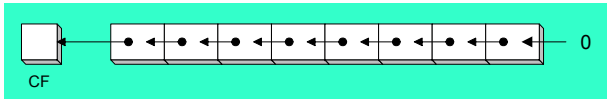

  - arithmetic shift = signed shift (부호bit를 보존함)



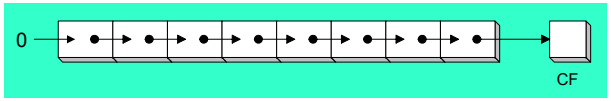

- Example 1-bit shift right
  - unsigned number: 11110000 (240) → **0**1111000 (120)
  - signed number: 11110000 (-16) → **1**1111000 (-8)
- Shift left and Shift right
  - 1-bit shift left: 2를 곱한 것과 같음
  - 1-bit shift right: 2로 나눈 것과 같음

## SHL and SHR Instruction

- logical shift instructions: SHL, SHR
- SHL dst, count




- SHR dst, count




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



## Example

- Fast Multiplication
  - n-bit shift left = multiply by $2^n$
  - $A * 5 = A * (2^2 + 1) = (A << 2) + A$

```
mov dl,A
mov al,dl                ; AL = A
shl dl,2                 ; DL = 4*A
add al,dl                ; AL = 5*A
```

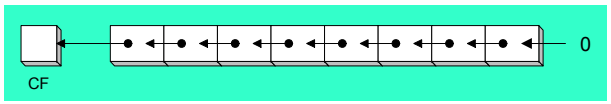
- Division
  - n-bit shift right = divide by $2^n$

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

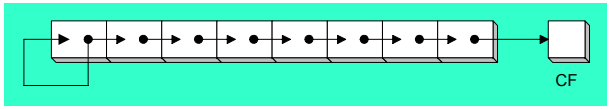


## SAL and SAR Instructions

- Arithmetic shift instructions: SAL, SAR
- SAL dst, count = SHL dst, count




- SAR dst, count



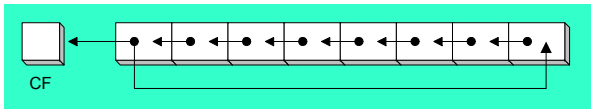

  - 부호 보존 (MSB)

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

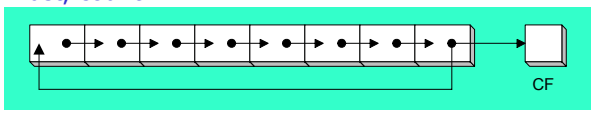


## ROL and ROR Instruction

- ROL dst, count



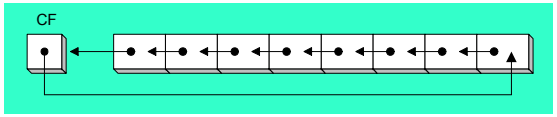

- ROR dst, count



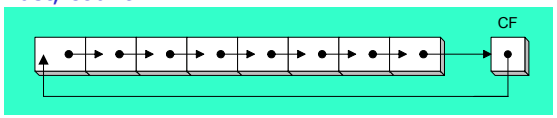

```
mov al,11110000b
rol al,2                 ; AL = 1100 0011b, CF=1

mov dl,3Fh               ; 0011 1111
ror dl,3                 ; DL = F3h(1110 0111), CF=1
```

## RCL and RCR Instruction

- RCL dst, count




- RCR dst, count

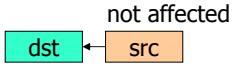



```
clc                     ; CF = 0
mov bl,88h              ; CF,BL = 0 10001000b
rcl bl,1                ; CF,BL = 1 00010000b
mov ah,10h              ; CF,AH = 1 00010000b
rcr ah,1                ; CF,AH = 0 10001000b
```

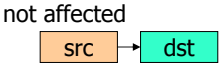


## SHLD and SHRD Instruction

- SHLD dst, src, count
  - shift **dst** by **count** to the left
  - filled LSB of **dst** by MSB of **src**
  - **src** is not affected




- SHRD dst, src, count
  - shift **dst** by **count** to the right
  - filled MSB of **dst** by LSB of **src**
  - **src** is not affected




- Operand format

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

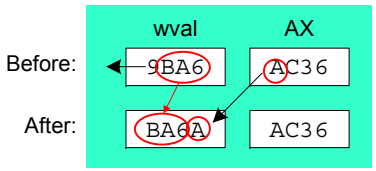
SHRD도 같음



## Example

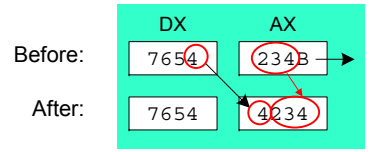
- 변수 wval을 4비트 왼쪽으로 이동하고
  변수 wval의 하위 4비트는 AX의 상위4비트로 채움

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




- AX를 4비트 오른쪽으로 이동하고
  AX의 상위 4비트는 DX의 하위 4비트로 채움

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






## 7.3 Shift and Rotate Applications

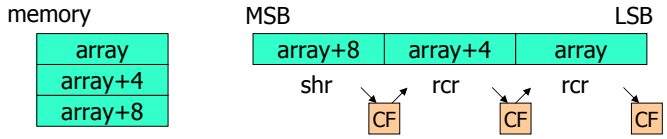
- Shifting Multiple Doublewords

- Binary Multiplication
- Displaying Binary Bits
  (ex) AX = 3Ah → string "00111010"으로 변환
- Isolating a Bit String
  (ex) 01001101 → 특정필드값을 추출 (001)

## Shifting Multiple Doublewords

- 3 doubleword를 1비트씩 오른쪽으로 이동

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




- 3 doubleword를 2비트씩 오른쪽으로 이동 ???



## Binary Multiplication

- EAX ← EAX*36 계산하기
  - EAX*($2^5$ + $2^2$)

```
mov eax,123
mov ebx,eax
add eax,0
shl ebx,2          ; mult by 2^2
add eax,ebx
shl ebx,3          ; mult by 2^5
add eax,ebx
```

- 두 memory operand를 shift를 사용하여 곱셈 수행
  - next slide



```
title multiplication using shift
include irvine32.inc
.data
operand1 dword 25
operand2 dword 2
result   dword ?
message byte 'Overflow',0
.code
main proc
        mov ebx, operand1       ; multiplicand
        mov edx, operand2       ; multiplier
        mov eax, 0
        mov ecx, 32             ; loop counter
L0:     test edx, 1
        jz  L1
        add eax, ebx            ; if (1) add
L1:     shr edx, 1
        jz  L2                  ; if (edx=0) exit loop
        shl ebx, 1
        jc  L3                  ; unsigned overflow
        loop L0
L2:     mov result, eax         ; write result
        call WriteDec
        call Crlf
        exit
L3:     mov edx, offset message ; write error message
        call WriteString
        call Crlf
        exit
main endp
end main
```



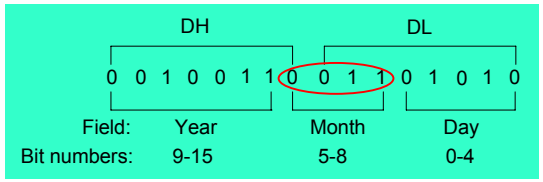
## Displaying Binary Bits

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

```
.data
value  DWORD 1234h
buffer BYTE 32 DUP(0),0
.code
   mov eax,value
   mov ecx,32
   mov esi,OFFSET buffer
L1:shl eax,1
   mov BYTE PTR [esi],'0'      ; default digit 0
   jnc L2
   mov BYTE PTR [esi],'1'      ; digit 1
L2:inc esi
   loop L1
```

## Isolating a Bit String

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




  - Isolate 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
```



## 7.4 Multiplication and Division Instructions

| Instructions | 동작 | 구분 |
|---|---|---|
| MUL src | edst ← dst * src | unsigned operation |
| DIV src | edst ← dst / src | |
| IMUL src | dst ← edst * src | signed operation |
| IDIV src | dst ← edst / src | |
| CBW | convert byte to word | convert (with sign extension) |
| CWD | convert word to dword | |
| CDQ | convert dword to qword | |

dst로 특정 레지스터를 사용하므로 instruction에 표기하지 않음
→ implied operand



## MUL and IMUL Instruction

- MUL src ; unsigned multiplication
- IMUL src ; signed multiplication

| Multiplicand | Multiplier(src) | Product |
|---|---|---|
| AL | r/m8 | AX |
| AX | r/m16 | DX:AX |
| EAX | r/m32 | EDX:EAX |

- MUL에서 CF는 다음의 경우에 1이됨
  - product가 너무 커서 원래의 multiplicand에 저장할 수 없을 때 즉, product의 upper half가 0이 아닐 때
- IMUL에서 OF와 CF는 다음의 경우에 1이됨
  - product의 upper half가 lower half의 sign extension이 아닌 경우



## MUL Examples

- val1 * val2 (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
```

## IMUL Instruction

- Multiply 48 * 4, using 8-bit operands

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

- Multiply 4,823,424 * (– 423):

```
mov eax,4823424
mov ebx,-423
imul ebx         ; EDX:EAX = FFFFFFFF86635D80h, OF=0
```



## DIV and IDIV Instruction

- DIV src ; unsigned division
- IDIV src ; signed division

| Dividend | Divisor (src) | Quotient | Remainder |
|---|---|---|---|
| AX | r/m8 | AL | AH |
| DX:AX | r/m16 | AX | DX |
| EDX:EAX | r/m32 | EAX | EDX |

- 나눗셈을 하기전에 dividend의 확장하여 upper half를 채워야 함
  - DIV : zero extension (upper half에 0을 저장)
  - IDIV : sign extension (upper half에 lower half의 부호를 저장)
    → CBW, CWD, CDQ Instruction 사용



## DIV Examples

- Divide 8003h by 100h, using 16-bit operands:

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

- Same division, using 32-bit operands:

```
mov eax,8003h         ; dividend, low
mov edx,0             ; clear dividend, high
mov ecx,100h          ; divisor
div ecx               ; EAX = 00000080h, DX = 3
```



## CBW, CWD, CDQ Instructions

- CBW ; AL:AH ← sign-extension(AL)
- CWD ; DX:AX ← sign-extension(AX)
- CDQ ; EDX:EAX ← sign-extension(EAX)

- Example:

```
mov ax, 0FF9Bh    ; (-101)
cwd               ; DX:AX = FFFF FF9Bh
```

## IDIV Examples

- 16-bit division of –48 by 5

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



## Divide Overflow

- 다음 프로그램의 수행 결과는?

```
mov dx,0087h
mov ax,6002h       ; dividend = 00876002h
mov bx,10h         ; divisor = 10h
div bx             ; quotient = 87600h
```

  - divide overflow : quotient를 AX에 저장할 수 없음

- 다음 프로그램의 수행 결과는?

```
mov  ax,0FDFFh     ; -513
cwd                ; dividend(DX:AX) = FFFF FDFF
mov  bx,100h       ; divisor(BX) = 0100h (256)
idiv bx
```

  - DX = FFFFh (-1),  AX = FFFEh (-2)



## Unsigned Arithmetic Expressions

- 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
   jmp  next
toobig:
   ; display error message and exit
next:
   ; continue
```



## Signed Arithmetic Expressions

- EAX = (-var1 * var2) + var3

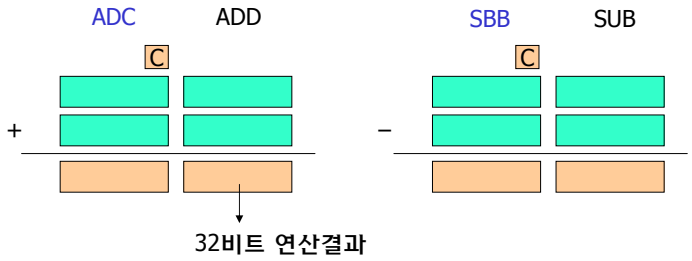
```
mov  eax,var1
neg  eax               ; EAX = -var1
imul var2              ; EAX = -var*var2
jo   TooBig            ; check for overflow
add  eax,var3          ; EAX = (-var1*var2)+var3
jo   TooBig            ; check for overflow
```

- var4 = (var1 * 5) / (var2 – 3)

```
mov  eax,var1          ; left side
mov  ebx,5
imul ebx               ; EDX:EAX = var1*5
mov  ebx,var2          ; right side
sub  ebx,3             ; EBX = var2-3
idiv ebx               ; EAX = (var1*5)/(var2-3)
mov  var4,eax
```

# 7.5 Extended Precision Arithmetic

- Extended Precision Arithmetic
  - 기계어는 현재 32비트 산술연산 까지만 제공함
  - 커다란 크기의 operand에 대한 산술연산은 32비트 연산을 반복하여 사용하여 수행가능
  - 아래 자리 연산의 CF 결과가 윗자리 연산에 사용됨
- Carry와 함께 덧셈, 뺄셈을 수행하는 Instruction 제공






# ADC and SBB Instruction

- ADC dst, src ; add with carry
  - 동작: dst ← dst + src + CF
- SBB dst, src ; subtract with carry(borrow)
  - 동작: dst ← dst – src – CF
- Example
  - Add two 32-bit integers (FFFFFFFFh + FFFFFFFFh), producing a 64-bit sum in EDX:EAX:

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



# Example

- EDX:EAX – 1
  - Starting value of EDX:EAX: 00000001 00000000h

```
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
```



# Extended Addition Example

```
.data
value1 QWORD 123456789abcdef0h
value2 QWORD 9abcdef012345678h
result DWORD 3 dup (?)
.code
   mov esi,OFFSET value1
   mov edi,OFFSET value2
   mov ebx,OFFSET result
   mov ecx,2
   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
   adc word ptr[ebx],0      ; add any leftover carry
```

## 7.6 ASCII and Packed Decimal Arithmetic

- Binary-Coded Decimal (BCD) – Packed Decimal
  - 10진수 1자리를 4-bit 2진수로 표현
- Unpacked decimal
  - 10진수 1자리를 8-bit 2진수로 표현
- ASCII Decimal
  - 10진수 1자리를 8-bit ASCII 코드로 표현
- Example
  - 10진수 5678의 표현

| | | | | |
|---|---|---|---|---|
| BCD | 0101 | 0110 | 0111 | 1000 |
| unpacked decimal | 00000101 | 00000110 | 00000111 | 00001000 |
| ASCII | 00110101 | 00110110 | 00110111 | 00111000 |



## ASCII Decimal Arithmetic

- Example: '8' + '2'

unpacked decimal

  - 00111000 + 00110010 = 01101010 →[Adjust]→ 00000001 00000000
    1 0

→[ASCII]→ 00110001 00110000
'1' '0'

- ASCII Adjust Instructions

| Instructions | 동작 |
|---|---|
| AAA | ASCII adjust after addition |
| AAS | ASCII adjust after subtraction |
| AAM | ASCII adjust after multiplication |
| AAD | ASCII adjust before division |

implicit operand: AX



## AAA Instructions

- AAA
  - 덧셈(ADD, ADC) 후에 AL > 9 이면 adjust, AH=AH+1
  - AL의 상위 4비트=0
- Add '8' and '2'

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

- Add '9' and '8'

```
mov ah,0
mov al,'8'        ; AX = 0038h
add al,'9'        ; AX = 0071h
aaa               ; AX = 0107h
or al,30h         ; AL = '17'
```



## AAS Example

- AAS
  - 뺄셈(SUB, SBB) 후에 AL < 00이면 adjust, AH = AH-1
  - AL의 상위 4비트=0
- Subtract '9' from '8'

```
mov ah,0
mov al,'8'        ; AX = 0038h
sub al,'9'        ; AX = 00FFh  (AL<0)
aas               ; AX = FF09h, CF=1
or al,30h         ; AL = '9'
```

28 – 9 = 19

## AAM Instruction

- AAM
  - unpacked decimal number에 대한 곱셈 결과를 adjustment
- Multiply 5 by 6

```
mov bl,'5'          ; first operand
mov al,'6'          ; second operand
and bl,0fh          ; BL=5 (unpacked decimal)
and al,0fh          ; AL=6 (unpacked decimal)
mul bl              ; AX = 001Eh
aam                 ; AX = 0300h
or  ax,3030h        ; AX = 3330h = '30'
```

1Eh = 30



## AAD Instruction

- AAD
  - 나눗셈하기 전에 AX에 저장된 unpacked decimal로 표현된 dividend를 adjustment (16비트 2진수로 변환)

```
.data
quotient  BYTE ?
remainder BYTE ?
.code
mov ax,3337h        ; dividend '37'
and ax,0f0fh        ; 0307h (unpacked decimal)
aad                 ; AX = 0025h
mov bl,'5'          ; divisor
and bl,0fh          ; 05h (unpacked decimal)
div bl              ; AX = 0207h
mov quotient,al     ; quotient=07h
mov remainder,ah    ; remainder=02h
```



## Packed Decimal Arithmetic

- Example: 18 + 22 = 40
  - 0001 1000 + 0010 0010 = 0011 1010 →[Adjust]→ 0100 0000
- Decimal Adjust Instruction

| Instructions | 동작 |
|---|---|
| DAA | Decimal adjust after addition |
| DAS | Decimal adjust after subtraction |

implicit operand: AL



## DAA Instruction

- DAA
  - 덧셈(ADD, ADC) 후에 packed BCD 형식이 되도록 adjust
  - CF: 상위4비트의 캐리, AF: 하위4비트의 캐리
- Examples

35 + 48

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

35 + 65

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

35 + 26

```
mov al,35h
add al,26h          ; AL = 5Bh,
daa                 ; AL = 61h, CF = 0, AF = 1
```

# DAS Instruction

- DAS
  - 뺄셈(SUB, SBB) 후에 packed BCD 형식이 되도록 adjust
  - CF: 상위4비트의 borrow, AF: 하위4비트의 borrow
- Examples

48 – 35

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

62 – 35

```
mov al,62h
sub al,35h          ; AL = 2Dh,
das                 ; AL = 27h, CF = 0, AF = 1
```

32 – 39

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