7장 정수 연산

7.1 이 장의 내용

- Shift와 Rotate 명령어
- Shift와 Rotate 응용
- 곱셈과 나눗셈 명령어
- 확장 덧셈과 뺄셈
- ASCII와 압축 10진 연산

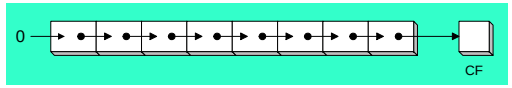
7.2 Shift와 Rotate 명령어

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	

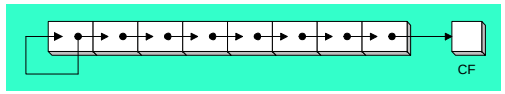
- dst는 r/m
- count는 imm8 또는 CL

논리 Shift와 산술 Shift

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



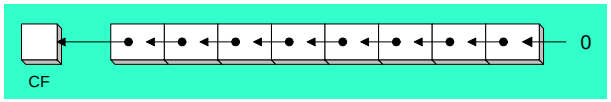
  - arithmetic shift = signed shift (부호bit를 보존함)



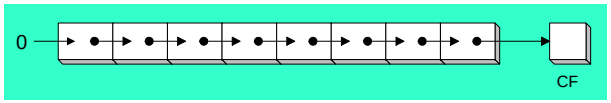
- 예 1-bit shift right
  - unsigned number: 11110000 (240) → **0**1111000 (120)
  - signed number: 11110000 (-16) → **1**1111000 (-8)
- Shift left와 Shift right
  - 1-bit shift left: 2를 곱한 것과 같음
  - 1-bit shift right: 2로 나눈 것과 같음

SHL와 SHR 명령어

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



- SHR dst, count




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

CF에는 마지막 shift 비트가 저장됨

예제

- Shift를 이용한 빠른 곱셈
  - n-bit shift left = multiply by 2^n
  - A * 5 = A * (2^2 + 1) = (A<< 2) + A

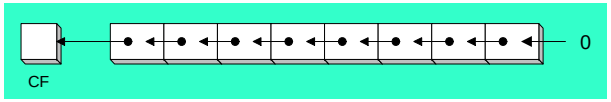
```
mov dl,A
mov al,dl                ; AL = A
shl dl,2                 ; DL = 4*A
add al,dl                ; AL = 5*A
```

- 나눗셈
  - n-bit shift right = divide by 2^n

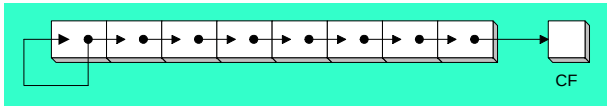
```
mov dl,80
shr dl,1                 ; DL = 40
shr dl,2                 ; DL = 10
```


SAL과 SAR 명령어

- Arithmetic shift 명령어: 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)
```

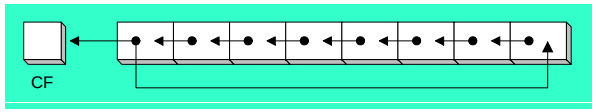

예제

- AX를 EAX로 부호 확장

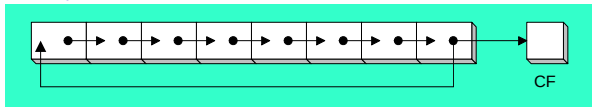
```
mov ax,-128              ; EAX = ????FF80h
shl eax, 16              ; EAX = FF800000h
sar eax, 16              ; EAX = FFFFFF80h
```

ROL과 ROR 명령어

- 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
```

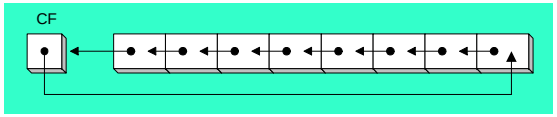

예제

- 비트 그룹의 교환
  - 바이트의 상위 4비트와 하위 4비트를 서로 교환

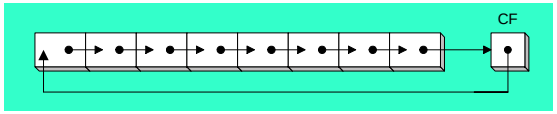
```
mov al, 26h
rol al, 4                   ; AL = 62h
```


RCL과 RCR 명령어

- 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
```


Shift/Rotate와 Overflow 플래그

- 1비트 Shift 또는 Rotate의 수행결과 부호비트가 반대로 되면 OF=1이 됨

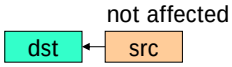
```
mov al, 127         ; AL = 01111111b
rol al, 1           ; AL = 11111110b, OF = 1

mov al, -128        ; AL = 10000000b
shr al, 1           ; AL = 01000000b, OF = 1
```

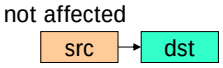
- shift count가 1보다 크면 OF는 정의되지 않음

SHLD와 SHRD 명령어

- SHLD dst, src, count
  - **dst** 를 **count** 횟수만큼 shift left
  - **src**의 상위 count 비트를 dst의 하위 count 비트에 복사
  - **src** 는 영향받지 않음



- SHRD dst, src, count
  - **dst** 를 **count** 횟수만큼 shift right
  - **src**의 하위 count 비트를 dst의 상위 count 비트에 복사
  - **src**는 영향받지 않음



- 명령어 형식

dst src count

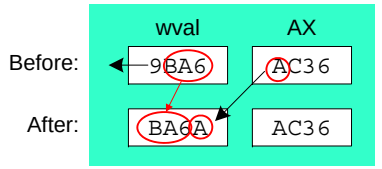
```
SHLD reg16/32, reg16/32, imm8/CL
SHLD mem16/32, reg16/32, imm8/CL
```

SHRD도 같은 형식

예제

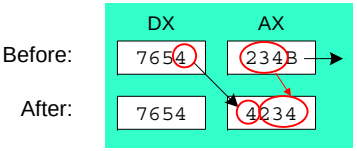
- 변수 wval을 4비트 왼쪽으로 이동하고
  변수 wval의 하위 4비트는 AX의 상위4비트로 채움

```
.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 Shift와 Rotate의 응용

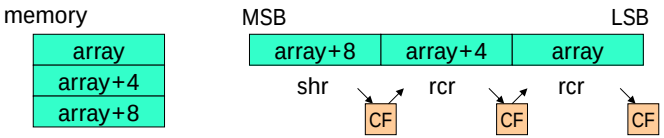
- 여러 개의 더블워드 shift

- 이진 곱셈
- 이진수 비트 표시
  (ex) AX = 3Ah → string "00111010"으로 변환
- 비트필드 분리
  (ex) 01001101 → 특정필드값을 추출 (001)

여러 개의 더블워드 Shift

- 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 ???

이진 곱셈

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

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

- shift를 사용한 32비트 unsigned integer 곱셈
  - 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
```

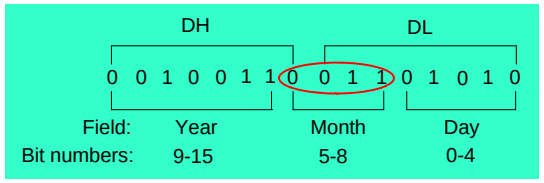

이진수 비트 표시

- Algorithm: 32비트 정수를 ASCII 2진 문자열로 변환
  - 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
```


비트 필드 분리

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



  - month field 분리

```
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 곱셈과 나눗셈 명령어

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

MUL과 IMUL 명령어

- 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 예제

- 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 예제

- 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
```

IMUL 명령어 - 2, 3 피연산자 형식

- IMUL (signed multiply)는 2-피연산자, 3-피연산자 형식도 지원
- 2-피연산자 IMUL

  IMUL reg, reg/mem

  IMUL reg, imm ; imm은 8비트 또는 reg와 같은 크기
- 3-피연산자 IMUL

  IMUL reg, reg/mem, imm ; imm은 8비트 또는 reg와 같은 크기

DIV와 IDIV 명령어

- 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, IDIV는 산술 상태 플래그가 정의되지 않음

DIV 예제

- 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 명령어

- 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
```

- 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 예제

- 16-bit division of –48 by 5

```
mov  ax,-48
cwd               ; extend AH 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

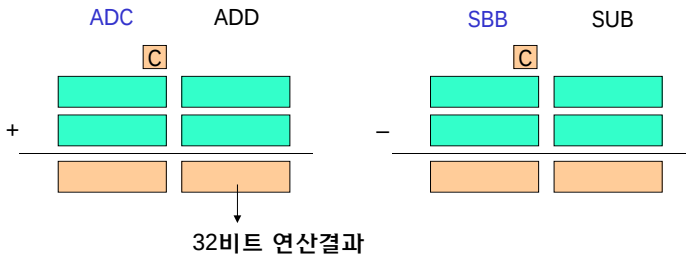
```
mov  eax,var1
neg  eax            ; EAX = -var1
imul var2           ; EAX = -var*var2
jo   TooBig         ; check for overflow
add  eax,var3       ; EAX = (-var1*var2)+var3
jo   TooBig         ; check for overflow
```

- 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 확장 정밀도 덧셈, 뺄셈

- 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



ADC와 SBB 명령어

- 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
```


예제

- 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
CF loop L1                  ; repeat the loop
   adc word ptr[ebx],0      ; add any leftover carry
```

7.6 ASCII와 압축 10진 산술연산

- 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 명령어

- 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 ax,3030h       ; AL = '17'
```


AAS 명령어

- AAS
  - 뺄셈(SUB, SBB) 후에 AL < 0이면 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 명령어

- 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 명령어

- 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 명령어

- 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 명령어

- 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
```