CS253 Laboratory session 3

Part 1: Evaluating floating point expressions using the maths co-processor.

CODE:

```
.286
.model medium
.8087
.stack 100h

.DATA
    SX      dd 3.0          ;
    SY      dd 4.0          ;
    cntrl   dw 03FFh        ;
    stat    dw 0            ;
    INTG    dw 0            ;
    CRLF    db 0Dh,0Ah,'$'  ;

.CODE
.STARTUP
    FINIT
    FLDCW   cntrl

    ;—— HY = sqrt(SX² + SY²) ——————————————————
    FLD     SX
    FMUL    ST,ST(0)        ; ST0 = SX*SX
    FLD     SY
    FMUL    ST,ST(0)        ; ST0 = SY*SY
    FADD    ST,ST(1)        ; ST0 = SX²+SY²
    FSQRT                   ; ST0 = √(SX²+SY²)
    FSTSW   stat
    mov     ax,stat
    and     al,0BFh
    jnz     DONE
    FSTP    HY              ;

   _
    mov     bx,OFFSET HY
    add     bx,2
    mov     ax,[bx]
    mov     bx,ax
    mov     cx,16
_P_HIGH:
    rol     bx,1
    jc      _H1
    mov     dl,'0'
    jmp     _H2
_H1:
    mov     dl,'1'
_H2:
    mov     ah,02h
    int     21h
    loop    _P_HIGH

    mov     bx,OFFSET HY
    mov     ax,[bx]
    mov     bx,ax
    mov     cx,16
_P_LOW:
    rol     bx,1
    jc      _L1
    mov     dl,'0'
    jmp     _L2
_L1:
    mov     dl,'1'
_L2:
    mov     ah,02h
    int     21h
    loop    _P_LOW
```

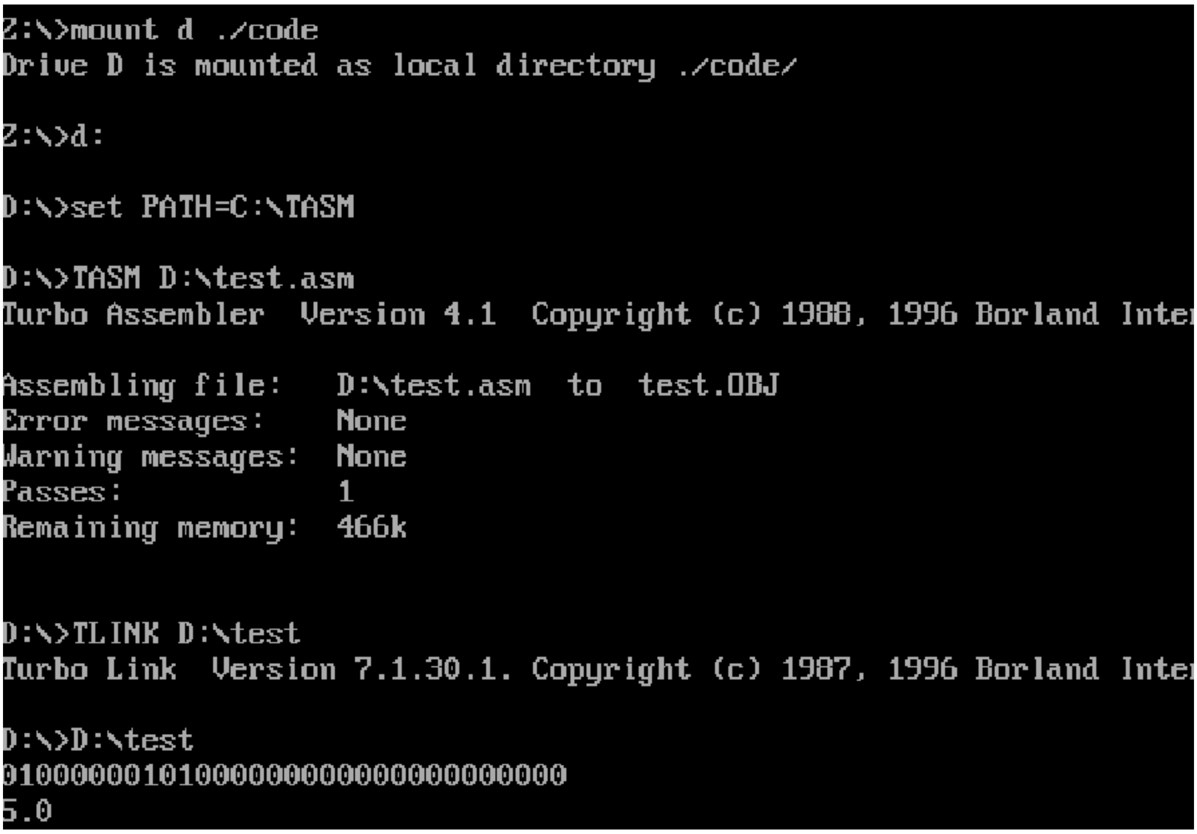
```
    mov     dx,OFFSET CRLF
    mov     ah,09h
    int     21h

    fld     HY
    fistp   word ptr INTG    ; INTG ← 5

    mov     ax,INTG
    xor     dx,dx
    mov     bx,10
    div     bx
    cmp     ax,0
    je      _PRINT_REM
    add     al,'0'
    mov     dl,al
    mov     ah,02h
    int     21h
_PRINT_REM:
    add     dl,'0'
    mov     ah,02h
    int     21h


    mov     dl,'.'
    mov     ah,02h
    int     21h
    mov     dl,'0'
    mov     ah,02h
    int     21h

DONE:
    .EXIT
END
```



Part 2: So far we have created and compiled a number of assembly language programs using the Microsoft assembler (MASM). Writing a big program using MASM is probably impractical. However, you can embed assembly language in your C\C++ programs. This is something that you could do in practice so as to optimise a piece of code or make use of instructions that are not accessible via the standard libraries (such a MMX, multimedia extension).

This approach has the benefit of allowing you to write the input and output in C/C++ and use the assembly language for high speed calculation.

It should be said that in practice compilers are so good that it is very difficult to write better machine code than they can generate.

CODE

```
// MASM_FP.cpp : Defines the entry point for the console application.
#include "stdafx.h"
#include <stdio.h>

void test(void); // Function prototype (description)
int _tmain(int argc, _TCHAR* argv[])
{
	test();
	return 0;
}

// Put our unmanaged asm code in here
void test()
{
	unsigned short num1;
	unsigned short num2;
	unsigned short result;

	printf("Enter first number: ");
	scanf(" %hd",&num1);
	printf("Enter second number: ");
	scanf(" %hd",&num2);

	__asm
	{
		mov ax, num1 ; Direct addressing to access num1
		mov bx, num2 ; put value in num2 into bx
		add ax, bx ; ax=ax+bx
		mov result,ax; ; put value into result
	}

	printf("%hd",result); // Display result

	// Wait for enter to be pressed before terminating
	while(getchar()!=10); // Clear buffer of previous <ret>
	while(getchar()!=10); // Wait for a new <ret>
}
```

Part 3: Use the sample code on the next page to generate a suitable frame work for creating code to implement a floating point calculation using Assembly Language from within a C++ program. The program puts the contents of variables on the floating stack and then takes the value on the floating point stack and puts it back into the variable C.

Program 2

```
// MASM_FP.cpp : Defines the entry point for the console application.
#include "stdafx.h"
#include <stdio.h>

void test(void); // Function prototype (description)
int _tmain(int argc, _TCHAR* argv[])
{
test();
return 0;
}

void test()
{
 float A=2,C=0;
 unsigned short cntrl=0x3FF,stat;
 __asm
 {
  FINIT
  FLDCW cntrl ; Round even, Mask Interrupts
  FLD A         ; Push SX onto FP stack

  FSTSW stat    ; Load FPU status into [stat]
  FSTP C              ; Copy result from stack into HY
 }

// Binary representation of the 4 bytes, (32 bits) coding HY
printf("Binary:");
unsigned char byt;
```

```
for(int x=3;x>=0;x--)
{
 byt=*((unsigned char *)&C+x);
 for(int y=128;y>0;y/=2)
 {
  if ((y&byt)==0) printf("0"); else printf("1");
 }
}

// Decimal format
printf("\nDecimal: %3.0f",C);

// Hex format
printf("\nHex:");
for(int x=3;x>=0;x--)
{
 byt=*((unsigned char *)&C+x);

 printf("%x",(unsigned int)byt);
}

// Decimal 4 byte format
printf("\nDecimal (4bytes):");
for(int x=3;x>=0;x--)
{
 byt=*((unsigned char *)&C+x);
 printf("%d,",(unsigned int)byt);
}
//
while(getchar()!=10);
while(getchar()!=10);
}
```

```
Binary:01000000000000000000000000000000
Decimal:   2
Hex:40000
Decimal (4bytes):64,0,0,0,
```

Part 4: The aim of the next section is to show how it possible using assembly language to directly access CPU instructions that are not directly available when you use a higher level language.

```
#include "stdafx.h"
#include <stdio.h>

void test(void); /

int _tmain(int argc, _TCHAR* argv[])
{
    test();
    return 0;
}

void test()
{
    //
    union mmx_word {
        unsigned char byte[8];
        unsigned __int64 value;
    };


    mmx_word NUM1 = { 0,1,2,3,4,5,6,7 };
    mmx_word NUM2 = { 1,1,1,1,1,1,1,1 };

    __asm {
        //
        movq mm0, NUM1
        movq mm1, NUM2

        paddb mm0, mm1

        movq NUM1, mm0

        emms
    }


    printf("Result bytes: ")
    for (int i = 0; i < 8; i++) {
        printf("%u", (unsigned int)NUM1.byte[i]);
        if (i < 7) printf(",");
    }
    printf("\n");

    /
    while (getchar() != '\n');
    while (getchar() != '\n');
}
```

```
Result bytes: 1,2,3,4,5,6,7,8
```