**MASM - Lecture #2** -  **Assembly Language using an x86 processor (Ch. 1 & 2)**

**Intrinsic Data Types**

**S** = signed

**5 Integers:**

* **BYTE**, **SBYTE** - 8-bit unsigned integer; 8-bit signed integer (BYTE also for ASCII characters)
* **WORD**, **SWORD** - 16-bit unsigned & signed integer
* **DWORD**, **SDWORD** - 32-bit unsigned & signed integer
* **QWORD** - 64-bit integer (Quad Word)
* **TBYTE** - 80-bit integer (Ten Byte - 10-8 = 80)

----------------------------------------------------------------------------------------------------------------------------

**3 real numbers (float)**:

* **REAL4** - 4-byte
* **REAL8** -8-byte
* **REAL10** - 10-byte

----------------------------------------------------------------------------------------------------------------------------

**Data Definition Statement** (variable declaration in C++)

* A data definition statement sets aside storage in memory for a variable.
* A name (label) is optional.

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

Ex: value BYTE 10

**Defining BYTE and SBYTE Data** - For small int values, characters, and strings (char arrays)

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 ( + sign is optional and usually not included)

value6 BYTE ? ; uninitialized byte

**Defining Byte Arrays**

list1 BYTE 10,20,30,40 - Define an array named list1 and initialize it with 4 values.

Offset Value

|  |
| --- |
| 10 |
| 20 |
| 30 |
| 40 |

0000

Each array element is1 byte.

0001

0002

0003

list2 BYTE 10,20,30,40

BYTE 50,60,70,80 Multiple lines can be used to initialize one array.

BYTE 81,82,83,84

**Defining Strings**

* A string is implemented as an array of characters.
  + Strings are enclosed in single or double quotation marks, and often null-terminated.

Null character

Examples:

str1 BYTE "Hello.",0

Each character is 1 byte. The null

character ( 0 ) occupies the last byte.

str2 BYTE 'Error: halting program',0

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

message BYTE "MASM is great! "

Zero (not letter O)

BYTE "I love it.",0

* **End-of-line character sequence**: **0Dh** = carriage return **0Ah** = line feed
* **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",0dh,0ah,

"Choice> ",0

I like assembly.

I hope you do.

msg BYTE "I like assembly. ",0dh,0ah

BYTE "I hope you do. ",0

Create a newLine string.

newLine BYTE 0Dh,0Ah,0

|  |
| --- |
| y |
| e |
| s |
| y |
| e |
| s |
| y |
| e |
| s |

**DUP Operator** (Duplicate occurrences)

* Use DUP to allocate (create space for) an array or string.

var1 BYTE 20 DUP(0) ; 20 bytes, all equal to zero

var2 BYTE 20 DUP(?) ; 20 bytes, uninitialized

var3 BYTE 3 DUP("yes") ; 9 bytes: "yesyesyes"

|  |
| --- |
| 10 |
| 0 |
| 0 |
| 20 |

var4 BYTE 10,2 DUP(0),20 ; 4 bytes

**Defining WORD and SWORD Data** - (2 bytes) (word and signed word)

* Define storage for 16-bit integers or double characters
  + Single value or multiple values (array)

word1 WORD 65535 ; largest unsigned value

word2 SWORD –32768 ; smallest signed value

|  |
| --- |
| 1 |
| 2 |
| 3 |
| 4 |
| 5 |

word3 WORD ? ; unsigned, uninitialized

|  |
| --- |
| A |
| B |

word4 WORD "AB"

|  |
| --- |
| ? |
| ? |
| ? |
| ? |
| ? |

myList WORD 1,2,3,4,5 ; array of 16-bit integers

array WORD 5 DUP(?) ; uninitialized array

**Defining DWORD and SDWORD Data** - ( 4 bytes) (double word and signed double word)

Storage definitions for signed and unsigned 32-bit integers:

12345678h

|  |
| --- |
| -332211 |
| -2 |
| -1 |
| 0 |
| 1111 |

val1 DWORD 12345678h ; unsigned

val2 SDWORD –2147483648 ; signed

val3 DWORD 20 DUP(?) ; unsigned array

val4 SDWORD –332211,–2,–1,0,1111 ; signed array

**Defining QWORD, TBYTE, Real Data** - (double word, ten byte values, and real numbers)

1234567812345678h

qVal1 **QWORD** 1234567812345678h

Note: TBYTE values must be hex

when initialized.

tVal2 **TBYTE** 1000000000123456789Ah

rVal3 **REAL4** -2.1

-2.1

|  |
| --- |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

rVal4 **REAL8** 3.2E-260

rVal5 **REAL10** 4.6E+4096

valuesArray REAL4 5 DUP(0.0)

**Little Endian Order**

* x86 processors store and retrieve data from memory using little endian order.
  + All data types larger than a byte store their individual bytes in reverse order.

|  |
| --- |
| 78 |
| 56 |
| 34 |
| 12 |

* + The least significant byte occurs at the first (lowest) memory address.

Ex: val1 DWORD 12345678h

**Big Endian Representation**

|  |
| --- |
| 12 |
| 34 |
| 56 |
| 78 |

* Some computer systems use big endian order (high to low).

Ex: val1 DWORD 12345678h

**Class Exercise #1** - Add 3 unsigned integer values using variables, and display the total.

1. Declare four 32-bit variables named: num1, num2, num,3, and total.

* Initialize the variables with these values:

num1 🡨 3

num2 🡨 4

num3 🡨 5

total 🡨 Don’t initialize with a value.

2. Add the numbers (first add num1 and num2, and then add num3)

3. Store the result in the variable total.

4. Include a string variable named to display the output as shown below.

5. Display the total with output like this:

Note: Call the **CRLF procedure** to display

“Press any key to continue” on the

next line.

The total equals 12

Press any key to continue . . .

6. Display the registers. (EAX should display C - 12 in hex)

NOTE:

* **To output numbers:** First move the numerical value to the EAX register, and then

call the **WriteInt procedure**, or the **WriteDec procedure**. They produce slightly different

results. Try each one.

* **To output c\_strings:** First move the OFFSET of the c\_string to the EDX register, and

then call the WriteString procedure.

**Symbolic Constants**

* Use the **Equal Sign** directive ( = ) to create a integer constant

Ex: COUNT = 500 Note: No memory is allocated for a symbolic constant.

mov eax, COUNT Same.

mov eax, 500

**Calculating the Sizes of Arrays and Strings**

**Current Location Counter** ( **$** ) - The **$** symbol represents the address of the current location.

Ex: myString BYTE “This is a long string, containing”

BYTE “any number of characters”

myString\_len = ($ - myString)

Note: The **$** returns the address of the current location. Therefore, the declaration must

immediately follow the string declaration.

* + - ($ - myString) The address of the first byte of myString (base address) is

subtracted from the current memory address, which gives

the size of myString.

**EQU Directive** - Use to define a **real number constant** **or text**

* Use the Equal-Sign Directive to create a constant

Ex: PI EQU 3.1416

------------------------------------------------------------------------------------------------------------------------------------------------------------

Ex: pressKey EQU <”Press any key ...”,0dh, 0ah, 0>

.data

prompt BYTE pressKey

.code

main PROC

mov edx, offset prompt

call WriteString ; displays “Press any key ...

**Class Exercise #1** - Add 3 unsigned integer values using variables, and display the total.

1. Declare four 32-bit variables named: num1, num2, num,3, and total.

* Initialize the variables with these values:

num1 🡨 3

num2 🡨 4

num3 🡨 5

total 🡨 Don’t initialize with a value.

2. Add the numbers (first add num1 and num2, and then add num3)

3. Store the result in the variable total.

4. Include a string variable named to display the output as shown below.

5. Display the total with output like this:

The total equals 12

6. Display the registers. (EAX should display C - 12 in hex)

**MASM Class Exercise #2** - Read in two input values, multiply them, and display the result.

**1.** Write a program that asks the user to enter two numbers, then adds the numbers and displays

the sum (see output).

**2.** Declare three 32-bit unsigned int variables named: *num1*, *num2*, and *result*.

**3.** Prompt the user to enter an integer value using a string variable named: *prompt*

**4.** Read in the first number and then store it in *num1*.

**5.** Prompt the user for another integer value.

**6.** Read in the number and then store it in *num2*.

**7.** Multiply the numbers and then display the result (see output).

Comment! ---- **OUTPUT** -----

Enter a number: 3 🡨 user enters 3

Enter a number: 5 🡨 user enters 5

3 \* 5 = 15 🡨 Display this using variables.

Press any key to continue . . .

**Note:** The following new Irvine procedures and MASM instructions are used:

* call ReadInt 🡨 (like cin >>) - Reads an int input from the keyboard

and stores the value in the EAX.

* mov num1, eax 🡨 Move the value in the EAX to num1.
* imul eax, num2 🡨 This instruction multiplies the value in the EAX by the value

in num2, with the result left in the EAX.

* Note: The *WriteInt* procedure outputs an integer value stored in the EAX register.

Therefore, the value must first be moved to the EAX register before

* calling *WriteInt*.
* call Crlf 🡨 (This procedure moves the output to the next line (like: cout << endl)