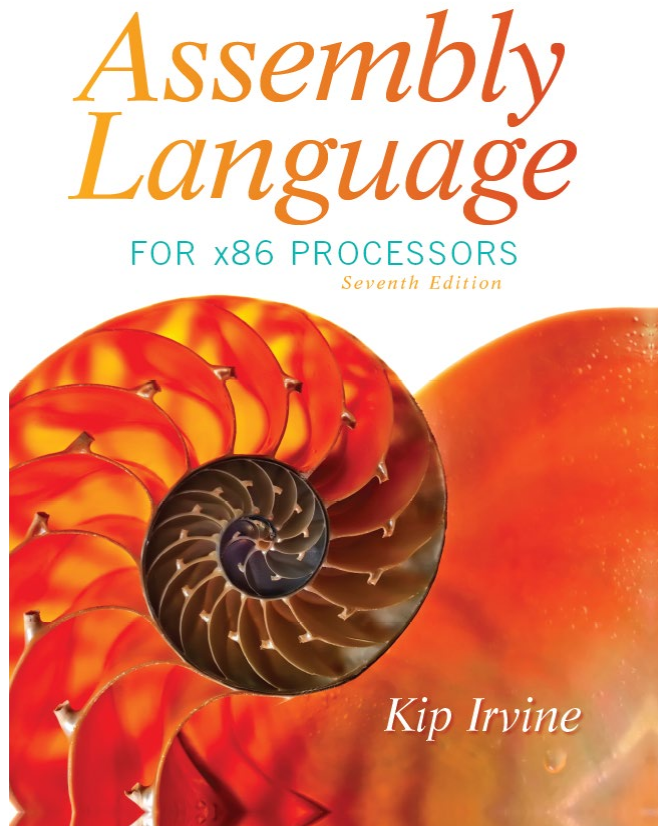


Assembly Language for x86 Processors

Seventh Edition



Chapter 3

Assembly Language Fundamentals

Chapter Overview

- Basic Elements of Assembly Language
- Example: Adding and Subtracting Integers
- Assembling, Linking, and Running Programs
- Defining Data
- Symbolic Constants

Integer Constants

- Optional leading + or – sign
- binary, decimal, hexadecimal, or octal digits
- Common radix characters:
 - h- hexadecimal
 - d- decimal
 - b-binary
 - r- encoded real

Examples: 30d, 6Ah, 42, 1101b

Hexadecimal beginning with letter: 0A5h

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

Reserved Words and Identifiers

- Reserved words cannot be used as identifiers
 - Instruction mnemonics, directives, type attributes, operators, predefined symbols
- Identifiers
 - 1-247 characters, including digits
 - not case sensitive
 - first character must be a letter, _, @, ?, or \$

Directives

- Commands that are recognized and acted upon by the assembler
 - Not part of the Intel instruction set
 - Used to declare code, data areas, select memory model, declare procedures, etc.
 - not case sensitive
- Different assemblers have different directives
 - NASM not the same as MASM

Instructions

- Assembled into machine code by assembler
- Executed at runtime by the CPU
- We use the Intel IA-32 instruction set
- An instruction contains:
 - Label (optional)
 - Mnemonic (required)
 - Operand (depends on the instruction)
 - Comment (optional)

Labels

- Act as place markers
 - marks the address (offset) of code and data
- Follow identifier rules
- Data label
 - must be unique
 - example: **myArray** (not followed by colon)
- Code label
 - target of jump and loop instructions
 - example: **L1:** (followed by colon)

Mnemonics and Operands

- Instruction Mnemonics
 - memory aid
 - examples: MOV, ADD, SUB, MUL, INC, DEC
- Operands
 - constant
 - constant expression
 - register
 - memory (data label)

Constants and constant expressions are often called immediate values

Comments (1 of 2)

- Comments are good!
 - explain the program's purpose
 - when it was written, and by whom
 - revision information
 - tricky coding techniques
 - application-specific explanations
- Single-line comments
 - begin with semicolon (;)

Comments (2 of 2)

- Multi-line comments
 - begin with COMMENT directive and a programmer-chosen character
 - end with the same programmer-chosen character

Instruction Format Examples

- No operands
 - `stc` ; set Carry flag
- 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

Example: Adding and Subtracting Integers

`; AddTwo.asm - adds two 32-bit integers`

`.386`

`.model flat,stdcall`

`.stack 4096`

`ExitProcess PROTO, dwExitCode:DWORD`

`.code`

`main PROC`

`mov eax,5 ; move 5 to the EAX register`

`add eax,6 ; add 6 to the EAX register`

`INVOKE ExitProcess,0`

`main ENDP`

`END main`

Program Template

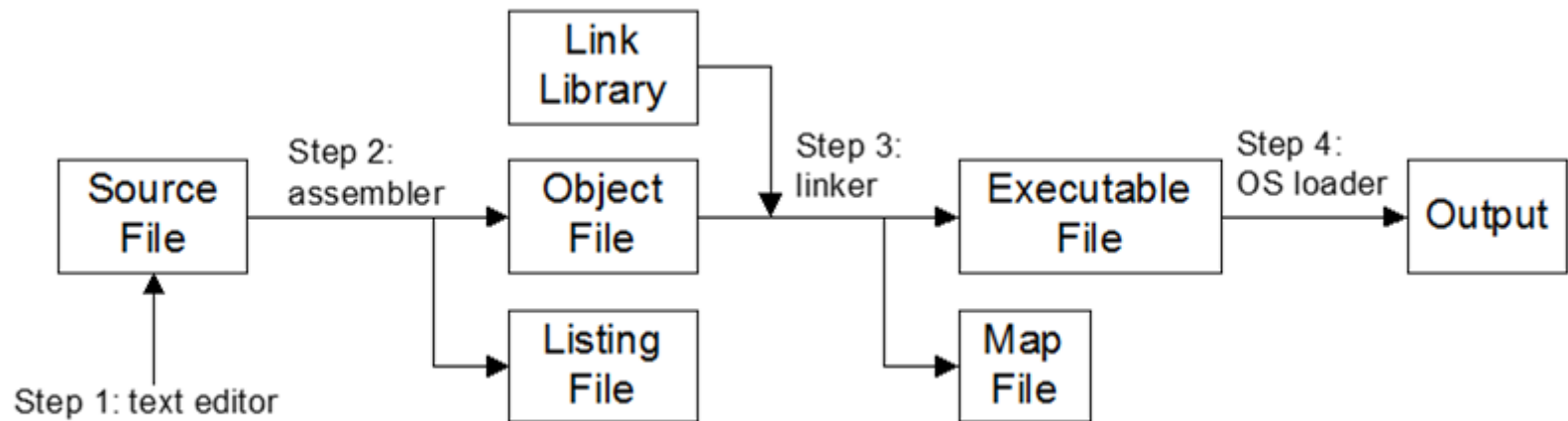
```
; Program Template                (Template.asm)

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

.386
.model flat,stdcall
.stack 4096
ExitProcess PROTO, dwExitCode:DWORD
.data
; declare variables here
.code
main PROC
    ; write your code here
    INVOKE ExitProcess,0
main ENDP
; (insert additional procedures here)
END main
```

Assemble-Link Execute Cycle

- The following diagram describes the steps from creating a source program through executing the compiled program.
- If the source code is modified, Steps 2 through 4 must be repeated.



Intrinsic Data Types

- BYTE, SBYTE
 - 8-bit unsigned integer; 8-bit signed integer
- WORD, SWORD
 - 16-bit unsigned & signed integer
- DWORD, SDWORD
 - 32-bit unsigned & signed integer
- QWORD
 - 64-bit integer
- TBYTE
 - 80-bit integer

Data Definition Statement

- A data definition statement sets aside storage in memory for a variable.
- May optionally assign a name (label) to the data

Syntax:

[name] directive initializer [,initializer] . . .

value1 *BYTE* *10*



- All initializers become binary data in memory

Defining BYTE and SBYTE Data

- Each of the following defines a single byte of storage:
- MASM does not prevent you from initializing a BYTE with a negative value, but it's considered poor style.
- If you declare a SBYTE variable, the Microsoft debugger will automatically display its value in decimal with a leading sign.

```
value1 BYTE 'A'           ; character constant
value2 BYTE 0              ; smallest unsigned byte
value3 BYTE 255            ; largest unsigned byte
value4 SBYTE -128          ; smallest signed byte
value5 SBYTE +127          ; largest signed byte
value6 BYTE ?              ; uninitialized byte
```

Defining Byte Arrays

- Examples that use multiple initializers:

```
list1 BYTE 10,20,30,40
```

```
list2 BYTE 10,20,30,40
```

```
        BYTE 50,60,70,80
```

```
        BYTE 81,82,83,84
```

```
list3 BYTE ?,32,41h,00100010b
```

```
list4 BYTE 0Ah,20h,'A',22h
```

Defining Strings (1 of 3)

- A string is implemented as an array of characters
 - For convenience, it is usually enclosed in quotation marks
 - It often will be null-terminated

- Examples:

```
str1 BYTE "Enter your name",0
```

```
str2 BYTE 'Error: halting program',0
```

```
str3 BYTE 'A','E','I','O','U'
```

```
greeting BYTE "Welcome to the Encryption Demo program "  
            BYTE "A demo from Li. ", 0
```

Defining Strings (2 of 3)

- To continue a single string across multiple lines, end each line with a comma:

```
menu BYTE "Checking Account",0dh,0ah,0dh,0ah,  
        "1. Create a new account",0dh,0ah,  
        "2. Open an existing account",0dh,0ah,  
        "3. Credit the account",0dh,0ah,  
        "4. Debit the account",0dh,0ah,  
        "5. Exit",0ah,0ah,  
        "Choice> ",0
```

Defining Strings (3 of 3)

- End-of-line character sequence:
 - 0Dh = carriage return
 - 0Ah = line feed
- Idea: Define all strings used by your program in the same area of the data segment.

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

```
newLine BYTE 0Dh,0Ah,0
```

Using the DUP Operator

- Use DUP to allocate (create space for) an array or string.
Syntax: counter DUP (argument)
- 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      ; 5 bytes
```

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

- Use the `.data?` directive to declare an uninitialized data segment:

`.data?`

- Within the segment, declare variables with "?" initializers:

`smallArray DWORD 10 DUP(?)`

- Advantage: the program's EXE file size is reduced.

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 ax,COUNT
```

Calculating the Size of a Byte Array

- current location counter: \$
 - subtract address of list
 - difference is the number of bytes

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