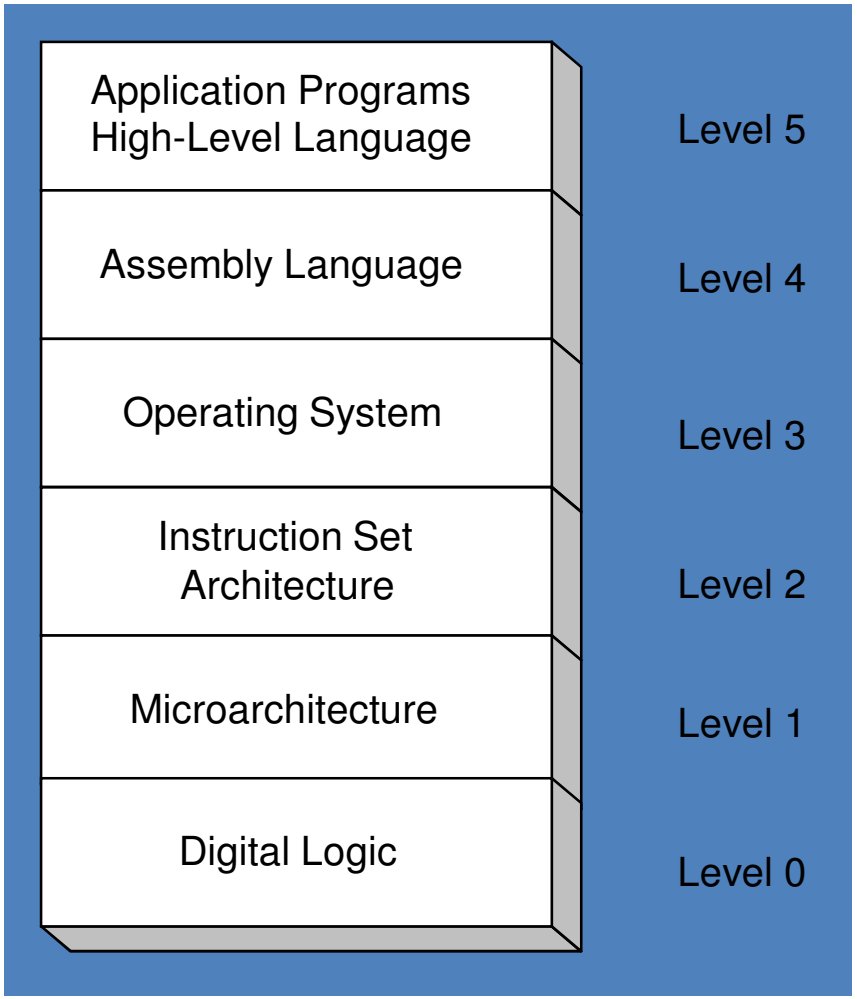
Writing Program For Use With An Assembler

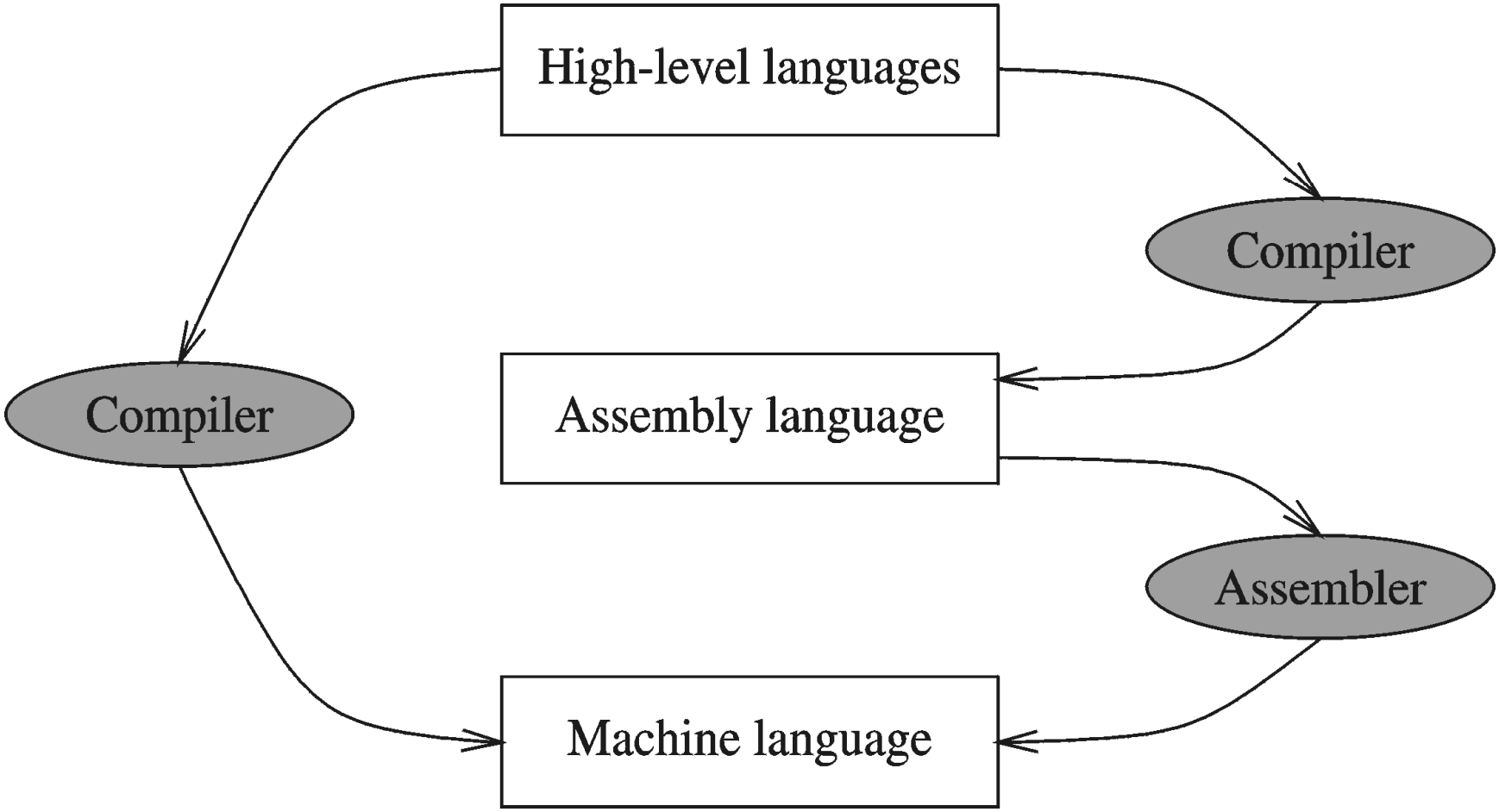
Programmer's View of a Computer System

Increased level of abstraction

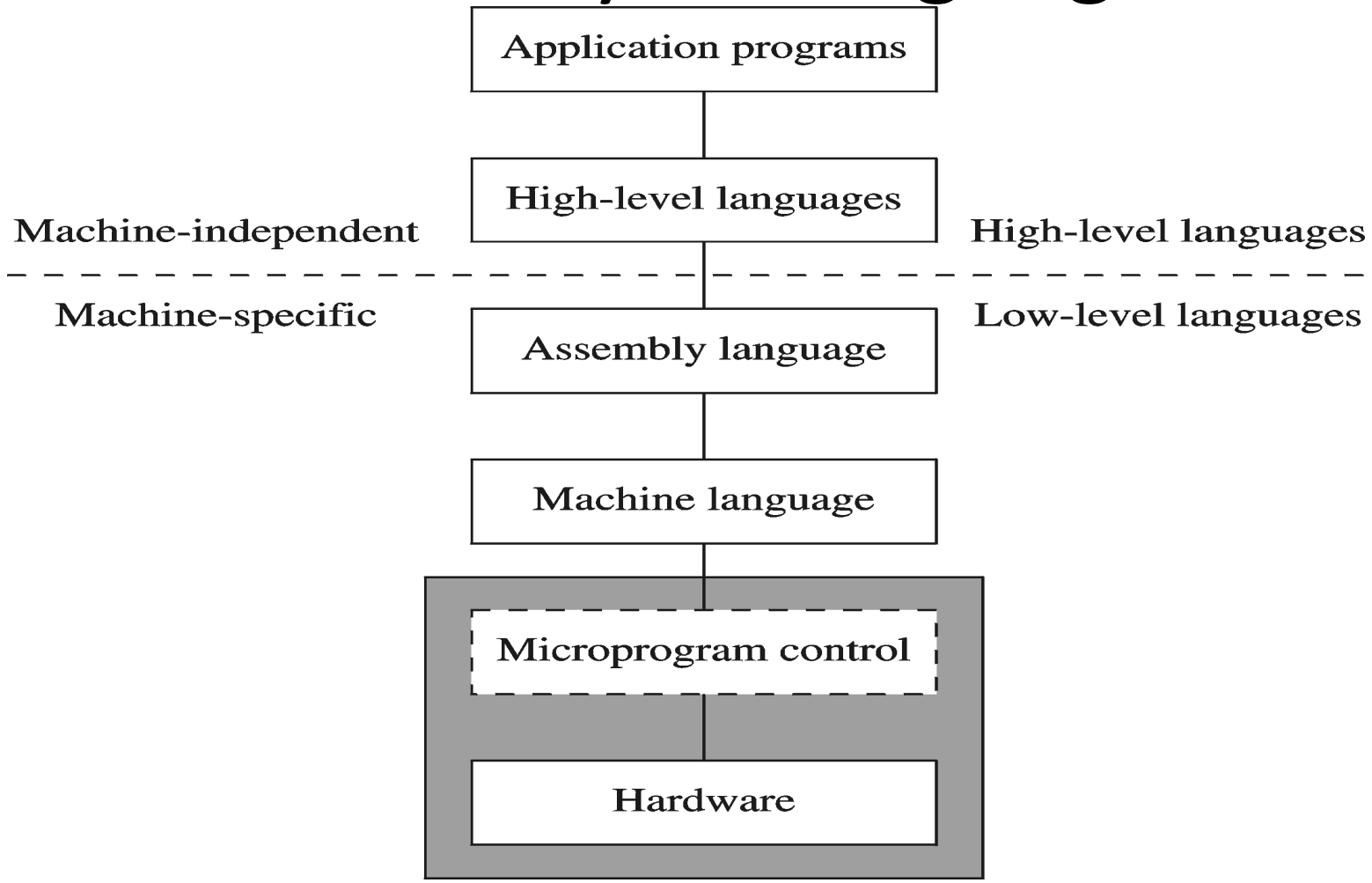


Each level hides the details of the level below it

Compiler and Assembler



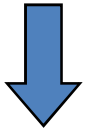
A Hierarchy of Languages



Translating Languages

English: D is assigned the sum of A times B plus 10.

High-Level Language: D = A * B + 10



Assembly Language:

```
Mov   ax, A
mul   B
Add   ax, 10
mov   D, ax
```

Machine Language:?

A1 00404000

F7 25 00404004

83 C0 0A

A3 00404008

Why Learn Assembly Language?

Two main reasons:

- Accessibility to system hardware
  - Assembly Language is useful for implementing system software
  - Also useful for small embedded system applications
- Space and Time efficiency
  - Understanding sources of program inefficiency
  - Tuning program performance
  - Writing compact code

Assembly vs High-Level Languages

Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

Assembler

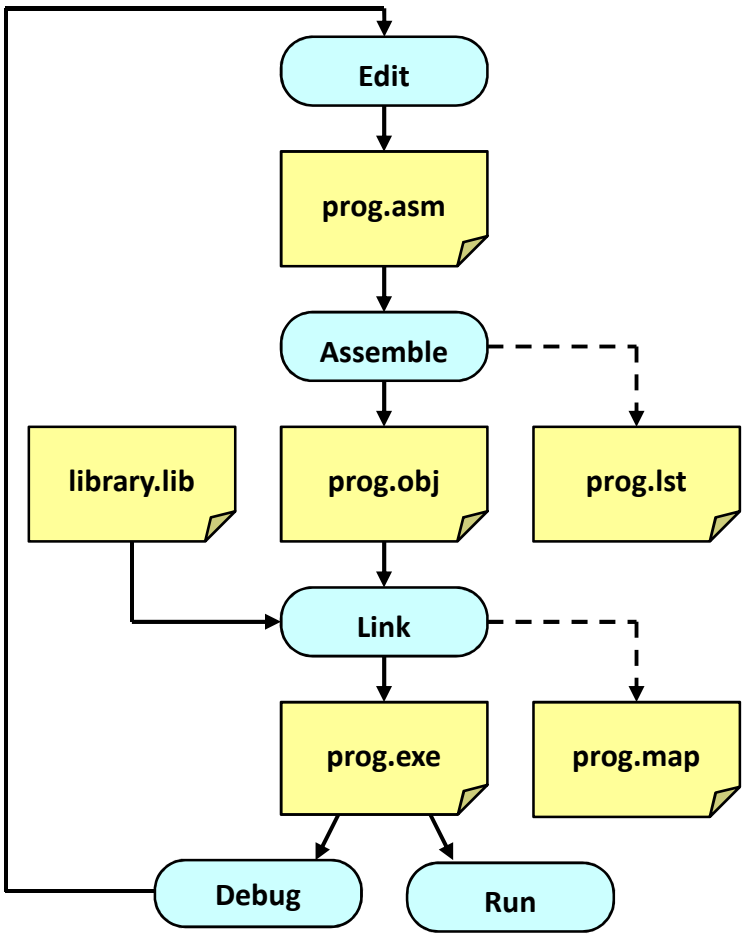
- Software tools are needed for editing, assembling, linking, and debugging assembly language programs
- An assembler is a program that converts source-code programs written in assembly language into object files in machine language
- MASM (Macro Assembler from Microsoft)

Linker and Link Libraries

- You need a linker program to produce executable files
- It combines your program's object file created by the assembler with other object files and link libraries, and produces a single executable program

Assemble-Link-Debug Cycle

- Editor
  - Write new (**.asm**) programs
  - Make changes to existing ones
- Assembler: **ML.exe** program
  - Translate (**.asm**) file into object (**.obj**) file in machine language
  - Can produce a listing (**.lst**) file that shows the work of assembler
- Linker: **LINK32.exe** program
  - Combine object (**.obj**) files with link library (**.lib**) files
  - Produce executable (**.exe**) file
  - Can produce optional (**.map**) file



Debugger

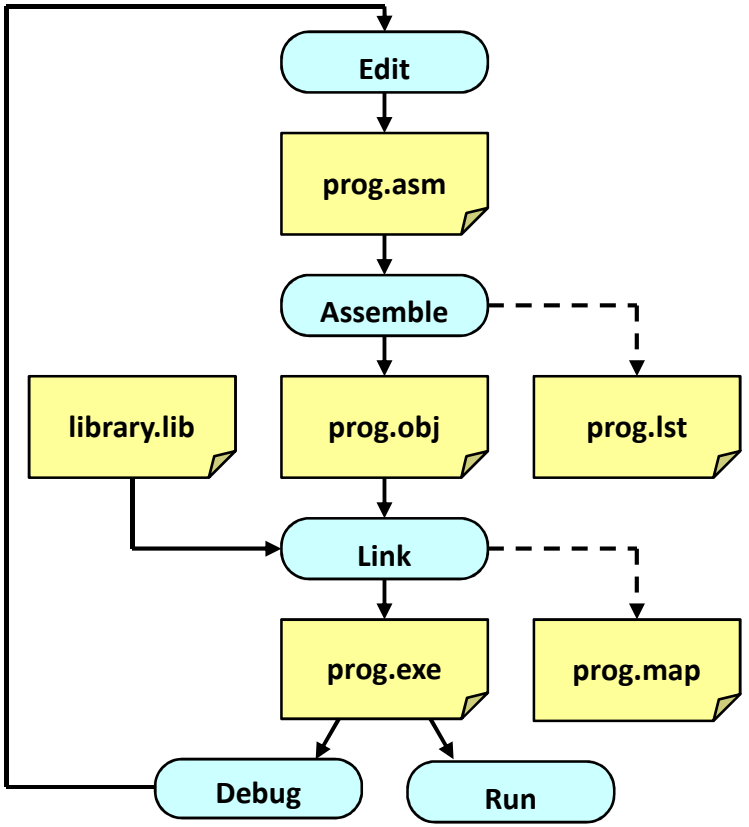
- Allows you to trace the execution of a program
- Allows you to view code, memory, registers, etc.

Editor

- Allows you to create assembly language source files

Assemble-Link-Debug Cycle – cont'd

- Debugger
  - Trace program execution
    - Either step-by-step, or
    - Use breakpoints
  - View
    - Source (**.asm**) code
    - Registers
    - Memory by name & by address
    - Modify register & memory content
  - Discover errors and go back to the editor to fix the program bugs



Assembly Language Program

- Additional instructions are required
- The purpose of these additional program instructions is:
  - To initialize various parts of the system
    - segment registers
    - flags
    - programmable port devices
  - Some of the instructions are to handle the stack of the 8086 based system
  - to handle the programmable peripheral devices such as ports, timers and controllers
    - should be assigned suitable control words

Assembly Language Program

- The best way to approach the initialization task is to make a checklist of all the registers, programmable devices and flags in the system we are working on

Assembly Language Program

- An 8086 assembly language program has five columns namely
  - Address
  - Data or code
  - Labels
  - Mmnemonics
  - Operands
  - Comments

Constants

- Integer Constants
  - Examples: –10, 42d, 10001101b, 0FF3Ah, 777o
  - Radix: b = binary, d = decimal, h = hexadecimal, and o = octal
  - If no radix is given, the integer constant is decimal
  - A hexadecimal beginning with a letter must have a leading 0
- Character and String Constants
  - Enclose character or string in single or double quotes
  - Examples: 'A', "d", 'ABC', "ABC", '4096'
  - Embedded quotes: "single quote ' inside", 'double quote " inside'
  - Each ASCII character occupies a single byte

Assembly Language Statements

Three types of statements in assembly language

1. Executable Instructions
   - Generate machine code for the processor to execute at runtime
   - Instructions tell the processor what to do
2. Assembler Directives
   - Provide information to the assembler while translating a program
   - Used to define data, select memory model, etc.
   - Non-executable: directives are not part of instruction set
3. Macros
   - Shorthand notation for a group of statements
   - Sequence of instructions, directives, or other macros

Instructions

- Assembly language instructions have the format:
  `[label:]  mnemonic  [operands]  [;comment]`
- Instruction Label (optional)
  - Marks the address of an instruction, must have a colon **:**
  - Used to transfer program execution to a labeled instruction
- Mnemonic
  - Identifies the operation (e.g. MOV, ADD, SUB, JMP, CALL)
- Operands
  - Specify the data required by the operation
  - Executable instructions can have zero to three operands
  - Operands can be registers, memory variables, or constants

Identifiers

- Identifier is a programmer chosen name
- Identifies variable, constant, procedure, code label
- May contain between 1 and 247 characters
- Not case sensitive
- First character must be a letter (A..Z, a..z), underscore(_), @, ?, or $.
- Subsequent characters may also be digits.
- Cannot be same as assembler reserved word.

Comments

- Comments are very important!
  - Explain the program's purpose
  - When it was written, revised, and by whom
  - Explain data used in the program
  - Explain instruction sequences and algorithms used
  - Application-specific explanations
- Single-line comments
  - Begin with a semicolon **;** and terminate at end of line
- Multi-line comments
  - Begin with **COMMENT** directive and a chosen character
  - End with the same chosen character

Suggested Coding Standards

- Some approaches to capitalization
  - Capitalize nothing
  - Capitalize everything
  - Capitalize all reserved words, mnemonics and register names
  - Capitalize only directives and operators
  - MASM is NOT case sensitive: does not matter what case is used
- Other suggestions
  - Use meaningful identifier names
  - Use blank lines between procedures
  - Use indentation and spacing to align instructions and comments
    - Use tabs to indent instructions, but do not indent labels
    - Align the comments that appear after the instructions

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
```

- 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.

Defining Strings

- A string is implemented as an array of characters
  - For convenience, it is usually enclosed in quotation marks
  - It is often terminated with a NULL char (byte value = 0)
- 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 "
          BYTE "Demo Program", 0
```

Defining Strings – cont'd

- 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
```

- End-of-line character sequence:
  - 0Dh = 13 = carriage return
  - 0Ah = 10 = line feed

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

TYPE Operator

- TYPE operator
  - Size, in bytes, of a single element of a data declaration

```
.DATA
var1 BYTE ?
var2 WORD ?
var3 DWORD ?
var4 QWORD ?

.CODE
mov eax, TYPE var1   ; eax = 1
mov eax, TYPE var2   ; eax = 2
mov eax, TYPE var3   ; eax = 4
mov eax, TYPE var4   ; eax = 8
```

Multiple Line Declarations

A data declaration spans multiple lines if each line (except the last) ends with a comma

The LENGTHOF and SIZEOF operators include all lines belonging to the declaration

```
.DATA
array WORD 10,20,
           30,40,
           50,60

.CODE
mov ax, LENGTHOF array ; 6
Mov bx, SIZEOF array   ; 12
```

In the following example, array identifies the first line WORD declaration only

Compare the values returned by LENGTHOF and SIZEOF here to those on the left

```
.DATA
array  WORD 10,20
       WORD 30,40
       WORD 50,60

.CODE
mov eax, LENGTHOF array ; 2
mov ebx, SIZEOF array   ; 4
```

LABEL Directive

- Assigns an alternate name and type to a memory location
- LABEL does not allocate any storage of its own
- Removes the need for the PTR operator
- Format: *Name* LABEL *Type*

```
.DATA
dval   LABEL DWORD
wval   LABEL WORD
blist  BYTE 00h,10h,00h,20h
.CODE
mov ax, dval        ; ax = 10002000h
mov cx, wval        ; cx  = 1000h
mov dl, blist       ; dl  = 00h
```