HOMEWORK 2

Supplemental: Basic Control Flow Using JECXZ – Unconditional vs. conditional jumps; JMP vs. JECXZ; translating do-while loops, while loops, and if statements.

§3.1 (Basic Elements of Assembly Language) – with focus on MASM syntax – Integer constants, radix suffixes; integer expressions and operator precedence; real number constants; character constants; string constants; reserved words; identifiers; directives vs. instructions; instruction format: label, mnemonic, operands, comments.

§3.4 (Defining Data) – Intrinsic data types (Table 3-2). Data definition statements. Defining BYTE and SBYTE data; question mark (?) initializer; multiple initializers; defining strings; DUP operator. Defining WORD and SWORD data; memory layout for arrays of words. Defining DWORD, SDWORD, QWORD data. Little vs. big endian. Declaring uninitialized data; .DATA? directive.

Directions: *This assignment is due by **11:00 a.m. on Friday, September 19**. Submit your answer to the programming problem (Question 0) as a .asm file in Canvas; submit your answers to the remaining questions as a PDF in Canvas.*

Scoring (50 total points): *#1: 30 points • #2–3: 3 points each • #4–5: 7 points each*

1. Submit a .asm file in Canvas for a program that behaves as follows. Do not use any assembly instructions or library procedures other than those presented in the lectures or labs. Comment your code. Solutions will be graded based on both correctness and readability.

   (1) Read a nonnegative integer from the keyboard. (We'll call it *n*.)

   (2) Use spaces, slashes, and backslashes to display an "ASCII art" V with *n* lines, shown below.

   If *n* = 0, don't display anything.

   If *n* = 1, display one line:

   ```
   \/
   ```

   If *n* = 2, display two lines (a gray dot · denotes a space):

   ```
   \··/
   ·\/
   ```

   If *n* = 3, display three lines:

   ```
   \····/
   ·\··/
   ··\/
   ```

   If *n* = 4, display four lines:

   ```
   \······/
   ·\····/
   ··\··/
   ···\/
   ```

   If *n* = 5, display five lines:

   ```
   \········/
   ·\······/
   ··\····/
   ···\··/
   ····\/
   ```

   And so forth. Your solution should work for an arbitrary value of *n*. For large values of *n,* the lines will wrap because the window is not wide enough; that is fine.

2. Consider the following pseudocode.

```
Initialize a DWORD in memory to 0
while (ECX ≠ 0) {
        Read a nonnegative integer from the keyboard into EAX
        Increase the value stored in memory by adding the value from EAX
        Decrease the value of ECX by 1
}
Display the value stored in memory
```

Add labels and jump instructions to the following code to implement the *while* loop.

```
.data
saved DWORD 0

.code


    ; Loop body: read from keyboard, add to memory, decrement ECX
    call ReadDec
    add saved, eax
    sub ecx, 1


    ; Display value stored in memory
    mov eax, saved
    call WriteDec
```

3. Storing the doubleword 6A7B8B6Ah requires four bytes of memory. How would these bytes be laid out in memory on a computer that uses (a) big endian byte ordering? (b) little endian?

4. Consider the following .data section.

```
.data
a SWORD 10, 10h, -3
  WORD  0, 2 DUP(3)
  DWORD 83829FC0h
```

(a) How many bytes of memory are allocated to store this data?

(b) How will this data be stored in memory as a sequence of bytes? Write the byte values in hexadecimal, starting from the byte at the lowest memory address.

5. For each of the following data declarations, determine if the data declaration is valid or invalid. (Try to figure it out yourself first – this is better preparation for exams – then use MASM to check.) If the declaration is invalid, (a) write the error message that MASM produces, then (b) describe, in your own words, what is wrong with the declaration. If there is an obvious way to correct the error, write the corrected declaration.

(a) `nums WORD 10 20 30 40        ; Array of four words`

(b) `BYTE ?`

(c) `BYTE 256`

(d) `WORD 'x'`

(e) `WORD "Hello",0`

(f) `twofiftyfive WORD FFh        ; Hexadecimal FF`

(g) `ebp BYTE "ebp",0             ; Null-terminated string ebp`

(h) `empty DWORD 4*1024 DUP(?)`