Atari/Epson Custom Characters

Transfer almost unlimited customized alphabets to paper - accurately.

by Mike Bassman

Requirements: Atari 400/600/800/1200, 850 Interface, Epson MX-80 with Graftrax +, or MX-80 FT or MX-100, or FX-80

When it comes to dealing with text, the Atari computers have a marvelous flexibility. Naturally, they can display the usual upper and lower case, numbers and punctuation. Besides that, they also have inverses of all the standard characters, plus lines, card suits and a host of other graphics characters. If you're not satisfied with this selection, you can make your own custom characters which has led to programs using gothic, script and other interesting fonts. You can generate your own fonts with one of the many character-editing programs that have been published. All these can easily be displayed on the screen, but transferring them to paper is not normally possible.

Why? There are a number of reasons. The main one is that printers are not designed for any one particular computer. Only alphanumerics and punctuation symbols are in the standard ASCII table. The maximum possible number of characters is 256 (each character stored uses one byte; a byte can be in the range of 0-255, hence, 256 possibilities). These standard characters fill up less than half of the available room, so Atari decided to pack the rest with inverse and graphics characters. Radio Shack, instead, throws in a combinations of block characters. Commodore has inverse characters and different graphics characters. The point here is that apart from normal characters, no two computers have the same set of 'extra' characters. As such, a printer manufacturer catering only to one computer would have a limited audience. Epson has a viable solution; they have their own characters, an italic set and a few graphics characters. The important fact is that the Epson printers have full graphics abilities. We can take advantage of this to generate Atari's own special and custom characters. All you need is the appropriate software. I've included listings of two somewhat similar programs; one useful and one frivolous.

Any-Text File Lister

The program shown in Listing 1 lists files to an Epson printer. At first this may not sound amazingly useful. I mean, from Basic, this is merely a matter of issuing a LIST "P:" command. From the DOS utilities menu you can copy a text file to P:. but what makes this program useful is that its listing is accurate: it includes all the graphics and inverse characters. If you've ever tried listing a program with graphics or inverse, you'll know that inverse shows up as italics and graphics characters show up as meaningless garbage, or some odd control character will throw the printer into a stupor that will mess up the rest of the listing. This program lists out a program in its exact form, graphics and all. If examined closely, you can see that this program was used to list itself.

Using this program is simple. If there is a Basic program you wish to list, load it in from disk (or cassette) and then re-save it out under a different name using a LIST "D:name" command, rather than the SAVE command. The purpose behind this is to have the program as text, rather than encoded in Basic keywords. If it's a text file you want to print, you clearly don't have to do this. Next run this program and enter the name of the file your text is in. That's all it takes.

Custom Font Message Printer

The program in Listing 2 will print anything you want in a custom character set. If you've ever had a desire to see a message in script or computer-type letters or whatever, this will do the trick. All you need to have are the custom fonts stored on disk (or cassette) in a nine sector file as generated by Instedit (APX) or just about all of the other character generators. There are a few examples of its handiwork shown in the accompanying chart. From top to bottom, the alphabet is shown in computer-style, gothic, fancy fonts and a few others. Making use of this program (custom font message printer) is even easier than the file lister above. Just run it and, when prompted, enter the name of your character set and then the message you want printed. If your character set is stored on cassette, type C: for the character set name, when asked.

Custom Font Variant

Listing 4 shows a program that looks a good deal like the custom font message printer. In fact, it is a cross between that program and the program lister. It does the same thing as the custom font program except that it prints out a whole file in the new font rather than a one line message. This would be useful to take a file generated with a word processing program and, after putting it through this program, end up with a professional looking document printed

in a pleasant typeface of your choice. There are some commercially available programs which do just this. Using it consists simply of entering the name of the font and the name of the file to be printed.

Entering The Programs

Typing in these programs can be a bit of a problem because of the machine language subroutine embedded in them. The Basic part is easy to do. The straightforward approach is to type those graphics characters just as you see them. The graphics keyboard included in the Atari Basic Reference manual is a guide to finding all the right keys (the back cover has a relatively easy to use diagram). If you do decide to do it this way, make sure that you save the program before running it. Any typo could bomb the computer; let this be a word to the wise.

Another method is to assemble the source code using the Assembler/ Editor cartridge or one of the many other assemblers available. The source code for the machine language portion is shown in Listing 3. The programs all use the same machine language subroutine, so the most difficult part only has to be done once, even if you want all the programs. The amount of code needed to be typed in this way is longer than typing the graphics characters, but fortunately the code is made up of normal alphanumeric characters. You then assemble the code to a disk file and modify either of the programs to load in this subroutine from that disk file. If you choose to do it this way (not highly recommended), the changes to listings 1 & 2 are as follows:

1 - Delete lines 11,12,13.
2 - Add lines 90-130 as shown.

```
90 OPEn #3,4,0,"D:ASSEM.OBJ"
100 FOR K=1 TO 6:GET #3,X:NEXT K
110 TRAP 130:K=0
120       K=K+1:GET       #3,X:
ML$(K,K)=CHR$(X):GOTO 120
130 CLOSE #3
```

The modification here can be used with both programs and with the variant by changing the line numbers. The filename in line 90 is your assembled version of the source code. These programs are very similar, so if you want to have all of them, I would recommend typing in one of them, saving it, and then modify it until you have the other program. Conversely, if you're only bothering with one program, keep in mind it doesn't take much effort to obtain the other ones.

Theory of Operation

You already have all you need to know to get these programs working. But if you want to know how they work, and maybe do clever things of your own with your Epson, read on.

You may ask how we get a large number of new character sets out of a printer normally limited to regular and Italic characters, in various sizes. Well, these programs don't exactly print out new characters, they draw them. All reasonably new Epson printers (or old ones retrofitted with Graftrax +) have the ability to do graphics. There's no reason to limit use of the graphics to charts or drawings; you can also improve on your regular text performance. The basic task to be accomplished is to get a character from the file, find out how the character is drawn, send this information over to the Epson and repeat this procedure until you reach the end of the file (or message).

Where Character Shapes Are

The character set in use can be found at the address specified by PEEK(756) *256. Location 756 (2F4 hex) is the Character Base register, holding the high byte of the address. The low byte is assumed to be zero. The standard character set is at $E000. Though the topic has been more comprehensively covered in other articles, let me quickly refresh your memory on how they (the characters) are stored. Each character can be 8 bits wide and 8 bits high, total of 8 bytes (64 bits). Usually there is a little room on top, bottom and the sides so that characters won't be squeezed too tightly together when shown next to each other. Each row of a character is one byte and there are eight rows going from the top to bottom of any one character. So the capital letter "E" is represented in memory like so:

```
00000000    byte = $00
01111110    byte = $7E
01100000    byte = $60
01111100    byte = $7C
01100000    byte = $60
01100000    byte = $60
01111110    byte = $7E
00000000    byte = $00
```

Getting a Character to the Printer

This is how the computer reads a normal or redefined character. Now we've got to send this information out to the printer. Things would be simple if the printer could be fed the character a byte (row) at a time, just like the computer understands them. But this isn't the case. While the computer reads a character a row at a time, from top to bottom, the printer head is a vertical column, so it does each character a column at a time, from left to right. This makes life difficult. What we're going to have to do is take each byte that forms a row of the character and take off the leftmost bit. We're going to take these bits off all eight rows, line them up in a column and then send the column off to the printer. Then we do this for the eight columns that make a character, from left to right. Visually, this means that instead of taking slices of bits off the top of say, that 'E' we saw earlier, we take slices vertically off the sides.

Theory into Code

This cut and paste type of operation with bits can be turned into a basic program. To output one complete character, we need two loops, one going from left to right sending out columns of data and an inner loop that puts together these columns. There's a chart showing each pin of the print head and what is needed to turn it on:

```
128  - o
 64  - o
 32  - o
 16  - o
  8  - o
  4  - o
  2  - o
  1  - o
```

For each of the pins you want to turn on, add that number. For example sending a 34 would turn on the third pin from the top and the second from the bottom. It's no surprise that each of these values is 2 to the power of the pin number (pins numbers range from 0 to 7, bottom to top), and we'll use this fact. To find out if we want to turn on a pin, we look at a row of the character, AND it with the column number we're up to and, if we get a positive value, we know to turn it on. Column numbers, not coincidentally, are represented just like the pin numbers but from right to left, instead of bottom

Listing 2

```
                ; CUSTOM CHARACTER DUMP
                ; MIKE BASSMAN
                ;
0342            ICCOM   EQU $342
0344            ICBAL   EQU $344
0345            ICBAH   EQU $345
0348            ICBLL   EQU $348
0349            ICBLH   EQU $349
0680            B       EQU $680
0681            G       EQU $681
0682            SUM     EQU $682
002C            VL      EQU $2C
0685            J       EQU $685
0687            AD      EQU $687
0689            VALUE   EQU $689
068A            MASK    EQU $68A
068B            RESULT  EQU $68B
068C            INVRS   EQU $68C
                ;
5000                    ORG $5000
                ;
                ; SAVE NUMBER OF ARGUMENTS
                ; CHAR MEM HI
                ; CHAR MEM LO
                ; SET INVERSE VIDEO FLAG
                ;
5000 68         START   PLA
5001 68                 PLA
5002 8D 88 06           STA AD+1
5005 68                 PLA
5006 8D 87 06           STA AD
5009 68                 PLA
500A 68                 PLA
500B 8D 85 06           STA J
500E A9 00              LDA #$00
5010 8D 8C 06           STA INVRS
5013 AD 85 06           LDA J
5016 C9 80              CMP #$80
5018 90 0A              BCC INIZ
501A 29 7F              AND #$7F
501C 8D 85 06           STA J
501F A9 FF              LDA #$FF
5021 8D 8C 06           STA INVRS
                ;
                ; SETUP ROW/COLUMN COUNTERS
                ;
5024 A9 07      INIZ    LDA #7
5026 8D 80 06           STA B
5029 A9 00      FIRSTG  LDA #0
502B 8D 81 06           STA G
502E 8D 82 06           STA SUM
5031 A9 00      SETVL   LDA #0
5033 85 2C              STA VL
5035 85 2D              STA VL+1
5037 AD 85 06           LDA J
503A A2 03              LDX #3
                ;
                ; MULTIPLY BY 8
                ;
503C 18         MULT8   CLC
503D 06 2D              ASL AL+1
503F 0A                 ASL A
5040 90 02              BCC LOWDO
5042 E6 2D              INC VL+1
5044 CA         LOWDO   DEX
5045 D0 F5              BNE MULT8
5047 85 2C              STA VL
5049 18                 CLC
504A 6D 87 06           ADC AD
504D 85 2C              STA VL
504F 90 02              BCC ADDHI
5051 E6 2D              INC VL+1
5053 18         ADDHI   CLC
5054 A5 2D              LDA VL+1
5056 6D 88 06           ADC AD+1
5059 85 2D              STA VL+1
505B 18                 CLC
505C A5 2C              LDA VL
505E 6D 81 06           ADC G
5061 85 2C              STA VL
5063 90 0A              BCC ANDIT
5065 E6 2D              INC VL+1
5067 B0 06              BCS ANDIT
5069 18         ELONG   CLC
506A 90 C5              BCC SETVL
506C 18         LONG    CLC
506D 90 BA              BCC FIRSTG
506F A0 00      ANDIT   LDY #0
5071 B1 2C              LDA (VL),Y
5073 4D 8C 06           EOR INVRS
5076 8D 89 06           STA VALUE
5079 AE 80 06           LDX B
507C E0 00              CPX #0
507E D0 05              BNE SMALL
5080 A9 01              LDA #1
5082 18                 CLC
5083 90 08              BCC AMAIN
5085 A9 02      SMALL   LDA #2
5087 CA         DO      DEX
5088 F0 03              BEQ AMAIN
508A 0A                 ASL A
508B D0 FA              BNE DO
508D 8D 8A 06   AMAIN   STA MASK
5090 2D 89 06           AND VALUE
5093 8D 8B 06           STA RESULT
5096 C9 00              CMP #0
5098 F0 1F              BEQ NEXTG
509A 38                 SEC
509B A9 07              LDA #7
509D ED 81 06           SBC G
50A0 C9 00              CMP #0
50A2 D0 05              BNE POWER
50A4 A9 01              LDA #1
50A6 18                 CLC
50A7 90 09              BCC DOSUM
50A9 AA         POWER   TAX
50AA A9 02              LDA #2
50AC CA         DO2     DEX
50AD F0 03              BEQ DOSUM
50AF 0A                 ASL A
50B0 D0 FA              BNE DO2
50B2 18         DOSUM   CLC
50B3 6D 82 06           ADC SUM
50B6 8D 82 06           STA SUM
50B9 EE 81 06   NEXTG   INC G
50BC A9 08              LDA #8
50BE CD 81 06           CMP G
50C1 D0 A6              BNE ELONG
50C3 A2 40              LDX #$40
50C5 A9 0B              LDA #$B
50C7 9D 42 03           STA ICCOM,X
50CA A9 82              LDA #SUM
50CC 9D 44 03           STA ICBAL,X
50CF A9 06              LDA /SUM
50D1 9D 45 03           STA ICBAH,X
50D4 A9 01              LDA #1
50D6 9D 48 03           STA ICBLL,X
50D9 A9 00              LDA #0
50DB 9D 49 03           STA ICBLH,X
50DE 20 56 E4           JSR $E456
50E1 18         NEXTB   CLC
50E2 CE 80 06           DEC B
50E5 10 85              BPL LONG
50E7 60                 RTS
50E8                    END
```

to top. This strategy is represented in the following piece of Basic-like code.

J=ASC(CHARACTER)
A=ADR(STARTOFCHARACTER-DATA)
FOR B=7 TO 0 STEP -1:REM the outer column loop.
SUM=0:REM clear the print head counter.
FOR G=0 TO 7:REM inner loop totals up a column.
Y=PEEK(A+G+J*8):REM get the row value.
X=INT(2 B+.5):REM the column number.
Z=X AND Y:REM you can't do a boolean AND in Basic, but you get the idea.
IF Z THEN SUM=SUM + INT(2 (7-G)+.5):REM add pin value to running total if we should.
NEXT G:REM do it for the entire column.
PUT #4,SUM:REM output the column to printer.
NEXT B:REM now do this for all 8 columns.

This is ridiculously slow when done in Basic, so the machine language subroutine just uses this algorithm, but runs infinitely faster. There is only one other thing you need to know to control the Epson. Before you start sending all this pin information, you have to tell it to go into high resolution mode and then say how many columns of graphics you want. Turning on graphics is done by sending an ESCAPE, then a "k". You tell it how many columns of graphics by sending out two more values, the first being the low byte of the # of columns, the second being the high byte.

That's all there is to making your Epson print anything you want. The programs listed here are only a few of the possible applications. Using some of the information shown here, you can invent new and interesting uses for your printer.

```
  13 ML$(198)="[illegible]"
  17 I=0:DIM X$(11),M$(255)
  20 X$="h[illegible]"
  30 A=PEEK(756)*256
  40 DIM N$(15),NM$(15)
  50 GRAPHICS 18
  60 POSITION 5,1:? #6;"ANY TEXT"
  70 POSITION 2,5:? #6;"[illegible]"
  80 POSITION 5,9:? #6;"for epson"
```

Printer Sample

Listing 1

```
  10 DIM ML$(300)
  11 ML$(1)="[illegible]"
  12 ML$(92)="[illegible]"
  13 ML$(198)="[illegible]"
  17 I=0:DIM X$(11),M$(255)
  20 X$="h[illegible]"
  30 A=PEEK(756)*256
  40 DIM N$(15),NM$(15)
  50 GRAPHICS 18
  60 POSITION 5,1:? #6;"ANY TEXT"
  70 POSITION 2,5:? #6;"[illegible]"
  80 POSITION 5,9:? #6;"for epson"
  90 FOR K=1 TO 2000:NEXT K
 190 GRAPHICS 0:POSITION 2,6
 200 ? "Please enter the name of the disk      file which cont
ains the program to be listed.   (ex: PROG.BAS)"
 210 ? :? "Name of program:"
 220 TRAP 350:NM$="D:":INPUT N$:NM$(LEN(NM$)+1)=N$
 230 OPEN #3,4,0,NM$
 240 INPUT #3,M$
 270 TRAP 400
 285 Q=LEN(M$):IF LEN(M$)>60 THEN Q=60
 290 I=0:J=Q*8:IF J>255 THEN I=1:J=J-256
 295 OPEN #4,8,0,"P"
 300 PUT #4,27:PUT #4,ASC("K"):PUT #4,J:PUT #4,I
 310 FOR K=1 TO Q
 320 J=ASC(M$(K,K)):IF J<96 AND J>31 THEN J=J-32:GOTO 330
 322 IF J<32 THEN J=J+64:GOTO 330
 324 IF J<224 AND J>159 THEN J=J-32:GOTO 330
 326 IF J<160 AND J>127 THEN J=J+64
 330 DUMMY=USR(ADR(ML$),A,J)
 340 NEXT K:PUT #4,27:PUT #4,64:CLOSE #4:LPRINT
 342 IF Q<LEN(M$) THEN M$=M$(61):GOTO 285
 345 TRAP 350:GOTO 240
 350 CLOSE #3:CLOSE #4:GRAPHICS 18
 360 POSITION 6,4:? #6;"all done"
 370 FOR K=255 TO 0 STEP -1:SOUND 0,K,12,5:SOUND 1,255-K,12,3
:NEXT K:SOUND 1,0,0,0
 380 GRAPHICS 0:END
 390 ? :? "[illegible]":? :CLOSE #3:GOTO
200
 400 ? "[illegible]"
 410 END
 420 ? "[illegible]"
 430 END
```

How about a $2000 TYPEWRITER !

Printer Sample

SORT OF A BIZZARE 1960'S HIPPY TYPE STYLE

TRANSLYVANIA GOTHIC TIMES - Midnight Edition

THE COMPLETE OUTLINE OF ALL COMPUTER KNOWLEDGE

Listing 3

```
10 DIM ML$(300)
11 ML$(1)="hh[illegible]"
12 ML$(92)="[illegible]"
13 ML$(198)="[illegible]"
17 I=0:DIM X$(11),M$(60)
20 X$="h[illegible]"
30 DIM B$(1024):A=ADR(B$)
40 DIM N$(15),NM$(15)
50 GRAPHICS 18
60 POSITION 4,1:? #6;"CUSTOM FONT"
70 POSITION 2,5:? #6;"message printer"
80 POSITION 5,9:? #6;"for epson"
100 FOR K=1 TO 2000:NEXT K
140 GRAPHICS 0:POSITION 2,6
150 ? "Remember: maximum of 60 characters"
160 ? :? "Please enter message you wish to print"
170 ? "Message:":TRAP 140:INPUT M$
180 IF LEN(M$)=0 THEN 140
190 GRAPHICS 0:POSITION 2,6
200 ? "Please enter the name of the disk      file which cont
ains the character fontto be used (ex: GOTH.SET)"
210 ? :? "Name of character set:"
220 TRAP 390:NM$="D:":INPUT N$:NM$(LEN(NM$)+1)=N$
230 OPEN #3,4,0,NM$
240 FOR K=0 TO 1023
250 GET #3,B
260 POKE A+K,B
270 NEXT K:CLOSE #3:TRAP 400
280 OPEN #4,8,0,"P":TRAP 420
290 J=LEN(M$)*8:IF J>255 THEN I=1:J=J-256
300 PUT #4,27:PUT #4,ASC("K"):PUT #4,J:PUT #4,I
310 FOR K=1 TO LEN(M$)
320 J=ASC(M$(K,K)):IF J<96 AND J>31 THEN J=J-32
330 DUMMY=USR(ADR(ML$),A,J)
340 NEXT K:PUT #4,27:PUT #4,64:CLOSE #4:LPRINT
350 GRAPHICS 18
360 POSITION 6,4:? #6;"all done"
370 FOR K=255 TO 0 STEP -1:SOUND 0,K,12,5:SOUND 1,255-K,12,3
:NEXT K:SOUND 1,0,0,0
380 GRAPHICS 0:END
390 ? :? "ERROR IN LOADING CHARACTER SET":? :CLOSE #3:GOTO
200
400 ? "ERROR: PRINTER OR INTERFACE NOT READY"
410 END
420 ? "ERROR IN OPERATION - PLEASE TRY AGAIN"
430 END
```

How would you like to have an entire issue of MICRO printed in a GOTHIC font? While it might be interesting at first, I am sure that the novelty would wear off very quickly.

Printer Sample

Listing 4

```
10 DIM ML$(300)
11 ML$(1)="hh[illegible]"
12 ML$(92)="[illegible]"
13 ML$(198)="[illegible]"
17 I=0:DIM X$(11),M$(255)
20 X$="h[illegible]"
30 DIM B$(1024):A=ADR(B$)
40 DIM N$(15),NM$(15),MM$(15)
50 GRAPHICS 18
60 POSITION 4,1:? #6;"CUSTOM FONT"
70 POSITION 2,5:? #6;"message printer"
80 POSITION 5,9:? #6;"for epson"
100 FOR K=1 TO 2000:NEXT K
110 ? "Please enter the name of the disk      file which cont
ains the character fontto be used (ex: GOTH.SET)"
120 ? :? "Name of character set:"
130 TRAP 390:NM$="D:":INPUT N$:NM$(LEN(NM$)+1)=N$
140 OPEN #3,4,0,NM$
150 FOR K=0 TO 1023
160 GET #3,B
170 POKE A+K,B
180 NEXT K:CLOSE #3:TRAP 400
200 GRAPHICS 0:POSITION 2,6
210 ? "Please enter then name of the disk      file to be prin
ted. (ex: FILE.BAS)"
220 ? :? "Name of program:"
230 TRAP 350:MM$="D:":INPUT N$:MM$(LEN(MM$)+1)=N$
240 OPEN #3,4,0,MM$
250 INPUT #3,M$
260 TRAP 400
270 Q=LEN(M$):IF LEN(M$)>60 THEN Q=60
280 OPEN #4,8,0,"P":TRAP 420
290 I=0:J=Q*8:IF J>255 THEN I=1:J=J-256
300 PUT #4,27:PUT #4,ASC("K"):PUT #4,J:PUT #4,I
310 FOR K=1 TO Q
320 J=ASC(M$(K,K)):IF J<96 AND J>31 THEN J=J-32
324 IF J<224 AND J>159 THEN J=J-32:GOTO 330
326 IF J<160 AND J>127 THEN J=J+64
330 DUMMY=USR(ADR(ML$),A,J)
340 NEXT K:PUT #4,27:PUT #4,64:CLOSE #4:LPRINT
342 IF Q<LEN(M$) THEN M$=M$(61):GOTO 270
345 TRAP 350:GOTO 250
350 CLOSE #3:GRAPHICS 18
360 POSITION 6,4:? #6;"all done"
370 FOR K=255 TO 0 STEP -1:SOUND 0,K,12,5:SOUND 1,255-K,12,3
:NEXT K:SOUND 1,0,0,0
380 GRAPHICS 0:END
390 ? :? "ERROR IN LOADING CHARACTER SET":? :CLOSE #3:GOTO
110
400 ? "ERROR: PRINTER OR INTERFACE NOT READY"
410 END
420 ? "ERROR IN OPERATION - PLEASE TRY AGAIN"
430 END
```

MICRO

Extended Precision Arithmetic in BASIC

Greater mathematical precision and a way to calculate the lunar-based Jewish Calendar.

by Rolf B. Johannesen

Many common implementations of BASIC in microcomputers today use a binary representation for real numbers which has either 24 or 32 bits for the mantissa and 7 bits for the characteristic. This translates to either 6.1 or 9.5 decimal digits of precision, respectively. Occasionally, greater precision is required: statistical calculations are notable in requiring many extra digits of precision during the intermediate stages of calculations because so many results are derived as the differences between two numbers that are almost equal, so that several of the most significant digits are lost in a single step.

Computer software for processing arithmetic statements never warns that bits have overflowed the mantissa, even though this will inevitably result in loss of precision. However, overflow of bits in the characteristic is always flagged. The program in Listing 1 will test any computer for the length of mantissa in its floating point representation and report it both in terms of bits and equivalent number of decimal digits. The largest integer that can be faithfully represented has a mantissa of all '1' bits, and is equivalent to 2↑M1 if there are M bits in the mantissa. The program will also test the number of bits used for the characteristic. In this case the program will be interrupted at the occurrence of floating point overflow. If your computer does not support the TRAP (or the equivalent ON ERROR GOTO) command in line 210, then line 270 will never be reached. However, the last value of I printed before overflow occurs is the number of bits in the characteristics; the last value of X is a trifle greater than half the largest possible number for that machine. The largest possible number, when there are N bits in the characteristic and M bits in the mantissa, is 2↑(2↑N-1) multiplied by a fraction, very nearly unity, whose numerator contains M '1' bits and whose denominator contains a '1' bit followed by M '0' bits. With a 7-bit characteristic and a 32-bit mantissa, this is very nearly 1.70141183E+38. The hexadecimal representation may vary slightly among BASIC interpreters due to differences in characteristic biasing and in the way the sign bit is expressed. In Microsoft BASIC, the largest possible number has the hex value $FF7FFFFFFF. If your machine lets you alter a number in BASIC's variable table, via monitor or POKEs, you can enter the above value and return to BASIC to print its decimal equivalent.

Extended precision routines in assembly language are perfectly straightforward, rapid and effective; though they tend to get messy for multiplication and especially so for division. Nevertheless, if extensive calculations are required, this method is recommended as being the fastest. It is possible to achieve workable results in BASIC by the procedure given here, in which a large number is broken up and arithmetic operations carried out on the separated parts, with the results combined at the end. If it is necessary for the final result to have greater precision than is available in BASIC, then it will have to be expressed in parts, but this is entirely feasible.

In brief, a large number is expressed as (M*10↑6+T*10↑3+U), where M is the coefficient of the millions place, T the coefficient of the thousands place, and U the units. Obviously, this scheme can be extended to both larger and smaller numbers by choosing the proper powers of ten as multipliers. In