UTM
UNIVERSITI TEKNOLOGI MALAYSIA

Module 3:
Introduction to
Assembly Language Programming

Assembly code …

```
TITLE MASM Template (main.asm)

; Description:
;
; Revision date:

INCLUDE Irvine32.inc

.data
myMessage BYTE "MASM program example",0dh,0ah,0

.code
main PROC
    call Clrscr

    mov  edx,OFFSET myMessage
    call WriteString

    exit
main ENDP

END main
```

Instructions

- Assembled into **machine code** by assembler
- Executed at runtime by the CPU after the program has been loaded into memory and started.
- Member of the Intel IA-32 instruction set
- Parts :
  - **A) Label (optional)**
  - **B) Mnemonic (required)**
  - **C) Operand (required)**
  - **D) Comment (optional)**

Example:
Standard Format for Instruction

	Label	Mnemonics	Operands	; Comments
Instruction 1		mov	eax, 1000h	; eax=1000h
Instruction 2	target:	add	ebx, 4000h	;ebx=ebx+4000h

A) Labels

- Act as **place marker**
  - marks the address (offset) of code and data
- Follow **identifier** rules *(refer slide # 18)*
- **Data** label
  - must be unique
  - example: **myVar WORD 100**
    mov ax, myVar
- **Code** label
  - target of jump and loop instructions
  - example: target:

```
target :
mov  ax,bx
…
jmp  target
```

B) Mnemonics and Operands

- Instruction Mnemonics is a short word that identifies the operation carried out by an instruction (useful name).
- Examples :
  - **MOV** move (assign one value to another)
  - **ADD** add two values
  - **SUB** subtract one value from another
  - **MUL** multiply two values
  - **INC** increment
  - **DEC** decrement
  - **JMP** jump to a new location
  - **CALL** call a procedure

C) Operands

- An assembly language can have between zero to three operands
- Types of operand :
  - **constant** (immediate value) eg **96, 2005h, 101011010b**
  - **constant expression** eg **2+4**
  - **register** eg **EAX, EBX, AX, AH**
  - **memory** (data label) eg **count**

.. C) Operands

- Examples assembly language instructions with various number of operands :
- No operand

```
stc                     ; set carry flag
```

- one operand

```
inc     ax              ; add 1 to AX
```

- two operands

```
mov    count, bx        ; move BX to count
```

D) 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 (;)
  - Ex : ; add BX to AX
- Multi-line comments
  - begin with COMMENT directive and a programmer-chosen character
  - end with the same programmer-chosen character

Example: Multi-line Comments

```
COMMENT   !
      This is a comment
      This line is also a comment
      Everything here are comments but after
      exclamation character it is no longer a comment
!
```

Example: Program Template

```
TITLE Budget Calculator     (budget.asm)
; Program Description:
; Author:
; Date Created:
; Last Modification Date:

INCLUDE Irvine32.inc
.data
(insert variables here)
val1    dword  10000h

:
```

```
:
.code
main PROC
(insert executable instructions here)
        mov ax,@data      ; initialize DS

        exit                    ; exit to o/s
main ENDP
(insert additional procedures here)

END main
```

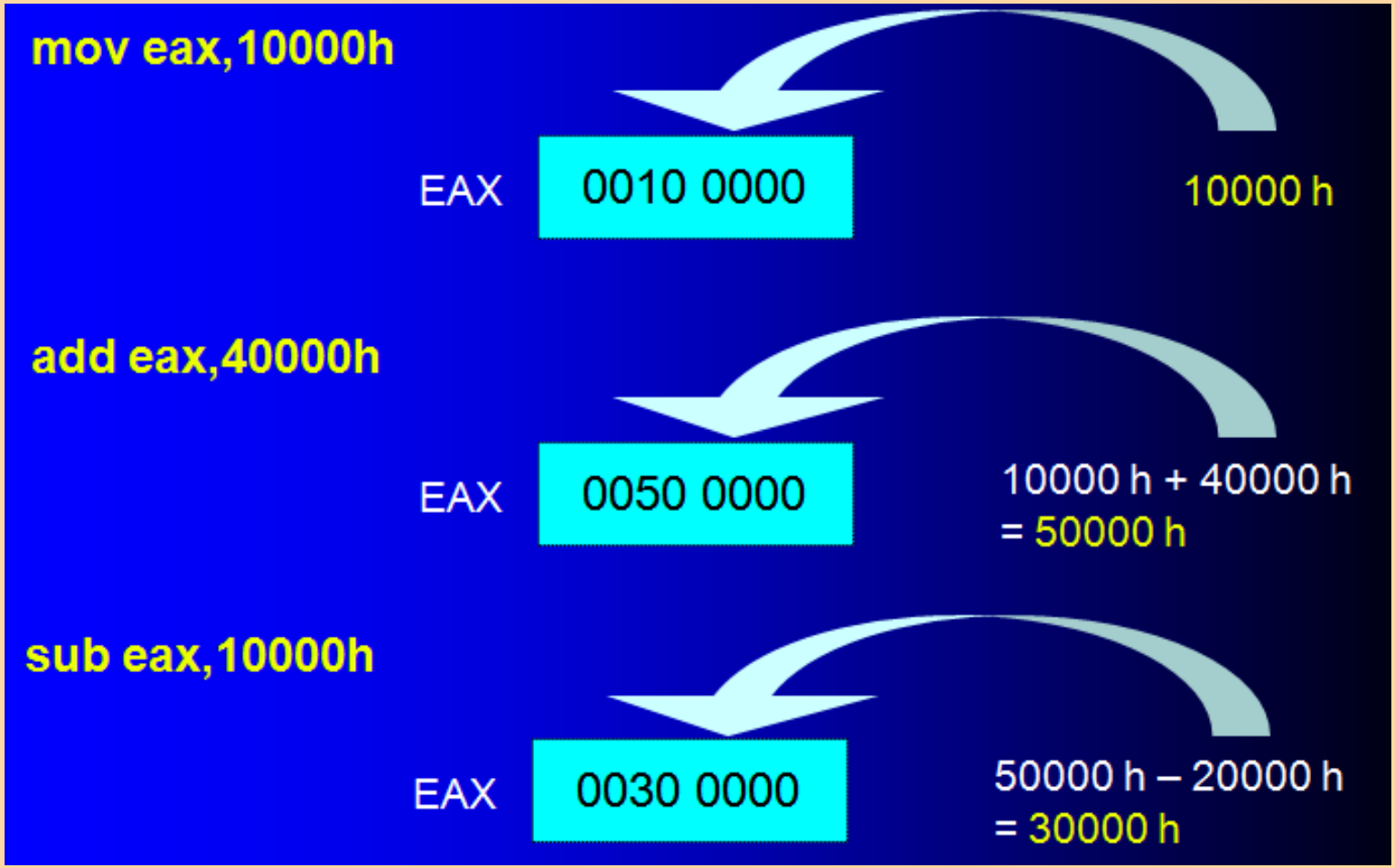
Example: Adding and Subtracting Integers

```
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
```

Copies setup information from text file named Irvine32.inc

Call procedure that displays the current values of the CPU registers. Verify program correctly

… Adding and Subtracting Integers



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
```

Integer Constants

- Optional leading '+' or '–' sign
- Binary, decimal, hexadecimal, or octal digits
- Common radix characters:
  - h – hexadecimal
  - d – decimal
  - b – binary
  - r – encoded real
  - q/o - octal
- Examples: 30d, 06Ah, 42, 1101b, 33q, 22o
- Hexadecimal number that beginning with letter must have a leading zero to prevent the assembler interpreting it as an identifier : 0A5h

If no radix - decimal

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

- **Reserved words** cannot be used as identifiers
  - Instruction mnemonics (MOVE, ADD, MUL)
  - Directives (INCLUDE)
  - type attributes (BYTE, WORD)
  - Operators (+, -)
  - predefined symbols (@data)
- **Directives**
  - command understood by the assembler
  - not part of Intel instruction set
  - case insensitive
  - E.g. WORD, DWORD, .data, .DATA

Identifiers

- **Identifiers** (programmer-chosen name)
  - May contain 1-247 characters, including digits
  - Not case-sensitive (by default)
  - first character must be a letter (A..Z,a..z), _, @, ? or $
  - Subsequent character may also be a digit
  - Identifier cannot be the same as assembler reserved word.
  - common sense – create identifier names that easy to understand

- Avoid using single @ sign as first character.

- Ex : Valid identifier

  Var1, Count, MAX, $first, open_file, _12345, @@myfile,_main

Directives

- Commands that are recognized / understood 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 (**.data, .DATA, .Data** – they are the same)
- Different assemblers have different directives
  - NASM != MASM, for example
- Examples :
  - **.DATA** – identify area of a program that contains variables
  - **.CODE** – identify area of a program that contains instructions
  - **PROC** – identify beginning of a procedure. *name PROC*

Intrinsic Data Types

- **BYTE, SBYTE** (signed-byte)
  - 8-bit unsigned integer; 8-bit signed integer
- **WORD, SWORD**
  - 16-bit unsigned & signed integer
- **DWORD, SDWORD**
  - 32-bit unsigned & signed integer
- **QWORD** (quadword)
  - 64-bit integer
- **TBYTE** (tenbyte)
  - 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*] . . .

  Nilai1 BYTE 34

  Nilai2 BYTE 'A'

  Nilai3 BYTE 27h, 4Ch, 6Ah

- All initializers become binary data in memory

  example : **00110010b, 32h, 50d**

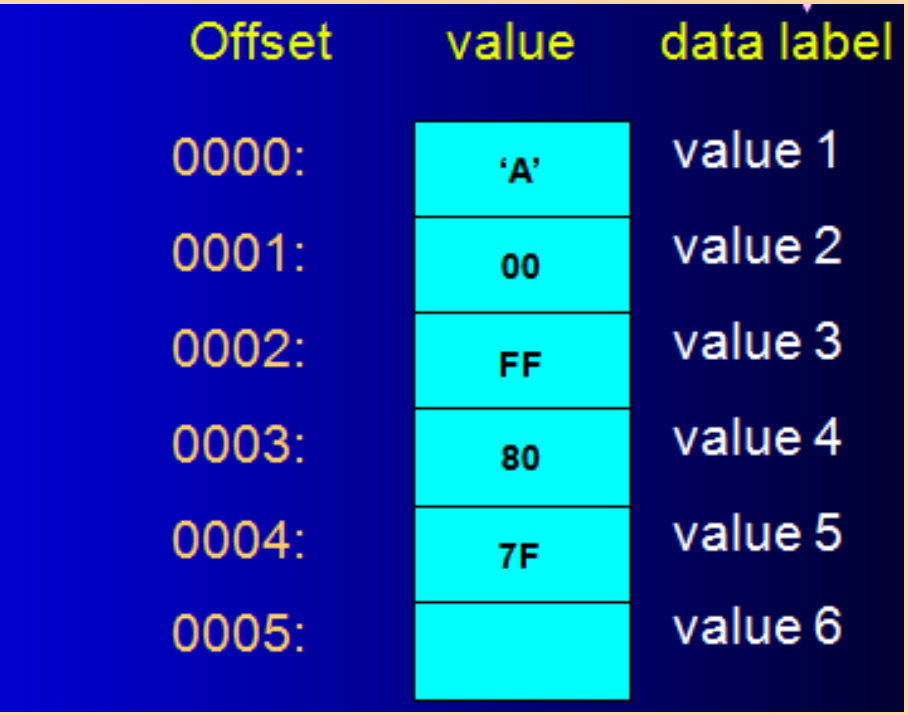
Defining BYTE and SBYTE Data (8 bits)

- 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
```

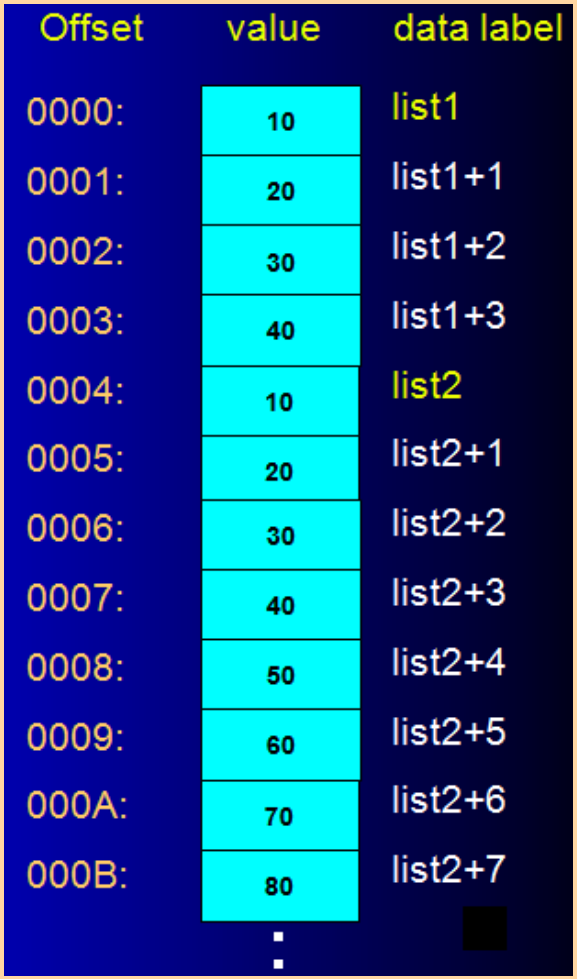
Example: Sequences of Bytes

```
value1 BYTE 'A'
value2 BYTE 0
value3 BYTE 255
value4 SBYTE -128
value5 SBYTE +127
value6 BYTE ?
```



Example: Defining Bytes

```
list1   BYTE 10h,20h,30h,40h
list2   BYTE 10h,20h,30h,40h
        BYTE 50h,60h,70h,80h
        BYTE 81h,82h,83h,84h
list3   BYTE ?,32,41h,00100010b
list4   BYTE 0Ah,20h,'A',22h
```



Defining Strings

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

```
str1 BYTE "Enter your name",0
str2 BYTE 'Error: halting program',0
str3 BYTE 'A','E','I','O','U'
greeting1 BYTE "Welcome to the Encryption Demo program "
          BYTE "created by Kip Irvine.",0
greeting2 \
    BYTE "Welcome to the Encryption Demo program "
    BYTE "created by Kip Irvine.",0
```

… Defining Strings

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

```
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 (duplicate) Operator

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

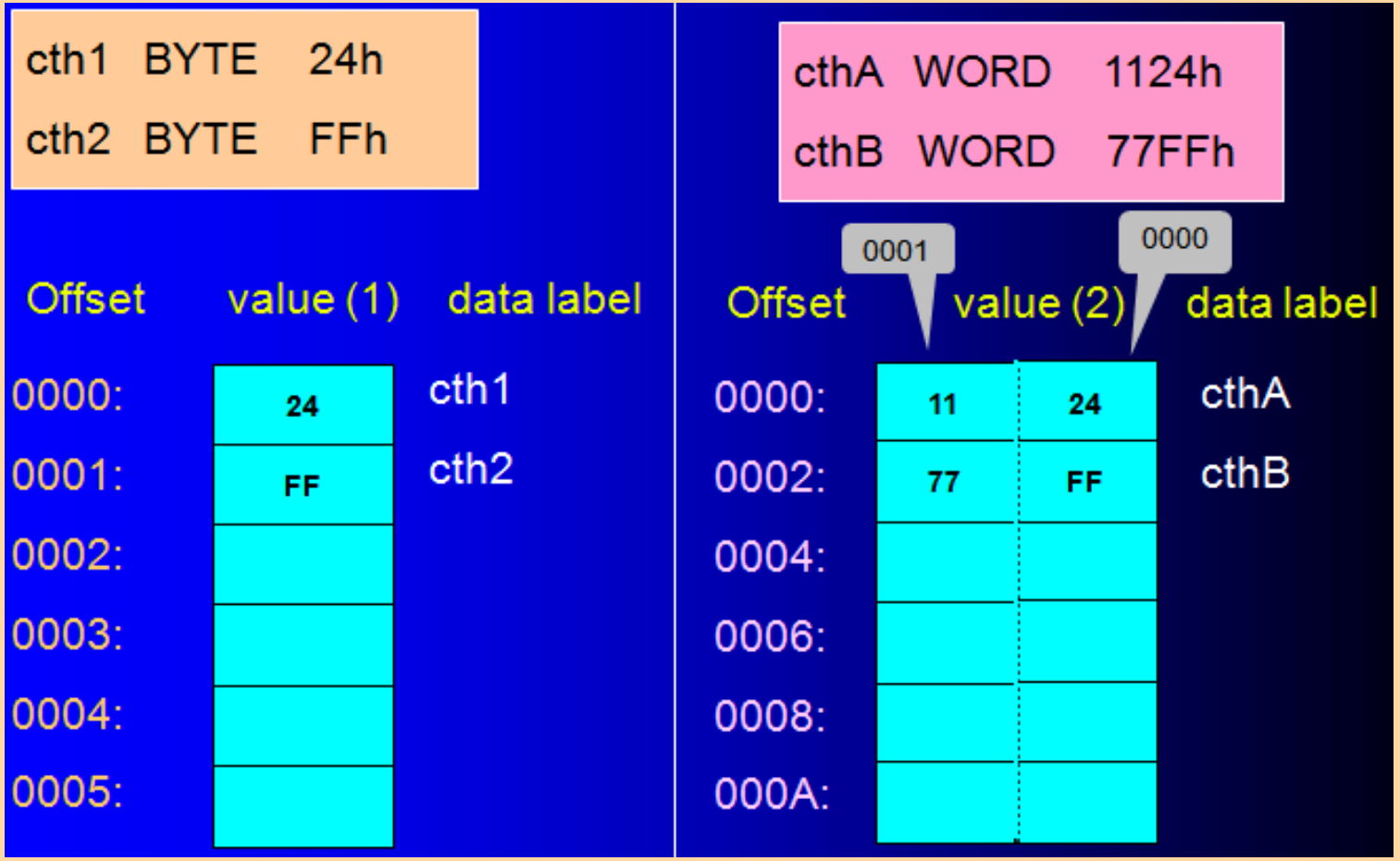
```
var1 BYTE 20 DUP(0)          ; 20 bytes, all equal to zero
var2 BYTE 20 DUP(?)          ; 20 bytes, uninitialized
var3 BYTE 4 DUP("STACK")     ; 20 bytes: "STACKSTACKSTACKSTACK"
var4 BYTE 10,3 DUP(0),20     ; 0A,00,00,00,14
```

Defining WORD and SWORD Data (2 bytes)

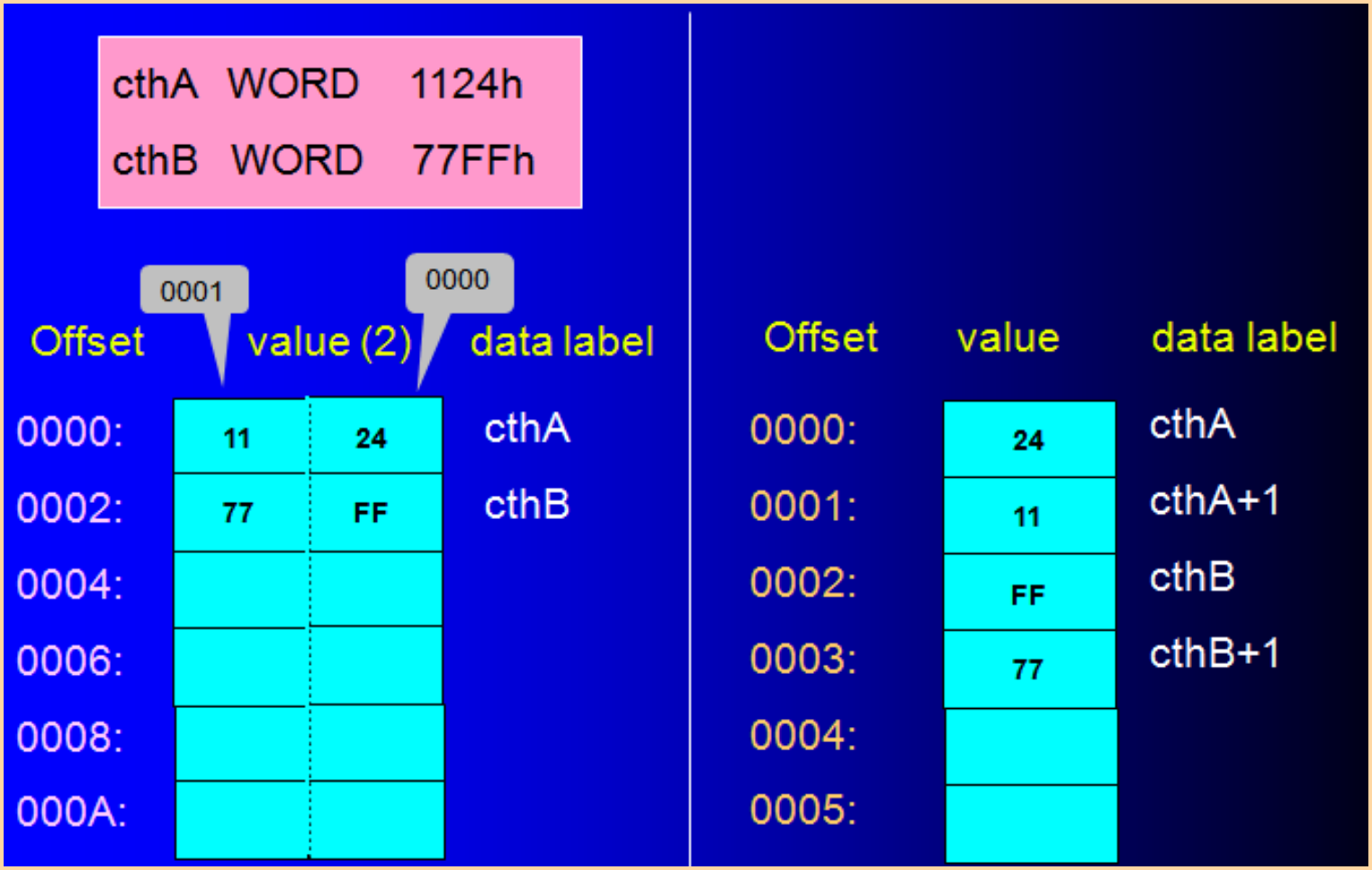
- Define storage for 16-bit integers
  - or double characters
  - single value or multiple values

```
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
```

BYTE vs WORD



WORD


cthA WORD 1124h
cthB WORD 77FFh
0001
0000
Offset
value (2)
data label
0000:
11
24
cthA
0002:
77
FF
cthB
0004:
0006:
0008:
000A:
Offset
value
data label
0000:
24
cthA
0001:
11
cthA+1
0002:
FF
cthB
0003:
77
cthB+1
0004:
0005:

Defining DWORD and SDWORD Data (4 bytes)

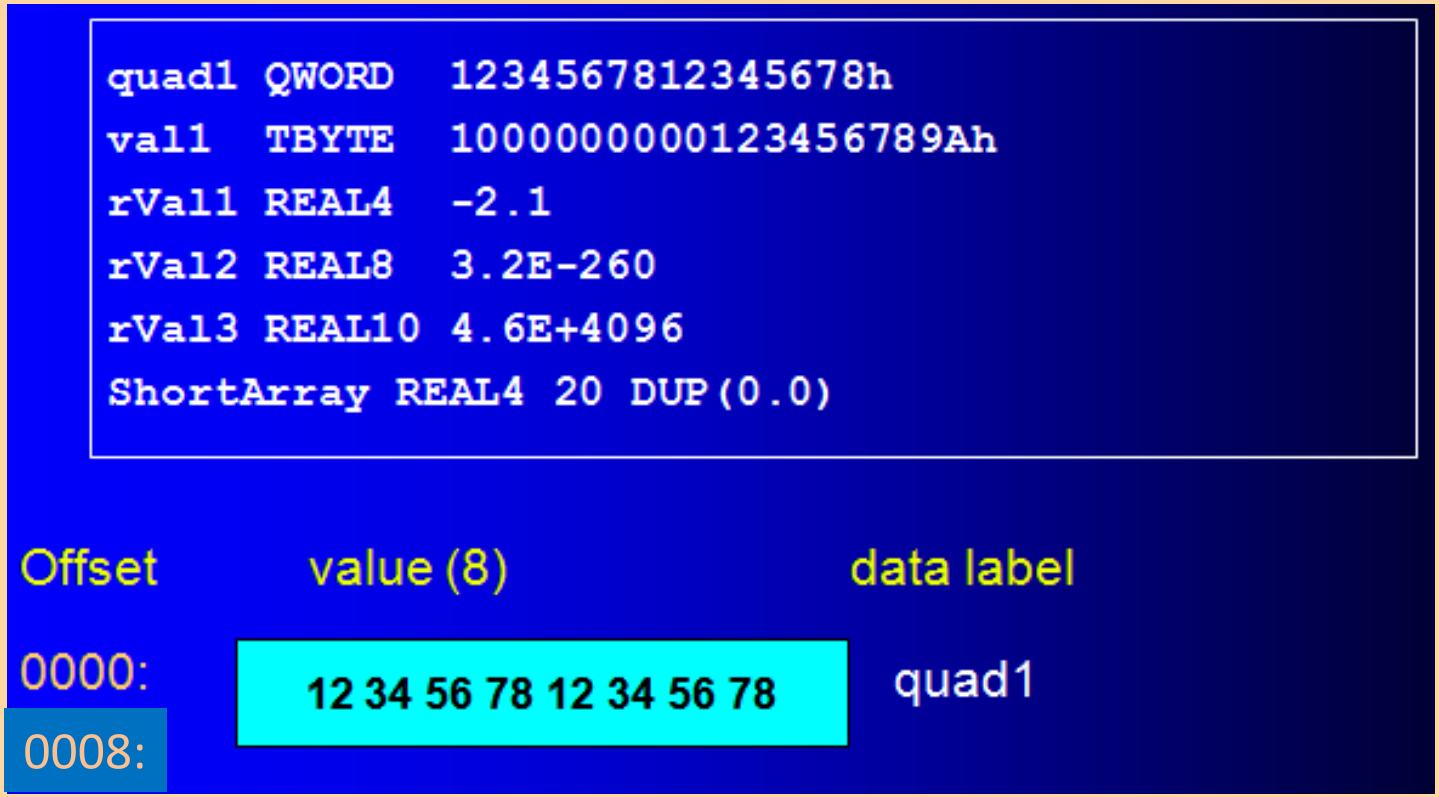
- Storage definitions for signed and unsigned 32-bit integers:

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

Offset	value (4)	data label
0000:	12 34 56 78	val1
0004:	80 00 00 00	val2

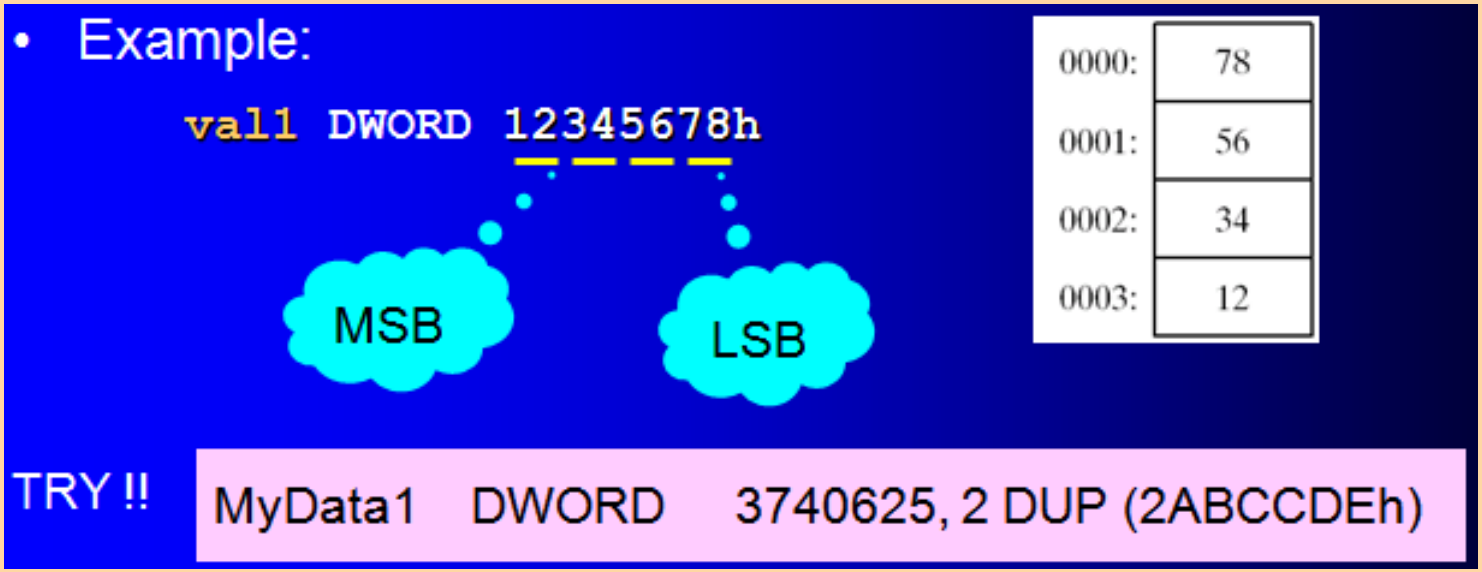
Defining QWORD (8 bytes) , TBYTE (10 bytes), Real Data

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



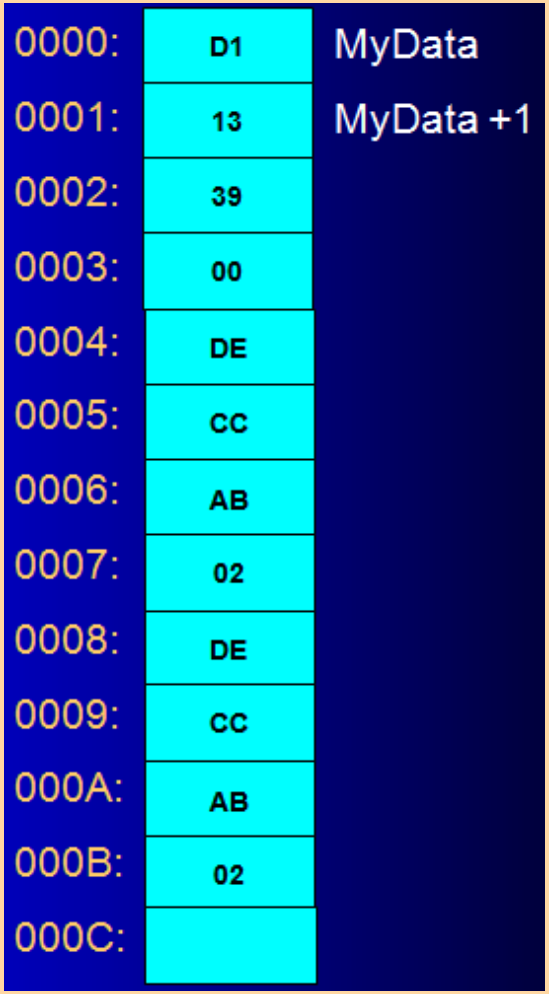
Little Endian Order

- All data types larger than a byte store their individual bytes in reverse order.
- The least significant byte (LSB) occurs at the first (lowest) memory address.



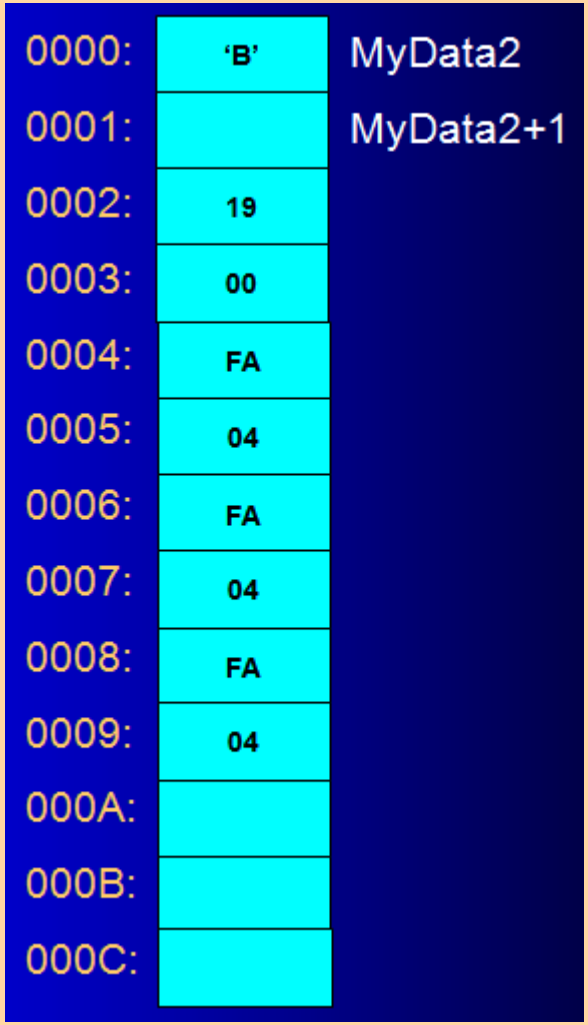
Example 1

- MyData DWORD 3740625, 2 DUP (2ABCCDEh)



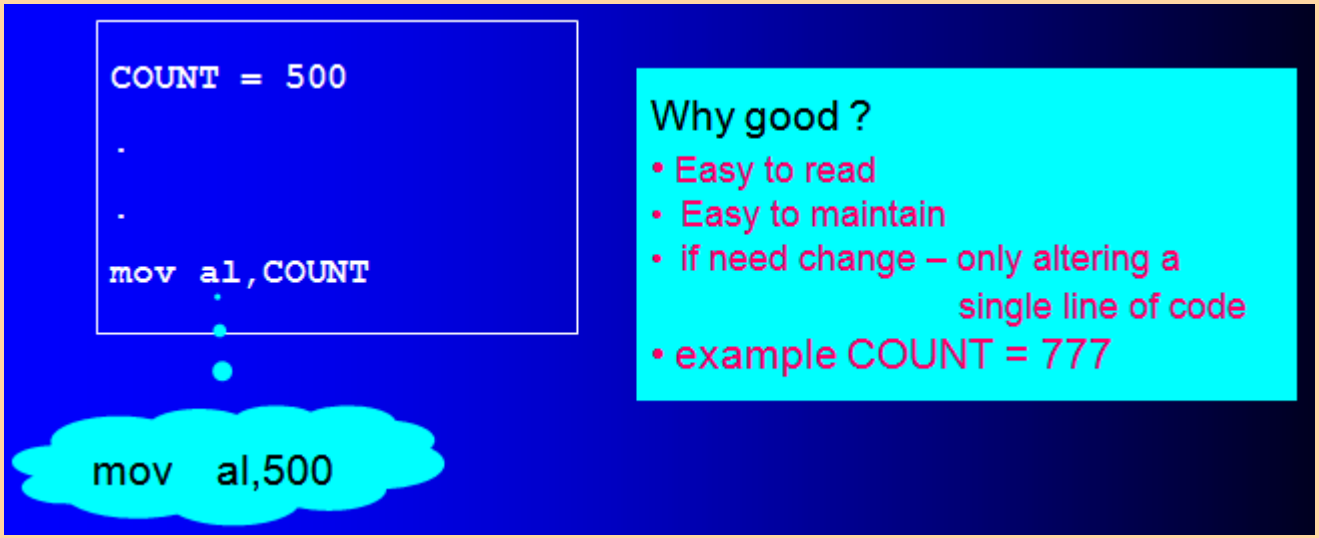
Example 2

- MyData2 WORD ‘B’, 25, 3 DUP (4FAh)



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



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
```