
汇编语言上机大作业报告

一、设计题目

四则运算

从键盘输入一个简单的表达式，如“ $S=4+6*9-1+8/5$ ”，按回车键结束输入，则屏幕显示 $S=58.6$ ，小数点保留 1 位。假设输入的表达式中只含个位十进制数和“+”、“-”、“*”、“/”运算符，且同一运算符最多出现 2 次。

二、设计说明

1. 整个程序的功能应该能完成的工作

这个程序应该能正确处理数字和数学表达式的输入。我的设想是使其进一步处理最多 12 位十进制小数的输入，以及带有括号、四则运算算式的正确处理，并给出可以精确到小数点后五位的结果。

该程序应完成工作：

- 1) 公式的输入，包括处理数字输入、符号输入，以及正确处理输入公式的句法。
- 2) 公式的计算。其中包括正确处理各种符号运算的优先级和结合性、中间数的临时保存、小数的正确处理等。
- 3) 结果的正确显示。

2. 多个任务（子程序）的划分、调用关系

- 1) 十进制浮点数字的输入
- 2) 十进制浮点数字的输出
- 3) 符号的输入
- 4) 对于输入的公式的综合处理（转化为程序内部各数字/数据（称作 token）一定的存储格式）。调用 1) 与 3)
- 5) 对于处理后的公式的运算
- 6) 主程序。调用 4)，再调用 5)，最后调用 2)。

3. 各子程序的功能与联系

(1)处理十进制小数的转换：ID.ASM

包含过程：INPUTDECIMAL，负责将内存中的十进制小数字符串转换为双精度浮点数，仍然存在内存中。

输入参数包括字符串的起始位置 DS:SI，和输出位置 ES:BX。

(2)处理十进制小数的输出：OD.ASM

包含过程：OUTPUTDECIMAL，负责将内存中的双精度浮点数以十进制小数形式输出。

输入参数包括字符串的起始位置 DS:SI。

(3)处理算式：PARSEEXP.ASM

包含过程：PARSEEXP，负责将内存中的一串算式字符串进行解析运算，并将最终结果存在内存中。

也同时输出算式的逆波兰符号形式，方便调试找出错误。

会调用 ID.ASM、OD.ASM 中的功能。

输入参数包括字符串的起始位置 DS:SI 和输出位置 DS:BX。

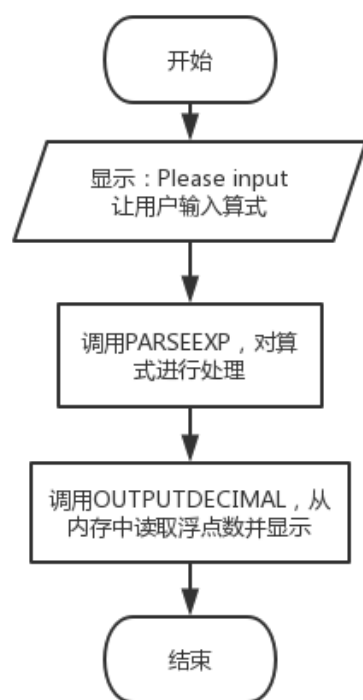
(4)主程序：MAIN.ASM

包含过程 MAIN，是程序的入口。

功能包括显示输入提示、让用户输入算式，以及调用 PARSEEXP 处理算式，最后用 OUTPUTDECIMAL 输出计算结果。

4. 程序框图

主程序的大致框图如下：



5. 子程序说明和流程图

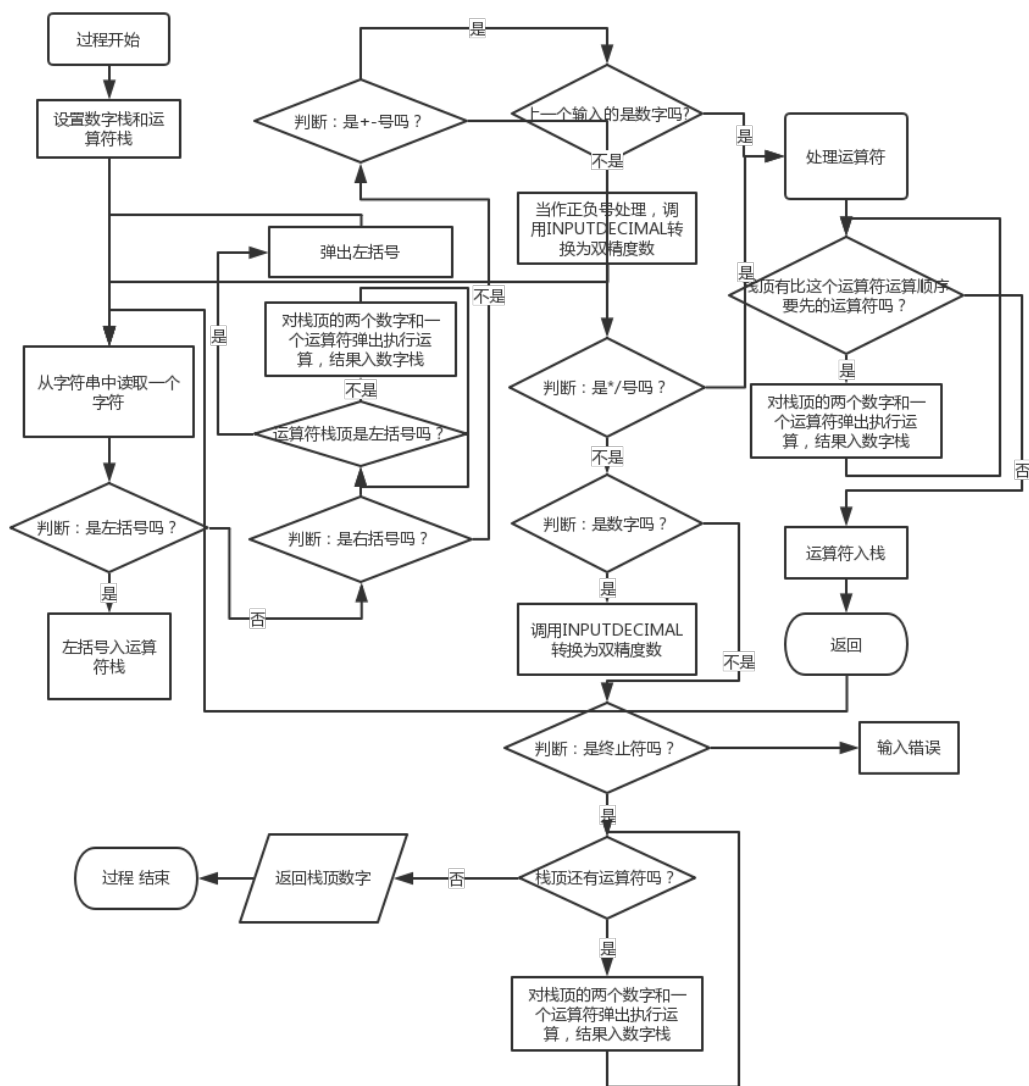
(1) PARSEEXP

逐字符进行读取，并根据读取到的字符判断算式中出现的 token 属于什么类型。若是数字，则调用 INPUTDECIMAL 将其处理成双精度浮点数；若是运算符，则对应处理（见下）。

对于算式的处理和运算，采用了调度场算法（Shunting yard algorithm）。采用两个堆栈，一个放数字，一个放运算符。当算法执行到将运算符放置到数字上的步骤时，立即进行运算。这样，分析完算式后，数字栈顶便是结果。

运算用 Intel 处理器的 FLD、FADD 等指令完成。

流程见图，省略了对部分错误状况的处理，详见源码。



(2) INPUTDECIMAL

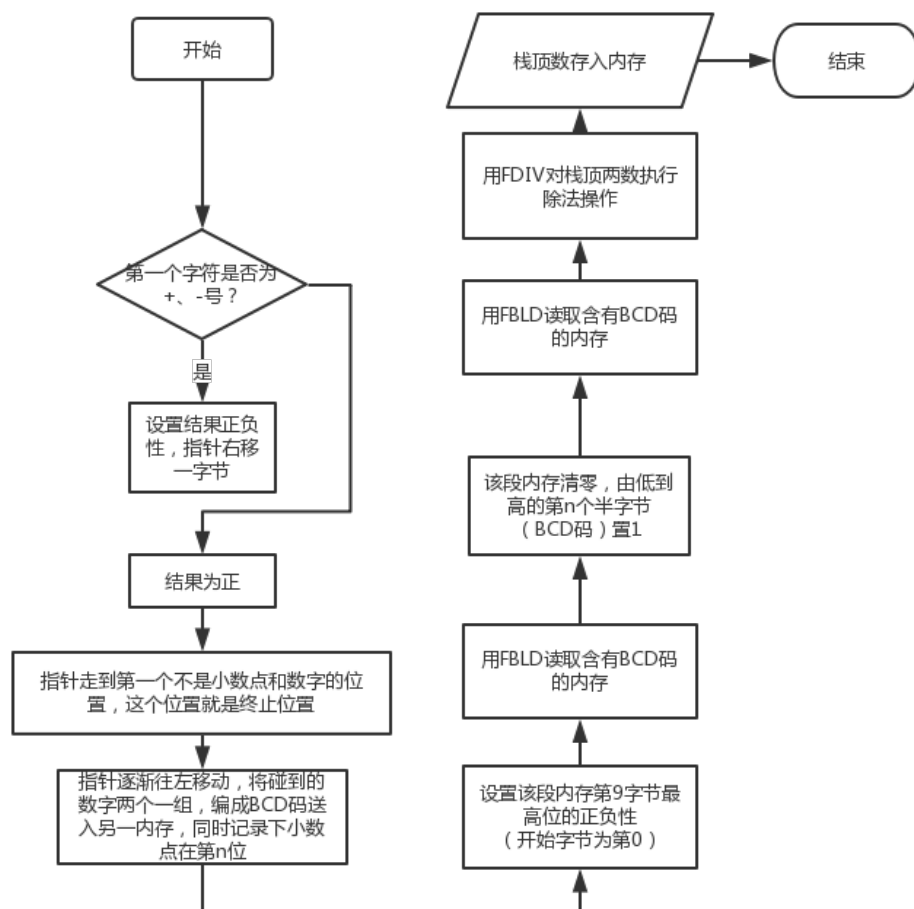
利用了英特尔处理器的 FBLD 指令，将十进制的小数转换为浮点数。

FBLD 指令可以将一段长度为 10 字节空间中的前 9 字节中的 18 位 BCD 码从十进制整数转换为 FPU 寄存器中的 REAL10 浮点数，最后一字节最高位决定符号位。

如一系列从低位到高位排列的字节：21 43 65 87 00 00 00 00 00 80 可以转换为-87654321.0，存储在 FPU 的栈顶。

对字符串倒序、去掉小数点转 BCD 码后，用 FBLD 转换为大浮点数；再根据小数点位置，用 FBLD 构造一个是 10 的若干次幂的浮点数，除前者，得到字符串表示的浮点数。再用 FSTP 将其作为双精度数存储于内存，该功能即得以实现。

大致框图见图。



(3) OUTPUTDECIMAL

利用的是 FBLD 的逆向指令 FBSTP。将双精度数用 FLD 载入 FPU 的寄存器后，乘以 100000，用 FBSTP 转换为包含小数点后五位的 BCD 数码，再从右往左解析输出，在第五位加小数点即可。

6. 程序清单

共四个源码文件：

- MAIN.ASM
- ID.ASM
- OD.ASM
- PARSEEXP.ASM

附件见下。

源码见附录（全文末端）



OD.ASM



MAIN.exe



MAIN.ASM



ID.ASM



PARSEEXP.ASM

三、 调试说明

1. 调试情况

本项目调试工作较重，一方面因为本人经验不足，一方面因为 16 位汇编本身难以调试的特性。在调试时，主要遇到以下问题：

(1)浮点数指令工作不正常

在用 FADD 等指令进行浮点运算时，发现加载的浮点数并无任何问题，但算出结果储存后是 FFF80000...(NaN)。最后发现，将各个子程序中分散的 FINIT 指令集中在 MAIN 程序中解决了此问题。可能 FINIT 执行多次会带来未定义的操作。

(2)在子程序中用地址表失效

在子程序 PARSEEXP 中，判断操作符类型时，采用了地址表。但无论是在 CS 段中定义还是在 DS 段中定义，地址表都无法正常工作。最后发现，地址表中的标号会在汇编时转换为偏移地址，但其对应的段地址和远调用时认定的子程序段地址并不一致，所以用地址表时会跳到不正确的地方造成 NTVDM 崩溃。最后，将地址表改为简单的条件跳转后问题解决。

其他问题皆是一些一眼能看出问题所在、由于粗心造成的小问题。

(3)程序调试技巧

在调试中我积累了一些经验：

- 程序源码的组织一定要经过深思熟虑，否则会在调试时造成更大麻烦；
- 善用 DEBUG 的一些指令以及其参数，如 -P+ 数字可以一次执行多个指令，对于跳跃到问题代码很有帮助。

2. 链接的要求

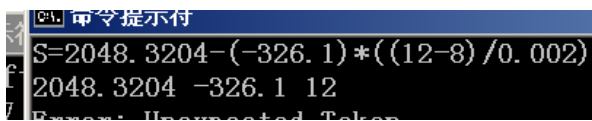
对四个文件分别汇编，生成 .OBJ 文件后，在控制台中输入 LINK

MAIN+PARSEEXP+ID+OD 回车，可以执行链接操作生成 MAIN.EXE。

3. 测试数据

```
Z:\INST\work-big>MAIN
Please input an expression.
S=3+4
3 4 +
Result: S = 7
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=3.14+90.52
3.14 90.52 +
Result: S = 93.66
```



```
命令提示符
S=2048.3204-(-326.1)*((12-8)/0.002)
2048.3204 -326.1 12
Result: Unrecognized Token
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=2048.3204-(-326.1)*((12-8))
2048.3204 -326.1 12 8 - * -
Result: S = 3352.7204
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=0
0
Result: S = 0
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=123456789012+111111111111
123456789012 111111111111 +
Result: S = 234567900123
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=0.00002+1/0.00002+50000
0.00002 1 0.00002 / + 50000 +
Result: S = 100000.00002
```

以下是各种输入错误的情形。

```
Z:\INST\work-big>MAIN
Please input an expression.
S=(

Error: Mismatched Parenthesis

Result: S = <Error>
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=

Result: S = <Error>

Z:\INST\work-big>
```

```
Result: S = <Error>

Z:\INST\work-big>MAIN
Please input an expression.
S=++9

Error: Incomplete Number or Unexpected Arrangement of +/

Result: S = <Error>

Z:\INST\work-big>_
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=3.14159+5*9)
3.14159 5 9 * +
Error: Mismatched Parenthesis

Result: S = <Error>

Z:\INST\work-big>_
```

```
Z:\INST\work-big>MAIN
Please input an expression.
S=3.14_
3.14
Error: Unexpected Token

Result: S = <Error>

Z:\INST\work-big>
```



```
Z:\INST\work-big>MAIN
Please input an expression.
S=6.284-9+
6.284 9 -
Error: Missing Operand Somewhere

Result: S = <Error>

Z:\INST\work-big>
```

4. 结果分析

就本人测试过的数据来看，正确输入的算式都得到了正确的结果。对于输入错误的算式而言，本程序并没有覆盖到太多情况，比如 $+.+6$ 判断为 $0+6=6$ 等。

综上所述，本程序基本达到了题目要求。

四、 使用说明

1. 程序环境、适用范围

本程序应运行于：

- Intel 的 16 位处理器（或模拟其工作的模拟器上），如 8086，并且应支持浮点运算功能；
- MS-DOS，或 Windows 7 及以下 x86 操作系统的命令提示符中（在 NTVDM 中运行）

本程序的测试环境为 DOSBox，以及 VMWare Workstation 下的 Windows 7 x86 虚拟机的 cmd.exe。

本程序适用于：

简单的、输入式的、包括 $+$ 、 $-$ 、 $*$ 、 $/$ 、 $()$ 的数学运算式，结果精度不超过 5 位小数。

2. 程序运行

(1)使用方法

- [1] 启动 MS-DOS 操作系统（或 DOSBox），或 Windows x86 系统的命令提示符（CMD.exe）。

-
- [2] 用 `cd` 命令和盘符命令，将当前路径定位到本程序可执行文件的目录下。
 - [3] 输入 `MAIN.EXE`，按回车键。
 - [4] 此时程序提示用户输入“`S=`”。用户应该输入一个算式，之后按回车键。
 - [5] 程序应能给出 `S` 的最终结果。

(2) 调试方法

- [1] 启动 MS-DOS 操作系统（或 DOSBox），或 Windows x86 系统的命令提示符（`CMD.exe`）。
- [2] 用 `cd` 命令和盘符命令，将当前路径定位到本程序可执行文件的目录下。
- [3] 输入 `DEBUG MAIN.EXE`。注意，操作系统的 `Path` 环境变量的条目路径下应能找到 16 位调试工具 `DEBUG.EXE`。
- [4] 使用命令 `R`、`P`、`T`、`D`、`G` 等，对程序进行调试。

(3) 输入信息的类型及格式

用户应输入一个长度小于等于 79 的字符串，其每个字符只能是 0 到 9、`+*/、()`、小数点、空格之一。输入的字符串应该是符合简单数学的格式要求的算式。

其中，输入数时，应注意数字的有效位数不得超过 12 位，得数的绝对值不应该超过 1.0×10^{12} ；不应出现除以 0 的情况。

(4) 出错信息的含义

- “Error: Mismatched Parenthesis” 表示你可能输入了一个没有匹配的括号。
- “Error: Missing Operand Somewhere” 表示有一个地方没有操作数，可能是在某两个符号之间没有输入数字。
- “Error: Standard Syntax Error (No Operator between two operands)” 发生的原因是两个数字挨在了一起。
- “Error: Unexpected Token” 表示用户输入了在要求限制之外的符号。
- “Error: Unknown Error” 是未知错误，可能有程序本身的缺陷产生。
- “Error: Incomplete Number or Unexpected Arrangement of +/-” 表示某个数字未输入完整，也有可能是`+`、`-`号输入语法错误。

五、 总结

1. 基本情况

(1)本人自学的内容和时间

- 7月25日-26日：浮点操作的基本方法
- 8月2日：浮点指令
- 8月14日-16日：宏

(2)参考书目

- 《汇编语言程序设计》，张光长
- 《Intel 汇编语言程序设计（第五版）》，电子工业出版社，[美] Kip R.Irvine
- 《Intel® 64 and IA-32 Architectures Software Developer's Manual》，VOL 1, VOL 2A

(3)上机

在家里完成上机设计。在7月25日进行初步设想和试验程序设计，在8月13日-16日完成全部程序设计。

本程序的所有部分均为我独立完成。读取、显示小数的算法来源于我自己的思考，解析表达式的方法来源于维基百科“调度场算法”词条。

2. 心得体会

本次大作业花了我很大的心思。无论是上课的过程、知识的积累、资料的搜集、真正编写、调试一个大工程的过程，都让我有了提高。

首先，我了解了汇编这门语言。它与机器码有着直接、密不可分的关系，是软件层面上一个基层。没有它，就没有各种高级语言构建的美妙代码。在了解计算机的任何一个层次时，除了本层次的内容，还要稍微了解构成这个层次的下一层次内容。这样，才能更好的理解本层次的知识来源及其所以然。所以，了解汇编，学习汇编，是重要的。

汇编让我知道了计算机软件的大厦是如何构建起来的，也锻炼了我的编程能力。我进一步地用汇编的思维，全面地考虑、简化问题，破解难题；思考在系统的底层层面中，诸如字符串的分析、数据的存储应该如何实现。要是不操作内存的话，就只有有限的十六位寄存器可以用。空间如何分配？代码如何简约？弄不好的话，不但代码可读性降低了，在调试时更是会让人抓狂。所以，

要给这方面以足够的重视。我想在任何一门编程语言中，都是这样的。代码简约、构造美观、可靠是重要的。

3. 存在的问题

由于学习汇编的时间不多，我对于很多已学过的指令，不能形成长期记忆；另外，诸如 32 位汇编、宏、Windows API 等更多的汇编知识，我没有进一步的了解，只能通过现学的知识来构造程序，这是一个不足之处；

同时，我在写本次大作业时，由于匆匆忙忙，对于一些小细节没有把握好，比如之前提到的错误覆盖不够全，没有处理浮点异常如除以零的情况，某些地方的代码不够精炼等。

改进方法包括完善一些小细节，优化代码；同时，可以加入更多的运算符，甚至加入定义函数、使用函数的功能。

程序清单

文件名称	MAIN.ASM
子过程名称	MAIN
描述	主要过程，程序入口，负责输入输出及运算表达式子过程调用

```
COUNT      = 80D

KEYBOARD    STRUCT
    maxInput    BYTE    COUNT
    inputCount  BYTE    ?
    buffer      BYTE    COUNT DUP (?)
KEYBOARD    ENDS

STACK       SEGMENT      STACK
            BYTE          80H    DUP    (?)
STACK       ENDS

DATA        SEGMENT

expData     KEYBOARD     <>
pmtinstr    DB           "Please input an expression.",0DH, 0AH,
"S=$"
pmtsstr     DB           "Result: S = $"
result      DQ           ?
errorstr    DB           "<Error>$"
DATA        ENDS

CODE        SEGMENT
            ASSUME CS:CODE, DS:DATA
            EXTERN PARSEEXP:FAR, INPUTDECIMAL:FAR,
OUTPUTDECIMAL:FAR

mWriteStr   MACRO        buffer
                PUSH      DX
                PUSH      AX
                MOV       DX, Offset buffer
                MOV       AH, 09H
                INT       21H
                POP       AX
                POP       DX
            ENDM

mWriteChr   MACRO        chr
                PUSH      DX
                PUSH      AX
```

```

                MOV     DL, chr
                MOV     AH, 02H
                INT     21H
                POP     AX
                POP     DX
ENDM

MAIN          PROC     FAR
                PUSH    DS
                MOV     AX, 0H
                PUSH    AX

                MOV     AX, DATA
                MOV     DS, AX

                FINIT

                mWriteStr  pmtinstr

                MOV     AH, 0AH
                MOV     DX, Offset expData
                INT     21H

                mWriteChr  0DH
                mWriteChr  0AH

                MOV     BX, Offset expData.buffer
                MOV     AL, expData.inputCount
                CBW
                ADD     BX, AX
                MOV     Byte Ptr[BX], '$'

                MOV     SI, Offset expData.buffer
                MOV     BX, Offset result
                CALL    PARSEEXP

                mWriteStr  pmtsstr

                MOV     SI, BX

                CMP     Word Ptr[BX], 0FFFFH
                JNZ     NO_ERROR
                ADD     BX, TYPE Word
                CMP     Word Ptr[BX], 0FFFFH
                JNZ     NO_ERROR
                ADD     BX, TYPE Word
                CMP     Word Ptr[BX], 0FFFFH
                JNZ     NO_ERROR
                ADD     BX, TYPE Word

```

```

                CMP        Word Ptr[BX], 0FFFFH
                JNZ        NO_ERROR
                JMP        ERROR_OUT

NO_ERROR:
                CALL       OUTPUTDECIMAL
                JMP        FINAL

ERROR_OUT:
                mWriteStr   errorstr

FINAL:
                mWriteChr   0dh
                mWriteChr   0ah

                RETF

MAIN          ENDP
CODE          ENDS
END           MAIN

```



```

;两次
;对与 SP 增长方向相反的 BP 定义压栈、弹出的
宏
    PUSHBP      MACRO      STH
                ADD        BP, TYPE STH
                MOV        [BP], STH
    ENDM
    POPBP       MACRO      STH
                MOV        STH, [BP]
                SUB        BP, TYPE STH
    ENDM
    PUSHBPFPST  MACRO      PUSH    BX      ;将之前备份的返回地址再次压入，之后的 EVAL_ONCE 过程不会改变栈。
                ADD
    QWord
                FST        QWord Ptr
    SS:[BP]
                CALL      EVAL_ONCE
                ENDM
                ;POP      BX
    PUSHBPFPST  MACRO      ;PUSH    BX
                ADD
                BP, TYPE
    QWord
                FSTP       QWord Ptr
    SS:[BP]
                EVAL_ADD:
                ENDM
                IF PAEX_DEBUG
                POPBPFLD  MACRO      FADD
                FLD        QWord Ptr
                OUTCHAR    '+'
    SS:[BP]
                ENDIF
                JMP AFTER_EVAL
    Qword
                SUB        BP, TYPE
                EVAL_SUB:
                ENDM
                IF PAEX_DEBUG
                OUTCHAR    '-'
                ENDIF
                JMP AFTER_EVAL
                ;输出字符
    OUTCHAR      MACRO      CHR
                PUSH      DX
                PUSH      AX
                MOV        DL, CHR
                MOV        AH, 02H
                INT        21H
                POP        AX
                POP        DX
    ENDM
                MOV        CX,
0H                ;CX 在本过程中作为一个全局量，指示刚刚读取的前一个 TOKEN 是否为一个数字。
                ;这样就可以为是否存在“1 2”
                这样堆叠数字的错误，和“+、-”到底表示加号减号
                还是正负号提供依据。
                JMP
READ_TOKEN_LOOP
                ;弹出栈顶的两个数和一个操作符，执行浮点运算，运算压入数栈。若开启了 DEBUG，按逆波兰符号法输出符号
                EVAL_STACKTOP:
                POP        BX      ;因为要用到压在返回地址下面的 AX，所以先弹出返回地址，存在 BX 里面。
                POP        AX
                压入栈
                REPEAT    2
                RETN

```

			AFTER_UNTIL_LBRA:	
READ_TOKEN_LOOP:		;主过程。不断移	XOR	CX, CX
动指针 SI, 检测每一个字节表示的符号, 采取操作			ADD	SP, TYPE Word
	MOV	DL, [SI]	JMP	
	INC	SI	READ_TOKEN_LOOP	
	CMP	DL, ' '	GET_PLUSSIGN:	
READ_TOKEN_LOOP	JE			;检测到+号时, 检测下 CX, 判断到
	CMP	DL, '('		底是按正号(跳到 INPUTDECIMAL) 还是加号处理
	JE	GET_LBRA	OR	CX, CX
	CMP	DL, ')'	JZ	
	JE	GET_RBRA	GET_FLOAT_SIGNED	;之前没有输入
	CMP	DL, '+'		过数字, 所以+当正号处理
	JE	GET_PLUSSIGN		
	CMP	DL, '-'		;否则当加号处理
GET_MINUSSIGN	JE		JMP	GET_ADDOP
	CMP	DL, '*'	GET_ADDOP:	
	JE	GET_MULOP	MOV	DX,
	CMP	DL, '/'	OP_ADD	;加号和减号优先级
	JE	GET_DIVOP		相同, 故某些部分重合。DX 指示当前运算的是加号
	CMP	DL, '.'		还是减号
GET_FLOAT_SIGNED	JE		JMP	
	CMP	DL, '\$'	START_ADDSUBOP	
	JE	GET_END_SIGN	GET_SUBOP:	
	CMP	DL, '0'	MOV	DX, OP_SUB
	JB		START_ADDSUBOP:	
ERR_INPUT_UNEX_TOKEN	CMP	DL, '9'	CMP	SP,
	JA		SP_BOTTOM	;栈里没东西的话, 直
ERR_INPUT_UNEX_TOKEN	JMP			接将+号进栈
GET_FLOAT_NOSIGN			JE	
			PUSH_OPSTACK_ADDSUB	
GET_LBRA:		;检测到(时, 压	MOV	BX,
入栈即可。	MOV	AX, OP_LBRA	SP	;否则先看一下栈顶
	PUSH	AX		的运算符, 如果优先级(或同优先级的左结合性先算)
	XOR	CX, CX		高于+号, 先计算。
READ_TOKEN_LOOP	JMP		MOV	AX,
			SS:[BX]	;当然, 因为只有
				+-*/, 并且都是左结合, 所以不是碰到(的话都会先
GET_RBRA:		;检测到)时, 不		计算的。
断执行操作符栈顶的操作符, 将括号内的得数算出			SHR	AX, 1H
来, 直到碰到(。之后弹出左括号。			CMP	AX,
LOOP_OP_UNTIL_LBRA?:				;第二优先级
	CMP	SP, SP_BOTTOM		
	JE			
ERR_INPUT_MISMAT_PAR	MOV	BX, SP	EVAL_STACKTOP_ADDSUB	
	MOV	DX, SS:[BX]	CMP	AX,
	CMP	DX, OP_LBRA		;第一优先级
	JE			
AFTER_UNTIL_LBRA			JE	
	CALL		EVAL_STACKTOP_ADDSUB	
EVAL_STACKTOP			CMP	AX,
	JMP			;左括号
LOOP_OP_UNTIL_LBRA?			JE	
			PUSH_OPSTACK_ADDSUB	

JMP		CALL	
ERR_INPUT_UNKNOWN_1	;循环直到可以	EVAL_STACKTOP	
将+号进栈即可。		JMP	
PUSH_OPSTACK_ADDSUB:		START_MULDIVOP	
PUSH DX		GET_FLOAT_SIGNED:	
XOR CX, CX		MOV DX,	
JMP		SI	;若是 SIGNED, 那么
READ_TOKEN_LOOP		为了防止只出现一个+号或.号后面没有数字的情况	
EVAL_STACKTOP_ADDSUB:		JMP	
CALL		GET_FLOAT	;需要将 SI 复制
EVAL_STACKTOP		一份, 然后比对读取数据之后的 DX 和 SI。若 SI 没	
JMP		有走动, 则判定为输入数字错误。	
START_ADDSUBOP		GET_FLOAT_NOSIGN:	
GET_MINUSSIGN:		MOV DX, SI	
OR CX, CX		DEC	
JZ		DX	;由于在主过程
GET_FLOAT_SIGNED		时, 读取后 SI 马上加一, 此时复制到的 DX 是往右	
JMP GET_SUBOP		偏移一位的, 如果是没带符号的数字的话要修正过来	
GET_MULOP:		GET_FLOAT:	
;乘除和加减基本相同, 只是优先级		OR CX, CX	
不同而已。		JNZ	
MOV DX, OP_MUL		ERR_INPUT_SYNTAX_ERROR	
JMP		DEC	
START_MULDIVOP		SI	;同上, 修正过来
GET_DIVOP:		ADD BP, TYPE	
MOV DX, OP_DIV		Qword	
START_MULDIVOP:		MOV BX, BP	
CMP SP, SP_BOTTOM		PUSH ES	
JE		MOV AX, SS	
PUSH_OPSTACK_MULDIV		MOV ES, AX	
MOV BX, SP		CALL	
MOV AX, SS: [BX]		INPUTDECIMAL	;备份、设置相应
SHR AX, 1H		的段寄存器, 调用输入数字的过程	
CMP AX,		POP ES	
2	;第二优先级	NOT CX	
*/, 对应 4 和 5		CMP DX, SI	
JE		JAE	
EVAL_STACKTOP_MULDIV		ERR_INPUT_INCOMPLETE_NUMBER	
CMP AX,		IF PAEX_DEBUG	
1	;第一优先级	PUSH DS	
+-, 对应 2 和 3		PUSH SI	
JE		PUSH AX	
PUSH_OPSTACK_MULDIV	;与加减不	MOV AX, SS	
同, 此处可以直接将乘除号压栈。		MOV DS, AX	
CMP AX,		MOV SI, BP	
0	;左括号	CALL	
JE		OUTPUTDECIMAL	;按逆波兰符号
PUSH_OPSTACK_MULDIV		法输出数字	
JMP		POP AX	
ERR_INPUT_UNKNOWN_2		POP SI	
PUSH_OPSTACK_MULDIV:		POP DS	
PUSH DX		OUTCHAR ' '	
XOR CX, CX		ENDIF	
JMP			
READ_TOKEN_LOOP			
EVAL_STACKTOP_MULDIV:			

```

                                OR      DL, DL
                                JNZ     MOV_FF
READ_TOKEN_LOOP                REP     MOVSB
                                JMP     AFTER_MOV
                                REP     STOSB

                                MOV_FF:
                                AFTER_MOV:
                                POP      ES
                                POP      DS

                                GET_END_SIGN:
                                XOR      DL,
DL                                ;获取到 endsign $后, 设置一下 DL
表示是否出现错误
                                GET_END_SIGN_LOOP:
                                CMP      SP,
SP_BOTTOM                      ;要执行操作符栈存留的所有运算
                                JE       END_PARSING
                                MOV      BX, SP
                                MOV      AX, SS: [BX]
                                CMP      AX,
OP_LBRA                        ;若碰到左括号, 说明有括号没有闭
合。
                                JE
ERR_INPUT_MISMAT_PAR           ERR_INPUT_SHOW      MACRO
ERR_STR                        ;定义错误宏。错误时统一将 DL 设置
成 FF 返回
                                ERR_INPUT_&ERR_STR :
                                IF PAEX_DEBUG
                                OUTCHAR  0DH
                                OUTCHAR  0AH
                                ENDIF
                                PUSH     DS
                                MOV      AX, PAEX_DATA
                                MOV      DS, AX
                                MOV      DX, Offset
ERR_INPUT_STR_&ERR_STR        MOV      AH, 09H
                                INT      21H
                                POP      DS
                                MOV      DL,
0FFH                            ;表示出现错误
                                JMP      END_PARSING
                                ENDM

                                ERR_INPUT_SHOW      MISMAT_PAR
                                ERR_INPUT_SHOW      NO_OPERAND
                                ERR_INPUT_SHOW      SYNTAX_ERROR
                                ERR_INPUT_SHOW      UNEX_TOKEN
                                ERR_INPUT_SHOW      UNKNOWN_1
                                ERR_INPUT_SHOW      UNKNOWN_2
                                ERR_INPUT_SHOW      UNKNOWN_3
                                INCOMPLETE_NUMBER

                                PARSEEXP      ENDP
                                PAEX_CODE    ENDS
                                END

                                MOV      DS, AX
                                MOV      SI, BP
                                MOV      DI,
BX                                ;此时可以 POP 出目标地址
                                PUSH     DS
                                PUSH     ES
                                MOV      AX, DS
                                MOV      ES, AX
                                MOV      AX,
PAEX_STACK
                                MOV      DS, AX
                                MOV      SI, BP
                                MOV      DI,
BX                                ;将原数字栈的栈顶一个字节一
个字节地 MOVSB 到目标地址, 共 8 次
                                MOV      CX, 8H
                                MOV      AL,
0FFH                            ;若出现异常 (DL 非 0), 返回
0FFFFFFFFFFFFFFFF

```

文件名称	ID.ASM
子过程名称	INPUTDECIMAL
描述	负责将十进制小数字符串转换为双精度

```

PUBLIC      INPUTDECIMAL                                PUSH    DX

DECM_DATA  SEGMENT                                     PUSH    DI
    BCDSPC  DB      10 DUP (?)                         PUSHF
    DEBUGSAVE DQ    ?                                PUSH    ES
    DEBUGSAVE2 DQ   ?                                PUSH    DS
    DEBUGGING_ =    1                                PUSH    BX
    ENDOFSTR  DW    ?
DECM_DATA  ENDS

DECM_CODE   SEGMENT

ISTHISDIGITVALID:
;判断这个位是否还是该小数的内容。判断依据：
; 小数只能出现一次小数点(1),也有可能不出现(0),
记录在 AH 中
; 存储小数的字符在内存中的起始位置,记录在 BX
中
;返回 DL, 指示该位是否是 decimals 的内容; 返回 AH, 指
示小数点是否被用过
;返回 SI, 指示读取过小数之后的终止位置的下一个
字节
    MOV     DL, [BX]
    CMP     DL, '0'
    JAE     DIGITNEXT
DIGITNOTNUM:
    CMP     DL, '.'
    JE      DIGITDOT
INVALID:MOV     DL, 0H
    RET
DIGITDOT:
;若是第一次碰到点, CX+1, 因为这个
点不算数字, 在循环中不计
    INC     CX
    OR      AH, AH
    JNZ     INVALID
    MOV     AH, 1H
    JMP     VALID

DIGITNEXT:
    CMP     DL, '9'
    JA      DIGITNOTNUM
VALID:  MOV     DL, 1H
    RET

INPUTDECIMAL PROC FAR
;输入: DS:SI 字符串首地址; ES:BX 64 位浮点数的目
标内存首地址
;输出: 在 ES:BX 输出目标浮点数
    ASSUME     DS:DECM_DATA,
CS:DECM_CODE
    PUSH     AX
    PUSH     BX
    PUSH     CX
    MOV     AX, DECM_DATA ;先将 DS
和 ES 设置成本过程自用的数据段
    MOV     ES, AX
    MOV     BX, SI        ; 判断是
否有符号 "+/-", 并根据情况将 DH 置 1(NEGATIVE)
或 0
    MOV     DL, [BX]
    CMP     DL, '+'
    JE      SIGNPOSITIVE
    CMP     DL, '-'
    JE      SIGNNEGATIVE
    MOV     DH, 0H
    PUSH    DX
    JMP     GOTOENDOFSTR

SIGNPOSITIVE:
    MOV     DH, 0H
    PUSH    DX
    INC     BX
    INC     SI
    JMP     GOTOENDOFSTR

SIGNNEGATIVE:
    MOV     DH, 1H
    PUSH    DX
    INC     BX
    INC     SI

; 指针走到最后一个符号
GOTOENDOFSTR:
    MOV     CX, 18D
    XOR     AH, AH
GOTOENDOFSTR_LOOP:
    CALL    ISTHISDIGITVALID
    OR      DL, DL
    JZ      HEREISENDOFSTR
    INC     BX
    LOOP    GOTOENDOFSTR_LOOP

HEREISENDOFSTR:
    PUSH    DS
    MOV     AX, DECM_DATA
    MOV     DS, AX
    MOV     ENDOFSTR, BX
    POP     DS

```

[illegible]

```
MOV    DS, AX
MOV    SI, ENDOFSTR
POP    DS

POPF
POP    DI

POP    DX
POP    CX
POP    BX
POP    AX

RET
INPUTDECIMAL ENDP
DECM_CODE    ENDS
END
```

文件名称	OD.ASM
子过程名称	OUTPUTDECIMAL
描述	负责输出双精度小数

```

PUBLIC          OUTPUTDECIMAL                                ;开始找 BCD 的最高位、最低位

OUTD_DATA      SEGMENT                                     有效数字
    PRECISION   EQU      6      ;一定要是      BCDSPC      MOV      BX, Offset
偶数 MUST BE EVEN
    BIGDECM     DQ      1000000.0
    BCDSPC      DT      ?
    MOSTSIG     DB      ?
    LESTSIG     DB      ?
OUTD_DATA      ENDS
OUTD_CODE      SEGMENT
    ASSUME CS:OUTD_CODE,
DS:OUTD_DATA

    OUTPUTDECIMAL PROC FAR
        ;输出一个小数。
        ;输入: SI, 64bit 小数的偏移
        地址。
        ;正常输出范围:
        -100000~-0.00000001,
        0.00000001~100000
        ;输出格式: 最多 8 位小数, 如
        25.00152225, 33.441.
        PUSH     AX
        PUSH     BX
        PUSH     CX
        PUSH     DX
        PUSHF
        PUSH     DS
        FLD      QWord Ptr
[SI]
        MOV      AX, OUTD_DATA
        MOV      DS, AX
        FMUL     QWord Ptr
BIGDECM        ;乘一百万, 就可以先化成整
数
        ;此时 MOSTSIG 和 LESTSIG 分别储存小数
        FBSTP    BCDSPC      ; 的最高位和最低位
        计算得到 BCD
        AFTERSETSIG:
            CMP      MOSTSIG, 6D
            JAE
        AFTER_CORRECT_MOSTSIG

```

```

        MOV        MOSTSIG, 6D
AFTER_CORRECT_MOSTSIG:
        CMP        LESTSIG, 6D
        JBE
AFTER_CORRECT_LETSIG
        MOV        LESTSIG, 6D
AFTER_CORRECT_LETSIG:
;开始打印
PRINTDIGIT MACRO
digit
        PUSH      DX
        MOV        DL, digit
        AND        DL, 0FH
        OR         DL, 30H
        MOV        AH, 02H
        INT        21H
        POP        DX
ENDM
;先打印符号
        MOV        DL, 9[BX]
        TEST       DL, 80H
        JZ         PRINTDIGITS
        PUSH      DX
        MOV        DL, '-'
        MOV        AH, 02H
        INT        21H
        POP        DX
PRINTDIGITS:
        XOR        CX, CX
        MOV        CL, MOSTSIG
        TEST       CL, 1H
        JNZ
PRINTDIGIT_ODD
        ROR        CX, 1H
        MOV        DI, CX
        ROL        CX, 1H
        AND        DI, 0FH
        MOV        DL, [BX][DI]
        JMP
PRINTDIGIT_ODD
PRINTDIGIT_ODD:
        PUSH      CX
        SHR        CX, 1H
        MOV        DI, CX
        AND        DI, 0FH
        MOV        DL, [BX][DI]
        MOV        CL, 4H
        ROR        DL, CL
        POP        CX
        CMP        CL, PRECISION
        JNZ
DIRECT_PRINTDIGIT
        PUSH      DX
        MOV        DL, '.'
        MOV        AH, 02H
        INT        21H
        POP        DX
DIRECT_PRINTDIGIT:
        PRINTDIGIT DL
        PUSH      CX
        MOV        CL, 4H
        ROR        DL, CL
        POP        CX
        DEC        CL
        CMP        CL, LESTSIG
        JL
PRINTDIGIT_END
PRINTDIGIT_EVEN:
        PRINTDIGIT DL
        DEC        CL
        CMP        CL, LESTSIG
        JNL
PRINTDIGIT_ODD
PRINTDIGIT_ODD:
        POP        DS
        POPF
        POP        DX
        POP        CX
        POP        BX
        POP        AX
        RET
OUTPUTDECIMAL ENDP
OUTD_CODE ENDS
END

```