MARK WILLIAMS COMPANY

Programming Manual

$25

This PDF version of the *XYBASIC Programming Manual* (Rev. 6) was generated in June 2014. An online version is available as a webpage at

nesssoftware.com/home/mwc/doc/xybasic/xybasic.php

This PDF version generally preserves the layout and typography of the original, but it does not preserve its pagenation or line filling. The original contains a comprehensive index, not reproduced here.

XYBASIC PROGRAMMING MANUAL

for process control,
data acquisition and
real time applications
with 8080-based computers

Mark Williams Company
1430 W. Wrightwood Avenue
Chicago, Illinois 60614
Telephone: (312) 472-6659

Revision 6, 2/9/82

Copyright Notice

TABLE OF CONTENTS

PREFACE: 5

HOW TO USE THIS MANUAL: 11

Chapter I: XYBASIC TUTORIAL: 12

Section 1: A Quick Introduction to XYBASIC: 12
- Initialization Dialog: 13
- A First Program: 13

Section 2: Traditional BASIC Commands: 16
- Integer and Extended Versions: 16
- Numbers: 16
- Variables: 17
- LET and PRINT: 18
- RUN: 20
- LIST: 20
- Errors and Correcting Your Program: 21
- NEW: 23
- CLEAR: 23
- GOTO and <control-C>: 24
- CONT: 25
- Control Characters: 26
- INPUT: 26
- REM and ': 28
- IF / THEN: 29
- STOP: 30
- END: 31
- GOSUB and RETURN: 31
- READ, DATA and RESTORE: 33
- FOR and NEXT: 35
- ON / GOTO and ON / GOSUB: 40
- DIM: 41
- Multiple Commands Per Line: 43

Section 3: Numeric Formulas: 45
- Arithmetic Operators: 45
- Conversions: 46
- Relations: 47
- ABS: 48
- SGN: 48
- MOD: 48
- SQR: 49
- LOG: 49
- EXP: 50

SIN, COS, TAN and ATN: 50
INT: 51
RND and RANDOMIZE: 51
FRE: 54
UNS: 55
DEF FN: 56
Variable Types: 58

Section 4: Strings: 60
Quoted Strings: 60
String Variables: 60
LEN: 61
Concatenation (+): 62
LEFT$, RIGHT$ and MID$: 63
CHR$: 65
ASC: 65
Relations: 65
String Arrays: 66
String Functions: 67
INSTR: 68
GET$: 69
STR$ and VAL: 70
CLEAR and FRE$: 71

Section 5: PRINT Related Commands: 73
SPC: 73
TAB: 73
POS: 74
CHR$: 75
NULL: 75

Section 6: Input/Output, Saving and Loading Programs: 77
ASSIGN: 77
IOBYTE: 77
SAVE and LOAD: 78

Section 7: Debugging: 81
TRACE and UNTRACE: 81
BREAK and UNBREAK: 83

Section 8: Bit Manipulation and Control Features: 88
Integer Representations: 88
TEST: 88
Logical Operators: 89
SET and RESET: 91
ROTATE, RSHIFT and LSHIFT: 92
BCD and BIN: 94
HEX$, OCT$ and BIN$: 95
MSBYTE, LSBYTE and JOIN: 96

GET: 97
DELAY: 99
TIME: 100

Section 9: Machine Control Functions: 101
OUT: 101
IN: 102
PEEK: 102
POKE: 103
SENSE: 104
WAIT: 105

Section 10: Interrupts: 107
ENABLE: 107
DISABLE: 108

Section 11: Machine Language Linkage: 110
CALL: 110
SCALL: 114

Section 12: ROMSQuared Features: 117
Working Space: 117
MOVE: 117
EXEC: 118
FIRST and LAST: 120
Default Initialization Options: 120

Section 13: Errors: 121
TRAP and UNTRAP: 121
Error Types: 122

Section 14: Editing Commands: 126
AUTO: 126
DELETE: 127
EDIT: 128
RENUM: 130

Section 15: CP/M Sequential Disk Commands: 132
Filenames: 132
OPEN: 133
CLOSE: 133
PRINT: 134
MARGIN: 134
INPUT: 135
LINPUT: 135
EOF: 136
DIR: 137
SCRATCH: 137
CLEAR: 137

Chapter II: VERSION DIFFERENCES: 139

Section 1: CP/M Version: 139

Section 2: ISIS-II Version: 140

Section 3: Custom I/O Version: 140
Device Driver Locations: 142
Sample I/O Patch: 143
SAVEd Program Format: 145
Saving and Loading Under Operating Systems: 147

Section 4: INTEL SBC Series Versions: 149

Section 5: AMD 9511 Floating Point Version: 150

Section 6: XYBASIC Compiler: 151
Compiler Operation: 151
Object File Execution: 152

Section 7: XYBASIC Runtime Module: 153

Section 8: Customized OEM Versions: 154

Chapter III: SHORT FORM DESCRIPTION: 155
Section 1: Conventions: 155
Section 2: Direct Commands: 156
Section 3: Traditional BASIC Commands: 156
Section 4: Numeric Formulas: 160
Section 5: String Formulas: 165
Section 6: Input / Output Commands: 167
Section 7: Control Commands: 168
Section 8: Debugging Commands: 170
Section 9: ROMSQuared Commands: 171
Section 10: Editing Commands: 171
Section 11: CP/M Sequential Disk Commands: 172
Section 12: Special Characters: 174

Appendix 1: Initialization Dialog: 177
Appendix 2: Speed and Space Hints: 178
Appendix 3: Reserved Word List: 180
Appendix 4: Character Set: 181
Appendix 5: ASCII Character Equivalents: 183

INDEX: 187

USER REACTION REPORT: 188

PREFACE

Congratulations! You are about to discover the unique and powerful properties of XYBASIC, the only BASIC interpreter specifically designed for process control, data acquisition and real time applications with 8080-based microcomputer systems.

Until now, the 8080 (like other computers) could be programmed for most applications of this type only in assembly language. But even experienced professional programmers admit that assembly language programming is awkward and time consuming, and therefore expensive. To develop a program in assembly language you must (1) write it, (2) load the editor, (3) type in the program, (4) load the assembler, (5) assemble the program, (6) load the loader, (7) load the program, (8) execute the program, and (9) debug the program. And every time you find a bug you must repeat all the steps.

The programming language BASIC, designed for people with no previous knowledge of computers as well as for the experienced programmer, makes programs much easier to write and understand. Since the 8080 actually executes machine language, though, an interpreter is required to translate BASIC programs; the interpreter permits the computer to execute the given BASIC commands immediately.

Others have implemented BASIC on the 8080, but until now there have been no BASIC interpreters specifically designed for process control applications. Now this vital need has been filled with the introduction of the powerful XYBASIC interpreter. Almost anyone can use it to create process control programs quickly and easily -- in as little as one tenth of the time it used to take!

Here's how it works

All you need is the XYBASIC interpreter, available on floppy disk, paper tape or EPROM, plus of course an 8080-based computer such as the 8080, Z-80 or 8085 with 14K of memory (8K for Integer XYBASIC) and a console. After you load XYBASIC into the computer in as little as five seconds and go through a short initialization dialog, you just type in your program and execute it. When you find a bug you can change the program and execute it again without loading any other programs.

Look at all these features

Of course XYBASIC includes the standard BASIC commands, such as READ, DATA, FOR, NEXT, GOTO, GOSUB, RETURN, ON / GOTO, ON / GOSUB, PRINT and CONT... plus arrays, user-definable functions and others. XYBASIC lets you use fast integer arithmetic, and Extended XYBASIC gives you the full power of floating point and strings.

But XYBASIC has many other features specifically designed for process control, giving you the power of assembly language programming with the ease of programming in BASIC.

You can examine and modify any locations in your computer's memory with PEEK and POKE. The IN and OUT commands let you perform machine-level input and output. You can use XYBASIC to look at an individual bit on an input port with SENSE, or to test a particular bit of a variable with TEST. You can wait for a particular event to occur with WAIT. And XYBASIC even lets you link programs with assembly language routines, using CALL and SCALL; if you already have assembly programs, you can use them from within a XYBASIC program and pass information to them in a natural way.

XYBASIC also lets you perform bit manipulation functions like ROTATE and SHIFT which were previously possible only in assembly language. And you can concatenate variables, split variables into 8-bit values, convert between binary and BCD representations, and perform logical operations such as AND, OR, XOR and NOT.

Software interrupt = much more power

XYBASIC offers a software interrupt feature which allows concurrent processing, effectively multiplying the power of your computer to a substantial degree. Suppose for example that you are conducting a chemical experiment: start with 250 cc of solution A, apply heat, and heat to 75 degrees C. With XYBASIC you can ENABLE an interrupt which will continuously monitor a digital thermometer hooked up to the experimental apparatus. You can let XYBASIC continue processing data from the experiment, and automatically shut down the heat when the solution reaches the desired temperature!

You can use the DELAY command to incorporate real time delays into a program -- it's like having a real time clock in your computer.

Software test instrument

If you are not certain whether your computer is operating correctly, or if you add new hardware to your system, you would normally write an assembly language test program. But a much better way to test is to use XYBASIC in conjunction with your usual test instruments, like an oscilloscope or voltmeter. Write a simple XYBASIC program, and use the interactive features of XYBASIC to help pinpoint problems. Comparing assembly language programming with XYBASIC is like comparing a manual typewriter with a word processor -- both can produce a beautiful manuscript, but there's a world of difference in the amount of effort needed.

Powerful direct mode

Here's another difference between assembly language and XYBASIC. Without the involved process of editing, assembling, loading and running an assembly language program, you can use the interactive capabilities of XYBASIC's direct mode to print the information you want instantly. For example, you can find the value on an input port or put a value on an output port directly.

Unique one-step debugging

Since XYBASIC includes an editor, you do not need to load it separately; therefore you can change your program instantly. The additional steps needed when using a compiler or assembler or debugger are eliminated.

To find and correct errors, just use one of the special debugging commands of XYBASIC. TRACE lets you trace program execution, showing you every line that gets executed and printing the name and value of any variables changed. BREAK lets you set breakpoints on line numbers or variables. If you set a line number break, XYBASIC prints the line number whenever it is executed, and then either returns control to you or continues with the next step in your program. If you set a variable break, XYBASIC prints the variable's name and new value whenever it is changed.

Under normal conditions most programmers spend about 40% of their time writing programs and 60% of their time discovering why the programs do not work. Using XYBASIC greatly reduces debugging time, since the execution of a program can be examined line by line; you can pinpoint problems and correct them immediately.

Available with editing commands

XYBASIC is available in versions which include a line-oriented editor. These versions allow programs to be modified without retyping entire lines. The AUTO command lets you enter a program without typing line numbers. The EDIT command lets you edit a line of the program. And the RENUM command renumbers your program, or a section of your program, automatically!

ROM SQuared features

Another unique advantage to XYBASIC is its ability to allow several user programs in memory simultaneously, in either RAM or ROM. You can just use the EXEC command to switch from one program to another. Once a program is debugged, you can burn and execute it from PROM. You can even build stand-alone systems which execute a specific program as soon as you turn on your computer.

Up to ten times faster

The maximum output of programmers, whether in assembly language or BASIC, is usually about ten lines of debugged program per hour, even for experienced programmers. Since each line of XYBASIC corresponds to about ten lines of assembly language, you can create debugged programs in as little as a tenth of the time assembly language would require.

Human engineered, thoroughly tested

The careful human engineering of the XYBASIC interpreter assures that anyone -- even with a minimum of programming experience -- can use it. Experienced programmers will find it well worth the investment; why continue to do things the hard way?

XYBASIC is a quality software product and has been thoroughly tested. And of course it is fully supported.

XYBASIC pays for itself

The XYBASIC interpreter can quickly pay for itself by saving substantial time and effort and increasing the usefulness of your 8080-based system. To summarize XYBASIC's benefits and features, it lets you:

write programs faster and easier...
write more working programs in a given time...
debug programs instantly...

conduct fast tests of hardware...
examine or modify memory locations...
input and output values...
direct your computer to wait a specified time...
perform bit manipulation...
link programs with assembly language routines...
have several programs in memory simultaneously...
store programs in either RAM or ROM...
switch between programs with a simple command...
convert between binary and BCD representations...
perform logical operations...
incorporate concurrent processing routines...
and examine any input or output port directly.

Available for standard systems

You can choose a version of XYBASIC which fits your needs precisely. For users with memory constraints, Integer XYBASIC offers powerful control features without requiring much memory space. For others, Extended XYBASIC retains the speed of Integer XYBASIC's integer arithmetic and the power of its control features, but offers the additional flexibility of floating point arithmetic and string manipulation.

XYBASIC is available in versions for the CP/M and ISIS-II disk operating systems and in versions for the Intellec 8/Mod 80, Intellec MDS, and INTEL SEC 80 series computer systems. You can also get a version which performs floating point operations faster by using the AMD 9511 floating point chip.

For users with CP/M systems, XYBASIC is available with additional commands which allow reading and writing of disk data files. These features allow easy handling of large data bases.

Another version is easily modified for special systems, letting you patch in I/O drivers for the specific hardware of your computer system. With this version you can have XYBASIC in ROM, ready to use as soon as you turn on your computer. And XYBASIC's unique ROM SQuared features let you store XYBASIC programs in ROM too, allowing you to build stand-alone systems with ease.

OEM modifications

In addition to the versions described above, Mark Williams Company can build a version of XYBASIC which meets your OEM requirements precisely. Call us today to discuss your needs.

Try it!

If you already know BASIC, you can start programming in XYBASIC right away. If not, this manual will teach you how to write XYBASIC programs. We think you will like XYBASIC, and we would greatly appreciate your comments.

HOW TO USE THIS MANUAL

This manual is designed both for the novice and for the experienced programmer. If you have little or no programming experience, you should start with Chapter I, a tutorial to help you learn XYBASIC simply and painlessly. By actually typing in the many examples and experimenting with XYBASIC you will learn how to interact with your computer, and soon you will be writing your own programs. Once you know XYBASIC you can use the concise descriptions in Chapter III to refresh your memory about specific features.

A few places in Chapter I refer to material introduced later. These clearly marked references are provided to make the manual more complete as a reference source on XYBASIC, and they can be disregarded on first reading. Some sections refer to features available in some versions of XYBASIC but not in others. These references should be ignored if they do not apply to your version.

If you are an experienced programmer already familiar with BASIC, you may want to skip much of the detail in Chapter I (especially Sections 1 through 5). Begin instead by reading Chapter III, a concise description of XYBASIC's features. To learn more about a command you can refer back to the examples and description in Chapter I when necessary.

Chapter II describes the available versions of XYBASIC. If you bought the manual before buying XYBASIC, you can use it to find out which version you want. Once you have XYBASIC it will give you necessary information about your version.

Chapter I: XYBASIC TUTORIAL

This tutorial chapter will teach you to use XYBASIC. You will learn a great deal by actually running the sample programs on your computer. Then you should try writing similar programs! By interacting with XYBASIC you will quickly learn how XYBASIC works and how to write your own programs.

The sample programs appear in their entirety, just as you will see them on your console. To make them stand out from the manual text, they appear in a different typeface. Both what you type and what XYBASIC types is given; of course you should not type in the lines typed by XYBASIC.

Section 1: A Quick Introduction to XYBASIC

XYBASIC is a versatile and powerful computer language which lets you perform complex calculations and solve difficult realtime control problems in either of two modes.

PROGRAM MODE:
In program mode you can prepare a complex series of numbered commands, called a program, and then execute it. XYBASIC's powerful debugging features make it easy for you to find mistakes and correct your program quickly.

DIRECT MODE:
In direct mode you just type a command and XYBASIC executes it immediately. The dozens of available commands let you do input and output, examine the contents of memory, list programs, and perform countless other tasks.

It's that simple! Using XYBASIC requires no prior computer experience or special mathematical knowledge. To see just how easy XYBASIC is to use, type the following lines (each followed by <carriage return>) after loading XYBASIC.

```
You type:          XYBASIC responds:
PRINT 5 + 4         9
                   OK
LET X = 10         OK
LET Y = 20         OK
PRINT X + Y         30
                   OK
```

The OK typed by XYBASIC is a prompt, its way of telling you it is ready for another command. When used like this, XYBASIC acts just like a pocket calculator, but it has much more power. You can use XYBASIC to monitor and control anything from machine tools to chemical experiments to oil pipelines. And XYBASIC's powerful features let you write programs faster and get them running correctly in less time, so XYBASIC can increase your productivity.

Initialization Dialog

Before you can use XYBASIC you must load it into your computer's memory; to learn how see Chapter II. After you load and start it, XYBASIC leads you through an initialization dialog to learn about your particular computer system. First XYBASIC will say

```
XYBASIC {version} REV n.m
COPYRIGHT 1978, 1979, 1980 BY MARK WILLIAMS COMPANY, CHICAGO
```

to tell you which {version} (such as CP/M or ISIS-II) and which revision of XYBASIC you are using. Then it will ask

```
WIDTH?
```

You should type a decimal number followed by a <carriage return> (the key labelled RETURN or CR) to tell XYBASIC the width of your terminal. If you just type a <carriage return>, XYBASIC assumes your terminal to be 80 columns wide. Next XYBASIC will ask

```
END OF MEMORY?
```

You should respond with the address (in decimal) of the highest usable RAM location in your computer's memory, again followed by a <carriage return>. If you just type a <carriage return>, XYBASIC automatically finds the highest usable address. In the CP/M and ISIS-II versions it finds the highest address from the operating system, and in Custom I/O versions it searches memory to find the highest RAM address. Finally XYBASIC says

```
xxxxx BYTES FREE
OK
```

Here xxxxx gives the number of bytes of memory which remain free for program and variable storage; of course this number will depend on the amount of memory available in your computer.

A First Program

A program is nothing more than a numbered series of commands or instructions to the computer. Before typing in a new program you should erase any existing old program by typing

```
NEW
```

followed by a <carriage return>; you must type a <carriage return> to terminate each line you type. XYBASIC responds with its OK prompt to tell you it has done the command. Now you can type in your first program; don't worry yet about how it works. This program converts numbers to binary representation. Type:

```
10 INPUT "NUMBER TO CONVERT?" N
20 FOR I = 15 TO 0 STEP -1
30 PRINT TEST (N, I);
40 NEXT I
50 PRINT
60 GOTO 10
```

Now check that you typed everything correctly by typing

```
LIST
```

XYBASIC will then LIST your program, followed by an OK prompt. If you made a mistake, retype the bad line and it will be replaced. Now you can run your program by typing

```
RUN
```

Your program will ask you for a number by typing

```
NUMBER TO CONVERT?
```

You type 5 followed by <carriage return>, and your program will say

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
NUMBER TO CONVERT?
```

Type some more values. When you want to stop, hold down the control key (sometimes labelled CNTL or CTRL or CTL) and type C at the same time. XYBASIC will say

```
^C
BREAK IN LINE 10
OK
```

Congratulations -- you have just run a program!

You probably noticed that every line of your program has a line number, and that line numbers are in ascending order (the lowest are first and the highest last). XYBASIC orders lines automatically, so line 10 will be LISTed before line 20 even if you type line 20 before line 10. You also may have noticed that the program has gaps between line numbers. The lines could have been numbered 1, 2, 3, 4, 5 and 6, but then if you decided to add another line between lines 2 and 3 you would have to retype and renumber lines 3, 4, 5 and 6, rather than

just adding line 25.

When you typed NEW, LIST and RUN you were giving XYBASIC commands in direct mode. Direct commands have no line numbers, and XYBASIC executes them as soon as the <carriage return> is typed. When you typed in the program, you were using program mode. Program lines have line numbers, and XYBASIC adds them to the current program but does not execute them until you type RUN.

Now let's look at your first program more closely to see how it works. After you type RUN, XYBASIC executes the numbered commands one at a time in the order they appear, unless a command tells it to do otherwise. Line 10 is an INPUT command, which gets information from your terminal. It gives the variable named N the value you type. XYBASIC may change the value of a variable, but its name always remains the same. Lines 20 and 40 create a FOR loop, causing the commands in between (namely line 30) to be executed repeatedly. Line 30 uses the PRINT command to print information on your console. The value it prints is computed by TEST, a function found only in XYBASIC which enables you to look at the value of a single bit of a variable. Line 60 is a GOTO command, telling XYBASIC to go back and execute line 10 again instead of executing the following command.

The sections which follow will teach you how to use XYBASIC. You will learn about the commands used in the program above in much greater detail, and you will learn about many other commands as well. Have fun!

Section 2: Traditional BASIC Commands

A command is a word which instructs XYBASIC to perform a specific action. This section describes commands which are common to all BASICs. Before you learn about commands, though, you will be introduced to the differences between the Integer and Extended versions of XYBASIC, and to two fundamental concepts: numbers and variables.

Integer and Extended Versions

XYBASIC is available in two fundamentally different versions, and this manual describes both. In Integer XYBASIC the numbers you may use must be integers in the range -32768 to 32767. Integer XYBASIC is ideal for users with memory size constraints, as well as for those who require fast arithmetic operations and control features but do not need floating point arithmetic.

Extended XYBASIC retains the speed of Integer XYBASIC's fast integer arithmetic and its powerful control features. In addition, it lets you use floating point numbers in the approximate range 1.7 * 10^-38 to 1.7 * 10^+38, with a precision of more than six decimal places. Extended XYBASIC also gives you full floating point functions and extensive string manipulation facilities.

Most of the examples in this manual will work in either the Integer or Extended version of XYBASIC. Examples which apply to only one version are always noted clearly. Sections of the manual which apply only to Extended XYBASIC are marked similarly.

Numbers

In either version of XYBASIC, you can specify integer numbers between -32768 and 32767 in the decimal representation you normally use. You can also specify integers in hexadecimal and binary representations. & indicates binary numbers, so &011 is a binary number (equal to 3 decimal). # indicates hexadecimal numbers, so #1FE is a hex number (equal to 510 decimal).

In Extended XYBASIC you may specify numbers in several additional ways. You can give a series of decimal digits, with or without a decimal point; for example, 3.14159 and 1000000 are legal numbers. You can precede the number by an optional + or - sign; for example, -123.1156 and +2.71828 are also legal numbers. You can also follow the number with a decimal exponent. The exponent consists of the letter E, an optional + or - sign, and decimal digits, and it specifies the power of 10 by which the number is multiplied to obtain its value. For example, 3E-4 is a number with the value .0003, and 1.5E6 is .a number with the value 1500000.

Variables

XYBASIC lets you perform simple computations which only use constants. For more complicated tasks, though, you will need to use variables. You can think of a variable as a box which can contain any arbitrary value; if you have used a calculator with memory, think of it as a memory register. You refer to a variable by its name, which can be arbitrarily long, although only its first eight characters are remembered. The first character must be a letter and other characters must be letters or digits, but some combinations of letters cannot be used. A variable name cannot be the same as or contain a reserved word (a function or command name -- see the list in Appendix 3). The following are legal XYBASIC variable names.

```
A
B3
BOY
DOG
LENGTHYNAME (identical to LENGTHYN)
```

The following are NOT legal variable names.

```
1A      first character not a letter
A#      character not a letter or digit
RND     reserved word
PORT    contains the reserved word OR
```

XYBASIC's long variable names can be very helpful to you. You should always try to choose meaningful variable names. If a variable name reminds you of its use, you are less likely to use it incorrectly in a program.

XYBASIC initializes all numeric variables to zero. That is, a value of 0 is put in each variable's "box" when the variable is first encountered.

XYBASIC allows three different types of variables: floating point, integer, and string. A variable name may optionally end in !, %, or $. If it ends in a letter or digit, or in the character !, it represents a floating point variable. The value stored in a floating point variable may be any number. If the variable name ends in the character %, it represents an integer variable. The value stored in an integer variable must be an integer in the range -32768 to 32767. If the variable name ends in the character $, it represents a string variable. The value stored in a string variable is a sequence of 0 to 255 characters, as described in Section 4 below. Additional information about variable types in Extended XYBASIC is given in Section 3.

LET and PRINT

The LET command is probably the most important in XYBASIC, as it allows you to give a value to a variable. If you type

```
LET X = 14
```

then the value of the variable X becomes 14. To see the value of a variable, you can just ask XYBASIC to PRINT it:

```
PRINT X
 14
OK
The variable on the left hand side of the equal sign in a LET command can b

LET TEMP = 45
LET PETS = DOGS + CATS
```

In the second example, the plus sign (+) represents addition, just as in normal mathematical notation. The command adds the values of the variables DOGS and CATS, and places the result in the variable PETS.

The word LET in a LET command is optional, so you can just type

```
PETS = DOGS + CATS
```

The PRINT command allows XYBASIC to communicate with you. You can PRINT numbers as well as variables:

```
PRINT 3
 3
OK
```

Alternatively you may use the abbreviation ? instead of PRINT:

```
?5
 5
OK
```

You can also PRINT the values of formulas.

Thus:

```
X = 3
OK
Y = 5
OK
```

```
PRINT X+Y, X-Y, X*Y, 2*(Y-X)
 8            -2            15             4
OK
```

The minus sign (-) is used in formulas to indicate subtraction or negation, * is used to represent multiplication, and a comma separates PRINT items into columns fourteen (eight in Integer XYBASIC) spaces wide. Now try this instead:

```
PRINT X+Y, X-Y, X*Y, 2*(Y-X)
 8 -2  15  4
OK
```

You can see the semicolon leaves only one or two spaces between values.

In Extended XYBASIC, numbers with magnitudes in the range .01 to 999999 are PRINTed as 1 to 6 decimal digits, with decimal point and sign where appropriate.

```
PRINT 1.5E2, -1.5E1, 1.5E0, 1.5E-1
 150             -15             1.5             .15
OK
```

Numbers with magnitudes less than .01 or greater than 999999 are PRINTed as a decimal fraction in the range 1 to 9.99999, followed by an exponent consisting of the letter E, a sign, and two decimal digits.

```
PRINT 1234567, -.0015
 1.23457E+06  -1.5E-03
OK
```

You can also PRINT messages. Try this example:

```
PRINT "THE SUM OF X AND Y IS"; X+Y
THE SUM OF X AND Y IS 8
OK
```

You need the quote marks (" ") around the message to allow XYBASIC to distinguish between message and program. If your console lets you use both upper and lower case alphabetic characters, you can use lower case within quoted strings. If you use lower case letters outside of quoted strings, XYBASIC automatically converts them to upper case.

On most consoles you can beep or ring a bell to audibly prompt the user of your program by PRINTing a quoted <control-G>. <control-G> is the character typed by simultaneously depressing the control (sometimes labelled CNTR or CTRL) and G keys; angle brackets (< >) are used throughout this manual to indicate nonprinting characters. XYBASIC echoes control characters by typing ^ followed by the character.

In Extended XYBASIC you can use LET to assign strings to string variables, and you can use PRINT to print any string, not just quoted strings. A string is PRINTed in the obvious way: each character of the string is simply PRINTed successively. More information about strings is given in Section 4.

In CP/M versions of XYBASIC with sequential disk operations, PRINT is also used to send information to disk data files. More information about this use of PRINT is given in Section 15.

RUN

In the above examples you typed commands in direct mode and XYBASIC executed them immediately. For more complicated examples, though, you will want to enter a sequence of commands, called a program, as a series of numbered lines. Any line preceded by a line number (from 1 to 65535) is not executed, but rather is added to the current program; this is called program (indirect) mode. XYBASIC does not prompt you with OK after you type a program mode line. To execute the current program, starting at the lowest line number present, you just type RUN. For example:

```
IO LET X = 3 * 5
20 LET Y = 5 * 5
30 PRINT X, Y, X * Y
RUN
 15             25            375
OK
```

You can also RUN starting from any line in a program, by giving the desired starting line number after RUN. Continuing with the above example:

```
RUN 20
 O             25              0
OK
```

Since line 10 was not executed, the value of X (and of X * Y) is 0 when XYBASIC executes the PRINT command in line 30.

The RUN command can only be used in direct mode. If you try to use it in a program, an II (Illegal Indirect) error will occur; as explained below.

LIST

To see the current program you type LIST. Try it with the above example:

```
LIST
10 LET X = 3 * 5
20 LET Y = 5 * 5
```

```
30 PRINT X, Y, X * Y
OK
```

Sometimes you will want to LIST only certain sections of a program. If you type LIST 10, 20 then all lines from 10 through 20 are LISTed:

```
LIST 10, 20
10 LET X = 3 * 5
20 LET Y = 5 * 5
OK
```

Similarly, LIST 20 lists all lines starting from line 20:

```
LIST 20
20 LET Y = 5 * 5
30 PRINT X, Y, X * Y
OK
```

And LIST ,10 lists all lines through line 10:

```
LIST ,10
10 LET X = 3 * 5
OK
```

You can abort a long LISTing by typing <control-C>, suppress part of it with <control-O>, or print it on your printer with <control-P>; these options are explained below.

Errors and Correcting Your Program

Nobody's perfect, so you will sometimes type lines which XYBASIC does not understand. It will then respond with an error message, described in detail in Section 13. The most common error message is SN ERROR, which stands for SyNtax error and means XYBASIC simply could not understand the line you gave it. XYBASIC tries to help you find your mistake by typing the line it could not understand, with a <linefeed> (that is, with the line split in two) roughly where the error occurred. Note that the <linefeed> is just a guide, and will not always point out your error. For example:

```
10 PRANT "HI"
RUN
SN ERROR: 10 PRANT
                  "HI"
OK
```

Since you typed PRANT instead of PRINT, XYBASIC did not understand line 10 and typed an error message. To fix your program just retype the line:

```
10 PRINT "HI"
RUN
HI
OK
```

If you type the wrong character in a line, you can erase it with the <rubout> key (labelled RUBOUT or RUB or DEL), written <rub> in the examples below. Any characters that you erase are echoed by XYBASIC to the console within slashes (/ and \), and then you can type the correct character. Thus:

```
10 PRA<rub>/A\INT "R<rub>/R\HI"
LIST
10 PRINT "HI"
OK
```

You can also erase characters by typing <control-H>. When you do so, XYBASIC echoes a <control-H> to the console rather than echoing the erased characters within slashes. On many consoles (including most CRT terminals), <control-H> backspaces the cursor, allowing you to type corrections "over" your mistakes.

If you make and correct several errors in one line it often becomes unreadable; then you can type <control-R> to have XYBASIC retype the line.

```
10 THA<rub>/A\IS IS AT<rub>/T\ TIA<rub>/A<rub>IEST^R
10 THIS IS A TEST
```

If you want XYBASIC to forget the line you are typing, type <control-U> and XYBASIC will ignore the line.

```
10 MUMBLE^U
LIST
10 THIS IS A TEST
OK
```

You can see that XYBASIC ignored the line with MUMBLE and LISTed the previous line 10 instead.

If you want to erase a line, just type its line number immediately followed by a <carriage return>.

```
10 MUMBLE
LIST
10 MUMBLE
OK
10
LIST

OK
```

NEW

If you have been using one program but are ready to enter another, you should use the NEW command to erase the old program entirely, as the following example demonstrates.

```
10 PRINT 1,2,3
LIST
10 PRINT 1,2,3
NEW
OK
LIST
OK
```

In addition to erasing your old program, NEW returns you to UNTRACE and TRAP modes (explained in Sections 7 and 13 below) and disables ENABLEd interrupts (explained in Section 10 below). Like RUN, NEW is legal only in direct mode; an II (Illegal Indirect) error will occur if you use it in a program.

CLEAR

You can use the CLEAR command to reset all variables to 0 without changing your program. For example:

```
X = 1
OK
CLEAR
OK
PRINT X
 0
OK
```

Whenever XYBASIC executes a NEW command it also CLEARs your variables automatically.

In Extended XYBASIC, the CLEAR command is also used to change the amount of space available for string storage. This use of CLEAR is explained in Section 14 below.

In CP/M versions of XYBASIC with sequential disk operations, CLEAR is also used to tell XYBASIC how may disk data files you need to use simultaneously. More information about this use of CLEAR is given in Section 15.

GOTO and <control-C>

Most program commands are executed sequentially; that is, XYBASIC executes the command with the lowest line number when a RUN is issued and then executes the following lines in increasing order. But you often want to break up this sequential flow, for example by creating a loop in your program to execute a number of commands repeatedly. The GOTO command tells XYBASIC to execute a specified line instead of the next sequential command line. If you say

```
GOTO 10
```

then XYBASIC will next execute the command at line 10. The following program shows how the GOTO command allows you to write a program where the same instructions are executed repeatedly.

```
NEW
OK
10 PRINT "A";
15 I = I + 1
20 GOTO 10
RUN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

This program is an infinite loop -- it will never stop unless you interrupt it. Now depress the control (CNTL or CTRL or CNT) key and the C key simultaneously, called <control-C>, and XYBASIC will respond

```
^C
BREAK IN LINE 10                          (or perhaps 15 or 20)
OK
```

Here lines 10, 15 and 20 are executed repeatedly, and are therefore called a loop; line 20 created the loop by telling XYBASIC to execute line 10 again. Loops are one of the most essential and powerful constructs in any programming language.

The next example calculates the squares of consecutive whole numbers; again, type <control-C> to stop it.

```
NEW
OK
10 I = 0
20 PRINT I, I * I
30 I = I + 1
40 GOTO 20
RUN
 0             0
```

```
 1             1
 2             4
 3             9
 4             16
 5             25
 6             36
^C
BREAK AT LINE 20
OK
```

If your program tries to GOTO a line number which does not exist, a US (Undefined Statement) error will occur.

CONT

In the example above you learned how to interrupt program execution with <control-C>. After typing <control-C> you can ask XYBASIC to LIST your program or to PRINT the values of variables. For example,

```
NEW
OK
10 PRINT "A";
1S I = I + 1
20 GOTO 10
RUN
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA^C
BREAK AT LINE 15
OK
PRINT I
 106

OK
```

Then you can CONTinue execution from where it was interrupted:

```
CONT
AAAAAAAAAAAAAAAAAAAAAAAAAA^C
BREAK AT LINE 10
OK
```

and to exit type <control-C> again. Like NEW and RUN, CONT is legal only in direct mode; an II (Illegal Indirect) error occurs if you use it in a program.

Sometimes it is impossible for XYBASIC to CONTinue, for example if you <control-C> out of a program and then edit it, or if you try to continue after an error. Under such circumstances a CN (can't CoNtinue) error will occur. Try it with the program above:

```
RUN
AAAAAAAAAAAAAAAAAAAA^C
BREAK AT LINE 15
OK
15 PRINT "B";
CONT

CN ERROR: CONT

OK
```

Control Characters

XYBASIC has several additional characters which control program execution. Typing <control-S> stops execution completely and waits until you type <control-Q> or another <control-S> before resuming. This is useful when you want to examine part of a long LISTing or a TRACE (described in Section 7) on a CRT console.

On the other hand, <control-O> actually suppresses console output; execution of your program continues, but output is not sent to the console until the next <control-O> is typed, or until an error occurs or the program returns to direct mode. By repeatedly toggling <control-O> you can watch a TRACE of your program intermittently without the time-consuming delay of writing all the trace information on the console.

If you have a lineprinter as the LST device of your computer system (as described in Section 6 below), you can have output printed on it by typing <control-P>. As with <control-O>, typing another <control-P> cancels the effect of the first.

If you are finished with XYBASIC and want to return to the operating system of your computer, just type <control-B> (for Bye) and XYBASIC will return you to the system.

INPUT

The INPUT statement allows your program to get data from the console while running, so the program can request information and then use it. You might for example say

```
10 INPUT A
```

When the INPUT statement is executed it prompts you by printing a '?' (question mark) on the console, and then waits for you to type in a value. After you enter the value and type a <carriage return> (so XYBASIC knows you are done), the

value you typed is assigned to the variable A. To see INPUT work try the following program, which prints the square of the input value.

```
NEW
OK
10 INPUT A
20 PRINT A * A
30 GOTO 10
RUN
? 10
 100
? 99
 9801
? 5
 25
? ^C
BREAK AT LINE 10
OK
```

To exit from this program type <control-C>and you will return to direct mode.

INPUT allows you to prompt the user with a message instead of just a question mark. Try changing the above program as follows:

```
10 INPUT "NUMBER TO SQUARE" A
RUN
NUMBER TO SQUARE? 15
 225
NUMBER T0 SQUARE? 3
 9
NUMBER TO SQUARE? ^C
BREAK AT LINE 10
OK
```

You can also INPUT more than one variable:

```
10 INPUT A,B,C
```

will wait for three values separated by a comma. The following program PRINTs the sum of the values typed in.

```
NEW
OK
10 INPUT A,B,C
20 PRINT A+B+C
30 GOTO 10
RUN
? 1,2,3
 6
```

```
? 59,129,-20
 168
? ^C
BREAK AT LINE 10
OK
```

If you make a mistake entering data, XYBASIC will say REDO and reprompt you with another ?. If you enter too much data, XYBASIC will say EXCESS IGNORED and disregard the extra values. INPUT is legal only in program mode; an ID (Illegal Direct) error will occur if you use it in direct mode.

In Extended XYBASIC you can also use INPUT to get strings and assign them to string variables. The strings you type may be either quoted or unquoted, and may contain both upper and lower case characters. More information on strings is given in Section 4.

In CP/M versions of XYBASIC with sequential disk operations, INPUT is also used to read information from disk data files. More information about this use of INPUT is given in Section 15.

REM and '

The REM command and special character ' allow you to explain your program with comments. REMarks allow you to note what a program is doing, but have no meaning to XYBASIC. When BASIC encounters a REM it ignores everything to the right of it on the same line. For example:

```
NEW
OK
10 REM THIS PROGRAM DEMONSTRATES REM
20 PRINT "REM TEST"
30 REM THAT'S ALL THERE IS TO IT
RUN
REM TEST
OK
```

You may find it more convenient to use ' instead of REM to put a comment on the same line as the command to which it applies. ' has the same effect as REM, but it can appear on a line right after any command. Again, XYBASIC ignores whatever follows ' on the line.

```
NEW
OK
10 I = I + 2  'USE EVEN VALUES ONLY
20 J = J + I  'SUM FOR AVERAGE
30 PRINT J    'THIS IS A REMARK
RUN
 2
```

```
OK
```

XYBASIC really ignores whatever follows a REM or '. In the following example XYBASIC does not execute either the PRINT or the GOTO.

```
NEW
OK
10 REM PRINT "THIS NEVER GETS PRINTED"
20 'GOTO 10 : AND THIS NEVER GETS EXECUTED
RUN
OK
```

IF / THEN

On many occasions you want a program to make a decision. For example, you might wish to reject a printed circuit board you are testing if a test voltage is too high. The IF / THEN command allows you to make such decisions. An example is:

```
IF A = 5 THEN 100
```

The IF command has two parts, an IF part (IF A = 5 in the example) and a THEN part (THEN 100). The IF part contains a logical formula (A = 5), and the THEN part contains a line number (100) or a command. When XYBASIC executes an IF command, first it evaluates the logical formula. If it is true, the THEN part is executed (by doing a GOTO to the given line number or executing the given command). If the logical formula is false, the line following the IF command line is executed. Try the following program, a simple number guessing game using IF / THEN.

```
NEW
0K
10 INPUT "YOUR GUESS" A
20 IF A < 5 THEN PRINT "TOO SMALL"
30 IF A > 5 THEN PRINT "TOO BIG"
40 IF A <> 5 THEN 10
50 PRINT "YOU GUESSED IT!"
RUN
YOUR GUESS? 7
TOO BIG
YOUR GUESS? 4
TOO SMALL
YOUR GUESS? 5
YOU GUESSED IT!
OK
```

In line 10, XYBASIC requests a number A from the user. Line 20 checks whether A < 5, and PRINTS an appropriate message if it is. Similarly, line 30 checks whether A > 5 and PRINTS an appropriate message. Line 40 returns to line 10 for another guess if the guess was wrong. Notice that the IF command of line 40 lets the program loop until a given condition (namely, that the guess is correct) is satisfied, unlike previous examples where loops continued execution until interrupted with <control-C>.

In the logical formula you may use XYBASIC's relational operators, namely:

=	equal
>	greater than
<	less than
<=	less than or equal to
>=	greater than or equal to
<>	not equal

You may also use XYBASIC's logical operators, namely:

AND	logical AND
OR	logical inclusive OR
XOR	logical exclusive OR
NOT	logical negation

The following examples give an idea of the wide variety of logical formulas you can use in IF commands.

```
IF (X AND Y)=0 THEN 100
IF X+Y*E = 4/Z THEN PRINT "GOTCHA"
IF A+B = 14 AND C*D = 12 OR Y=Z THEN J=K
```

STOP

The STOP command interrupts execution of your program, prints the line number at which the STOP occurs, and returns you to direct mode. For example:

```
NEW
OK
10 PRINT "DONE"
20 STOP
RUN
DONE
BREAK AT LINE 20
OK
```

You can use STOP to determine whether you have reached a given point in your program, and then CONTinue execution after the STOP. You may find the powerful BREAK commmand described in Section 7 more useful for this purpose, though.

END

The END command tells XYBASIC to return to direct mode. An END command can occur anywhere in your program but need not occur at all -- XYBASIC will return to direct mode after executing the program's highest line number. Try this:

```
NEW
OK
10 PRINT "A"
20 GOTO 40
30 END
40 PRINT "B"
50 GOTO 30
RUN
A
B

OK
```

Now type CONT and notice that you can CONTinue after an END:

```
CONT
B
OK
```

GOSUB and RETURN

When writing programs in any language there are often several places where the program must perform the same task. The GOSUB and RETURN commands in XYBASIC allow you to enter and return from a subroutine (or subprogram) which does such a task. The subroutine is written only once, but may be used from many different points in your program. Besides conserving space (i.e. the memory used to store your program), subroutines make your programs easier to write, understand and maintain.

Like GOTO, GOSUB tells XYBASIC to execute the command on a specified line instead of the command following the GOSUB. But GOSUB also has a powerful RETURN feature. When the next RETURN command is executed, XYBASIC RETURNS to the command following the GOSUB. Therefore you can enter a subroutine from different places in your program, and RETURN to the appropriate command each time. The following example uses the TRACE feature described in Section 7 to demonstrate the flow of control with GOSUB and RETURN commands.

```
NEW
OK
10 TRACE
20 PRINT"BEGINNING";
30 GOSUB 100
40 PRINT"MIDDLE";
50 GOSUB 100
60 PRINT"END";
70 UNTRACE
80 END
100 PRINT "SUBROUTINE";
110 RETURN
RUN
[20 PRINT "BEGINNING";]     BEGINNING
[30 GOSUB 100]
[100 PRINT "SUBROUTINE";]   SUBROUTINE
[110 RETURN]
[40 PRINT "MIDDLE";]        MIDDLE
[50 GOSUB 100]
[100 PRINT "SUBROUTINE";]   SUBROUTINE
[110 RETURN]
[60 PRINT "END";]           END
[70 UNTRACE]
OK
```

After executing the TRACE command in line 10, XYBASIC prints the bracketed line number and contents of each line it executes. You can see that executing line 110 RETURNs control first to line 40 and then to line 60.

If your program executes a RETURN without a corresponding GOSUB, an RG (Return without Gosub) error will occur, as XYBASIC does not know where to return. If your program GOSUBs to a nonexistent line number, a US (Undefined Statement) error will occur.

Each time XYBASIC executes a GOSUB it uses memory space to store information about where to RETURN, and the used space is reclaimed when the corresponding RETURN is executed. If insufficient memory space remains, an OM (Out of Memory) error will occur. The FRE example in Section 3 below demonstrates how memory space is used and later reclaimed.

The following example uses GOSUB and RETURN to calculate the greatest common divisor (G.C.D.) of three numbers, i.e. the largest number which divides each with no remainder. First it finds the G.C.D. of X and Y, then finds the G.C.D. of that value and Z; the result is easily shown to be the G.C.D. of X, Y and Z.

```
NEW
OK
10 INPUT "THREE POSITIVE NUMBERS" X, Y, Z
```

```
20 GOSUB 100   'G.C.D. 0F X AND Y RETURNED IN Y
30 LET X = Z
40 GOSUB 100   'G.C.D. OF Z AND G.C.D. RETURNED IN Y
50 PRINT "G.C.D. ="; Y
60 GOTO 10
100 'SUBROUTINE RETURNS G.C.D. OF X AND Y IN Y
110 IF X >= Y THEN 150
120 TEMP = X   'SWITCH TO FORCE X >= Y
130 X = Y
140 Y = TEMP
150 TEMP = X MOD Y
160 IF TEMP <> 0 THEN 130  'KEEP TRYING
17D RETURN  'DONE WHEN X MOD Y = O
RUN
THREE POSITIVE NUMBERS? 10,35,95
G.C.D. = 5
THREE POSITIVE NUMBERS? 22,121,999
G.C.D. = 1
THREE POSITIVE NUMBERS? ^C
BREAK AT LINE 10
OK
```

The subroutine to compute the G.C.D. of X and Y starts at line 100, and is called from lines 20 and 110. Executing the RETURN of line 170 transfers control to the statement following the GOSUB, i.e. to line 30 or line 50. The remainder operator MOD used in line 150 is explained in Section 3.

READ, DATA and RESTORE

The DATA command lets you insert tables of data into your program, and the READ command gives you access to this data. Execution of a READ command READs the next value from a DATA command and assigns it to a specified variable. Try the following example:

```
NEW
OK
10 READ X
20 PRINT x;
40 DATA 1, 2, 3, 4
RUN
 1
OK
```

If you run out of DATA values, an OD (Out of Data) error occurs. Try adding the following line to the above example.

```
30 GOTO 10
RUN
 1 2 3 4
OD ERROR: 10 READ X

OK
```

To read the same DATA again (and not get the OD error) you can use the RESTORE command to start READing from the first DATA item again. Now make the following addition to the program:

```
25 IF X = 4 THEN RESTORE
RUN
 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2
 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 3 4 1 2 ^C
BREAK AT LINE 25
OK
```

To exit from this program type <control-C>. Whenever XYBASIC executes a RUN or a NEW it also automatically RESTOREs.

If you include a line number after RESTORE, XYBASIC will READ the next DATA item after the specified line number. This allows you to select which of several DATA areas you wish to read from. Continuing with the above example:

```
25 IF X = 8 THEN RESTORE 50
50 DATA 5, 6, 7, 8
RUN
 1 2 3 4 5 6 7 8 5 6 7 8 5 6 7 8 5 6 7 8 ^C
BREAK AT LINE 25
OK
```

In Extended XYBASIC you can also use READ to assign string DATA to string variables. The string DATA may be either quoted or unquoted, and may contain both upper and lower case characters. More information about strings is given in Section 4.

The DATA command can only be used in program mode; an ID (Illegal Direct) error occurs if you use it in direct mode.

The next example shows how you can use the DATA statement to hold information used in controlling a process.

Example:
Microx Corporation manufactures microcomputer products. One of their highly advanced machines requires its operator to push one of two buttons a specified number of times. Microx uses the following XYBASIC program to instruct the operator.

```
NEW
OK
10 PRINT "START OF BUTTON INSTRUCTIONS, HOPE YOU'RE READY!"
20 GOSUB 200
30 READ X
40 PRINT "PUSH THE RED BUTTON"; X; "TIMES"
50 GOSUB 200
60 READ Y
70 PRINT "PUSH THE YELLOW BUTTON"; Y; "TIMES"
80 GOSUB ZOO
90 I = I + 1
100 IF I < 3 THEN 30  'LOOP 3 TIMES
110 PRINT "ALL DONE";
120 END
150 DATA 2,7,13,1,10,21
200 'SUBROUTINE TO PROMPT USER
210 INPUT "TYPE 1 TO CONTINUE" N
220 IF N <> 1 THEN 210
230 RETURN
RUN
START OF BUTTON INSTRUCTIONS, HOPE YOU'RE READY!
TYPE 1 TO CONTINUE? 1
PUSH THE RED BUTTON 2 TIMES
TYPE 1 TO CONTINUE? 1
PUSH THE YELLOW BUTTON 7 TIMES
TYPE 1 TO CONTINUE? 1
PUSH THE RED BUTTON 13 TIMES
TYPE 1 TO CONTINUE? 1
PUSH THE YELLOW BUTTON 1 TIMES
TYPE 1 TO CONTINUE? 1
PUSH THE RED BUTTON 10 TIMES
TYPE 1 TO CONTINUE? 1
PUSH THE YELLOW BUTTON 21 TIMES
TYPE 1 TO CONTINUE? 1
ALL DONE
OK
```

This program reads the button pushing information from the DATA statement of line 150. When lines 30 and 60 are executed they READ the next piece of DATA (first 2 to X, then 7 to Y, then 13 to X, etc.). The counter I prevents an OD error.

FOR and NEXT

The FOR and NEXT commands allow you to execute a group of commands several times, that is to write controlled loops. The FOR and NEXT construction greatly simplifies XYBASIC programming by doing some of the programmer's work automatically, making programs both easier to write and easier to understand. To see how FOR and NEXT work, first look at the following

program using IF / THEN and GOTO.

```
NEW
OK
10 REM PROGRAM TO PRINT SQUARES OF NUMBERS
20 LET I = 1                    'INITIALIZE COUNTER TO 1
30 IF I > 10 THEN 70            'DONE AFTER 10
40 PRINT "THE SQUARE OF"; I; "IS"; I * I
50 LET I = I + 1                'INCREMENT COUNTER
60 GOTO 30                      'TRY NEXT VALUE
70 END
RUN
THE SQUARE OF 1 IS 1
THE SQUARE OF 2 IS 4
THE SQUARE OF 3 IS 9
THE SQUARE OF 4 IS 16
THE SQUARE OF 5 IS 25
THE SQUARE OF 6 IS 36
THE SQUARE OF 7 IS 49
THE SQUARE OF 8 IS 64
THE SQUARE OF 9 IS 81
THE SQUARE OF 10 IS 100
OK
```

This program just PRINTs the squares of numbers between 1 and 10. Line 20 sets I to 1, and the IF command in line 30 tests whether I is greater than 10; execution ENDs at line 70 if it is. Line 40 PRINTs the desired information, line 50 increments I, and the GOTO of line 60 defines the loop. The same program can be written in a simpler and clearer way using FOR and NEXT:

```
NEW
OK
10 REM PROGRAM TO PRINT SQUARES OF NUMBERS
20 FOR I=1 TO 10
40 PRINT "THE SQUARE OF"; I; "IS"; I * I
60 NEXT I
70 END
RUN
THE SQUARE OF 1 IS 1
THE SQUARE OF 2 IS 4
THE SQUARE OF 3 IS 9
THE SQUARE OF 4 IS 16
THE SQUARE OF 5 IS 25
THE SQUARE OF 6 IS 36
THE SQUARE OF 7 IS 49
THE SQUARE OF 8 IS 64
THE SQUARE OF 9 IS 81
THE SQUARE OF 10 IS 100
OK
```

Here the FOR command of line 20 first sets I to 1, and then tests whether I is less than or equal to 10. Since it is, the command in line 40 is executed. The NEXT command in line 60 then increments I by 1 and again tests whether I is less than or equal to 10. If it is, the command after the FOR is executed (namely, line 40); if not, the command after the NEXT is executed (namely, line 70). You should notice that this program does the same thing as the previous example, but without using IF / THEN or GOTO.

You can use NEXT without specifying the FOR variable, and let XYBASIC find the most recent FOR automatically:

```
60 NEXT
```

A variation of the FOR command lets you use an increment other than 1. Try changing the example given above:

```
20 FOR I = 1 TO 10 STEP 2
RUN
THE SQUARE OF 1 IS 1
THE SQUARE OF 3 IS 9
THE SQUARE OF 5 IS 25
THE SQUARE OF 7 IS 49
THE SQUARE OF 9 IS 81
OK
```

Notice that now each NEXT I increments I by 2 instead of 1. You can use any number, variable or formula to specify the increment with STEP, but the value of the increment is computed only once (when the FOR command is executed). Similarly, the value of the bound specified by TO is computed only once. If the value of the increment is negaive, XYBASIC steps backwards through the values:

```
20 FOR I = 10 TO 1 STEP -2
RUN
THE SQUARE OF 10 IS 100
THE SQUARE OF 8 IS 64
THE SQUARE OF 6 IS 36
THE SQUARE OF 4 IS 16
THE SQUARE OF 2 IS 4
OK
```

Sometimes you might make a mistake in constructing a FOR loop and try to execute a NEXT command without a corresponding FOR. If you do, a NF (Next without For) error will occur, as shown in the following example.

```
NEW
OK
10 PRINT "NF EXAMPLE"
```

```
20 NEXT I
RUN
NF EXAMPLE

NF ERROR: 20 NEXT
                  I
OK
```

The variable you use to control a FOR loop must be a simple numeric variable such as I; an SN (SyNtax) error will occur if you try to use an array element such as A(I) instead.

You will frequently want to nest one FOR loop within another, as in the following example.

```
NEW
OK
10 FOR I = 1 TO 5
20 FOR J = I TO 5
30 PRINT I, J, I*J
40 NEXT J
50 NEXT I
RUN
 1             1             1
 1             2             2
 1             3             3
 1             4             4
 1             5             5
 2             2             4
 2             3             6
 2             4             8
 2             5             10
 3             3             9
 3             4             12
 3             5             15
 4             4             16
 4             5             20
 5             5             25
OK
```

Because this construction is so common, XYBASIC lets you combine successive NEXT commands into one. Here you can replace lines 40 and 50 with

```
40 NEXT J, I
50
```

A single NEXT can specify as many variables as there are corresponding FOR commands. You must be careful, however, to specify the correct order of nesting (NEXT **J**, I rather than NEXT I, **J** in the example), or a NF error will occur when

XYBASIC encounters the wrong variable name.

If the conditions of a FOR command are initially false (namely, the initial value is greater than the bound when the increment is positive, or less than the bound when the increment is negative), the body of the FOR loop is not executed. Instead, XYBASIC searches through the program for the matching NEXT and executes the following command. For example:

```
NEW
OK
10 FOR I = 1 TO 0
20 PRINT "THIS NEVER GETS PRINTED"
30 NEXT I
40 PRINT "END OF LOOP"
RUN
END OF LOOP
OK
```

If XYBASIC is unable to find the matching NEXT in this case, a FR (FoR) error will occur.

In Extended XYBASIC you can write FOR loops which use either integer variables or floating point variables to control the loop. Since integer arithmetic is considerably faster than floating point arithmetic, you can often speed up a program by replacing floating point FOR commands with integer FOR commands. For example, the command

```
FOR I = 5 T0 6.5 STEP .03
```

could be replaced by

```
FOR I% = 500 TO 650 STEP 3
```

Like GOSUB, FOR uses memory space to store information about the FOR loop, and the space is reclaimed when you exit from the loop. An OM (Out of Memory) error will occur if insufficient space remains.

The following example shows how useful FOR loops can be in printing tables.

Example:
Charles Squaro works for a company which makes cubic boxes with sides between 5 and 30 inches. He wants to know how much wood is used and the resulting volume for each type of box, and wants a table he can refer to when ordering. Since the volume of a cube is the cube of its side, and the surface area of a cube is six times the square of its side, the following program prints the desired information.

```
NEW
OK
10 PRINT "SIZE", "AREA", "VOLUME"
20 FOR I = 5 TO 30
30 PRINT I, I*I*6, I*I*I
40 NEXT I
```

ON / GOTO and ON / GOSUB

The ON / GOTO and ON / GOSUB commands let you use the value of a variable or formula to choose which of a group of tasks to perform. For example, execution of the command

```
ON VAR GOTO 10,20,30,40
```

transfers control to line 10 if the value of VAR is 1, to line 20 if the value of VAR is 2, etc. In general control is transferred to the Nth line number in the list, with the value of N determined by the given formula. Try the following program, which uses the ON / GOTO command to type a message appropriate to the number you supply.

```
NEW
OK
10 INPUT "WHICH NUMBER (1-4) DO YOU WANT" X
20 ON X GOTO 30, 50, 70, 90
30 PRINT "ONE"
40 END
50 PRINT "TWO"
60 END
70 PRINT "THREE"
80 END
90 PRINT "FOUR"
100 END
RUN
WHICH NUMBER (1-4) DO YOU WANT? 3
THREE
OK
```

Of course the ON command of line 20 could be replaced with IF commands, but the program is much simpler when you use ON instead:

```
20 IF X = 1 THEN 30
22 IF X = 2 THEN 50
24 IF X = 3 THEN 70
26 IF X = 4 THEN 90
28 STOP
```

The ON / GOSUB command works similarly to ON / GOTO, but executes a GOSUB to the subroutine at the given line number instead of a GOTO. For example:

ON (X+Y) / 3 + 1 GOSUB 100, 200, 300

If the value of the given formula is less than or equal to zero, or is larger than the number of line numbers in the list, an ON error will occur.

In Extended XYBASIC the value of the formula is automatically truncated to the least integer less than or equal to its value, as described under Conversions in Section 3.

DIM

In XYBASIC you can use arrays as well as simple variables. An array is a collection of simple variables sharing the same name but distinguished by an index or subscript. The DIM command tells XYBASIC to set aside space for an array. If you say

```
DIM A(7)
```

then XYBASIC will create space for a one dimensional array named A with eight elements; the allowed subscripts of A are 0 through 7. You can think of A as an indexed series of simple variables, as illustrated by the following diagram.

```
A(0) A(1) A(2) A(3) A(4) A(5) A(6) A(7)
```

Similarly, if you say

```
DIM B(2,5)
```

XYBASIC will create space for a two dimensional array named B with 3 * 6 = 18 elements. You can think of B as a rectangular collection of variables, with the first subscript specifying the row and the second specifying the column of a variable:

```
B(0,0)   B(0,1)   B(0,2)   B(0,3)   B(0,4)   B(0,5)
B(1,0)   B(1,1)   B(1,2)   B(1,3)   B(1,4)   B(1,5)
B(2,0)   B(2,1)   B(2,2)   B(2,3)   B(2,4)   B(2,5)
```

XYBASIC lets you define arrays with any number of subscripts, lets you define each dimension with any numeric formula, and lets you use a single DIM command to DIMension more than one array. In Extended XYBASIC you can also define arrays of any type -- floating point, integer, or string. For example,

```
DIM S$ (I), B (2, I * J), X% (2, 2, 2, 2)
```

tells Extended XYBASIC to set aside space for a one dimensional string array S$ with I + 1 elements, a two dimensional floating point array B with 3 * (I * **J** + 1) elements, and a four dimensional integer array with 3 * 3 * 3 * 3 = 81 elements. More information about string arrays is given in Section 4.

The following example demonstrates how array elements may be manipulated.

```
NEW
OK
10 DIM A(10)
20 A(O) = 1
30 A(1) = 1
40 FOR I = 2 TO 10
50 A(I) = A(I-1) + A(I-2)
60 NEXT I
70 FOR I = 0 TO 10
80 PRINT A(I);
90 NEXT I
RUN
 1 1 2 3 5 8 13 21 34 55 89
OK
```

Here line 10 is a DIMension statement, creating an array named A with 11 elements. After each element is given an initial value, the FOR loop starting at line 70 prints the values.

The following example uses a two-dimensional array. First each element is given an initial value computed from its subscripts, then the array is printed.

```
NEW
OK
10 DIM A(5,4)
20 FOR I = 0 TO 5
30 FOR J = 0 TO 4
40 A(I,J) = 10*I + J
50 NEXT J,I
60 FOR I = 0 TO 5
70 FOR J = 0 TO 4
80 PRINT A(I,J),
90 NEXT J
100 PRINT
110 NEXT I
RUN
 0            1            2            3            4
 10           11           12           13           14
 20           21           22           23           24
 30           31           32           33           34
```

```
 40            41            42            43            44
 50            51            52            53            54
OK
```

If you use an array element in your program before executing a DIM statement to define the array, a SN (SyNtax) error will occur because XYBASIC will not understand the subscript. If the value of a subscript expression is less than zero or greater than the size of the array declared in its DIM statement, a BS (Bad Subscript) error will occur.

In Extended XYBASIC, any floating point values you use to specify array DIMensions or subscripts will be truncated to integer values automatically, as described under Conversions in Section 3.

Multiple Commands Per Line

The special character : (colon) lets you include more than one command on a single line by just putting a : between each command. For example, you could rewrite the ON / GOTO example given above as follows:

```
NEW
OK
10 INPUT "WHICH NUMBER (1-4) DO YOU WANT" X
20 ON X GOTO 30, 50, 70, 90
30 PRINT "ONE" : END
50 PRINT "TWO" : END
70 PRINT "THREE" : END
90 PRINT "FOUR" : END
RUN
WHICH NUMBER (1-4) DO YOU WANT? 2
TWO

OK
```

Writing several commands on one line not only conserves memory space (see Appendix 2), but also may help make your program more readable. If you need to pass information to a subroutine, writing the necessary LET commands on the same line as the GOSUB allows you to see just what your program is doing, as illustrated by the logical operator program in Section 8.

XYBASIC even lets you write FOR loops in direct mode, using : . Try it:

```
FOR I = 1 TO 10 : PRINT I * I; : NEXT I
 1  4  9  16  25  36  49  64  81  100
OK
```

When the logical formula of an IF / THEN command is false, XYBASIC executes the next line of the program rather than the next command. This allows you to write several commands on the same line as the IF / THEN, using :, and execute them only if the condition is true. For example,

```
IF X < 0 THEN X = -X : GOSUB 1000 : GOTO 100
```

will replace X by -X, execute the subroutine at line 1000, and GOTO line 100 if X < 0, but will do nothing if X >= 0.

REMarks can be terminated only by <carriage return>, not by :. Of course you may write REMarks after a :, although using ' is more convenient.

Section 3: Numeric Formulas

You can express numeric values in XYBASIC with numbers and variables, but very frequently you will want to use more complicated formulas. You can build formulas with operators and functions. The wide variety of specialized numeric operators and functions available in XYBASIC is detailed in this section.

Arithmetic Operators

In XYBASIC you can use the arithmetic operators + (addition), - (subtraction or negation), * (multiplication), / (division), MOD (remainder), and JOIN (concatenation, described in Section 8 below). To use an operator, just write it between two numbers, variables or formulas. For example, to add 1 to 2 you write 1 + 2; to divide the value of I by 2 you write I / 2.

In Integer XYBASIC, / represents integer division, so I / 2 gives the integer part of the result and I MOD 2 gives the remainder.

Extended XYBASIC allows two different division operators: / represents floating point division (so 3 / 2 returns the value 1.5) , and \ represents integer division (so 3 \ 2 returns 1, the integer part of 3 / 2). In either version, dividing any quantity by 0 will produce an OV (OVerflow) error.

Extended XYBASIC also allows you to use the exponentiation operator ^, which returns its first argument to the power given by its second argument. For example, 2 ^ 3 returns 8. A FC (Function Call) error occurs if the first argument is negative and the second is not an integer.

If the result of an arithmetic operation is too small or too large, an OV (OVerflow) error will occur. This happens when a result is not in the range -32768 to 32767 in Integer XYBASIC, or not in the range -1 .7 * 10^38 to 1 .7 * 10^38 in Extended XYBASIC.

A formula can be any legal combination of numbers, variables, operators and functions, but you must be aware of the order in which computations will be performed. In normal mathematical usage the formulas 1 + 2 * 3 and 2 * 3 + 1 are both considered to have the value 7 (not 9 or 8) , and XYBASIC uses similar conventions to evaluate unparenthesized formulas. You can think of operators as being arranged in the following order:

- (negation)
JOIN
^ [Extended]
*, /, MOD, \ [Extended]
+, - (subtraction)

Operators which occur higher on this list are performed first; when two operators are on the same level, XYBASIC evaluates from left to right. If you want to perform operations in a specific order, you can just enclose the subformulas you want evaluated first within parentheses. The examples below are legal formulas, with the fully parenthesized version indicating the order of operator evaluation.

3 + X * - 5	3 + (X * (-5))
N1 + 2 * Y / Z	N1 + ((2 * Y) / Z
(N1 + 2) / Y JOIN Z	(N1 + 2) / (Y JOIN Z)

Because integer arithmetic is faster than floating point arithmetic, Extended XYBASIC uses integer arithmetic whenever possible. You can often both conserve memory space and increase the execution speed of your program by using integer variables to store values which you know will always be integers in the range -32767 to 32767.

Conversions

As noted in Section 2, you can use both integer and floating point numeric variables in Extended XYBASIC. Because XYBASIC converts between these two variable types automatically, you can write programs in the most natural way.

Of course any integer value can be converted to an equivalent floating point value. Whenever XYBASIC needs to convert a floating point value to an integer, it truncates it to the largest integer less than or equal to the floating point value. For example:

```
NEW
OK
10 I% = 2.5
20 J% = -3.1
30 PRINT I%, J%
RUN
 2             -4
OK
```

Here XYBASIC converted the value 2.5 to the integer 2 and the value -3.1 to the integer -4. An OV (OVerflow) error occurs if the floating point value is less than -32767 or greater than 32767.

Conversions are also performed automatically whenever a command or function requires an integer argument. For example, the value in an ON command, the dimensions in a DIM command and the subscripts of an array variable reference must be integers, and are converted if they are floating point values.

Since string values cannot be converted to numeric values, a TM (Type Mismatch) error occurs whenever XYBASIC finds a string value where it expects a numeric value, or vice versa. For example:

```
I = "DOG"
TM ERROR: I = "DOG"

OK
S$ = 1.5
TM ERROR: S$ = 1.5

OK
```

In the first example the string "DOG" could not be assigned to the floating point variable I; in the second example the numeric value 1.5 could not be assigned to the string variable S$. The functions CHR$, ASC, STR$ and VAL convert between strings and numeric values, and are discussed in detail in Section 4.

Relations

In writing IF commands you use logical formulas, which have a value of true or false. You can build logical formulas by using the arithmetic relations < (less than), > (greater than), = (equal to), <= (less than or equal to), >= (greater than or equal to), and <> (not equal to) between two formulas. You can also combine logical formulas with the logical operators AND, OR (inclusive or), XOR (exclusive or) and NOT (negation), as described in detail below. You can think of the relations and logical operators as arranged in the following order:

<, >, =, <=, >=, <>
NOT
ABS
OR, XOR

In addition, arithmetic operations are always performed before logical operations. For example:

```
X = 1 AND Y = 3 + X * 2 means
   (X = 1) AND (Y = (3 + (X * 2)))
PET = DOG AND NOISE = BARK OR PET = CAT AND NOISE = MEOW means
   ((PET = DOG) AND (NOISE = BARK)) OR ((PET = CAT) AND (NOISE = MEOW))
A * B + C = D AND E < F * G OR H JOIN J * K + L = - M means
   ((((A*B)+C)=D) AND (E<(F*G))) OR ((((H JOIN J)*K)+L)=(-M))
```

The remainder of this section describes the numeric functions available in XYBASIC. A function is like an operator but is written differently: its arguments (if any) are enclosed in parentheses following the function name. In Extended XYBASIC, a TM (Type Mismatch) error will occur if a string value is used as the argument of a numeric function, and an OV (OVerflow) error will occur if a numeric value less than -32767 or greater than 32767 is used where an integer value is expected.

ABS

The ABS function returns the absolute value of its argument. Try some examples:

```
PRINT ABS (-1)
 1
OK
PRINT ABS (100)
 100
OK
PRINT ABS (-125)
 125
OK
```

SGN

SGN is a function which gives the sign of a number. Its value is 1, O or -1 , depending on whether its argument is positive, zero or negative. For example:

```
PRINT SGN (1)
 1
OK
PRINT SGN (O)
 0
OK
PRINT SGN (-1)
-1
OK
PRINT SGN (-100)
-1
OK
PRINT SGN (#324)
 1
OK
```

MOD

The MOD operator computes the remainder of the integer division operation. To see how MOD works (in Extended XYBASIC; in Integer XYBASIC you should use / in place of \ in line 20) try the following program.

```
NEW
OK
10 INPUT "A, B = " A, B
```

```
20 PRINT A; " \ "; B; "="; A \ B
30 PRINT A; "MOD"; B; "="; A MOD B
40 GOTO 10
RUN
A, B = ? 10,5
 10 \  5 = 2
 10 MOD 5 = 0
A, B = ? 23,7
 23 \  7 = 3
 23 MOD 7 = 2
A, B = ? 5,2
 5 \  2 = 2
 5 MOD 2 = 1
A, B = ? ^C
OK
```

The equality A MOD B = A - (A \ B) * B will help you understand the result of MOD, especially when the value of A or B is negative. If the value of B is zero, an OV (OVerflow) error will occur.

SQR

In Extended XYBASIC you can compute square roots using the SQR function. Try the following simple example:

```
PRINT SQR (2)
 1.41421
OK
```

The argument of SQR must be a positive number. If you try to compute SQR of a negative value, a FC (Function Call) error will occur.

LOG

The Extended XYBASIC function LOG returns the natural logarithm of its argument. The following example uses a simple FOR-loop to print a table of the values of LOG between 2 and 3.

```
NEW
OK
10 FOR I = 2 TO 3 STEP .1
20 PRINT 1, LOG(I)
30 NEXT
RUN
 2             .693147
 2.1           .741937
 2.2           .788457
```

```
 2.3            .832909
 2.4            .875469
 2.5            .916291
 2.6            .955511
 2.7            .993252
 2.8            1.0296
 2.9            1.06471
 3              1.09861

OK
```

The argument of LOG must be a positive number. A FC (Function Call) error will occur if the value of the argument is negative or zero.

EXP

The Extended XYBASIC function EXP (X) returns the value of the exponential e ^ X, where e is the Euler number 2.71828... For example:

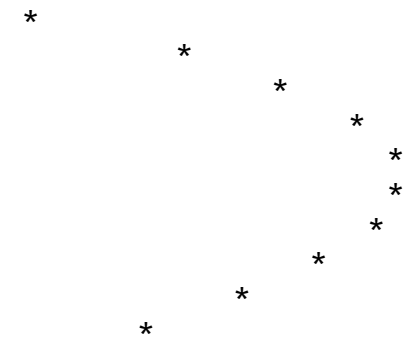
```
PR1NT EXP (2)
 7.38906
OK
```

SIN, COS, TAN and ATN

Extended XYBASIC provides the functions SIN, COS, TAN and ATN to compute the trigonometric functions sine, cosine, tangent and arotangent. The arguments of SIN, COS and TAN are given in radians. The result of ATN is in radians (in the range -pi/2 to pi/2). Since 360 degrees equals 2 * pi radians, you can convert degrees to radians by multiplying by (2 * pi / 360) = .0174533, and you can convert radians to degrees by multiplying by (360 / (2 * pi)) = 57.2958.

The following example uses SIN to print a sine wave on your console. You can interrupt it with <control-C>, vary the width of the wave by changing W and its frequency by changing D, and then CONTinue. The TAB function used in line 40 is described in Section 5.

```
NEW
OK
10 W = 30
20 D = .23
30 I = I + D
40 PRINT TAB (W * (1 + SIN (I))); "*"
50 GOTO 30
RUN
                                                *
```



```
^C
BREAK AT LINE 40
OK
```

INT

In Extended XYBASIC you can use the function INT to find the integer part of a (floating point) numeric value. INT returns the largest integer less than or equal to the value of its argument. You can see how INT works from the following example.

```
PRINT INT (-1.3), INT (1.3)
-2  1
OK
```

Extended XYBASIC automatically performs an INT whenever it finds a floating point value when it expects an integer. For example, a reference to the array element A (1.3, 2.75) is evaluated as a reference to A (1, 2), since array subscripts must be integers.

RND and RANDOMIZE

In a real time or process control environment you may wish to simulate a random process. For example, the designer of a telephone switching system may want to simulate the random calling patterns which occur when users place telephone calls. The function RND is useful for this purpose. In Integer XYBASIC, RND returns a positive pseudorandom number between 0 and 32767. In Extended XYBASIC, RND (X) returns a pseudorandom number between 0 and X. Its value is "pseudo" random because it is generated by a mechanistic process; if you know the process you can predict the next number, which you cannot do with a truly random number. But successive values of RND have a random distribution, and repeat only after 65535 values.

The first example illustrates the use of RND in Integer XYBASIC.

```
NEW
OK
10 FOR I = 1 TO 10
20 PRINT RND;
30 NEXT I
RUN
 5266 25294 28895 14655 12996 17448 28033 8137 6742 1634
OK
```

In Extended XYBASIC, the following program (like the example above, but with line 20 changed slightly) generates random values between 0 and 1.

```
NEW
OK
10 FOR I = 1 TO 10
20 PRINT RND (1);
30 NEXT I
RUN
 .160721 .771912 .881821 .447235 .396622 .532471 .855515
 .248322 .205765 .049866
OK
```

As a convenience to users familiar with other versons of BASIC, Extended XYBASIC treats END (0) as a special case and returns a value between 0 and 1 (rather than always returning 0).

The next sample program gives a simple game which uses RND to generate a pseudorandom value. In Extended XYBASIC RND (10) generates a value between 0 and 10, so INT (RND (10)) is between 0 and 9 and INT (RND (10)) + 1 is between 1 and 10.

```
NEW
OK
10 I = INT (RND (10)) + 1 'GET RANDOM VALUE BETWEEN 1 AND 10
20 INPUT "YOUR GUESS (1 - 10)" J
30 IF I = J THEN 70
40 IF I < J THEN PRINT "TOO BIG"
50 IF I > J THEN PRINT "TOO SMALL"
60 GOTO 20
70 PRINT "YOU GUESSED IT!"
80 INPUT "WANT TO PLAY AGAIN (0 OR 1)" J
90 IF J = 1 THEN 10
RUN
YOUR GUESS (1 - 10)? 5
TOO BIG
YOUR GUESS (1 - 10)? 3
TOO BIG
YOUR GUESS (1 - 10)? 2
YOU GUESSED IT!
```

```
WANT TO PLAY AGAIN (0 OR 1)? 1
YOUR GUESS (1 - 10)? 5
TOO SMALL
YOUR GUESS (1 - 10)? 8
TOO BIG
YOUR GUESS (1 - 10)? 7
YOU GUESSED IT!
WANT TO PLAY AGAIN (0 OR 1)? 0

OK
```

If you are using Integer XYBASIC and want a pseudorandom value between X and Y, you can use the MOD operator to find X + RND MOD (Y - X + 1). Here MOD returns a value between 0 and Y - X, and then adding X to the result gives a value in the desired range. The above program could be modified for Integer XYBASIC by changing line 10:

```
10 I = 1 + RND MOD 10      'GET RANDOM VALUE BETWEEN 1 AND 10
```

The pseudorandom number generator will give the same sequence of values whenever you load XYBASIC unless you use the RANDOMIZE command to start a new sequence. If for example you say

```
RANDOMIZE 15
```

then 15 is used to reinitialize the pseudorandom number generator, and a new series of pseudorandom values will be returned by RND. You can modify the above guessing game as follows to get a different sequence of values each time you play.

```
5 INPUT "TIME OF DAY" N
7 RANDOMIZE N
RUN
TIME OF DAY? 1240
YOUR GUESS (1 - 10)? 5
TOO SMALL
YOUR GUESS (1 - 10)? 6
TOO SMALL
YOUR GUESS (1 - 10)? 9
TOO BIG
YOUR GUESS (1 - 10)? 8
YOU GUESSED IT!
WANT TO PLAY AGAIN (0 OR 1)? 0
OK
```

Here the user types the time, and its value is used to reinitialize RND. You can also use RANDOMIZE to get the same sequence of random values each time you run a program, by reinitializing to a fixed value with a RANDOMIZE command at the start of the program.

FRE

Storing your current XYBASIC program uses part of your computer's memory. Each variable in your program uses memory too. Executing some commands (such as FOR and GOSUB) uses memory to store information needed later. You can use the function FRE to find out how much free memory space you have left. If you type

```
PRINT FRE
 20893
OK
```

then XYBASIC will respond by printing the number of bytes still available. Of course the number XYBASIC prints when you try this depends on the memory configuration of your computer system.

In Extended XYBASIC you can use the similar function FRE$ to find how much space you have left to store strings. FRE$ is described in Section 4 below.

The space used by FOR and GOSUB commands is reclaimed when the loop or subroutine is completed, as the following program shows.

```
NEW
OK
10 PRINT FRE; "BYTES FREE INITIALLY"
20 FOR I = 1 TO 2
30 PRINT FRE; "BYTES FREE INSIDE FOR LOOP"
40 GOSUB 100
50 NEXT I
60 PRINT FRE; "BYTES FREE AFTER LOOP"
70 GOSUB 100
80 PRINT FRE; "BYTES FREE BEFORE STOP"
90 END
100 PRINT FRE; "BYTES FREE INSIDE GOSUB WITH I ="; I
110 RETURN
RUN
 15489 BYTES FREE INITIALLY
 15466 BYTES FREE INSIDE FOR LOOP
 15461 BYTES FREE INSIDE GOSUB WITH I = 1
 15466 BYTES FREE INSIDE FOR LOOP
 15461 BYTES FREE INSIDE GOSUB WITH I = 2
 15481 BYTES FREE AFTER LOOP
 15476 BYTES FREE INSIDE GOSUB WITH I = 3
 15481 BYTES FREE BEFORE STOP
OK
```

You can see from the example that the space used by the commands is recovered. Of course the values printed will depend on the size of your computer system's memory, as noted above.

The value of FRE is really an unsigned 16-bit representation. If your system has more than 32K of free memory, you should always use the UNS function (described below) when PRINTing FRE:

```
PRINT UNS (FRE)
```

This will prevent the number of free bytes from appearing as a negative number.

UNS

As described in detail under Integer Representations in Section 8 below, XYBASIC uses 16 bits to store integer values. When you PRINT an integer-valued formula, XYBASIC normally gives the value considered as a two's complement representation. But for some purposes you may want to use the function UNS to PRINT its value as an unsigned 16-bit representation instead. UNS is particularly useful in conjunction with commands and functions which take memory addresses (between 0 and 65535, rather than -32768 and 32767) as arguments. For example, you should say

```
PRINT UNS (FRE)
```

rather than PRINT FRE to find how many bytes remain free, as a value greater than 32767 would otherwise be printed as a negative number. Try the following program to see what UNSigned values various bit patterns represent.

```
NEW
OK
10 INPUT "TEST VALUE" X
20 PRINT "TWO'S COMPLEMENT VALUE IS", X
30 PRINT "UNSIGNED VALUE IS", UNS(X)
40 GOTO 10
RUN
TEST VALUE? -1
TWO'S COMPLEMENT VALUE IS    -1
UNSIGNED VALUE IS             65535
TEST VALUE? #7fff
TWO'S COMPLEMENT VALUE IS     32767
UNSIGNED VALUE IS             32767
TEST VALUE? -175
TWO'S COMPLEMENT VALUE IS    -175
UNSIGNED VALUE IS             65361
TEST VALUE? ^C
BREAK AT LINE 10
OK
```

Again, type <control-C> to exit from this program.

In Integer XYBASIC, UNS may only be used in PRINT commands. In Extended XYBASIC, UNS may be used anywhere in a formula, like any other function.

DEF FN

XYBASIC has many useful functions, but it may lack one you need. Therefore the very powerful DEF command allows you to DEFine your own functions. For example,

```
10 DEF FN DOUBLE (I) = I * 2
```

defines a function named DOUBLE which multiplies the value of its argument by 2. You can use any variable name for the name of the function, and a DEFined function can then be used in any formula. If you say

```
X = FN DOUBLE (Y)
```

then XYBASIC gives X the result of applying the function DOUBLE to the value of Y. Try the following program.

```
NEW
OK
10 DEF FN DOUBLE (I) = I * 2
20 INPUT "VALUE" X
30 PRINT "DOUBLE ("; X; ") ="; FN DOUBLE (X)
40 GOTO 20
RUN
VALUE? 2
DOUBLE ( 2 ) = 4
VALUE? 100
DOUBLE ( 100 ) = 200
VALUE? -165
DOUBLE (-165 ) = -330
VALUE? ^C
BREAK AT LINE 20
OK
```

DEF is legal only in program mode; an ID (Illegal Direct) error will occur if you try to use it in direct mode. An FC (Function Call) error will occur if you try to use a function DEFined in terms of itself, either directly or indirectly.

The variable I in the function DEFinition is called a dummy parameter. The value of the variable used as a dummy parameter is not changed when you evaluate the function. You can also write function DEFinitions with as many parameters as you desire or without any parameters. The following useful

example uses a function without parameters to convert numbers (given in decimal, binary or hexadecimal) to octal.

```
NEW
OK
10 DEF FN OCTAL = TEST(N,I) + TEST(N,I+1) + TEST(N,I+2)*4
20 INPUT "VALUE" N
30 PRINT "OCTAL ("; N; ")=";
40 PRINT TEST(N,15)
50 FOR I = 12 TO 0 STEP -3
60 PRINT FN OCTAL;
70 NEXT I
80 PRINT
90 GOTO 20

RUN
VALUE? 257
OCTAL( 257)= O  O  O  4  0  1
VALUE? -1
OCTAL(-1 )= 1  7  7  7  7  7
VALUE? #F0F0
OCTAL(-3856)= 1  7  0  3  6  0
VALUE? &101101010001O1
OCTAL( 11589)= 0 2 6 5 0 5
VALUE? ^C
BREAK AT LINE 20
OK
```

This program DEFines a function OCTAL which calculates one digit of the octal representation of N by TESTing three binary digits (which correspond to one octal digit). A FOR loop calls OCTAL to get successive octal digits of N.

In Extended XYBASIC you can DEFine functions of any type -- integer, floating point or string. The function name you choose determines the result type of the function. The parameters of the function may also be of any type. A TM (Type Mismatch) error will occur if the actual parameters in the FN call are incompatible with the dummy parameters in the DEFinition, or if the result of evaluating the function body is incompatible with the function's result type. For example, the following commands define three different functions.

```
10 DEF FN F (X, Y) = (X + Y) / 2
20 DEF FN ROT$ (A$, I) = MID$ (A$, I + 1) + LEFT$ (A$, I)
30 DEF FIN BAD (X) = "BAD" + STR$ (X)
```

The floating point function FN F returns the average of its two floating point arguments. The string function FN ROT$ rotates its string argument left by the number of characters given by its numeric argument; examples of its use are given under String Functions in Section ll, where you will also find definitions of the functions +, MID$ and LEFT$. The function FN BAD is a floating point

function, but evaluating its function body gives a string value; therefore any call of FN BAD will result in a TM error.

Variable Types

In Extended XYBASIC variables can be of three types: floating point, integer, or string. As noted in Section 2, variable names may consist of a letter followed by letters or digits, optionally followed by one of the characters !, %, or $. Variable names ending in a letter or digit, or in !, are floating point variables; variable names ending in % are integer variables; and variable names ending in $ are string variables. The optional type characters are not considered part of the variable name, so for example DOG and DOG! represent the same floating point variable. However, the string variable DOG$; and the integer variable DOG% are different from the floating point variable DOG, and different from each other.

Initially the default variable type for all variables in Extended XYBASIC is floating point, so any variable name not ending in % or is assumed to be the name of a floating point variable. You can change the default variable type for variable names beginning with a given letter or letters by using the DEF INT, DEF SNG, and DEF STR commands. For example, the command

```
DEF INT I
```

tells XYBASIC that all variable names starting with I (and not ending in ! or $) represent integer variables. Similarly, the command

```
DEF STR A-B
```

tells XYBASIC that all variable names starting with A and B (and not ending in ! or %) represent string variables. And the command

```
DEF SNG A-Z
```

tells XYBASIC that all variables starting with A through Z (i.e. with any letter) represent floating point variables. Here SNG abbreviates SiNGle, indicating that the variable is a single precision floating point value.

As noted above, integer arithmetic is faster than floating point arithmetic. If a program only uses numbers which are integers in the range -32767 to 32767, you can make it run faster by using integer variables throughout. The simplest way to do so is to include the command

```
10 DEF INT A-Z
```

as the first line of your program, to tell XYBASIC all variables are integer variables.

The default variable type for all letters is reset to floating point whenever Extended XYBASIC executes a NEW, CLEAR, RUN or LOAD command.

Section 4: Strings

In addition to numeric values, Extended XYBASIC lets you use strings containing up to 255 ASCII characters. You can write string formulas which construct new strings, and you can assign string values to string variables. Strings are an extremely powerful programming tool. You can use the string capabilities of XYBASIC to write text processing programs for a wide variety of applications. This section describes in detail the string manipulation facilities of Extended XYBASIC.

Quoted Strings

You have already used the simpest type of string, the quoted string, in Section 2 above. A quoted string is just a sequence of characters enclosed in quote marks (" "), as in the command:

```
PRINT "HELLO"
HELLO
OK
```

The characters you can include within a quoted string include any printable ASCII characters. In particular you can use both upper and lower case alphabetic characters:

```
PRINT "Hello again"
Hello again
OK
```

You can also use the character <control-G> within quoted strings. On most terminals <control-G> will be "PRINTed" as an audible bell or beep. To include nonprintable ASCII characters (or the quote mark character) within strings you can construct a string formula using the CHR$ function, as described below.

String Variables

Extended XYBASIC lets you assign string values to string variables, in exactly the same way as numeric values are assigned to numeric variables. Any legal variable name followed by the character $ is a string variable. String variables are always initialized to the null (or empty) string, the string containing no characters, when first encountered. You can assign a new value to a string variable with LET:

```
LET A$ = "DOG"
OK
PRINT A$
```

```
DOG
OK
```

Of course the word LET is optional, as before. In this example the value assigned to A$ is given by a quoted string. You can also use string variables or more complicated string formulas (built up from the string-valued functions described below) as the right side of a LET command. However, a TM (Type Mismatch) error will occur if you try to assign a numeric value to a string variable, or a string value to a numeric variable. A TM error will also occur if you use a numeric value as an argument to a function which should have a string argument, or vice versa.

You can also assign a new value to a string variable with the READ and INPUT commands. The DATA or INPUT values you specify may be either quoted or unquoted strings. Of course you must enclose DATA or INPUT items in quotes to include a comma in a string, since commas are used to separate DATA and INPUT items. Leading spaces are also removed from unquoted strings. The following example illustrates these points.

```
NEW
OK
10 INPUT "Name, Salary" A$, I
20 PRINT "The salary of "; A$; " is $"; I
30 PRINT
40 GOTO 10
RUN
Name, Salary? Carter, 200000
The salary of Carter is $ 200000

Name, Salary? Jones, John, 3.65
REDO? "Jones, John", 3.65
The salary of Jones, John is $ 3.65

Name, Salary? ^C
BREAK AT LINE 10
OK
```

Notice that Extended XYBASIC prompted with REDO? after the second line typed in response to the INPUT command of line 10, as the string "Jones" was INPUT for A$ and the string "John" could not be assigned to the numeric variable I. Enclosing "Jones, John" in quote marks in the following line made the problem disappear.

LEN

The function LEN returns the length of its string argument, which is the number of characters the string contains. Its value is an integer between 0 and 255. For example:

```
NEW
OK
10 INPUT A$
20 PRINT LEN (A$)
30 GOTO 10
RUN
? "CAT"
 3
? "Dog house"
 9
? ""
 0
? ^C
BREAK AT LINE 10
```

As you can see from the second example, spaces are treated like any other character within strings.

Concatenation (+)

The string concatenation operator + allows strings to be "added", by simply joining one string to another. For example:

```
NEW
OK
10 PRINT LEN(A$), A$
20 A$ = A$ + "HA!"
30 GOTO 10
RUN
 0
 3            HA!
 6            HA!HA!
 9            HA!HA!HA!
 12           HA!HA!HA!HA!
 15           HA!HA!HA!HA!HA!
 18           HA!HA!HA!HA!HA!HA!
 21           HA!HA!HA!HA!HA!HA!HA!
 24           HA!HA!HA!HA!HA!HA!HA!HA!
^C
BREAK AT LINE 10
OK
```

Since the maximum string length permitted is 255 characters, an LS (Long String) error will occur if the result of + is a string longer than 255 characters. The result will be truncated to 255 characters.

If you use a very complicated string formula in a program, an ST (STring) error may occur. You can avoid the ST error by rewriting the string formula in terms of several simpler formulas.

LEFT$, RIGHT$ and MID$

The substring functions LEFT$, RIGHT$ and MID$ are used to take apart strings. LEFT$ (A$, I) returns the leftmost I characters of A$, as you can see in the following example.

```
NEW
OK
10 INPUT A$
20 FOR I =1 TO LEN (A$)
30 PRINT LEFT$ (A$, I)
40 NEXT I
50 GOTO 10

RUN
? CATFISH
C
CA
CAT
CATF
CATFI
CATFIS
CATFISH
? ^C
BREAK AT LINE 10
OK
```

Similarly, RIGHT$ (A$, I) returns the rightmost I characters of A$. Change the above example as follows:

```
30 PRINT RIGHT$ (A$, I)
RUN
? DOGHOUSE
E
SE
USE
OUSE
HOUSE
GHOUSE
OGHOUSE
DOGHOUSE
? ^C
BREAK AT LINE 10
OK
```

The function MID$ may be used in either of two forms. MID$ (A$, I) returns the right part of A$ starting at the Ith character; the null string is returned if I > LEN (A$). Continuing with the above example:

```
30 PRINT MID$ (A$, 1)
RUN
? XYBASIC
XYBASIC
YBASIC
BASIC
ASIC
SIC
IC
C
? ^C
BREAK AT LINE 10
OK
```

The second form of MID$ is MID$ (A$, I, **J**), which returns the string of **J** characters starting at the Ith character. Change the program as follows:

```
30 PRINT MID$ (A$, I, 2)
RUN
? FISH
IS
SH
H
? ^C
BREAK AT LINE 10
OK
```

The next example uses MID$ to reverse the order of characters in a string.

```
NEW
OK
10 INPUT A$
20 B$ = ""
30 FOR I = 1 TO LEN (A$)
40 B$ = MID$ (A$, I, 1) + B$
50 NEXT I
60 PRINT "The reverse of "; A$ " IS "; B$
70 GOTO 10
RUN
? DOG
The reverse of DOG is GOD
? SAW
The reverse of SAW is WAS
? gorilla
```

```
The reverse of gorilla is allirog
? ^C
BREAK AT LINE 10
OK
```

For each of the functions LEFT$, RIGHT$ and MID$, a nonfatal FC (Function Call) error occurs if the value of I is less than 0 or greater than 255, and the bad value is replaced by O or 255. If I > LEN (A$), LEFT$ and RIGHT$ return all of A$, while MID$ returns the null string.

CHR$

The CHR$ function lets you include nonprintable characters within strings. CHR$ (I) returns the string containing the character with ASCII value I; Appendix 5 gives a table of ASCII character equivalents. For example, the ASCII value of <carriage return> is 13 and the ASCII value of <1inefeed> is 10, so you can define a string CRLF$ consisting of the two characters <carriage return> and <1inefeed> by typing

```
LET CRLF$ = CHR$ (13) + CHR$ (10)
OK
```

A BY (BYte) error occurs if the argument of CHR$ is not a legal (8-bit) character value. Although Integer XYBASIC does not contain other string functions, CHR$ is allowed within PRINT commands in Integer XYBASIC. Additional information on CHR$ is given in Section 5.

ASC

The function ASC returns an integer giving the ASCII value of the first character of A$. An FC error occurs (and ASC returns 0) if A$ is the null string. For example,

```
PRINT ASC ("CAT
 67
OK
PRINT ASC ("DOG")
 68
OK
```

Relations

The relations <, =, >, <=, >= and <> may be used to compare two strings as well as to compare numeric values. Two strings are equal if they are the same, that is if both have the same length and the ASCII values of corresponding characters are equal. For example, "DOG" = "DOG" is true but "DOG" = "DOG "

is false; since the space is not ignored, the lengths of "DOG" and "DOG " are not the same.

A$ < B$ if the ASCII value of the first character of A$ is less than that of the first character of B$; for example, "BAT" < "C". If the first character (or characters) is the same, A$ < B$ if the first nonmatching character is less; for example, "DOGFOOD" < "DOGHOUSE" because "F" < "H". The null string is less than any nonnull string, so "DOG" < "DOGFOOD" (because the null string "" < "FOOD"). Since the relation defines the usual alphabetic order on strings, you can use it to alphabetize a set of strings. A sample program which performs a bubble sort on names is given in the following section on string arrays.

The remaining relations >, <=, >= and <> are defined in the usual way in terms of = and <. For example, A$ <> B$ is true if A$ = B$ is false.

String Arrays

Extended XYBASIC lets you DIMension arrays of string variables in precisely the same way you DIMension arrays of numeric variables. If you are writing a program which processes payroll information about employees, you might use the command

```
DIM NAME$ (50)
```

at the start of the program to allocate 51 string variables called NAME$ (O), NAME$ (1), ... NAME$ (50). Then these variables can be used in the program just like simple string variables. The following program gets names from the console and sorts them into alphabetical order with a simple bubble sort. A bubble sort works by interchanging pairs of values, letting the least ("lightest") values "bubble" to the top. In this program S(1) is compared to S(2), ..., S(N) to assure S(1) is the least; then S(2) is compared to S(3), ..., S(N), and so on. Notice that the DEF STR S command in line 10 allows subsequent commands to refer to the string array as S rather than S$.

```
NEW
OK
10 DEF STR S
20 REM FIRST GET THE DESIRED NUMBER OF NAMES
30 INPUT "Number of names" N
40 DIM S(N)
50 REM NEXT GET THE NAMES
60 FOR I = 1 TO N
70 INPUT "Name" S(I)
80 NEXT I
90 REM BUBBLE SORT THE NAMES AND PRINT THE SORTED RESULT
100 PRINT "Sorted name List:"
110 FOR I = 1 TO N-1
120 FOR J = 1+1 TO N
```

```
130 IF S(1) > S(J) THEN GOSUB 200
140 NEXT J
150 PRINT S(I)
160 NEXT I
170 END
ZOO REM SWITCH THE VALUES S(I) AND S(J)
210 STEMP = S(I)
220 S(I) = S(J)
230 S(J) = STEMP
240 RETURN
RUN
Number of names? 5
Name? Wilson
Name? Smith
Name? Jones
Name? Adams
Name? Smithson
Sorted name list:
Adams
Jones
Smith
Smithson
Wilson

OK
```

String Functions

You can DEFine your own string functions in Extended XYBASIC, in the same way as described under DEF FN in Section 3 above. The following example DEFines a function named ROT$ which rotates its string argument to the left. Note the power of XYBASIC's user-DEFined functions, which let you write functions taking both string and numeric arguments.

```
NEW
OK
10 DEF FN ROT$ (A$, I) = MID$ (A$, I+1) + LEFT$ (A$, I)
20 INPUT "STRING, COUNT" A$, I
30 PRINT A$; " ROTATED LEFT"; I; "PLACES IS "; FN ROT$ (A$, I)
40 PRINT
50 GOTO 20

RUN
STRING, COUNT? HOUSEBOAT,5
HOUSEBOAT ROTATED LEFT 5 PLACES IS BOATHOUSE

STRING, COUNT? abcdef,4
abcdef ROTATED LEFT 4 PLACES Is efabcd
```

```
STRING, COUNT? ^C
BREAK AT LINE 20
OK
```

INSTR

The purpose of INSTR is to provide a convenient and powerful way of finding occurences of one string within another. INSTR (A$, B$) returns the least integer n such that the substring of A$ which starts at the nth character matches B$. If no substring of A$ matches B$, INSTR returns 0. For example:

```
PRINT INSTR ("DOG", "G
 3
OK
PRINT INSTR ("DOG", "C
 0
OK
```

In the first example INSTR returns 3, since "G" is the third character of "DOG". In the second example INSTR returns 0, since the character "C" does not occur in "DOG".

A common use of INSTR is to break up long strings into simpler component parts. The next example uses INSTR to find spaces in a sentence and break the sentence into separate words.

```
NEW
OK
10 DIM S$ (50)
20 INPUT "Sentence" A$          'GET THE SENTENCE
30 I = INSTR (A$, " ")          'FIND THE NEXT SPACE
40 N = N + 1                    'BUMP WORD COUNT
50 IF I = 0 THEN 90             'DONE IF NO MORE SPACES
60 S$ (N) = LEFT$ (A$, I-1)     'SAVE CURRENT WORD
70 A$ = MID$ (A$, 1+1)          'LET SENTENCE BE REMAINDER
80 GOTO 30
90 S$ (N) = A$                  'LAST WORD IS REMAINDER
100 FOR I = 1 TO N              'PRINT THE WORDS
110 PRINT S$ (I)
120 NEXT I

RUN
Sentence? The world is everything which is the case.
The
world
is
everything
```

```
which
is
the
case.

OK
```

Another form of INSTR lets you specify an offset, so you can look for occurences of a substring starting at any character of a string. INSTR (I, A$, B$) returns the least integer n >= I such that the substring of A$ starting at the nth character matches B$. For example:

```
PRINT INSTR (4, "VOODOO", "OO")
 5
OK
```

The string "VOODOO" contains the substring "OO" starting at positions 2 and 5, so here INSTR returns the first position greater than 4. The sample program above may be simplified by making the following changes, using this form of INSTR and leaving the value of A$; unchanged.

```
30 LAST = I + 1
35 I + INSTR (LAST, A$, " ")
60 S$ (N) = MID$ (A$, LAST, I-LAST)
70
90 S$ (N) = MID$(A$, LAST)
RUN
Sentence? There is much here to excite admiration and perplexity.
There
is
much
here
to
excite
admiration
and
perplexity.
OK
```

GET$

The GET$ function lets you check whether a character has been typed while a program is running. If a character has been typed, GET$ returns a string value consisting of the typed character. If no character has been typed, GET$ returns the null string.

You can use GET$ to let a user respond to a question by typing Y or N, or to define control characters to monitor program execution without using <control-C> and CONT. The following example just increments the value in I, and prints its value whenever you type P.

```
NEW
OK
10 IF GET$ = "P" THEN GOSUB 100
20 I = I + 1
30 GOTO 10
100 PRINT I;
11D RETURN
RUN
57  113  181  193  201  305  369 ^C
BREAK AT LINE 20
OK
```

Since XYBASIC automatically removes the parity bit from any character it reads, you cannot GET$ any character with an ASCII value greater than 127 (7F hexadecimal). And of course you should not try to GET$ characters such as <control-C> which have special meanings to XYBASIC.

The function GET is similar to GET$, but returns the ASCII value of the typed character instead of a string consisting of the character. GET is described in Section 8 below.

STR$ and VAL

STR$ and VAL allow conversion between numeric values and strings, and are especially helpful when used with other string functions to reformat numerical output. STR$ (X) turns a numeric value into its string equivalent, returning the string of characters which Extended XYBASIC would PRINT as the value of X. For example, the following program converts an amount to dollars and cents, and then uses STR$ to convert the dollars and cents to strings which are PRINTed as part of the string S$.

```
NEW
OK
10 INPUT "Amount" X
20 D = INT (X)
30 C = 100 * (X - INT (X))
40 S$ = STR$ (D) + " dollars and" + STR$ (C) + " cents"
50 PRINT X; X; "is"; S$
60 GOTO 10
RUN
Amount? 2.25
$ 2.25 is 2 dollars and 25 cents
Amount? .37
```

```
.37 is O dollars and 37 cents
Amount? 11.95
$ 11.95 is 11 dollars and 95 cents
Amount? ^C
BREAK AT LINE 10
OK
```

Conversely, VAL converts a string representation of a number into its numeric value; that is, VAL (A$) gives the numeric value of the constant represented by the string A$. In the following program VAL finds the value of the amount typed after the dollar sign, ignoring the string preceding the dollar sign.

```
NEW
OK
10 INPUT S$
20 I = INSTR (S$, "$")        ' FIND THE $ IN S$
30 X = VAL (MID$ (S$, I+1))   'FIND VALUE OF DOLLAR AMOUNT
40 PRINT "The amount is"; X
50 GOTO 10
RUN
? The unit cost is $6.34
The amount is 6.34
? The price is $9.95
The amount is 9.95
? ^C
BREAK AT LINE 10
OK
```

If the value of A$ is not a legal constant, an FC error occurs and VAL returns 0.

CLEAR and FRE$

Instead of wasting memory by statically allocating space for each string variable to contain up to 255 characters, Extended XYBASIC stores the values of string variables dynamically in a part of memory called string space. String space is also used for temporary storage during the evaluation of some string formulas. Extended XYBASIC initially allocates 256 bytes of string space, which should be adequate for programs which do not use strings extensively. For more complex string programs, the amount of allocated string space may be changed by using the CLEAR command with a numeric argument. For example,

```
CLEAR 1000
OK
```

tells Extended XYBASIC to allocate 1000 bytes of string space. The argument of CLEAR may be any numeric formula. Of course an OM (Out of Memory) error will occur if you try to allocate more string space than the amount of free memory allows.

Just as before, the CLEAR command (without an argument) may be used to clear all variables; this form of CLEAR leaves the amount of string space unchanged.

You can use the function FRE$ to find the number of unused bytes remaining in string space, as in the following example.

```
CLEAR 256
OK
PRINT FRE$
 256
OK

A$ = "DOG"
OK
PRINT FRE$
 253
OK
```

As you can see, assigning the string variable A$ the value "DOG" used three bytes of string space.

If Extended XYBASIC runs out of string space, an OS (Out of String space) error will occur. You should then use CLEAR to increase the amount of available string space.

For programs which perform extensive string manipulation, you can sometimes improve execution speed dramatically by allocating more string space. This happens because the extra space allows Extended XYBASIC to perform string operations more quickly.

Section 5: PRINT Related Commands

The functions and commands in this section help you format output and control your console in different ways. These features are used in conjuction with PRINT commands.

SPC

The SPC function allows easy formatting of output lines. It can only be used within PRINT commands, and simply PRINTS the number of spaces specified by its argument. For example:

```
NEW
OK
10 FOR I = 0 TO 10
20 PRINT "*"; SPC(I); "*"
30 NEXT I
RUN
**
* *
*  *
*   *
*    *
*     *
*      *
*       *
*        *
*         *
*          *

OK
```

TAB

TAB allows you to format your output by spacing to the column on your console specified by its argument. It can only be used within PRINT commands. Try this:

```
PRINT TAB(7);"*"
       *
OK
```

The following program further demonstrates the TAB function.

```
NEW
OK
10 FOR I = O TO 71
20 PRINT TAB(I); "*"
30 NEXT I
40 FOR I = 70 TO 0 STEP -1
50 PRINT TAB(I); "*"
60 NEXT I
70 GOTO 10

RUN
*
 *
  *
   *
    *
     *
      *
       *
        *
         *
          *
           *
^C
BREAK AT LINE 20
OK
```

To exit from this program just type <control-C>.

POS

The function POS takes no arguments, and returns the current position of the cursor on the XYBASIC print line. If no characters have been printed since the last <carriage return> and <linefeed>, its value is 0; otherwise its value is the column in which the most recent character was printed. Modifying line 20 of the SPC example above:

```
NEW
OK
10 FOR I = 0 TO 10
20 PRINT "*"; SPC(I); "*"; POS
30 NEXT (
RUN
** 2
* * 3
*  * 4
*   * 5
*    * 6
*     * 7
```

```
*       * 8
*        * 9
*         * 10
*          * 11
*           * 12

OK
```

CHR$

Console devices (e.g. teletypes and CRT terminals) usually can be controlled by nonprinting control characters. Such characters may for example turn on a teletype reader or backspace the cursor on a CRT. The CHR$ function allows you to PRINT these characters. In Extended XYBASIC, CHR$ returns a one character string with the ASCII value of its argument, as described in Section 4 above, and its value can be PRINTed like any other string value. In Integer XYBASIC, CHR$ can only be used within PRINT commands, and PRINTS the character with the ASCII value of its argument. Appendix 5 contains a table of ASCII equivalents.

For example, the ASCII value of A is 65, so you can print an A by saying

```
PRINT CHR$(65);
A
OK
```

Try the following program:

```
NEW
OK
10 FOR I = 1 TO 26
20 PRINT CHR$(64+I);
30 NEXT I
RUN
ABCDEFGHIJKLMNOPQRSTUVWXYZ
OK
```

CHR$ is also used by the hexadecimal conversion program using RSHIFT in Section 8.

If the argument of CHR$ is greater than 255, a BY (BYte) error will occur.

NULL

Some consoles require fill characters (nulls) after they put out a <carriage return> and <linefeed>. Without fill characters, the first few characters after a <carriage return) and <linefeed> may be lost or may be typed before the carriage

reaches the left margin.

XYBASIC sends no fill characters until you use the NULL command to specify how many fill characters your console needs. If it needs four fill characters, just say

```
NULL 4
```

Then four fill characters are sent after each subsequent <carriage return> and <linefeed>. That's all there is to it.

Section 6: Input/Output, Saving and Loading Programs

The commands in this section allow you to redirect your input or output from one physical device to another, to learn what devices you are using, to SAVE programs for later use and to LOAD programs you have SAVEd.

ASSIGN

If your operating system supports an I/O byte (as the Intellec MDS does, and as CP/M can), you may use the ASSIGN command to reassign physical devices to logical devices. A logical device is a device type (such as CONsole or PUNch) which may have several different physical device implementations; for example your system may let you use either a teletype or a CRT terminal as your console device.

The ASSIGN command allows you to switch between physical devices under program control. ASSIGN changes the value of the I/O byte, so subsequent operations are directed to the selected physical device. For example, to change the CONsole device to device 1 you can just type

```
ASSIGN CON#1
```

The logical device you specify must be CON# (CONsole), RDR# (ReaDeR), PUN# (PUNch), or LST# (LiST). The physical device must be specified by a value between 0 and 3 or an FC (Function Call) error will occur.

XYBASIC normally performs all input and output to the CONsole device. Output is echoed to the LiST device whenever <control-P> is typed. The PUNch and ReaDeR devices are used to SAVE and LOAD programs in Custom I/O versions of XYBASIC, as described below.

IOBYTE

The function IOBYTE returns the current value of the I/O byte, so you can use it to check which physical devices you currently have ASSIGNed. If you say

X = IOBYTE

then X is set to the current value of the I/O byte. The I/O byte contains a two-bit field for each of the four logical devices (CONsole, ReaDeR, PUNch, and LiST), and is organized as indicated by the following diagram.

```
+---------------+---------------+---------------+---------------+
|   LiST field  |  PUNch field  |  ReaDeR field | CONsole field |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
```

```
+--------------+--------------+--------------+--------------+
```

To find the current value of the punch field (using the shift function RSHIFT described in Section 8), you could for example type

```
PRINT RSHIFT (IOBYTE, 4) AND &11
 0
OK
```

The CP/M and ISIS-II versions of XYBASIC store the current value of IOBYTE in the system I/O byte at location 3. Custom I/O versions store the IOBYTE value in the first RAM location (normally 4000H for Extended XYBASIC and 2000H for Integer XYBASIC), and set its value to 0 during initialization.

SAVE and LOAD

SAVE lets you to preserve a program for later use, and LOAD allows you to recall a SAVEd program. In the CP/M and ISIS-II versions of XYBASIC programs are SAVEd to and LOADed from disks. In the Custom I/O version of XYBASIC, SAVE uses the logical PUNch device and LOAD uses the logical ReaDeR device (which may e.g. be audio cassette or paper tape -- don't forget to turn on whatever device you are using!). If you type

```
SAVE "EXAMPL"
```

then a file called EXAMPL.XYB containing the current program will be SAVEd in XYBASIC's internal representation on the currently logged disk under CP/M, on :F0: under ISIS-II, or through the PUNch device in Custom I/O versions.

To load a SAVEd program you just type

```
LOAD "EXAMPL"
```

LOAD also can be used within programs. The current program is deleted when another is LOADed, but you can use LOAD to execute several programs in succession. An RO (ROmsq feature) error will occur if you try to LOAD a program while XYBASIC is addressing a program outside its working space.

In Custom I/O versions, the specified filename may consist only of one to eight upper case letters or digits. You can also type LOAD without a filename to LOAD a program. If a filename is specified, XYBASIC reads from the ReaDeR until it finds the SAVEd program with the given filename; if not, it LOADs the first program it finds.

Because programs are SAVEd or LOADed in XYBASIC's internal representation, SAVE and LOAD will work correctly in Custom I/O versions only if your RDR and PUN routines pass eight bit bytes -- routines which change the parity bit will NOT work! The format of SAVEd programs is described in the Custom I/O

version section of Chapter II. In Custom I/O versions, a CS (CheckSum) error occurs if part of a program LOADs incorrectly.

If you have a teletype with a paper tape reader and punch, you can save programs in ASCII by turning on the punch and using the LIST command. Typing NEW and reading the resulting paper tape then loads your program.

In ISIS-II versions, the filename may consist of one to six letters or digits, and may be preceded by an optional device name; for example,

```
SAVE :F1:"EXAMPL"      'SAVE AS :F1:EXAMPL.XYB UNDER ISIS-II
```

ISIS-II versions also have the ability to SAVE and LOAD programs as an Intel compatible HEX file. A file is SAVEd in HEX format by adding ,H to the end of a SAVE command.

```
SAVE "EXAMPL",H       'SAVE IN HEX AS :F0:EXAMPL.HEX
```

Of course, a HEX file which has been SAVEd can be LOADed using a similar syntax:

```
LOAD "EXAMPL",H       'LOAD THE FILE EXAMPL.HEX
```

This allows users with the Intel PROM programmer UPM to burn PROMs easily. Once a program is debugged, it is saved in HEX format, read into memory using UPM, and then burned into PROM in the normal manner. The HEX file created by SAVE contains an image of the current program relocated to location 0, and should be read with an appropriate offset.

In both CP/M and ISIS-II versions, XYBASIC will SAVE or LOAD the program in a printable and editable ASCII representation, as EXAMPL.BAS, if the specified filename is followed by ,A.

```
SAVE "EXAMPL",A       'SAVE IN ASCII AS EXAMPL.BAS
```

LOADing a .BAS file is much slower than LOADing an .XYB file, so programs should generally be SAVEd in internal .XYB format if the ASCII version is unneeded.

In CP/M versions, the filename may be specified by any string, either quoted or unquoted. The string may consist of an optional disk name, such as A: or B:, followed by one to eight letters or digits. Lower case characters are converted to upper case within the filename, and the currently logged disk is assumed if no disk name is given. For example, if the value of S$ is "EXAMPLE",

```
LOAD S$               'LOAD FROM LOGGED DISK UNDER CP/M
SAVE "B:EXAMPLE"      'SAVE TO DISK B: UNDER CP/M
```

In CP/M versions another form of LOAD allows you to LOAD a program and execute it immediately, without typing RUN. For example,

```
LOAD "TEST", R
```

will load TEST.XYB and RUN it. Similarly,

```
LOAD "B:TEST2", A, R
```

will load B:TEST2.BAS and RUN it. With this enhanced form of LOAD you can LOAD a XYBASIC program during execution of another program, and RUN it immediately without typing anything on your console. In this way you can build chains of XYBASIC programs which run without user intervention.

If a CP/M or ISIS-II version of XYBASIC cannot SAVE or LOAD your program successfully (because of a full disk, for example), a DK (DisK) error results. Under ISIS-II, an ISIS-II error message specifying the nature of the error is also printed. Since XYBASIC takes some time to process a typed program line before being ready for the next line, the first few characters of some lines might be lost unless you use the NULL command described in Section 5 to punch some nulls (15 is usually enough) at the end of each line.

Section 7: Debugging

Even the most experienced programmers write programs which do not work correctly. When a program does not work in the intended way, it is said to have a bug. Getting rid of bugs is called debugging, and can be one of the most difficult tasks confronting the programmer. XYBASIC has a number of features which simplify debugging and enable you to get your program running correctly sooner than would be possible with other BASICS. Rather than trying to find bugs by passively examining listings, you can interact with XYBASIC and let it help you find your mistakes with its debugging features.

TRACE and UNTRACE

TRACE lets you watch the execution of your program on a command by command basis. In TRACE mode XYBASIC prints the bracketed line number and contents of each command executed and the name and value of any modified variable. The following program printing prime numbers shows what TRACE execution looks like.

```
NEW
OK
10 TRACE
20 PRINT 2; "IS PRIME"
30 N = 1
40 N = N + 2
50 FOR I = 3 TO N/2 STEP 2
60 IF N MOD I = 0 THEN 40
70 NEXT I
80 PRINT N; "IS PRIME"
90 GOTO 40
RUN
[20 PRINT 2; "IS PRIME"] 2 IS PRIME

[30 N = 1 ]   N= 1
[40 N = N + 2]              N= 3
[50 FOR I = 3 TO N/2 STEP 2]              I= 3
[80 PRINT N; "IS PRIME"]     3 IS PRIME

[90 GOTO 40]
[40 N = N + 2]              N= 5
[50 FOR I = 3 TO N/2 STEP 2]              I= 3
[80 PRINT N; "IS PRIME"]     5 IS PRIME

[90 GOTO 40]
[40 N = N + 2]              N= 7
[50 FOR I = 3 TO N/2 STEP 2]              I= 3
```

```
[60 IF N MOD I = 0 THEN 40]
[70 NEXT I]   I= 5
[80 PRINT N; "IS PRIME"]     3 IS PRIME

[90 GOTO 40]
[40 N = N + 2]               N= 9
[50 FOR I = 3 TO N/2 STEP 2]              I= 3
[60 IF N MOD I = 0 THEN 40]
[40 N = N + 2]               N= 11
[50 FOR I = 3 TO N/2 STEP 2]              I= 3
[60 IF N MOD I = 0 THEN 40]
[70 NEXT I]   I= 5  ^C
BREAK AT LINE 70
OK
```

After execution of the line 10 TRACE command, XYBASIC prints the bracketed line number and contents of each command before it is executed. If the command is LET, FOR, NEXT, READ or INPUT, the name and new value of the modified variable is also printed.

If a program is not working correctly, you can interrupt it with <control-C>, type TRACE and then CONTinue execution. By following the TRACE you can often find why your program does not work in the way you expected.

Execution of the UNTRACE command disables the TRACE feature, so you can TRACE precisely the sections of program you desire. Try changing the program:

```
45 UNTRACE
75 TRACE
RUN
[20 PRINT 2; "IS PRIME"]     2 IS PRIME

[30 N = 1]    N= 1
[40 N = N + 2]                N= 3
[45 UNTRACE]
[80 PRINT N; "IS PRIME"      3 IS PRIME

[90 GOTO 40]
[40 N = N + 2]                N= 5
[45 UNTRACE]
[80 PRINT N; "IS PRIME"      5 IS PRIME

[90 GOTO 40]
[40 N = N + 2]                N= 7
[45 UNTRACE]  ^C
BREAK AT LINE 45
OK
```

XYBASIC also disables TRACE whenever you execute a NEW.

If your system supports a LST device such as a lineprinter, you may find it convenient to use <control-P> and <control-O> to print TRACEs on the printer rather than on the console.

BREAK and UNBREAK

For some purposes TRACE provides more information than you need; some bugs are easier to find if you just check what happens when certain lines are executed or certain variables are changed. The BREAK command gives you a flexible and powerful tool to do so by letting you set breakpoints on variables or line numbers. BREAK will greatly aid you in debugging your program and let you get it running correctly much faster, increasing your productivity and decreasing programming costs.

To set a breakpoint on the variable I, you just type

```
BREAK I
```

Then the value of I will be printed whenever it is changed, whether by a LET, FOR, NEXT, INPUT or READ command, and the bracketed line number and contents of the command is also printed. The following example demonstrates how a variable breakpoint works.

```
NEW
OK
10 BREAK I
20 I = 1
30 FOR J = 1 TO 10
40 T = I
50 I = I + L
60 L = T
70 NEXT J
RUN
[20 I = 1]    I= 1
[50 I = I + L]              I= 1
[50 I = I + L]              I= 2
[50 I = I + L]              I= 3
[50 I = I + L]              I= 5
[50 I = I + L]              I= 8
[50 I = I + L]              I= 13
[50 I = I + L]              I= 21
[50 I = I + L]              I= 34
[50 I = I + L]              I= 55
[50 I = I + L]              I= 89
OK
```

A single BREAK command can set more than one variable breakpoint. Try changing the program as follows:

```
10 BREAK I, J
RUN
[20 I = 1]    I= 1
[30 FOR J = 1 TO 10]        J= 1
[50 I = I + L]              I= 1
[70 NEXT J]   J= 2
[50 I = I + L]              I= 2
[70 NEXT J]   J= 3
[50 I = I + L]              I= 3
[70 NEXT J]   J= 4
[50 I = I + L]              I= 5
[70 NEXT J]   J= 5
[50 I = I + L]              I= 8
[70 NEXT J]   J= 6
[50 I = I + L]              I= 13
[70 NEXT J]   J= 7
[50 I = I + L]              I= 21
[70 NEXT J]   J= 8
[50 I = I + L]              I= 34
[70 NEXT J]   J= 9
[50 I = I + L]              I= 55
[70 NEXT J]   J= 10
[50 I = I + L]              I= 89
[70 NEXT J]   J= 11
OK
```

You can use the UNBREAK command to remove a variable breakpoint. Add this:

```
35 UNBREAK J
RUN
[20 I = 1]    I= 1
[30 FOR J = 1 TO 10]        J= 1
[50 I = I + L]              I= 1
[50 I = I + L]              I= 2
[50 I = I + L]              I= 3
[50 I = I + L]              I= 5
[50 I = I + L]              I= 8
[50 I = I + L]              I= 13
[50 I = I + L]              I= 21
[50 I = I + L]              I= 34
[50 I = I + L]              I= 55
[50 I = I + L]              I= 89
OK
```

You can even set variable breakpoints on array variables. However, you must be sure to DIMension the array before setting the breakpoint. Change the program as follows:

```
5 DIM A(10)
10 BREAK A
35
65 A(J) = I
RUN
[65 A(J) = I] A(1) = 1
[65 A(J) = I] A(2) = 2
[65 A(J) = I] A(3) = 3
[65 A(J) = I] A(4) = 5
[65 A(J) = I] A(5) = 8
[65 A(J) = I] A(6) = 13
[65 A(J) = I] A(7) = 21
[65 A(J) = I] A(8) = 34
[65 A(J) = I] A(9) = 55
[65 A(J) = I] A(10) = 89
OK
```

Another form of BREAK lets you set breakpoints on line numbers instead of variables. XYBASIC then prints the bracketed line number and contents whenever the line is executed. Change the program again to see how a line breakpoint works.

```
5
10 BREAK 50
65
RUN
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
[50 I = I + L]
OK
```

With line breakpoints you have the option of making the break occur every few times the line is executed, rather than every time. For example:

```
10 BREAK 50, 3
RUN
[50 I = I + L]
[50 I = I + L]
```

```
[50 I = I + L]
OK
```

The break message is now printed every third time the line is executed. When setting a line break you may also specify a variable or list of variables, preceded by a semicolon. The values of the variables in the list are then printed whenever the break occurs.

```
10 BREAK 50; I
RUN
[50 I = I + L]              I= 1
[50 I = I + L]              I= 1
[50 I = I + L]              I= 2
[50 I = I + L]              I= 3
[50 I = I + L]              I= 5
[50 I = I + L]              I= 8
[50 I = I + L]              I= 13
[50 I = I + L]              I= 21
[50 I = I + L]              I= 34
[50 I = I + L]              I= 55
10 BREAK 50, 4; I, J
RUN
[50 I = I + L]              I= 3 J= 4
[50 I = I + L]              I= 21 J= 8
OK
```

Finally, you can use the $ option to set a line break which returns you to direct mode (rather than continuing execution) when the line break occurs.

```
10 BREAK 50; $
RUN

BREAK AT LINE 50
OK
```

The $ option can be used in conjunction with the other BREAK options.

```
10 BREAK 50, 5; I, J; $
RUN
[50 I = I + L]              I= 5 J= 5
BREAK AT LINE 50
OK
```

UNBREAK lets you remove line breakpoints as well as variable breakpoints.

```
10 BREAK 50
55 UNBREAK 50
RUN
[50 I = I + L]
```

```
OK
```

If you just type UNBREAK without giving a line number or variable name, it removes all breakpoints. CLEAR also removes all breakpoints.

Section 8: Bit Manipulation and Control Features

With the powerful bit manipulation and control features of XYBASIC you can perform tasks which would otherwise require assembly language programs. Additional commands make it possible to perform real time delays without a real time clock.

Integer Representations

XYBASIC lets you specify integers in the decimal notation you normally use, but it actually stores them in a notation which is more convenient (and efficient) for your computer. The memory of your 8080 system consists of many bytes, each capable of storing 8 bits, i.e. 2 ^ 8 = 256 possible values. XYBASIC uses two bytes (16 bits) to store 2 ^ 16 = 65536 Possible integer values, and uses those values to represent integers between -32768 and 32767. Positive integers between 0 and 32767 are stored in their binary representation; for example, 4 decimal is 0000 0000 0000 0100 binary and 32766 decimal is 0111 1111 1111 1110 binary. Integers between -32768 and -1 are stored in two's complement representation, found by complementing each digit of the binary representation of the integer's absolute value, and then adding 1 to the result. For example, -4 decimal is 1111 11 11 1111 1011 + 1 = 1111 1111 1111 1100 binary, and -32766 decimal is 1000 0000 0000 0001 + 1 = 1000 0000 0000 0010 binary.

In keeping with standard 8080 conventions the rightmost (least significant) bit of an integer value is called bit 0, and the leftmost (most significant) bit is called bit 15. Many of the bit manipulation functions described below use bit numbers to specify bits to be examined or changed.

Sometimes you may want to consider the representation of an integer to be an unsigned 16-bit value between 0 and 65535, so that for example 1000 0000 0000 0010 binary represents 32770 instead of -32766. In particular, commands or functions (like POKE and PEEK) which take memory addresses as arguments consider such arguments to be unsigned 16-bit representations. You can PRINT the unsigned value of a formula with the function UNS, described in Section 3.

In Extended XYBASIC, any floating point values you use as arguments to control and bit manipulation functions are automatically truncated to integer values, as described under Conversions in Section 3.

TEST

The TEST function allows you to examine a specified bit in an integer value, returning the value (zero or one) of the bit. The first argument of TEST specifies the variable or formula you wish to TEST, the second which bit you wish to look

at. Suppose for example that you want to look at bit 2 of I, which contains 7 (binary &111). Just type

```
PRINT TEST (I,2)
 1
OK
```

The following binary conversion program using TEST will show you how useful it ccan be.

```
NEW
OK
10 PRINT "BINARY CONVERSIONS"
20 INPUT "NUMBER TO CONVERT" N
30 FOR I = 15 TO 0 STEP -1
40 PRINT TEST(N,I);
50 NEXT I
60 PRINT
70 GOTO 20
RUN
BINARY CONVERSIONS
NUMBER TO CONVERT? 45
 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0 1
NUMBER TO CONVERT? #FF
 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1
NUMBER T0 CONVERT? -1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
NUMBER TO CONVERT? 1025
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1
NUMBER TO CONVERT? ^C
BREAK AT LINE 20
OK
```

This program uses TEST to find the binary representation of the number you type. in. Since XYBASIC accepts hexadecimal constants as well as decimal, this program converts either decimal or hexadecimal to binary, as the examples show. You can also convert numbers to binary by using the function BIN$, described below.

Since representations of integer values have 16 bits in XYBASIC, the second argument of TEST is evaluated mod 16.

Logical Operators

XYBASIC lets you use the logical operators AND, OR (inclusive or), XOR (exclusive or) and NOT. Each operates bitwise on its 16-bit integer arguments. That is, the value of bit i of A AND B will be the value of bit i of A ANDed with the value of bit i of B, where i = 0, 1, ..., 15. The truth table for each bit is:

A	B	A AND B	A OR B	A XOR B	NOT A
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	1	0

If you think of 1 as representing true and 0 as representing false, you can see that A AND B is true when both A and B are true; A OR B is true when either is true; A XOR B is true when exactly one is true; and NOT A is true when A is false.

You may want to use logical operators for two different purposes. First, they let you build complicated conditions in logical formulas. For example,

```
IF X = 0 AND ( Y = 1 OR Z <= 10) THEN GOSUB 100
```

allows you to test for several cases with a single IF command. Second, they perform bit manipulation. For example, the Extended XYBASIC command

```
LET I% = (J% AND #F000) OR #FFF
```

sets the most significant four bits of I% to the corresponding bits of J%, and sets the other twelve bits of I% to 1. Since Integer XYBASIC only allows integer variables, this command corresponds to the Integer XYBASIC command

```
LET I = (J AND #F000) OR #FFF
```

The following program demonstrates the logical operators by printing the representations of two values and of the results of applying logical operators to them.

```
NEW
OK
10 INPUT "TYPE TWO NUMBERS" A, B
20 PRINT A, " IS", : TEMP = A : GOSUB 200
30 PRINT B, " IS", : TEMP = B : GOSUB 200
40 PRINT A; "AND"; B; "IS", : TEMP = A AND B : GOSUB 200
50 PRINT A; " OR"; B; "IS", : TEMP = A  OR B : GOSUB 200
60 PRINT A; "XOR"; B; "IS", : TEMP = A XOR B : GOSUB 200
70 PRINT "NOT "; A; " IS", : TEMP = NOT A : GOSUB 200
80 GOTO 10
200 REM SUBROUTINE TO PERFORM BINARY CONVERSIONS
210 FOR I = 15 TO 0 STEP -1
220 PRINT TEST(TEMP, I);
230 NEXT I
240 PRINT
250 RETURN
RUN
```

```
TYPE TWO NUMBERS? 15,255
 15       IS       0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1
 255      IS       0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
 15 AND 255 IS     0  0  0  0  0  0  0  0  0  0  0  0  1  1  1  1
 15  OR 255 IS     0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
 15 XOR 255 IS     0  0  0  0  0  0  0  0  1  1  1  1  0  0  0  0
NOT  15  IS       1  1  1  1  1  1  1  1  1  1  1  1  0  0  0  0
TYPE TWO NUMBERS? -1,2
-1        IS      1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
 2        IS      0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
-1 AND 2 IS       0  0  0  0  0  0  0  0  0  0  0  0  0  0  1  0
-1  OR 2 IS       1  1  1  1  1  1  1  1  1  1  1  1  1  1  1  1
-1 XOR 2 IS       1  1  1  1  1  1  1  1  1  1  1  1  1  1  0  1
NOT -1   IS       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
TYPE TWO NUMBERS? #FFE,65
-32       IS      1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  0
 65       IS      0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  1
-32 AND 65  IS    0  0  0  0  0  0  0  0  0  1  0  0  0  0  0  0
-32  OR 65  IS    1  1  1  1  1  1  1  1  1  1  1  0  0  0  0  1
-32 XOR 65  IS    1  1  1  1  1  1  1  1  1  0  1  0  0  0  0  1
NOT -32  IS       0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1
TYPE TWO NUMBERS? ^C
BREAK AT LINE 10
OK
```

SET and RESET

In a control systems environment you often need to turn on or turn off a particular bit. XYBASIC lets you do so with the SET and RESET functions. The first argument of each is an integer value, the second is the number of the bit you wish to set. Try the following:

```
LET X = SET (0,4)
PRINT X
 16
OK
```

Here XYBASIC SET bit 4 (remember that the least significant bit is bit 0) to 1. RESET changes the specified bit to zero instead of one, so now try

```
LET X = RESET (X,4)
PRINT X
 0
OK
```

Here XYBASIC changed bit 4 of X While leaving all other bits unchanged. In both SET and RESET the second argument is evaluated mod 16, since the range of bit numbers is 0 through 15.

ROTATE, RSHIFT and LSHIFT

If you need to use only certain bits of an integer's value, you might want to use one of the XYBASIC functions ROTATE (for right rotate), RSHIFT and LSHIFT (for right shift and left shift). These functions rotate or shift their first argument the number of binary places specified by the second. Try this:

```
PRINT ROTATE (1,2)
 16384
OK
PRINT LSHIFT (5,3)
 40
OK
```

In the first example XYBASIC ROTATEd 1 right two places to give 16384 (#4000); in the second XYBASIC shifted 5 (binary &101) left three places to give 40 (&10 1000). The following program uses a binary conversion routine to demonstrate ROTATE, LSHIFT and RSHIFT; to exit type <control-C>.

```
NEW
OK
10 INPUT "NUMBER TO BE SHIFTED" NUM
20 INPUT "NUMBER OF PLACES" P
30 PRINT NUM; "     IS       ";
40 TEMP = NUM : GOSUB 200
50 PRINT NUM; "RSHIFTED" P; "IS";
60 TEMP = RSHIFT(NUM, P) : GOSUB 200
70 PRINT NUM; "LSHIFTED"; P; "IS";
80 TEMP = LSHIFT(NUM, P) : GOSUB 200
90 PRINT NUM; "ROTATED "; P; "IS";
100 TEMP = ROTATE(NUM, P) : GOSUB 200
110 PRINT
120 GOTO 10
200 REM SUBROUTINE TO CONVERT TEMP TO BINARY
210 FOR I = 15 TO 0 STEP -1
220 PRINT TEST(TEMP,I);
230 NEXT I
240 PRINT
250 RETURN
RUN
NUMBER TO BE SHIFTED? 1
NUMBER OF PLACES? 2
 1      IS       0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  1
 1 RSHIFTED 2 IS 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 1 LSHIFTED 2 IS 0  0  0  0  0  0  0  0  0  0  0  0  0  1  0  0
 1 ROTATED  2 IS 0  1  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

```
NUMBER TO BE SHIFTED? 400
NUMBER OF PLACES? 9
 400      IS       0  0  0  0  0  0  0  1  1  0  0  1  0  0  0  0
 400 RSHIFTED 9 IS 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
 400 LSHIFTED 9 IS 0  0  1  0  0  0  0  0  0  0  0  0  0  0  0  0
 400 ROTATED  9 IS 1  1  0  0  1  0  0  0  0  0  0  0  0  0  0  0

NUMBER TO BE SHIFTED? #FF
NUMBER OF PLACES? 3
 255      IS       0  0  0  0  0  0  0  0  1  1  1  1  1  1  1  1
 255 RSHIFTED 3 IS 0  0  0  0  0  0  0  0  0  0  0  1  1  1  1  1
 255 LSHIFTED 3 IS 0  0  0  0  0  1  1  1  1  1  1  1  1  0  0  0
 255 ROTATED  3 IS 1  1  1  0  0  0  0  0  0  0  0  1  1  1  1  1

NUMBER TO BE SHIFTED? ^C
BREAK AT LINE 10
OK
```

The next example uses RSHIFT to print the hexadecimal representation of a number. One digit of the hex representation is PRINTed each time the subroutine at line 100 is called.

```
NEW
OK
10 INPUT "NUMBER TO CONVERT" N
20 PRINT N; "IN HEX IS ";
30 FOR I = 12 TO 0 STEP -4
40 GOSUB 100
50 NEXT I
60 PRINT
70 GOTO 10
100 DIGIT = RSHIFT (N,I) AND #F
110 IF DIGIT < 10 THEN PRINT CHR$(48 + DIGIT);
120 IF DIGIT >= 10 THEN PRINT CHR$(55 + DIGIT);
130 RETURN
RUN
NUMBER TO CONVERT? 255
 255 IN HEX IS 00FF
NUMBER TO CONVERT? -1
-1 IN HEX IS FFFF
NUMBER TO CONVERT? 11
 11 IN HEX IS 000B
NUMBER TO CONVERT? ^C
BREAK AT LINE 10
OK
```

BCD and BIN

Many instruments (e.g. digital thermometers and voltmeters) output their measurements in BCD, Binary Coded Decimal. BCD is a representation in which each four bits represent a decimal digit between 0 and 9. Since XYBASIC uses 16-bit integer values, a value can represent 4 BCD digits or 16 binary digits. For example, #1234 represents 4660 if considered as a binary representation, but 1234 if considered as a BCD representation. The functions BCD and BIN are provided to convert between representations. BIN takes a BCD argument and converts it to a binary number, while BCD takes a binary argument and converts it to its BCD representation.

The following program demonstrates BCD and BIN by printing the binary representations of NUM, BCD(NUM) and BIN(NUM) .

```
NEW
OK
10 INPUT "NUMBER" NUM
20 PRINT "REP OF"; NUM; "IS";
30 TEMP = NUM
40 GOSUB 200
50 PRINT "BCD OF"; NUM; "IS";
60 TEMP = BCD(NUM)
70 GOSUB 200
80 PRINT "BIN OF"; NUM; "IS";
90 TEMP = BIN(NUM)
100 GOSUB 200
110 GOTO 10
200 REM SUBROUTINE TO PRINT BINARY REPRESENTATION
210 FOR I = 15 TO 0 STEP -1
220 PRINT TEST(TEMP, I);
230 NEXT I
240 PRINT
250 RETURN
RUN
NUMBER? 17
REP OF 17 IS 0  0  0  0  0  0  0  0  0  0  0  1  0  0  0  1
BCD OF 17 IS 0  0  0  0  0  0  0  0  0  0  0  1  0  1  1  1
BIN OF 17 IS 0  0  0  0  0  0  0  0  0  0  0  0  1  0  1  1
NUMBER? 25
REP OF 25 IS 0  0  0  0  0  0  0  0  0  0  0  1  1  0  0  1
BCD OF 25 IS 0  0  0  0  0  0  0  0  0  0  1  0  0  1  0  1
BIN OF 25 IS 0  0  0  0  0  0  0  0  0  0  0  1  0  0  1  1
NUMBER? 1025
REP OF 1025 IS 0  0  0  0  0  1  0  0  0  0  0  0  0  0  0  1
BCD OF 1025 IS 0  0  0  1  0  0  0  0  0  0  1  0  0  1  0  1
BIN OF 1025 IS 0  0  0  0  0  0  0  1  1  0  0  1  0  0  0  1
```

```
NUMBER? 950
REP OF 950 IS 0  0  0  0  0  0  1  1  1  0  1  1  0  1  1  0
BCD OF 950 IS 0  0  0  0  1  0  0  1  0  1  0  1  0  0  0  0
BIN OF 950 IS

FC ERROR: 90 TEMP = BIN(NUM)

OK
```

If the argument to BIN is not a legal BCD number (i.e. one of its 4-bit components is not a legal decimal digit), an FC (Function Call) error will occur. Similarly, if the argument to BIN is not convertible to a legal 4-digit BCD number (i.e. its value is less than 0 or greater than 9999), an FC error will occur.

HEX$, OCT$ and BIN$

The Extended XYBASIC functions HEX$, OCT$ and BIN$ convert integers to hexadecimal, octal and binary representations.

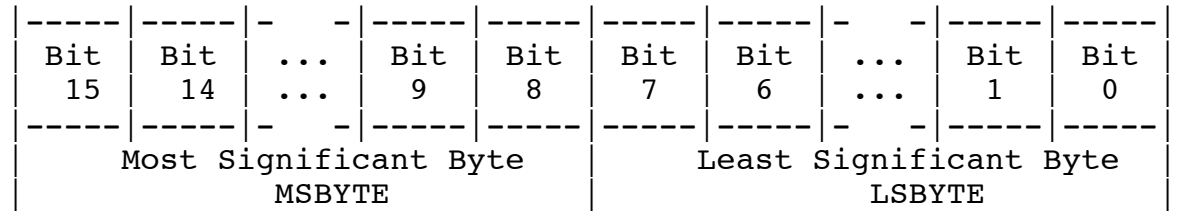
Each of the conversion functions HEX$, OCT$ and BIN$ takes an integer argument and returns a string containing the hexadecimal, octal or binary representation of the argument's value. The following sample program demonstrates the use of these functions.

```
NEW
OK
10 INPUT NUMBER
20 PRINT "THE DECIMAL NUMBER" NUMBER "IS:"
30 PRINT TAB(30); HEX$(NUMBER), "IN HEX,"
40 PRINT TAB(30); OCT$(NUMBER), "IN OCTAL, AND"
50 PRINT TAB(30); BIN$(NUMBER), "IN BINARY."
60 GOTO 10
RUN
? 170
THE DECIMAL NUMBER 170 IS:
                              AA          IN HEX,
                              252         IN OCTAL, AND
                              10101010    IN BINARY.
? 512
THE DECIMAL NUMBER 512 IS:
                              200         IN HEX,
                              1000        IN OCTAL, AND
                              1000000000  IN BINARY.
? ^C
BREAK AT LINE 10
OK
```

Since each conversion function returns a string value, the converted result may be manipulated as desired with other string functions.

MSBYTE, LSBYTE and JOIN

XYBASIC stores integer values internally in 16-bit representations, but the 8080 uses a byte of 8 bits for input, output and data. The functions MSBYTE and LSBYTE allow you to take apart 16-bit integer values into two 8-bit components.



Assume I = 1110 1101 1010 0110 binary. Then MSBYTE (I) = 1110 1101 and LSBYTE (I) = 1010 0110.

Try the following program to get a feeling for MSBYTE and LSBYTE.

```
NEW
OK
10 FOR I = 0 TO 1024 STEP 64
20 PRINT I, MSBYTE(I), LSBYTE(I)
30 NEXT I
OK
RUN
 0              0              0
 64             0              64
 128            0              128
 192            0              192
 256            1              0
 320            1              64
 384            1              128
 448            1              192
 512            2              0
 576            2              64
 640            2              128
 704            2              192
 768            3              0
 832            3              64
 896            3              128
 960            3              192
 1024           4              0

OK
```

Conversely, the JOIN operator lets you concatenate two 8-bit quantities into a 16-bit quantity. For example, say

```
PRINT 1 JOIN 2
 258
OK
PRINT #FF JOIN #FF
-1
OK
```

Here 1 (binary &0000 0001) and 2 (binary &0000 0010) were JOINed to give 258 (binary &0000 0001 0000 0010). Then #FF and #FF were JOINed to give -1 (#FFFF). A BY (BYte) error will occur if either argument of JOIN is not an 8-bit quantity.

The following example uses the OUT command described in Section 8 to show how useful MSBYTE and LSBYTE can be in process control.

Example:
Arnie Zintel of the Margarito Button Company uses an 8080 to control his button assembly line. He wants to send an increasing ramp to the digital to analog converter, which gets a 16-bit number in two 8-bit bytes from ports 0 and 1. He writes the following XYBASIC program:

```
NEW
10 DEF INT I
20 FOR I = 1 TO 1000
30 OUT 0,LSBYTE (I)
40 OUT 1,MSBYTE (I)
50 NEXT I
```

This program obtains the necessary 8-bit quantities with the MSBYTE and LSBYTE functions, which are then used as arguments of the OUT command.

GET

The GET function lets you check whether a character has been typed while a program is running. If a character has been typed, GET returns its ASCII value, as given in Appendix 5. GET returns 0 if no character has been typed. You can use GET to define control characters to monitor program execution without using <control-C> and CONT. The following simple example just increments the value in I, and prints its value whenever you type <control-T> (ASCII 20).

```
NEW
OK
10 IF GET=20 THEN GOSUB100
20 I = I + 1
```

```
30 GOTO 10
100 PRINTI;
110 RETURN
RUN
 46  100  126  169  183  201 ^C
BREAK AT LINE 20
OK
```

Since XYBASIC automatically removes the parity bit from any character it reads, you cannot GET a value greater than 127 (7F hexadecimal). And of course you should not try to GET characters such as <control-C> which have special meanings to XYBASIC.

You might want to use GET to let the user answer a question with Y or N. For example, the MOD program in Section 3 could be modified as follows:

```
NEW
OK
10 INPUT "A, B =" A, B
20 PRINT A; "\"; B; "="; A \ B
30 PRINT A; "MOD"; B; "="; A MOD B
40 PRINT "ANOTHER (Y OR N)?";
50 X = GET
60 IF X = 0 THEN 10
70 PRINT CHR$(X)
80 IF X = 89 THEN 10
RUN
A, B =? 10,3
 10 \ 3 = 3
 10 MOD 3 = 1
ANOTHER (Y OR N)?Y
A, B =? 15,4
 15 \ 4 = 3
 15 MOD 4 = 3
ANOTHER (Y OR N)?N

OK
```

Here XYBASIC executes lines 50 and 60 until a character is typed, making X = 0 false. Then the typed character is echoed with CHR$, and if the character is Y (ASCII 89) the program returns to line 10. Notice that the program would NOT work correctly if lines 50 through 80 were replaced by

```
50
60 IF GET 0 THEN 60
70 PRINT CHR$(GET)
80 IF GET = 89 THEN 10
```

In this case line 60 waits for a character, but its value is not saved and line 70 tries to GET the next character (and probably PRINTS ASCII O, a null character). Then line 80 tries to GET another character.

In Extended XYBASIC you can use the string function GET$ as well as the numeric function GET. GET$ is described in Section 4.

DELAY

For many applications you may want your computer to act like a stop watch, measuring a certain amount of time. XYBASIC allows you to do this with the DELAY command. Its first argument specifies the number of minutes you wish to delay, its second the number of seconds, and its third the number of hundredths of seconds. For instance,

```
DELAY 0, 4, 50
```

will delay for four and one-half seconds. The second and third arguments are optional.

You can interrupt a DELAY with <control-C>, suspend it with <control-S>, or abort it and proceed with the next instruction by typing any other character. ENABLEd interrupts (described in Section 10) are not active during a DELAY.

Unless you recalibrate DELAY with the TIME command, XYBASIC assumes your machine to be a standard 2 MHz 8080 with 500 ns memory and no wait states. If your machine is nonstandard, DELAY will not work correctly until you use TIME to calibrate it!

The following program gets the current time from the user and then PRINTs the time at intervals of roughly five seconds. Of course the overhead of executing other XYBASIC instructions makes the time between successive exections of the line 20 PRINT command slightly longer than five seconds; the DELAY in line 70 could naturally be modified to compensate for the overhead.

```
NEW
OK
10 INPUT "TIME" H, M, S
20 PRINT H; ":"; M; ":"; S
30 S = S + 5
40 IF S >= 60 THEN GOSUB 100
50 IF M >= 60 THEN GOSUB 200
60 IF H >= 24 THEN H = H - 24
70 DELAY 0, 5
80 GOTO 20
100 S = S - 60 : M = M + 1 : RETURN
200 M = M - 60 : H = H + 1 : RETURN
RUN
```

```
TIME? 7,15,45
 7 : 15 : 45
 7 : 15 : 50
 7 : 15 : 55
 7 : 16 : O
 7 : 16 : 5
^C
BREAK AT LINE 70
OK
```

TIME

The TIME statement calibrates the DELAY command for systems which run at different speeds than a standard 8080 with a 2 MHz clock and 500 ns memories; this includes systems using the Z-80, 8085, and NEC 8080. TIME prompts you with a bell, audible on most terminals, and then waits for two carriage returns separated by exactly 60 seconds. This 60 second interval is used as the standard for executing subsequent DELAY commands. If you type <control-C> during execution of TIME, the previous calibration is retained.

Section 9: Machine Control Functions

XYBASIC is designed to be especially useful for control applications. The functions described in this section let you examine and control external devices connected to your computer in a particularly simple and straightforward fashion, and to examine and modify specific locations in your computer's memory. When combined with the bit manipulation features discussed in Section 8, these functions let you write easily understood XYBASIC control programs instead of assembly language programs. And by interacting with XYBASIC you can get your control programs running correctly much more quickly.

OUT

An important feature of XYBASIC is the ability to communicate with external devices, i.e. to perform port input and output. The OUT command and the IN function work like the assembly language commands OUT and IN. For instance, to OUTput the value 1 to port 55 you just say

```
OUT 55,1
```

Either argument can be any formula, but each must evaluate to an 8 bit quantity or else a BY (BYte) error will occur.

Example:
Buck Mulligan works at XYZ Manufacturing Co., which just bought an 8080 to control their widget production. Output port 17 is supposed to control the widget former, which uses a nonstandard connector. Buck uses XYBASIC to help him learn which connector pin corresponds to which bit on the port.

```
NEW
10 INPUT "BIT NUMBER" 1
20 OUT 1?, SET (0, I)
30 OUT 1?, 0
40 GOTO 20
```

After typing RUN, Buck responds to the INPUT prompt at line 10 with a bit number between 0 and 7. XYBASIC then turns the desired bit of output port 17 on and off while Buck checks the connector pins with a probe. After he finds and labels the pin, he types <control-C> and RUN to test another bit.

IN

The IN function allows you to perform port input, the equivalent of an assembly language IN instruction. If you say

```
LET X = IN (10)
```

XYBASIC will find the value on input port 10 when the command is executed and assign it to the variable X. The input port number can be any formula (allowing the program itself to determine which port to read), but it must evaluate to an 8 bit quantity or a BY (BYte) error will occur.

Example:
Harry Matthews is in charge of widget quality control for XYZ "
. "Manufacturing. He wants to measure widget resistance with a 16-bit digital ohmmeter attached to input ports 12 and 13, but like Buck Mulligan (in the OUT example above) he is confused about which bit each connector pin represents. Harry uses the following XYBASIC program:

```
NEW
10 X = IN (12) JOIN IN (13)
20 IF X = #FFFF THEN 10
30 FOR I = 0 TO 15
40 IF TEST (X, I) = 0 THEN PRINT "^GBIT #"; I
50 NEXT I
```

Line 10 constructs the value read from the ohmmeter ports, and line 20 loops until Harry puts a signal on a connector pin. Then the FOR loop starting at line 30 rings a bell (with <control-G>) and tells Harry which bit the pin represents.

PEEK

The PEEK function allows you to examine any memory location in your computer system. If you use memory mapped I/O (that is, if specific locations are used for input and output), you can use PEEK to perform inputs. To find out what is in location 10 of your computer's memory, just type

```
PRINT PEEK (10)
```

and XYBASIC will print the current value of location 10. The PEEK function will always return an 8 bit value (i.e. a number between 0 and 255).

The following program determines the values of the first 10 locations in memory and prints them on the console. Note that I goes from 0 to 9, since the address space of the 8080 starts at location 0 by convention.

```
NEW
10 FOR I = 0 TO 9                                    'SET UP LOOP
20 PRINT "LOCATION"; I; "CONTAINS"; PEEK (I)     'PRINT VALUE
30 NEXT I                                            'DO THE NEXT ONE
```

Although the largest integer value that can be represented in XYBASIC is 32767, you can PEEK at locations above 32767 by using negative integer arguments, because PEEK considers its argument to be an unsigned integer representation. You can use the UNS function described in Section 3 to see the actual location that PEEK will examine, as the following program shows.

```
NEW
OK
10 FOR I=-32767 TO -1
20 PRINT I, UNS(I)
30 NEXT I
-32767         32769
-32766         32770
-32765         32771
^C
BREAK AT LINE 20
OK
```

Use <control-S> to suspend execution so you can look at the values, and use <control-C> when you wish to exit from this program.

POKE

In addition to letting you to examine memory locations with the PEEK function, XYBASIC allows you to put data directly into memory with the POKE command. If your system uses memory mapped I/O, you can use POKE to perform output. To modify location 4 to contain 0, just say

```
POKE 4,0
```

Don't try this unless you are sure modifying location 4 will not endanger your system! POKE is very dangerous -- you can easily modify the operating system or the XYBASIC interpreter itself with careless POKEing, with unpredictable consequences -- so be extremely careful!

Example:
Herman Wayl has just bought a new memory board for his 8080 and wants to test it. It is an 8K board which occupies address space 16K to 24K. Herman tests it for bit failures with the following program.

```
NEW
10 FOR I = 16*1024 TO 24*1024 - 1          'SET UP LOOP PARAMETER
20 N = 0                                   'CLEAR TEST VALUE BITS
```

```
30 GOSUB 100                                    'CHECK FOR ERROR
40 N = #FF                                      'SET TEST VALUE BITS
50 GOSUB 100                                    'CHECK FOR ERROR
60 NEXT I                                       'AND TRY NEXT LOCATION
70 PRINT "TEST CONCLUDED"                       'DONE
80 END
100 REM SUBROUTINE TO TEST LOCATION I WITH VALUE N
110 POKE I, N
120 IF PEEK (I) = N THEN RETURN                 'VALUE IS CORRECT
130 PRINT "FAILURE AT LOCATION"; I; ":"; PEEK (I); "SHOULD BE"; N
140 RETURN
```

This program POKEs a 0 into a memory location on the new board, then PEEKs to check that the data reads back correctly and PRINTS an appropriate error message if it does not. Then the test is repeated with #FF (all 1's). After testing all memory locations, the program prints a concluding message.

SENSE

The SENSE function allows you to find the value of a single bit of an input port, and might for example be used to find the status of a switch attached to a port. To find the status of bit 14 on port 7, just say

```
PRINT SENSE (7,4)
```

or

```
LET X = SENSE (7,4)
```

The latter command assigns the value (0 or 1) of bit 14 on port 7 to the variable X. A BY (BYte) error will occur if the first argument is not between 0 and 255, since the 8080 only has 256 input ports. Try the following program; note that the result of RUNning it will vary depending on the use of port 5.

```
NEW
OK
10 FOR I = 0 TO 7
20 PRINT SENSE(5,I);
30 NEXT I
RUN
 1  0  1  1  0  0  0  0
OK
```

WAIT

Sometimes you want your program to wait (suspend processing) until an external event signals it to continue. This event could be a switch closing, a temperature exceeding a given value, or an electronic device indicating that it is through with a task. The WAIT command allows you to do this, WAITing until a given input port has a given value before executing the next command in your program. For example, if you say

```
WAIT 10,0
```

then XYBASIC waits until the value on input port 10 is 0 before responding with its OK prompt. The optional third argument of WAIT is a mask which lets you ignore the values of bits you don't care about when trying to match the value. Bits set to 1 in the mask are ignored, so

```
WAIT 10,0,&11111100
```

will wait until bits 0 and 1 of input port 10 are 0, ignoring bits 2-7. Notice that the mask byte contains 0 in bits which are significant and 1 in bits which are ignored, not vice versa.

The optional fourth argument of WAIT is $ (dollar sign), which tells XYBASIC that the WAIT should end if the value of ANY unmasked bit on the port matches the given value. Thus

```
WAIT 10,&10,&11111100
```

tells XYBASIC to WAIT until either bit 0 of input port 10 is 0, or bit 1 of input port 10 is 1, or both. As before, the values of all bits set to 1 in the mask are ignored.

Although normal execution stops during a WAIT, ENABLEd interrupts (see Section 10 below) remain active, and special characters such as <control-C> and <control-S> have their usual effect.

Example:
Emil Post works (sometimes) at the Do It Later Corporation. He wants his 8080 to warn him when his boss arrives, which trips a switch on one of five doors. The switches are connected to bits 0-4 of input port 5, and go high (i.e. become 1) when a door opens. Emil uses the following program:

```
NEW
OK
10 WAIT 5,&11111,&11100000,$
20 PRINT "WAKE UP!  HERE HE COMES!"
```

Line 10 WAITs until port 5 has a 1 on any of bits 0-4. The mask indicates that bits 5-7 are ignored, and the option indicates that the WAIT ends when ANY of the doors are opened. Notice how simple it is to specify values and masks by using binary literals.

Section 10: Interrupts

The ENABLE feature gives XYBASIC the ability to handle interrupts and thus allows concurrent processing; you can continually check for the occurrence of an event while running a program.

ENABLE

Sometimes you may want to monitor or control external devices continuously while simultaneously executing a program. XYBASIC lets you use the ENABLE command to implement interrupts which do so. ENABLE uses the same syntax as the WAIT command, described in Section 9 above. Its operation is similar to WAIT, except that XYBASIC does not suspend processing until the condition you specify is fulfilled. Instead, the condition is checked before executing each program command (although no checking occurs before direct mode commands are executed). If it is not fulfilled, the program command is executed. If it is fulfilled, program execution is interrupted and the subroutine located at the line number specified in the ENABLE is executed instead. When the routine RETURNs, the interrupted program is resumed.

For example, suppose you wish to perform some computations and simultaneously print data on a lineprinter as fast as possible. If bit 0 of output port 6 becomes 1 when your printer is ready to receive more data, you can use ENABLE as in the following program fragment.

```
NEW
10 ENABLE 100, 6, 1, &11111110
20 GOTO 200
100 'SUBROUTINE TO SEND NEXT DATA TO PRINTER
...
190 RETURN
200 'MAIN PROGRAM
...
```

Here the main program is executed until bit 0 of port 6 becomes 1. Then the program is interrupted, and the subroutine at line 100 sends data to the printer. The RETURN of line 190 causes the main program to be resumed where it was interrupted.

Whenever an interrupt occurs, it is suspended (that is, its condition is not checked) until the return from the specified routine; this prevents the interrupt from interrupting itself. You must be careful not to RETURN from an interrupt routine unless your program has either DISABLEd the interrupt (as explained below) or done something to make the ENABLE condition false. Otherwise the interrupt will occur again immediately after the RETURN, and only the interrupt routine will be executed. Interrupts remain active during WAIT but not during

TIME and DELAY.

You may not ENABLE more than eight interrupts at once; if you try to ENABLE more than eight, an EN (ENABLE) error will occur. The order of ENABLEing determines the priority of the interrupts; the conditions are tested in the order of the ENABLE commands. ENABLE is legal only in program mode; an ID (Illegal Direct) error will occur if you use it in direct mode,

Since XYBASIC tests the condition specified by each interrupt before executing each command, interrupts slow down XYBASIC considerably. You should notice that ENABLE interrupts are controlled by software, not using the hardware interrupt facilities of your computer. Of course you can use the CALL and SCALL commands to access machine language routines for activation and servicing of hardware interrupts.

DISABLE

You can deactivate all ENABLEd interrupts by just saying DISABLE. Alternatively, you can deactivate a specific interrupt:

```
DISABLE 10
```

disables only the interrupt which you ENABLEd in line 10. If you try to DISABLE a nonexistent interrupt, an EN error will occur. XYBASIC also disables all interrupts when you execute a RUN.

Example:
In the WAIT example of Section 9, Emil Post used the following program to warn him of the arrival of his boss:

```
NEW
10 WAIT 5, &11111, &11100000, $
20 PRINT "WAKE UP!  HERE HE COMES!"
```

Emil has now been given a XYBASIC program which plays blackjack, but he does not want his boss to catch him playing. He therefore uses ENABLE instead of WAIT, allowing him to play blackjack and still be warned when his boss arrives.

```
NEW
10 ENABLE 20, 5, &11111, &11100000, $
15 GOTO 100
20 PRINT "BET FAST! HERE HE COMES!"
25 DISABLE 10
30 RETURN
100 'START OF BLACKJACK PROGRAM
...
```

The ENABLE command of line 10 specifies that an interrupt will occur when the value of any of bits 0 through 4 of input port 5 becomes 1. After the ENABLE the program plays blackjack with Emil. When his boss enters, one of bits of port 5 becomes 1 and an interrupt occurs, transferring control to the subroutine at line 20. A warning message is printed by line 20. Line 25 DISABLEs the interrupt, so that the message will not be repeated even though one of the bits of port 5 is still 1, and line 30 RETURNs control to the blackjack program at the point it was interrupted.

Section 11: Machine Language Linkage

You may find it necessary to perform a task with a routine written in assembly language. You might need the speed of an assembly language routine, or need to conserve memory, or need to use a machine control feature not provided by XYBASIC, or you may want to use an already written assembly language program (for example to process hardware interrupts). XYBASIC lets you access machine language routines stored anywhere in your computer's memory with the CALL and SCALL commands.

CALL

CALL lets you access an assembly language routine. For example,

```
CALL #A000
```

executes an assembly language CALL to the routine at location 0A000H. When the routine executes an assembly language RETurn instruction, the next command after the CALL in the XYBASIC program is performed. The location may be specified by any numeric formula.

One of the few machine control functions which is not directly executable under XYBASIC is to enable and disable the computer's hardware interrupt facility. You can accomplish this with machine language linkage. Suppose you have the following machine language routines stored in ROM at location OFCOOH.

```
FC00                ORG      0F000H   ;ROM ROUTINE ADDRESS
FC00 FB    ENAB:    EI                ;ENABLE INTERRUPTS
FC01 C9             RET               ;RETURN TO XYBASIC
FC02 F3    DISAB:   DI                ;DISABLE INTERRUPTS
FC03 C9             RET               ;RETURN TO XYBASIC
```

These routines may be called with the CALL command. For example,

```
CALL #FC00
```

will enable hardware interrupts. The following program fragment assigns the addresses of the machine language routines to variables to make the function of the CALLs clear.

```
10 ENAB = #FC00       'ADDRESS OF ENABLE ROUTINE
20 DISAB = #FC02      'ADDRESS OF DISABLE ROUTINE
...
100 CALL DISAB        'DISABLE INTERRUPTS FOR SENSITIVE CODE
```

```
...
190 CALL ENAB         'ENABLE INTERRUPTS
...
```

This example did not require the passing of information between XYBASIC and machine language. But it is often necessary to pass information to a machine language routine.

To make CALL more useful, XYBASIC lets you pass information, called parameters. You can specify as many parameters as you wish on the same line as the CALL command. For example (in Extended XYBASIC):

```
CALL #A000, I%, S$, Y(1,2),*Z
```

Each parameter may be either a variable reference, or the character * followed by an array variable name. To find the value of a parameter, your machine language routine calls the subroutine GTPAR, which returns information about the next parameter in the CALL command's parameter list. Chapter II describes how to access GTPAR in your version of XYBASIC. You should execute a machine language CALL to location 103H in CP/M versions, location 3283H in ISIS-II versions, or location 103H in Custom I/O versions of XYBASIC.

GTPAR returns the type of the next parameter in the A register: 0 if no more parameters, 1 if integer, 2 if string, 3 if floating. The B register returns the number of bytes in the parameter's value: 2 if integer, 3 if string, 4 if floating, as explained in greater detail below. The C register returns the number of dimensions of the parameter: 0 if a simple variable or an array element, n if the parameter is * followed by the name of an n-dimensional array variable. For simple variables and array elements, HL returns the address of the parameter's first value byte. For arrays, DE returns the address of the first dimension byte and HL the address of the first value byte.

For GTPAR to be useful you need to understand how XYBASIC stores values. Integer values are represented by two bytes in two's complement representation, with the least significant byte first in keeping with 8080 convention. For example, the integer value 10 is stored as the two bytes 0AH, 00H; the integer value -10 is stored as the two bytes 0F6H, 0FFH.

String values are stored in three bytes. The first contains the length of the string. The second and third are meaningless if the length is 0 (null string); otherwise they give the location (in string space) of the first character of the string. Successive characters are stored in successive locations in string space. For example, the string value "ABCD" contains 4 characters, so it is stored as a length byte containing 4 followed by two bytes giving a location in string space. The four characters "A", "B", "C" and "D" are stored in ASCII as the byte values 41H, 42H, 43H and 44H, starting at the given location in string space.

Floating point values are stored in four bytes. The first byte is zero and the remaining bytes meaningless for a floating point zero. Otherwise the first byte contains the binary exponent of the normalized value, with a bias of 80H. Bytes 2 - 11 contain the binary mantissa of the normalized value, with an assumed binary point preceding byte 2 and an assumed 1 replacing bit 7 of byte 2. Bit 7 of byte 2 contains the sign of the value, 0 if positive and 1 if negative. For example, the floating point value 10.5 decimal is equal to 1010.1 binary, which is normalized as .10101 binary times 2 to the 4th power. Therefore 10.5 decimal is represented by the four bytes 84H, 28H, OOH, OOH. The first byte contains the biased exponent (80H + 4H = 84H). Bit 7 of byte 2 is 0, indicating that the value is positive. Bytes 2, 3 and 4 contain the binary mantissa, with the leading 1 "hidden" by the sign bit.

For an array, the first element is stored in the first location and successive elements of the array are stored by rows. For example, a floating point array declared with DIM A(2,3) is stored A(0,0), A(0,1), ..., A(0,3), A(1,0), ..., A(1,3), A(2,0), ..., A(2,3). Each element is a floating point value, and is therefore stored in four bytes.

The following routines illustrate the use of GTPAR. They move the values of XYBASIC variables to and from memory. They could be used to save and load variable values on a mass storage device such as a digital cassette drive which communicates with memory. They could also be used to save variables during a power failure if a portion of the computer's memory had battery backup.

The routine STOVAR expects two parameters, an integer variable giving the memory location to be used as the storage destination and the variable to be saved. Similarly, the routine RCLVAR expects two parameters, an integer variable giving the memory location used and the variable to be recalled. The machine language routines are as follows:

```
0100 =          XYBASIC EQU     100H        ;START OF XYBASIC
0103 =          GTPAR   EQU     XYBASIC+3   ;GTPAR ENTRY POINT
0168 =          ERROR   EQU     XYBASIC+68H ;ERROR ENTRY POINT
F000                    ORG     0FC00H
                ;STORE VARIABLE
FC00 CD3CFC     STOVAR: CALL    GETIP   ;STORE ADDRESS IN DE
FC03 D5                 PUSH    D       ;SAVE STORE ADDRESS
FC04 CDO301             CALL    GTPAR   ;POINTERS TO VARIABLE
FC07 D1                 POP     D       ;RESTORE STORE ADDRESS
FC08 B7                 ORA     A       ;PARAMETER PRESENT?
FC09 CA6801             JZ      ERROR   ;NO - TAKE ERROR EXIT
FC0C FE03               CPI     2       ;STRING VARIABLE?
FC0E C230FC             JNZ     MOVER   ;NO - ALL SET TO MOVE
FC11 46                 MOV     B,M     ;LENGTH OF STRING TO B
FC12 23                 INX     H       ;POINT TO STRING ADDRESS
FC13 7E                 MOV     A,M     ;LOW STRING ADDRESS
FC14 23                 INX     H
FC15 66                 MOV     H,M     ;HIGH STRING ADDRESS
```

```
F016 6F              MOV     L,A     ;STRING ADDRESS TO HL
FC17 C330FC          JMP     MOVER   ;STORE IT
                ;RECALL VARIABLE
FC1A CD3CFC     RCLVAR: CALL    GETIP   ;RECALL ADDRESS TO DE
FC1D D5              PUSH    D       ;SAVE RECALL ADDRESS
FC1E CD0301          CALL    GTPAR   ;POINTERS TO VARIABLE
FC21 D1              POP     D       ;RESTORE RECALL ADDRESS
FC22 EB              XCHG            ;RECALL ADDRESS TO HL,
                                     ;VARIABLE ADDRESS TO DE
FC23 FE03            CPI     2       ;STRING VARIABLE?
FC25 C230FC          JNZ     MOVER   ;NO - ALL SET TO MOVE
FC28 EB              XCHG            ;VARIABLE ADDRESS TO HL
FC29 46              MOV     B,M     ;STRING LENGTH TO B
FC2A 23              INX     H
FCZB 7E              MOV     A,M     ;LOW STRING ADDRESS
FC2C 23              INX     H
FC2D 66              MOV     H,M     ;HIGH STRING ADDRESS
FC2E 6F              MOV     L,A     ;STRING ADDRESS TO HL
FC2F EB              XCHG            ;RECALL ADDRESS TO HL,
                                     ;STRING ADDRESS T0 DE
                                     ;MOVE FROM RECALL ADDRESS
                                     ;TO STRING AND RETURN

               ;MOVER MOVES B BYTES FROM THE ADDRESS IN HL TO
               ;THE ADDRESS IN DE
FC30 78         MOVER:  MOV     A,B
FC31 B7                 ORA     A       ;IS LENGTH ZERO?
FC32 C8                 RZ              ;YES - RETURN TO XYBASIC
FC33 7E         MOVES:  MOV     A,M     ;FETCH SOURCE BYTE
FC34 12                 STAX    D       ;STORE IT
FC35 23                 INX     H       ;INCREMENT SOURCE POINTER
FC36 13                 INX     D       ;INCREMENT DEST POINTER
FC37 05                 DCR     B       ;DECREMENT BYTE COUNTER
F038 C233FC             JNZ     MOVES   ;KEEP MOVING
FC3B C9                 RET             ;RETURN TO XYBASIC
                ;GETIP GETS AN INTEGER PARAMETER T0 DE.
                ;JUMPS TO ERROR IF PARAMETER IS NOT
                ;A SIMPLE INTEGER VARIABLE.
FC3C CD0301     GETIP:  CALL    GTPAR   ;GET A PARAMETER
FC3F FE01               CPI     1       ;INTEGER?
FC41 C26801             JNZ     ERROR   ;NO - TAKE ERROR EXIT
FC44 79                 MOV     A,C     ;EXPECTING SIMPLE VARIABLE
FC45 B7                 ORA     A       ;ARRAY PASSED?
FC46 C26801             JNZ     ERROR   ;YES - TAKE ERROR EXIT
FC49 5E                 MOV     E,M     ;LO BYTE OF VALUE
FC4A 23                 INX     H       ;POINT TO HIGH BYTE
FC4B 56                 MOV     D,M     ;HIGH BYTE
FC4C C9                 RET             ;RETURN TO CALLER
```

With these routines resident in memory, it is a simple matter to save the contents of a string variable in nonvolatile memory (say, at address 0C000H). The string variable DATE$ could be saved as follows:

```
NVM% = #C000: CALL #FC00, NVM%, DATE$
```

It could be recalled at power-up time by:

```
NVM% = #COOO : DATE$ = "          " : CALL #FC1A, NVM%, DATE$
```

Note that a string of blanks is assigned to DATE$ before the call to RCLVAR. This is an absolutely necessary formality, because storage space for string variables is allocated dynamically by XYBASIC. This allows the length of a string variable's value to vary dynamically without wasting memory space. If a string variable contains the null string (as it does if it has not been previously defined), no storage space has been allocated for it. Thus a string variable must be assigned a string of length at least as great as the length of the string to be recalled before the CALL to RCLVAR. This extra step is necessary only for string variables.

Another example using GTPAR is included in Section 3 of Chapter II, in the paragraph Saving and Loading Under Operating Systems.

SCALL

The SCALL (Short CALL) command is similar to CALL, except that parameter values are passed to and from your assembly language routine directly (through registers) rather than with GTPAR. This makes linking to an assembly language routine faster and easier. However, the parameters which are passed must be integers; you must use CALL rather than SCALL to pass floating point or string values. You also may not pass more than three integer values with SCALL. If you say (in Extended XYBASIC):

```
SCALL #8000, I%, J%, K%
```

then XYBASIC will first load registers BC, DE and HL with the values of I%, J% and K%, and then execute an assembly language CALL to the routine at location 8000H. Executing an assembly language RETurn will return control to the next command after the SCALL, and the values in registers BC, DE and HL will be assigned to the variables I%, J% and K%.

As with CALL, the location of the machine language routine may be specified by any formula. However, the parameters of SCALL must be integer variables. The parameters are optional, but a MC (Machine language Call) error will occur if you try to pass more than three parameters, or if you use a non-integer variable as a parameter.

Note that the SCALLed routine must preserve the values in BC, DE and HL if you wish to leave the parameter values unchanged! If you need to use an assembly language routine which does not preserve registers, you can assign the parameters to temporary variables first. In the above example you could for example say:

```
LET ITEMP% = I%
LET JTEMP% = J%
LET KTEMP% = K%
SCALL #8000, ITEMP%,  JTEMP%,  KTEMP%
```

where ITEMP%, JTEMP% and KTEMP% are integer variables not used elsewhere in your XYBASIC program. The values of these temporaries may be altered by SCALL, but the values of I%, J% and K% and will remain unchanged.

Of course the above examples should be written without the % signs in Integer XYBASIC, since only integer variables are allowed.

Example:
Archie Goodwin must write a program to control a high speed stepper motor and print positional information after its rotation is complete. The motor moves one step for each output to port 0, regardless of the value output, and accepts a new step pulse every millisecond. Archie could issue pulses to the motor with an OUT command inside a FOR/NEXT loop. However, his stepper is capable of a much higher step rate; XYBASIC is a high level interpretive language and cannot compete with machine language for speed. But the data manipulation required by the program is very difficult for Archie to perform in machine language.

Archie's solution is to code only the time critical section for motor control in machine language and write the rest of his task in XYBASIC. The following simple machine language routine provides the needed speed.

```
FC00                    ORG     0FC00H
FC00 78         STEPN:  MOV     A,B     ;NUMBER OF STEPS IS IN BC
FC01 B1                 ORA     C       ;ZERO STEPS LEFT?
FC02 C8                 RZ              ;YES - RETURN TO XYBASIC
FC03 D300               OUT     STEP    ;NO - SEND STEP PULSE
FC05 3E83               MVI     A,83H   ;TIMING CONSTANT FOR 1MS
F007 3D         DELAY:  DCR     A       ;DONE WITH DELAY?
FC08 C207FC             JNZ     DELAY   ;NO - KEEP LOOPING
FCOB 0B                 DCX     B       ;DECREMENT STEP COUNTER
FCOC CBOOFC             JMP     STEPN   ;AND SEE IF DONE STEPPING
```

With this routine in memory, the stepper can be turned at any time by setting the variable I% to the desired number of steps and executing the command :

```
SCALL #FC00,I%
```

After execution of the machine language routine the value of the variable I% will always be zero. The machine language routine decrements the BC register pair to zero to determine when enough step pulses have been sent. When the subroutine returns, XYBASIC assigns the new value in the BC register pair to the variable I%, making it zero.

Section 12: ROMSQuared Features

Conventional BASIC interpreters must reside in RAM, and allow only one user program to be in memory at any time, also in RAM. The Custom I/O version of the XYBASIC interpreter, on the other hand, may reside in either RAM or ROM.

Unlike other BASIC interpreters, XYBASIC allows several user programs to be in memory simultaneously, in either RAM or ROM, and lets you use a simple XYBASIC command to switch between them. The unique features which make this possible allow both the interpreter and the user program to reside in ROM, and therefore are called ROMSQuared features.

You can use the ROMSQuared features described in this section to execute an application program as soon as you turn on your computer. This allows you to build convenient stand-alone systems with ease. You can debug a program in RAM and then burn it and execute it out of PROM.

Working Space

CP/M and ISIS-II versions of XYBASIC reside in one contiguous segment of RAM. Custom I/O versions reside in one area of memory, either RAM or ROM, and use a separate area of RAM. In any version, the contiguous segment of memory from the first RAM location used by XYBASIC to the location specified in response to the END OF MEMORY prompt during initialization must be RAM, and is used as working space by the interpreter. Any segment of memory not used by XYBASIC may be used to store machine language routines or XYBASIC user programs.

The working space always contains a XYBASIC program (initially, an empty program), and the commands you give normally refer to that program. The commands described in this section allow you to copy programs to and from working space, and to execute programs residing elsewhere in memory, either in RAM or ROM.

MOVE

The command MOVE allows XYBASIC programs to be moved to or from the working space. For example,

MOVE TO #8000

copies the current program from working space to memory (RAM) starting at location 8000H. Thus MOVE TO creates a new image of the current program, which might then be burned into PROM.

The internal representation of a XYBASIC program is completely relocatable (position independent); it may be executed from any address. Therefore a PROM programmed with a XYBASIC program may be plugged into any available socket.

XYBASIC checks that the address given in the MOVE TO command does not overlap the interpreter or working space, and that there is sufficient RAM at the specified address to store the program; if not it issues an RO (ROmsq feature) error. The program in the working space is unchanged by he MOVE operation.

Similarly, the command

```
MOVE FROM #8000
```

copies a XYBASIC program from memory (RAM or ROM) to working space. An RO error occurs if the specified location is within the interpreter or working space, or if it is not the start of a XYBASIC program. An OM error occurs if the working space is too small to contain the program. (In the latter case, CLEAR might provide the extra space required for the program.) A successful MOVE FROM will of course destroy the previous program in the working space.

You can use MOVE FROM to fetch a XYBASIC program from ROM into the RAM working space, modify the program, and then burn a new PROM containing the modified program. The MOVE command thus makes it easy to manipulate XYBASIC programs on ROM based systems.

EXEC

The EXEC command lets XYBASIC access programs outside the working space. If you wish to RUN the program at location 8000H, you type

```
EXEC #8000
```

If the specified address is not the start of a XYBASIC program, or is within the interpreter or working space, an RO error occurs. Otherwise all subsequent XYBASIC commands will refer to the specified program rather than to the program in working space. You just type LIST to list it, RUN to execute it, <control-C> to interrupt it, and so on. If you type

```
EXEC
```

(without an address specified), the program stored in working space becomes the current program again.

The location you specify in the EXEC command may be either RAM or ROM. Therefore you can use EXEC to access programs stored in PROMs, which are available immediately (without LOADing) when you turn on your computer.

EXEC has no effect on either the program in working space or the program at the specified address. Rather, it simply tells XYBASIC to which program subsequent commands will refer. Notice that the MOVE command does not change which program is addressed; you must perform an EXEC after a MOVE if you wish to address the MOVEd copy of the program.

Only the program in working space may be edited. If you attempt to add, delete or alter a line after using EXEC to refer to a different program, an RO error will occur. Of course the editing may be performed by MOVEing the program FROM its location to working space, editing it, and finally MOVEing it back TO its location. Similarly, the NEW and LOAD commands apply only to the program in working space; an RO error occurs if you type NEW or LOAD while referring to a program specified with EXEC. The SAVE command will always SAVE the current program, whether in working space or elsewhere.

Another form of the EXEC command allows execution of chains of XYBASIC programs without user intervention. For example, in the command

```
EXEC #A000, G
```

the address #A000 specifies the location of a XYBASIC program, as usual. The suffix ,G indicates that XYBASIC should execute the specified program immediately (by doing a GOTO to its first line), rather than returning to direct mode after performing the EXEC command. The EXEC <location>, G command does not clear variables. However, it does destroy any GOSUB/RETURN and FOR/NEXT context, making it illegal to have a FOR statement in one program and the matching NEXT in another. Of course, a CLEAR command can be included as the first line of the specified program; in this case the command acts like an EXEC immediately followed by a RUN.

The command

```
EXEC ,G
```

with the address omitted but the suffix remaining, tells XYBASIC to execute the program currently in its RAM workspace.

You can use EXEC <location>, G to build complex systems containing several independent programs. A master control program can type a list of available programs (called a menu), and allow the user to choose which program he wishes by typing a character. Then by executing an appropriate EXEC command depending on the character, the master can transfer control directly to the chosen program.

One additional type of RO error may occur in conjunction with EXEC. If the location specified in the EXEC command is ROM (EXEC is especially useful for executing programs stored on PROMs), line breakpoints may not be used. When a line BREAK or UNBREAK command is executed in this situation, an RO error occurs and the command is ignored.

FIRST and LAST

You may sometimes want to know the location of a XYBASIC program, for example to burn it into PROM. Therefore the functions FIRST and LAST return the locations of the first and last bytes of the current program. The command

```
PRINT FIRST, LAST, LAST-FIRST+1
```

prints the location and length of the current program, whether in working space or not.

Suppose that you have a ROM based XYBASIC system, including a PROM burning routine at location 0A000H. Assume also that the PROM burning routine operates by burning the bytes starting at RAM location 0C000H, and that registers BC tell the routine how many bytes are to be burned. If the routine is successful, it returns 0 in BC; if unsuccessful, BC returns the first location at which the burned PROM disagrees with the desired value. Then you can burn a XYBASIC program into PROM by executing the following subroutine, which uses the SCALL command explained in Section 11 above.

```
1000 I% = LAST - FIRST + 1           'LENGTH OF CURRENT PROGRAM
1010 MOVE TO #C000                   'MOVE TO BURNING LOCATION
1020 SCALL #A000, I%                 'EXECUTE BURNING ROUTINE
1030 IF I% <>0 THEN PRINT "FAILURE AT LOCATION"; I%
1040 RETURN
```

Default Initialization Options

Another unique ROMSQuared feature of XYBASIC is its ability to accept specified values for terminal WIDTH and END OF MEMORY and to execute a program from a specified address in memory (in ROM, for example) without prompting during initialization. This feature allows you to execute a XYBASIC program from ROM without any initialization, i.e. to "load and go" automatically on startup.

Just below its base, XYBASIC contains the following five bytes:

```
        ORG     RSQORG
        DB      0         ;default WIDTH @ RSQORG
        DW      1         ;default END OF MEMORY @ RSQORG+1
        DW      0         ;default program address @ RSQORG+3
```

The value of RSQORG depends on which version of XYBASIC you use, as described in Chapter II. Normally its value is 106H for CP/M versions, 3286H for ISIS-II versions, and 163H for Custom I/O versions.

If the byte at RSQORG contains zero, XYBASIC will prompt for WIDTH in the usual way. If not, the specified value is taken as the WIDTH.

XYBASIC takes the word at RSQORG+1 as the value for the top of its RAM working space if the value specified is acceptable. If the value is nonzero but does not leave XYBASIC sufficient RAM, XYBASIC prompts for END OF MEMORY? during initialization in the usual way. If the value is zero, XYBASIC searches for the end of RAM at runtime (as when <carriage return> is typed in response to the initialization prompt).

Finally, if the word at RSQORG+3 is nonzero, XYBASIC assumes it to be the address of a XYBASIC program and attempts to RUN the program at that location. Specifying a value in this word has the same effect as using EXEC to refer to the desired program and then typing RUN; and as with EXEC a RO error will occur if no program is found at the desired location. hr();# XYBASIC Programming Manual

Section 13: Errors

Even experienced programmers make mistakes or ask XYBASIC to do something it cannot; for example, the result of a multiplication might be too large. When XYBASIC finds an error in your program, it prints an error message giving the type and location of the error. Suppose your program contains the line

```
10 LET 3 = J
```

Since the left hand side of a LET command must be a variable, XYBASIC gives you an error message:

```
SN ERROR: 10 LET
                  3 = J

OK
```

Here SN is a code indicating a SyNtax error; the two-letter codes for other errors are given below. The line in which the error occurs is then printed, with a linefeed indicating the approximate location of the error. Using this information you can correct the error and try running the program again.

TRAP and UNTRAP

Sometimes you want your program to continue even after XYBASIC detects an error. For example, you might be running an important experiment and want your program to avoid stopping. This is done by using UNTRAP mode, in which XYBASIC attempts to continue execution after reporting errors to the console. Certain errors are always fatal, i.e. there is no way to recover from them, but for

other errors XYBASIC applies a particular recovery procedure and continues. To get into UNTRAP mode you just type UNTRAP, and to get out you type TRAP. XYBASIC is initially in TRAP mode, and returns to TRAP mode whenever a NEW is executed.

Try the following example.

```
NEW
20 FOR I=1 TO 3
30 MUMBLE
40 NEXT I
RUN

SN ERROR: 30 MUMBLE

OK
```

Note that XYBASIC returns to direct mode when it finds the syntax error. Now say

```
10 UNTRAP
RUN

SN ERROR: 30 MUMBLE

SN ERROR: 30 MUMBLE

SN ERROR: 30 MUMBLE

OK
```

Notice that the bad line is ignored and execution continues in UNTRAP mode.

Error Types

The following list explains the code for each type of error and the recovery procedure XYBASIC uses in UNTRAP mode. Square brackets ([ ]) indicate that the error occurs only in the bracketed version of XYBASIC.

BF Bad File number [CP/M Disk]: File number in OPEN not between 1 and 255, or no file OPEN with given file number.
Recovery: none.

BS Bad Subscript: subscript negative or greater than dimension of array, or subscript formula too complex.
Recovery: if negative, uses smallest legal subscript (0). If too large, uses largest legal subscript.

BY BYte: value negative or greater than 255 where 8-bit quantity is required.
Recovery: most significant byte is ignored.

CN CoNtinue: XYBASIC cannot continue, usually because program has been changed since execution was interrupted.
Recovery: none.

CS CheckSum [Custom I/O]: unsuccessful attempt to LOAD.
Recovery: none.

DD Doubly Defined: array DIMensioned or function DEFined more than once.
Recovery: none.

DF Disk Full [CP/M Disk]: No space remaining on disk.
Recovery: none.

DK DisK [CP/M, ISIS-II]: unsuccessful disk operation.
Recovery: none.

EF End of File [CP/M Disk]: attempt to read past end of file.
Recovery: none.

EN ENable: too many interrupts ENABLEd or a nonexistant line DISABLEd.
Recovery: command ignored.

EX EXception [Editing]: attempt to edit a line longer than 72 characters.
Recovery: none.

FC Function Call: argument not in domain of function, function DEFined in terms of itself, or illegal argument to function.
Recovery: argument truncated to domain.

FI File Input [CP/M Disk]: bad INPUT item.
Recovery: none.

FM File Mode [CP/M Disk]: attempt to read from a file OPEN for Update, or attempt to write to a file OPEN for Output.
Recovery: none.

FN File Not found [CP/M Disk]: attempt to OPEN a nonexistant file for Input or Update.
Recovery: none.

FO File Open [CP/M Disk]: attempt to OPEN file with already OPENed filename or file number.
Recovery: none.

FR FoR: FOR without corresponding NEXT.
Recovery: none.

ID Illegal Direct: command not legal in direct mode.
Recovery: none.

II Illegal Indirect: command legal only in direct mode.
Recovery: none.

LS Long String [Extended] : Result of concatenation longer than 255 characters.
Recovery: result truncated to 255 characters.

MC Machine Call: too many parameters to machine language SCALL.
Recovery: excess parameters ignored.

NF Next without For: NEXT executed without corresponding FOR or incorrect variable name specified.
Recovery: NEXT is ignored.

OD Out of Data: READ executed after all DATA items used.
Recovery: XYBASIC does a RESTORE.

OM Out of Memory: Too many variables, program too long, GOSUBs or FORs nested too deeply, or formula too complex.
Recovery: Purges GOSUB and FOR information, if possible.

ON ON: argument of ON is negative, zero or greater than number of specified line numbers.
Recovery: If negative or zero, takes first line number in list. If too large, takes last line number in list.

OP OPen [CP/M Disk]: attempt to OPEN too many files simultaneously.
Recovery: none.

OS Out of String space [Extended]: Insufficient string space remains.
Recovery: none.

OV OVerflow: [Integer] Result of operation less than -32768 or greater than 32767. [Extended] Result of operation is of magnitude greater than 1.7 * 10^38, or attempt to use value of magnitude greater that 32767 where integer quantity is required.
Recovery: result replaced with largest possible value.)

RG Return without Gosub: RETURN executed without corresponding GOSUB.
Recovery: none.

RO ROmsq feature: insufficient RAM to store program, illegal MOVE destination, EXEC location not a program. [Editing]: attempt to edit program outside working space.
Recovery: none.

SN SyNtax: command not in required form.
Recovery: command ignored.

ST STring [Extended] : String expression too complex.
Recovery: none.

TM Type Mismatch [Extended] : Attempt to assign string value to numeric variable, or numeric value to string variable.
Recovery: none.

UF Unimplemented Feature [Compiler, Runtime Module]: command not implemented in XYBASIC Compiler or Runtime Module.
Recovery: command ignored.

US Undefined Statement: specified line number nonexistant.
Recovery: none.

Section 14: Editing Commands

XYBASIC is available with a line oriented editor, allowing you to modify XYBASIC programs without retyping entire lines. The additional editing facilities consist of the commands AUTO, DELETE, EDIT, and RENUM. Your copy of XYBASIC includes the editing commands described here only if the word EDIT appears in the initialization dialog version message.

AUTO

The AUTO command allows you to enter successive new lines of a program without typing line numbers. AUTO takes two optional arguments, a starting line number and an increment. For example, the command

```
AUTO 100, 20
```

tells XYBASIC to accept a sequence of lines starting at line 100 with an increment of 20. XYBASIC responds by prompting

```
100
```

followed by a space, and then waits for you to enter a line. If you just type <carriage return>, XYBASIC returns to direct mode, gives the usual OK prompt and waits for another command. If you type a line, XYBASIC adds it to the current program as line 100 and then types

```
120
```

followed by a space, and waits for you to enter another line. You may exit from AUTO mode by typing <control-C> at any time, as well as by typing <carriage return> at the start of a line.

If AUTO encounters a line number which is already in the current program, it prompts with an asterisk instead of a space after the line number:

```
100*
```

If a <carriage return> is typed, the old line 100 is retained and XYBASIC returns to direct mode. If a new line is entered, the old line 100 is replaced by the typed line.

If the second argument of AUTO is not specified, it is assumed to be 10. If neither argument is specified, both arguments are assumed to be 10. Thus the commands

```
AUTO
AUTO 10
AUTO 10, 10
```

are all equivalent.

AUTO is legal only in direct mode. An II (Illegal Indirect) error will occur if XYBASIC attempts to execute an AUTO command in program mode. An SN (SyNtax) error occurs if a line number is included at the beginning of any line typed in AUTO mode. The AUTO command is legal only if XYBASIC is currently addressing its working space; an RO (ROmsq feature) error will occur if an AUTO command is attempted while XYBASIC is addressing a program outside its working space.

DELETE

The DELETE command deletes sections of a XYBASIC program. It takes two arguments, a starting and an ending line number. For example, the command

```
DELETE 110, 150
```

tells XYBASIC to delete all lines of the current program from line 110 to line 150, inclusive. If the second line number is omitted, XYBASIC deletes only the specified line:

```
DELETE 130
```

tells XYBASIC to delete only line 130. This has the same effect as typing

```
130<carriage return>
```

except that DELETE 130 will give a US (Undefined Statement) error if line number 130 is not found in the current program.

If the specified line numbers are not found, DELETE will delete all lines following the first line number and preceding the second.

DELETE is legal only in direct mode. An II (Illegal Indirect) error will occur if XYBASIC attempts to execute a DELETE command in program mode. The DELETE command is legal only if XYBASIC is currently addressing its working space; an RO (ROmsq feature) error will occur if a DELETE command is attempted while XYBASIC is addressing a program outside its working space.

EDIT

The EDIT command allows a line of the current program to be changed without retyping the entire line. It takes a single argument giving the line number of the line which you wish to edit. For example,

```
EDIT 120
```

tells XYBASIC to edit line 120. XYBASIC responds by printing line 120, followed by a <carriage return> and <linefeed>. XYBASIC then waits for you to type editing commands to change the contents of line 120. The editing commands usually consist of a <control character> or a <control character> followed by a <printable character>, as detailed below. An imaginary cursor is initially located to the left of the line being edited, and moves when editing commands are performed. The <control character>s typed as editing commands are not echoed, so characters to the left of the cursor may be read by examining the current line.

If the line number is omitted from the EDIT command, XYBASIC will edit the line most recently added to the program or the line in which the most recent error occurred. Thus errors may be corrected by simply typing

```
EDIT
```

and then editing the bad command line.

The editing facilities of XYBASIC are available without typing the EDIT command through the use of <control-E>, as explained below. This form of line editing is particularly useful for correcting errors in typing INPUT data and direct mode commands.

A US (Undefined Statement) error will occur if the line number given in the EDIT command is not found in the current program. EDIT is legal only in direct mode; an II (Illegal Indirect) error will occur if XYBASIC attempts to execute an EDIT command in program mode. An EX (EXception) error will occur if a line containing more than 80 characters is EDITed; this can occur through the use of the abbreviation "?" for PRINT, for example. The EDIT command is legal only if XYBASIC is currently addressing its working space; an RO (ROmsq feature) error will occur if an EDIT command is attempted while XYBASIC is addressing a program outside its working space.

The available editing commands are:

<carriage return>
Terminates the editing process. The characters to the right of the cursor are typed, the line is added to the current program, and XYBASIC returns to direct mode with the usual OK prompt.

<printable character>
Any <printable character> typed is echoed and inserted in the line being edited at the current position of the cursor, with the cursor positioned to the right of the inserted character. If the line will hold no more characters, a <control-G> (bell or beep) is echoed instead of the character.

<rubout>
Deletes the character to the left of the cursor, echoing the deleted character within slashes (/ and \).

<control-B>
[Boot] As always, <control-B> exits from XYBASIC and returns to the operating system or monitor.

<control-C>
Exits from editing mode and returns to direct mode, leaving the previous contents of the line being edited unchanged.

<control-D>
[Delete] Deletes the character to the immediate right of the cursor. The deleted character is not echoed.

<control-E>
[Edit] Allows the use of editing facilities any time you enter a line to XYBASIC. If <control-E> is typed as the first character of a line, XYBASIC enters the editor with the contents of the most recently typed line; this facility is particularly useful for correcting errors in direct mode commands and INPUT data. If <control-E> is typed after the first character, XYBASIC enters the editor with the characters preceding the <control-E>.

<control-F> <printable character>
[Find] Moves the cursor to the right of the next occurrence of the specified <printable character> in the line being edited, printing all characters which the cursor passes. If the remainder of the line contains no occurrence of the <printable character>, XYBASIC echoes <control-G> (bell or beep) and leaves the cursor position unchanged. The search character is not echoed.

<control-G>
[Bell] Inserts a <control-G> (bell or beep) in the line at the current cursor position. As elsewhere in XYBASIC, <control-G> is treated as a <printable character>.

<control-H>
[Backspace] Erases the character to the left of the cursor and echoes <control-H>. For terminals which recognize <control-H> as a backspace, such as most CRTs, characters should be deleted with <control-H> rather than <rubout>.

<control-K>
[Kill] Kills all characters to the right of the cursor.

<control-L>
[Left] Types the characters remaining to the right of the cursor on the line being edited, followed by <carriage return> and <linefeed>, and leaves the cursor to the left of all characters on the line.

<contro1-N>
[Next] Finds the next occurrence of the <printable character> last specified in a <control-F> search command, as described above. Echoes <control-G> and leaves the cursor position unchanged if no <control-F> command has been given or if the line contains no more occurrences of the <printable character>.

<control-R>
[Retype] Types the characters to the right of the cursor on the line being edited, followed by <carriage return> and <linefeed>, and then retypes the characters to the left of the cursor. The cursor position is unchanged.

<control-T>
Type] Moves the cursor position one character to the right, echoing the character passed by the cursor.

<control-U>
[Undo] Discards the current contents of the line being edited and begins the editing process anew with the original contents of the line, allowing easy recovery from editing mistakes.

RENUM

The RENUM command automatically renumbers the current XYBASIC program. It takes up to three arguments. The first argument gives the current line number of the first line to be renumbered. The second argument gives the desired increment. The third argument gives the desired line number for the first renumbered line. For example,

```
RENUM 10, 100, 1000
```

tells XYBASIC to renumber the current program starting at line 10, with line 10 renumbered as line 1000 and successive lines numbered 1100, 1200, and so on. If all three arguments are omitted, XYBASIC renumbers by leaving the line number of the first program line unchanged and incrementing successive line numbers by 10. If the second and third arguments are omitted, XYBASIC leaves the line number of the given line unchanged and increments successive line numbers by 10. If the third argument is omitted, it is assumed to be the same as the first. For example,

```
RENUM                       'SAME AS RENUM first, 10, first
RENUM 20                    'SAME AS RENUM 20, 10, 20
RENUM 100, 20               'SAME AS RENUM 100, 20, 100
```

A US (Undefined Statement) error will occur if the specified first line number does not exist in the current program. A US error also will occur if renumbering the program with the specified arguments would result in a line number greater that 65535, or if the specified renumbering would change the order of lines in the program. In any of these cases no renumbering takes place.

A US (Undefined Statement) error will occur if any command in the current XYBASIC program refers to a nonexistent line number. For example, attempting to RENUM 10, 20, 100 with the current program

```
10 GOTO 5
```

would give a US error, since the program contains no line number 5. In this case the line renumbering does occur, but references to nonexistent lines remain unchanged and XYBASIC lists the lines containing nonexistent line number references before issuing the US error:

```
100 GOTO 5
US ERROR: RENUM 10, 20, 100
```

RENUM is legal only in direct mode. An II (Illegal Indirect) error will occur if XYBASIC attempts to execute a RENUM command in program mode. The RENUM command is legal only if XYBASIC is currently addressing its working space; an RO (ROmsq feature) error will occur if a RENUM command is attempted while XYBASIC is addressing a program outside its working space.

Section 15: CP/M Sequential Disk Commands

The CP/M version of Extended XYBASIC is available with sequential disk operations, allowing you to store and manipulate information on disk files under XYBASIC. This section describes the facilities available in this version. Your copy of XYBASIC includes the commands described in this section only if the word DISK appears in the initialization dialog version message.

The additional features available in this version are the commands OPEN, CLOSE, LINPUT, MARGIN, DIR, and SCRATCH, and the function EOF. Also, additional forms of the commands PRINT, INPUT and CLEAR are allowed. The operation of all other XYBASIC commands and functions in this version is unchanged, with the exception of different error messages when disk errors occur.

Filenames

The name of a CP/M file in CP/M Sequential Disk XYBASIC consists of an optional disk name, a filename, and an optional filetype. It may be specified by any string, either a quoted string or a string formula. The filename must be a string of one to eight letters or digits. The diskname may be "A:", "B:", "C:", "D:" or "@:" (indicating the currently logged disk). The filetype consists of "." followed by from zero to three letters or digits. Lower case alphabetic characters in the filename and filetype are converted to upper case automatically.

The following are examples of legal file names.

```
"EXAMPLE"
"SMITH.DAT"
"a:temp.fil"    (lower case converted to UPPER)
S$              (where S$ has the value "@:PROG.XYB")
S$+".bas"       (where S$ has the value "b:prog2")
```

The SAVE and LOAD commands described in Section 6 use filenames of the same format, but the filetype .XYB or .BAS is assumed automatically. Therefore to SAVE a program in ASCII as B:EXAMPLE.BAS, you just type

```
SAVE "B:EXAMPLE",A
```

Notice that the disk name B: is inside the quote marks rather than outside. A SN (SyNtax) error will occur if a file name is specified incorrectly in a command.

OPEN

The OPEN commmand tells XYBASIC the name of a file you wish to use and whether you want to read from the file or write to it. It also associates an integer file number (between 1 and 255) with the file for use in subsequent commands. For example,

```
OPEN I, @1, "OLD.DAT"
```

indicates that you want to read (Input) information from the file "OLD.DAT", referring to it as file number @1. Similarly,

```
OPEN O, @2, "NEW.DAT"
```

indicates that you want to write (Output) information to the file "NEW.DAT", referring to it as file number @2. Finally,

```
OPEN U, @3, "UPDATE.DAT"
```

indicates that you want to Update the file "UPDATE.DAT" by appending additional information to its previous contents.

The file number in an OPEN command may be given by any integer formula, but a BF (Bad File number) error will occur if its value is not between 1 and 255. An FN (File Not found) error will occur if you try to OPEN a nonexistant file for Input or Update. An FO (File Open) error will occur if the filename or number is already associated with an OPEN file.

The number of files you can have OPEN simulaneously is limited to two when you first load XYBASIC, and an OP (OPen) error occurs if you try to OPEN too many files. You can use the CLEAR command as described below to indicate that you need more or less than two OPEN files simultaneously.

CLOSE

The CLOSE command tells XYBASIC that your operations with a given disk file or files are completed. If you say

```
CLOSE @1
```

then file @1 is closed; a BF (Bad File number) error occurs if no OPEN file exists. Similarly,

```
CLOSE
```

will close all OPEN files. XYBASIC performs a CLOSE automatically whenever it executes a RUN, END or NEW command. You must CLOSE files before you exit from XYBASIC, or the information in them may be lost! The simplest way to do so is to always write an END command at the end of disk XYBASIC programs.

PRINT

The PRINT command lets you send information to files open for Output or Update in CP/M Sequential Disk XYBASIC. If you type the command

```
PRINT @1, I
```

then XYBASIC will append the characters giving the value of I and a <carriage return> and <linefeed> to the file @1. Similarly,

```
PRINT @1, "J = "; J;
```

will add "**J** = " and the value of **J** to @1. Since this command ends in a <semicolon>, no <carriage return> and <linefeed> are sent to @1.

The file number may be given by any integer formula. A BF (Bad File number) error will occur if the value of the formula is not 0 or the number of an OPEN disk file. If the value is 0, the desired information is just PRINTed on the console, so the command

```
PRINT @0, A; B; C
```

has the same effect as

```
PRINT A; B; C
```

A DF (Disk Full) error will occur if the disk is full. You can use the SCRATCH command described below to delete unwanted files after a DP error occurs, leaving additional space for data.

An FM (File Mode) error will occur if you try to PRINT to a file OPEN for Input.

MARGIN

The MARGIN command lets you format your output files by specifying a maximum width for each output line. For example, if you say

```
MARGIN @1, 50
```

then all lines PRINTed to file @1 will have a maximum width of 50 characters. The width may be specified by any integer formula with a value less than 256. If you do not set the file width with MARGIN, XYBASIC assumes a default width of

72 characters.

You can also use MARGIN to change the width of output lines on the console device. For example,

MARGIN @0, 50

changes the width of the console from the value specified during initialization to 50 characters.

A BF (Bad File number) error occurs if the file number specified is not 0 or the number of an OPEN file, and a BY (BYte) error occurs if the specified width is greater than 255.

INPUT

In CP/M Sequential Disk XYBASIC you can use INPUT to read information from disk files as well as to get information from the console. For example,

```
10 INPUT @2, I, J, K
```

gets new values for the variables I, J and K from the file @2. When XYBASIC excutes an INPUT from a file, it does not prompt and does not give REDO or EXCESS IGNORED messages.

The file number may be given by any integer formula. A BF (Bad File number) error will occur if its value is not 0 or the number of a file OPEN for Input. If the file number is 0, XYBASIC just does a normal INPUT, prompting with ? and waiting for information from the console.

An FM (File Mode) error will occur if you try to INPUT from a file OPEN for Output or Update, and an FI (File Input) error will occur if a bad item is read during INPUT. INPUT is legal only in program mode, and an ID (Illegal Direct) error occurs if you try to use it in direct mode.

A EF (End of File) error will occur if you use INPUT to read past the end of a file. You can avoid such errors by using the EOF function described below to check if all the information in the file has been read.

LINPUT

The LINPUT (Line INPUT) command reads a line of characters from the console or from a disk file and assigns a string variable the value given by the string of characters in the line. To use LINPUT to get a line from the console, you just say

```
10 LINPUT S$
```

When XYBASIC executes line 10, it just waits for you to type a line at the console; unlike INPUT, LINPUT does NOT print a ? to prompt you before waiting for the typed line. When you terminate the line you type with a <carriage return>, XYBASIC constructs a string consisting of the characters typed (not including the <carriage return>), and this string becomes the new value of S$.

You can also get lines from disk files with LINPUT. The command

10 LINPUT @2, S$

will get characters from the file @2 until either a <carriage return> is found or 255 characters have been read without encountering a <carriage return>. Then the string consisting of the characters read (not including the <carriage return>) becomes the new value of S$. A BF (Bad File number) error will occur if the specified file number is not 0 or the number of a file OPEN for Input. If the file number is 0, XYBASIC just does a LINPUT from the console.

A EF (End of File) error will occur if you use LINPUT to read past the end of a file. You can avoid such errors by always writing a <carriage return> and <linefeed> at the end of a file (for example with the command PRINT @2) and then using the EOF function described below to check if all the information in the file has been read.

An FM (File Mode) error will occur if you try to LINPUT from a file OPEN for Output or Update. Like INPUT, LINPUT is legal only in program mode, and an ID (Illegal Direct) error will occur if you try to use it in direct mode.

EOF

EOF is a function which tells you whether a file OPEN for Input contains any more characters. Its value is true (-1) if there are no more characters, and false (0) if there are more. You can use EOF to avoid EF (End of File) errors which would otherwise occur when a program tried to INPUT or LINPUT information past the end of a file.

For example, consider the following program to read in a file and list it on the console.

```
10 INPUT "Filename" FILE$
20 PRINT "Listing of file "; FILE$
30 PRINT
40 OPEN I, @1, FILE$
50 LINPUT @1, S$
60 PRINT S$
70 GOTO 50
```

When you RUN this program, it prompts for a filename and then prints each line of the program on the console. After it reads the last line, it tries to LINPUT another line and an EF error occurs. To avoid the error you can replace line 70 with

```
70 IF NOT EOF (1) THEN 50
80 END
```

Line 70 uses EOF to test whether more lines exist in the file before returning to line 50 to LINPUT the next line.

A BF (Bad File number) error occurs if the argument of EOF is not the number of an OPEN file. An FM (File Mode) error occurs if the argument is the number of a file OPEN for Output or Update.

DIR

The DIR command lets you print file directories on the console. If you say

```
DIR
```

then XYBASIC will print the name of all files on the currently logged disk. Similarly,

```
DIR "B:*.XYB"
```

will print the names of all files on disk B: with filetype .XYB (XYBASIC programs in internal format).

SCRATCH

The SCRATCH command lets you erase files from a disk. For example,

SCRATCH "TEMP.DAT"

will delete the file TEMP.DAT from the currently logged disk.

CLEAR

As noted above, CP/M Sequential Disk XYBASIC initially assumes that you will need at most two files OPEN simultaneously. If you need more (or less), you can use a modified form of the CLEAR command to tell XYBASIC how many you need. If you say

```
CLEAR @3
```

then XYBASIC will allow you to have up to three files OPEN simultaneously. Each potentially open file requires 166 bytes of RAM, so an OM (Out of Memory) error may occur if the number of files you try to allocate is too large. The amount of string space allocated is left unchanged by this form of CLEAR, but the values of your variables are all CLEARed.

Chapter II: VERSION DIFFERENCES

The XYBASIC interpreter is available in a number of different versions, depending on the user's computer and operating system. This chapter outlines specific differences between versions. It also describes two related programs, the XYBASIC Compiler and XYBASIC Runtime Module, which allow debugged programs to be executed with less overhead than under the interpreter.

Section 1: CP/M Version

The CP/M version of XYBASIC is supplied as a COM file XYBASIC.COM residing in RAM starting at 100H, so to load XYBASIC under CP/M you just type XYBASIC. CP/M copies of XYBASIC are normally delivered on a single density eight inch floppy disk, but are also available on a single density five inch floppy disk (e.g. for NorthStar).

In CP/M versions the IOBYTE resides at location 3, and the default value for the top of available RAM is determined from locations 6 and 7. Location 103H contains JMP GTPAR, and after initialization the JMP instruction at location 100H may be used to return to direct mode from user routines (e.g. after an error condition in an assembly language routine). The ROMSQuared default value bytes are at locations 106H to 10AH.

By including a filename on the CP/M command line, you can load XYBASIC together with a XYBASIC program and execute the program immediately. For example,

```
XYBASIC B:EXAMPLE
```

first loads XYBASIC. Then the WIDTH is defaulted to 80 and the END OF MEMORY is defaulted to the maximum value allowed under your CP/M system, exactly as if you typed <carriage return> in response to the initialization prompts. Finally the specified XYBASIC program (in this case B:EXAMPLE.XYB) is LOADed and RUN. An FN (File Not found) error will occur if no .XYB file with the given filename is found.

The load and go option can be especially useful within SUBMIT files, where any number of programs can be executed (with non-XYBASIC programs interspersed) in sequence. You can use the command

```
CALL 0
```

within XYBASIC programs to exit from the program and return to CP/M automatically, without typing <control-B>.

Section 2: ISIS-II Version

The ISIS-II version of XYBASIC is supplied as an absolute file XYBAS starting in RAM at 3280H, so to load XYBASIC under ISIS-II you just type XYBAS. ISIS-II copies of XYBASIC are normally delivered on either a single density or double density eight inch floppy disk.

The IOBYTE resides at location 3, and the default value for the top of available RAM is determined from the monitor routine MEMCHK. Location 3283H contains JMP GTPAR, and after initialization the JMP instruction at location 3280H may be used to return to direct mode from user routines. The ROMSQuared default value bytes are at locations 3286H to 328AH.v

Section 3: Custom I/O Version

The Custom I/O version of XYBASIC allows XYBASIC to run on any 8080-based microprocessor, with or without an operating system, from either RAM or ROM. The Custom I/O version normally begins at 100H and assumes that RAM begins at 2000H (8K) for Integer XYBASIC and 4000H (16K) for Extended XYBASIC, but it may be specially ordered to begin at any specified location and to assume RAM beginning at any specified location. The delivery medium for Custom I/O versions may be paper tape (Intel HEX format), eight inch floppy disk (CP/M or ISIS-II compatible, single or double density), or programmed 2708 or 2716 EPROMs.

Before using the Custom I/O version of XYBASIC, the addresses of routines to perform elementary I/O operations must be patched into the JMP vector beginning at 118E, as explained below.

The IOBYTE is stored in the first byte of RAM (normally 4000H) for Custom I/O versions, and is set to 0 during initialization. Location 100H contains a JMP to the initialization routine, and may be used to restart XYBASIC (repeating the initialization dialog). Location 103H contains JMP GTPAR. Locations 106H through 16211 contain various JMP instructions, as described below. The ROMSQuared default value bytes are at locations 163H to 167H. After initialization, user routines may return to direct mode by branching to location 168H.

To perform input / output operations in the CP/M and ISIS-II versions, XYBASIC just makes calls through the operating system. In the Custom I/O version XYBASIC instead calls one of the user routines located below the base of XYBASIC, in a JMP vector at locations 118H through 15FH. As in the CP/M and ISIS-II versions, the user can use four logical devices: CONsole (input and output), ReaDeR (input only), PUNch (output only) and LiST (output only). The routines involved are:

CI (Console In)	Returns a character in the A register from the active CON device.
CO (Console Out)	Sends a character from the C register to the active CON device.
RI (Reader In)	Returns an 8-bit character in the A register from the active RDR device and returns Carry reset (0); when end of file occurs the Carry is set (1) and the A register returns 0.
PO (Punch Out)	Sends an 8-bit character from the C register to the active PUN device.
LO (List Out)	Sends a character from the C register to the active LST device.
CS (Console Status)	Returns 255 (OFFH) in the A register if a character has been typed on the active CON device, 0 if not.

The Custom I/O version of XYBASIC uses RI to LOAD programs, PO to SAVE programs, and the CONsole for all other I/O operations. All output is echoed to LO if <control-P> is typed. Because programs are SAVEd and LOADed in XYBASIC's internal format, SAVE and LOAD will not operate correctly unless the PUN and RDR routines pass full 8-bit bytes, without manipulation of the parity bit.

Each I/O routine first checks the IOBYTE and then branches to the desired user-implemented device through the JMP vector at 118H to 15FH. The user must patch a JMP to a device driver into the appropriate location in the vector for each device he implements. The driver routines should always leave unchanged all registers not specifically used by the routine. Notice that each user CONsole device must have Console Status implemented (as well as Console In and Console Out) for XYBASIC to operate correctly. If Console Status is not implemented correctly, you will be unable to interrupt program execution by typing <control-C>!

Section 6 of Chapter I above explains the use of the ASSIGN command and IOBYTE function to change and interrogate the system IOBYTE, determining which physical device implements a logical device.

It is possible to run a Custom I/O version of XYBASIC under CP/M or ISIS-II, by patching the jump vector accordingly. However, doing so makes it difficult to SAVE and LOAD programs as disk files.

Device Driver Locations

The locations of each I/O routine or driver are given in the code below. The JMP vector starting at 106H allows the user access to XYBASIC's I/O system (with devices selected by the IOBYTE), and should NOT be patched by the user. The JMP at location 160H is executed when you type <control-B>, and should be patched to specify the entry point to your monitor or operating system.

```
        ORG     100H
        JMP     XYBASIC  ;start Of XYBASIC
        JMP     GTPAR    ;GTPAR (to get CALL parameters) @ 103H

;JMP VECTOR FOR I/O SYSTEM (do not patch)
        JMP     CI       ;console in @ 106H
        JMP     CO       ;console out @ 109H
        JMP     RI       ;reader in @ 10CH
        JMP     PO       ;punch out @ 10FH
        JMP     LO       ;list out @ 112H
        JMP     CS       ;console status @ 115H

;JMP VECTOR FOR USER-DEFINED DEVICES (to be patched)
        JMP     UC0I     ;user console 0 in @ 118H
        JMP     UC1I     ;user console 1 in @ 11BH
        JMP     UC2I     ;user console 2 in @ 11EH
        JMP     UC3I     ;user console 3 in @ 121H
        JMP     UC0O     ;user console 0 out @ 124H
        JMP     UC1O     ;user console 1 out @ 127H
        JMP     UC2O     ;user console 2 out @ 12AH
        JMP     UC3O     ;user console 3 out @ 12DH
        JMP     UR0I     ;user reader 0 in @ 130H
        JMP     UR1I     ;user reader 1 in @ 133H
        JMP     UR2I     ;user reader 2 in @ 136H
        JMP     UR3I     ;user reader 3 in @ 139H
        JMP     UP0O     ;user punch 0 out @ 13CH
        JMP     UP1O     ;user punch 1 out @ 13FH
        JMP     UP2O     ;user punch 2 out @ 142H
        JMP     UP3O     ;user punch 3 out @ 145H
        JMP     UL0O     ;user list 0 out @ 148H
        JMP     UL1O     ;user list 1 out @ 14AH
        JMP     UL2O     ;user list 2 out @ 14DH
        JMP     UL3O     ;user list 3 out @ 151H
        JMP     UC0S     ;user console 0 status @ 154H
        JMP     UC1S     ;user console 1 status @ 157H
        JMP     UC2S     ;user console 2 status @ 15AH
        JMP     UC3S     ;user console 3 status @ 15DH

;MONITOR ENTRY POINT (to be patched)
```

```
        JMP     XYBASIC  ;exit (when <control-B> typed) @ 160H

;ROMSQuared DEFAULT VALUE BYTES (to be patched if desired)
        DB      0        ;default WIDTH @ 163H
        DW      1        ;default END OF MEMORY @ 164H
        DW      0        ;default program address @ 166H
```

Sample I/O Patch

This section gives a sample i/o patch for a Custom I/O version of XYBASIC. You can use the routines given here as a model for construction of an i/o patch for your computer system.

This patch assumes that XYBASIC has been specially ordered to begin at location 0A000H. The computer is assumed to support a single console device, with port use as defined below. When a restart (RST 0) is performed, the computer executes the code starting at location 40H, which resets the USART used for console I/O and begins execution of XYBASIC. All other RST instructions are ignored. Typing <control-B> also restarts XYBASIC.

Since the computer supports a single console device, the XYBASIC i/o JMP table contains the same JMP instruction four times for each driver routine. To simplify this patch the ReaDeR and PUNch devices are defined to be the same as the CONsole, and characters sent to the LiST device are ignored.

```
                ;XYBASIC ENTRY POINT
A000 =          XYBASIC EQU     0A000H          ;START OF XYBASIC

                ;USART PORT NUMBERS
00EC =          CIN     EQU     0ECH            ;CONSOLE DATA INPUT PORT
00EC =          COUT    EQU     0ECH            ;CONSOLE DATA OUTPUT PORT
00ED =          CCTL    EQU     0EDH            ;CONSOLE CONTROL PORT

                ;I/O BIT NUMBERS
0001 =          TBA     EQU     01H             ;TRANSMIT BUFFER AVAILABLE
0002 =          RBR     EQU     02H             ;RECEIVE BUFFER READY

                ;I/O DEVICE COMMANDS
00CF =          MODE    EQU     0CFH            ;SET USART MODE
0025 =          CMD     EQU     025H            ;SELECT NORMAL USART

                ;CODE GENERATION FOR RST INSTRUCTIONS
0000                    ORG     0H              ;RST 0
0000 C34000             JMP     INIT            ;INITIALIZE ON RESTART
0008                    ORG     8H
0008 C9                 RET
0010                    ORG     10H
0010 C9                 RET
```

```
0018                    ORG     18H
0018 C9                 RET
0020                    ORG     20H
0020 C9                 RET
0028                    ORG     28H
0028 C9                 RET
0030                    ORG     30H
0030 C9                 RET
0038                    ORG     38H
0038 C9                 RET

                ;DEVICE INITIALIZATION
0040                    ORG     40H
0040 3ECF       INIT    MVI     A,MODE
0042 D3ED               OUT     CCTL            ;SET USART MODE
0044 3E25               MVI     A,CMD
0046 D3ED               OUT     CCTL            ;SET USART COMMAND MODE
0048 C9                 RET                     ;BEGIN XYBASIC

                ;I/O DEVICE DRIVERS

                ;CONSOLE STATUS
004B DBED       CSTAT:  IN      CCTL            ;READ STATUS
004D E602               ANI     RBR             ;MASK TO CHARACTER READY
004F C8                 RZ                      ;NOT READY, RETURN 0 IN A
0050 3EFF               MVI     A,0FFH          ;READY, RETURN 0FFH IN A
0052 C300A0             RET

                ;CONSOLE IN
0053 DBED       CONIN:  IN      CCTL            ;READ STATUS
0055 E602               ANI     RBR             ;MASK TO CHARACTER READY
0057 CA5300             JZ      CONIN           ;WAIT IF NOT READY
005A DBEC               IN      CIN             ;READ THE CHARACTER
005C C9                 RET

                ;CONSOLE OUT
005D DBED       CONOUT: IN      CCTL            ;READ STATUS
005F E601               ANI     TBA             ;MASK TO BUFFER AVAILABLE
0061 CA5D00             JZ      CONOUT          ;WAIT IF NOT AVAILABLE
0064 79                 MOV     A,C             ;CHARACTER TO A
0065 D3EC               OUT     COUT            ;WRITE THE CHARACTER
0067 C9                 RET

                ;PUNCH OUT, DEFINED TO SEND CHARACTERS TO CONSOLE
005D =          PUNOUT  EQU     CONOUT

                ;READER IN, DEFINED TO READ CHARACTERS FROM CONSOLE
0053 =          RDRIN   EQU     CONIN
```

```
                ;LIST OUT, DEFINED TO DO NOTHING
0068 C9         LOUT:   RET

                ;XYBASIC DEVICE DRIVER JMP TABLE OVERLAY
A018                    ORG     XYBASIC+18H
                ;CONSOLE IN
A018 C35300             JMP     CONIN
A01B C35300             JMP     CONIN
A01E C35300             JMP     CONIN
A021 C35300             JMP     CONIN
                ;CONSOLE OUT
A024 C35D00             JMP     CONOUT
A027 C35D00             JMP     CONOUT
A02A C35D00             JMP     CONOUT
A02D C35D00             JMP     CONOUT
                ;READER IN
A030 C35300             JMP     RDRIN
A033 C35300             JMP     RDRIN
A036 C35300             JMP     RDRIN
A039 C35300             JMP     RDRIN
                ;PUNCH OUT
A03C C35D00             JMP     PUNOUT
A03F C35D00             JMP     PUNOUT
A042 C35D00             JMP     PUNOUT
A045 C35D00             JMP     PUNOUT
                ;LIST OUT
A048 C36800             JMP     LOUT
A04B C36800             JMP     LOUT
A04E C36800             JMP     LOUT
A051 C36800             JMP     LOUT
                ;CONSOLE STATUS
A054 C34B00             JMP     CSTAT
A057 C34B00             JMP     CSTAT
A05A C34B00             JMP     CSTAT
A05D C34B00             JMP     CSTAT
                ;MONITOR ENTRY POINT
A060 C34000             JMP     INIT
A063                    END
```

SAVEd Program Format

A program which is stored in n bytes of memory is SAVEd in the Custom I/O version as a header block followed by ((n+1)/255) + 1 data blocks. The header block is 13 bytes long:

Byte 1: Start byte = 3AH
Byte 2: Type byte = 0H
Bytes 3-10: File name, one to eight ASCII characters padded by ASC

Bytes 11-13:File type, ASCII characters XYB = 58H, 59H, 42H

Each data block contains the following bytes:

Byte 1: Start byte = 3AH
Byte 2: Type byte = 0H
Byte 3: Length byte = m, # of data bytes following
Bytes 4-(3+m):Data bytes containing source program
Byte 4+m: Checksum byte = sum of data bytes, mod 256

Additional data blocks follow only if the length byte contains 255.

The following assembly language routine indicates how SAVEd programs can be loaded into memory, and may be used as a template for applications in which the user wishes to manipulate SAVEd programs without using XYBASIC. The desired file name should be stored at FILNAM. The file is read from the RDR device and loaded starting at address DEST.

```
STBYTE  EQU     3AH     ;start byte
TYBYTE  EQU     0       ;type byte

HEADER: DB      STBYTE, TYBYTE
FILNAM: DS      8       ;ASCII filename, padded by spaces
        DB      'XYB'   ;file type
LOAD:   LXI     H,HEADER;header location to HL
        MVI     C,13    ;header length to C
LOAD1:  CALL    RDRIN   ;read a character
        CMP     M       ;compare to header character
        JNZ     LOAD    ;no match, try again
        DCR     C       ;decrement count
        JNZ     LOAD1   ;read next header character
        LXI     H,DEST  ;destination to HL
LOAD2:  CALL    RDRIN   ;read character
        CPI     STBYTE  ;compare to start byte
        JNZ     ERROR
        CALL    RDRIN   ;read another
        CPI     TYBYTE  ;compare to type byte
        JNZ     ERROR
        CALL    RDRIN   ;read length byte
        ORA     A       ;check if length 0
        RZ              ;done if length 0
        MOV     E,A     ;length to E
        INR     A
        PUSH    PSW     ;save length+1
        MVI     D,0     ;checksum to D
LOAD3:  CALL    RDRIN   ;read a source character
        MOV     M,A     ;store it
        INX     H       ;point to next destination
```

```
        ADD     D
        MOV     D,A     ;update checksum
        DCR     E       ;decrement count
        JNZ     LOAD3   ;load more characters from data block
        CALL    RDRIN   ;read the checksum
        CMP     D       ;compare to computed checksum
        JNZ     ERROR   ;checksum error
        POP     PSW     ;recover length+1
        JZ      LOAD2   ;length was 255, load more data blocks
        RET             ;otherwise LOAD is completed

ERROR:  ...             ;LOAD error routine
```

Saving and Loading Under Operating Systems

This section describes a simple way to use XYBASIC's ROMSQuared features to save and load XYBASIC user programs as files under operating systems other than CP/M and ISIS-II. The method used is a CALL from XYBASIC of an assembly language routine which first obtains parameters from XYBASIC and then executes the appropriate system-specific file manipulation routine.

For purposes of illustration we assume that XYBASIC starts at location 100H, that the assembly language saving and loading routines begin at 0A000H and 0A800H, and that memory above 0B000H can be used as a scratch RAM area during program loading.

We also assume that the system-dependent routine DSAVE is passed the following information:

A	number of characters in filename
BC	location of first filename character
DE	location of first byte to save
HL	location of last byte to save

Similarly, we assume that the system-dependent routine DLOAD is passed the following:

A	number of characters in filename
BC	location of first filename character
DE	location of first byte of scratch area

The XYBASIC commands to save a program are:

```
F% = FIRST                  'FIRST LOCATION
L% = LAST                   'LAST LOCATION
F$ = "FILENAME"             'DESIRED FILE
CALL #A000, F%, L%, F$      'SAVE IT
```

and the assembly language save routine at location 0A000H is:

```
GTPAR   EQU     103H            ;entry point of routine GTPAR
        ORG     0A000H
        CALL    GTPAR           ;get first location address
        MOV     E,M
        INX     H
        MOV     D,M
        PUSH    D               ;save first location
        CALL    GTPAR           ;get last location address
        MOV     E,M
        INX     H
        MOV     D,M
        PUSH    D               ;save last location
        CALL    GTPAR           ;get filename address
        MOV     A,M             ;length to A
        INX     H
        MOV     C,M
        INX     H
        MOV     B,M             ;first filename char loc to BC
        POP     H               ;last location to HL
        POP     D               ;first location to DE
DSAVE:  [System-dependent saving routine]
        RET
```

Similarly, the commands to load a XYBASIC user program from the system are:

```
I% = #8000                  'SCRATCH AREA ADDRESS
F$ = "FILENAME"             'DESIRED FILENAME
CALL #A800, I%, F$          'LOAD TO SCRATCH AREA
MOVE FROM I%                'FETCH FROM SCRATCH AREA
```

and the assembly language load routine at 0A800H is:

```
        ORG     0AB00H
        CALL    GTPAR           ;get scratch area address
        MOV     E,M
        INX     H
        MOV     D,M
        PUSH    D               ;save scratch area location
        CALL    GTPAR           ;get filename address
        MOV     A,M             ;length to A
        INX     H
        MOV     C,M
        INX     H
        MOV     B,M             ;location to BC
        POP     D               ;scratch area location to DE
```

```
DLOAD:  [System-dependent loading routine]
        RET
```

Section 4: INTEL SBC Series Versions

For users with INTEL SBC 80/10, 80/20 or 80/30 systems, the Custom I/O version of XYBASIC is available with the device driver jump vector patched accordingly. When ordering an SEC version, the user should specify whether XYBASIC should operate coresident with the monitor or as a stand-alone program.

Users with SBC 80/20 or 80/30 systems may also order a version of XYBASIC which uses a hardware realtime clock. In this version the TIME command described in Section 8 of Chapter I is eliminated, the DELAY command is modified slightly, and the command SETTIME and function TIME$ are added to XYBASIC.

The command SETTIME is used to initialize the real time clock. For example,

```
SETTIME H, M, S
```

initializes the realtime clock to H hours, M minutes and S seconds. The M and S parameters are optional, and are defaulted to zero if not present.

The string function TIME$ returns the current time as a string of eight characters, of the form "hh:mm:ss"; for example,

```
PRINT "TIME IS CURRENTLY "; TIME$
```

results in XYBASIC printing

```
TIME IS CURRENTLY hh:mm:ss
```

If no SETTIME has been executed, TIME$ returns the time since XYBASIC was entered.

The DELAY command takes the form

```
DELAY formula1, formula2, formula3
```

where formula1 specifies the number of minutes, formula2 the number of seconds, and formula3 the number of tenths of seconds to DELAY. The second and third parameters are optional, and are defaulted to zero if not present. For example, the command

```
DELAY 2, 15, 1
```

DELAYS for 2 minutes and 15.1 seconds before executing the next command.

In the hardware realtime SBC 80/20 version of XYBASIC, interval timer zero of the onboard 8253 chip is programmed to interrupt the CPU every 50 milliseconds (mode 2). XYBASIC counts the number of interrupts since the last SETTIME command to compute the current time of day. The accuracy of XYBASIC's time functions is determined by the accuracy of the CPU clock crystal.

XYBASIC's initialization code sets the 8259 interrupt controller for standard 8080 interrupts, locations 0 thru 38H. RST 0 is reserved for entry to XYBASIC and RST 2 is reserved for the realtime clock interrupt. All other RSTs are available for user interrupts.

The SEC 80/20 is shipped from the factory with interval timer zero strapped to generate interrupt request 2 when the timer expires. This wire (from option pin 26 to 35, near the 8259 interrupt controller chip) must be present for the realtime functions to work correctly. It is normally present because the interval timer implements the single step function of the Intel monitor. It is available for use by XYBASIC because the monitor is never resident at the same time as XYBASIC.

Section 5: AMD 9511 Floating Point Version

Since the 8080 series of microprocessors does not support hardware floating point operations, XYBASIC normally performs floating point manipulation by software. However, XYBASIC is also available by special order in versions which perform floating point operations with the AMD 9511 Arithmetic Processing Unit.

The 9511 version of XYBASIC resides in less memory space than the software floating point version. Floating point arithmetic operations are somewhat faster than in the software floating point version, and trigonometric functions (SIN, COS, TAN, ATN) are much faster.

The 9511 represents a floating point value in four bytes. A floating point 0 is represented by four 0 bytes. For any other value, byte 1 contains the mantissa sign in bit 7 and the two's complement binary exponent in bits 6-0. Bytes 2-4 contain the normalized binary mantissa, with an assumed binary point to the left of byte 2. Because of the representation used by the 9511, the range of floating point values is roughly -9 * 10^18 to 9 * 10^18, as opposed to roughly -1.7 * 10^38 to 1.7 * 10^38 for the software floating point version.

To order a 9511 version of XYBASIC, you must specify what ports your computer uses to communicate with its 9511, in addition to the usual information about which operating system you use.

Section 6: XYBASIC Compiler

The XYBASIC Compiler is a program which takes a SAVEd XYBASIC program as input and produces as output an INTEL HEX format object file containing a runtime package plus the program; this HEX file may then be loaded and executed. The resulting program will run in ROM, with the user specifying RAM and ROM locations for the desired memory configuration. The compiler is available in CP/M and ISIS-II versions.

Compiler Operation

To use the CP/M version of the compiler, you type either COMPILE or COMPILE [filename]. To use the ISIS-II version, you type COMPIL or COMPIL [filnam]. In either case the compiler will respond with

```
XYBASIC COMPILER {version} REV n.m
```

where {version} is CP/M or ISIS-II and REV n.m indicates the revision of the compiler being used. If you specified a filename, the program [filename].XYB is loaded; if not, the compiler prompts:

```
SOURCE FILE?
```

and waits for you to type a filename (up to eight characters long for CP/M, up to six for ISIS-II) , optionally preceded by a disk name. The prompt is repeated if the desired source file is not found. In responding to any compiler prompt the user of the CP/M version can type either <rubout> or <control-U> to try again after a typing error, or <control-C> to abort execution and return to CP/M. The user of the ISIS-II version can use the usual line editing features of ISIS-II. Each response must be terminated by a <carriage return>.

Next you specify the memory configuration. The compiler prompts:

```
START OF ROM (HEX)?
```

You should respond with one to four hexadecimal digits specifying the first ROM address to be used for the object program. If you just type a <carriage return>, a default value of 100H for CP/M or 3280H for ISIS-II is assumed. The compiler prompts with:

```
START OF RAM (HEX)?
```

and you respond with another hexadecimal address. If a <carriage return> is typed, the first address following ROM used by the object program is assumed. Next the compiler prompts:

```
END OF RAM (HEX)?
```

and you type a third hexadecimal address. If you type <carriage return>, the object program will find the highest available RAM address at runtime. Finally the compiler prompts:

```
WIDTH (DECIMAL)?
```

and you type a decimal number less than 256 to specify the desired output device column width. A default value of 72 is assumed if <carriage return> is typed.

Next the compiler prints a map of memory usage, of the form

```
ROM USE: aaaaH TO bbbbH
PROGRAM: ccccH TO bbbbH
RAM USE: eeeeH TO ffffH
RAM  BYTES  FREE: ggggH
```

The first two lines indicate that the object program will reside in ROM between aaaaH and bbbbH, with the runtime package itself at aaaaH to ccccH-1 and the XYBASIC program at ccccH to bbbbH. For given values of START OF ROM and START OF RAM the runtime package is independent of the source program; therefore object program locations ccccH to bbbbH can be altered to execute different compiled programs without changing locations aaaaH to ccccH-1. The next line indicates RAM statically allocated for runtime package use. The final line indicates the initial number of free bytes remaining below the end of RAM, and appears only if you specified an END OF RAM address. If no free bytes remain, or if the object program's RAM and ROM use areas overlap, an error message will appear and you can try specifying different addresses in response to the compiler prompts.

Finally the compiler writes the object file as [filename].HEX, with the first ROM location as the starting address. The object file is written on the disk specified as the location of the source file.

Object File Execution

The result of executing the object file [filename].HEX is the same as LOADing and RUNning [filename].XYB under the XYBASIC interpreter, with the following exceptions.

1) UNTRAP mode is assumed, i.e. execution continues after nonfatal errors. Error messages give the error line number, but do not print the bad line.

2) Any action which would return you to direct mode (fatal errors, STOP or END, or typing <control-C>) will instead have the same effect as <control-B>, i.e. will exit from XYBASIC and return to the monitor or operating system.

3) Direct commands (NEW, RUN, LIST, CONT), program saving and loading (SAVE and LOAD), and debugging commands (TRACE/UNTRACE, TRAP/UNTRAP, BREAK/UNBREAK) produce nonfatal UF (Unimplemented Feature) errors but have no other effect.

4) Input and output operations (Console In, Console Out, List Out, Console Status) are performed through a jump vector located at the base of the runtime package, just as in the Custom I/O version described in Section 3 above. In the ISIS-II version the jump vector initially contains the appropriate monitor addresses for the Intellec MDS.

In addition, REMarks and spaces (not within quoted strings) are removed from the compiled program to conserve space.

Section 7: XYBASIC Runtime Module

The XYBASIC Runtime Module allows XYBASIC programs to be executed with smaller memory overhead than the XYBASIC Interpreter. Like the XYBASIC Compiler, the Runtime Module does not include direct mode, debugging commands or program saving and loading; its intended use is the execution of previously debugged programs rather than program development.

Like the XYBASIC interpreter, the Runtime Module examines five bytes near the beginning of the Module for default values of WIDTH, END OF MEMORY and program address. If the default width and end of memory values are not specified, the Runtime Module prompts the user in the same manner as the Interpreter. If the default program address is not specified, or if the location specified does not contain the start of a XYBASIC program, the Runtime Module prints the error message

```
PROGRAM NOT FOUND
```

and terminates execution, returning to the monitor or operating system (as if <control-B> were typed) .

If the specified address does contain the start of a XYBASIC program, the program is executed. The result is the same as LOADing and RUNning the program under the XYBASIC Interpreter, with the following exceptions.

Direct commands (NEW, RUN, LIST, CONT), program saving and loading (SAVE and LOAD) , ROMSQuared commands (MOVE and EXEC), editing commands (AUTO, DELETE, EDIT and RENUM), and debugging commands (TRACE, UNTRACE, BREAK and UNBREAK) produce nonfatal UF (Unimplemented Feature) errors but have no other effect.

UNTRAP mode is assumed, i.e. execution continues after nonfatal errors. Error messages give the error line number, but do not print the bad line.

Any action which would return control to direct mode (fatal errors, STOP or END, or typing <control-C>) will instead have the same effect as <control-B>, namely to exit from the XYBASIC Runtime Module and return to the monitor or operating system.

Section 8: Customized OEM Versions

In addition to supporting the versions outlined above, Mark Williams Company will perform modifications to customize XYBASIC to specific OEM requirements.

Chapter III: SHORT FORM DESCRIPTION

This chapter describes each XYBASIC command and function and its error conditions. The descriptions are intended for the user familiar with XYBASIC; the tutorial in Chapter I explains commands in greater detail.

Section 1: Conventions

command	A sequence of typed characters instructing XYBASIC to perform a specific action. Example: PRINT
direct mode	A command typed without a line number is in direct mode, and is executed immediately by XYBASIC. Example: RUN
formula	Any number, numeric variable, or legal combination of numbers, numeric variables, operators and functions.
line number	An integer between 1 and 65535 identifying a command in a program.
logical device	A logical device name: LST#, PUN#, RDR# or CON#.
logical formula	A formula which evaluates to true or false, using relational or logical operators.
program	A sequence of numbered commends.
program mode	A command preceded by a line number is in program (or indirect) mode. XYBASIC does not execute program mode commands immediately, but adds them to the current program instead.
quoted string	A string of characters enclosed in quotes (
reserved word	A sequence of characters indicating a XYBASIC command or function.
string	Any quoted string, string variable, or legal combination of quoted strings, string variables, and string functions.
unquoted string	Any sequence of printable characters.

variable	A name used to refer to stored data. The name must begin with a letter and may contain up to eight letters or digits, but must not contain any reserved word. [Extended] The variable name may end with !, %, or $. Examples: A, DOG, S$, TEMP1%
<control-chars>	Angle brackets are not to be typed, but rather indicate control characters (nonprinting characters typed by depressing the CONTROL key and another key simultaneously). Example: <control-C>
[ ]	Brackets are not to be typed, but rather indicate that the user is to supply the bracketed item. Example: GOTO [line number] indicates GOTO 50 is legal.
...	Ellipses indicate that the information to be supplied can be repeated an arbitrary number of times.

Section 2: Direct Commands

The following commands may be used only in direct mode. An II error occurs if the command is used in program mode.

NEW
Deletes the current program and clears all variables to zero or [Extended] the null string. Turns error TRAP on and TRACE off, and clears breakpoints and ENABLEd interrupts.
[Extended] Resets default variable type to floating point.
[CP/M Disk] CLOSES all OPENed files.

RUN [line number]
Begins execution of the current program, starting at the specified line number. If no line number, starts at the lowest line number. Clears all variables and ENABLEd interrupts, but does not clear line number breakpoints. RESTOREs the READ pointer.
[CP/M Disk] CLOSEs all OPENed files.

CONT
Continues execution after STOP, END, <control-C>, or BREAK with # option. A CN error occurs if XYBASIC is unable to continue, for example if an error occured or if the program has been edited since it was interrupted.

Section 3: Traditional BASIC Commands

LET [variable] = [formula]
Assigns the variable the value of the formula. The LET is optional.
[Extended] A TM error occurs if a string value is assigned to a numeric variable,

or vice versa.
Examples:
LET X = 15 * Y
S$ = "CAT" [Extended]

PRINT [item list]
Prints the given items, which may be formulas or strings. Prints items seperated by semicolon (;) adjacent to each other. Tabs to next fourteen-character column field (eight in integer XYBASIC) between items seperated by comma (,). Prints <carriage return> and <1inefeed> if line is not terminated with comma or semicolon. Question mark (?) may be used as abbreviation for PRINT.
Examples:
PRINT "X ="; X, "Y ="; Y
? I, **J**

LIST [line number1], [line number2]
Lists the current program. If no line number is specified, the entire program is listed. If line number1 is present, the listing starts at that line. If line number2 is present, the listing ends at that line. A listing may be aborted by <control-C>, suspended by <control-S> and resumed by <control-Q>, echoed to the LST device by <control-P>, or suppressed and resumed on the CON device by repeated <control-O>.
Example:
LIST 10, 100

CLEAR
Sets all variables to zero or [Extended] the null string. Removes all breakpoints.

CLEAR [formula]
[Extended] Allocates amount of string space given by value of formula. Clears all variables and removes breakpoints.
Example:
CLEAR 1000

GOTO [line number]
Transfers control to the specified line number. A US error occurs if the line does not exist.
Example:
GOTO 10

INPUT [quoted string] [variable1], [variable2]...
Requests data from the console, prompting the user with the optional string followed by ?. Assigns values typed in to the variables listed. If too many values are typed, the message EXCESS IGNORED is printed. If a bad value is typed, the message REDO is printed and the user is reprompted. An ID error occurs if INPUT is used in direct mode.
[Extended] Leading spaces are removed from unquoted string values.
Examples:
10 INPUT X, Y, Z

```
10 INPUT "VALUE FOR A" A
```

REM [unquoted string]
Allows insertion of comments into program. The string is ignored. REMarks can be terminated only with <carriage return>, not with colon (:).
Example:

```
REM THIS IS A COMMENT
```

IF [logical formula] THEN [line number]
IF [logical formula] THEN [command]
Evaluates the logical formula. If true, transfers control to the line number or executes command following THEN. If false, transfers control to line following the IF / THEN command.
Examples:

```
IF X = 15 THEN 100
IF X = 10 OR Y > 14 THEN PRINT "DONE"
```

STOP
Stops program execution, prints a break message on the console and returns to direct mode.

END
Ends program execution and returns to direct mode.
[CF/M Disk] CLOSEs all OPENed files.

GOSUB [line number]
RETURN
GOSUB transfers control to the subroutine at the specified line number. A US error occurs if the line does not exist. RETURN exits from the subroutine and returns control to the command following the most recent GOSUB executed. A RG error occurs if RETURN is executed without a corresponding GOSUB. RETURN also exits from a routine used to process an ENABLEd interrupt, returning control to the program where it was interrupted. The depth of GOSUB nesting allowed is limited by available memory space, and an OM error occurs if insufficient space remains.
Example:

```
GOSUB 500
```

READ [variable1], [variable2]...
DATA [item1], [item2]...
RESTORE [line number]
READ assigns the specified variables the values from the next unused DATA command. The DATA items may be numbers or [Extended] strings; leading spaces are removed from unquoted string data. RESTORE allows DATA to be reused by resetting the DATA pointer to the specified line number, or to the beginning of the program if no line number is specified. An OD error occurs if a READ is executed when no more DATA are available, and an ID error occurs if DATA is used in direct mode.
Examples:

READ X, Y
10 DATA 10, -20, 50
RESTORE 5000

FOR [variable] = [formula1] TO [formula2] STEP [formula3]
Indicates the beginning of a FOR / NEXT loop. First the initial value formula1 is assigned to the variable. The bound formula2 and increment formula3 are evaluated; if STEP [formula3] is omitted the increment is assumed 1. Then the command after the FOR is executed, unless either the increment is positive and the bound is less than the initial value, or the increment is negative and the bound is greater than the initial value. In that case the loop is not entered: XYBASIC scans through the program for the matching NEXT command and executes the command after it instead. A FR error occurs if the matching NEXT is not found.
Examples:
FOR I = 1 TO 100
FOR I = 0 TO 30 * X STEP 10

NEXT [variable1], [variable2]...
Indicates the end of a FOR / NEXT loop. The increment is added to the current value of the matching FOR variable and the result is compared to the bound. If the loop continues, the command after the matching FOR is executed; otherwise the command after the NEXT is executed. A NF error occurs if NEXT is executed without a corresponding FOR or if the optional variable names given do not match.
Examples:
NEXT
NEXT J, I

ON [formula] GOTO [line number], [line number]...
ON [formula] GOSUB [line number], [line number]...
Transfers control to the Ith line number in the list, where I is the truncated value of the formula. An ON error occurs if the value of I is less than one or greater than the number of line numbers in the list.
Examples:
ON I GOTO 100,200,300
ON (N MOD 2) + 1 GOSUB 1000, 2000

DIM [variable] (formula, ...)
Allocates space for an array with name specified by variable, and initializes all array elements to zero or [Extended] the null string. A DD error occurs if an array is DIMensioned more than once. A BS error occurs if the value of a subscript is less than zero or greater than the given array bound. An OM error occurs if the array is too large for remaining free memory.
Example:
10 DIM A(100,10), B(N*2)

DEF FN [variable1] (variable2, ...) = [formula]
Defines a function named FN variable1. The parameters (variable2, ...) are

optional. A DD error occurs if a function is DEFined more than once, and an ID error occurs if DEF FN is used in direct mode.
Example:
10 DEF FN A (X) = X * 3 + 4

DEF INT [letter1] - [letter2]
DEF SNG [letter1] - [letter2]
DEF STR [letter1] - [letter2]
[Extended] Resets the default variable type for variable names starting with letter1 through letter2 to INTeger, SiNGle precision floating point or STRing. - [letterZ] is optional; if omitted, resets the default variable type for variable names starting with letter1.
Examples:
DEF INT I
DEF STR A - B

Section 4: Numeric Formulas

A variable name is a letter followed by up to seven additional letters or digits, but not containing any reserved word. An array variable (declared in a DIM command) must have a subscript (formula, ...). A BS error occurs if each subscript is not between 0 and the declared size of the array. A SN error occurs if a variable has too few or too many subscripts.
[Extended] The variable name may be followed by !, %, or $.
Examples:
X
DOG$ [Extended]
A (I, J)

Integer numbers between -32768 and 32767 may be specified in decimal, in binary (prefixed by &) or in hexadecimal (prefixed by #).
Examples:
100
-32768
&11011

[Extended] Numbers may be specified as sequences of decimal digits with optional decimal point, followed by an optional exponent. The exponent (if any) consists of the letter E, an optional sign, and decimal digits. The vaiile of the number may be in the approximate range -1 .7 * 10^38 to 1 .7 * 10^38, and has a precision of more than six decimal digits.
Examples:
3.14159
1.5E-4

A formula is any legal combination of numeric variables, numbers, parentheses, and the operators and functions listed below.

+	addition
-	subtraction, negation
*	multiplication
/	division
\	integer division [Extended]
^	exponentiation [Extended]
MOD	remainder

Arithmetic operators apply the desired operation to the given arguments. An OV error occurs if the result is outside the range of representable values.
Examples:
X = Y + 1
X = Y * 5 + (Z + 1) / 2
X = Y MOD 10
X = Y ^ (Z \ 2) [Extended]

=	equal to
<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

Relational operators compute the result (true or false) of comparing formulas or [Extended] string formulas.
Examples:
IF X = 0 THEN GOSUB 100
IF I <> 0 THEN X = A (I)
IF A$ < "CAT" THEN = A$ = STR$ (X) [Extended]

AND	logical and
NOT	logical not
OR	logical inclusive or
XOR	logical exclusive or

));

Logical operators apply the desired operation bitwise to the given 16-bit integer arguments, and can be used for bit manipulation or for constructing logical formulas.
Examples:
IF NOT (X = 0 AND Y = 0) THEN GOSUB 100
X = Y XOR &1100

Within a formula, operators occurring higher on the following list are evaluated first; operators on the same level are evaluated from left to right.

JOIN
^ [Extended]
*, /, MOD, \ [Extended]
+, -
=, <, >, <=, >=, <>

NOT
AND
OR, XOR

Example:
IF NOT X = 0 AND Y + 3 * Z <= 2 * -I + 1 THEN 100
means IF ((NOT (X = 0)) AND (Y + (3 * Z)) <= (2 * (-I) + 1)) THEN 100

ABS (formula)
Returns the absolute value of the formula.
Example:
X = ABS (-5 * Y)

SGN (formula)
Returns one if the sign of the formula is positive, zero if zero, and minus one if negative.
Example:
X = SGN (Y)

SQR (formula)
[Extended] Returns the square root of the value of formula. An FC error occurs if the argument is negative.
Example:
PRINT SQR (2)

LOG (formula)
[Extended] Returns the natural logarithm of formula. An FC error occurs if the argument is less than or equal to 0.
Example:
X = LOG (10)

EXP (formula)
[Extended] Returns e ^ formula, where e is the Euler number 2.71828...
Example:
X = EXP (Y + Z)

COS (formula)
SIN (formula)
TAN (formula)
[Extended] Returns the cosine, sine or tangent of the value given in radians by formula.
Example:
X = SIN (1 - Y ^ 2)

ATN (formula)
[Extended] Returns the arctangent in radians in the range -pi/2 to pi/2 of the value of formula.
Example:
X = ATN (I + J)

INT (formula)
[Extended] Returns the integer part of the formula.
Example:
I$ = 2 * INT (X)

RND
[Integer] Returns a pseudorandom number between 0 and 32767. The formula X + RND MOD (Y-X+1) returns a pseudorandom number between X and Y.
Examples:
X = RND
X = 1 + RND MOD 10

RND (formula)
[Extended] Returns a pseudorandom number between 0 and the value of the given formula. RND (O) returns a pseudorandom number between 0 and 1.
Example:
X = RND (1)

FRE
Returns the amount of free memory in bytes.
Example:
PRINT UNS (FRE)

UNS (formula)
[Extended] Returns the value of the formula, considered as an unsigned 16-bit integer representation.
[Integer] May be used only in PRINT commands.
Example:
PRINT UNS (-1)

FN [variable] (formula, ...)
Evaluates the referenced user-defined function. The number of parameters (formula, ...) must agree with the number in the DEFinition.
Example:
X = FNA(45)

RESET (formula1, formula2)
SET (formula1, formula2)
TEST (formula1, formula2)
RESET and SET set the bit specified by formula2 to zero or one in integer formula1. TEST returns the value of the bit specified by formula2 in formula1.
Examples:
X = SET(Y, 0)
X = RESET(X, 5)
X = TEST (Y, 0)

ROTATE (formula1, formula2)
LSHIFT (formula1, formula2)

RSHIFT (formula1, formula2)
Right rotates, left shifts or right shifts the integer value of formula1 the number of binary places specified by formula2.
Examples:
 = ROTATE (Y, 5)
X = LSHIFT (X, 3)
X = RSHIFT (X, 5)

BCD (formula)
BIN (formula)
BCD converts the given formula from binary to BCD representation, and BIN converts from BCD to binary representation. An FC error occurs if the argument is outside the domain of the function.
Examples:
X = BCD (100)
X = BIN (#64)

LSBYTE (formula)
MSBYTE (formula)
[formula1] JOIN [formula2]
LSBYTE and MSBYTE return the least or most significant 8 bits of the 16-bit integer value of the formula. JOIN concatenates two 8-bit values into a 16-bit value.
Examples:
X = LSBYTE (30050)
X = MSBYTE (30050)
X = #F JOIN I

GET
Returns ASCII value (with parity bit reset) of any character typed; returns 0 if no character typed.
Example:
 IF GET = 89 THEN PRINT "YES"

IN (formula)
SENSE (formula1, formula2)
IN returns the value on the input port specified by formula. SENSE returns the value of bit number formula2 on the input port formula1.
Examples:
X = IN (10)
X = SENSE (10, 7)

PEEK (formula)
Returns the value in the memory location specified by formula.
Example:
X = PEEK (48)

IOBYTE
Returns the value of the system I/O byte.

Example:
X = IOBYTE

SPC (formula)
TAB (formula)
SPC prints formula spaces. TAB spaces to the column specified by formula. Each may only be used in PRINT commands.
Example:
PRINT TAB (10); "THIS STARTS IN COLUMN 10"
PRINT SPC (10)

POS
Returns the column number of the most recent character PRINTed by XYBASIC.
Example:
IF POS < 20 THEN PRINT TAB(20);

FIRST
LAST
Returns the address of the first or last location used to store the current XYBASIC program.
Example:
PRINT LAST - FIRST + 1

Section 5: String Formulas

The string functions in this section may only be used in Extended XYBASIC.

A quoted string is a string of characters enclosed in quote marks (" "). The string containing no characters is called the null string. A string formula is any quoted string, string variable, or legal combination of quoted strings, string variables, and string functions. A TM error occurs if a numeric argument is used where a string is expected, or vice versa. An OS error occurs if XYBASIC runs out of space for string storage.

LEN (string)
Returns the length of the string, i.e. the number of characters it contains.
Example:
PRINT LEN (A$), A$

[string1] + [string2]
Returns the string consisting of string1 followed by string2, i.e. concatenates the strings. A LS error occurs if the result is longer than 255 characters. An ST error occurs if string1 or string2 is too complicated a string formula.
Example:
A$ = B$ + "ABC"

LEFT$ (string, formula)
RIGHT$ (string, formula)

Returns the leftmost or rightmost formula characters of string. An FC error occurs if the value of formula is less than 0 or greater than 255.
Example:
PRINT RIGHT$ (A$, LEN (A$) - 2)

MID$ (string, formula1, formula2)
Returns the substring of string formula2 characters long, starting at character formula1. If formula2 is omitted, returns the right part of the string starting at character formula1. An FC error occurs if either formula is less than 0 or greater than 255.
Examples:
A$ = MID$ (B$, 4, 3)
A$ = MID$ (B$, I) + LEFT$ (B$, I)

CHR$ (formula)
Returns the string containing the character with ASCII value formula.
[Integer] May be used only in PRINT commands.
Example:
PRINT CHR$ (ASC ("A") + I)

ASC (string)
Returns the ASCII value of the first character of string. An FC error occurs if its argument is the null string.
Example:
PRINT ASC ("A")

INSTR (formula, string1, string2)
Returns the first character position at which string1 contains the substring string2. If optional formula is specified, returns the first character position greater than or equal to formula at which string1 contains string2.
Examples:
I = RIGHT$ (A$, INSTR (A$, "Mr. ") + 4)
I = INSTR (5, A$, "CA")

GET$
Returns string consisting of character typed, if any; returns null string if no character typed.
Example:
IF GET$ = "Y" THEN PRINT "YES"

STR$ (formula)
Returns the value of formula as a string of characters, as PRINTed by XYBASIC.
Example:
A$ = "$" + STR$ (X)

VAL (string)
Returns the numeric value of the constant represented by string.
Example:
PRINT VAL ("1.5E3")

FRE$
[Extended] Returns the amount of free string space in bytes.
Example:
PRINT FRE$

BIN$ (formula)
HEX$ (formula)
OCT$ (formula)
Returns string consisting of binary, hexadecimal or octal representation of integer formula.
Example:
PRINT HEX$ (I + J)

Section 6: Input / Output Commands

NULL [formula]
Instructs XYBASIC to send the console the number of nulls specified by formula after each <carriage return> and <line feed>; this number is initially 0. A BY error occurs if the value of the formula is not an 8-bit quantity.
Example:
NULL 5

ASSIGN [logical device] [formula]
Reassigns the logical device to the physical device specified by formula. A FC error occurs if the value of the formula is not between 0 and 3.
Example:
ASSIGN RDR# 2

SAVE [quoted string]
[Custom I/O] Saves the current program through the PUNch device in format described in Chapter II, Section 3. The quoted string may consist of one to eight upper case letters and digits.
Example:
SAVE "EXAMPLE"

LOAD [quoted string]
[Custom I/O] Loads a program through the ReaDeR device. If the quoted string is omitted, XYBASIC loads the first program it finds. Otherwise XYBASIC continues reading until it finds the specified program. An RO error occurs if XYBASIC is not addressing its working space. A CS error occurs if the LOAD is unsuccessful.
Example:
LOAD "SAMPLE"

SAVE [diskname] [quoted string]
SAVE [diskname] [quoted string] ,A
SAVE [diskname] [quoted string] ,H

[ISIS-II] Saves the current program as a disk file. The diskname is optional, and :F0: is assumed if it is omitted. The quoted string may contain one to six upper case letters and digits. If neither ,A nor ,H is specified, the program is saved in XYBASIC internal representation, as a file string.XYB. If ,A is specified, the program is saved in ASCII representation, as a file string.BAS. If ,H is specified, the program is saved in Intel HEX format, as a file string.HEX. A DK error occurs if a disk operation is unsuccessful.
Examples:
SAVE "EXAMPL" 'SAVE :F0:EXAMPL.XYB
SAVE :F1:"EX2", H 'SAVE :F1:EX2.HEX

LOAD [diskname] [quoted string]
LOAD [diskname] [quoted string] ,A
LOAD [diskname] [quoted string] ,H
[ISIS-II] Loads the program from the specified file. An RO error occurs if XYBASIC is not addressing its working space. A DK error occurs if a disk operation is unsuccessful.
Examples:
LOAD "EXAMPLE" 'LOAD :F0:EXAMPL.XYB
LOAD :F1:"EX2",H 'LOAD :F1:EX2.HEX

SAVE [string] ,A
[CP/M] Saves the current program as a disk file. The string specifies the file name, and may consist of an optional disk name followed by one to eight letters or digits. The currently logged disk is assumed if no disk name is given. The suffix ,A is optional; if omitted, the program is saved in XYBASIC internal representation, as a file string.XYB. If ,A is specified, the program is saved in ASCII representation, as a file string.BAS. A DK error occurs if a disk operation is unsuccessful.
Examples:
SAVE "EX1" 'SAVE EX1.XYB ON LOGGED DISK
SAVE "B:GAME",A 'SAVE B:GAME.BAS

LOAD [string] ,A ,R
[CP/M] Loads the program from the specified file. The suffix ,A is optional; if specified, XYBASIC loads the ASCII file string.BAS. The ,R suffix is optional; if specified, XYBASIC RUNs the file after loading it. An RO error occurs if XYBASIC is not addressing its working space. A DK error occurs if a disk operation is unsuccessful.
Example:
LOAD "EX1", R 'LOAD EX1.XYB AND RUN IT
LOAD "B:GAME",A 'LOAD B:GAME.BAS

Section 7: Control Commands

TIME
DELAY [formula1], [formula2], [formula3]
TIME calibrates the DELAY command for nonstandard 8080 systems; it sends a

bell character to the console, then waits for two <carriage return>s seperated by exactly 60 seconds. DELAY suspends program execution for formula1 minutes, formula2 seconds and formula3 hundredths of seconds, where formula2 and formula3 are optional. Typing any character terminates the DELAY and resumes program execution.
Example:
DELAY 0,5,50

OUT [formula1], [formula2]
Outputs the value of formula2 on the output port specified by formula1. A BY error occurs if either formula is not an 8-bit quantity.
Example:
OUT 100, X

POKE [formula1], [formula2]
Puts the value of formula2 into the memory location specified by formula1. A BY error occurs if formula2 is not an 8-bit quantity.
Example:
POKE 25, X

CALL [number], [parameter1], [parameter2]...
Calls the machine language subroutine at location specified by number. The parameters are optional, and may be either [variable] or *[array variable]. The subroutine GTPAR returns information about the next parameter in the list, with type information passed in the A register and additional information in other registers as required.
Example:
CALL #8000, I, *A, B(1)

SCALL [number], [integer var1], [integer var2], [integer var3]
Calls the machine language subroutine at location specified by number. If the optional variables are present, their values are passed in registers BC, DE and HL, and the values in BC, DE and HL when the routine returns are assigned to the variables. An MC error occurs if more than three parameters are specified or if the parameters are not integer variables.
Example:
SCALL #7400, X

WAIT [formula1], [formula2], [formula3], $
Suspends processing until the port specified by formula1 has the value specified by formula2, masked by optional formula3. The optional $ indicates processing is to continue if any bit matches. A BY error occurs any of the formulas are not 8-bit quantities.
Example:
WAIT 10, 0

ENABLE [line number], [formula1], [formula2], [formula3], $
Specifies an interrupt condition to be tested before each command is executed. The condition is fulfilled if the value on input port formula1 matches the value

of formula2, masked by optional formula3; if the optional $ is present, the condition is fulfilled if any bit matches. If it is not fulfilled, program execution continues normally; if it is fulfilled, an interrupt occurs and control is transferred to the subroutine at the given line number. An EN error occurs if more than eight interrupts are ENABLEd simultaneously, and an ID error occurs if ENABLE is used in direct mode.
Example:
10 ENABLE 100, 22, &101

DISABLE [line number]
Removes the interrupt set by the given line number; removes all interrupts if no line number is given.
Example:
DISABLE 10

RANDOMIZE [formula]
Uses formula to reinitialize the pseudorandom number generator.
Example:
RANDOMIZE 125

Section 8: Debugging Commands

TRACE
UNTRACE
Turns the trace feature on or off. When on, the number and contents of each executed line and the value of each modified variable is printed on the console.

TRAP
UNTRAP
Turns the error trap on or off. When on, XYBASIC returns to direct mode after errors. When off, XYBASIC tries to recover and continue execution after errors.

BREAK [line number], [formula]; [variable1], [variable2]... ; $
Sets a breakpoint on the given line number, so a break message is printed whenever the line is executed. If the optional formula is present, the break occurs only after the line has been executed formula times. If any variables are listed, their names and values are also printed when the break occurs. XYBASIC then returns to direct mode if the optional $ is present.
Examples:
BREAK 100; $
BREAK 10, 20; A, B; $

BREAK [variable1], [variable2]...
Sets breakpoints on the given variables, so whenever they are modified the variable name and value are printed.
Example:
BREAK A, B, C

UNBREAK [line number]
UNBREAK [variable1], [variable2]...
Removes breakpoints set on line numbers or variables, and removes all breakpoints if the optional line number is omitted.
Examples:
UNBREAK 100
UNBREAK I, J, K

Section 9: ROMSQuared Commands

EXEC [formula]
If optional formula is given, XYBASIC addresses the program at specified location rather than the program in working space. If the formula is omitted, addresses the program in working space.
Example:
EXEC #8000

MOVE FROM [formula]
MOVE TO [formula]
MOVE FROM copies the program from the specified location to XYBASIC's working space. MOVE TO copies the program from working space to the specified location. MOVE does not change the location of the program XYBASIC is addressing. An RO error occurs if the specified location does not contain a legal XYBASIC program (MOVE FROM), or if the location is not RAM (MOVE TO).
Example:
MOVE TO #A000

Section 10: Editing Commands

The following commands may be used only in editing versions of XYBASIC. An RO error occurs if an editing command is attempted while XYBASIC is addressing a program outside its working space. An II error occurs if an editing command is attempted in program mode.

AUTO [line number 1], [line number 2]
Allows entering new lines of a program without typing line numbers, starting at line number 1 with increment line number 2. Both arguments are optional, and are defaulted to 10 if not present. XYBASIC prompts with line number of next line (followed by an asterisk if a line with the given number exists in the current program, or a space if not), and waits for user to enter a line. XYBASIC exits from AUTO mode when <control-C> is typed or when <carriage return> is typed at the beginning of a line. A SN error occurs if a line number is typed at the beginning of an AUTO mode line.
Examples:
AUTO 100,20
AUTO

DELETE [line number 1], [line number 2]
Deletes section of XYBASIC program starting at line number 1 and ending at line number 2. If line number 2 is omitted, deletes line number 1. If specified line numbers are not found, deletes all lines following line number 1 and preceding line number 2. A US error occurs if line number 2 is omitted and line number 1 is not found.
Example:
DELETE 110,150

EDIT [line number]
Allows changing line given by line number of current program without retyping entire line. If the line number is omitted, EDITs the line most recently added to the program or line in which most recent error occurred. XYBASIC types the specified line, then waits for user to type editing commands as described under Special Characters in Section 12 below. Typing <carriage return> ends the editing process and returns user to direct mode. A US error occurs if the specified line is not found. An EX error occurs if a line containing too many characters is EDITed.
Example:
EDIT 120

RENUM [line number 1], [line number 2], [line number 3]
Renumbers the current program, with line number 1 becoming line number 3 and successive line numbers incremented by line number 2. If line number 3 is omitted, it is assumed to be the same as line number 1. If line number 2 is omitted, it is assumed to be 10. If line number 1 is omitted, it is assumed to be the first line number of the program. A US error occurs and no renumbering takes place if the specified renumbering is impossible. A US error occurs and renumbering takes place if the program contains references to nonexistent line numbers.
Examples:
RENUM 10, 100, 1000
RENUM

Section 11: CP/M Sequential Disk Commands

The following commands and function may be used only in CP/M XYBASIC with sequential disk operations.

OPEN I, @[formula], [string]
OPEN O, @[formula], [string]
OPEN U, @[formula], [string]
Opens a disk file for Input, Output or Update. The formula specifies an integer file number between 1 and 255 associated with the file. The string specifies a filename, and may consist of an optional disk name, a filename, and an optional filetype. A SN error occurs if the filename is specified incorrectly. A BF error occurs if the formula is not between 1 and 255. A FN error occurs if a

nonexistant file is OPENed for Input or Update. A FO error occurs if the filename or file number is already associated with an OPEN file. An OP error occurs if too many files are OPENed simultaneously.
Examples:
OPEN I, @1, "OLD.DAT"
OPEN O, @2, "NEW.DAT"
OPEN U, @3, "B:UPDATE.DAT"

CLOSE
CLOSE @[formula]
Closes all files or the file with file number formula. A BF error occurs if the specified file is not OPEN.
Example:
CLOSE @2

PRINT @[formula], [item list]
Sends information in item list to file number formula. The items may be formulas or strings. Sends items separated by semicolon (;) adjacent to each other. Tabs to next fourteen-character column field between items seperated by comma (,). Sends <carriage return> and <11nefeed> if line not terminated with comma or semicolon. Question mark (?) may be used as abbreviation for PRINT. If value of formula is 0, PRINTs items on the console. A BF error occurs if the value of formula is not 0 or the number of an OPEN file. A DF error occurs if the disk is full. A FM error occurs if the specified file is OPEN for Input.
Examples:
PRINT @1; "**J** = "; **J**;
PRINT @I; A; B; C

MARGIN @[formula1], [formula2]
Specifies formula2 as maximum width for output lines on file number formula1. If value of formula1 is 0, changes width of console device output line to formula2. A BF error occurs if the file number formula1 is not 0 or the number of an OPEN file. A BY error occurs if the width formula2 is greater than 255.
Example:
MARGIN @1, 50

INPUT @[formula], [variab1e1], ...
Reads information from file number formula and assigns to variable1,... If value of formula is 0, performs a normal INPUT from the console. Does not prompt or give REDO or EXCESS IGNORED messages when reading from file. A BF error occurs if the file number formula is not 0 or the number of an OPEN file. A FM error occurs if the specified file is OPEN for Output or Update. A FI error occurs if a bad item is read during INPUT. An EF error occurs if INPUT tries to read past the end of file. An ID error occurs if INPUT is used in direct mode.
Example:
10 INPUT @2, I, **J**, K

LINPUT [string variable]
LINPUT @[formula], [string variable]

Reads a line of characters from the console or from file number formula and assigns string variable the string consisting of the characters, not including <carriage return>. Does not prompt. If value of formula is 0, does a LINPUT from the console. A BF error occurs if the file number formula is not 0 or the number of an OPEN file. A FM error occurs if the specified file is OPEN for Output or Update. An EF error occurs if LINPUT tries to read past the end of file. An ID error occurs if LINPUT is used in direct mode.
Example:
10 LINPUT @I, S$

EOF (formula)
Returns -1 if the file number formula contains more characters, and returns 0 if not. A BF error occurs if formula is not the number of an OPEN file. An FM error occurs if formula is the number of a file OPEN for Output or Update.
Example:
70 IF NOT EOF (1) THEN GOTO 50

DIR [string]
Prints the names of all files with filenames matching the given string. If string is omitted, prints the names of all files on the currently logged disk.
Example:
DIR "B:*.XYB"

SCRATCH [string]
Erases the file named string from the disk.
Example:
SCRATCH "TEMP.DAT"

CLEAR @[formula]
Tells XYBASIC to allocate space allowing up to formula disk files to be OPEN simultaneously. Variables are CLEARed. An OM error occurs if insufficient memory exists to allocate the given number of files.
Example:
CLEAR @3

Section 12: Special Characters

:	Used between commands to allow multiple commands on a single line. Example: PRINT "HI" : X = 14 : GOTO 100
’	Delimits an on-line comment. The characters following the ’ are ignored. Example: PRINT "START" ’START OF PROGRAM
?	Abbreviation for PRINT. Example: ?X

!	[Extended] Indicates a floating point variable.
%	[Extended] Indicates an integer variable.
$	[Extended] Indicates a string variable.
<return>	Carriage return (the RETURN or CR key) must be typed at the end of each line typed to XYBASIC.
<rubout>	Erases the last character typed, echoing erased characters within slashes.
<control-b>	Exits from XYBASIC and returns the user to the resident operating system.
<control-C>	Interrupts program execution, prints break message and returns to direct mode.
<control-G>	May be used only within quoted strings in PRINT commands; prints as an audible bell or beep on most console devices.
<control-H>	Erases the last character typed and echoes <control-H> to backspace the cursor on a CRT.
<control-O>	Suppresses console output until next <control-O> or until an error, an INPUT command or a return to direct mode occurs.
<control-P>	Echoes all output to the selected LST device until next <control-P> typed.
<control-Q>	Resumes execution suspended by <control-S>.
<control-R>	Retypes the current input line.
<control-S>	Suspends program execution until <control-S> or <control-Q> typed.
<control-U>	Deletes the current input line.

The following characters may be used during line editing in editing versions of XYBASIC.

<return>	Terminates editing and returns to direct mode.
<printable>	Printable characters are inserted in the line at current cursor position.

<rubout>	Delete character left of cursor.
<control-B>	[Boot] Exits from XYBASIC and returns to operating system or monitor.
<control-C>	Terminates editing and returns to direct mode, leaving previous contents of line being edited unchanged.
<control-D>	[Delete] Deletes character right of the cursor.
<control-E>	[Edit] Enters the editor with contents of most recently typed line.
<control-F>	[Find] Moves cursor to right of next occurence of <printable character> typed after <control-F>; echoes <control-G> and leaves cursor unchanged if not found.
<control-G>	[Bell] Inserts <control-G> (bell or beep) in line.
<control-H>	[Backspace] Erases character left of cursor and echoes <control-H> to backspace CRT cursor.
<control-K>	[Kill] Kills all characters right of the cursor.
<control-L>	[Left] Types the characters right of the cursor, followed by <carriage return> and <linefeed>, and leaves cursor at left of line.
<control-N>	[Next] Finds next occurence of <printable character> last specified in <control-F> search command; echoes <control-G> and leaves cursor unchanged if not found.
<control-R>	[Retype] Types characters right of cursor, <carriage return> and <linefeed>, and retypes characters left of cursor; cursor unchanged.
<control-T>	[Type] Moves cursor one character right, echoing character passed over.
<control-U>	[Undo] Discards current contents of line being edited and begins editing anew with original contents of line.

Appendix 1: Initialization Dialog

When you load XYBASIC it leads you through the following dialog to obtain information about your system.

```
XYBASIC {version} REV n.m
COPYRIGHT 1978, 1979, 1980 BY MARK WILLIAMS COMPANY, CHICAGO
WIDTH?
END OF MEMORY?
xxxxx BYTES FREE
OK
```

The first line indicates that you are using revision n.m of XYBASIC. The indicated {version} may be CP/M or ISIS-II, with EDIT to indicate a version including editing commands and DISK to indicate a version including CP/M Disk commands,

The second line asks you the width of your terminal; XYBASIC uses this information for formatting output. You should respond by typing a decimal number (up to 255), followed by a <carriage return>. If you just type a <carriage return> XYBASIC assumes a width of 80 columns.

The third line asks you how much memory you have in your system. You should respond by typing in decimal the highest RAM memory address in your computer system, followed by a <carriage return>. If you wish to reserve memory for machine language routines, type the highest address you want XYBASIC to use. If you just type <carriage return>, XYBASIC will find and use the highest available memory address automatically. Under CP/M or ISIS-II this information is obtained from the operating system.

The fourth line tells you how many bytes of memory remain free for program and variable storage. You are then given the OK prompt which indicates that you are talking to XYBASIC.

Appendix 2: Speed and Space Hints

For some purposes it makes little difference how fast a program runs, while for others it is critical. The best way to make a program faster starts with a careful consideration of how it works and elimination of unnecessary steps. For example, a value computed within a FOR-loop which does not depend on the value of the FOR variable can often be computed once (before the loop) instead of many times (inside the loop).

In Extended XYBASIC integer arithmetic is faster than floating point arithmetic, and integer variable storage uses less memory space than floating point variable storage. When you know the value of a variable will always be an integer in the range -32767 to 32767, you conserve both speed and space by using an integer variable rather than a floating point variable. Similarly, execution of an integer FOR-loop is much faster than execution of the corresponding floating point FOR-loop. Since the subscripts of an array variable are always integers, you can save considerable conversion time by using integer variables rather than floating point variables in subscript formulas.

The execution speed of Extended XYBASIC programs which perform extensive string manipulation often may be improved by allocating additional string space. The extra string space lets XYBASIC require time-consuming garbage collection less frequently, and the resulting time savings can be dramatic. You can also save time by replacing a quoted string used repeatedly in string relations (for example, "a" in IF S$ >="a" THEN...) with a reference to a string variable set to the given value outside the loop containing the relation (for example, A$ = "a" and IF S$ > A$ THEN...).

Since XYBASIC is an interpreter, it looks up each variable used in a command every time the command is executed. The lookup procedure searches the symbol table starting with the most recently defined variable. XYBASIC spends a great deal of its execution time looking up variables, so rearranging programs to facilitate variable lookup can produce substantial speed improvements.

Several techniques let you speed up a program at the expense of using more memory space. Writing out a subroutine 'in-line' instead of using GOSUB makes the program longer, but saves the time used executing GOSUBs and RETURNS. Replacing an array variable with several simple variables often improves speed (since XYBASIC does not need to compute subscripts), although it usually results in an incomprehensible program.

Using XYBASIC's TRACE and BREAK commands for debugging also slows down program execution. Usually the execution speed when TRACEing is limited by the baud rate of the console; that is, XYBASIC waits for your console to print a line number before executing the next command. By repeatedly typing <control-O> to suspend and resume output, you can keep track of program execution while wasting less time waiting for console printing. Similarly, using the many

powerful optional forms of BREAK instead of TRACE often lets your program run faster while you find bugs.

The ENABLE command slows down program execution a great deal, since the specified condition is checked before each command is executed. To improve program speed you should DISABLE interrupts whenever they are unnecessary, and naturally you should avoid ENABLEing interrupts which you do not need.

If your computer does not have enough memory space to run a large XYBASIC program, you may still find it possible to run it after you rewrite it to decrease its memory usage. Just thinking carefully about a program often lets you save space by eliminating extra steps, but sometimes you will need to know how XYBASIC uses space. Some of the byte-saving techniques suggested below are good programming practice, while others are recommended only if you must conserve space at all costs.

Each line in a program has a fixed overhead, regardless of the number of decimal digits in the line number. You cannot save space by using lower line numbers, but you can conserve by using : and ' to decrease the total number of lines. Grouping related commands and comments on a single line also makes programs more comprehensible.

XYBASIC stores each reserved word (command or function name) of a program in a single byte, and stores every other character (including spaces) in a single byte. You can therefore conserve space by using fewer characters, for example by eliminating spaces, eliminating REMarks, eliminating LETs, and shortening variable names. Each of these techniques usually makes your program harder to read and debug, though.

Using XYBASIC's full variety of commands and functions also lets you write more compact programs. A repeated section of code can be made into a subroutine and called with GOSUB. FOR-loops can replace some loops defined with IF / THEN and GOTO. ON commands can replace sequences of IF / THEN commands. Using the right function, for example TEST (I,2) instead of RSHIFT(I,2) AND 1, also produces more efficient programs.

Besides the space used to store your program, XYBASIC uses free memory to hold information about variables and control information about GOSUBs and FORs. Each variable uses space, so you can save bytes by using the same variable at different places in a program. Each GOSUB uses space until you RETURN, so you must be careful to always RETURN from subroutines. Similarly, each FOR uses space until NEXT exits from the loop, so you should avoid GOTOing out of FOR loops. However, the space used is recovered if you reenter the loop.

Appendix 3: Reserved Word List

ABS
AND
ASC [Extended]
ASSIGN
ATN [Extended]
AUTO [Editing]
BCD
BIN
BIN$
BREAK
CALL
CHR$
CLEAR
CLOSE [CP/M Disk]
CON#
CONT
COS [Extended]
DATA
DEF
DELAY
DELETE [Editing]
DIM
DIR [CP/M Disk]
DISABLE
EDIT [Editing]
EOF [CP/M Disk]
ENABLE
END
EXEC
EXP [Extended]
FIRST
FN
FOR
FRE
FRE$
FROM
GET
GET$ [Extended]
GOSUB
GOTO
HEX$
IF
IN
INPUT
INSTR [Extended]
INT [Extended]
IOBYTE
JOIN
LAST
LEFT$ [Extended]
LEN
LET
LINPUT [CP/M Disk]
LIST
LOAD
LOG [Extended]
LSBYTE
LSHIFT
LST#
MARGIN [CP/M Disk]
MID$ [Extended]
MOD
MOVE
MSBYTE
NEW
NEXT
NOT
NULL
OCT$
ON
OPEN [CP/M Disk]
OR
OUT
PEEK
POKE
POS
PRINT
PUN#
RANDOMIZE
RDR#
READ
REM
RENUM [Editing]
RESET
RESTORE
RETURN
RIGHT$ [Extended]
RND
ROTATE
RSHIFT
RUN
SAVE
SCALL
SCRATCH [CP/M Disk]
SENSE
SET
SGN
SIN [Extended]
SNG [Extended]
SPC
SQR [Extended]
STEP
STOP
STR [Extended]
STR$ [Extended]
TAB
TAN [Extended]
TEST
THEN
TIME
TO
TRACE
TRAP
UNBREAK
UNS
UNTRACE
UNTRAP
VAL [Extended]
WAIT
XOR

Appendix 4: Character Set

XYBASIC lets you use any printable characters (ASCII codes 20 through 7E hexadecimal -- see Appendix 5) in a program. In particular you can use both upper and lower case alphabetic characters. Lower case characters are converted to upper case automatically, except within quoted strings, DATA, and REM commands. XYBASIC also recognizes some nonprintable (or control) characters, namely:

<carriage return>	Terminates current line
<rubout>	Erases last character typed, echoing erased characters within slashes (/)
<control-B>	Exits from XYBASIC and returns to the operating system or monitor
<control-C>	Interrupts program execution and returns to direct mode
<control-E>	[Editing version] Enters editor with contents of most recently typed line
<control-G>	PRINTS as audible bell or beep when used in quoted string
<control-H>	Erases last character typed and echoes <control-H> to backspace CRT cursor
<control-O>	Suppresses console output until next <control-O>
<control-P>	Echoes all output to the selected LST device until next <control-P>
<control-Q>	Resumes program execution after <control-S>
<control-R>	Retypes the current line
<control-S>	Suspends program execution until <control-Q> or <control-S> typed
<control-U>	Deletes the current line

In versions of XYBASIC which include editing commands (as described in Chapter I, Section 13), the following control characters may be used during line editing with the EDIT command.

<carriage return>	Terminates editing and returns to direct mode
<rubout>	Deletes character left of cursor, echoing deleted character within slashes (/)
<control-B>	Exits from XYBASIC and returns to the operating system or monitor
<control-C>	Terminates editing and returns to direct mode, leaving previous contents of edited line unchanged
<control-D>	Deletes character to the immediate right of cursor
<control-F>	Moves cursor to the right of the next occurrence of the following character typed
<control-G>	Inserts <control-G> (bell or beep) at current cursor position
<control-H>	Erases character left of cursor and echoes <control-H> to backspace CRT cursor
<control-K>	Kills all characters to the right of cursor
<control-L>	Types characters to the right of the cursor, followed by <carriage return> and <linefeed>, and leaves cursor to the left of first character
<control-N>	Finds the next occurrence of the search character last specified in a <control-F> search command
<control-R>	Types the characters right of cursor, followed by <carriage return> and <linefeed>, and then retypes the characters left of cursor; cursor position unchanged
<control-T>	Types the character right of cursor and moves cursor to right
<control-U>	Discards the current contents of the line being edited and begins editing anew with original contents of line

All other nonprintable characters have no meaning to XYBASIC and are ignored.

Appendix 5: ASCII Character Equivalents

Like many other computers, the 8080 uses the ASCII (American Standard Code for Information Interchange) code to communicate with peripheral devices. Bits 0 through 6 are used to represent 128 possible characters. Bit 7 is a parity bit; its value may be 0 or 1, or it may be ignored, depending on the parity conventions of a particular device. XYBASIC ignores bit 7 of any character it receives. Appendix 4 details which characters XYBASIC accepts.

The 128 characters fall into four groups. Codes 0 through 31 (hex 0 through 1F) represent control characters. Codes 32 through 63 (hex 20 through 3F) represent special characters and digits. Codes 64 through 95 (hex 40 through 5F) include the upper case alphabetic characters. Finally, codes 96 through 127 (hex 60 through 7F) include the lower case alphabetic characters. The table below gives the decimal, binary and hexadecimal values corresponding to each ASCII character. Some consoles use other characters for a few values; for example, NUL (ASCII O) is sometimes <control-shift-P> rather than <control-@>.

Decimal	Binary	Hex	Character	ASCII Mnemonic
0	0000 0000	00	<control-@>	NUL
1	0000 0001	01	<control-A>	SOH
2	0000 0010	02	<control-B>	STX
3	0000 0011	03	<control-C>	ETX
4	0000 0100	04	<control-D>	ROT
5	0000 0101	05	<control-E>	ENQ
6	0000 0110	06	<control-F>	ACK
7	0000 0111	07	<control-G>	BEL
8	0000 1000	08	<control-H>	BS
9	0000 1001	09	<control-I>	HT
10	0000 1010	0A	<control-J>	LF
11	0000 1011	0B	<control-K>	VT
12	0000 1100	0C	<control-L>	FF
13	0000 1101	0D	<control-M>	CR
14	0000 1110	0E	<control-N>	SO
15	0000 1111	0F	<control-O>	SI
16	0001 0000	10	<control-P>	DLE
17	0001 0001	11	<control-Q>	DC1
18	0001 0010	12	<control-R>	DC2
19	0001 0011	13	<control-S>	DC3
20	0001 0100	14	<control-T>	DC4
21	0001 0101	15	<control-U>	NAK
22	0001 0110	16	<control-V>	SYN
23	0001 0111	17	<control-W>	ETC
24	0001 1000	18	<control-X>	CAN
25	0001 1001	19	<control-Y>	EM
26	0001 1010	1A	<control-Z>	SUB
27	0001 1011	1B	<control-[>	ESC

28	0001 1100	1C	<control-\>	FS
29	0001 1101	1D	<control-]>	GS
30	0001 1110	1E	<control-^>	RS
31	0001 1111	1F	<control-rubout>	US
32	0010 0000	20	<space>	SP
33	0010 0001	21	!	
34	0010 0010	22	"	
35	0010 0011	23	#	
36	0010 0100	24	$	
37	0010 0101	25	%	
38	0010 0110	26	&	
39	0010 0111	27	'	
40	0010 1000	28	(	
41	0010 1001	29	)	
42	0010 1010	2A	*	
43	0010 1011	2B	+	
44	0010 1100	2C	,	
45	0010 1101	2D	-	
46	0010 1110	2E	.	
47	0010 1111	2F	/	
48	0011 0000	30	0	
49	0011 0001	31	1	
50	0011 0010	32	2	
51	0011 0011	33	3	
52	0011 0100	34	4	
53	0011 0101	35	5	
54	0011 0110	36	6	
55	0011 0111	37	7	
56	0011 1000	38	8	
57	0011 1001	39	9	
58	0011 1010	3A	:	
59	0011 1011	3B	;	
60	0011 1100	3C	<	
61	0011 1101	3D	=	
62	0011 1110	3E	>	
63	0011 1111	3F	?	
64	0100 0000	40	@	
65	0100 0001	41	A	
66	0100 0010	42	B	
67	0100 0011	43	C	
68	0100 0100	44	D	
69	0100 0101	45	E	
70	0100 0110	46	F	
71	0100 0111	47	G	
72	0100 1000	48	H	
73	0100 1001	49	I	
74	0100 1010	4A	J	

75	0100 1011	4B	K
76	0100 1100	4C	L
77	0100 1101	4D	M
78	0100 1110	4E	N
79	0100 1111	4F	O
80	0101 0000	50	P
81	0101 0001	51	Q
82	0101 0010	52	R
83	0101 0011	53	S
84	0101 0100	54	T
85	0101 0101	55	U
86	0101 0110	56	V
87	0101 0111	57	W
88	0101 1000	58	X
89	0101 1001	59	Y
90	0101 1010	5A	Z
91	0101 1011	5B	[
92	0101 1100	5C	\
93	0101 1101	5D	]
94	0101 1110	5E	^
95	0101 1111	5F	_
96	0110 0000	60	‘
97	0110 0001	61	a
98	0110 0010	62	b
99	0110 0011	63	c
100	0110 0100	64	d
101	0110 0101	65	e
102	0110 0110	66	f
103	0110 0111	67	g
104	0110 1000	68	h
105	0110 1001	69	i
106	0110 1010	6A	j
107	0110 1011	6B	k
108	0110 1100	6C	l
109	0110 1101	6D	m
110	0110 1110	6E	n
111	0110 1111	6F	o
112	0111 0000	70	p
113	0111 0001	71	q
114	0111 0010	72	r
115	0111 0011	73	s
116	0111 0100	74	t
117	0111 0101	75	u
118	0111 0110	76	v
119	0111 0111	77	w
120	0111 1000	78	x
121	0111 1001	79	y
122	0111 1010	7A	z

123	0111 1011	7B	{	
124	0111 1100	7C	|	
125	0111 1101	7D	}	
126	0111 1110	7E	~	
127	0111 1111	7F	<rubout>	DEL

INDEX

[INDEX not reproduced in online version of manual.]

USER REACTION REPORT

To keep this manual and XYBASIC free of bugs and facilitate future improvements, we would appreciate receiving your reactions. Please fill in the appropriate sections below, detach and mail to:

Mark Williams Company
1430 W. Wrightwood Avenue
Chicago, IL 60614

Thank you.

- -

Name :

Company:

Address:

Phone:

Date:

XYBASIC version and revision used:

Did you find any errors in the manual?

Can you suggest any improvements to the manual?

Did you find any bugs in XYBASIC'?

Can you suggest any improvements or enhancements to XYBASIC?

Additional comments: