FAST BASIC

BEYOND TRS-80™ BASIC

EORGE A. GRATZER
HOMAS G. GRATZER

Program Disk Available!

FAST BASIC: Beyond TRS-80(TM) BASIC

GEORGE A. GRATZER
University of Manitoba
Winnipeg, Manitoba
Fort Richmond Software Co.

assisted by **THOMAS G. GRATZER**
Fort Richmond Software Co.

1807 175 YEARS OF PUBLISHING 1982

John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto • Singapore

. . . it takes all the running *you* can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that!

—L. Carroll in *Through the Looking Glass*

Dedicated to the people
who did all the running:

to the mathematicians
who worked out the algorithms
utilized in the TRS-80 ROM;

to the programmers
who put it all together;

to the many experts
who, through their writings and correspondence,
taught us the secrets of TRS-80 BASIC.

Publisher: Judy V. Wilson
Editor: Dianne Littwin
Composition and Make-up: Cobb/Dunlop, Inc.

Library of Congress Cataloging in Publication Data
Gratzer, George A.
FAST BASIC
Includes index.
1. Basic (Computer program language) 2. TRS-80 (Computer)—Programming. I. Gratzer, Thomas G. II. Title.
QA76.73.B3G7 001.64'24 81-16207
ISBN 0-471-09849-3 AACR2

Printed in the United States of America

82 83 10 9 8 7 6 5 4 3 2 1

Preface

Promise, large promise,
is the heart of any advertisement.

—S. Johnson in *The Idler*

The TRS-80* computer is equipped with an excellent programming language: TRS-80 BASIC (called Level II BASIC in the Model I and Model III BASIC in the Model III). The design of TRS-80 BASIC emphasizes ease of operation, and so it is not surprising that in some business and game programs the computer is too slow to respond. This book suggests a solution. Write your programs in FAST BASIC.

FAST BASIC is introduced in two stages. The first stage is called CONTROLLED BASIC: the systematic use of PEEK and POKE to gain control over TRS-80 BASIC. To program in CONTROLLED BASIC we must have a good understanding of the structure of TRS-80 BASIC. Many of your wishes for an improved TRS-80 BASIC can come true with a few PEEKs and POKEs *if you know where*. In CONTROLLED BASIC you can send to the printer what is displayed on the screen, merge BASIC programs in a cassette system, and check the status of the printer or the disk.

The presentation of CONTROLLED BASIC is self-contained. Some readers may find CONTROLLED BASIC adequate for the improvements they seek; they should read Parts I and II of this book.

The second stage is FAST BASIC itself: We use TRS-80 BASIC to accomplish what TRS-80 BASIC can do well, but enhance it with machine language routines to overcome some of its shortcomings.

Almost any task FAST BASIC has to do in machine language can be built up from the routines that can be found in the TRS-80 ROM (read-only memory; the ROM is a part of the TRS-80 computer). By learning how to use *fewer than 20 machine language instructions* and the names of about 60 ROM routines, we can write our enhancements.

Examples in this book demonstrate that we can easily make our programs run faster by a factor of 3 to 4 for arithmetic calculations, and by a factor of 1000 for string sorts.

Machine language programming has two main drawbacks: the hundreds of instructions one has to learn and the difficulty of debugging long pro-

*TRS-80 is a trademark of Tandy Corporation.

grams. These problems do not arise in FAST BASIC. In a few minutes any user of TRS-80 BASIC can learn the few machine language instructions needed in FAST BASIC. All tasks, for example, addition, are performed by ROM routines. Debugging is done in BASIC.

In FAST BASIC we select the time-consuming lines, especially FOR NEXT loops, from a BASIC program and replace each by a small group of machine language program lines, mostly subroutine calls to the ROM. These small groups of machine language instructions are presented in this book; almost no machine language debugging is needed.

Finally, we consider enhancing FAST BASIC with machine language routines that are not translations of BASIC program lines. We do this when even FAST BASIC is not fast enough or when we want to implement something that cannot be done in TRS-80 BASIC.

To understand what is happening in the computer's memory, we have to speak the language of the computer; this language is written in binary and hex. In Part I we teach you binary numbers, the shorthand representation of binary numbers—hex—and ways of representing negative numbers. We also discuss the various codes, including ASCII and the codes for the BASIC keywords. If you are familiar with these topics, you can safely skip Part I.

CONTROLLED BASIC and FAST BASIC rely heavily on how the memory is organized. In Part II we learn the organization of the memory and the significance of many memory locations. Dozens of applications are interspersed throughout the discussion.

In Part III we acquire a rudimentary knowledge of the Z-80 microprocessor, along with some machine language instructions.

FAST BASIC is developed in Part IV. We start with the most important ROM routines that carry out arithmetic operations and move variables around. Then we learn how to do FOR NEXT loops. We achieve a great increase in speed (a loop that takes 48 to 99 seconds in BASIC is executed in less than 0.5 second in FAST BASIC).

We then learn how to handle string variables. This means extra work on our part but the reward—a 984 times increase in the speed of our benchmark program—makes the effort worthwhile.

A step-by-step guide is given for turning a BASIC program into FAST BASIC.

To illustrate how FAST BASIC can be enhanced, several new Z-80 instructions are introduced and their utilities demonstrated. We conclude the discussion with a case study. This provides a rather dramatic example of the speedup from BASIC to enhanced FAST BASIC: In TRS-80 BASIC the FIND command of the example program takes about 5 hours and 20 minutes to search the complete (64K) memory; in enhanced FAST BASIC this is accomplished in 1 second. The machine language enhancement that makes this speedup possible is only 83 bytes (about the length of an average BASIC line).

We assume that the reader has some familiarity with TRS-80 BASIC and, preferably, has access to a Model I or Model III TRS-80. To understand Parts

III and IV, the reader should also have an assembler such as the Radio Shack Editor Assembler.

We often refer to the TRS-80 BASIC Manual. For Model I users this is the *Level II BASIC Reference Manual;* for Model III users this is the "BASIC Language Section" of the *TRS-80 Model III Operation and BASIC Language Reference Manual.*

Most of the material covered in this book applies to both the TRS-80 Model I and Model III computers, although the examples are oriented a bit more toward the Model I. The differences between Model I and Model III, as they relate to this book, are discussed in Appendix 9. When reading a chapter, the Model III user should refer to the corresponding section in Appendix 9.

With small modifications FAST BASIC can be used for any computer that uses Microsoft BASIC (TRS-80 Model II, Heath, Sorcerer, Apple II with the Z-80 card, and so on). The ideas of FAST BASIC can be adopted for any microcomputer where enough information is available about the interpreter.

We would like to express our appreciation to H. McCracken and his son, Iain; to the members of the TRS-80 Users Group, in particular to D. Rigg, D. Toews, and D. Wood, who made many helpful suggestions; to Prof. H. Lakser, who was always there to help us out with an explanation when we needed it the most; and to C. Dillon, our developmental editor, for a most professional job.

George A. Gratzer
Thomas G. Gratzer

Contents

Preface vii

PART I **BACKGROUND FOR CONTROLLED BASIC** 1

Chapter 1 **Representing the Contents of Memory Locations** 3
- Binary Numbers 3
- Hex and Base 256 11
- Review and Programming Practice 15

Chapter 2 **Binary Codes** 19
- Negative Numbers 19
- Integers 22
- Characters and Keywords 23
- Epilogue 25
- Review and Programming Practice 27

PART II **CONTROLLED BASIC** 31

Chapter 3 **PEEKing and POKEing** 33
- General Memory Organization 33
- The BASIC Work Area 36
- The Control Blocks 42
- Review and Programming Practice 46

Chapter 4 **Five BASIC Tables** 49
- The BASIC Program 49
- The Simple Variable Table 52
- The Array Variable Table 57
- The String Space 57
- The BASIC Stack 61
- Review and Programming Practice 61

Chapter 5 **Devices** 67
- Buses and Ports 67
- The Video Display 69
- The Keyboard 69
- The Cassette 72
- Device Programming 73
- Review and Programming Practice 77

PART III **BACKGROUND FOR FAST BASIC** 81

Chapter 6 **The Z-80 Register Set** 83
The Registers 83
The Program Counter 86
Assembly Language 87
Review and Programming Practice 88

Chapter 7 **Some Z-80 Instructions** 91
Data Movement 91
Branching, Increment, and Decrement 95
Pseudo Operations and Labels 96
Review and Programming Practice 100

PART IV **FAST BASIC** 105

Chapter 8 **FAST BASIC Introduced** 107
USR: User Subroutine 107
Arithmetic 109
Finding Variables 110
Moving Variables 112
Converting a Single-Precision Addition 114
A Shorthand for Assembly Language Programs 116
Three Examples of Single-Precision Arithmetic 118
Integer and Double-Precision Arithmetic 120
Review and Programming Practice 122

Chapter 9 **Special Techniques: Loops and Strings** 129
The Most Important Loop 130
One More Loop Implementation 134
Strings 136
String Sorting 138
Concatenating Strings 142
Review and Programming Practice 145

Chapter 10 **Step by Step** 151
Preliminary Steps 151
Conversion to SIMPLE BASIC 152
Conversion from SIMPLE BASIC to Machine Language 156
Packing Machine Language Programs into Variables 158
Final Hints 163
Review and Programming Practice 164

Chapter 11 **Enhancing FAST BASIC** 167
Block Transfer and Block Search 167
Other Useful Instructions 170
Extending TRS-80 BASIC 174
Review and Programming Practice 179

Chapter 12 **A Case Study** 187
Analyzing the Program 187
The Machine Language Programs 189
The Converted Program 193

Appendix 1 **Assorted ROM Routines** 199
Math Functions 199
Input/Output 200
Graphics 201
Restarts 202
Expression Evaluation 202

Appendix 2 **The Z-80 Instruction Set** 203
Flags 203
Eight-bit Data Transfer 204
Sixteen-bit Data Transfer 204
Block Transfer and Search 206
Eight-bit Arithmetic and Logical Operations 206
Sixteen-bit Arithmetic Operations 208
Bit Manipulation 208
Jumps, Calls, and Returns 211
Miscellaneous Instructions 212

Appendix 3 **Two Listings of the Z-80 Instruction Set** 213
Alphabetic Listing 214
Numeric Listing 219

Appendix 4 **Binary Arithmetic** 225
Adding and Subtracting Binary Numbers 225
Binary Fractions 227

Appendix 5 **Hex Tables** 231
Hex Addition Table 231
Hex Conversion Table 231

Appendix 6 **Label Listing** 235

Appendix 7 **Additional Reading and Software** 241
Books 241
Magazines 245
Software 245

Appendix 8 **BASIC Program Listings** 247
TUTOR 247
VARPTR 251
VIEW/MOD 254
DISASSEM 257

Appendix 9 **The Model III** 267
Chapter 2 267
Chapter 3 267
Chapter 5 270
Part IV 271
Appendixes 272
Concluding Comments 272

Index 275

PART I
Background for Controlled BASIC

CHAPTER ONE

Representing the Contents of Memory Locations

Practice yourself, for heaven's sake, in little things;
and thence proceed to greater.
—Epictetus in *Discourses*

CONTROLLED BASIC uses the BASIC functions PEEK and POKE to gain control over TRS-80 BASIC. These two BASIC functions can read information in the computer's memory and change it.

In Part I you learn in what form information is stored in the computer's memory. We use this in Part II to make TRS-80 BASIC send to the printer what is displayed on the screen, merge BASIC programs in a cassette system, and so on.

In this chapter we discuss a representation of memory contents: binary numbers. Because binary numbers tend to be long and cumbersome, we also introduce a shorthand: hex.

BINARY NUMBERS

Memory Locations

A memory location in the TRS-80 computer has an "address": the address can be any integer from 0 to 65535.

Turn the computer on and make sure it shows the BASIC READY prompt. To find the contents of a memory location of a given address we use the BASIC PEEK function. For instance,

```
        PRINT PEEK(2423)
```

will return the number 205. If the address, X, is greater than 32767, we should replace X by –1 * (65536 – X). (See the TRS-80 BASIC Manual.) Thus to find the contents of the memory location with address 65535, type PRINT PEEK (-1) since –1* (65536–65535) = –1.

The BASIC PEEK function represents the contents as an integer from 0 to 255. Another way of representing the contents of a memory location is provided by the program TUTOR.

TUTOR

We shall often refer to the program TUTOR. We assume that you either purchased the program on cassette or disk, or copied it in from Appendix 8 for your own use.

RUN the program TUTOR and respond to the question:

```
TYPE THE NUMBER OF THE DESIRED OPTION?_
```

by typing 1 (and press the ENTER key). Answer the prompt

```
TYPE IN ADDRESS (0 - 65535)?_
```

with any address from 0 to 65535; we get a response such as

```
(2423) = 1100 1101
```

indicating that the contents of memory location 2423 are 1100 1101. (To make it easier to read TUTOR puts a space between the first 4 and last 4 digits.) To get the contents of memory location 65535 type 65535; there is no need to do any subtraction when you work with TUTOR.

Even if we have a system with 16K memory (that is, the computer has memory at locations 0 to 32767), we can still request numbers over 32767. The computer does not know how much memory it has. Observe, however, that the contents of such nonexistent high memory are always 1111 1111.

There is no memory at addresses 12288 to 15359. At these locations TUTOR will give us mostly 1111 1111; however, a few locations will yield varying results. This will be explained in Part II.

Examine the contents of memory locations 0 to 12287; these contain Microsoft's BASIC Interpreter, the program that makes the TRS-80 computer run. At locations 15360 to 16385 we are looking at the memory locations containing what is displayed on the screen. Since, as a rule, most of the screen is blank, we usually get 0010 0000, the code for a blank (see Chapter 2).

As the program TUTOR suggests, the contents of a memory location can be represented by a sequence of zeros and ones, an 8-digit "binary number." Thus our understanding of memory locations has to start with binary numbers.

What is really at a memory location? Imagine that a memory location contains eight light bulbs; each bulb can be on or off. We can describe the contents by naming which bulbs are on and which are off, for instance: on on off off off on on off. By writing 1 for on and 0 for off, we can represent the contents by 11000110.

In reality the contents of a memory location are magnetic rather than electric. However, the little magnets also have *two* states that can be represented by a 0 and 1, so again we can represent the magnetic contents by 11000110.

Remember: PEEK represents the contents of a memory location with a number from 0 to 255 whereas TUTOR represents it with an 8-digit binary number!

Binary Numbers

What is a binary number? We know that the number 592 stands for

$$5 \times 100 + 9 \times 10 + 2$$

If we agree to write 10^2 (10 squared or 10 to the power 2) for 100, 10^1 (10 to the power 1) for 10, and 10^0 (10 to the power 0) for 1, then our number becomes

$$5 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$$

This is called the "expanded form" of 592. Numbers expressed in powers of 10 are called "base 10" or "decimal" numbers; 5, 9, and 2 are the "digits."

If we replace base 10 by base 2 and permit only 0 and 1 as digits, then we get "binary numbers." Examples:

$$101 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 4 + 0 + 1 = 5$$
(the decimal number 5)

$$1000 = 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 0 \times 2^0 = 8 + 0 + 0 + 0 = 8$$
(the decimal number 8)

0110 1001 =

$$0 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =$$
$$0 + 64 + 32 + 0 + 8 + 0 + 0 + 1 =$$
105
(the decimal number 105)

In a binary number 0 and 1 are called "binary digits," or "bits" for short. To make the binary numbers easier to read we put a space after every fourth digit (starting from the right).

We saw above that we can represent the contents of a memory location by an eight-character string of zeroes and ones, for example, 0110 1110. We shall regard this as an 8-bit binary number.

An 8-bit binary number is called a "byte," and its bits are numbered from 0 to 7, from right to left. We illustrate this with the byte 0110 1001:

```
bits:   7 6 5 4    3 2 1 0
        ↓ ↓ ↓ ↓    ↓ ↓ ↓ ↓
        0 1 1 0    1 0 0 1
```

Bit 7 often is called the "high-order bit" and bit 0 the "low-order bit." Note that the bit number corresponds to the exponent of 2 when 0110 1001 is written in the expanded form.

Set and Reset

In the early days of electronics binary numbers were used to denote whether an electric switch was set or reset. We still use this terminology. If the bit in a binary number is 1, we say the bit is "set"; if it is 0, we say the bit is "reset." For instance, in the binary number 0110 1001 bit 6 is set and bit 1 is reset.

Converting Binary to Decimal

What is 1100 1101 in decimal? Writing the binary number in expanded form:

$$1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 =$$
$$128 + 64 + 0 + 0 + 8 + 4 + 0 + 1$$
$$= 205$$

So 1100 1101 is 205 in decimal. Surprise! Compare this with PEEK(2423) and the contents of 2423 as given by TUTOR. We conclude that the *PEEK* function *gives the decimal conversion of the binary representation of the contents of a memory location.*

Practice binary to decimal conversion with option 2 of TUTOR.

Table 1.1 lists the first 16 powers of 2. This table is useful in converting from binary to decimal.

TABLE 1.1. Powers of 2

$2^0 =$	1	$2^8 =$	256
$2^1 =$	2	$2^9 =$	512
$2^2 =$	4	$2^{10} =$	1024
$2^3 =$	8	$2^{11} =$	2048
$2^4 =$	16	$2^{12} =$	4096
$2^5 =$	32	$2^{13} =$	8192
$2^6 =$	64	$2^{14} =$	16384
$2^7 =$	128	$2^{15} =$	32768

Binary Arithmetic

Binary addition and multiplication are very easy to learn. There are only 2 digits, so there are only four pairs of numbers to learn to add and multiply.

$$\begin{array}{ll} 0 + 0 = 0 & 0 + 1 = 1 \\ 1 + 0 = 1 & 1 + 1 = 10 \end{array}$$

$$\begin{array}{ll} 0 \times 0 = 0 & 0 \times 1 = 0 \\ 1 \times 0 = 0 & 1 \times 1 = 1 \end{array}$$

It is convenient to represent addition and multiplication in tabular form:

+	0	1
0	0	1
1	1	10

×	0	1
0	0	0
1	0	1

As you see, the only rule to learn is 1 + 1 = 10, which in multidigit addition will take the form: 1 + 1 = 0 and carry 1. Now let us do a multidigit addition:

$$\begin{array}{rr} & 1\,0\,1\,1 \\ + & 1\,1\,1\,1\,0 \\ \hline \end{array} \qquad \begin{array}{rrl} & 1\,1\,1\,1\,0 & \text{Carry line} \\ & 1\,0\,1\,1 & \\ + & 1\,1\,1\,1\,0 & \\ \hline & 1\,0\,1\,0\,0\,1 & \text{Result line} \end{array}$$

We start the addition from the right, that is, in the last column: 1 + 0 = 1 (write 1 in the result line) with no carry (carry 0 is written on the carry line one column to the left); moving one column to the left, 0 + 1 + 1 = 0 (write 0 in the result line), carry 1 (write this in the carry line); next, 1 + 0 + 1 = 0, carry 1, and so on.

We can subtract binary numbers in much the same way as decimal numbers. For a closer look at binary arithmetic turn to Appendix 4. Option 3 of TUTOR will help you practice.

Keep in mind, however, that our primary goal is to recognize and understand the contents of a memory location. We shall always have help (from BASIC or from machine language instructions) in carrying out arithmetic operations.

Logical Operations

In addition, there are also logical operations (also called "Boolean operations") that can be performed on the binary digits: AND, OR, exclusive OR (XOR), and negation (NOT). TRS-80 BASIC users do not have much difficulty learning the logical operations. After all, if we start a BASIC line with

```
IF X = 0 AND Y = 9
```

we use AND exactly as we do in everyday English.

OR and XOR are quite different. XOR (exclusive OR) means one or the other, but not both. A OR B is TRUE if A is TRUE and B is FALSE, or if A is FALSE and B is TRUE, or if both A and B are TRUE. A XOR B is TRUE if A is TRUE and B is FALSE, or if A is FALSE and B is TRUE. If A and B are both TRUE, then A OR B is TRUE while A XOR B is FALSE.

Examples: The lawn is wet if it rains or if it has been watered. Tonight we go to the movies; we go to the Odeon Theater or to the Venus Theater. In the first sentence "or" means OR; in the second, "or" means XOR (we cannot go to both theaters at the same time).

In conditional statements in BASIC (IF condition THEN . . .) we very seldom need XOR. TRS-80 BASIC does not provide XOR; if you find it convenient to use it, substitute

> (condition1 OR condition2) AND NOT (condition1 AND condition2)
> for XOR.

The logical operations are applied to TRUE and FALSE; the result is again TRUE or FALSE. Table 1.2 describes the logical operations.

If you think of 0 as FALSE and 1 as TRUE, Table 1.3 repeats the information of Table 1.2.

To perform AND on 0 and 1: First find the AND table; at the intersection of the 0 line and 1 column is 0; so 0 AND 1 is 0. Similarly, 1 XOR 1 is 0. NOT has only one argument; the NOT of 0 is 1 and the NOT of 1 is 0.

The logical operations can be applied to bytes as well, carrying them out bit by bit:

```
      0100 1110          1100 1010          1110 0001
AND   1000 0010     OR   0001 1110     XOR  1010 1010
      ---------          ---------          ---------
      0000 0010          1101 1110          0100 1011

NOT   0010 1100
      ---------
      1101 0011
```

We carry out the AND in eight steps: first, for bit 7 (0 AND 1 is 0), then for bit 6 (1 AND 0 is 0), and so on.

This may help explain how the logical operations work in BASIC. We have seen in the foregoing example that 0100 1110 AND 1000 0010 = 0000 0010. Now 0100 1110 is 78 decimal; 1000 0010 is 130 decimal; 0000 0010 is 2 decimal. Thus in BASIC the logical expression

```
78 AND 130
```