A BEGINNERS GUIDE TO

PEGASUS BASIC

INTRODUCTION

This book will teach you how to communicate with your Pegasus Computer. You will learn how to speak its language so that by giving it meaningful instructions you can make it do what you want it to do. That's all programming is, by the way.

There are many computer languages. Your Pegasus understands a language called PEGASUS BASIC which is a simplified form of BASIC (BASIC stands for Beginner's All-purpose Symbolic Instruction Code).

BASIC is perhaps the best language for the beginning micro-computer programmer. It is easily learned (as you will soon see) and programs may be developed quickly. For the more experienced programmer BASIC can form the basis of a system whose sophistication may be indefinitely extended.

So let's get started. Get to know your Pegasus. It can do an infinite number of things for you.

Chapter 1

GETTING STARTED

In this chapter we will introduce you to your Pegasus. You will learn how to use your keyboard and how to control the output display on your TV screen.

Connect your computer by referring to the appropriate section in your Computer Operation Manual.

Switch it on and you will be greeted by the following heading on your television screen:

```
AAMBER Pegasus 6809
Technosys Research Laboratories
Basic      1.0
Monitor    1.0
Select one of the above:
```

Press T and your Pegasus will be 'ready' to go.

Do you see the flashing light? This is called the cursor and it indicates to you where on the screen the characters you type in on the keyboard will be displayed.

Try it. Type the following exactly as shown below:

```
PRINT    "HI, I'M YOUR PEGASUS COMPUTER"
```

When you reach the end of the line on the screen, keep on typing. The last part of the message will appear on the next line automatically. (Notice that the screen can display a maximum of 32 characters.)

Now check your line. Is it alright?

If you made a mistake, no problem. Simply press the BACK SPACE key and you will observe the last character you typed will disappear. Press again, and the next will disappear, and so on .....

This is what you should see on the screen:

```
Ready

PRINT  "HI, I'M YOUR PEGASUS COMP
UTER"
```

Now press RETURN. This key tells the computer that you have finished the line. The computer then proceeds to execute it.

Your screen will then display:

```
Ready

PRINT  "HI, I'M YOUR PEGASUS COMP
UTER"

HI, I'M YOUR PEGASUS COMPUTER

Ready
```

As you can see, the computer has obeyed your command and is 'Ready' for more.

Now type:

```
PRINT "2 * 2"
```

and press return. The computer obeys and prints your message:

```
2 * 2
```

How about some answers! Alright, try it without the quotation marks:

```
PRINT 2 * 2  (RETURN)
```

This time the computer prints something different - the answer to the expression 2* 2

Experiment further by typing the following:

```
PRINT     3+4  (RETURN)
PRINT     "3+4"  (RETURN)
PRINT     "3+4 EQUALS", 3+4  (RETURN)
PRINT     8/2, "IS 8/2" (RETURN)
PRINT     "6/2" (RETURN)
PRINT     6/2  (RETURN)
```


This demonstrates that the computer sees everything you type as either strings or numbers. If it is in quotation marks it is a string. If it is not in quotes, it is a number. The computer sees it exactly as it is. The number might be in the form of a numerical expression (e.g. 3+4) in which case the computer reduces it to a single value.

By now it is likely that the computer has printed some unknown messages on your screen. If it hasn't, type the following, deliberately mispelling the word PRINT.

```
PRIINT "HI"  (RETURN)
```

The computer prints: ERROR #4

This indicates that the computer has detected a syntax error. You will have to type the line again properly.

There are other types of errors, too. Try:

```
PRINT 5/0  (RETURN)
The computer prints:
ERROR #8
```

This indicates an impossible division by zero command.

So whenever PEGASUS BASIC detects an error while executing a line it generates an error message. A listing of error numbers and their corresponding meanings is given in Appendix 1.

CHAPTER 2

NUMBERS, VARIABLES AND EXPRESSIONS

Before we go on it is important that we understand the meanings of numbers, variables and expressions.

NUMBERS

PEGASUS BASIC is an integer BASIC, which means that all numbers in it have no fractional part, e.g. 3.7, 4.02 and 3.1415926 are not integers.

Besides this there are two operating modes - signed and unsigned. When you first switch the machine on the computer automatically goes into the signed mode. In this mode integers in the range of -32768 to 32767 are only allowed.

You can change to the unsigned mode by using the USIG statement.

Type: USIG (RETURN)

In this mode integers in the range 0 to 65535 only are allowed. To return to the signed mode use the SIG statement.

Type: SIG (RETURN)

If you input a number outside the allowed range, or the intermediate or final result to a calculation is outside the allowed range, then an error message will be returned.

VARIABLES

In PEGASUS BASIC a variable is represented by a single capital letter (A to Z) which directly corresponds to a location in the computer memory - we call this the name of the variable. The value of the variable is the number stored there.

For example, assign the variable A the value of 13 and the variable B the value of 7 by typing:

```
LET A =  13 (RETURN)
LET B =   7 (RETURN)
```


As LET is used very often in computer programs, the computer will understand you if you leave out the work 'LET' altogether. From now on that is what we will do.

OK. Now have the computer print out your numbers:

```
PRINT A,",",B
```

Notice the use of commas in the print statement. Your computer will remember your assigned values for A and B as long as it is switched on, or until you decide to change them.
Do this by typing:

```
A = 15 (RETURN)
```

Then, when you ask it to print A it will print 15.

EXPRESSIONS

An expression is a combination of one or more numbers, variables or functions joined by operators. You are probably most familiar with the following mathematical operators.

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division

Lets say that we want to divide the sum of 9 and 6 by 3. You might write this as:

```
9 + 6 / 3
```

Now try it on your computer.

Type: PRINT 9 + 6 / 3 (RETURN)

Is this the right answer to the problem? No, it isn't! This is because your computer has first worked out 6 divided by 3 (that's 2) and added this to 9 to give 11.

This demonstrates the way that the computer works out arithmetic problems. The computer looks first at the expression and does multiplication and division first. Then it does addition and subtraction.

So, to get the computer to solve the problem differently, you must use parentheses.

Type it as: PRINT (9 + 6) /3 (RETURN)

That's better. The computer solves the expression in parentheses before doing anything else.

What will your computer print in answer to the following problems.

```
PRINT  12 - (6 - 4) /2
PRINT  12 - 6 - 4 /2
PRINT  (12 - 6 - 4) /2
PRINT  (12 - 6) - 4 /2
PRINT  12 - (6 - 4 /2)
```

Check by typing them out.
Now see what happens if you type in:

PRINT (12 - (6 - 4)) /2 (RETURN)

If the computer sees a problem with more than one set of parentheses it solves the inside parentheses first and moves to the outside parentheses.
In other words, it does it like this:

```
(12 - (6 - 4) /2
      └──────────→6 - 4 = 2

(12 - 2) /2
    └──────────→12 - 2 = 10

19/2
  └──────────→10/2 = 5
```

Can you imagine any problem with integer division? What is 13/5?

Try it: PRINT 13/5 (RETURN)

It gives 2 which is the whole number part of the result. If you want the remainder use the MOD operation.

Try it: PRINT 13 MOD 5

It gives 3. You can use MOD just like you use *,/,+, and -.

There are other classes of operators available in PEGASUS BASIC besides the mathematical operators - we'll look into these in later chapters.

As stated in the definintion you can also include variables in expressions.

Try it by typing: PRINT A/3 + B (RETURN)

(Remember A was 15 and B was 7). This feature is particularly useful in programs that we shall soon see.

CHAPTER 3

INTRODUCTION TO PROGRAMMING

Type: NEW (RETURN)

This is just to erase anything that might be in the computer memory.

Now type this line (don't forget the line number, 10):

10 PRINT "HI, I'M YOUR PEGASUS COMPUTER" (RETURN)

Nothing happened, did it? What you have done is to type your first program.

Next type: RUN (RETURN)

And now you have just run it. Type RUN again - and yes it runs again. Add another two lines to the program.

Type: 20 PRINT "GIVE ME A NUMBER" (RETURN)
30 PRINT "AND I WILL DOUBLE IT" (RETURN)

Then type: LIST (RETURN)

Your computer obeys by listing the program. Your screen should look like this:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"

20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
```

Don't attempt to type in the number because your computer is not ready for it.

Add the line: 40 INPUT T (RETURN)

Add one more line: 50 PRINT "2 TIMES",T," IS ",2*T

Now list again, and your program should look like this:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"
20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
40 INPUT T
50 PRINT "2 TIMES",T," IS ",2*T
```


Now run it. The Input statement prompts you with a question mark. Type in a number (integers only, remember, which the computer will label T) and then (RETURN),

Didn't you do well! This is what you should have got (depending on the number, of course);

```
HI, I'M YOUR PEGASUS COMPUTER
GIVE ME A NUMBER
AND I WILL DOUBLE IT
? 9
2 TIME 9 IS 18
```

Run this program a few more times, inputting different numbers.

Now add another line.

Type: 60 GOTO 10 (RETURN)

And run it.....the program runs over and over again without stopping. That last GOTO statement tells the computer to go back to line 10. This is called a loop, and in this program it will cause it to run perpetually. However, you can get out of it by pressing the BREAK key, then any number and RETURN.

Change line 60 so that it goes to another line number. How do we change a program line? Simply by retyping it, using the same line number.

Type: 60 GOTO 50

Your program listing should then look like:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"
20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
40 INPUT T
50 PRINT "2 TIMES ",T," IS ",2*T
60 GOTO 50
```

Run it.....OK, press the BREAK key when you have seen enough. There is a more desirable way of getting out of the loop. Why not get the computer to politely ask if you want to end it?

Change line 60 as follows: 60 PRINT "DO YOU WANT IT DONE AGAIN?"

And add these lines: 70 R = INKEY : IF R = 0 GOTO 70

80 IF R = 89 GOTO 20

Then run the program ....type your number... then type Y and the program loops back again. If you type anything else (e.g. 'N'), the program stops.

This is what the program looks like:

```
10 PRINT "HI, I'M YOUR PEGASUS CO
MPUTER"
20 PRINT "GIVE ME A NUMBER"
30 PRINT "AND I WILL DOUBLE IT"
40 INPUT T
50 PRINT "2 TIMES ",T," IS ",2*T
60 PRINT "DO YOU WANT IT DONE AGAIN?"
70 R = INKEY: IF R = 0 GOTO 70
80 IF R = 89 GOTO 20
```

What are these new lines? Line 60 is simply a printed question. Line 70 is in fact two lines, the two statements being separated by the colon ':'. The first part assigns the ASCII equivalent of the key depressed on the keyboard to the variable R. (ASCII is the American Standards Code for Information Interchange). If no key is pressed then R is assigned 0. The second part of the line tests for this condition and loops back to INKEY if it is true. However, as soon as a key is depressed it gets out of the loop and proceeds to the next line...

Line 80 tells the computer to go to Line 20, IF, (and only if) the Y key (That's ASCII 89) has been depressed. If not the program ends as there are no more lines after this.

This chapter has covered a lot of important concepts of PEGASUS BASIC. Don't worry if some of these things are not absolutely clear at this stage. Experiment with your computer, and above all, enjoy it.

CHAPTER 4

MORE PROGRAMMING

In this chapter we will practise using functions and statements in PEGASUS BASIC.

Type this:

```
10 FOR X = 1 TO 10
20 PRINT "X =", X
30 NEXT X
40 PRINT "FINISHED"
```

Run the program. See how it has printed X for X = 1 TO 10. Now replace line 10 with the following:

```
10 FOR X = 5 TO 8
```

And run again. Let's look at the program listing.

```
10 FOR X = 5 TO 8
20 PRINT "X =", X
30 NEXT X
40 PRINT "FINISHED"
```

It's clear that line 10 determines that starting and ending values of the variable X. Line 30 tells the computer to get the next number - the next X - and to jump back to the line following the FOR ... TO ... line. (i.e. line 20) until it reaches the last number. At this stage it goes straight on to execute the final statement. We can further investigate the path of program execution by using the TRON statement. Try it.

Type: TRON (RETURN)

Now run the program again. This statement has turned on a trace, which provides a line number listing for statements as they are executed. The trace should look like this:

```
<10>  <20>  X = 5
<30>  <20>  X = 6
<30>  <20>  X = 7
<30>  <20>  X = 8
<30>  <40>  FINISHED
```

See how the program keeps jumping from line 30 to line 20 until it eventually goes from line 30 to line 40 and stops. To turn the trace off, type: TROFF (RETURN)

If you like, run your program again to see if the race has gone.

An extra feature of the FOR...TO... statement is that you can specify the actual STEP size. Change line 10 to read:

```
10 FOR X = 2 TO 10 STEP 2
```

And run the program. See how X goes from 2 to 10 in steps of 2. Before, when we didn't specify the step size, it assumed STEP 1. What will happen if line 10 is replaced with:

```
10 FOR X = 3 TO STEP 3
```

Try it... and see that it loops back only for X 10

How about: 10 FOR X = 10 TO 1 STEP - 1

Yes, it counts backwards too.

Now try a new program - that's right, type NEW and (RETURN) - the type:

```
10 FOR X = 1 TO 3
20 PRINT "X =", X
30 FOR Y = 1 TO 2
40 PRINT "Y = ", Y
50 NEXT Y
60 NEXT X
```

Run it ... This is what you should get:

```
X = 1
Y = 1
Y = 2
X = 2
Y = 1
Y = 2
X = 3
Y = 1
Y = 2
```

Notice how it loops within another loop. Programmers call this a 'nested' loop.

Now for something completely different. Type in this new program:

```
10 S = RND /26
20 PRINT "GUESS THE NUMBER"
30 INPUT G
40 IF G = S THEN GOTO 70
```



```
50 PRINT "NO, TRY AGAIN"
60 GOTO 30
70 PRINT "YES, THAT'S IT"
```

And run it... guess numbers between 0 and 9 inclusive (the division by 26 in line 10 gives us this range).

The new statement type encountered here is the IF ... THEN conditional statement. The statement tests the expression G = 5 and IF false will skip immediately to the next line; but IF that statement is true THEN it executes the next statement GOTO 70.

This condition is often the result of a relational operation. In BASIC these are;

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

These are often combined with logical operators AND, OR, NOT to perform quite complex tests, here's an example:

```
600 IF A = 0 OR (C 127 AND D   0)  GOTO 100
```

This will cause a branch to line 100 if A is equal to 0 or if both C is less than 127 and D is not equal to zero.

This type of expression essentially evaluates to 0 for false and -1 for true. Besides being used from true/false evaluation, logical operators can operate on binary numbers.

For example, type: PRINT 6 and 7 (RETURN)

This gives decimal 6 which is 0110 ANDed with 0111

So far we have been looking at relatively short programs. Before long you will be so proficient with your Pegasus that you will be writing quite long and complex programs.

We'll now look at some expressions which will help us keep things in order. Type and run the following program.

```
10 PRINT " EXECTING THE MAIN PROGRAM"
20 GOSUB 400
30 PRING "NOW, BACK IN MAIN PROGRAM"
40 END
```

```
400 PRINT "EXECUTING THE SUBROUTINE"
410 RETURN
```

Line 20 tells the computer to go to the subroutine beginning at line 400. RETURN tells the computer to continue execution with the line following the GOSUB expression. The END expression is necessary to separate the main program from the subroutine.

Subroutines are written for operations that are frequently required. They result in economy of effort when it comes to writing programs.

One final point - you can use the REM statement to place remarks and comments throughout your program. Anything following the REM statement is ignored. These remarks are often placed at different points in a program, particularly at the beginning of subroutines, to explain how unclear or complicated sections of the program work.

Here is a final program that illustrates these points. Try it.

```
10 REM THIS PROGRAM RAISES A
20 REM NUMBER TO AN EXPONENT
30 INPUT "NUMBER"N
40 INPUT "EXPONENT"E
50 GOSUB 1000
60 PRINT:PRINT N, " EXPONENT ",E" IS ",A
70 END
80 REM ------------//------------
1000 REM THIS SUBROUTINE DOES
1010 REM THE ACTUAL EXPONENTIATION
1015 IF E=0 THEN A=1:RETURN
1020 A=1
1030 FOR X-1 TO E
1040 A=A*N
1050 NEXT X
1070 RETURN
```

By now you should feel that you are in complete control of your Pegasus. Try writing some programs of your own.

Good luck, and have fun!

A GENTLE INTRODUCTION TO PEGASUS BASIC

The BASIC Language

BASIC is the most commonly used computer language in the world today. The word BASIC is an acronym, standing for Beginners All-purpose Symbolic Instruction Code.

BASIC is a computer program that was originally developed at Dartmouth College in the U.S. as a means of teaching students the principles of computer fundamentals, as well as making it easier to write computer programs. BASIC itself is usually written in machine code assembler, although higher-level languages have been used.

Bells and Whistles

Hundreds of BASIC interpreters (i.e. programs that will accept and interpret a program written in BASIC) have been written since the first version, and each one is usually unique in its features and limitations. Theoretically anyone with enough knowledge and time can write a BASIC interpreter, although not many people so do. However, when they do, each likes to add a personal touch, in the form of special features, and this is known as adding Bells and Whilstles. (We have not stinted in this tradition). Thus, although BASIC is common, there are many dialects.

Where Do I Start?

At the beginning, of course! We'll look at the idea that a computer program is like a recipe. For example, let's make a milkshake:

```
Fetch container
Fetch milk
Pour milk into container
Fetch flavoured powder
Add powder to milk in container
Pick up container
Shake!
Oops!
Put down container
Clean up mess
Put lid on container
Shake!
Take lid off
Drink milkshake
End of recipe
```

A trivial, almost useless, example. Each line or statement is a command or instruction (apart from 'Oops!' which is a comment or perhaps an invective). The statements were executed sequentially, starting from the top. Note that each statement leaves out a very great amount of detail - like what sort of container was used, where the milk came from, what flavoured powder was used, even whether it was enjoyed or not!

Computer programs are quite like this in their detail - a great deal is implicit or assumed. Computer programs are much simpler, however, in the actions that they describe, in that the tasks a computer performs are (usually) logical and straightforward - unlike the real world of gravity and spilt milk.

Using Numbers

BASIC, like many other computer languages, is designed to work with numbers. Usual operations in BASIC are addition, subtraction, multiplication and division (+,-,*,/). There are two ways that numbers are used in BASIC - constants and variables.

A constant has a value which it keeps for as long as the program runs. Typical constants are 7, 24, 0, -32768, 2000. Variables are symbols for memory cells that may contain numbers. In Pegasus BASIC we use the letters A through to Z to represent these variables. Thus we can refer to a variable in a computer program without having to know its value. When a computer program is first RUN, all variables, A to Z, will have a value of zero.

Number Size

Pegasus BASIC is an integer BASIC, which means that all numbers in it have no fractional part. E.g. 3.7, 4.02 and 3.1415926 are not integers. Further, the Pegasus has 16 bit signed two's complement and 16 bit unsigned numbers, which means that for signed numbers you are limited to -32768 to 32767, while unsigned integers have a range of 0 to 65535. Any outside this range will cause an error.

Number Representation

Numbers are stored internally in binary, but to make it easier for people to handle them, we have provided two forms of integer format: numbers may be output (printed) in decimal or hexadecimal (base 16). For instance, if variable A contains 19, then we can print the two forms thus:


```
  PRINT A," ",HEX(A)
```

which will print out:

```
  19 13
```

For inputting numbers, they must always be in decimal, but may be signed or unsigned. Hexadecimal numbers may be used directly in a program by preceeding them with a dollar sign ($), e.g.:

```
  PRING $13
```

will print:

```
  19
```

on your television screen. Both signed and unsigned numbers may be used and may be selected with two statements, SIG and USIG, which stand for signed and unsigned. Signed numbers have a range of -32768 to +32767, while unsigned are in the range of 0 to 65535. Note that an unsigned number greater than 32767 will be printed as a negative number if the program switches back to the signed mode.

Arithmetic

In BASIC,arithmetic may be done with expressions. An expression is a group of tokens, each of which has a definite value associated with it, that is built up using a set of possible operators, and is solved as an algebraeic expression that returns a single numeric value. Now that we have confused you, let's clear it up with some examples:

`A*3+7*R`	
`(3+Q)-(21/L+(8*I))`	Note that the parentheses must match
`2+2`	
`1`	yes, a number is an expression too
`$4F OR 51`	note the Boolean operator
`ABS(-R)`	functions are expressions too

A variable or constant by itself may also be considered an expression, and expressions may consist of other expressions, as long as they are logically organised, and the number of left and right parentheses match correctly. Unlike some BASICs, nearly any complexity of expressions can be used.

Operators

Constants, functions, variables and expressions may be related by operators to form a new expression. All the operators work with 16 bit integers, and return 16 bit integers as results.

+	simple addition
-	subtraction
*	multiplication
/	divison
MOD	modulus, same as taking remainder after a division instead of the quotient. e.g. 7 MOD 6 yeilds 1.
+	unary plus, e.g. +7 by itself
-	unary minus, e.g. - 12
NOT	returns one's complement, e.g. NOT $F012 returns $0FED.
AND	logical AND, may also be used as Boolean connector
OR	logical OR, similar to AND

Note that expressions are no good unless you do something with them, using one of the statements available. The simplest statement to use is the assignment statement, LET. This is used for assigning values to variables, e.g.

```
LET Q=I+9          The '=' means 'is assigned'
L=17*(8+T) MOD 15  The LET is optional
I=I+1
```

The last statement is of particular interest since it illustrates how a variable is fetched, incremented, and then stored back in the same memory cell again. The '=' sign does not mean 'equals', but means is assigned the value of '. Note that 3=A or 3=7 are illegal and will give an error message. Spaces may be used freely in expressions, however, they may not be imbedded inside function or statement names.

A quick way of using your Pegasus for math is to use the PRINT statement in conjunction with an expression. Remember that the question mark, '?', is shorthand for PRINT. For instance, ? 7*8 gives 56. When expressions are evaluated, they are executed in an order defined by the operator precedence. This means that values that are conjoined by certain operators will be executed before others in an expression. The precedence order is as follows:

1st	constants, variables
2nd	functions (includes special @ function)
3rd	unary - or +, NOT
4th	operators * / MOD AND
5th	operators + - OR

The order of evaluation may be changed by using parentheses. Some examples are given for your enjoyment:


```
3+4 * 2+5 resolves to 16
(3+4) * (2+5)evaluates to 49
```

A special class of operator, the relational operator, is covered in the section on Booleans.

6th	relational operators (lowest precedence)

Booleans

A Boolean expression is similar to an arithmetic expression, apart from the use of the realtional operators. Any relation evaluates to 0 for false and non-zero for true. The most usual non-zero value found will be -1 (hex FFFF). The relational operators are:

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

Boolean expressions may be mixed with arithmetic expressions, leading to results like:

```
A=B=C+1
```

Boolean expressions may be used with the IF statement, e.g.

```
IF Q=7 THEN END
IF T THEN GOTO L
```

Here, L is treated as an unsigned line number that the program will GOTO if T is non-zero.

Statements and the Editor

Program lines in BASIC are usally organised in a strictly sequential manner, using line numbers in the range of 1 to 65535. A program will consist of a series of lines, where each line consists of one or more statements (separated by the colon, ':') and is executed sequentially, except where a special statement will change the flow of program logic. Here is a sample program that will print out the integers between 1 and 10.

```
10  I=0 : REM I IS ASSIGNED A VALUE OF ZERO
20  I=I+1 : REM I IS INCREMENTED
30  PRINT I : REM PRINT OUT THE VALUE CONTAINED
    IN I
40  IF I=10 THEN STOP : REM STOP WHEN I REACHES
    10
50  GOTO 20
```

Follow the program through by hand, or better still, try it on your Pegasus. When typing the program in, terminate each line with the RETURN key, and correct any typing mistakes by using the BACK SPACE key. If you notice a mistake on a line that you have already left, simply re-type the correct version (with the same line number) and the old line will be automatically replaced. To remove a line entirely type the line number by itself, followed by the RETURN key.

Experiment with your own programs to print out different sorts of number sequences, until you are fully satisfied with the material covered so far. If you have trouble stopping a program once you have started it, tap the break key.

Summary of Statements

PRINT expressions, string constants

This statement will evaluate and print results of expressions, as well as printing string constants.

A string constant is a collection of characters delimited by double quotes, e.g.

"FRED NURKE WAS HERE"

"THAT'S all FOLKS"

Expressions will be evaluated, and the results printed, with no leading or trailing spaces. String constants and expressions MUST be separated by commas. Upon completion of the print statement, the cursor will move to the beginning of the next line, unless the PRINT statement is terminated with a comma. The cursor may be positioned to anywhere on the screen at any stage in the PRINT by using the form [x,y]. For example, PRINT [10,2], "HELLO", will move the cursor to column 10, line 2, and print "HELLO", leaving the cursor immediately after the 'O'. The vertical position 'y' is optional, but if it is included then it must be separated from the 'x' column position by a comma. There are three functions that may only be used with the PRINT statement, since all of them produce some sort of output, without returning a value. These output functions are detailed below:

CHR (expression)

This will output the ASCII character that is represented by the result of the expression. The result is forced into the range of 0 to 255 (decimal), or $00 to $FF (hex). If the number is greater than 127, then the character will be inverted. Note that characters in the range of 0 to 31 and 128 to 159 will not print, but will cause one of the control functions to be executed.

HEX (expression)

The expression is evaluated, range 0 to 255, and the appropriate hex number is output, range $00 to $FF.

RAW (expression)

This function is very similar to CHR, except that values in the range of 0 to 31 and 128 to 159 will have a special character output, without executing the appropriate control function.

Note that all functions that require an expression in brackets, must not have a space before the left parentheses. (The RAW function is associated with the RAWON and RAWOFF statements, covered later in this document.) Examples of their use are given below, for you to try on your Pegasus.

```
PRINT "Print a hex number:",HEX(19), CHR(10),
      RAW(0)
PRINT CHR($46),CHR($52),RAW($45),CHR($44)
PRINT "There are ",Q," beans in the box."
PRINT CHR(12)  :  REM Clear screen
PRINT RAW(12)  :  REM Output Greek letter 'nu'
```


LIST starting line, ending line

Program lines may be listed out, either as individual lines, subranges of lines, or the entire program. The expressions are both optional, and are unsigned numbers always. If a line is specified that is not in the program, then the nearest one to it will be used. This is the only case in which such leniency is tolerated. If you try to force BASIC to use the 'nearest' line number in other statements, then a small quantity of plastic explosives attached to your Pegasus will be detonated, removing your typing fingers.
YOU HAVE BEEN WARNED.
Note that the starting and ending lines may be expressions, and the LIST statement may be part of a BASIC program.

RUN expression

The RUN statement will initialize all variables, then start program execution at the line number specified. If no number is specified, the program will start at the beginning.

INPUT "string constant" input list

This statement, unlike many others, can only be executed with a line number as part of a program. Its purpose is to request numbers from the user for input to the program. The string constant (if specified) will be printed out as a prompt to the user before input is requested, and must not be followed by a comma. When each input expression (yes, expressions can be input) is typed, it must be terminated with a RETURN key. Only one string constant may be specified, and if used it must be immediately after the INPUT.

FOR variable = start value TO end value STEP step-size. This is the standard BASIC looping statement. This will cause all statements between the FOR and its appropriate NEXT to be executed repeatedly until the variable's value reaches or exceeds the end value. Note that the step size may be positive or negative. If the step is not given, it will default to one.

NEXT variable name
Terminating statement for FOR loops.

GOTO expression
The expression will be evaluated to an unsigned 16 bit integer, and if a line is found with a matching line number, then that line will be executed next.

GOSUB expression
The expression will be evaluated, and the subroutine which starts with the matching line number will be called, returning to after the GOSUB statement when it reaches and executes the RETURN statement. GOSUBs may be nested to any depth, depending upon free ram space for the stack.

RETURN

This statement indicates the logical end of a BASIC subroutine.

EXIT

The EXIT statement will return you back to the Pegasus Menu selection mode.

NEW

This statement will zero all variables, as well as deleting all program lines.

STOP

The STOP statement will cause program execution to terminate, returning to the line edit mode. Execution may be continued with the CONT statement, as long as the program has not been changed. Any other immediate mode statement may be executed however.

END

The END is similar to the STOP statement, except that the CONT statement will not continue program execution after an END

CONT

The CONT will cause program execution to continue, as defined by the STOP and END statements.

REM

Any user remarks may appear after this statement, since they will be ignored by the BASIC interpreter. The REMark is terminated by the end of line or a colon.

LET variable name = expression
The assignment operation assigns the value of an expression to the named variable. Only variables and the special function '@' may be used on the left side of the '=' sign.

IF expression THEN statement or expression
The IF statement will evaluate the first expression, and if it is zero, then the remainder of the statement will be skipped, going to the next line. Upon a true state, then the part after the THEN will be executed if it is a statement, or if it is an expression, then it will be evaluated, and a GOTO will be executed.

TRON

This statement will bring the trace mode into effect, whereby each line number will be printed out as the line is executed, following the flow of program execution as the RUN proceeds.

TROFF

This statement will turn the trace mode off.

SIG

This forces the system to accept and print only signed numbers, in the range of -32768 to +32767. A point to note here is this example:
PRINT HEX($B010/256) will yield B1, instead of the expected value of B0. This is because although the hex number is unsigned, SIGned mode is in effect, and must be disabled using USIG before the correct result may be achieved.

USIG

The system can accept unsigned integers, in the range of 0 to 65535, for input, output, and arithmetic expressions. Note that this mode is checked when determining whether the result of an expression is outside its range.

SAVE

BASIC programs are saved on cassette tape, with a filename that you may specify (8 characters only). The BLUE tagged lead goes into the MIC jack, while the YELLOW lead goes into the EAR jack.

LOAD

Previously SAVEd programs may be loaded from cassette tape. The filename and load area will be printed.

POKE expression , expression
The first value resolves as an unsigned 16 bit address, which gives the location to poke the second value into.

LINES expression
This statement controls the number of lines displayed on the screen. The expression must resolve to a number in the range of 1 to 16, or an error will stop execution of the program. Reducing the number of lines displayed has the result of speeding up program execution proportionally.

RAWON

This statement will turn on the RAWMODE flag. This means that any control code that is echoed to the screen through the normal PRINT routine, or through typing in lines, will cause a special character from the character generator ROM to be printed, without executing the control function.

RAWOFF

Turns RAWMODE off.

CLS

This statement, when executed, will clear the video display screen, and move the cursor to the top left hand corner.

BASIC Functions

Pegasus BASIC has a number of functions, each of which may be used in expressions, (apart from the ones specified in the PRINT statement description). Note that there must be no spaces between the function name and the left parenthesis.

ABS(expression)
: Takes absolute value of the argument.

PEEK(expression)
: Returns byte at address given by expression.

FREE
: Returns number of free bytes available in system RAM.

RND
: Returns pseudo-random number in the range 0 to 255.

USR(expression)
: Calls a machine language routine subroutine in memory at the address specified by the expression - the function value returned reflects the state of the X register, and may be data or an address pointing at more data.

@(expression)
: Special function that implements a one dimensional integer array that utilises all available RAM space. The function may be used anywhere that a variable name is used, including in assignment statements. Unlike variables, the @ array is not cleared by the NEW statement. The array index is

unsigned, starts at zero, and its size is FREE/2-1

INKEY

This function will scan the keyboard to see if a key has been pressed - if one has, then its ASCII value in the range of 1 to 127 will be returned, else if no key has been pressed then zero will be returned.

Information for Experts

The variables, since they are in fixed locations in RAM,(in the 4K system only), can be accessed by machine code subroutines by referencing directly their addresses. There are 26 variables, and they start at $B03C.

Nearly all tokens in BASIC have a shorthand form, for instances, '?' means PRINT, and 'e' means PEEK(. Try finding out what the rest are - this information will be published in the newsletter.

When a program is executing on the Pegasus, it may be stopped in its tracks by using the BREAK key on the keyboard. This is functionally equivalent to the program encountering a STOP statement.

Output that is being sent to the screen may be paused by use of the ESCAPE key, on the upper left of the keyboard. Tapping once will stop, tapping again will start.

If inverse video characters are required inside strings, they may be effected by tapping the blank key on the extreme lower right of the keyboard. Tapping the key again will remove the inverted state. Note that only characters inside double quotes will remain inverted. Inverted characters may also be generated by setting the most significant bit of the byte for the ASCII character (ASCII equivalent greater than 128).

RAWMODE may be set and resent by tapping the blank key on lower right of keyboard, second one in. When in effect, any control characters typed will appear as a special printing character, without the appropriate control function being executed.

Note that the RETURN key, being a control code, will not work as it should until the RAWMODE flag is turned off by tapping the second blank key again. This feature allows you to insert control codes into strings, and then the output of those control codes as characters or functions may be governed by use of the BASIC statements RAWON and RAWOFF.

BASIC may be re-entered from the monitor after using the PANIC button by jumping to $0B offset from its start.

Basic Error Numbers and Messages

Whenever a syntax or eduction error occurs, then an error number will be printed out, each number matching to one of the error messages given below:

(1) Out of Memory
This means that there is insufficient RAM space between the program end and the stack to perform the last operation.

(2) Invalid Line Number
A line number was specified that either does not exist, or is illegal

(3) Next Without For
A NEXT statement was found without the appropriate FOR statement

(4) Syntax Error
This is a general error that occurs whenever there is incorrect syntax in a program line

(5) Return Without Gosub
A RETURN statement was executed, but the system did not find a GOSUB to return to

(6) Immediate Mode Illegal
A statement was executed in immediate mode that is illegal for that mode

(7) Overflow Error
The results of an arithmetic operation exceed the current range specified

(8) Divide by Zero
An attempt was made to divide a number by zero.

(9) Screen length Error
The LINES statement must have an argument in the range of 1 to 16 only.

BASIC STRING INPUT/OUTPUT

Subroutine 1000 stores a string of 32 characters beginning at $B8ØØ + 32 * N, where N is an integer from 0 to 31 if 1K of RAM is used from $B8ØØ to $BBFF.

Line 1020 — I is the memory location for string element of string number N

Line 1030 — Inkey checks for a key pressed

Line 1040 — backspaces and deletes previous character from RAM

Line 1050 — returns from subroutine

Line 1060 — prints the character and stores it in location S + I

Line 1070 — puts a null at the end of the string

Subroutine 2000 outputs the string for a given N

Line 2010 — -PEEK's the element of the string

2020 — -checks for the end null

2030 — -prints the string element and increment S for the next one.

To input or output a string, specify the string number and then use the appropriate subroutine. The input routine will wait for Keyboard input, ending with a RETURN.

```
10 REM SAMPLE STRING I/O
20 REM FOR BASIC
30
50 REM INPUT STRING
100 N=1:GOSUB1000:END
150 REM OUTPUT STRING
200 N=1:GOSUB2000:END
500
1000 REM STRING INPUT
1020 I=$B800+32*N:S=1
1030 K=INKEY:IF K=0 THEN 1030
1040 IF K=8 THEN PRINTCHR(8),:S=S-1:POKE(S+I),0:GOTO1030
1050 IF K=13 THEN RETURN
1060 PRINTCHR(K),:POKE(S+I),K:IF S<33 THEN S=S+1:GOTO1030
1070 POKE(S+I),0:RETURN
1100
2000 REM STRING OUTPUT
2010 I=$B800+32*N:FOR S=1 TO 32:C=PEEK(S+I)
2020 IF C=0 THEN S=32:NEXT S:RETURN
2030 PRINTCHR(C),:NEXT S:RETURN
9999 END
```