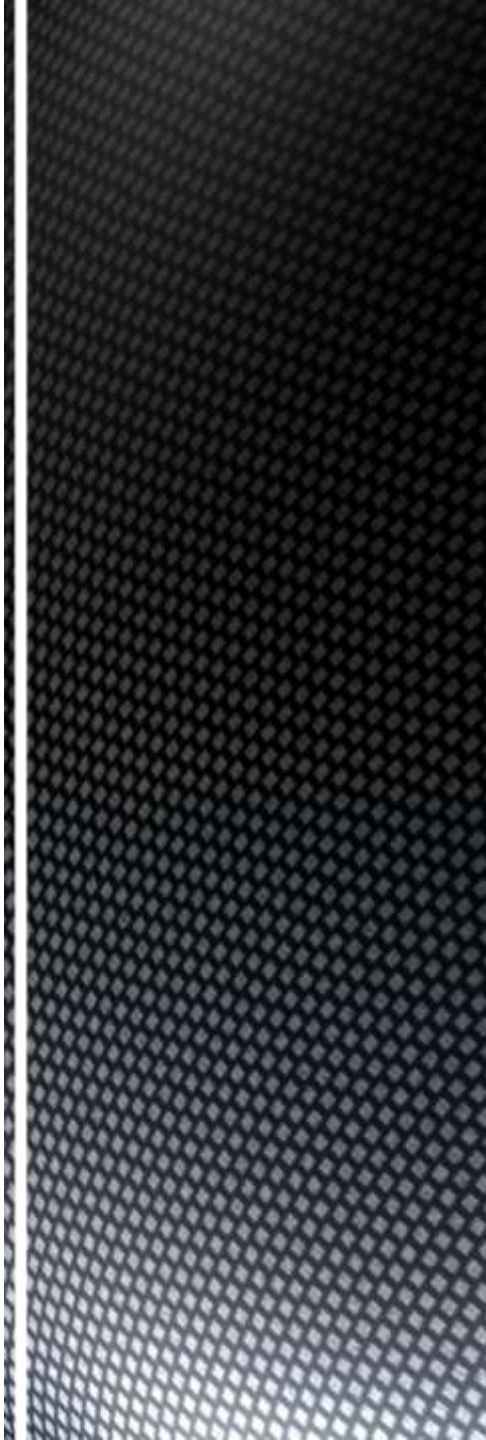
COMP 211 | Assembly Programming

Chapter 3: Assembly Language Fundamentals

Cristina G. Rivera



Chapter Overview

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

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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Integer Constants

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

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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Character and String Constants

- Enclose character in single or double quotes
  - 'A', "x"
  - ASCII character = 1 byte
- Enclose strings in single or double quotes
  - "ABC"
  - 'xyz'
  - Each character occupies a single byte
- Embedded quotes:
  - 'Say "Goodnight," Gracie'

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 $

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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
  - TASM not the same as MASM, for example

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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)

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Labels

- Act as place markers
  - marks the address (offset) of code and data
- Follow identifier rules
- Data label
  - must be unique
  - example: **count** DWORD 100 (not followed by colon)
- Code label
  - target of jump and loop instructions
  - example: **L1:** (followed by colon)

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Comments

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

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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

What's Next

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

Example: Adding and Subtracting Integers

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

; This program adds and subtracts 32-bit integers.

INCLUDE Irvine32.inc
.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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Example Output

Program output, showing 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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Suggested Coding Standards (1 of 2)

- Some approaches to capitalization
  - capitalize nothing
  - capitalize everything
  - capitalize all reserved words, including instruction mnemonics and register names
  - capitalize only directives and operators
- Other suggestions
  - descriptive identifier names
  - spaces surrounding arithmetic operators
  - blank lines between procedures

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Suggested Coding Standards (2 of 2)

- Indentation and spacing
  - code and data labels – no indentation
  - executable instructions – indent 4-5 spaces
  - comments: begin at column 40-45, aligned vertically
  - 1-3 spaces between instruction and its operands
    - ex: mov ax,bx
  - 1-2 blank lines between procedures

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

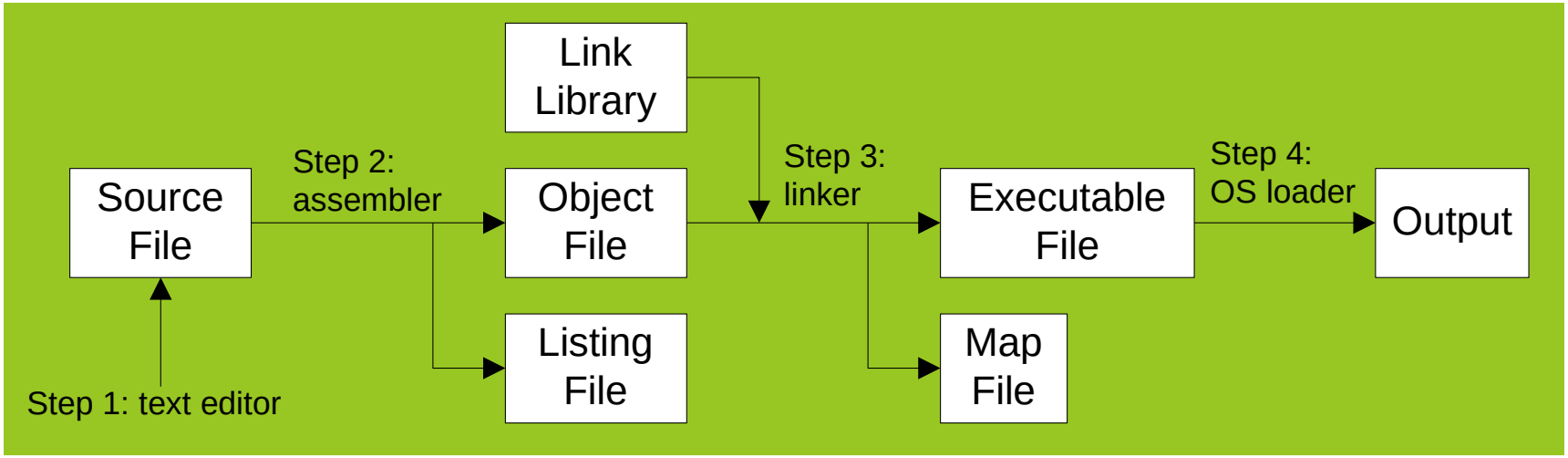
What's Next

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

Assembling, Linking, and Running Programs

- Assemble-Link-Execute Cycle
- make32.bat
- Listing File
- Map File

- 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.



Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Listing File

- Use it to see how your program is compiled
- Contains
  - source code
  - addresses
  - object code (machine language)
  - segment names
  - symbols (variables, procedures, and constants)
- Example: addSub.lst

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Map File

- Information about each program segment:
  - starting address
  - ending address
  - size
  - segment type
- Example: addSub.map (16-bit version)

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

What's Next

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

Defining Data

- Intrinsic Data Types
- Data Definition Statement
- Defining BYTE and SBYTE Data
- Defining WORD and SWORD Data
- Defining DWORD and SDWORD Data
- Defining QWORD Data
- Defining TBYTE Data
- Defining Real Number Data
- Little Endian Order
- Adding Variables to the AddSub Program
- Declaring Uninitialized Data

Intrinsic Data Types (1 of 2)

- 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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Intrinsic Data Types (2 of 2)

- REAL4
  - 4-byte IEEE short real
- REAL8
  - 8-byte IEEE long real
- REAL10
  - 10-byte IEEE extended real

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Defining BYTE and SBYTE Data

Each of the following defines a single byte of storage:

```
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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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 "created by Kip Irvine.",0
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Defining Strings (3 of 3)

- End-of-line character sequence:
  - 0Dh = carriage return
  - 0Ah = line feed

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

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

Idea: Define all strings used by your program in the same area of the data segment.

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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Defining WORD and SWORD Data

- Define storage for 16-bit integers
  - or double characters
  - single value or multiple values

```
word1   WORD   65535          ; largest unsigned value
word2   SWORD  -32768         ; smallest signed value
word3   WORD   ?              ; uninitialized, unsigned
word4   WORD   "AB"           ; double characters
myList  WORD   1,2,3,4,5      ; array of words
array   WORD   5 DUP(?)       ; uninitialized array
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Defining DWORD and SDWORD Data

Storage definitions for signed and unsigned 32-bit integers:

```
val1 DWORD  12345678h          ; unsigned
val2 SDWORD -2147483648        ; signed
val3 DWORD  20 DUP(?)          ; unsigned array
val4 SDWORD -3,-2,-1,0,1       ; signed array
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Defining QWORD, TBYTE, Real Data

Storage definitions for quadwords, tenbyte values, and real numbers:

```
quad1 QWORD  1234567812345678h
val1  TBYTE  10000000000123456789Ah
rVal1 REAL4  -2.1
rVal2 REAL8  3.2E-260
rVal3 REAL10 4.6E+4096
ShortArray REAL4 20 DUP(0.0)
```

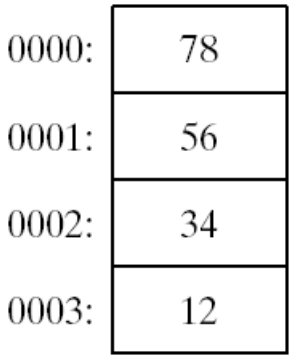
Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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
```



Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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 Unitialized Data

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

```
.data?
```

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

```
smallArray DWORD 10 DUP(?)
```

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

What's Next

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

Symbolic Constants

- Equal-Sign Directive
- Calculating the Sizes of Arrays and Strings
- EQU Directive
- TEXTEQU Directive

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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

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)
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Calculating the Size of a Word Array

Divide total number of bytes by 2 (the size of a word)

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

Calculating the Size of a Doubleword Array

Divide total number of bytes by 4 (the size of a doubleword)

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

EQU Directive

- Define a symbol as either an integer or text expression.
- Cannot be redefined

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

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

TEXTEQU Directive

- Define a symbol as either an integer or text expression.
- Called a text macro
- 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
setupAL TEXTEQU <mov al,count>

.code
setupAL                           ; generates: "mov al,10"
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

What's Next

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

Real-Address Mode Programming (1 of 2)

- 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

Real-Address Mode Programming (2 of 2)

- Requirements
  - INCLUDE Irvine16.inc
  - Initialize DS to the data segment:

```
mov ax,@data
mov ds,ax
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Add and Subtract, 16-Bit Version

```
TITLE Add and Subtract, Version 2      (AddSub2r.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
```

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

Summary

- Integer expression, character constant
- directive – interpreted by the assembler
- instruction – executes at runtime
- code, data, and stack segments
- source, listing, object, map, executable files
- Data definition directives:
  - BYTE, SBYTE, WORD, SWORD, DWORD, SDWORD, QWORD, TBYTE, REAL4, REAL8, and REAL10
  - DUP operator, location counter ($)
- Symbolic constant
  - EQU and TEXTEQU

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.

```
TITLE Add and Subtract  (AddSub2.asm)

INCLUDE Irvine.inc
.data
val1 WORD 10000h
val2 WORD 40000h
val3 WORD 20000h
Val4 SDWORD -3
Val5 SDWORD -5
finalVal WORD ?
.code
main PROC
  mov eax,val1
  add eax,val2
  sub eax,val3
  mov finalVal,eax
  call dumpregs
  mov eax, val4
  Mul eax,val5
  call DumpRegs
  exit
main ENDP
END main
mov eax, val5
mul val4
call DumpRegs
```

Exercise:

Encircle the Error

Irvine, Kip R. Assembly Language for Intel-Based Computers, 2007.