ASSEMBLY LANGUAGE FUNDAMENTALS

MODULE OUTLINE

1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE
2. ADDING AND SUBTRACTING INTEGERS
3. ASSEMBLING, LINKING AND RUNNING PROGRAMS
4. DEFINING DATA
5. SYMBOLIC CONSTANTS
6. REAL-ADDRESS MODE PROGRAMMING

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 01

5.1.1. INTEGER CONSTANTS

- **Radix**
  - **h, Hexadecimal**
  - **q/o, Octal**
  - **d, Decimal**
  - **b, binary**
  - **r, Encoded Real**
  - **t, Decimal (alternate)**
  - **y, Binary (alternate)**
- **Put a leading zero for an hexadecimal constant beginning with letter.**

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 02

5.1.2. INTEGER EXPRESSIONS

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

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 03

5.1.3. REAL NUMBER CONSTANTS

- **[sign]integer.[integer][exponent]**
- **Examples:**
  - **2.**
  - **+3.0**
  - **-44.2E+05**
  - **26.E5**
- **Encoded Reals:**
  - **0011 1111 1000 0000 0000 0000 0000 0000b**
  - **3F800000r**

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 04

5.1.4. CHARACTER CONSTANTS

- **MASM stores characters in ASCII code**
- **Examples:**
  - **'A'**
  - **"d“**

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 05

5.1.5. STRING CONSTANTS

- **Examples:**
  - **'ABC'**
  - **'X'**
  - **"Good night, Gracie"**
  - **'4096‘**
- **Embedded quotes are permitted**
  - **"This isn't a test"**
  - **'Say "Good night," Uvulwa'**

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 06

5.1.6. RESERVED WORDS

- **Instructions: MOV, ADD, MUL**
- **Registers Names: EAX, ECX, CS, SS**
- **Directives: ORG (origin), DWORD (double word)**
- **Attributes: BYTE, WORD**
- **Operators:**
- **Predefined symbols: @**

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 07

5.1.7. IDENTIFIERS

- **May contain 1 to 247 characters**
- **Not case sensitive**
- **First character (A…Z, a…z) or _, @, ? or $**
- **Do not use Reserved words**

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 08

5.1.8. DIRECTIVES

- Commands instructing the assembler
- Do not execute at runtime
- Can define variables, macros and procedures
- Assign names to memory
- Not case sensitive;
  - e.g.: .DATA, .data and .Data are equivalent

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 09

5.1.9. INSTRUCTIONS

- **Have four basic parts:**
  - **Label (optional)**
  - **Instruction mnemonic (required)**
  - **Operand(s) (usually required)**
  - **Comment (optional)**
- **Basic syntax:**
  - ***[label:] mnemonic[operands] [;comment]***

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 10

5.1.9. INSTRUCTIONS - contd

- **Label is a place marker**
  - ***Data Labels***
    - **e.g.:**
```
count DWORD 100 ; create a variable 'count'
      array DWORD 1024, 2048
            DWORD 4096, 8192    ; multiple data items
```
  - ***Code Labels***
    - **e.g1.:**
```
target:
        mov     ax, bx
        …
        jmp     target
```
    - **e.g2:**
```
label1: mov     ax, bx  ; same line with
                        ; instruction
```

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 11

5.1.9. INSTRUCTIONS - contd

- **Instruction Mnemonic**
  - `mov  ;Move (assign) one value to another`
  - `add  ;Add two values`
  - `sub  ;Subtract one value from another`
  - `mul  ;Multiply two values`
  - `jmp  ;Jump to a new location`
  - `call ;Call a procedure`

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 12

5.1.9. INSTRUCTIONS - contd

- **Operands**
  - `stc ;Set carry flag – no operand`
  - `inc eax ;Add 1 to EAX`
  - `mov count, ebx ;Move EBX to count`
  - `imul eax, ebx, 5 ;Store (5*EBX) in EAX`

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 13

5.1.9. INSTRUCTIONS - contd

- **Comments**
  - ***Program's purpose***
  - ***Author or reviser***
  - ***Program date stamp***
  - ***Technical notes***
- ***Multi-line comments***
  - **e.g.:**

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

5.1. BASIC ELEMENTS OF ASSEMBLY LANGUAGE - 14

5.1.10.THE NOP (No Operation) INSTRUCTION

- **Takes up one byte of storage**
- **Sometimes used by assemblers/compilers to align code to even-address**
  - **e.g.:**

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

5.2. ADDING AND SUBTRACTING INTEGERS - 01

5.2.1. EXAMPLE OF ADD & SUBTRACT

```
; 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
```

5.2. ADDING AND SUBTRACTING INTEGERS - 02

5.2.1. PROGRAM TEMPLATE

```
TITLE Program Template (Template.asm)
; Program Description:
; Author:
; Creation Date:
; Revisions:
; Date:
INCLUDE Irvine32.inc
.data
    ; (insert variables here)
.code
main PROC
    ; (insert executable instructions here)
exit
main ENDP
    ; (insert additional procedures here)
END main
```

5.3. ASSEMBLING, LINKING AND RUNNING PROGRAMS

5.3.1. THE ASSEMBLE-LINK-EXECUTE CYCLE

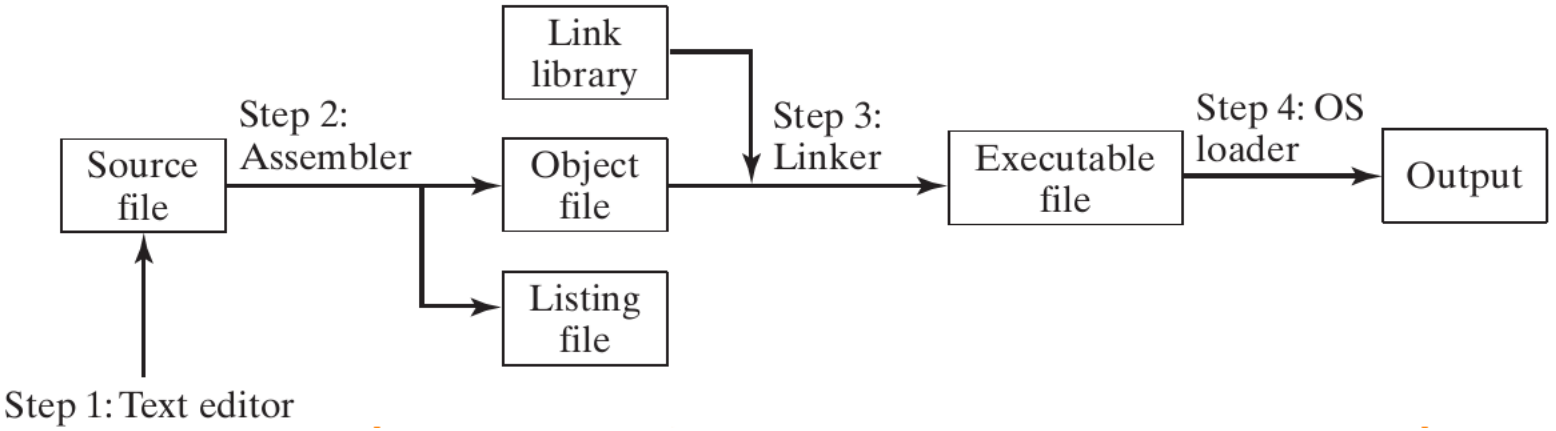

Figure 5.3.1.1: Assemble-Link-Execute Cycle

5.4. DEFINING DATA - 01

5.4.1. INTRINSIC DATA TYPES

Type	Usage
BYTE	8-bit unsigned integer. B stands for byte
SBYTE	8-bit signed integer. S stands for signed
WORD	16-bit unsigned integer (can also be a Near pointer in real-address mode)
SWORD	16-bit signed integer
DWORD	32-bit unsigned integer (can also be a Near pointer in protected mode). D stands for double
SDWORD	32-bit signed integer. SD stands for signed double
FWORD	48-bit integer (Far pointer in protected mode)
QWORD	64-bit integer. Q stands for quad
TBYTE	80-bit (10-byte) integer. T stands for Ten-byte
REAL4	32-bit (4-byte) IEEE short real
REAL8	64-bit (8-byte) IEEE long real
REAL10	80-bit (10-byte) IEEE extended real

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 03

5.4.3. DEFINING BYTE AND SBYTE DATA

```
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
```

5.4. DEFINING DATA - 04

5.4.3. DEFINING BYTE AND SBYTE DATA – contd

- ***DB Directive***

```
val1 DB 255      ; unsigned byte
val2 DB -128     ; signed byte
```

- ***Multiple Initialisers***

```
list BYTE 10, 20, 30, 40
```

5.4. DEFINING DATA - 05

5.4.3. DEFINING BYTE AND SBYTE DATA – contd

- ***Multiple Initialisers***

```
list BYTE 10, 20, 30, 40
```

Offset	Value
0000:	10
0001:	20
0002:	30
0003:	40

Figure 5.4.3.1: Memory Layout of a Byte Sequence

5.4. DEFINING DATA - 06

5.4.3. DEFINING BYTE AND SBYTE DATA – contd

- ***Multiple Initialisers (contd)***
  - ***array***

```
list BYTE 10,20,30,40
     BYTE 50,60,70,80
     BYTE 81,82,83,84
```

  - ***Using different radixes***

```
list1 BYTE 10, 32, 41h, 00100010b
list2 BYTE 0Ah, 20h, 'A', 22h
```

5.4. DEFINING DATA - 07

5.4.3. DEFINING BYTE AND SBYTE DATA – contd

- ***Defining Strings***

```
greeting1 BYTE "Good afternoon",0
greeting2 BYTE 'Good night',0
greeting1 BYTE "Welcome to the Encryption Demo program"
          BYTE "created by Kip Irvine.",0dh,0ah
          BYTE "If you wish to modify this program, please"
          BYTE "send me a copy.",0dh,0ah,0
```

5.4. DEFINING DATA - 08

5.4.3. DEFINING BYTE AND SBYTE DATA – contd

- ***DUP Operator* - allocates storage for multiple data items**

```
BYTE 20 DUP(0) ; 20 bytes, all equal to zero
BYTE 20 DUP(?) ; 20 bytes, uninitialized
BYTE 4 DUP("STACK") ; 20 bytes: "STACKSTACKSTACKSTACK"
```

5.4. DEFINING DATA - 09

5.4.2. DEFINING WORD AND SWORD DATA

- **WORD, SWORD, directives to create one or more 16-bit integers**
- e.g.:

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

=> Legacy DW directive

```
val1 DW 65535      ;unsigned
val2 DW -32768     ;signed
```

5.4. DEFINING DATA - 10

5.4.2. DEFINING WORD AND SWORD DATA - contd

- **ARRAYS OF WORDS**

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

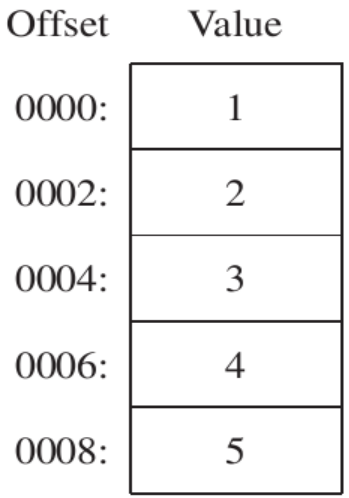

Figure 5.4.2.1: Memory Layout, 16-bit Word array

- **DUP Operator is more convenient**

```
array WORD 5 DUP(?) ; 5 values, uninitialized
```

5.4. DEFINING DATA - 11

5.4.2. DEFINING DWORD AND SDWORD DATA

- **DWORD, SDWORD, directives to create one or more 32-bit integers**
- **e.g.:**

```
val1 DWORD 12345678h            ;unsigned
val2 SDWORD –2147483648         ;signed
val3 DWORD 20 DUP(?)            ;unsigned array
pVal DWORD val3                 ;offset of val3
```

=> Legacy DD directive

```
val1 DD 12345678h       ;unsigned
val2 DD –2147483648     ;signed
```

5.4. DEFINING DATA - 12

5.4.2. DEFINING DWORD AND SDWORD DATA - contd

- **ARRAYS OF DOUBLEWORDS**

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

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

Figure 5.4.2.2: Memory Layout, 32-bit Doubleword Array

5.4. DEFINING DATA - 13

5.4.2. DEFINING PACKET BINARY CODED DECIMAL (BCD, TBYTE) DATA

- **Intel stores BCD in 10-byte package**

Decimal Value	Storage Bytes
+1234	34 12 00 00 00 00 00 00 00 00
−1234	34 12 00 00 00 00 00 00 00 80

5.4. DEFINING DATA - 14

5.4.2. DEFINING PACKET BINARY CODED DECIMAL (BCD, TBYTE) DATA - contd

- **MASM uses TBYTE directive for BCD**

```
intVal TBYTE 8000000000000001234h ; valid
intVal TBYTE -1234                ; invalid
```

- **To correct the second statement**

```
.data
posVal REAL8 1.5
bcdVal TBYTE ?
.code
fld posVal      ;load onto floating-point stack
fbstp bcdVal    ;rounds up to 2 as packed BCD
```

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.4. DEFINING DATA - 02

5.4.2. DATA DEFINITION STATEMENT

- `[name] directive initializer [,initializer]...`
- `e.g.:`

```
count DWORD 12345
```

Table 5.4.1.1: Intrinsic Data Types

5.5. SYMBOLIC CONSTANTS

2.1.1. INTRODUCTION

- **X86 family of processors include:**
  - **All Intel IA-32 processors: Pentium, Core-Duo;**
  - **AMD (Advanced Micro Devices): Athlon, Phenom, and Opteron**
- **Knowledge of hardware – good for Assembly programming**
- **Not exhaustive treatment of x86 processors**

5.6. REAL-ADDRESS MODE PROGRAMMING

2.1.1. INTRODUCTION

- **X86 family of processors include:**
  - **All Intel IA-32 processors: Pentium, Core-Duo;**
  - **AMD (Advanced Micro Devices): Athlon, Phenom, and Opteron**
- **Knowledge of hardware – good for Assembly programming**
- **Not exhaustive treatment of x86 processors**