6장 조건부 처리

이 장의 내용

- 부울과 비교 명령어
- 조건부 점프
- 조건부 루프 명령어
- 조건부 구조

6.2 부울과 비교 명령어

부울 명령어

Instructions	동작
AND dst, src	dst ← dst AND src
OR dst, src	dst ← dst OR src
XOR dst, src	dst ← dst XOR src
NOT dst	dst ← NOT dst

src: source operand
dst: destination operand

- 비트단위(bitwise) 부울 연산
- operand rule: MOV, 산술연산과 같음

Status Flags - 복습

- CPU Status Flags

Flags	동작
ZF (zero)	결과가 0이면 set
SF (sign)	음수이면 set (MSB와 같음)
CF (carry)	unsigned 결과가 표현범위 벗어나면 set
OF (overflow)	signed 결과가 표현범위 벗어나면 set
PF (parity)	결과의 1값을 갖는 bit수가 짝수이면 set
AF (auxiliary)	하위4비트에 대한 carry발생하면 set

AND 명령어

- AND
  - selective clear (mask operation)

```
        0 0 1 1 1 0 1 1
AND     0 0 0 0 1 1 1 1
        -----------------
        0 0 0 0 | 1 0 1 1
cleared           unchanged
```

AND

x	y	x ∧ y
0	0	0
0	1	0
1	0	0
1	1	1

- (예) 소문자를 대문자로 변환
  - ASCII code:

    01000001 ~ 01011010 A ~ Z

    01100001 ~ 01111010 a ~ z
  - 소문자 → 대문자 변환: bit 5를 0으로 바꿈

```
mov al,'a'                  ; AL = 01100001b
and al,11011111b            ; AL = 01000001b
```

OR 명령어

- OR
  - selective set

```
          0 0 1 1 1 0 1 1
OR        0 0 0 0 1 1 1 1
          -----------------
          0 0 1 1 1 1 1 1
unchanged ---------------- set
```

OR

x	y	x ∨ y
0	0	0
0	1	1
1	0	1
1	1	1

- (예) 대문자를 소문자로 변환
  - 대문자 → 소문자 변환 : bit 5를 1로 바꿈

```
mov al,'A'                  ; AL = 01000001b
or  al,00100000b            ; AL = 01100001b
```

XOR 명령어

- XOR
  - selective complement (invert)

```
          0 0 1 1 1 0 1 1
XOR       0 0 0 0 1 1 1 1
          ---------------
          0 0 1 1 | 0 1 0 0
unchanged ------------------ inverted
```

XOR

x	y	x ⊕ y
0	0	0
0	1	1
1	0	1
1	1	0

- (예) Parity Flag 검사 (even parity일때 PF=1)
  - 어떤 값을 0과 XOR 연산 수행시 결과는 변하지 않으며 flag만 설정됨

```
mov al, 10110101b                ; 5 1's
xor al, 0                        ; PF=0 (odd parity)
mov al, 11001100b                ; 4 1's
xor al, 0                        ; PF=1 (even parity)
```

  - 16-bit parity - 두 바이트 간에 XOR 연산 수행

```
mov ax, 64C1h                    ;0110 1000 1100 0001
xor ah, al                       ;PF=1 (even parity)
```

NOT 명령어

- NOT
  - 1's complement

NOT 0 0 1 1 1 0 1 1

1 1 0 0 0 1 0 0 ——— inverted

NOT

X	¬X
F	T
T	F

예제

- 비트맵 집합
  - 비트값이 1이면 집합 원소를 표시 (32비트: 최대 32개의 원소 표시)
- 여집합 - NOT 명령어 사용
- 교집합 - AND 명령어 사용
- 합집합 - OR 명령어 사용
- (예)

```
SetX = 80000007h
SetY = 81500763h
```

```
mov eax, SetX          ; 10000000 00000000 00000000 00000111
not eax                ; eax=X의 여집합
```

```
mov eax, SetX
and eax, SetY          ; eax=X와 Y의 교집합
```

```
mov eax, SetX
or eax, SetY           ; eax=X와 Y의 합집합
```

TEST와 CMP 명령어

- TEST dst, src
  - dst ∧ src 연산 (AND연산) 수행 (결과를 저장하지 않음)
  - 비트 검사에 사용됨
- CMP dst, src
  - dst – src 연산 수행 (결과를 저장하지 않음)
  - 크기 비교에 사용됨
- TEST와 CMP instruction의 공통점
  - 연산 결과를 저장하지 않고 flag에만 영향을 줌 – dst는 변하지 않음
  - 주로 conditional jump와 함께 사용

CPU 개별 플래그 설정

- ZF 설정

```
test al, 0      ; ZF = 1
and al, 0       ; ZF = 1 (AL 수정)
or al, 1        ; ZF = 0 (AL 수정)
```

- SF 설정

```
or al, 80h      ; SF = 1
and al, 7Fh     ; SF = 0
```

- CF 설정

```
stc             ; CF = 1
clc             ; CF = 0
```

- OF 설정

```
mov al, 7F
inc al          ; OF = 1
or eax, 0       ; OF = 0 (논리연산)
```

6.3 조건부 점프

- *J**cond*** 명령어
  - 형식: *J**cond*** label
  - 이전의 연산 결과가 주어진 조건을 만족하면 label 위치의 instruction으로 jump (이전의 연산결과는 주로 flag상태를 참조함)
- Condition 종류와 조건부 점프

Condition 종류	Instructions
특정 Flag상태	JZ, JNZ, JC, JNC, JO, JNO, JS, JNS, JP, JNP
equality	JE, JNE, JCXZ, JECXZ
signed 비교	JG, JGE, JL, JLE (Greater, Less) JNLE, JNL, JNGE, JNG
unsigned 비교	JA, JAE, JB, JBE (Above, Below) JNBE, JNB, JNAE, JNA

  - JZ와 JE, JNZ와 JNE는 같은 명령어

특정 플래그 값에 의한 점프

Mnemonic	Description	Flags
JZ	Jump if zero	ZF = 1
JNZ	Jump if not zero	ZF = 0
JC	Jump if carry	CF = 1
JNC	Jump if not carry	CF = 0
JO	Jump if overflow	OF = 1
JNO	Jump if not overflow	OF = 0
JS	Jump if signed	SF = 1
JNS	Jump if not signed	SF = 0
JP	Jump if parity (even)	PF = 1
JNP	Jump if not parity (odd)	PF = 0

동등 비교 점프

Mnemonic	Description
JE	Jump if equal (*leftOp* = *rightOp*)
JNE	Jump if not equal (*leftOp* ≠ *rightOp*)
JCXZ	Jump if CX = 0
JECXZ	Jump if ECX = 0

부호없는 비교 점프

Mnemonic	Description
JA	Jump if above (if *leftOp* > *rightOp*)
JNBE	Jump if not below or equal (same as JA)
JAE	Jump if above or equal (if *leftOp* >= *rightOp*)
JNB	Jump if not below (same as JAE)
JB	Jump if below (if *leftOp* < *rightOp*)
JNAE	Jump if not above or equal (same as JB)
JBE	Jump if below or equal (if *leftOp* <= *rightOp*)
JNA	Jump if not above (same as JBE)

부호있는 비교 점프

Mnemonic	Description
JG	Jump if greater (if *leftOp* > *rightOp*)
JNLE	Jump if not less than or equal (same as JG)
JGE	Jump if greater than or equal (if *leftOp* >= *rightOp*)
JNL	Jump if not less (same as JGE)
JL	Jump if less (if *leftOp* < *rightOp*)
JNGE	Jump if not greater than or equal (same as JL)
JLE	Jump if less than or equal (if *leftOp* <= *rightOp*)
JNG	Jump if not greater (same as JLE)

예제 – 크기 비교

- 동등 비교

```
cmp eax,ebx      ; eax - ebx
je  L1
```

if (eax==ebx)
  goto L1

- 크기 비교
  - unsigned 비교

```
cmp eax,ebx
ja  Larger
```

if (eax>ebx)
  goto Larger

  - signed 비교

```
cmp eax,ebx
jg  Larger
```

예제 – 비트 검사

비트검사

- AL의 bit 0 또는 bit 1이 1이면 jump

```
test al,00000011b
jnz L1
```

- AL의 bit 0과 bit 1이 모두 0이면 jump

```
test al,00000011b
jz L1
```

- AL의 bit 0과 bit 1이 모두 1이면 jump (AL 내용 변경)

```
and al,00000011b        ; clear unwanted bits
cmp al,00000011b        ; check remaining bits
je  L1                  ; all set? jump to L1
```

예제 – 짝수, Zero 검사

- 짝수 검사
  - if (wordVal is even) goto L1

```
mov ax,wordVal
test ax,1           ; bit 0 set?
jz  L1              ; jump if ZF set
```

- Zero/Nonzero 검사
  - if (AL ≠ 0) goto L1

```
or  al,al       ; AL unchanged, Flag 영향
jnz L1          ; jump if not zero
```

  - if (AL = 0) goto L1

```
or  al,al       ; AL unchanged
jz L1           ; jump if zero
```

예제 – 최대값 찾기

- v1, v2, v3에 있는 unsigned 값의 최대값을 max에 저장

```
.data
v1   word 10
v2   word 50
v3   word 30
max  word ?
.code
     mov ax,v1          ; ax=v1 (assume v1 is larger)
     cmp ax,v2
     jae L1
     mov ax,v2          ; if (ax<v2)ax = v2
L1:  cmp ax,v3
     jae L2
     mov ax,v3          ; if (ax<v3)ax = v3
L2:  mov max,ax         ; max = ax
```

배열 검색 – 조건을 만족하는 원소 찾기

- 0이 아닌 원소를 찾으면 배열 검색 종료

```
.data
array SWORD 0,0,0,1,20,0,-1
.code
    mov ebx, OFFSET array          ; array address
    mov ecx, LENGTHOF array        ; loop counter
L1:cmp WORD PTR [ebx], 0
    jnz found
    add ebx, 2
    loop L1
    jmp notfound
found: ...
    jmp quit
notfound: ...
quit:
```

문자열 암호화

- XOR의 성질: $(X \oplus Y) \oplus Y = X$
- 암호화 및 해독
  - 암호화: $S = X \oplus K$
  - 암호해독: $S \oplus K = (X \oplus K) \oplus K = X$

```
KEY = 239                       ; can be any byte value
BUFMAX = 128
.data
buffer  BYTE BUFMAX+1 DUP(0)
bufSize DWORD BUFMAX

.code
        mov ecx,bufSize         ; loop counter
        mov esi,0               ; index 0 in buffer
L1:
        xor buffer[esi],KEY ; translate a byte
        inc esi                 ; point to next byte
        loop L1
```

6.4 조건부 루프 명령어

LOOPZ (LOOPE) 명령어

- 형식: LOOPZ *dest*
- 동작: ECX ← ECX – 1
  if ECX > 0 and ZF=1, jump to destination
- 용도: 주어진 값과 일치하지 않은 첫 배열 원소를 찾음
  (ECX가 0이거나 비교결과가 같지 않을 때 루프 종료)

LOOPNZ (LOOPNE) 명령어

- 형식: LOOPNZ *dest*
- 동작: ECX ← ECX – 1
  if ECX > 0 and ZF=0, jump to destination
- 용도: 주어진 값과 일치하는 첫 배열 원소를 찾음
  (ECX가 0이거나 비교결과가 같을 때 루프 종료)

예제

- 양수를 찾을 때까지 배열 검색

```
.data
array SWORD -3,-6,-1,-10,10,30,40,4
sentinel SWORD 0
.code
    mov esi,OFFSET array
    mov ecx,LENGTHOF array
next:
    test WORD PTR [esi],8000h  ; test sign bit
    pushfd                     ; push flags on stack
    add esi,2
    popfd                      ; pop flags from stack
    loopnz next                ; continue loop
    jnz quit                   ; none found
    sub esi,2                  ; ESI points to value
quit:
```

flag

6.5 조건부 구조

- 블록구조 IF 문
- AND 복합 수식
- OR 복합 수식
- WHILE 루프

블록구조 IF 문

- C언어

```
if( op1 == op2 ){
  X = 1;
  Y = 2;
} else {
  X = 2;
  Y = 1;
}
```

반대

어셈블리언어

```
    mov eax,op1
    cmp eax,op2
    jne L1
    mov X,1          } then block
    mov Y,2
    jmp L2
L1: mov X,2          } else block
    mov Y,1
L2:
```

블록구조 IF 문 - 다른방법

- C언어

```
if( op1 == op2 ){
  X = 1;
  Y = 2;
} else {
  X = 2;
  Y = 1;
}
```

어셈블리언어

```
    mov eax,op1
    cmp eax,op2
    je L1
    mov X,2          } else block
    mov Y,1
    jmp L2
L1: mov X,1          } then block
    mov Y,2
L2:
```

AND 복합 수식

- AND의 Short-circuit 평가
  - if (비교식1 && 비교식2 && 비교식3) ...
    앞의 비교식이 거짓이면 뒤의 비교식을 실행하지 않음

```
if (AL>BL && BL>CL)
  X = 1;
```

```
    cmp al,bl    ; 비교식1
    ja  L1
    jmp next
L1:
    cmp bl,cl    ; 비교식2
    ja  L2
    jmp next
L2:

    mov X,1     ; then block
next:
```

AND 복합 수식 - 다른방법

- C언어

```
if (AL>BL && BL>CL)
  X = 1;
```

반대

어셈블리언어

```
    cmp al,bl    ; 비교식1
    jbe next
L1:
    cmp bl,cl    ; 비교식2
    jbe next
L2:

    mov X,1      ; then block
next:
```

OR 복합 수식

- OR의 short-circuit 평가
  - if (비교식1 || 비교식2 || 비교식3) ...
    앞의 비교식이 참이면 뒤의 비교식을 실행하지 않음

```
if (AL>BL || BL>CL)
  X = 1;
```

```
    cmp al,bl    ; 비교식1
    ja  L1
    cmp bl,cl    ; 비교식2
    jbe next
L1:
    mov X,1
next:
```

WHILE 루프

- C언어

```
while( a < b) {
    a++;
    b--;
}
```

반대

어셈블리언어

```
    mov eax,a
while:
    cmp eax,b
    jnl endwhile
    inc eax
    dec b
    jmp while
endwhile:
    mov a,eax
```