9장. 스트링과 배열

9.1 이 장의 내용

- 스트링 프리미티브 명령어
- 2차원 배열
- 정수배열 검색 및 정렬

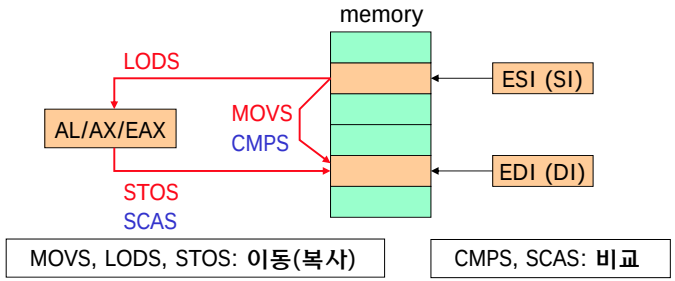
9.2 스트링 프리미티브 명령어

- String Primitive Instructions

Instructions	설명	동작
MOVS(B,W,D)	Move string data	M[EDI] ← M[ESI]
CMPS(B,W,D)	Compare strings	M[EDI]와 M[ESI]를 비교
SCAS(B,W,D)	Scan string	M[EDI]와 AL/AX/EAX와 비교
STOS(B,W,D)	Store string	M[EDI] ← AL/AX/EAX
LODS(B,W,D)	Load string	AL/AX/EAX ← M[ESI]

  - MOVS**B**, MOVS**W**, MOVS**D** 와 같이 명령어의 끝에 오퍼런드의 크기를 나타내는 B(byte), W(word), D(double word)를 사용함
  - Operand를 명시하지 않음 – Implicit Operand

String Primitive Instructions의 동작



- **ESI**와 **EDI**는 Direction Flag(DF)의 값에 따라서 자동적으로 증가 또는 감소됨 → String(배열) 처리에 유용
  - DF=0: **증가** (1,2,4)
  - DF=1: **감소** (1,2,4)
- 16-bit 모드에서는 ESI와 EDI 대신에 SI와 DI가 사용됨

MOVSB, MOVSW, MOVSD 명령어

- MOVSB, MOVSW, MOVSD
  - 동작: M[ES:EDI] ← M[DS:ESI]
    EDI와 ESI 증가 또는 감소 (B: 1, W: 2, D: 4)

```
.data
source DWORD 0FFFFFFFFh
target DWORD ?
.code
  mov esi,OFFSET source
  mov edi,OFFSET target
  movsd                    ; [edi] ← [esi]
                           ; target ← source
```


Direction Flag와 CLD, STD 명령어

- Direction Flags (DF) – ESI와 EDI의 증가, 감소 방향을 제어
  - DF = 0: 증가
  - DF = 1: 감소
- CLD와 STD Instruction
  - CLD ; clear DF (0)
  - STD ; set DF (1)
  - DF를 명시적으로 지정하여 증가/감소 방향을 정할 때에 사용

REP prefix

- REP (repeat prefix)
  - MOVSB, MOVSW, MOVSD 명령어 앞에 사용하여 명령어의 반복 수행을 지시함
  - ECX에 반복 횟수 지정

rep movsb
=
L1: movsb
loop L1

- Example: 20 doubleword를 source 배열에서 target 배열로 복사

```
.data
source DWORD 20 DUP(?)
target DWORD 20 DUP(?)
.code
   cld                     ; DF=0 (forward)
  mov esi,OFFSET source
  mov edi,OFFSET target
  mov ecx,LENGTHOF source  ; set REP counter(20)
   rep movsd               ; 20번 반복 수행
```


CMPSB, CMPSW, CMPSD 명령어

- CMPSB, CMPSW, CMPSD
  - 동작: M[ES:ESI] – M[DS:EDI] (compare)
    EDI와 ESI 증가 또는 감소
- Example: if (source > target) goto L1
  else goto L2

```
.data
source DWORD 1234h
target DWORD 5678h
.code
mov esi,OFFSET source
mov edi,OFFSET target
cmpsd              ; compare [esi] with [edi]
ja L1              ; if source > target, jump L1
jmp L2             ; else jump L2
```

REPE, REPNE prefix

- conditional REP prefix
  - REPE (REPZ) ; ECX>0, ZF=1인 동안 string instruction 반복 수행
  - REPNE (REPNZ) ; ECX>0, ZF=0인 동안 string instruction 반복 수행
  - 반복할 때마다 ECX가 1씩 감소
  - CMPS와 SCAS와 같은 string 비교 명령어와 함께 사용

```
repe cmpsb
  ||
L1: cmpsb
    loope L1
```

```
repne scasb
  ||
L1: scasb
    loopne L1
```


Example: 여러 개의 더블워드 비교

- 예: 더블워드 배열 비교

```
.data
source DWORD 1234h, 2345h, 4567h
target DWORD 5678h, 789ah, 9abch
.code
   cld
   mov esi,OFFSET source
   mov edi,OFFSET target
   mov ecx, LENGTHOF source
   repe cmpsd    ; repeat while equal
   ; Now, if equal, then ZF=1, else ZF=0
   je Equal
   jmp NotEqual
```


SCASB, SCASW, SCASD 명령어

- SCASB, SCASW, SCASD
  - 동작: AL/AX/EAX – M[DS:EDI] (compare)
    EDI 증가 또는 감소
- 용도
  - 배열에서 특정한 값을 검색
  - 배열에서 특정한 값이 아닌 원소를 검색

Example: 문자열에서 특정 문자 검색

```
.data
alpha BYTE "ABCDEFGH",0
.code
mov al,'F'                  ; search for 'F'
mov edi,OFFSET alpha
mov ecx,LENGTHOF alpha
cld
repne scasb                 ; repeat while not equal
jnz quit
dec edi                     ; EDI points to 'F'
```

- repeat가 종료되는 경우
  - ZF=1 : 검색 성공(EDI를 감소시켜서 검색문자 위치 저장)
  - ZF=0, ECX=0 : 검색 실패

STOSB, STOSW, STOSD 명령어

- STOSB, STOSW, STOSD
  - 동작: M[DS:EDI] ← AL/AX/EAX (store)
    EDI 증가 또는 감소
- Example: 배열을 0FFh 로 채움

```
.data
str BYTE 100 DUP(?)
.code
   mov al,0FFh             ; value to be stored
   mov edi,OFFSET str      ; ES:DI points to target
   mov ecx,LENGTHOF str    ; character count
   cld                     ; direction = forward
   rep stosb               ; fill with contents of AL
```


LODSB, LODSW, LODSD 명령어

- LODSB, LODSW, LODSD
  - 동작: AL/AX/EAX ← M[DS:EDI] (load)
    ESI 증가 또는 감소
- Example: 숫자를 ASCII 코드로 변환하여 출력

```
.data
array BYTE 1,2,3,4,5,6,7,8,9
.code
     mov esi,OFFSET array
     mov ecx,LENGTHOF array
     cld
L1:  lodsb                   ; load byte into AL
     or al,30h               ; convert to ASCII
     call WriteChar          ; display it
     loop L1
```

  - LODS를 REP와 같이 사용하는 것은 의미가 없음

Example: 배열 곱셈

- doubleword 배열의 각 원소에 특정한 값을 곱하여 저장

```
.data
array DWORD 1,2,3,4,5,6,7,8,9,10
multiplier DWORD 10
.code
   cld                     ; direction = up
   mov esi,OFFSET array    ; source index
   mov edi,esi             ; destination index
   mov ecx,LENGTHOF array  ; loop counter
L1: lodsd                  ; copy [ESI] into EAX
   mul multiplier          ; multiply by a value
   stosd                   ; store EAX at [EDI]
   loop L1
```


Exercise

- unpacked BCD가 저장된 byte 배열 원소를 ASCII로 변환하여 새로운 byte 배열에 저장하는 프로그램 작성

```
.data
array BYTE 1,2,3,4,5,6,7,8,9
dest  BYTE (LENGTHOF array) DUP(?)
```

```
   mov esi,OFFSET array
   mov edi,OFFSET dest
   mov ecx,LENGTHOF array
   cld
L1: lodsb                  ; load into AL
   or al,30h               ; convert to ASCII
   stosb                   ; store into memory
   loop L1
```

9.4 2차원 배열

- Base-Index Addressing
  - operand 주소 = reg + reg
  - 형식: [reg + reg]

  (예) `mov al, [ebx+esi]` ; al ← M[ds:ebx+esi]

- Base-Index Displacement Addressing
  - operand 주소 = reg + reg + constant
  - 형식: [reg + reg + constant] 또는 constant[reg + reg]

  (예) `mov al, table[ebx+esi]` ; al ← M[ds:ebx+esi+table]

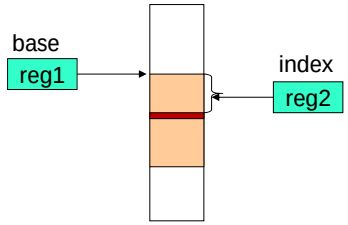
  - constant를 displacement라고 부름

Base-Index Operand

- Base-Index Operand:

  형식: [reg + reg] operand주소 = reg + reg

  - protected mode에서는 general purpose register 사용 가능
  - real mode에서는 bx, bp (베이스 레지스터)와 si, di (인덱스 레지스터)의 결합으로 사용함
    - bp가 포함되면 stack segment를 사용



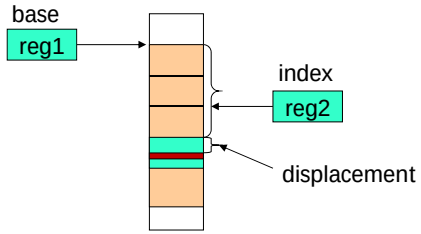
Base-Index-Displacement Operand

- Base-Index-Displacement Operand

  형식: [reg + reg + disp]
  disp[reg + reg]

  operand주소 = reg + reg + disp

  - 사용가능한 register는 base-index operand와 같음



Scaling Factor 사용

- scaling factor를 사용한 base-index operand

  형식: [reg + reg*scale]

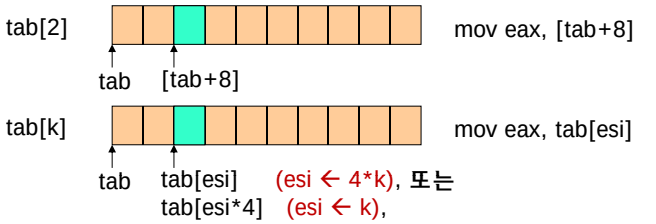
- scaling factor를 사용한 base-index-displacement operand

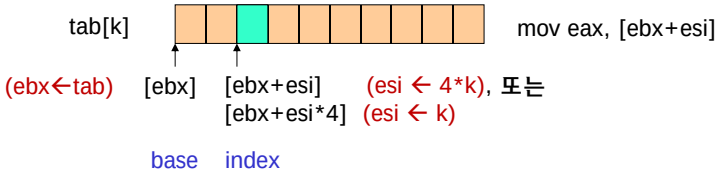
  형식: [reg + reg*scale + disp]
  disp[reg + reg*scale]

- word, double word 배열에 대한 코드 작성에 주로 사용

Array – 1-dimension

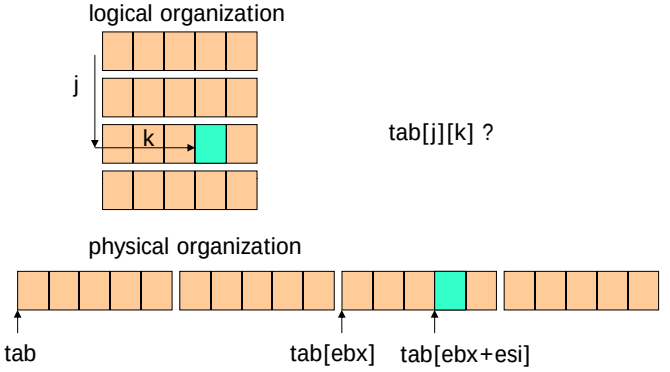
- 1-dimensional array: int tab[10];





Array – 2-dimension

- 2-dimensional array: int tab[4][5];

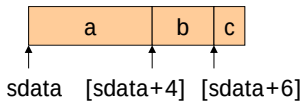


단, ebx ← 4*j*5 (행 시작 offset)
esi ← 4*k (열의 위치)

Structure

- structure

```
struct misc {
    int a;
    short b;
    char c;
} sdata;
```



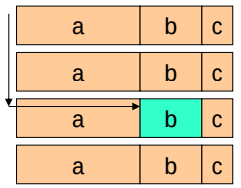
sdata.b = ax ⟶ mov [sdata+4], ax

p=&sdata;
p->b = ax ⟶

```
mov ebx, offset sdata
mov [ebx+4], ax
   또는
mov ebx, offset sdata
mov esi, 4
mov [ebx+esi], ax
```


Array of Structure

- array of structure: struct misc sarray[4];



sarray[k].b = ax ⟶

```
mov [esi+sarray+4], ax
mov sarray[esi+4], ax

mov [ebx + esi + 4], ax
```

단, ebx ← sarray 시작주소
esi ← k*구조체크기 (행 시작 offset)

Example: 2-Dimensional Table

- Table with 3 x 5 array data

```
table BYTE  10h, 20h, 30h, 40h, 50h
      BYTE  60h, 70h, 80h, 90h, 0A0h
      BYTE 0B0h, 0C0h, 0D0h, 0E0h, 0F0h
NumCols = 5
```

```
Row = 1
Col = 2
   mov ebx,NumCols * Row
   mov esi,Col
   mov al,table[ebx + esi]
```


Example: 2차원 배열의 한 행의 합 계산

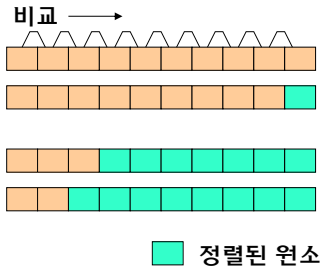
```
   mov eax, 2       ; row index
   mov ecx, 5       ; row size
   mul ecx          ; row index * row size
   add ebx,eax      ; ebx = row offset
   mov eax,0        ; sum = 0
   mov esi,0        ; column index
L1: movzx edx,BYTE PTR[ebx + esi] ; get a byte
   add eax,edx      ; add to sum
   inc esi          ; next byte in row
   loop L1
```

```
L1: movzx edx,WORD PTR[ebx + esi*2] ; get a word
   add eax, edx
```

```
L1: add eax,[ebx + esi*4]
```


9.5 정수 배열의 검색과 정렬

- Bubble Sort
  - 인접원소와 비교후 정렬 순서가 아니면 서로 교환하면서 정렬함
  - 총 n-1 번 반복
    - pass 1: n-1번 비교
    - pass 2: n-2번 비교
    - . . .
    - pass n-2: 2번 비교
    - pass n-1: 1번 비교



Bubble Sort Pseudocode

- N: 배열원소수

```
CX1 = N - 1
while (CX1 > 0) {
  esi = array의 시작주소
  CX2 = CX1
  while (CX2 > 0) {
  {
     if( [esi+4] < [esi] )  // 인접원소 비교
        exchange [esi] with [esi+4]
     add esi,4
     dec CX2
  }
  dec CX1
}
```

Bubble Sort Implementation

```
title Bubble Sort
INCLUDE irvine32.inc

.data
array   sdword  10, 30, 20, 80, 70, 90, 25, 80

.code
main proc
        mov  ecx, lengthof array        ; array size
        dec  ecx                        ; decrement count by 1
L1:     push ecx                        ; save outer loop count
        mov  esi, offset array          ; point to first value
L2:     mov  eax, [esi]                 ; get array value
        cmp  [esi+4],eax                ; compare a pair of values
        jge  L3                         ; if [esi+4] <= [esi], skip
        xchg eax, [esi+4]               ; else exchange the pair
        mov  [esi], eax
L3:     add  esi,4                      ; increment pointer
        loop L2                         ; inner loop
        pop  ecx                        ; retrieve outer loop count
        loop L1                         ; else repeat outer loop
```


Bubble Sort Implementation (계속)

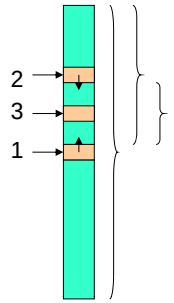
```
        mov ecx, lengthof array
        mov esi, offset array
L4:     mov eax, [esi]
        call WriteDec
        mov al, ' '
        call WriteChar
        add esi, 4
        loop L4
        exit
main endp
end main
```

; sort 결과 출력

Binary Search

- Binary Search
  - 순서대로 정렬된 배열 원소들을 검색하는 알고리즘
  - Divide and conquer 방법 사용
  - 검색 범위를 반으로 줄이면서 검색 수행 → O(log n) 알고리즘
- 최대 비교 횟수

Array Size (n)	Maximum Number of Comparisons: $(\log_2 n) + 1$
64	7
1,024	11
65,536 (2^{16})	17
1,048,576 (2^{20})	21
4,294,967,296 (2^{32})	33



Binary Search Pseudocode

- values: 배열, N: 배열원소 수, sVal: 검색값

```
first = 0;                      검색범위 시작
last = count - 1;               검색범위 끝
while( first <= last ){         시작>끝 이면 종료
    mid = (last + first) / 2;
    if(values[mid] < sVal)
        first = mid + 1;        중간값<sVal이면 검색범위←위쪽
    else if(values[mid] > sVal)
        last = mid - 1;         중간값<sVal이면 검색범위←아래쪽
    else
        "success"               검색성공
}
"fail"
```

Binary Search Implementation

```
title Binary Search
include irvine32.inc

.data
values  dword 10, 25, 33, 42, 55, 60, 72, 88, 90, 111,
              125, 133, 145, 167, 200, 222, 236
sval    dword ?
first   dword ?
last    dword ?
mid     dword ?
prompt  byte 'Input a number: ',0
success byte 'Search Success, Index = ',0
fail    byte 'Search Failure ',0

.code
main proc
        mov  edx, offset prompt
        call WriteString
        call ReadInt
        mov  sval, eax
```

검색할 숫자 입력

Binary Search Implementation (계속)

```
        mov  first,0                    ; first = 0
        mov  eax, lengthof values
        dec  eax
        mov  last,eax                   ; last = (N - 1)
        mov  edi,sval                   ; EDI = sval
        mov  ebx,offset values          ; EBX = array pointer
L1:
        mov  eax,first
        cmp  eax,last
        jg   L5                         ;  if (first > last)  exit search

        mov  eax,first
        add  eax,last
        shr  eax,1
        mov  mid,eax                    ; mid = (first + last) / 2

        mov  esi,mid
        shl  esi,2                      ; ESI = 4*mid
        mov  edx,[ebx+esi]              ; EDX = values[mid]
```


Binary Search Implementation (계속)

```
        cmp  edx,edi                    ; now EDX = values[mid], EDI = sVal
        jge  L2                         ; if ( values[mid] < sVal)
        mov  eax,mid                    ;       first = mid + 1
        inc  eax
        mov  first,eax
        jmp  L1                         ; continue the loop

L2:     jle  L3                         ; if ( values[mid] > sVal)
        mov  eax,mid                    ;       last = mid - 1
        dec  eax
        mov  last,eax
        jmp  L1                         ; continue the loop

L3:     mov  edx, offset success        ; search success
        call WriteString
        mov  eax, mid
        call WriteDec
        call Crlf
```


Binary Search Implementation (계속)

```
L5:     mov  edx, offset fail           ; search failure
        call WriteString
        call Crlf

L9:     exit

main endp
end main
```