### Experiment Guidebook 1

#### Experimental requirements and objective

1. Be able to code, assemble, execute and debug a program with Visual Studio and MASM.
2. Know how to link your programs to an external code library.
3. Know how to implement arithmetic operations, memory accessing and counting loops using assembly language.

#### Experimental environment

1. Hardware environment

The x86 microcomputer CPU more than Pentium, more than 120GB capacity hard drive, more than 1GB of memory.

1. Software environment

Visual Studio 2008 or above version.

#### Experimental contents

1. Arithmetic Expressions

Write a program that implements the following arithmetic expressions:

val2 = −val2

val2 = val2 – 1

val3 = val3 + 2

AL = −val2 + 7 − val3 + val1

Use the following data definitions:

val1 SBYTE 8

val2 SBYTE -128

val3 SBYTE 127

Use **register window** and **watch window** in debugging mode to show the changes made by each instruction of your program to the **variables, registers and flags**.

Your report needs to include your code and record the changes made by your program and explain why these changes happen. In terms of flags, you only need to concern carry flag (CY), overflow flag (OV), zero flag (ZR) and sign flag (PL).

; AddTwo.asm - adds two 32-bit integers.

; Chapter 3 example

.386

.model flat,stdcall

.stack 4096

ExitProcess proto,dwExitCode:dword

.data

array DWORD 1,1,3,5,7,9,11,13,15,17 ; Fibonacci number of DWORD's array

.data

val1 SBYTE 8

val2 SBYTE -128

val3 SBYTE 127

.code

main proc

mov al,val2 ;move val2 value -128 to al,CF=0,OF=0

neg al ;let the value of al be negative al=128,CF=0,OF=1

dec al ;al=128-1,al=127,CF=0,OF=0

mov ah,val3 ;move val3 with the value of 127 to ah,CF=0,OF=0

add ah,2 ;add 2 to ah, ah=127+2, ah=129, OF=1,CF=1

neg al ;let the value of al to be negative,al=-127, CF=0,OF=0

add al,7 ;move the value of 7 into al,al=-120,CF=0,OF=0

neg ah ;let the number of ah be negative, ah=-129,CF=0,OF=1

add al,ah ;add the value of al and ah,al=al+ah=-120-129=-249,CF=1,OF=1

mov bh,val1 ;move the value of 8 to bh,bh=8,CF=0,OF=0

add al,bh ;add the value of al and bh,al=al+bh,al=-249+8=-241,CF=1,OF=1

invoke ExitProcess,0

main endp

end main

1. Fibonacci Numbers

Write a program that uses a loop to calculate the first ten values of the Fibonacci number sequence, described by the following formula: Fib(1) = 1, Fib(2) = 1, Fib(n) = Fib(n -1) + Fib(n - 2). Place these values in an array and display it by inserting the following statements immediately after the loop to display the hexadecimal contents of the target array:

mov esi,OFFSET target ; offset of target variable

mov ebx,TYPE target ; byte format

mov ecx,LENGTHOF target ; counter

call DumpMem

(The DumpMem method is from the Irvine32 library. See Chapter 5.3.2 for details.)

Your report needs to include your code and explain each line of your code and record the output of DumpMem in the consoler.

; AddTwo.asm - adds two 32-bit integers.

; Chapter 3 example

.386

.model flat,stdcall

.stack 4096

ExitProcess proto,dwExitCode:dword

DumpMem PROTO

.data

array DWORD 1,1,3,5,7,9,11,13,15,17 ; Fibonacci number of DWORD's array

.code

main proc

mov esi,OFFSET array ; offset of target array

mov ebx,TYPE array ; byte format

mov ecx,LENGTHOF array ; counter

call DumpMem ; call DumpMem subroutine to display hexadecimal contents

mov eax, 1 ; initialize Fib(1) = 1

mov ebx, 1 ; initialize Fib(2) = 1

mov edi, 10 ; counter for loop 10 times

L1:

add eax, ebx ; calculate the next Fibonacci number

mov [esi + ecx\*4], eax ; store the number in the array

mov eax, ebx ; update Fib(n-1)

mov ebx, [esi + ecx\*4] ; update Fib(n-2)

loop L1 ; loop until counter reaches 0

invoke ExitProcess,0

main endp

end main

1. Matrix Transposition

Write a program to declare a two-dimensional matrix named **sourceTable** with 4 rows and 3 columns and use **nested loops** to transpose the matrix and save the transposed matrix to another matrix named **targetTable**. Use the SIZEOF, TYPE, and LENGTHOF operators to make the program as flexible as possible if the matrix size and type should be changed in the future. Use DumpMem to display sourceTable and targetTable after the transposition.

Tip: the declaration of a two-dimensional matrix:

sourceTable BYTE 1, 2, 3

BYTE 4, 5, 6

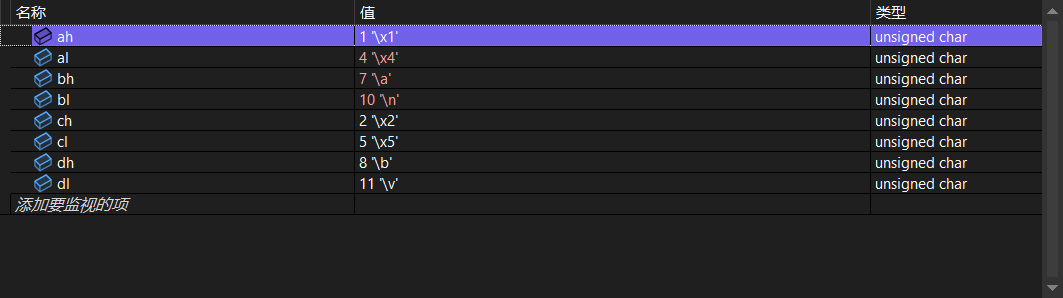
BYTE 7, 8, 9

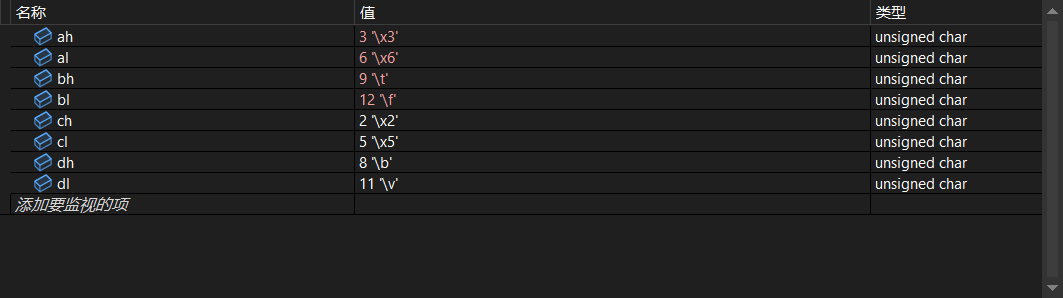
BYTE 10, 11, 12

tableSize = ($ - sourceTable)

targeTable BYTE tableSize DUP(?)

Your report needs to include your code and explain each line of your code and record the output of DumpMem in the consoler.





DumpMem PROTO

.data

sourceTable BYTE 1, 2, 3

BYTE 4, 5, 6

BYTE 7, 8, 9

BYTE 10, 11, 12

tableSize = ($ - sourceTable) ;calculate the size of the sourcetable

targetTable BYTE tableSize DUP(?)

.code

main proc

mov ah,sourceTable ;move the first value of array in sourceTable to ah

mov al,sourceTable+3 ;move the fourth value of array in sourceTable to al

mov bh,sourceTable+6 ;move the seventh value of array in sourceTable to bh

mov bl,sourceTable+9 ;move the tenth value of array in sourceTable to bl

mov ch,sourceTable+1 ;move the second value of array in sourceTable to ch

mov cl,sourceTable+4 ;move the fifth value of array in sourceTableto cl

mov dh,sourceTable+7 ;move the eight value of array in sourceTable to dh

mov dl,sourceTable+10 ;move the eleventh value of array in sourceTable to dl

mov ah,sourceTable+2 ;move the third value of array in sourceTable to ah

mov al,sourceTable+5 ;move the sixth value of array in sourceTable to al

mov bh,sourceTable+8 ;move the nine value of array in sourceTable to bh

mov bl,sourceTable+11 ;move the twelve value of array in sourceTable to bl

call DumpMem