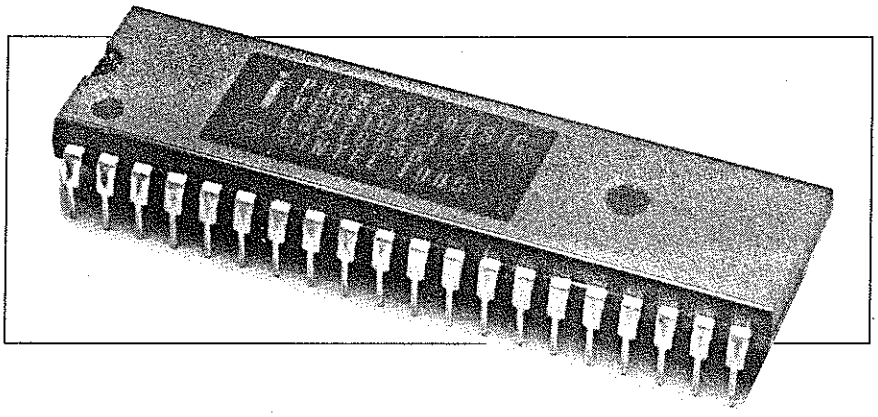
UPGRADE FOR MCS BASIC-52 V.1.1 (Part 2)

Following last month's dealings with the floating-point nucleus and the hex-to-BCD conversion routine in the MCS-52 BASIC interpreter, the authors now tackle some problems with multiplications.



by Z. Stojsavljevic and D. Mudric

Continued from the October 1991 issue.

INCONSISTENCIES in the multiplication of two numbers as performed by version 1.1 of Intel's MCS BASIC-52 interpreter can be demonstrated by running the following three small programs, all of which produce wrong results.

```
10    a=1.E-65
20    b=1.E-65
30    ?a*b          (result: 1.0E+126)

10    a=1.E-65
20    b=1.E-64
30    ?a*b          (result: 0)

10    a=1.E-64
20    b=1.E-64
30    ?a*b          (result: 1.0E-0)
```

In all three cases, the interpreter should have produced

ERROR: ARITH. UNDERFLOW - IN LINE 30

The above examples point to inconsistencies in limit cases when two numbers are multiplied. On investigating the operation of Intel's BASIC interpreter, a multiplication algorithm of the type shown in Fig. 1 was found. Apparently, the inconsistencies brought to our attention by the above example programs were caused by the exponent adjustment routine, which is listed in Fig. 2. Unfortunately, the errors found in this routine cannot be corrected in BASIC-52 machine code, because some expansion of the machine code is in order. This means that a number of lines should be added to the assembler source file for compensation.

A problem occurs when the result of the instruction SUBB A,#81H is 0FFH, which equals exponent E+127. If the program part for mantissa multiplication includes a result that begins with a 0 after the decimal point, the exponent is not incremented but remains

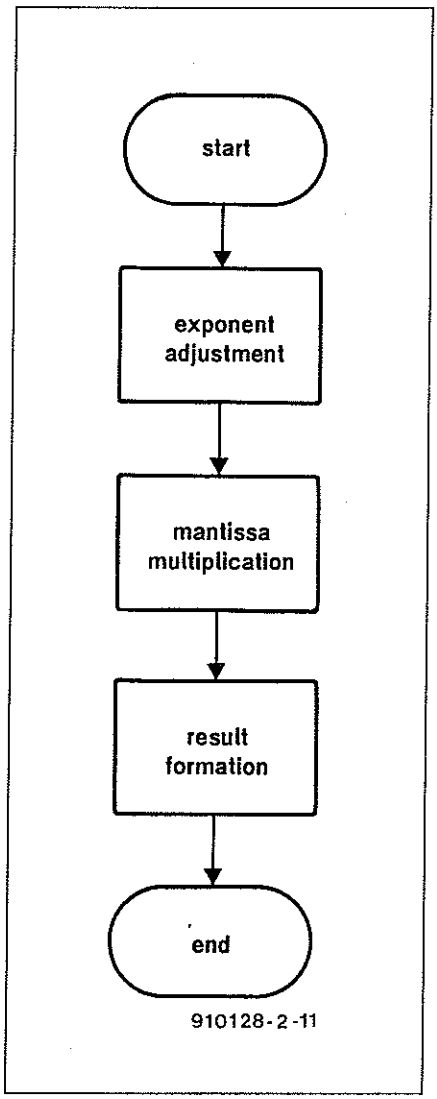


Fig. 1. Flowchart of the routine that handles multiplication of two numbers.

```
1A9AH 9175      ACALL   1C75H
1A9CH BE0002    CJNE    R6,#00H,1AA1H
1A9FH 6188      AJMP    1B88H
1AA1H 8D2F      MOV     2FH,R5
1AA3H EF        MOV     A,R7
1AA4H 60F9      JZ      1A9FH
1AA6H 2E        ADD     A,R6
1AA7H 20E705    JB      ACC.7,1AAEH
1AAAH 10D706    JBC     CY,1AB3H
1AADH 61B2      AJMP    1BB2H
1AAFH 5002      JNC     1AB3H
1AB1H 61A1      AJMP    1BA1H
1AB3H 9481      SUBB    A,#81H
1AB5H FE        MOV     R6,A
1AB6H 718B      ACALL   1B8BH
1AB8H 7B04      MOV     R3,#04H
1ABAH AC01      MOV     R4,01H
1ABCH 8C01      MOV     01H,R4
1ABEH E3        MOVX    A,@R1
1ABFH FA        MOV     R2,A
1B30H 7834      MOV     R0,#34H
1B32H E6        MOV     A,@R0
1B33H FE        MOV     R6,A
1B34H 6003      JZ      1B39H
1B36H 717F      ACALL   1B7FH
1B38H 18        DEC     R0
1B39H 08        INC     R0
1B3AH 7408      MOV     A,#08H
1B3CH F9        MOV     R1,A
1B3DH 28        ADD     A,R0
1B3EH F8        MOV     R0,A
1B3FH B60500    CJNE    @R0,#05H,1B42H
1B42H 4013      JC      1B57H
1B44H D3        SETB    C
1B45H E4        CLR     A
1B46H 18        DEC     R0
1B47H 36        ADDC    A,@R0
1B48H D4        DA      A
1B49H D6        XCHD    A,@R0
1B4AH 30E409    JNB     ACC.4,1B56H
1B4DH D9F5      DJNZ    R1,1B44H
1B4FH 18        DEC     R0
1B50H 7601      MOV     @R0,#01H
1B52H 717F      ACALL   1B7FH
1B54H 8006      SJMP    1B5CH
1B56H 19        DEC     R1
1B57H E9        MOV     A,R1
1B58H C3        CLR     C
1B59H C8        XCH     A,R0
1B5AH 98        SUBB    A,R0
1B5BH F8        MOV     R0,A
1B5CH 792B      MOV     R1,#2BH
1B5EH E6        MOV     A,@R0
1B5FH C4        SWAP    A
1B60H 08        INC     R0
1B61H D6        XCHD    A,@R0
1B62H 4206      ORL     06H,A
1B64H F7        MOV     @R1,A
1B65H 08        INC     R0
1B66H 09        INC     R1
1B67H B92FF4    CJNE    R1,#2FH,1B5EH
1B6AH EE        MOV     A,R6
1B6BH 7003      JNZ     1B70H
1B6DH 753000    MOV     30H,#00H
1B70H 0530      INC     30H
1B81H E530      MOV     A,30H
1B83H 70F9      JNZ     1B7EH
1B85H D0E0      POP     ACC
1B87H D0E0      POP     ACC
1B89H 61A1      AJMP    1BA1H
                                910128-2-12
```

Fig. 2. Original program part for exponent adjustment.

Fig. 3. that corr

```
        ORG     1A9AH

        ; TOS   - Top Of arithmetic Stack
        ; NXTOS - NeXt TOS (position behind)

        ; TOS_MUL1
        ; routine for multiplication of two FP numbers, one of which
        ; is located on TOS and the other on NXTOS (NXTOS*TOS)

TOS_MUL1:ACALL  PREP_MUL; clearing of FP working space and reg. prep.
         CJNE   R6,#0,NXTNM_0   ; is NXTOS equal 0?
TOSM_0:  AJMP   ZERO_MANTISSA   ; in case TOS or NXTOS are 0
NXTNM_0: MOV    SIGN,R5         ; result mark in SIGN
         MOV    A,R7    ; is TOS equal 0?
         JZ     TOSM_0
         ADD    A,R6    ; addition of exponent degree (exp. multipl.)
         JB     ACC.7,CMPM_EXP  ; exp. with different mark or overflow?
         JBC    CY,CORM_EXP     ; result is bigger than 0.1
         AJMP   UNDERFLOW       ; in this case exp. sum is < -127
CMPM_EXP:JNC    CORM_EXP; in this case result is smaller than 0.1
OVER_MD: AJMP   OVERFLOW; in this case exp. sum is > 127
CORM_EXP:CLR    MUL_LIMIT_CASE  ; flag of multiplication limit case
         SUBB   A,#82H  ; exp. multipl. results are within the limits
         INC    A       ;  deduction  with  82H  because  of  reduced
         JNC    NMARK_L ;  conditions
         SETB   MUL_LIMIT_CASE  ; limit case
NMARK_L: MOV    R6,A    ; exp. adjustment to real value
         ACALL  BCD2_1  ; disassembl. of TOS mantissa in LEN_MANTISSA
         MOV    R3,#LEN_BYTE    ; acc. (one number in each byte)
         MOV    R4,AR1  ; R1 is pointer of NXTOS
MUL_NOOV:MOV    AR1,R4  ; @R1 contains few numbers and disassembled
         MOVX   A,@R1   ; mantissa is successively multiplied by each
         MOV    R2,A    ; of  them  in  argument  accumulator  (R2  is
                        ; auxilliary register)

;---------------------------------------

        ; TRANSFER
        ; routine  for  adjustment  of  multiplication  and  division
        ; results of  FP  numbers  in  argument  stack  and  mantissa
        ; conversion in BCD packed  format  in  argument  accumulator

TRANSFER:MOV    R0,#ASCII_DEC   ; pointer to MSB result number
         MOV    A,@R0   ; MSB result number is in acc.
         MOV    R6,A
         JZ     MSB_EQ_0; is MSB result number equal 0?
         ACALL  INC_EXP ; exp=exp+1 because from the start it was
         DEC    R0      ; taken that MSB result number is equal 0
MSB_EQ_0:INC    R0      ; pointer to MSB-1 number
         MOV    A,#LEN_MANTISSA ; positioning to LSB-1 number
         MOV    R1,A    ; R1 is counter
         ADD    A,R0
         MOV    R0,A    ; R0 contains the address of LSB-1 number
         CJNE   @R0,#05,ROUND1  ; test of remainder to rounding
ROUND1:  JC     NO_ROUND; in case remainder is smaller than 5 there
C_BCD:   SETB   C       ; is no rounding
         CLR    A       ; carry takes one because of rounding
         DEC    R0      ; transfer of carry into higher byte
         ADDC   A,@R0   ; addition of 1 to higher byte
         DA     A       ; adjustment to BCD format
         XCHD   A,@R0   ; result storing in higher byte
         JNB    ACC.4,NO_C_BCD  ; transfer in higher BCD nibble?
         DJNZ   R1,C_BCD; is @R0 the address of MSB number?
         DEC    R0      ; transfer memorizing in the place MSB+1
         MOV    @R0,#01
         ACALL  INC_EXP ; transfer is outside decimal point frame
         SJMP   BCD1_2
NO_C_BCD:DEC    R1
NO_ROUND:MOV    A,R1    ; address return in R0 to the first bigger BCD
         CLR    C       ; number which is not equal 0
         XCH    A,R0
         SUBB   A,R0
         MOV    R0,A
BCD1_2:  MOV    A,EXPONENT      ; test of exceeding in limit case
         JNB    MUL_LIMIT_CASE,N_LCASE
         JZ     UNDER_MD; message about underflow
         INC    A
         JZ     UNDER_MD; message about underflow
N_LCASE: MOV    R1,#ARGUMENT_ACC; repacking of BCD number in one byte
XFER_1:  MOV    A,@R0   ; in two BCD numbers in one byte
         SWAP   A       ; higher number in BCD packed  format is in
         INC    R0      ; upper nibble
         XCHD   A,@R0   ; BCD packed format is in ACC
         ORL    AR6,A   ; the indication if mantissa is 0 is in R6
         MOV    @R1,A   ; packing in argument accumulator mantissa
         INC    R0      ; two BCD packed numbers in one byte
         INC    R1
         CJNE   R1,#SIGN,XFER_1
         MOV    A,R6
         JNZ    RESULT_MAT_OP_TOS       ; is mantissa equal 0?
         MOV    EXPONENT,#0

;---------------------------------------

INC_EXP: INC    EXPONENT; exp. adjustment by 1 (consequence of
         MOV    A,EXPONENT      ; transfer after addition)
         JNZ    EXP_OK  ; test of exponent exceeding
         POP    ACC     ; removal from return address stack
         POP    ACC
         JBC    MUL_LIMIT_CASE,UNDER_MD ; in limit case it is
         AJMP   OVERFLOW; underflow
UNDER_MD:AJMP   UNDERFLOW
```

910128-2-13

Fig. 3. Source code of the improved multiplication routine. The vertical bars indicate lines that correct the limit case multiplication errors mentioned in this article.

0FFH. After the result formation (see Fig. 1) and the serial output (display) routine, this gives an exponent E+126.

To avoid inconsistencies, and eliminate errors in limit case multiplications, the program listed in Fig. 3 was developed. The lines marked by vertical bars in particular correct the limit case multiplication errors.

On the basis of the information contained in this and last month's instalment, the authors developed an improved, error-free floating-point (FP) arithmetic for the 8051, on the basis of the FP nucleus extracted from the 8052AH-BASIC. The new FP nucleus uses a modified way of accessing the arithmetic stack, and has faster, shorter code for a number of algorithms. It allows the length of the mantissa of a FP number to range from 2 to 16 digits, while the entire memory map of the FP arithmetic variables is located in the internal memory space of the 8051 microcontroller. ■