3장 어셈블리언어의 기초

이 장의 내용

- 어셈블리 언어의 기본 구성 요소
- 간단한 어셈블리 언어 프로그램 예
- 프로그램 어셈블리, 링크, 실행
- 데이터 정의
- 기호 상수
- 실제주소모드 프로그래밍

3.1 어셈블리언어 기본 구성요소

- 정수 상수
- 정수 수식
- 실수 상수
- 문자, 문자열 상수
- 예약어, 식별자
- 디렉티브(Directives)와 명령어
- 레이블(Labels)
- 니모닉(Mnemonics)과 피연산자(Operands)
- 주석(Comments)

정수 상수

- 부호(+ 또는 –)는 선택사항
  - 5 +5 –5
- 진수(radix): binary, decimal, hexadecimal, or octal
  - d – decimal (ex) 26, 26d
  - h – hexadecimal (ex) 2Bh, 55H, **0**A3h
  - b, y – binary (ex) 10110001b
  - o, q – octal (ex) 42q, 42o

문자(A-F)로 시작하는 16진수 앞에 0을 넣음

  - r – encoded real (실수를 2진수형태로 표기, IEEE754 표준형식)

정수 수식(Expressions)

- 연산자와 우선순위

Operator	Name	Precedence Level
( )	parentheses	1
+, -	unary plus, minus	2
*, /	multiply, divide	3
MOD	modulus	3
+, -	add, subtract	4

- 어셈블될 때 계산되며, 기계어로 바뀌어 실행시간에 수행되지 않음
- 예

Expression	Value
16 / 5	3
-(3 + 4) * (6 - 1)	-35
-3 + 4 * 6 - 1	20
25 mod 3	1

실수 상수

- 10진 실수

  (예)

  2.

  +3.0

  -44.2E+05

  26.E5 … 적어도 한자리 수와 소수점이 필요함

- 부호화 실수

  (예) +1.0 의 IEEE-754 32-bit 부동소수점 형식(short real)

  0011 1111 1000 0000 0000 0000 0000 0000 (2진수)

  → 3F800000**r**

문자 상수와 문자열 상수

- 문자와 문자열 구분 없이 " " 또는 ' ' 로 둘러싸여 표시
- 문자 상수
  - 'A'
  - "x"
  - ASCII 문자 = 1 byte
- 문자열 상수
  - "ABC" … 3 byte
  - 'wxyz' … 4 byte

- Embedded quotes: 다른 따옴표에 포함됨
  - 'Say "Goodnight," Gracie'
  - "This isn't a test"

예약어와 식별자

- 예약어 (부록 A.2 참조)
  - instruction mnemonics (ex) mov, add, mul
  - register (ex) eax, ebx, esi
  - directives (ex) proc, endp, end, include
  - type attributes (ex) byte, word
  - operators (ex) mod
  - predefined symbols (ex) @data
- 식별자(Identifiers)
  - 1 ~ 247 문자
  - 대소문자 구분하지 않음(case insensitive)
    - 어셈블할 때 옵션(-Cp)을 주어 구분할 수도 있음
  - 첫 문자는 다음 문자들만 가능함
    - 알파벳(letter), underscore(_), @, ? 또는 $
  - 예약어는 사용할 수 없음

(예) var1 Count _main $first @@myfile

디렉티브(Directives)

- 디렉티브
  - 어셈블러가 인식하여 어셈블할 때에 처리하는 명령어
  - 기계어와 관련이 없으며, 실행시간에 실행되지 않음
  - 대소문자를 구분 없음(case insensitive)
  - 코드, 데이터 영역(세그먼트) 선언, 메모리 모델 선택, 변수, 프로시저 선언 등과 같은 작업에 사용됨
  - (예) .data .code .model dword proc
- 디렉티브와 명령어

```
myVar   dword  26          ... 변수용 메모리 예약 디렉티브
        mov    eax, myVar   ... 데이터 전송(복사) 명령어
```

- 어셈블러와 디렉티브
  - 명령어 : 모든 어셈블러가 같은 명령어 집합을 공유
  - 디렉티브: 어셈블러마다 디렉티브 집합이 다름

명령어(Instructions)

- **명령어**
  - 어셈블러에 의해서 CPU의 명령어 집합에 속하는 기계어로 변환됨
    (cf) 디렉티브 : 기계어와 관련 없음
  - 실행시간에 메모리에 적재되어 CPU에서 실행됨
- **형식**
  - 일반적 형식

label :	*mnemonic*	*operand(s)*	; *comment*
(optional)	(required)	(usually required)	(optional)

  - 예:

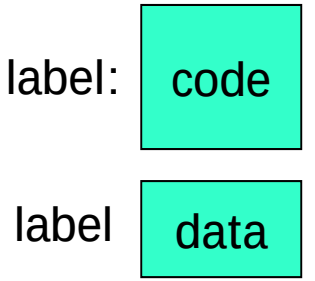
code label

data label

```
            mov   ax, 5          ; ax ← 5h
target:     inc   ax             ; ax ← ax + 1
            mov   myVar, ax      ; myVar ← ax
            ...
            jmp   target
            ...
myVar       word  100            ; 16-bit variable
```

레이블, 니모닉(Mnmonics), 피연산자

- 레이블(Label)
  - 명령어 또는 데이터의 위치를 표시하는 식별자
  - Code label (ex) target : ...
    - jump와 loop 명령어의 target (콜론을 함께사용)
  - Data label (ex) myVar byte 'A'
    - 변수 위치의 식별자(변수 값을 정의)



- 명령어 니모닉(Instruction Mnemonics)
  - 명령어의 연산의 유형에 대한 힌트를 제공하는 이름
  - 예: MOV, ADD, SUB, MUL, INC, DEC
- 피연산자(Operands)
  - 즉시값: 상수, 상수 수식 96 2+4
  - 레지스터(register) eax esi
  - 메모리: data label, ... count

피연산자 수와 명령어 형식

- No operand
  - stc ; set Carry Flag(CF) → implicit operand
- One operand
  - inc eax ; register
  - inc myByte ; memory (변수)
- Two operands
  - add ebx, ecx ; register, register
  - sub myByte, 25 ; memory, constant
  - add eax, 36*25 ; register, constant-expression

주석(Comments)

주석

- 한 줄 주석(single-line comment)

```
add  ax, 10          ; add 10 into ax
```

- 블록 주석(block comment) – 여러 줄

```
COMMENT &
    This line is a comment.
    This line is also a comment.
&
```

사용자 정의 기호

NOP 명령어

- NOP 명령어
  - 1 바이트 크기로 아무 동작도 하지 않음
  - 코드를 짝수 또는 4의 배수 주소로 정렬시키기 위해서 사용함

```
00000000  66 8B C3      mov ax,bx
00000003  90            nop             ; align next instruction
00000004  8B D1         mov edx,ecx
```

- x86 프로세서는 4의 배수의 주소에 있는 코드와 데이터(32비트 크기)를 더 빠르게 읽을 수 있음

3.2 예제: 정수의 덧셈, 뺄셈

- MS-DOS용 프로그램 (real-address mode)

```
title add and subtract          ... 프로그램 제목을 나타내는 주석

.model small                    ... 메모리 모델: real (CS,DS 각 1개)
.code                           ... 코드 세그먼트 시작
main proc                       ... 프로시저 시작 (main)
        mov ax, 1000h           ; ax ← 1000h
        add ax, 400h            ; ax ← ax + 400h
        sub ax, 200h            ; ax ← ax – 200h
        mov ax, 4c00h
        int 21h                 ; terminate the program
main endp                       ... 프로시저 끝 (main)
end main                        ... 프로그램 끝, main부터 시작함
```

- 이 코드는 memory operand를 사용하지 않음
- debug를 사용하여 동작 확인 가능 (debug는 windows 7에는 없음)

예제: 정수의 덧셈, 뺄셈

- protected mode용 32-bit 코드

```
TITLE Add and Subtract                (AddSub.asm)
; This program adds and subtracts 32-bit integers.

INCLUDE Irvine32.inc                  ... 저자 제공 파일을 include
.code
main PROC
    mov eax,10000h                    ; EAX = 10000h
    add eax,40000h                    ; EAX = 50000h
    sub eax,20000h                    ; EAX = 30000h
    call DumpRegs                     ; display registers
    exit                              ; macro for termination
main ENDP
END main
```

- Program output (call DumpRegs)
  - Call Dump Regs : 저자가 제공하는 출력 프로시저 호출
  - 레지스터와 플래그 값을 출력함

```
EAX=00030000  EBX=7FFDF000  ECX=00000101  EDX=FFFFFFFF
ESI=00000000  EDI=00000000  EBP=0012FFF0  ESP=0012FFC4
EIP=00401024  EFL=00000206  CF=0  SF=0  ZF=0  OF=0
```

AddSub의 다른 버전 – no include

```
TITLE Add and Subtract                    (AddSubAlt.asm)

; This program adds and subtracts 32-bit integers.
.386                                  ... 386 명령어 사용
.MODEL flat,stdcall                   ... 보호모드(균일세그먼트)
.STACK 4096

ExitProcess PROTO, dwExitCode:DWORD ... 함수 원형선언
DumpRegs PROTO

.code
main PROC
    mov eax,10000h                    ; EAX = 10000h
    add eax,40000h                    ; EAX = 50000h
    sub eax,20000h                    ; EAX = 30000h
    call DumpRegs
    INVOKE ExitProcess,0              ... ExitProcess 함수호출(반환값 0)
main ENDP
END main
```

Program 템플릿(Template)

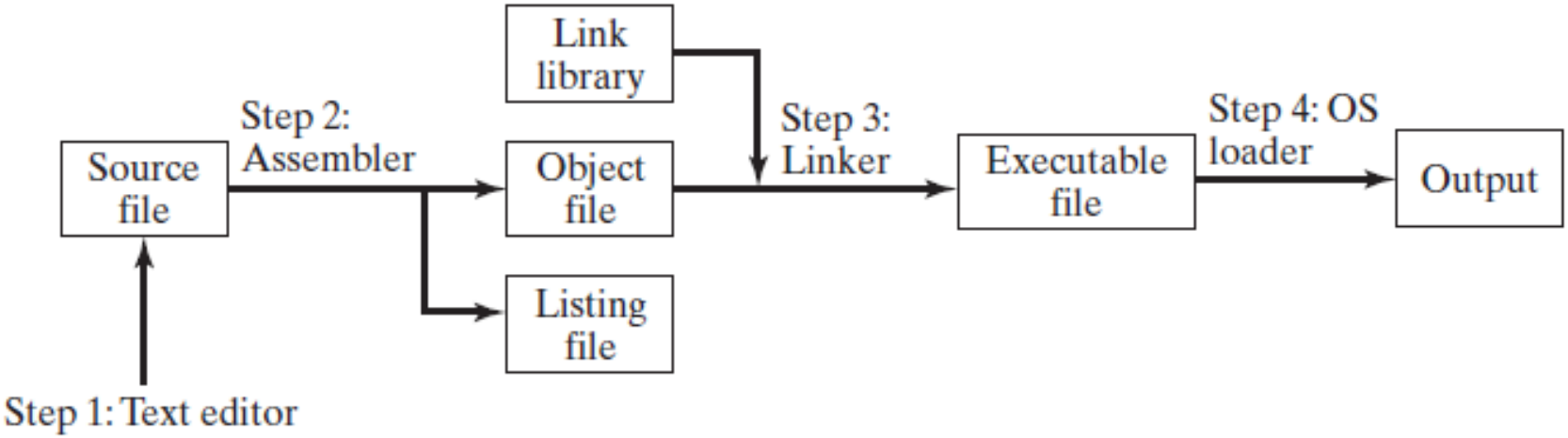
```
TITLE Program Template                (Template.asm)

; Program Description:
; Author:
; Creation Date:
; Revisions:
; Date:                 Modified by:

INCLUDE Irvine32.inc
.data
    ; (insert variables here)
.code
main PROC
    ; (insert executable instructions here)
    exit
main ENDP
    ; (insert additional procedures here)
END main
```

3.3 프로그램 어셈블, 링크, 실행

- Assemble-Link-Execute 사이클



1. text editor사용 → source file 작성 (확장자 .asm)
2. assembler 실행 → object file 생성 (확장자 .obj)
3. linker 실행 → executable file 생성 (확장자 .exe)
4. loader 실행 → 프로그램 실행

어셈블러와 링커

- 어셈블러(Assembler)
  - MASM 6.xx 까지는 어셈블러를 별도로 설치해야함
  - Visual C++ 2008 이후부터는 어셈블러를 포함하고 있음
    - ML.EXE Microsoft Macro Assembler 9.00
- 링커
  - LINK.EXE 32-bit code linker
  - LINK16.EXE 16-bit code linker (저자 제공)
    - link할 object file과 library file 이름을 인수로 주어야 함
- 어셈블리 편의를 위한 batch 파일 (저자 제공)
  - Assembler와 Linker를 연속하여 수행, 필요한 인수 제공
    - MAKE16.BAT 16-bit source program용 (DOS용)
    - ASM32.BAT 32-bit source program용 (Windows용)
    - MAKE32.BAT 저자가 제공한 batch 파일을 수정한 것
  - 명령어 기반 실행을 위해서는 적절한 path 설정이 필요

어셈블러 Batch 프로그램 실행하기

- 시작-실행-cmd (또는 command)
- 경로설정

  C> path %path%;c:\Irvine

  또는 [제어판 > 시스템 > 고급 > 환경변수] 선택 후 PATH 변수 설정

(cf) [시작-Microsoft VisualStudio2010 – VistualStudioTools – **VisualStudio2010명령프롬프트**] **를 선택하면 명령어 창을 생성하면** PATH **설정 불필요**

- source 프로그램이 있는 directory로 이동

  C> cd c:\user\gdhong

- assemble & link

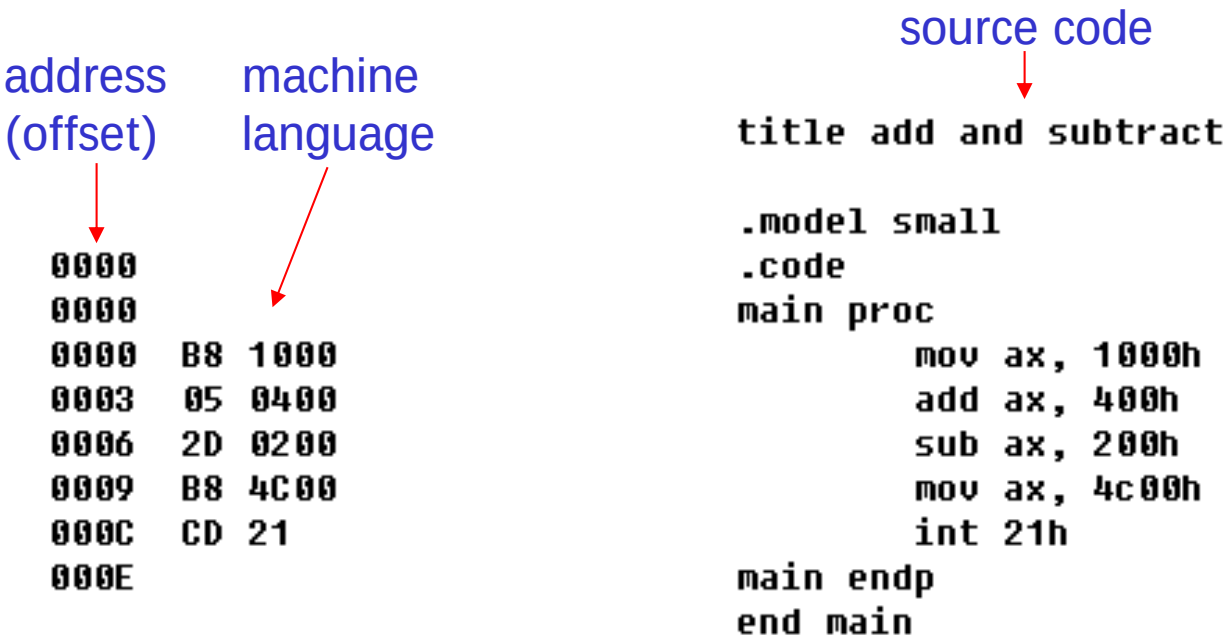
  C> make32 asmfile **또는**

  C> make16 asmfile

  (확장자 .asm**은 생략)**

리스트 파일(Listing File)

- 리스트 파일 (확장자 .lst) : 다음 항목을 포함
  - source code, addresses, object code (machine language)
  - segment names, symbols (variables, procedures, and constants)
- 예:



```
Segments and Groups:

                N a m e                 Size     Length   Align   Combine Class

DGROUP . . . . . .              GROUP
_DATA  . . . . . .              16 Bit   0000    Word    Public  'DATA'
_TEXT  . . . . . .              16 Bit   000E    Word    Public  'CODE'


Procedures,  parameters and locals:

                N a m e                 Type     Value    Attr

main . . . . . .                P Near   0000    _TEXT Length= 000E Public

Symbols:

                N a m e                 Type     Value    Attr

@CodeSize  . . . . . .          Number   0000h
@DataSize  . . . . . .          Number   0000h
@Interface . . . . . .          Number   0000h
@Model . . . . . .              Number   0002h
@code  . . . . . .              Text     _TEXT
@data  . . . . . .              Text     DGROUP
@fardata?  . . . . . .          Text     FAR_BSS
@fardata . . . . . .            Text     FAR_DATA
@stack . . . . . .              Text     DGROUP
```

3.4 데이터 정의

정수 자료형

BYTE, SBYTE (DB)	8-bit unsigned / signed integer
WORD, SWORD (DW)	16-bit unsigned / signed integer
DWORD, SDWORD (DD)	32-bit unsigned / signed integer
QWORD (DQ)	64-bit integer
FWORD	48-bit integer (16-bit segment:32-bit offset)
TBYTE	80-bit integer

- DB, DW, DD, DQ는 **구형으로서 사용을 권장하지 않음**

실수 자료형

REAL4	4-byte(32-bit) real, single precision
REAL8	8-byte (64-bit) real, double precision
REAL10	10-byte (80-bit) real, extended

- IEEE 754 standard format

데이터 정의문

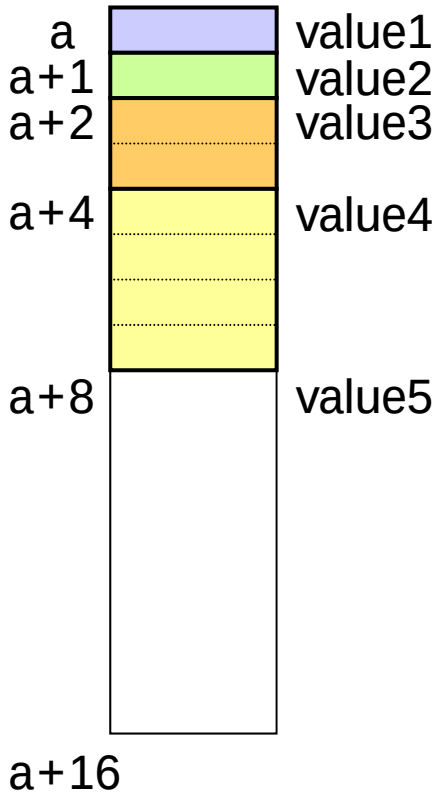
데이터 정의문

- 메모리에 변수용 메모리 공간을 확보함

예

```
value1  BYTE        127
value2  SBYTE       -50
value3  WORD        65535
value4  SDWORD      12345678h
value5  QWORD       ?
```

- ? : uninitialized data



구식 디렉티브 사용

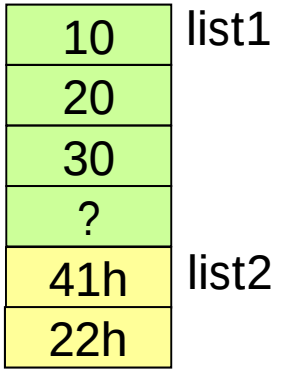
```
a   DB  'A'
b   DW 55aah
```

여러 개의 초기값 정의

여러 개의 초기값 정의

```
list1 BYTE 10,20,30,?
list2 BYTE 41h, 00100010b
```

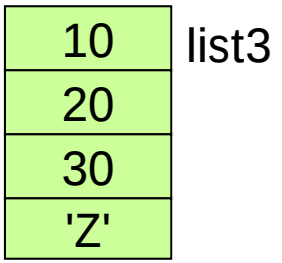
- 초기값들은 서로 다른 진수를 사용가능



레이블이 없는 데이터 정의문

```
list3 BYTE 10, 20
      BYTE 30, 'Z'
```

- 초기값들을 계속하여 정의할 때에 주로 사용



문자열 정의

- 문자열은 문자의 배열로 구현됨
  - 문자열은 따옴표로 둘러싸서 표기 (큰 따옴표 또는 작은 따옴표)
  - 대개 널 바이트(0)으로 명시적으로 종료함 → 널 종료 문자열

- 예:

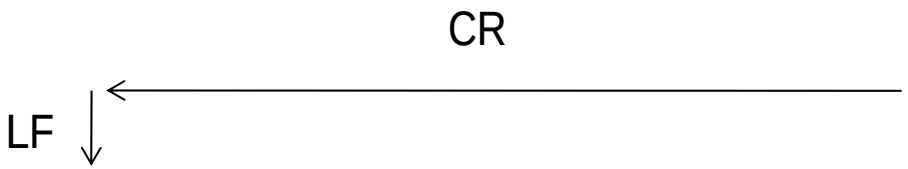
명시적 사용

```
str1 BYTE "Enter your name",0
str2 BYTE 'Error: halting program',0
str3 BYTE 'A','E','I','O','U'
greeting  BYTE "Welcome to the  Demo program "
          BYTE "created by Kip Irvine.",0
greeting1 BYTE "Welcome to the  Demo program ",
               "created by Kip Irvine.",0
```

하나의 문자열

End of Line 문자열

- CR/LF 문자열 (end-of-line 문자열)
  - 0Dh = carriage return(CR) … 커서를 line 처음으로 이동
  - 0Ah = line feed(LF) … 커서를 다음 line으로 이동)




```
str1 BYTE "Enter your name:    ",0Dh,0Ah
     BYTE "Enter your address: ",0
newLine BYTE 0Dh,0Ah,0
```

DUP 연산자

DUP 연산자

- 상수를 반복 카운터로 사용하여 여러 개의 데이터용 공간 확보
- 배열과 문자열 공간 할당에 유용

Array byte 20 dup (0)

카운터 초기값인수

- 카운터와 초기값 인수는 상수 또는 상수 수식이어야 함

```
var1 BYTE 20 DUP(0)          ; 20 bytes, all equal to zero
var2 BYTE 20 DUP(?)          ; 20 bytes, uninitialized
var3 BYTE 4 DUP("STACK")     ; 20 Bytes
                             ; "STACKSTACKSTACKSTACK"
var4 BYTE 10,3 DUP(0),20
```

WORD, SDWORD 데이터 정의

- 워드(16-bit) 데이터 정의

```
word1   WORD    65535           ; largest unsigned value
word2   SWORD   -32768          ; smallest signed value
word3   WORD    ?               ; uninitialized, unsigned
```

- 워드 배열

```
myList  WORD 1,2,3,4,5
```

Offset	Value
0000:	1
0002:	2
0004:	3
0006:	4
0008:	5

DWORD, SDWORD 데이터 정의

- 더블워드(32비트) 데이터 정의

```
val1 DWORD   12345678h          ; unsigned
val2 SDWORD -2147483648         ; signed
val3 DWORD   20 DUP(?)          ; unsigned array
```

- 더블워드 배열

```
myList DWORD 1,2,3,4,5
```

Offset	Value
0000:	1
0004:	2
0008:	3
000C:	4
0010:	5

- 변수의 오프셋 주소(32비트) 저장

```
pVal DWORD val3
```

QWORD, TBYTE 데이터 정의

- 64비트 데이터 정의

```
quad1 QWORD 1234567812345678h
```

- 10바이트 packed BCD 데이터 정의
  - 각 바이트를 10진수 1자리 저장에 사용함

```
intVal TBYTE 80000000000001234h
```

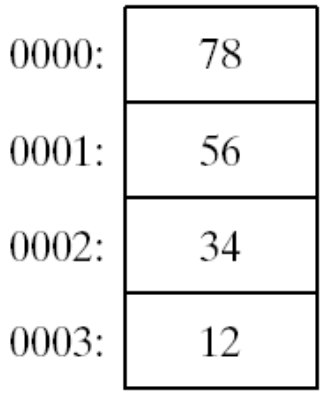
Decimal Value	Storage Bytes
+1234	34 12 00 00 00 00 00 00 00 00

  - TBYTE 초기값을 16진수로 정의하는 것에 유의
  - packed BCD에 대해서는 7장에서 소개

Little Endian 순서

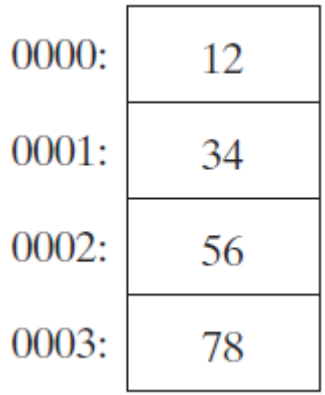
- Little Endian 순서
  - 여러 바이트로 구성된 데이터 자료형을 메모리의 연속된 주소에 저장되며, 최하위 바이트는 첫번째(최하위) 메모리 주소에 저장됨
  - x86 프로세서는 이 순서를 사용함
- 예:

```
val1 DWORD 12345678h
```



(cf) Big Endian 순서

- 최상위 바이트가 첫번째(최하위) 메모리 주소에 저장됨



AddSub2: AddSub 프로그램에 변수 추가

```
TITLE Add and Subtract, Version 2                (AddSub2.asm)
; This program adds and subtracts 32-bit unsigned
; integers and stores the sum in a variable.
INCLUDE Irvine32.inc
.data
val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
finalVal DWORD ?
.code
main PROC
    mov eax,val1                ; start with 10000h
    add eax,val2                ; add 40000h
    sub eax,val3                ; subtract 20000h
    mov finalVal,eax            ; store the result (30000h)
    call DumpRegs               ; display the registers
    exit
main ENDP
END main
```

비초기화 데이터 선언

- .data? 디렉티브
  - 비초기화 데이터 세그먼트 선언에 사용함
  - 이 세그먼트에 있는 변수는 "?" 초기값을 사용해야 함
  - 장점: 이 세그먼트를 실행파일에 포함시키지 않아서 프로그램 실행파일 크기를 줄여줌
- Example

```
.data
smallArray DWORD 10 DUP(0)
.data?
bigArray DWORD 5000 DUP(?)
```

→ 40B

→ 실행파일에 포함되지 않음

```
.data
smallArray DWORD 10 DUP(0)
bigArray DWORD 5000 DUP(?)
```

→ 40B

→ 20KB

실행파일크기 > 20KB

Code와 Data의 혼합

- code 세그먼트와 data 세그먼트 영역을 섞어서 작성 가능함
- 어셈블러가 같은 세그먼트끼리 모아놓음

```
.code
        move eax, ebx
.data
temp DWORD ?
.code
        mov temp, eax
```

같은 세그먼트끼리 합쳐져서
명령어의 실행흐름을 방해하지 않음

```
.code
        move eax, ebx
        mov temp, eax
```

```
.data
temp DWORD ?
```

3.5 기호 상수

- 기호(symbol) 상수
  - 정수 수식 또는 텍스트를 식별자(기호)와 연관시켜 정의
  - 정수수식 또는 텍스트 대신 사용할 수 있음
- 기호 상수와 변수의 비교

	Symbol	Variable
저장공간 사용	no	yes
실행시간에 값의 변화	no	yes

- 기호 상수 정의용 디렉티브
  - 등호 (=)
  - EQU
  - TEXTEQU

등호(=) 디렉티브

- *name = expression*
  - name: 기호이름(식별자), 기호상수
  - expression: a 32-bit 정수 (상수, 상수 수식)

```
COUNT = 500
. . .
mov al, COUNT                    ; 500
```

- 값을 재정의 가능

```
COUNT = 5
mov al,COUNT                     ; AL = 5
COUNT = 10
mov al,COUNT                     ; AL = 10
COUNT = 100
mov al,COUNT                     ; AL = 100
```

- 상수가 나중에 바뀔 가능성이 있다면 기호상수를 정의하여 사용하는 것이 프로그램 유지보수에 좋음

배열과 문자열의 크기 계산

- 현재 위치 카운터(location counter): $
  - 현재의 프로그램 문장에 대한 오프셋을 나타냄
- 배열의 크기 계산
  - 배열 자료를 정의한 바로 다음에서 다음 형태의 등호 디렉티브를 사용하여 계산

    size = ($ - array시작번지) / (array원소 크기)

```
list BYTE 10,20,30,40
ListSize = ($ - list)
```

byte array

```
list WORD 1000h,2000h,3000h,4000h
ListSize = ($ - list) / 2
```

word array

```
list DWORD 1,2,3,4
ListSize = ($ - list) / 4
```

dword array

EQU 디렉티브

- EQU
  - 기호를 정수 또는 텍스트와 연관시킴

```
name EQU expression          ; integer expression
name EQU symbol              ; existing symbol name
name EQU <text>              ; any text
```

  - 재정의 될 수 없음

```
matrix1 EQU 10 * 10          → 100
matrix2 EQU <10 * 10>        → 10 * 10
matrix3 EQU matrix1
```

```
PI EQU <3.1416>
pressKey EQU <"Press any key to continue...",0>
.data
prompt BYTE pressKey
```

TEXTEQU 디렉티브

- TEXTEQU (텍스트 매크로)
  - 3가지 형식

    name TEXTEQU <text>

    name TEXTEQU textmacro

    name TEXTEQU %constExpr
  - 재정의 가능

```
continueMsg TEXTEQU <"Do you wish to continue (Y/N)?">
rowSize = 5
.data
prompt1 BYTE continueMsg

count   TEXTEQU %(rowSize * 2) ; evaluates the expression <10>
move    TEXTEQU <mov>
setupAL TEXTEQU <move al,count>
.code
setupAL                        ; generates: "mov al,10"
```

3.6 Real-Address Mode 프로그래밍

- 16-bit MS-DOS 프로그램 생성
- 장점
  - MS-DOS 및 BIOS 함수 호출 가능
  - 메모리 접근 제한 없음 (가상 8086 모드에서는 제한이 있음)
- 단점
  - 세그먼트와 오프셋을 인식해야 함
  - Win32 function을 호출할 수 없음
  - 사용가능 메모리가 640KB에서 제한됨 (DOS 사용공간 포함)
- 수정사항
  - INCLUDE Irvine16.inc → 저자가 제공하는 파일
  - DS값을 프로그램의 데이터 세그먼트 시작주소로 초기화해야 함

    mov ax, @data

    mov ds, ax → DS = @data
- make16.bat를 이용하여 어셈블리 … 저자 제공
- Windows 7이상 64비트 운영체제에서 지원하지 않음

AddSub2, 16-Bit Version

```
TITLE Add and Subtract, Version 2      (AddSub2.asm)
INCLUDE Irvine16.inc                ; (changed)
.data
val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
finalVal DWORD ?
.code
main PROC
    mov ax,@data                    ; initialize DS (new)
    mov ds,ax                       ;                (new)
    mov eax,val1                    ; get first value
    add eax,val2                    ; add second value
    sub eax,val3                    ; subtract third value
    mov finalVal,eax                ; store the result
    call DumpRegs                   ; display registers
    exit
main ENDP
END main
```