HOMEWORK 4

Directions: *This assignment is due by **11:59 p.m. on Monday, October 20**. Five bonus points will be awarded if it is submitted by 11:59 p.m. on Wednesday, October 15. Submit your answer to the programming problems (Questions 1–3) as a single .asm file in Canvas. Submit your answers to the remaining questions as a PDF in Canvas.*

Scoring (50 total points + 5 bonus): *#1–2: 10 pts each • #3: 14 points • #4–7: 4 pts each*

QUESTIONS 1–3: PROGRAMMING

Download **HW4.asm** from Canvas. In the TITLE line, replace TODO: YOUR NAME with your name. Then, fill in the missing procedures following Questions 1–3 below. Note that the .asm file includes several tests for each of the procedures you will write.

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. **Conversion to uppercase.** Fill in the ucase procedure as described in HW4.asm.
   Hint: Look at an ASCII table. How can you determine if a letter is lowercase? What's the relationship between the ASCII code for a lowercase letter and the ASCII code for the same letter in uppercase?

2. **Implementation of strrchr.** Fill in the strrchr procedure as described in HW4.asm.

3. **Exponentiation.** Fill in the pow procedure in HW4.asm as follows:

   - The procedure receives two register arguments. We'll call them *a* and *n*; they're passed in EAX and EBX, respectively. Both are 32-bit signed integers. The goal is to compute a^n.

   - Compute a^n using the (recursive) algorithm described by the following pseudocode:[1]

```
POW(a, n) {
    if n == 1 {
        return a
    } else {
        x = POW(a, n/2)      // Call Pow recursively
        if n is even {
            return x * x
        } else {
            return x * x * a
        }
    }
}
```

   - You may **assume** that $n \geq 1$ and that the result will be in the range of values that can be represented by 32-bit signed integers. (You do not need to test for these.)

   - *Note: There is nothing unusual about calling a procedure recursively in MASM. It's exactly the same as any other procedure call.*

[1] If you took my Algorithms course and this looks familiar, it was used as an example of a Decrease & Conquer algorithm. We started with a straightforward mathematical identity and eventually built up this $\Theta(\lg n)$ algorithm, which we called FASTPOWER.

QUESTIONS 4–7: ANALYSIS OF MEMORY ACCESSES

The program below contains a procedure `lastElement` that returns the last element of an array. The `main` procedure calls `lastElement` to retrieve the last value of an array, then calls `WriteHex` to display it.

```
INCLUDE Irvine32.inc

.data
start = $
     BYTE  0, 0, 0
arr1 DWORD 1, 2, 3
arr2 DWORD 0AABBCCDDh, 0FFEEDDCCh
     BYTE  0, 0, 0

.code
main PROC
       ; Displays the last element of arr1: 00000003
       mov esi, OFFSET arr1
       mov ecx, LENGTHOF arr1
       call lastElement
       call WriteHex
       call Crlf

       ; Displays the last element of arr2: FFEEDDCC
       mov esi, OFFSET arr2
       mov ecx, LENGTHOF arr2
       call lastElement
       call WriteHex
       call Crlf

       exit
main ENDP

; Returns the value of the last element in an DWORD array
; Receives: ESI -- Pointer to a DWORD array
;           ECX -- Total # of elements in array (must be >= 1)
; Returns:  EAX -- Value of the last element
lastElement PROC
       mov eax, DWORD PTR [esi + ecx*SIZEOF SDWORD - SIZEOF SDWORD]
       ret
lastElement ENDP

END main
```

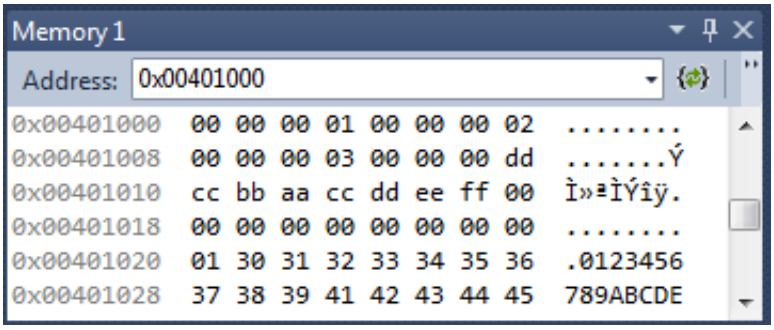
Each of the remaining questions contains a `main` procedure and an implementation of `lastElement`. Either `lastElement` is implemented incorrectly, or it is called incorrectly.

Each of these programs is "wrong." It's easy to fix them (just compare them to the code above). Your goal is not to fix them: it is to understand *why* these "wrong" programs behave the way they do.

For each of questions 4–7, answer the following three subquestions:

a. Consider the memory operand for the `mov` instruction in the `lastElement` procedure. If `start` = 00401000h, what memory address does it access?

b. The following image shows the 48 bytes of the memory, beginning at 00401000h. `start` = 00401000h, what four bytes comprise the DWORD value that is copied into EAX? Circle them. If the memory address that is accessed is not shown, write "Not shown."



c. For every program, the expected output is 00000003—the last value in the `arr1` array.. What will the *actual* output be? Or will the program crash due to an invalid memory access?

IMPORTANT: For parts (a) and (b), use the .data section from the code above, assume `start` = 00401000h, and arrive at your answers analytically. Don't just assemble the code and write down whatever shows up in the debugger. MASM will almost certainly choose a different starting address, and your answers will be wrong.

4.
```
main PROC       ; Display the last element of arr1
     mov esi, arr1
     mov ecx, LENGTHOF arr1
     call lastElement
     call WriteHex
     exit
main ENDP

lastElement PROC
     mov eax, DWORD PTR [esi + ecx*SIZEOF SDWORD - SIZEOF SDWORD]
     ret
lastElement ENDP
```

5.
```
main PROC       ; Display the last element of arr1
     mov esi, OFFSET arr1
     mov ecx, LENGTHOF arr1
     call lastElement
     call WriteHex
     exit
main ENDP

lastElement PROC
     mov eax, DWORD PTR [esi + ecx*SIZEOF SDWORD]
     ret
lastElement ENDP
```

6.
```
main PROC       ; Display the last element of arr1
     mov esi, OFFSET arr1
     mov ecx, LENGTHOF arr1
     call lastElement
     call WriteHex
     exit
main ENDP

lastElement PROC
     mov eax, DWORD PTR [esi + ecx - SIZEOF SDWORD]
     ret
lastElement ENDP
```

7.
```
main PROC       ; Display the last element of arr1
     mov esi, OFFSET arr1
     mov ecx, LENGTHOF arr1
     call lastElement
     call WriteHex
     exit
main ENDP

lastElement PROC
     mov eax, DWORD PTR [esi + ecx*SIZEOF SDWORD - 2*SIZEOF SDWORD]
     ret
lastElement ENDP
```