XXXX 大学

课程设计

课　　程：微机原理与接口技术设计

专业班级：　XXXXXXXXXX

学　　号：　XXXXXXXXXX

姓　　名：　　XX

目录

一、 设计题目及要求

1. 通过小键盘做加、减、乘、括号运算。数码管显示输入数据和计算结果数据。
2. 按键规定：
   （1） 数字用小键盘 0~9 输入。
   （2） 功能按键设定：
   “A”——“+”
   “D”——“-”
   “B”——“*”
   “F”——“括号”
   “E”——“=”
   “C”——输入过程中撤消当前运算，此时最右侧数码管显示“0”。
3. 其它要求：
   （1） 输入计算数据时，数码管应跟随显示。若超出显示范围则不响应超出部分。
   （2） 按“+”、“-”、“*”或“括号”对应按键时，数码管当前显示内容不变。
   （3） 按“ =”时，显示器显示最终计算结果。若结果超出显示范围时，则最右侧三个数码管显示“ ERR”。
   （4） 按“C”时，最右侧数码管显示“0”。
   （5） 需要考虑运算的优先级问题。
   （6） 只考虑正整数运算，不考虑负数和实数运算。

二、 设计思想

本实验完成了课程设计题目的所有要求，可以进行 32 位加减法以及 16 位乘法带括号的的混合四则运算（计算结果不大于“99999999”）。初步可分为硬件实现与软件算法实现两方面任务。

硬件方面，参照《微机原理》与接口技术实验指导书上对键盘的按键识别扫描可以实现将所按下的键值显示在数码管上，并将对应键值的 ASCII 码存入内存缓冲区 myinstr 中供算法计算。对应的 ASCII 码如下：

0-9	30H-39H
A(“+”)	2BH
B(“*”)	2AH
D(“-”)	2DH
F(“(”)	28H
“)”（非输入，为后续转换）	29H
E(“=”)	24H

使用 ASCII 码的目的是为了方便后续识别数字和运算符，以及括号、优先级的判断。

工具箱的连线如下：

D3 区：CS、A0、A1	——	A3 区：CS1、A0、A1
D3 区：PC0、PC1	——	F5 区：KL1、KL2
D3 区：JP20(PB 口)、JP16(B)、JP17(C)	——	F5 区：A、B、C

软件算法封装在 cclt 子程序中，计算的步骤主要如下：

（1） 将输入的括号转换成对应的左括号和右括号，并进行表达式的正误判断。由于键盘按下的“F”表示括号，因此现需要判断出表达式中相应的右括号。通过 update 子程序进行识别右括号，具体的方法是从头遍历一遍输入表达式（如“（5+3）*（（5-4）*（3+2））=”），若当前遍历到的字符是数字，判断其下一位是否是括号，若是则将数字后接着的所有括号变为右括号，直到识别到一个非括号字符。在完成括号的转换后进行对表达式的正误判断；

（2） 为了便于拓展成 32 位加减法以及 16 位乘法的运算，将数据格式拓展为双字，重新将运算符进行编码（999999999D=05F5E0FFH）：

运算符	存放数值
+	05F60000H
*	05F80000H
-	05F70000H
(	05FA0000H
)	05FB0000H
=	05F90000H

（3） 将输入的表达式转换成逆波兰式（trans 子程序）。具体算法如下：

a) 初始化一个空堆栈 track，将结果字符串变量置空；
b) 从左到右读入表达式，每次一个字符；
c) 如果字符是操作数，将它添加到结果字符串 exp；
d) 如果字符是操作符，弹出操作符，直至遇见左括号、优先级较低的操作符或者同一优先级的右结合符。把这个操作符压入堆栈；
e) 如果字符是个左括号，把它压入堆栈；
f) 如果字符是个右括号，在遇见左括号前，弹出所有操作符，然后把它们添加到结果字符串；
g) 如果到达输入字符串的末尾，弹出所有操作符并添加到结果字符串。

（4） 逆波兰式表达式求值（value 子程序）。在逆波兰式中，不需要括号，而且操作符的优先级也不再起作用。可用如下算法对后缀表达式求值：

a) 初始化一个空堆栈 track；
b) 从左到右读入逆波兰表达式 exp；
c) 如果字符是一个操作数，把它压入堆栈；
d) 如果字符是个操作符，弹出两个操作数，执行对应计算操作，然后把结果压入堆栈；

e) 到逆波兰表达式末尾，从堆栈中弹出结果。

（5） 与硬件的接口（print 子程序）。将计算完成的结果重新转化为 ASCII 码供硬件部分识别并显示。

注：加减法溢出、表达式错误等会对对应地调用 error 子程序，具体判断方法不再赘述。

三、 功能流程图

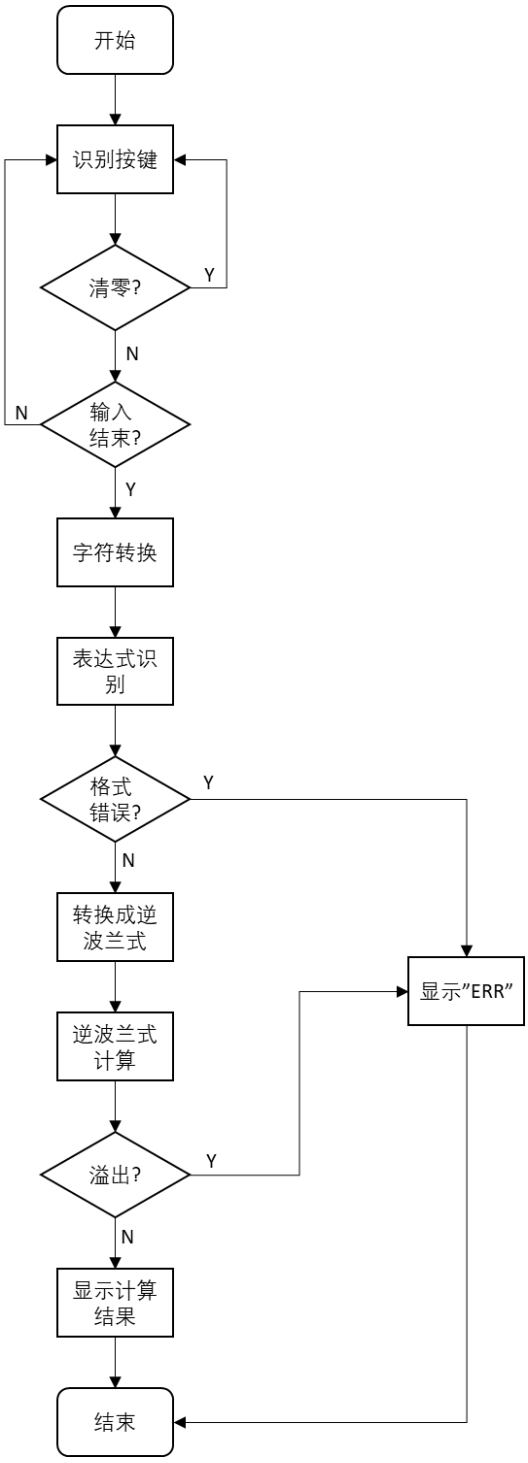

图 3-1 程序流程图

四、 结果讨论

本次课程设计为模拟简单计算器，可以进行 32 位加减法以及 16 位乘法带括号的的混合四则运算。需要对输入字符串进行处理，首先转换为相应的能被识别的正确的表达式，然后，对表达式进行有效处理，使其由原来的中缀表达式转换为便于计算机计算的逆波兰表达式。在进行计算的过程中，利用逆波兰表达式和堆栈结果是非常容易进行四则混合运算的。但是，在设计的过程中，如何将中缀表达式转换为逆波兰表达式，是本实验的一个难点。在中缀式向逆波兰转换的过程中，用到了堆栈结构。得益于先前数据结构课程的学习，成功地利用堆栈算法完成了表达式的转换。

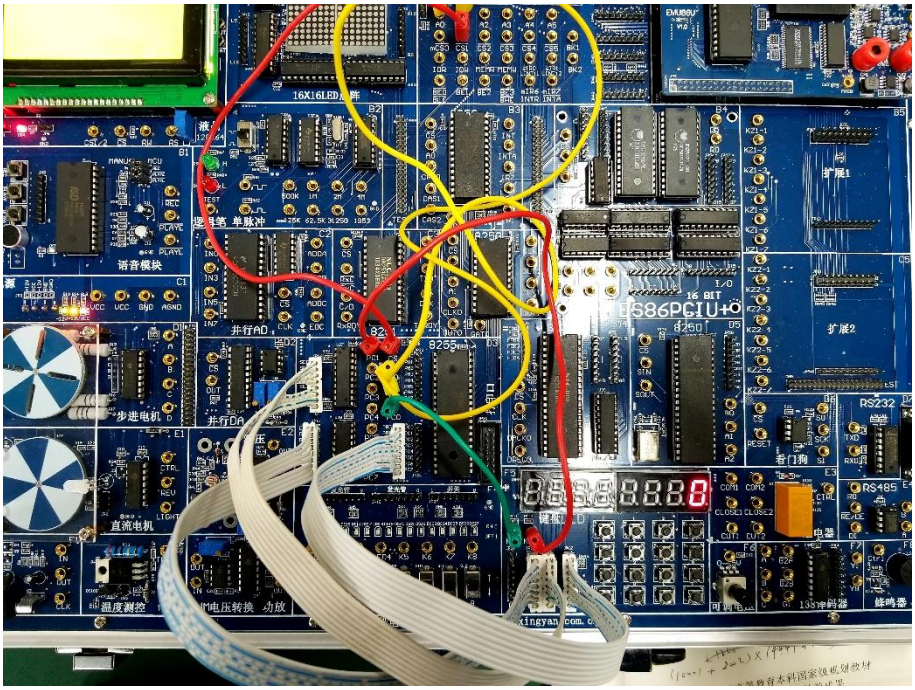

图 4-1 初始化

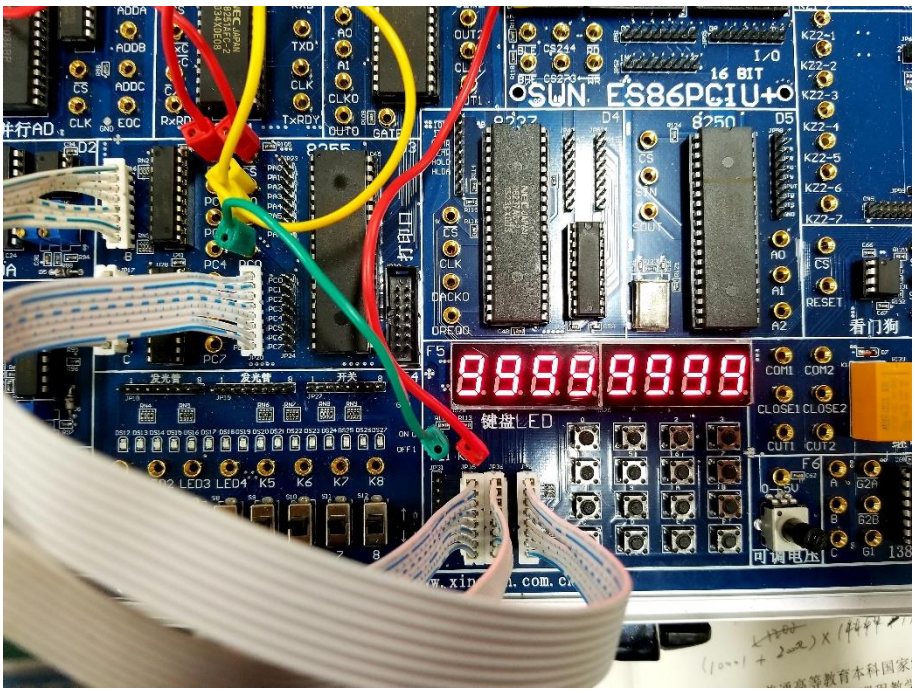

图 4-2 计算(4444-1111)*(30000+3)=99999999

通过本次微机原理课程设计，我对汇编语言的基本知识的使用更加熟练，同时也增加了我对 8086 处理器和 8255A 芯片的一些认识，在课设完成过程中通过和同学的交流，也增加了合作的技巧，通过查阅资料也学到了一些课本上没有的东西，拓宽了自己的知识面，全面地锻炼了自己。

五、 数字签名

六、 附录：实验代码：（完整的源程序）

```
stack1 segment para stack
    dw 512 dup(0)
stack1 ends

data segment
    ; store the input expression
    myinstr db 50 dup('$')
    ; store the checked expression
    stri dw 50 dup(0)
    ; store the rpn expression
    exp dw 50 dup(0)
```

```
    ; auxiliary stack
    track dw 50 dup(0)
    ; store the output
    result1 db 8 dup('$')
    result db 50 dup('$')
    ; 8 bytes to display the buffer
    buffer db 30 dup(?)
    ; common anode
    seg_tab  db 0c0h,0f9h,0a4h,0b0h, 99h, 92h, 82h,0f8h
             db 080h, 90h, 88h, 83h,0c6h,0a1h, 86h, 8eh,0ffh,0afh,0a3h
    ; 8255a ports
    com_8255 equ 0273h
    pa_8255 equ 0270h
    pb_8255 equ 0271h
    pc_8255 equ 0272h
data ends

code segment
     assume cs:code,ss:stack1,ds:data
start: mov ax,stack1
           mov ss,ax
           mov ax,data
           mov ds,ax
           mov es,ax
           cld
           mov dx,com_8255
           ; 10001001b
           mov al,89h
           ; output: pa, pb input pc
           out dx,al
           call clear
           call init
         call dir

main0:     call keyi          ;buffer 区第一位输入，不需要左移
         cmp al, 09h
         ja   n0
           call init
```

```
            lea di, buffer
            ;add di,7
            ;std
            stosb
            ;cld
            call input
            call dir


main1:      call keyi           ;返回后，寄存器 al 内为键值 0~f
            cmp al, 09h
            ja  n0

            ; digit key
            ; left shift


          lea di, buffer
            mov si, di
            add di, 08h
            add si, 07h
         std
            mov cx,8
         rep movsb
            cld
         lea di, buffer
            stosb

            ;left
            ; std
            ; stosb
            ; cld



            call input
            call dir
            jmp main1
```

```
          ; function key
n0:       cmp al,0ch
          ; clear
          jz l1
          cmp al, 0eh
          ; euqal
          jz l2
         call input
          jmp main0
l1:       call clear
         call init
         call dir
         jmp main0
l2:    call outv
         jmp main0


dir proc near
          push ax
          push bx
          push dx
          ; set the display buffer initial value
          lea si,buffer
          mov ah,0feh
          lea bx,seg_tab
ld0:   mov dx,pa_8255
          lodsb
          cmp al, 24h
          jnz n4
          mov al,10h
n4:       cmp al, 13h
         jb n1
         sub  al, 30h
          ; get the display value
n1:    xlat
          ; segment data -> 8255a pa port
          out dx,al
          ; scan mode -> 8255a pb port
```

```
            inc dx
            mov al,ah
            out dx,al
            ; delay 1ms
            call dl1
            mov dx,pb_8255
            mov al,0ffh
            out dx,al
            test ah,80h
            jz ld1
            rol ah,01h
            jmp ld0
ld1:   pop dx
            pop bx
            pop ax
            ret
dir endp

; delay 1ms
dl1 proc near
            push cx
            mov cx,500
            loop $
            pop cx
            ret
dl1 endp

keyi proc near
            push bx
            push dx
lk:         call allkey  ;调用判有无闭合键子程序
           jnz lk1
            call dir
            call dir ;调用显示子程序,延迟 6ms
            jmp lk
lk1:        call dir
           call dir
            call allkey  ;调用判有无闭合键子程序
```

```
          jnz lk2
          call dir
          jmp lk
lk2:      mov bl,0feh  ;r2
         mov bh,0 ;r4
lk4:      mov dx,pb_8255
         mov al,bl
          out dx,al
          inc dx
          in al,dx
         test al, 03h
         jz right_kuohao
          test al,01h
          jnz lone
          xor al,al ;0 行有键闭合
          jmp lkp
lone:  test al,02h
         jnz next
          mov al,08h ;1 行有键闭合
lkp:      add  bh,al
lk3:      call dir ;判断释放否
         call allkey
          jnz lk3
          mov al,bh ;键号->al
         jmp ex
right_kuohao: mov bh,29h
         jmp lk3
ex:       pop dx
          pop bx
          ret
next:  inc bh ;列计数器加 1
         test bl,80h
          jz knd ;判是否已扫到最后一列
          rol bl,01h
          jmp lk4
knd:      jmp lk
keyi endp
```

```
allkey proc near   ;检测是否有键按下
          mov dx,pb_8255
          xor al,al
          out dx,al ;全"0"->扫描口
          inc dx
          in al,dx ;读键状态
          not al
          and al,03h ;取低二位
          ret
allkey endp

; put the ascii key into myinstr
input proc
          push di
          push dx
          cmp al,0ah
          jz p1
          cmp al, 0bh
          jz p2
          cmp al, 0dh
          jz p3
          cmp al, 0fh
          jz p4
          cmp al, 29h
          jz instring
          or al, 30h
instring: mov [bx], al
          inc bx
          jmp f
          ; +
p1:       mov al, 2bh
          jmp instring
          ; *
p2:    mov al, 2ah
          jmp instring
          ; -
p3:    mov al, 2dh
          jmp instring
```

```
          ; (
p4:    mov al, 28h
          jmp instring
f:        pop dx
         pop di
         ret
input endp

; expression input ends
outv  proc
          push ax
          ; =
          mov al, 3dh
          mov [bx], al
          ; calculate
          call cclt
          ;error right
          lea di, buffer

          ;error left
          ; lea di,buffer
          ; add di,5

          ; "err"?
          cmp byte ptr[di], 11h
          jz rt
          call  display_value
rt:    call clear
          pop ax
          ret
outv endp

; reset the memory
clear proc
          push ax
          push di
          push cx
          ; reset myinstr
```

```
            mov al, '$'
            lea di, myinstr
          mov cx, 50
          rep stosb
            ; reset result
            lea di, result
            mov cx, 50
          rep stosb
            ; reset stri
          mov al, 00h
          lea di,stri
          mov cx, 50
          rep stosb
            ; reset exp
          lea di,exp
          mov cx, 50
          rep stosb
            ; reset track
          lea di,track
          mov cx, 50
          rep stosb
          lea bx, myinstr
            pop cx
            pop di
            pop ax
            ret
clear endp

; display "err"
error  proc
            push ax
            push bx
            push dx
          lea di,buffer

            ; show or
             ; mov al, 11h
            ; stosb
```

```
            ; mov al, 12h
            ; stosb

            ;err rigth
            mov al, 11h
            mov cx, 02h
            rep stosb
           mov al, 0eh
            stosb
            mov  al,10h
            mov cx,05h
            rep stosb

            ;err letf
            ; std
            ; add di,7
            ; mov al, 0eh
            ; stosb
            ; mov cx,2
            ; mov al, 11h
            ; rep stosb
            ; cld



            call dir
            pop dx
            pop bx
            pop ax
            ret
error endp

; display the result
display_value proc
            push ax
            push bx
            push dx
            push cx
```

```
            lea di, buffer
            lea si, result

            ; show right
            mov cx,si
yyt:        cmp byte ptr[si],24h
            jz  find
            inc si
            jmp yyt
find:       dec si
            cmp si,cx
            jz ok
            mov al,[si]
            mov [di],al
            inc di
            jmp find
ok:        mov al,[si]
            mov [di],al

            ; show left
            ; add si, 7
             ; mov cx, 8
         ; myl: mov al, [si]
            ; dec si
            ; stosb
            ; loop myl

            call dir
            pop cx
            pop dx
            pop bx
            pop ax
            ret
display_value endp

; initical
init proc
            push ax
```

```
            push bx
            push dx
            lea di,buffer
            ; ; the lowest digit shows "0"
            mov al, 00h
            stosb
            mov  al,10h
            mov cx,07h
            rep stosb

            ; the heightest digit shows "0"
            ; mov  al,10h
            ; mov cx,07h
            ; rep stosb
            ; mov al, 00h
            ; stosb

            pop dx
            pop bx
            pop ax
            ret
init  endp

; calculate the value
cclt proc near
            push ax
            push bx
            push cx
            push dx
            push si
            push di
            call near ptr update
            xor ax,ax
                   ; read input expression
            mov si,offset myinstr
            mov ah,30h
            ; record '('
            mov ch,0
```

```
            ; record ')'
            mov cl,0
l3:     mov al,[si]
            inc si
            cmp al,3dh
            ; meeting '=' means over
            jz l3_over
            cmp al,2ah
            ; check
            jnb may_wrong
            cmp al,29h
            jz l3_29
            ; == '('
            inc ch
            jmp l3_right
            ; == ')'
l3_29: inc cl
            ;*******
            ;cmp cl,ch
            ;jne severe
            ; ()
            cmp ah,28h
            je severe
            jmp l3_right
severe:     call error
            jmp over3
            ;*******

            ;jmp l3_right

may_wrong: cmp al,30h
            jnb l3_right
            cmp al,28h
            jz l3_right
            ; check if the previous character is ')' or a digit
            cmp ah,29h
            jz l3_right
            cmp ah,30h
```

```
          jnb l3_right
          ; wrong input
          call error
          jmp over3
l3_right: mov ah,al
          jmp l3
l3_over: cmp ch,cl
          jz input_wright
          ; ')' and '(' does not match
          call error
          jmp over3
;chech over
input_wright: lea  di, stri
          lea si, myinstr
          ;load instr into stri
read:  lodsb
          ; (
          mov ah,28h
          cmp ah,al
          jz in_stri_le
          ; )
          mov ah,29h
          cmp ah,al
          jz in_stri_ri
          ; *
          mov ah,2ah
          cmp ah,al
          jz in_stri_mu
          ; +
          mov ah,2bh
          cmp ah,al
          jz in_stri_add
          ; -
          mov ah,2dh
          cmp ah,al
          jz in_stri_sub
          ; =
          mov ah,3dh
```

```
          cmp ah,al
          jz in_stri_eq
          call near ptr mult
          jmp read
in_stri_add: mov ax,0
            stosw
          mov ax, 05f6h
          stosw
          jmp read
in_stri_sub:mov ax,0
            stosw
          mov ax, 05f7h
          stosw
          jmp read
in_stri_mu:mov ax,0
            stosw
          mov ax, 05f8h
          stosw
          jmp read
in_stri_le:mov ax,0
            stosw
          mov ax, 05fah
          stosw
          jmp read
in_stri_ri:mov ax,0
            stosw
          mov ax, 05fbh
          stosw
          jmp read
in_stri_eq:mov ax,0
          stosw
            mov ax, 05f9h
            stosw

over:     call near ptr trans
over1: call near ptr value
over2: call near ptr print
over3: pop di
```

```
          pop si
          pop dx
          pop cx
          pop bx
          pop ax
          ret
cclt endp

 update proc near
          push ax
          push bx
          push cx
          push dx
          push si
          push di
          lea si,myinstr
          inc si
nexx:  cmp byte ptr[si],30h
          jnb alert
nexxx:       inc si
          cmp byte ptr[si], 24h
          jne nexx
          jmp overt
alert: mov bx,1
          cmp byte ptr[si+bx], 28h
          jnz nexxx
          mov byte ptr[si+bx],29h
          inc bx
yt:    cmp byte ptr [si+bx], 28h
          jnz nexxx
          mov byte ptr [si+bx],29h
          inc bx
          jmp yt


overt: pop di
          pop si
          pop dx
```

```
          pop cx
          pop bx
          pop ax
          ret
 update endp

 ; convert ascii to decimal
mult   proc near
          push dx
          push cx
          push ax

          mov dx, si
search_c: lodsb
          cmp al, 30h
          jb search_over
          cmp al, 39h
          jna  search_c
search_over:dec dx
          mov cx, dx ; the ad of the first of the numeric string
          dec si
          dec si
          mov dx, si
          mov si, cx
          mov cx, dx ; the ad of the last of the numeric string
          xor dx, dx
          xor ax, ax


qjshao:   xor bx, bx
   push ax


          lodsb
          sub al, 30h
          mov bl, al
          pop ax
          add ax, bx
```

```
          jnc the_last
          adc dx, 0
the_last: dec si
          cmp si, cx
          jz cun
          inc si
          push cx
          mov cx, 3
          push ax
          push dx
mul_ten: add ax, ax
          adc dx,dx
          loop  mul_ten
          mov bx, ax
          mov cx, dx
          pop dx
          pop ax
          add bx, ax
          adc cx, dx
          add ax, bx
          adc dx, cx
          pop cx
          jmp qjshao
cun:      inc si
          stosw
          mov ax, dx
          stosw
          pop ax
          pop cx
          pop dx
          ret
mult  endp

; convert to rpn
trans proc near ;
          push ax
          push bx
          push cx
```

```
            push dx
            push si
            push di
            xor cx,cx
            mov bx,offset stri
            mov si,offset exp
            mov di,offset track
trans_while: mov ax,[bx]
            inc bx
            inc bx
            mov dx,[bx]
            inc bx
            inc bx
            ; equals '$'?
            cmp dx, 05f9h
            ; over
            jz yyyyt
            cmp dx, 05f6h
            ; character is an operator
            jnb no_digital
            ; if character is a number, insert into exp
            mov [si],ax
            inc si
            inc si
            mov [si],dx
            inc si
            inc si
yyyt:          jmp trans_while
no_digital: cmp dx, 05fah
            jnz no_9
            ; if character is '(', push into track
            mov [di],ax
            inc di
            inc di
            mov [di],dx
            inc di
            inc di
            jmp trans_while
```

```
no_9:  cmp dx, 05fbh
           jnz no_0
; if character is ')', pop track until meets '('
pop_while: dec di
           dec di
           mov dx,[di]
           dec di
           dec di
           mov ax,[di]
           cmp dx,05fah
           jz trans_while
           mov [si],ax
           inc si
           inc si
           mov [si], dx
           inc si
           inc si
           jmp pop_while
no_0:   cmp dx, 05f6h
           jz or1_yes
           cmp dx, 05f7h
           ; if character is not  '+' or '-'
           jnz no_or1
or1_yes: cmp di,offset track
           ; jump if stack is empty
           jz stack_blank
           dec di
           dec di
           mov cx,[di]
           dec di
           dec di
           cmp cx, 05fah
           jz over_or1_yes
           ; if track's top is not '(', push and add it into exp
           mov word ptr[si],0000h
           inc si
           inc si
           mov [si],cx
```

```
          inc si
          inc si
          jmp or1_yes
yyyyt:        jmp trans_over
over_or1_yes: mov word ptr[di],0000h
          inc di
          inc di
          mov [di], cx
          inc di
          inc di
; if track is empty, push character into it
stack_blank: mov [di],ax
          inc di
          inc di
          mov [di],dx
          inc di
          inc di
          jmp trans_while
no_or1: cmp dx,05f8h
          jnz yyyt
or2_yes: dec di
          dec di
          mov cx,[di]
          dec di
          dec di
          cmp cx, 05f8h
          jnz or2_over
          ; if character is '*', add it into exp
          mov word ptr[si],0000h
          inc si
          inc si

          mov [si],cx
          inc si
          inc si

          jmp or2_yes
or2_over: mov word ptr[di],0000h
```

```
            inc di
            inc di
            mov [di],cx
            inc di
            inc di
            mov word ptr [di], 0000h
            inc di
            inc di
            mov [di], dx
            inc di
            inc di
            jmp trans_while
trans_over: cmp di,offset track
            jz pop_over
            dec di
            dec di
            mov dx,[di]
              dec di
              dec di
              mov ax, [di]
            mov [si],ax
            inc si
            inc si
            mov [si],dx
            inc si
            inc si
            jmp trans_over
pop_over:
            mov word ptr[si],0000h
            inc si
            inc si
            mov [si],05f9h
            pop di
            pop si
            pop dx
            pop cx
            pop bx
            pop ax
```

```
            ret
trans endp

; calculate the value
value proc near
            push ax
            push bx
            push cx
            push dx
            push si
            push di
            mov di, offset exp
            mov si,offset track
            xor ax,ax
            xor dx,dx
value_while: mov ax,[di]
            mov dx,[di+2]
            add di,4
            ; over?
            mov cx,05f9h
            cmp dx,cx
            jz rtttt
            mov cx,05f5h
            cmp dx,cx
            jnbe value_no_digital
            ; if character is a digit, add it into track
            mov [si],ax
            mov [si+2],dx
            add si,4
            jmp value_while
rtttt:        jmp rtt
value_no_digital: mov cx,05f6h
            cmp dx,cx
            jnz no_add
            ;add
            push ax
            push bx
            push cx
```

```
            push dx
            sub si,4
            mov bx,[si]
            mov cx,[si+2]
            sub si,4
            mov ax,[si]
            mov dx,[si+2]
            add ax,bx
            adc dx,cx
            jc omg
            cmp dx, 05f5h
            jb haode
            cmp dx, 05f5h
            ja omg
            cmp ax,0e0ffh
            ja omg
haode: mov [si],ax
            mov [si+2],dx
            add si,4
            pop dx
            pop cx
            pop bx
            pop ax
            jmp value_while
no_add: mov cx,05f7h
            cmp dx,cx
            jnz value_no_sub
            ;sub
            push ax
            push bx
            push cx
            push dx
            sub si,4
            mov bx,[si]
            mov cx,[si+2]
            sub si,4
            mov ax,[si]
            mov dx,[si+2]
```

```
            cmp dx,cx
            jb omg
            cmp dx,cx
            ja sub_right
            cmp ax,bx
            jb omg
            jmp sub_right
omg:        call error
            pop dx
            pop cx
            pop bx
            pop ax
rtt:            jmp value_over
sub_right: sub ax,bx
            sbb dx,cx
            mov [si],ax
            mov [si+2],dx
            add si,4
            pop dx
            pop cx
            pop bx
            pop ax
rttt:           jmp value_while
value_no_sub: mov cx,05f8h
            cmp cx,dx
            jnz rttt
            ; mul
            push ax
            push bx
            push cx
            push dx
            sub si,4
            mov bx,[si]
            cmp word ptr[si+2],0
            jnbe omg
            sub si,4
            mov ax,[si]
            cmp word ptr[si+2],0
```

```
            jnbe omg
            mul bx
        ;   jc omg
            cmp dx, 05f5h
            jb right
            cmp dx, 05f5h
            ja omg
            cmp ax,0e0ffh
            ja omg
right: mov [si],ax
            mov [si+2],dx
            add si,4
            pop dx
            pop cx
            pop bx
            pop ax
            jmp value_while
value_over: pop di
            pop si
            pop dx
            pop cx
            pop bx
            pop ax
            ret
value endp

; convert to ascii
print proc near
            push ax
            push bx
            push cx
            push dx
            push si
            mov si,offset track
            mov ax,[si]
            mov dx,[si+2]
            mov cx,0ah
            mov si,offset result
```

```
            call dtoc
            pop si
            pop dx
            pop cx
            pop bx
            pop ax
            ret
print endp

dtoc proc near
            push bx
            push cx
            push dx
            push di
            push si
            ; the number of yushu
            mov di,0
s1:     mov cx,10
            call divdw
            inc di
            add cx,30h
            push cx
            cmp dx,0
            jne s1
            cmp ax,0
            jne s1
            mov cx,di
p:          pop [si]
            inc si
            loop p
            mov byte ptr[si],24h
            pop si
            pop di
            pop dx
            pop cx
            pop bx
            ret
dtoc endp
```

```
divdw proc near
          push bx;
          push di;
          mov bx,ax
          mov ax,dx
          mov dx,0
          div cx
          mov di,ax
          mov ax,bx
          div cx
          mov cx,dx
          mov dx,di
          pop di
          pop bx
          ret
divdw endp

code ends
end start
```