Chapter 3:
Assembly Language Fundamentals

Assembly Language 소개

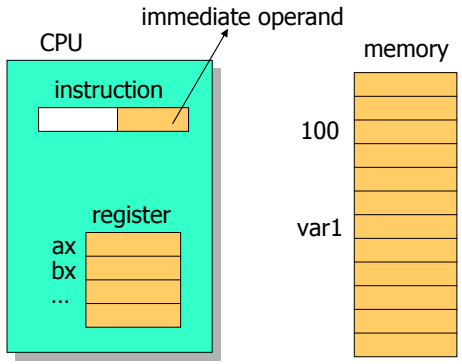
- Assembly language instruction
  - a symbolic representation of a single machine instruction
  - consists of (1) mnemonic and (2) a list of operands
    - mnemonic: CPU 명령어의 기억을 돕는 짧은 이름
- Examples

```
clc          ; no operand, just a mnemonic
inc ax       ; single operand
mov ax, bx   ; two operand
```

- An operands can be a register, a variable/a memory location, or an immediate value

10	immediate value	
ax	register	
count	variable	memory
[200]	memory location	memory

Operand의 위치



DEBUG 명령어와 assembly program 작성

- debug
  - MS-DOS용 utility 프로그램
  - 16-bit x86용 프로그램의 디버깅 등의 기능 수행
- Sample assembly language program for debug (DOS용)

```
mov ax, 5        ; ax ← 5h
add ax, 10       ; ax ← ax + 10h
sub ax, 6        ; ax ← ax – 1h
mov [120], ax    ; M[DS:120] ← ax
int 20           ; half the program (DOS system call)
```

- debug용 주요 명령어

a : assemble	r : display registers
g : go (execute)	u : unassemble
q : quit	t : trace
d : dump memory	p : proceed program

- debug 실행

C> debug

- a 100 ; 100번지부터 프로그램 입력 (assemble)

```
1A3C:0100 mov ax,5
1A3C:0103 add ax,10
1A3C:0106 sub ax,4
1A3C:0109 mov [120], ax
1A3C:010C int 20
1A3C:010E
```

- p 또는 t ; 프로그램 단계적 수행

```
AX=0005  BX=0000 CX=0000  DX=0000  SP=FFEE  BP=0000 SI=0000 DI=0000
DS=1A3C  ES=1A3C  SS=1A3C  CS=1A3C  IP=0103   NV UP EI PL NZ NA PE NC
1A3C:0103 051000       ADD     AX,0010
```

- g ; 프로그램 실행 (현재의 IP부터)
- g 100 ; 100번지부터 실행

- debug 실행 (계속)
  - d 120 ; 메모리 내용 출력
  - d 120 121 ; 120번지부터 121번지까기 내용 출력
  - u ; 메모리 내용 unassemble
  - u 100 ; 100번지부터 unassemble
  - r ; register 내용 출력
  - r ax ; register AX 내용 변경

    AX 000F 현재의 값

    : 0015 (단순히 enter입력시 변경되지 않음)
  - q ; debug 종료
- debug 프로그램의 제한점
  - 복잡한 프로그램 작성 불가능
  - 80386이상에서 제공되는 명령어 사용 불가

3.1 Basic Elements of Assembly Language

- Integer constants
- Integer expressions
- Character and string constants
- Reserved words and identifiers
- Directives and instructions
- Labels
- Mnemonics and Operands
- Comments
- Examples

Integer Constants

- Optional leading + or – sign
  - 5 +5 -5
- binary, decimal, hexadecimal, or octal digits
  - d – decimal (ex) 26, 26d
  - h – hexadecimal (ex) 2Bh, 55H, 0A3h → hexadecimal beginning with letter
  - b – binary (ex) 10110001b
  - o, q – octal (ex) 42q, 42o
  - r – encoded real (실수를 2진수형태로 표기, IEEE754 표준형식)

Integer Expressions

- Operators and precedence levels:

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

- Examples:

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

Character and String Constants

- Character constants
  - 'A', "x" ; enclosed in either single or double quotes
  - ASCII character = 1 byte
- String constants
  - "ABC", 'xyz' ; Enclose strings in single or double quotes
  - Each character occupies a single byte
- Embedded quotes:
  - 'Say "Goodnight," Gracie'
  - "This isn't a test"

Reserved Words and Identifiers

- Identifiers
  - 1-247 characters, including digits
  - case insensitive (by default)
  - first character must be a letter, underscore(_), @, or $

  (examples)

  var1 Count _main $first @@myfile

- Reserved words (Appendix D) cannot be used as identifiers
  - instruction mnemonics (ex) mov, add, mul
  - directives (ex) proc, endp, end, include
  - type attributes (ex) byte, word
  - operators (ex) mod
  - predefined symbols (ex) @data

Directives

- Directives
  - command understood by the assembler
  - but, not part of Intel instruction set
  - case insensitive
  - Used to declare code, data areas, select memory model, declare procedures, etc.

    (ex) .data, .code, .model, proc
- Different assemblers have different directives
  - TASM ≠ MASM, for example

Instructions

- Instructions
  - Assembled into machine code by assembler
  - Executed at runtime by the CPU after being loaded into memory.
  - Member of the Intel IA-32 instruction set (cf) directives
- Parts
  - general format

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

  - example:

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

instruction label → start:
data label → myVar

Labels, Mnemonics and Operands

- Label
  - Act as place markers
  - Code label (ex) target: …
    - target of jump and loop instructions
  - Data label (ex) myVar byte 'A'
    - define variables
- Instruction Mnemonics
  - "reminder"
  - examples: MOV, ADD, SUB, MUL, INC, DEC
- Operands
  - immediate value: constant, constant expression
  - register
  - memory: data label

label: code

label data

Instruction Format Examples

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

Comments

- Comments
  - single-line comment

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

  - multi-line comment

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

a user-specified symbol

3.2 Example: Adding and Subtracting Integers

- MS-DOS용 코드 (real-address mode)

```
title add and subtract

.model small
.code
main proc
       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
end main
```

  - 이 코드는 memory operand를 사용하지 않음
  - debug를 사용하여 동작 확인

Example: Adding and Subtracting Integers

- protected mode용 32-bit 코드

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

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


Example Output

- Program output (call DumpRegs)
  - shows registers and flags:

```
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
```


Alternative Version of AddSub

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

; This program adds and subtracts 32-bit integers.
.386
.MODEL flat,stdcall
.STACK 4096

ExitProcess PROTO, dwExitCode:DWORD
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
main ENDP
END main
```

Program Template

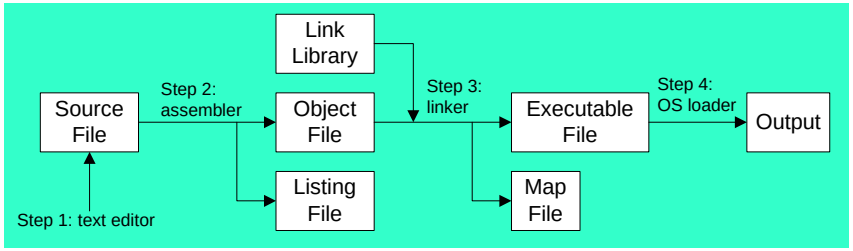
```
TITLE Program Template              (Template.asm)

; 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 Assembling, Linking, and Running Programs

- Assemble-Link-Execute Cycle



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 → 실행

Assembler and Linker

- Assembler
  - MASM.EXE Microsoft Macro Assembler (6.11)
  - ML.EXE Microsoft Macro Assembler (6.15)
    - object file이외에도 listing file과 map file 생성
- Linker
  - LINK.EXE 16-bit code liker
  - LINK32.EXE 32-bit code linker
    - link할 object file과 library file 이름을 인수로 주어야 함
- 교재의 저자가 제공하는 batch file
  - MAKE32.BAT 32-bit source program용 (Windows용)
  - MAKE16.BAT 16-bit source program용 (DOS용)
    - Assembler와 Linker를 연속하여 수행, 필요한 인수 제공

Running Assembler Batch program

- 시작-실행-cmd (또는 command)
- 경로설정

  C> path %path%;c:\masm615
- source 프로그램이 있는 directory로 이동

  C> cd c:\user\gdhong
- assemble & link

  C> make32 srcfile 또는

  C> make16 srcfile (확장자 .asm은 생략)
  - assemble & link가 성공하면 dir 명령어를 수행하고 키보드 입력을 기다림

Listing File

- Listing File (확장자 .lst) contains
  - source code, addresses, object code (machine language)
  - segment names, symbols (variables, procedures, and constants)
- Example:

address (offset) / machine language / source code

```
0000                            title add and subtract
0000                            
0000                            .model small
0000                            .code
0000  B8 1000                   main proc
0003  05 0400                           mov ax, 1000h
0006  2D 0200                           add ax, 400h
0009  B8 4C00                           sub ax, 200h
000C  CD 21                             mov ax, 4c00h
000E                                    int 21h
                                main endp
                                end main
```



```
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
```


Map File

- Information about each program segment:
  - segment starting address, length, name, type
  - a list of public symbols, address of the entry point
- Example:

```
Timestamp is 3cfae306 (Sun Jun 02 23:31:18 2002)

Preferred load address is 00400000

Start         Length     Name                   Class
0001:00000000 00001c40H .text                  CODE
0002:00000000 00000121H .rdata                 DATA
0002:00000121 00000000H .edata                 DATA
0003:00000000 00000e03H .data                  DATA
0003:00000e04 00000224H .bss                   DATA

 Address         Publics by Value              Rva+Base     Lib:Object

0001:00000010       _main@0                    00401010 f   AddSub.obj
0001:00000034       _ClrScr@0                  00401034 f   irvine32:Irvine32.obj
0001:00000083       _Crlf@0                    00401083 f   irvine32:Irvine32.obj

entry point at        0001:00000010
```


3.4 Defining Data

- Integer data types

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

- Real data types

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

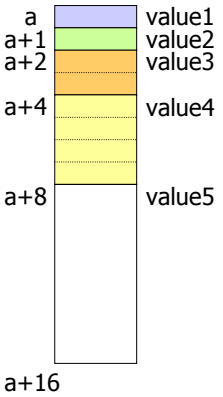
  - IEEE 754 standard format

Data Definition Statement

- A data definition statement
  - sets aside storage in memory for a variable.
- Example

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

  - ? : uninitialized data



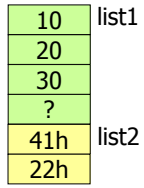
- Earlier version

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


Multiple Initializers and Optional Label

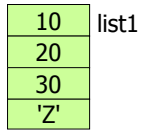
- Multiple initializer

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



- Optional Label

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



Defining Strings

- A string is implemented as an array of characters
  - For convenience, it is usually enclosed in quotation marks (single or double)
  - It usually has a null byte at the end
- Examples:

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 string

- End-of-line character sequence:
  - 0Dh = carriage return (커서를 line 처음으로 이동)
  - 0Ah = line feed (커서를 다음 line으로 이동)

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

DUP Operator

- Use DUP to allocate (create space for) an array or string.
- Counter and argument must be constants or constant expressions

```
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
```


Defining Real Data

- Storage definitions for real numbers
  - intializer is an real number constant initializer or ?

```
rVal1 REAL4   -2.1              ; 32-bit (float)
rVal2 REAL8   3.2E-260          ; 64-bit (double)
rVal3 REAL10 4.6E+4096          ; 80-bit (long double)
ShortArray REAL4 20 DUP(?)
```


Little Endian Order

- Little Endian Order
  - All data types larger than a byte store their individual bytes in reverse order.
  - The least significant byte occurs at the first (lowest) memory address.
- Example:
  val1 DWORD 12345678h

0000:	78
0001:	56
0002:	34
0003:	12

Adding Variables to AddSub

```
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
```

Declaring Uninitialized Data

- `.data?` directive
  - used it to declare an uninitialized data segment:
  - Within the segment, declare variables with "?" initializers:
  - advantage: the program's EXE file size is reduced
- Example

```
.data
smallArray DWORD 10 DUP(0)      → 40B
.data?
bigArray DWORD 5000 DUP(?)      → 실행파일에 포함되지 않음
```

```
.data
smallArray DWORD 10 DUP(0)      → 40B
bigArray DWORD 5000 DUP(?)      → 20KB
```

3.5 Symbolic Constants

- Symbolic constants
  - associates an identifier and either an integer expression or some text

	Symbol	Variable
Uses storage?	no	yes
Value changes at run time	no	yes

- Directives for symbolic constants
  - equal-sign (=)
  - EQU
  - TEXTEQU

Equal-Sign Directive

- *name* = *expression*
  - expression is a 32-bit integer (expression or constant)
  - may be redefined
  - *name* is called a symbolic constant
- good programming style to use symbols

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

Calculating the Size of an Array

- Current location counter: $
- The size of an array
  - 배열 자료를 정의한 바로 다음에서 다음 식을 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 Directive

- EQU
  - Define a symbol as either an integer or text expression.
    - name EQU expression ; integer expression
    - name EQU symbol ; existing symbol name
    - name EQU <text> ; any text
  - Cannot be redefined

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

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


TEXTEQU Directive

- TEXTEQU (called text macro)
  - Define a symbol as either an integer or text expression.
    - name TEXTEQU <text>
    - name TEXTEQU textmacro
    - name TEXTEQU %constExpr
  - Can be redefined

```
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 Programming

- Generate 16-bit MS-DOS Programs
- Advantages
  - enables calling of MS-DOS and BIOS functions
  - no memory access restrictions
- Disadvantages
  - must be aware of both segments and offsets
  - cannot call Win32 functions (Windows 95 onward)
  - limited to 640K program memory
- Requirements
  - INCLUDE Irvine16.inc → 저자가 제공하는 파일
  - Initialize DS to the data segment:
    - mov ax,@data
    - mov ds,ax → DS = @data

Add and Subtract, 16-Bit Version

```
TITLE Add and Subtract, Version 2      (AddSub2.asm)
INCLUDE Irvine16.inc
.data
val1 DWORD 10000h
val2 DWORD 40000h
val3 DWORD 20000h
finalVal DWORD ?
.code
main PROC
    mov ax,@data              ; initialize DS
    mov ds,ax
    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
```