Experiment 1： Loop and Branch Programming

Name:韦杨婧 Student Number:2020329621199

1. Experiment Requirements and Objective

a) Be able to code, assemble, and execute a program with Visual C++ and MASM.

b) Know how to link your programs to an external code library.

c) Know how to create conditional and looping structures using assembly language.

2. Experiment Environment

a) Hardware environment

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

b) Software environment

Visual Studio 2008 and above versions of applications.

3. Experiment Content

a) The sum and the maximum

Write a program that inputs 10 decimal integers from the keyboard and outputs the sum and the maximum value of them on the console window.

b) Print Fibonacci until overflow

Write a program that calculates and displays the Fibonacci number sequence {1, 1, 2, 3, 5, 8, 13 …}, stopping only when the Carry flag is set. Display each unsigned decimal integer value on a separate line.

c) Test Score Evaluation

Using the following table as a guide, write a program that asks the user to enter multiple integer test scores until the test score is less than 0 or greater than 100.

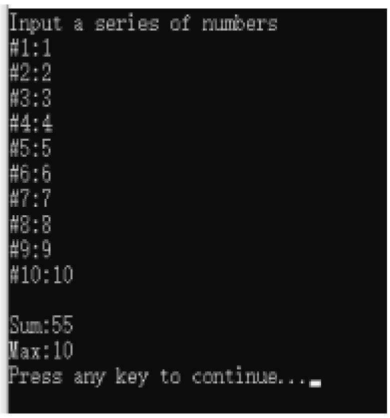
Score Range	Letter Grade
90 to 100	A
80 to 89	B
70 to 79	C
60 to 69	D
0 to 59	F

The program should accumulate a counter of the number of test scores, and display test scores in descending order with an appropriate letter grade for each test score

4. Experiment Result

Screenshots of program execution:

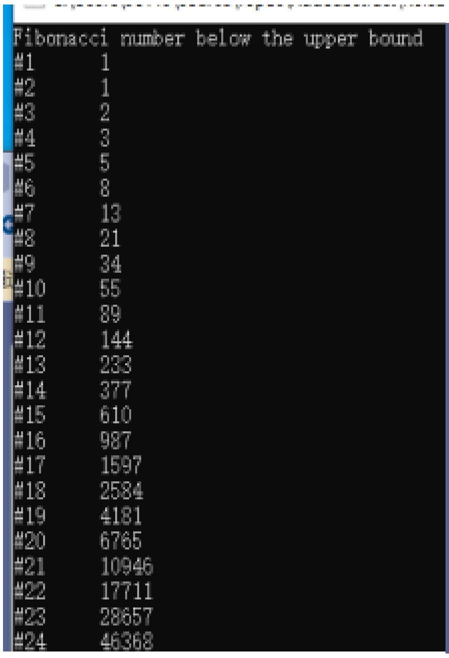
a) Content one:



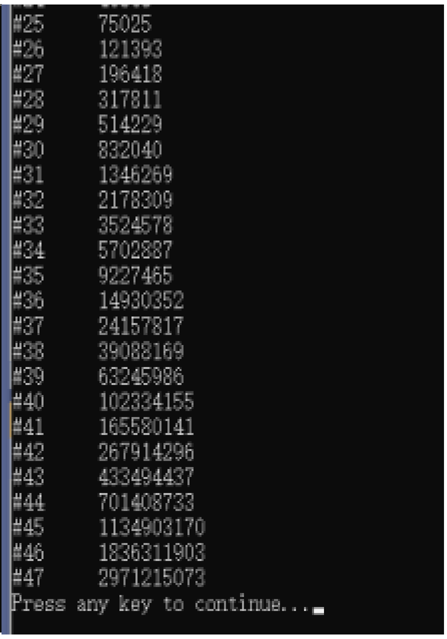
```
Input a series of numbers
#1:1
#2:2
#3:3
#4:4
#5:5
#6:6
#7:7
#8:8
#9:9
#10:10

Sum:55
Max:10
Press any key to continue...
```

b) Content two:

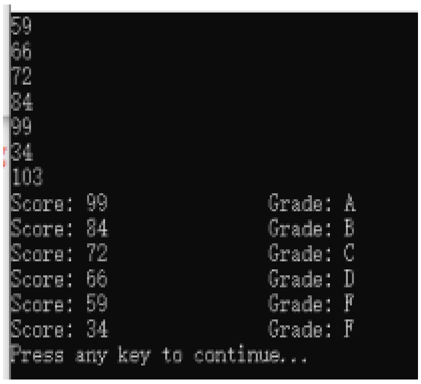


```
Fibonacci number below the upper bound
#1      1
#2      1
#3      2
#4      3
#5      5
#6      8
#7      13
#8      21
#9      34
#10     55
#11     89
#12     144
#13     233
#14     377
#15     610
#16     987
#17     1597
#18     2584
#19     4181
#20     6765
#21     10946
#22     17711
#23     28657
#24     46368
```

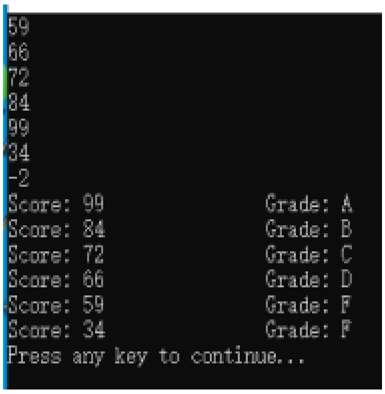


```
#25     75025
#26     121393
#27     196418
#28     317811
#29     514229
#30     832040
#31     1346269
#32     2178309
#33     3524578
#34     5702887
#35     9227465
#36     14930352
#37     24157817
#38     39088169
#39     63245986
#40     102334155
#41     165580141
#42     267914296
#43     433494437
#44     701408733
#45     1134903170
#46     1836311903
#47     2971215073
Press any key to continue...
```

c) Content three:



```
59
66
72
84
99
34
103
Score: 99                Grade: A
Score: 84                Grade: B
Score: 72                Grade: C
Score: 66                Grade: D
Score: 59                Grade: F
Score: 34                Grade: F
Press any key to continue...
```



```
59
66
72
84
99
34
-2
Score: 99                Grade: A
Score: 84                Grade: B
Score: 72                Grade: C
Score: 66                Grade: D
Score: 59                Grade: F
Score: 34                Grade: F
Press any key to continue...
```

End with the number larger than 100   End with the number smaller 0

5. Source Code of Programs

```
title one
include Irvine32.inc
.data
    num      dword 10 dup(?)
    sum      dword 0
    max      dword ?
    hint BYTE ″Input a series of numbers″,0
    hints    BYTE ″Sum:″,0
    hintm    BYTE ″Max:″,0

.code
main proc
    mov      ecx,10
    mov      esi,0
    ;mov sum,0
    mov      edx,OFFSET hint
    call WriteString
    call Crlf
l1:
    mov      al,’#’
    call WriteChar
    mov      ebx,esi
    sar      ebx,2    ;右移2位
    add      ebx,1
    mov      eax,ebx
    call WriteDec
    mov      al,’:’
    call WriteChar
    call ReadInt  ;读入后放入eax寄存器里
    add      sum,eax
    cmp      esi,0
    jne      notFirstInt
    mov      max,eax

notFirstInt:
    cmp      max,eax
    jnb      l2
    mov      max,eax

l2:
    add      esi,4
    loop l1
    call Crlf
    mov      edx,OFFSET hints
    call WriteString
    mov      eax,sum
    call WriteDec
    call Crlf
    mov      edx,OFFSET hintm
    call WriteString
    mov      eax,max
    call WriteDec
    call Crlf
    call WaitMsg
    exit

main endp
end main
;.data
;    num dword 10 dup(?)
;    sum dword ?
;    max dword ?

;.code
;    mov ecx,10
;    mov esi,0
;    mov sum,0
;next:
;    call readdec
;    mov num[esi],eax
;    add sum,eax
```

```
title two

INCLUDE Irvine32.inc

.data
    temp DWORD 0
    count    DWORD 1
    hint BYTE ″Fibonacci number below the upper bound″,0

.code
main proc
    mov      edx,OFFSET hint
    call WriteString
    call Crlf
    MOV      AL,’#’
    CALL WriteChar
    mov      eax,count
    call WriteDec
    add      count,1
    mov      al,’ ’
    call WriteChar
    mov      eax,1
    call WriteDec
    call Crlf
    MOV      AL,’#’
    CALL WriteChar
    mov      eax,count
    call WriteDec
    add      count,1
    mov      al,’ ’
    call WriteChar
    mov      eax,1
    call WriteDec
    call Crlf
    mov      ebx,1
    mov      edx,1
    jmp      processing

processing:
    mov      temp,edx
    add      edx,ebx
    jc       outstep
    MOV      AL,’#’
    CALL WriteChar
    mov     eax,count
    call WriteDec
    add     count,1
    mov     al,' '
    call WriteChar
    mov     eax,edx
    call WriteDec
    call Crlf
    mov     ebx,temp
    jmp     processing

outstep:
    call WaitMsg
    exit

main endp
end main
```

```
title  Again

INCLUDE  Irvine32.inc

.data
    array    dword    128     DUP(?)
    time dword    0
    desOut   dword    ?
    desIn    dword    ?
    total    dword    ?
    four DWORD    4
    hintScore        BYTE "Score: ",0
    hintGrade        BYTE "       Grade: ",0

.code
main proc
    mov     ebx, 0
    mov     edx, 0
Input:
    call ReadInt
    cmp     eax,0
    jl      Bubble
    cmp     eax,100
    jg       Bubble
    mov      [ebx+array],eax
    add      ebx,4
    mov      eax,lengthof array
    mul      four
    cmp      ebx,lengthof array
    jge      Bubble
    jmp      Input

Bubble:
    mov      eax,ebx
    dec      eax
    mul      four
    mov      desOut,eax
    add      eax,4
    mov      desIn,eax
    mov      ebx,0
    mov      edx,0
    .while   ebx < desOut
        mov      edx,ebx
        add      edx,4
        .while  edx  < desIn
            mov  ecx,[edx+array]
            .if [ebx+array] < ecx
                mov  eax,[ebx+array]
                mov  esi,[edx+array]
                mov [edx+array],eax
                mov [ebx+array],esi
            .endif
            add   edx,4
        .endw
        add   ebx,4
    .endw

    mov  ebx,0
    .while   ebx < desIn
Loop1:
        mov      eax,[ebx+array]
        cmp      eax,0
        jle      Over
        jmp      levelA
Iter:
        add      ebx,4
    .endw
```

```
levelA:
    cmp      eax,90
    jl       levelB
    mov      edx,OFFSET hintScore
    call WriteString
    call WriteDec
    mov      edx,OFFSET hintGrade
    call WriteString
    mov      al,’A’
    call WriteChar
    call Crlf
    jmp      Iter

levelB:
    cmp      eax,80
    jl       levelC
    mov      edx,OFFSET hintScore
    call WriteString
    call WriteDec
    mov      edx,OFFSET hintGrade
    call WriteString
    mov      al,’B’
    call WriteChar
    call Crlf
    jmp      Iter

levelC:
    cmp      eax,70
    jl       levelD
    mov      edx,OFFSET hintScore
    call WriteString
    call WriteDec
    mov      edx,OFFSET hintGrade
    call WriteString
    mov      al,’C’
    call WriteChar
    call Crlf
    jmp      Iter

levelD:
    cmp      eax,60
    jl       levelF
    mov      edx,OFFSET hintScore
    call WriteString
    call WriteDec
    mov      edx,OFFSET hintGrade
    call WriteString
    mov      al,’D’
    call WriteChar
    call Crlf
    jmp      Iter

levelF:
    mov      edx,OFFSET hintScore
    call WriteString
    call WriteDec
    mov      edx,OFFSET hintGrade
    call WriteString
    mov      al,’F’
    call WriteChar
    call Crlf
    jmp      Iter

Over:
    call WaitMsg
    exit
main endp
end main
```

6. Summary

Preparations:
For the whole experiment, we prepare the environment first:

- Install Visual Studio and choose the corresponding addon.
- Install the library: Irvine32.inc.
- Add and link the path in the properties (Three places need to be modified).

Then, we can start with the creating a project

- Add the source file
- Modify the properties of the project → Generate the dependencies → Generate the self-defined → choose the masm
- Start to write

Writing the program:
a) For the content one, we can turn the requirements into the following contents:

- Get the input from the keyboard
- Find the maximal number
- Sum the numbers up.

We can decide whether to use the array or not. Here we choose not to use the array. Use two variables. One for storing the total, the other for storing the maximal number. The registers inside:

- Eax → Store the number input
- Al → Output the character
- Edx → Output the string

Add more details to make the interface more friendly.
Use some procedures in the library:

- WriteString → Output the string
- WriteChar → Output a single character
- WriteDec → Output an unsigned integer
- ReadInt → Get an integer from the keyboard
- Crlf → Line feed
- WaitMsg → Pause when complete the work

b) For the content two, we can turn the requirements into the following contents:

- Repeated calculation between the numbers, with the help of the registers. (We cannot operate with two memory operands)
- Check the carry flag instead of the overflow flag because of the unsigned integers.
- Output the number until it overflows. (Conditional jump)

We can also decide whether to use the array or not. Here we choose not to use the array. Use two variables. One for storing the current number. The other for counting the times we output the Fibonacci number. Notice that flags will change through each arithmetic. The register inside:

- Eax → Store the number input
- Al → Output the character
- Edx → Output the string/Storing the Fibonacci number (Larger one)
- Ebx → Storing the Fibonacci number (Smaller one)

Add more details to make the interface more friendly.

Use some procedures in the library: The same as content one.

c) For the content three, we can turn the requirements into the following contents:

- Get the score input from the keyboard
- Store a series of scores in the array (Connect to the offset)
- Sort them in descending order (Conditional jump and loops)
- Judge the grade according to the score (Conditional jump)
- Output the number and the corresponding grade

Here we must use the array since we need to do the sorting. Pay attention to:

- Memory operands, it's too easy to make mistakes like operating with two memory operands.
- The same as the offset due to the "DWORD" variables. Each time we shift two bits instead of self-increasing.
- Data transformation through the memory operands and registers. Mistakes may be occurred (Covered by other numbers) due to the incorrect and careless operations.

Registers:

- Eax → Store the number input
- Al → Output the character
- Edx → Output the string/ Inner loop iterative number
- Ebx → Outer loop iterative number/Data transformation
- Esi/Ecx → Data transformation

Add more details to make the interface more friendly.

Use some procedures in the library: The same as content one.

In conclusion:

- Easy operations but lengthy procedures
- Sometimes what we thought is easy to realize with high-level language may not be easy to realize with the assembly language.
- More precisely coding, otherwise it has a higher probability to make mistakes.