CONTENTS

TCT BASIC

TAPE DETAILS.

SIDE A: TCT BASIC
110 baud binary format with
loader in PIPBUG format.

SIDE B: TCT BASIC OPTIONS
1. RaNDom
110 baud PIPBUG format.
2. SINe
110 baud pipbug format.

SYNTACTIC DESCRIPTIONS AND BACKUS-NAUR FORM

The 'Backus-Naur' form of syntactic description is used throughout this manual to define the legal construction of statements.

In Backus-Naur form syntactic constructs are denoted by English words enclosed in '<' and '>' signs. These words are chosen to suggest the nature and meaning of the construct which they represent. For example, '<expression>' is used often and denotes any legal combination of arithmetic variables and operators (addition, multiplication, etc).

The vertical slash '/' is used to separate mutually exclusive possibilities and may be read as 'or'. For example

<expression>::=(<expression>) / <expression>

means that an expression may be another expression enclosed in brackets or just another expression. (This form of recursive definition is common in the description of 'high level languages'.)

Finally there exists the symbol '::=' which may be read as 'is defined as being', or just 'is'. An example is given above.

These three basic symbols are used to describe the syntax of statements and atatement construction.

Those symbols which appear on their own (i.e. are not enclosed by '<' and '>') are actual characters and symbols which appear in the text which is being described. The simplest example of this is the Backus-Naur of the 'RETURN statement'

<return statement>::= RETURN

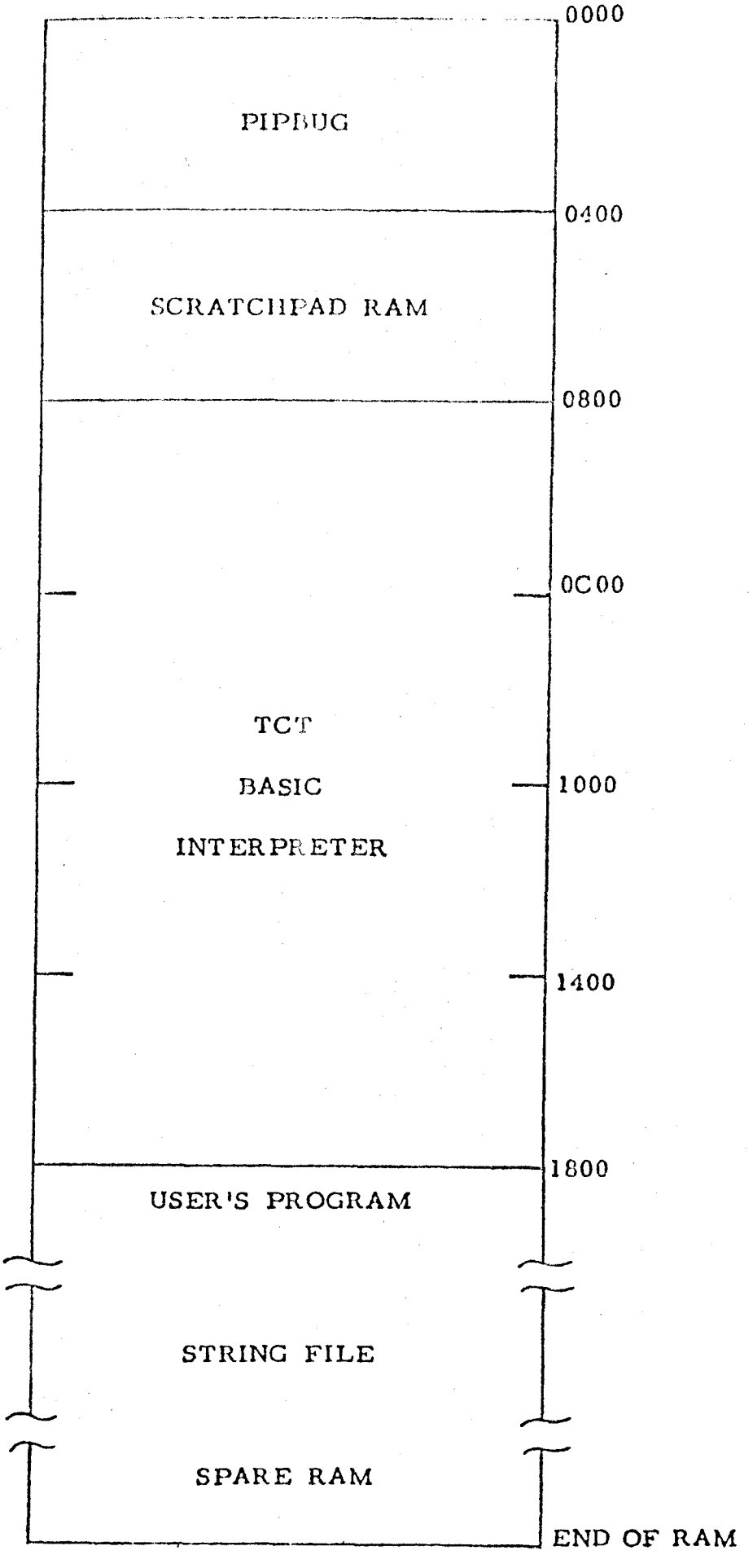
ENTERING COMMANDS AND TEXT

All input to the Basic interpreter is made through an inbuilt single line text editor. Whilst the user is in communication with the editor certain control characters are reserved for special functions.
These characters are:

CR	Used to delimit the current line of input.
BS	Deletes one character from the end of the current line.
DEL	Deletes the entire line which has been input up to that point and re-prompts.

When CR is pressed control is passed to the rest of the interpreter. At this stage there are two possibilities, that the line is entered into the users program file, or it is interpreted as a command and executed immediately. The criteria which determines which of these is performed is whether or not a number preceeds the statements on a line.

In general if an input line starts with a number it will be interpreted as a line to be entered into the user's program file at the appropriate point. If an input line has no recognisable number preceeding it then it will be executed immediately, irrespective of whether it is a command or an executable statement.
To delete a line from the program file the line's number should be typed followed by a CR.


0000
PIPBUG
0400
SCRATCHPAD RAM
0800
0C00
TCT
BASIC
INTERPRETER
1000
1400
1800
USER'S PROGRAM
STRING FILE
SPARE RAM
END OF RAM

STRING HANDLING CAPABILITIES

The manner in which strings are handled in TCT BASIC is somewhat different from that of most other BASICs and so will be discussed in some detail.

In most implementations of the BASIC language strings are identified by a letter of the alphabet with either a leading or trailing '$' sign. This construct has severe limitations in that there is only a very limited number of strings available, and more importantly, strings cannot be referenced by some calculation, which also limits the number of stings which can be effectively handled.

To overcome this limitation it was decided to identify strings with four digit numbers, and allow expressions to be used to determine these numbers in all string handling operations. These string identifying numbers obey all the rules associated with line numbers, i.e. they must be greater than zero and less than 10000. Truncation of string identifying numbers is performed in all operations.

In this manner strings may be handled in volume and with ease, yet there are of course limitations imposed by the physical size of the BASIC interpreter. These points are noted below.

1) Strings may not be 'merged' or in any way 'put together' or 'taken apart'. I.e. constructs of the form $1=$2+$3 are not allowed. (This is perhaps the only major limitation.)

2) The only comparisons which may be made between strings are those of equality or inequality, and these comparisons must be made in terms of string identifyers, not literals enclosed in quote marks.

3) Due to the large number of possible strings it is of course necessary to release memory space occupied by a null string (i.e. an empty string, which all strings are before they are assigned a value) completely. For this reason it is impossible to operate on a null string. However it is possible to detect that a string is empty by the provision that it returns a true value for all comparisons. This is a useful facility when dealing with user input where the response to a request is often just a carriage return.

Strings are stored in memory from the end of the program text onwards through available RAM. It is the user's responsibility to ensure that the length of the file does not exceed the machines memory limit.

THE COMMAND 'DUMP'

The DUMP command is the means by which an existing BASIC program may be recorded for latter loading and use.

The form of the DUMP statement is the word DUMP followed by an optional line number.

The DUMP command is fundamentaly the same as the LIST command except that no line feed is performed at the end of each listed line and a delay is inserted instead. This allows a program listed in this manner to simply be played back from the tape recorder and inserted as text in the program file.

The proceedure for using this command is:

1) Type the word 'DUMP' followed by a line number if desired.
2) Set the tape recorder to 'RECORD'.
3) Type carriage return.
4) when dumping is completed stop the tape recorder.

To reload a recorded program:

Merely play the tape back whilst in command mode. The interpreter will initialy respond with a syntax error, this should however be ignored and is only due to an unavoidable 'hash' on the tape.

We are sorry to say that some tapes posses errors effecting the execution of the DUMP command, if you have one of these tapes the fault may be remedied by performing the the following operations in PIPBUG.

```
*A17D3
17D3  3F  77
17D4  02  10
17D5  86  06
17D6  E4  08
17D7  0D  12
17D8  14  1A
17D9  3F  7D
17DA  02  3F
17DB  B4  02
17DC  77  AD
17DD  10  C1
17DE  02  3F
17DF  75  17
17E0  10  F7
17E1  17  3F
*
```

THE COMMAND 'NEW'

The typing of the word NEW followed by a carriage return eliminates all trace of any program file which may have been present, resets all internal stacks and pointers and clears all variables including strings.

If the BASIC interpreter is entered at the HEX address 0800 then a NEW command is executed automaticaly. However if it is entered at 080A then the program file will be unaltered (although variables will be cleared).

If the NEW command is ever entered accidentally your program file may be recovered by exiting the BASIC interpreter and changeing the locations starting at HEX 1801 to:

1801	The high order BCD code of the first line number.
1802	The low order BCD code of the first line number.
1803	The ASCII code of the first character of your text.
1804	The ASCII code of the second character of your text.

Then entering the BASIC interpreter at 080A.

THE COMMAND 'CLEAR'

The CLEAR command sets all variables to zero, and eliminates all strings.

It is implemented by typing the word CLEAR followed by a carriage return.

THE COMMAND 'LIST'

The LIST command is the means by which the program file may be inspected, in part or whole.

The form of the LIST command is the word LIST followed by a carriage return, or the word LIST followed by a line number. The first form will start listing the program file from the first line, while the second form will start from that line with a line number greater than or equal to the specified number.

To suspend the listing process hold down the 'break' key, if the 'break' key is not implemented or connected on your keyboard then hold down 'rept' (repeat) and 'space', listing should cease within a few lines.

THE COMMAND 'RUN'

The RUN command causes the interpreter to enter 'execution mode' and begin execution of the program stored in the program file at that line which posesses the lowest line number.

The RUN command is implemented by typing RUN followed by a carriage return.

After a RUN command has been executed the user's BASIC program will begin execution and continue until a STOP command or an error is encounted, or the 'break' key is depressed.

If it is desired to start program execution at some point other than the first line then a GOTO statement may be used. This will automaticaly put the interpreter in 'execution mode' and begin execution from the line specified. (c.f. 'THE GOTO STATEMENT')

THE COMMAND 'SIZE'

The SIZE command returns the HEX values of the start of the program file and the end for the user to check on the available RAM left.
(NOTE: This includes that area of RAM tacken up with strings.)

The form of the SIZE statement is simply the word SIZE followed by a carriage return.

The response is:

XXXX
YYYY

Where XXXX is the address of the first byte of your file
and YYYY is the address of the last byte of your file

(On this version of TCT BASIC XXXX will of course always be 1800.)

THE 'LET' OR ASSIGNMENT STATEMENT

The LET statement is the basic operational statement of the BASIC langusge. It is by means of this statement that data is transferred and transformed between variables, whether they be numeric variables or literal strings.

The fundamental form of the LET statement is the word LET (optional), followed by either a letter (for a numeric variable), or a '$' followed by any legal numeric expression (for a literal or 'string' variable.) This variable identifier is followed by an '=' , which is in turn followed by any legal numeric expression or a '$' followed by any legal numeric expression or a simple string.

This construct, when executed, will place the value derived from the right hand side into the variable specified on the left hand side. To make this aspect clearer it is best read 'is assigned the value of' in place of the '=' sign.

NOTE 1) The types of the expressions or variables on each side of the '=' should agree: that is if the right hand side returns a numeric value then the left hand side should be a numeric variable, and if the right hand side is a string, then the left hand side should be a string identifier (i.e. have '$' prefix).

2) The initial LET keyword is entirely optional and may be deleted if desired.

3) The values of the variables on the right of the '=' remain unchanged unless they apear on the left of the statement.

The syntax of the LET statement may be sumarised as follows:

<let statement> ::= LET <assignement statement> / <assignment statement>
<assignement statement> ::= <variable> = <expression> / <string identifier> = <string>
<string> ::= <simple string> / <string identifier>

EXAMPLES OF THE LET STATEMENT

```
LET A=127*3
C=2*PI*R*R
$1="YES"
$A*5=$INT(B/27)
```

The following are illegal uses of the LET statement due to mixed types:

```
A="YES"
LET $1=2*PI*R*R
```

THE 'GOTO' STATEMENT

The GOTO statement is the means by which program flow is broken and resumed at another point.

The form of the GOTO statement is GOTO <expression>. Where the value returned by the expression must be a number greater than zero and less than 10000 (truncation is automatic).

When executed, program flow will be resumed at the begining of the line which has a number corresponding to the value of the expression, if no such line exists a 'NOGO ERROR' will result.

NOTE 1) A space may be inserted between the GO and the TO, that is the GOTO statement may read GO TO<expression>.

2) The value of the expression need not be an integer, it will be truncated automatically.

The syntax of the GOTO statement is,

<goto statement>::= GOTO <expression>

EXAMPLES OF THE 'GOTO' STATEMENT

```
GOTO  120
GOTO A+B*10
GO TO 970
```

EXECUTION MODE

The following pages describe those commands which may be executed within a program. Most of these commands or 'statements' may also be executed in command mode. However no commands are executable in execution mode (That is those commands which appear on the preceeding pages may not appear in a program.)

There are certain executable statements which may not be executed in command mode, these are:

INPUT
DO
UNTIL
FOR
NEXT

THE 'IF' STATEMENT

The IF statement is the mechanism by which desicions are made within a BASIC program, and different action taken depending on some condition.

The fundamental form of the IF statement is the word IF followed by either a string identifier or an expression. This is followed by a 'relational operator' which is in turn followed by another string identifier or expression. This is followed by any collection of statements on the same line.

If expressions were used in the IF statement, then upon execution these will be evaluated to two single numbers and compared in relation to the given 'relational operator', or if strings were used they will be compared letter for letter.
If the resultant 'relational expression' is true, then the rest of the line will be executed, however if it is false then control will immediately pass to the next line.

NOTE 1) Permissible relational operators for expressions are:

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

Permissible relational operators for strings are:

= equal to

2) TCT BASIC's implementation of the IF statement is non-standard, strictly only a GOTO statement or the word THEN followed by a line number should follow the IF statement.

3) The word THEN may be placed between an IF statement and its succeeding statement, yet this has no effect on its operation.

4) Simple strings may not be compared directly, i.e. all string comparisons must be made between predefined strings denoted by string identifiers.

The syntax of the IF statement is:

<if statement> ::= IF <if value> <relational operator> <if value> <if terminator>
<if value> ::= <string identifier> / <expression>
<relational operator> ::= =/<> /=</>=/</>
<if terminator> ::= <> / THEN

THE 'PRINT' STATEMENT

The PRINT statement is the means by which output is obtained from a BASIC program while it is executing; output of both the numeric values of expressions or literal strings may be obtained.

The form of the PRINT statement is the word PRINT which may be abreviated to PR in most circumstances, followed by a list of items to be printed. Separate items in the list are separated by commas and may be either strings or expressions. Expressions are printed as a numeric value the format of which may be controlled by the FIX statement (c.f.), while strings are reproduced verbatim less their leading and trailing quote marks. Normally a carriage return line feed is transmitted at the end of each PRINT statement. However this may be supressed if desired by the inclusion of a semicolon after the last item of the 'print list'.

NOTE 1) There is one circumstance in which the abbreviation PR may not be used. This is something of the form PRINT INT(<expression>), for if the abbreviated form is used this becomes PR INT(<expression>), which will cause the value of the expression to be printed, not the integer part as would happen in the first instance. (However anything of the form PR "HELLO", INT(A / PI) is still legal.)

2) If a trailing ';' is used then the next PRINT statement will print on the same line.

3) The word PRINT by itself will not cause a line to be fed as in some BASICs: that is a PRINT statement without a print list is not allowed and the form PR"" should be used instead.

4) There is no mechanism for the inclusion of " marks to appear in strings and therefore they may not be printed, this is true of all special symbols except carriage return, which may be printed indirectly.

The syntax of the PRINT statement may be summarised as followes:

<print statement> ::= PRINT <print list> <print terminator> / PR <print list> <print terminator>
<print list> ::= <print item> / <print list> <print item>
<print item> ::= <expression> / <string>
<print terminator> ::= <> / ;

EXAMPLES OF THE PRINT STATEMENT

```
PRINT "YOUR CURRENT POSITION IS ", X, Y, "AND VELOCITY ", V
PR RND(R)
PR $1, A, $4;
```

THE 'FIX' STATEMENT

The FIX statement is the means by which the format of numeric output may be controlled.

The form of the FIX statement is the word FIX followed by a single digit or the letter 'S' .

Execution of the FIX statement merely sets a flag as to the format of numeric output, it has no effect on the internal calculations or manipulations of numbers at all. When a PRINT statement is executed and a number is to be printed, this flag is inspected, if a fix of 'S' was specified the number is printed in scientific notation: that is, as a ten digit number followed by an 'E' and then the power of ten to which it should be raised. If a digit was specified then the number will be printed in floating point format with the specified number of decimal places displayed; if the number is too large or too small to represent in floating point format then it will be printed in scientific format automatically.

NOTE 1) NO rounding is performed on the printing of floating point numbers.

The syntax of the FIX statement is,

<fix statement>::= FIX<fix>
<fix>::=0/1/2/3/4/5/6/7/8/9/S

EXAMPLES OF THE 'FIX' STATEMENT

FIX 1	result of a subsequent 'PR 127.89' is 127.8
FIX S	result of a subsequent 'PR 127.89' is 0.1278900000E 03
FIX 9	result of a subsequent 'PR 127.89' is 127.890000000

THE 'INPUT' STATEMENT

The 'INPUT' statement is the means by which an executing BASIC program may receive information from the operator. Input may be in the form of an expression (for a numeric variable) or a string.

The form of the input statement is the word INPUT followed by a single character, this character is the 'prompt' which will be printed when the statement is executed. This prompt character is followed by an 'input list' which may be composed of single letter variables or string identifiers separated by commas.

Upon execution, the prompt character will be printed and the desired input should be entered from the keyboard or tape. The values which are typed in are assigned to the corresponding variable in the variable list (input list); the typed values must, of course, agree in type with the variables in the list. Numeric responses (if more than one) should be separated by spaces or commas. String, or literal variables, should be separated by carriage return.

NOTE 1) Expressions are allowable in place of a number when a numeric variable is being assigned a value. That is it would be legal to respond with 'PI/2' to the statement 'INPUT ? A'

2) It should be remembered that if a numeric variable appears in the input list then the response must be an expression, and if a string is required then the response must be a string also.

3) When responding to an 'INPUT' statement 'backspace' and 'delete' perform the same function as when entering normal commands and text.

The syntax of the INPUT statement is;

<input statement> ::= INPUT <prompt> <input list>
<input list> ::= <input item> / <input list> , <input item>
<input item> ::= <string identifier> /<variable>
<prompt> ::= <non special character>

EXAMPLES OF THE 'INPUT' STATEMENT

INPUT ?A, B, C a valid response would be, ?123.3, 4, 4.5E-20

INPUT * $N a valid response would be, *HI THERE

The GOSUB statement is the mechanism by which subroutines may be called in a BASIC program.

The form of the GOSUB statement is GOSUB expression. Where the value of the expression is a number greater than zero and less than 10000 (truncation is automatic).

When executed program flow will be temporarily diverted to the line with the number returned by the expression. Upon encountering a RETURN statement program flow will be resumed from the line following the line where the last GOSUB appeared. If subroutines are nested too deeply a 'NST ERROR' will result. If the line number specified in the expression does not correspond to an actual line a 'NOGO ERROR' will result.

The syntax of the GOSUB statement is;

<gosub statement>::= GOSUB<expression>

EXAMPLES OF THE 'GOSUB' STATEMENT

```
GOSUB 1000
GOSUB INT( N/RND(R))
```

THE 'RETURN' STATEMENT

The RETURN statement is the means by which control is reverted to some main program after a GOSUB has been executed.

The form of the RETURN statement is merely the word RETURN.

When executed, control will pass to the line after the line on which the last GOSUB occured. If a RETURN is encountered by a program and no GOSUB has been executed corresponding to it (i. e. the program is at zero subroutine level) a 'RTN ERROR' will result.

NOTE 1) A RETURN statement must be the last statement on a line, for all statements after it will be ignored due to the fact that program flow has been resumed at another point.

The syntax of the RETURN statement is ;

RETURN

EXAMPLE OF THE RETURN AND GOSUB STATEMENTS:

```
100 GOSUB  500


500 PRINT "PLEASE ANSWER ONLY 'YES' OR 'NO' "
510 RETURN
```

THE 'FOR' STATEMENT

The FOR statement is the standard method of creating loops in BASIC. A FOR NEXT loop, as it is called, will repeatedly execute a set of BASIC statements while incrementing a specified variable by a specified amount, until that variable reaches, or exceeds, a particular value.

The form of the FOR statement is the word FOR followed by a numeric variable (the 'for-variable') this is in turn followed by '=' then an expression (the 'start value') followed by the word TO then a second expression (the 'finish value') . After this an optional 'step' may be specified by the word STEP and an expression (the 'step value').

When executed the 'for-variable' is set to the 'start value' and control is passed to the susequent set of statements. When the corresponding NEXT statement is reached the 'for-variable' is incremented by the 'step value' (if no step value was specified it is assumed to be 1), and compared to the 'finish value'. If the 'for variable' is greater than or equal to the 'finish value' control passes to the next statement.
If the 'for-variable' is less than the 'finish value' a GOTO is executed to the statement after the FOR statement.
As a result if a construct of the form

```
FOR I=1 TO N
    any collection of statements
NEXT I
```

is used then the collection of statements will be executed N times.

NOTE 1) Fractional 'step values' are allowed, yet if recurring decimals are used it should be remembered that they do not return an exact value.

2) It is not permissable to have a 'final value' less than the 'start value' or a negative 'step value'.

3) FOR loops may only be nested 4 levels deep else a 'NST ERROR' will result.

4) FOR loops are the fastest method of performing recursive operations in TCT BASIC.

5) 'Offset' nesting of FOR loops is of course not allowed, i. e. The first NEXT after a particular FOR must match that FOR.

The syntax of the FOR statement is;

<for statement> ::= <simple for> / <simple for> <step>
<simple for> ::= FOR <numeric variable> = <expression> TO <expression>
<step> ::= STEP <expression>

(FOR EXAMPLES SEE 'THE NEXT STATEMENT')

THE 'NEXT' STATEMENT

The NEXT statement is the loop delimeter corresponding to the FOR statement.

The form of the NEXT statement is the word NEXT followed by the same variable as was specified in the last FOR statement. If a different variable is given a 'NXT ERROR' will result.

For a description of the effect of the NEXT statement see the FOR statement.

NOTE 1) A 'NXT ERROR' will result if a NEXT is encounted before a corresponding FOR statement.

The syntax for the NEXT statement is;

<next statement> ::= NEXT <for variable>

EXAMPLES OF THE FOR AND NEXT STATEMENTS

```
FOR I=1 TO 10
PRINT ""
NEXT I

FOR A=1 TO INT(RND(R)*N): PRINT "JUST ONCE MORE": NEXT A

FOR N=0.0 TO 25 STEP 2
M=M+N
PRINT M
NEXT N
```

THE 'DO' STATEMENT

The DO statement is a non standard provision of TCT BASIC for the construction of loops. The DO UNTIL construct is in fact identical to the REPEAT UNTIL construct found in PASCAL. The DO statement is used when it is unknown how many times a particular operation is to be performed (unlike the FOR statement where it is necessary to know this at the loop's comencment).

The form of the DO statement is simply the word DO followed by any group of statements.

When executed, no immediate action is taken which has any effect on the users program, the address in text of the DO statement is merely stored on an internal stack for reference by the next UNTIL statement. (See 'THE UNTIL STATEMENT' for a description of the operation of a DO-UNTIL loop.)

NOTE 1) A DO statement need not be the last statement on a line.

2) DO loops are non-standard BASIC and bear no relation to the DO loops of FORTRAN, yet are similar to the REPEAT loops of PASCAL.

The syntax of the DO statement is;

<do statement>::= DO

(FOR EXAMPLES SEE 'THE UNTIL STATEMENT'.)

THE 'UNTIL' STATEMENT

The UNTIL statement is the loop delimiter corresponding to the DO statement.

The form of the UNTIL statement is the word 'UNTIL' followed by any relational expression (for an explanation of relational expressions see the IF statement.)

The effect of a DO UNTIL loop is to repeat the set of statements between the 'DO' and the 'UNTIL' repeatedly until the relational expression after the 'UNTIL' returns a true value.

NOTE 1) The statements between the 'DO' and the 'UNTIL' will always executed at least once.

The syntax of the UNTIL statement is:

until statement ::= UNTIL relational expression

EXAMPLES OF THE DO AND UNTIL STATEMENTS

```
DO
A=A+1
UNTIL $A=$1000

DO:INPUT ? A:UNTIL A<25
```

THE 'REM' STATEMENT

The REM statement is the means of inserting documentation into a BASIC program.

The form of the REM statement is the word REM followed by any string of characters.

The REM statement is ignored completely during execution of a program.

THE 'STOP' STATEMENT

The STOP statement is used to terminate an executing Basic program.

The form of the STOP statement is the word STOP followed by any string of characters.

When this statement is executed the string of characters following the word STOP is printed out and the interpreter returns to command mode.

THE 'PIPBUG' STATEMENT

The PIPBUG statement is used to return control to the Philips monitor program 'PIPBUG' or any other program located at 0000.

The form of the PIPBUG statement is simply the word 'PIPBUG'.

FUNCTIONS

There are four inbuilt functions in TCT BASIC. These are:

ABS	Returns the absolute value of its argument.
MOD	Calculates A*FRAC(B/A) where 'A' is the first argument and 'B' is the second.
INT	Returns the integer portion of its argument
FRAC	Returns the fractional portion of its argument.

The arguments to the functions are listed within brackets after the function name and separated by commas.

Two optional functions are available (RND and SIN) for details of their operation see the sheet supplied with the tape of TCT BASIC.

While on certain other tapes FRAC may not work correctly until the following is performed.

```
*A161A
161A  D9  DE
*
```

To implement RND and SIN the relevant section of the tape must be loaded.

SIN(A) will return the sin of A
The function will only return true value if the number is between plus and minus pi.

RND(G) will randomize the variable and return the number.

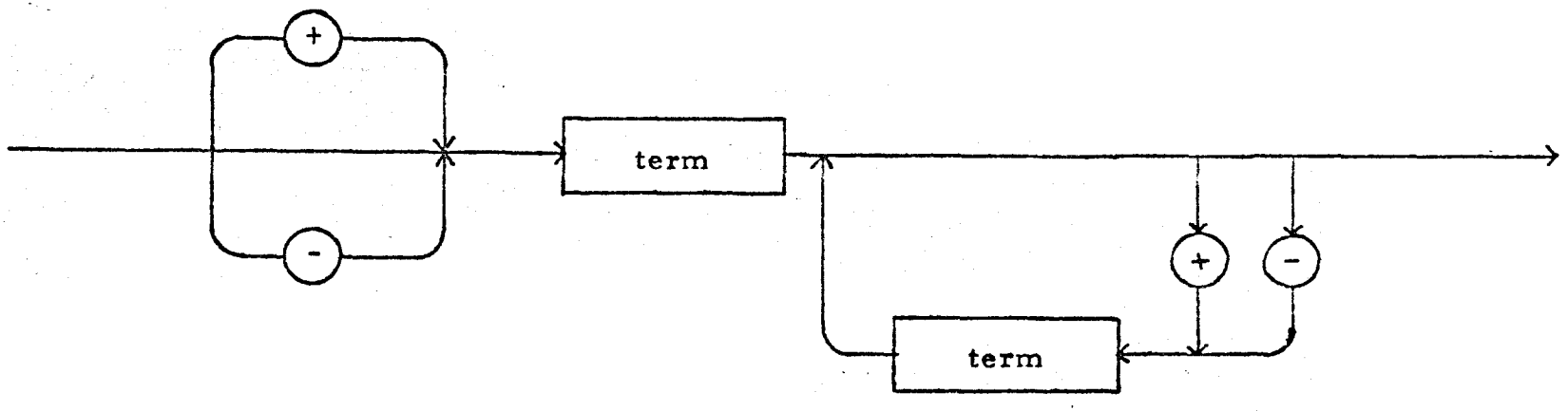
BACUS NAUR

<basic program> ::= <basic line> / <basic program> <basic line>
<basic line> ::= <sequence number> <basic statements> CARRIAGE RETURN
<sequence number> ::= NUMBER
<basic statements> ::= <basic statement> / <basic statements> : <basic statement>
<basic statement> ::= <let statement> / <fix statement> / <if statement> / <until statement> / <do statement> /
<goto statement> / <gosub statement> / <return statement> / <next statement> /
<for statement> / <print statement> / <input statement> / <stop statement> /
<machine statement> / <rem statement>
<let statement> ::= LET <assignment statement> / <assignment statement>
<assignment statement> ::= <variable> = <expression> / <string identifier> = <string>
<fix statement> ::= FIX <fix>
<fix> ::= <digit> / S
<if statement> ::= IF <relational expression> <if terminator> <basic statements>
<if terminator> ::= <> / THEN
<until statement> ::= UNTIL <relational expression>
<relational expression> ::= <string relation> / <expression> <relational operator> <expression>
<relational operator> ::= <>/=/<=/>=/</>
<string relation> ::= <string identifier> = <string identifier>
<do statement> ::= DO
<gotot statement> ::= GOTO <expression>
<gosub statement> ::= GOSUB <expression>
<return statement> ::= RETURN
<next statement> ::= NEXT <variable>
<for statement> ::= FOR <variable> = <expression> TO <expression><step>
<step> ::= <> / STEP <expression>
<print statement> ::= PRINT <print list> <print terminator> / PR <print list> <print terminator>
<print list> ::= <print item> / <print list> , <print item>
<print item> ::= <expression> / <string>
<print terminator> ::= <> /
<input statement> ::= INPUT <prompt> <input list>
<input list> ::= <input item> / <input list> , <input item>
<input item> ::= <string identifier> / <variable>
<prompt> ::= <non-special character>

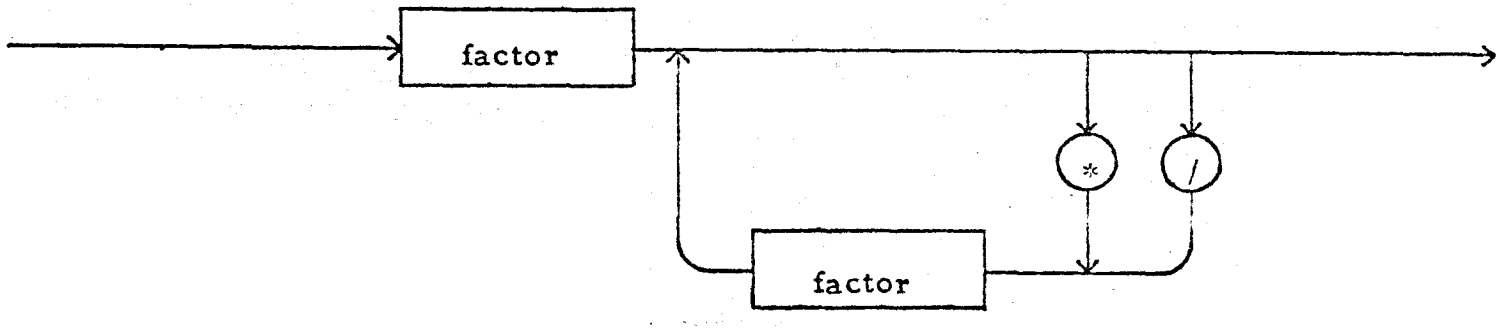
‹stop statement› ::= STOP ‹comment›
‹rem statement› ::= REM ‹comment›
‹comment› ::= ‹› / ‹character list›
‹machine statement› ::= PIPBUG
‹expression› ::= (‹expression›) / ‹expression› ‹operator› ‹expression› / ‹value identifier›
‹value identifier› ::= ‹number› / ‹variable identifier› / ‹function› / ‹constant›
‹function› ::= ‹function identifier› (‹expression list›)
‹expression list› ::= ‹expression› / ‹expression› , ‹expression list›
‹operator› ::= +/-/*/⁄
‹variable identifier› ::= A/B/C/D/E/F/G/H/I/J/K/L/M/N/O/P/Q/R/S/T/U/V/W/X/Y/Z
‹function identifier› ::= ABS/MOD/INT/FRAC/ ‹optional function›
‹optional function› ::= RND/SIN
‹constant› ::= PI
‹number› ::= ‹decimal part› / ‹decimal part› ‹exponent›
‹decimal part› ::= ‹integer› / ‹integer› . ‹integer› / . ‹integer› / ‹integer› .
‹integer› ::= ‹digit› / ‹integer› ‹digit›
‹digit› ::= 0/1/2/3/4/5/6/7/8/9
‹exponent› ::= E ‹integer› / E ‹sign›‹integer›
‹sign› ::= +/-
‹string› ::= ‹simple string› / ‹string identifier›
‹string identifier› ::= $ ‹expression›
‹simple string› ::= " ‹character list› "
‹character list› ::= ‹non special character› / ‹character list› ‹non special character›
‹non special character› ::= ‹any ASCII character except CR " DEL BS ›

FLOW CHART REPRESENTATION OF THE SYNTAX OF AN EXPRESSION

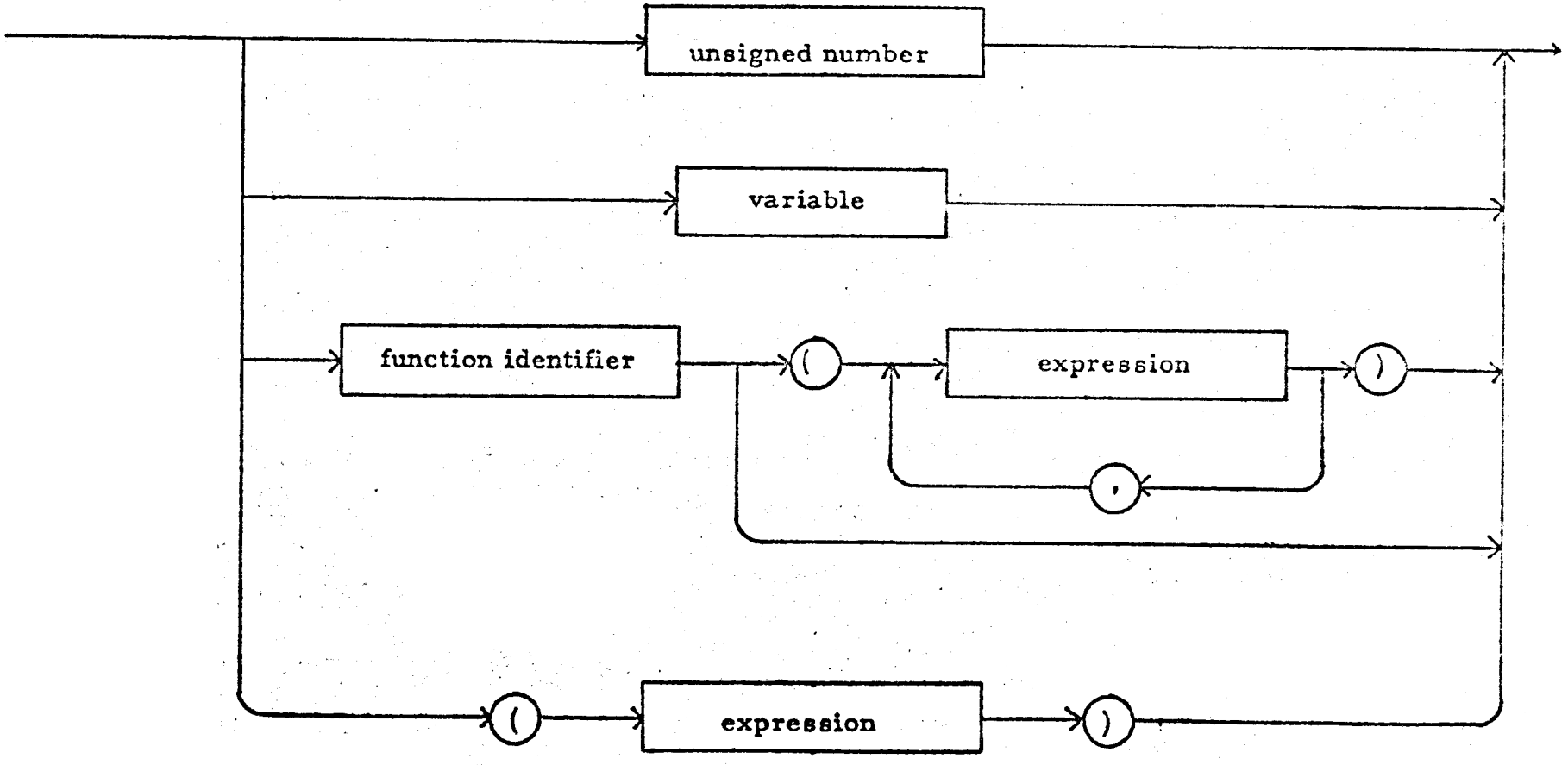
expression



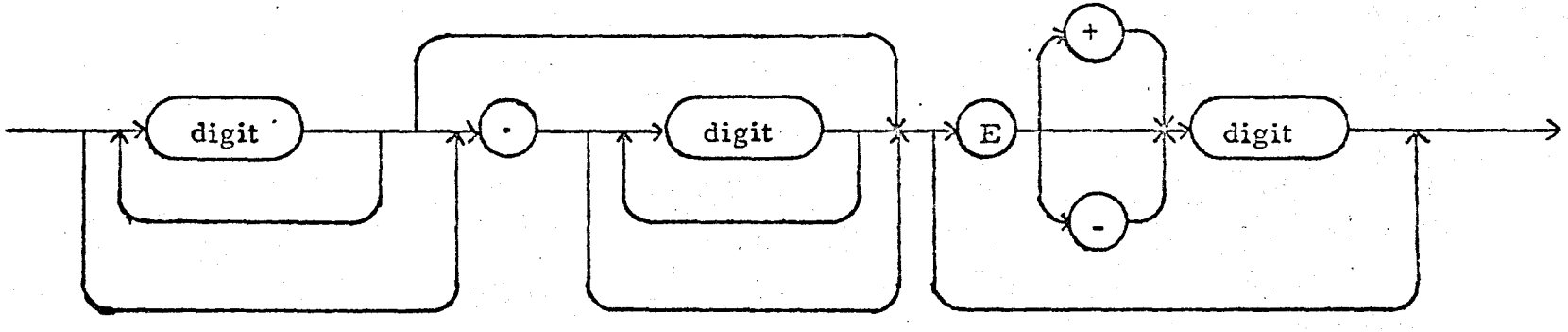
term



factor


unsigned number
variable
function identifier
(
expression
)
,
(
expression
)

FLOW CHART REPRESENTATION OF THE SYNTAX OF AN UNSIGNED NUMBER



ERROR MESSAGES

Line numbers given are those at which the error is detected.

STP ERROR	No STOP on the end of the program. Line number given is the line number of the last line executed.
STMT ERROR	Character(s) remaining after the logical end of statement.
VALU ERROR	Computed value of an expression is out of range for a function or an overflow has occured.
NO " ERROR	A string definition has no " to terminate it.
NOGO ERROR	Line number evaluated does not exist.
RTRN ERROR	A RETURN has been encounted without a GOSUB.
NEST ERROR	Too many pending opperations in an arithmetic expression or too many nested FOR-NEXT or DO-UNTIL loops, or subroutines.
DIV 0 ERROR	A zero divisor has occured in an expression.
UNTL ERROR	An UNTIL has occured without a DO.
NO $ ERROR	A NEXT has been encouted without a FOR, or the variable of the NEXT statement is not the same as that of the preceeding FOR statement.
SNTX ERROR	Incorrect syntax - see Bacus-Naur.
BUFF OVF ERROR	Input buffer length is exceeded.

CHAR ERROR	Indicates that a string was not found.
NEXT ERROR	Indicates that a NEXT was encounted without a FOR, or the NEXT variable did not match that in the previous FOR statement.

```
0001 REM     THIS IS A PROGRAM TO DEMONSTRATE CERTAIN UNIQUE
0002 REM     FEATURES OF TCT BASIC
0003 REM
0004 REM     IT TAKES A NUMBER BETWEEN 0 AND 999 INCLUSIVE AND
0005 REM     WRITES IT OUT IN WORDS
0006 REM
0010 $1="ONE "                      :REM     SET UP A LOOKUP TABLE
0020 $2="TWO "                      :REM     OF KEY WORDS
0030 $3="THREE "
0040 $4="FOUR "
0050 $5="FIVE "
0060 $6="SIX "
0070 $7="SEVEN "
0080 $8="EIGHT "
0090 $9="NINE "
0100 $10="TEN "
0110 $11="ELEVEN "
0120 $12="TWELVE "
0130 $13="THIR"                     :REM     ITS EASIER TO PRINT
0140 $14="FOUR"                     :REM     "TEEN" LATER THAN
0150 $15="FIF"                      :REM     REPEATEDLY TYPE IT NOW
0160 $16="SIX"
0170 $17="SEVEN"
0180 $18="EIGH"
0190 $19="NINE"
0200 $20="TWEN"
0500 PRINT "PLEASE GIVE ME A NUMBER"
0510 DO:INPUT ? A: UNTIL A<1000:REM     INORE BAD INPUTS
0515 IF A=0 PRINT "ZERO"GOTO 510:REM    ZERO IS A SPECIAL CASE
0520 IF A<100 GOTO 600              :REM     TEST FOR ABSENCE OF
0530 PRINT $INT(A/100),"HUNDRED";:REM: "HUNDRED'S" DIGIT
0560 A= MOD(A,100)
0570 IF A<>0 PRINT " AND ";
0600 IF A<20 GOTO 700               :REM     TEST FOR AN ABSENSE OF
0605 IF INT(A/10)=2 PRINT "TWEN";:GOTO 650 :REM   "TEN'S" DIGIT
0610 PRINT $INT(A/10)+10;
0650 PRINT "TY ";
0660 A= MOD(A,10)
0700 IF A=0 GOTO 800
0710 PRINT $A;
0720 IF A>=13 PRINT "TEEN";
0800 PRINT""""
0810 GOTO 510
```

```
0800 04 0D CC 18 00 04 FE CC 18 01 3F 0E 27 07 32 0D
0810 87 F2 3F 08 53 E5 FE 98 76 04 0D CC 87 F2 3B F3
0820 04 FF CC 87 F2 3F 0E E0 12 9E 0E 68 07 3A 3B 23
0830 0D 87 FA 3B 1E 0E 87 FA 01 44 1F CC 07 E3 CE 07
0840 E4 F5 20 1C 08 AA F5 80 18 2F F5 40 18 22 3F 87
0850 E3 1B 55 0F 67 C1 D8 0C CF 67 C1 0F 67 C0 84 01
0860 CF 67 C0 17 CF 67 C1 17 28 6A AC 53 6E 00 00 17
0870 CC 07 FA CE 07 FB 1F 08 28 45 1F 0F 07 F9 E7 30
0880 1A 05 07 21 1F 0A 12 0C 07 FA CF 25 D7 0C 07 FB
0890 CF 25 D7 CB E7 C9 F1 CA F5 1F 08 28 07 28 0C 87
08A0 E8 E4 20 15 16 3F 08 53 1B 74 C1 3B 6F 06 00 01
08B0 1A 1B 0E A7 FA C1 44 7F EC 87 E8 98 04 3B E7 1B
08C0 6E A6 01 1C 0E F7 3F 0C C3 FA 7B 1B F7 07 3A 3B
08D0 D5 FA 7C 1F 08 28 0F 07 F9 0F 65 D7 CC 07 FB 0F
08E0 45 D7 CC 07 FA A7 01 CB EE 17 0F 07 FC E7 10 1A
08F0 05 07 21 1F 0A 12 0C 07 E8 CF 26 77 0C 07 E9 CF
0900 26 77 CB E7 17 00 00 00 00 00 00 00 00 00 3F 08
0910 0C 0D 87 E8 3F 08 53 E5 0D 14 E5 3A 14 07 0D 1B
0920 52 0F 07 FC 19 04 07 1D 1B 75 0F 46 78 C2 0F 46
0930 78 CB EF C1 E4 05 18 0C CC 07 E8 CE 07 E9 04 01
0940 CC 07 F4 17 20 1B 79 07 88 3F 0A 1B 1F 08 00 07
0950 28 0D 87 E8 3F 08 53 01 E4 22 14 E4 0D 18 05 3F
0960 02 B4 1B 60 07 15 1F 0A 12 1A 07 04 20 75 02 1F
0970 02 B4 04 2D 1B 79 3F 14 C1 0F 66 9A 3B 6B C0 C0
0980 0F 66 99 C2 E6 10 9E 09 F0 F6 80 18 FA 0D 07 E2
0990 18 F5 0F 66 98 18 06 01 A4 02 A2 1A EA 1F 16 26
09A0 0F 66 98 1A 14 3B 2F FA 7C A5 01 18 0A 3B 23 3B
09B0 25 F7 05 18 02 F9 78 1B 13 A5 01 3B 11 02 18 08
09C0 04 30 3B A2 A5 01 FA 78 3B 0C F9 7C 1B AB 04 30
09D0 3B 94 04 2E 1B 90 0F 66 9B B5 02 98 0B 75 02 87
09E0 01 44 0F 64 30 1F 02 B4 77 02 50 50 50 50 1B 71
09F0 3B 5C 05 0A 3B 60 F9 7C 3F 09 6B 04 45 3B E7 0F
0A00 66 93 3F 09 69 06 02 0F 66 94 3B 4D FA 79 1B E9
0A10 07 39 74 07 3F 00 8A 3B 02 1B 0E 0F 2A 63 50 C1
0A20 44 7F 3F 02 B4 01 9A 73 17 07 00 3B 6E 0C 07 F4
0A30 1C 0E 68 3B 66 3B 20 CB 9F 04 04 CF 66 91 0C 07
0A40 F8 CF 66 94 0C 07 F7 CF 66 93 04 2C CC 07 FB 04
0A50 12 CC 07 FA 1F 08 28 0F 07 FD 20 06 08 CF 26 97
0A60 FA 7B 17 17 40 8A A4 A4 9E A5 40 82 A9 A6 A8 A1
0A70 40 A6 A8 9A A9 AC 82 98 AB 9C 9E 40 45 9C 9E 8E
0A80 9F A4 A8 A4 9D 9C 8A A6 A9 54 88 55 88 92 AC 40
0A90 61 AA 9C A8 99 86 90 82 A5 9C 8A B0 A9 A6 9C A8
0AA0 B1 84 AA 8C 8C 40 9E AC 8D 0C 07 F4 18 05 0C 07
0AB0 E7 1B 02 04 3E 06 00 3F 02 B4 3F 17 D3 C1 18 7A
0AC0 E4 08 98 06 A6 01 1A 61 1B 70 CE 25 77 E4 0D 18
0AD0 0E E4 7F 1C 14 F7 E6 60 98 60 07 3D 1F 0A 12 3F
0AE0 00 8A 05 05 CD 07 E8 05 78 1F 14 F3 A8 86 A8 40
0AF0 84 82 A6 92 86 76 40 AE A4 92 A8 A8 8A 9C 40 84
0B00 B2 74 1A 14 A8 5C 98 9E 9C 8E 58 40 A8 5C AE 9E
0B10 9E 98 98 8A A4 58 40 86 5C 84 82 A4 A4 82 A8 A8
0B20 5C 1A 14 50 86 52 40 62 72 6E 70 41 40 40 40 75
0B30 02 05 0A 3F 0A 57 CF 07 FD 0A FC 3F 08 9C 0C 87
0B40 E8 E4 2D 98 08 04 F0 CE 66 92 3F 08 53 0C 87 E8
0B50 C0 C0 C0 C0 3B 30 18 39 E4 2E 9C 14 C1 3B EC 0C
```

```
0B60 87 E8 E4 30 98 04 F9 75 1B 00 04 0A A1 CE 66 91
0B70 18 05 04 F0 CE 66 90 05 0A 3B 0B 98 06 3B 33 3B
0B80 CA F9 76 1F 0B D5 0C 87 E8 E4 30 16 E4 39 15 E0
0B90 17 3B 73 18 14 04 0A A1 0F 07 FD CF 66 91 0C 87
0BA0 E8 E4 2E 98 5E 07 28 1B 56 3B 07 3F 08 53 F9 61
0BB0 1B 1B 44 0F B5 02 18 0A D0 D0 D0 D0 CE 66 93 77
0BC0 02 17 75 02 6E 66 93 CE 66 93 86 01 17 0A CA 04
0BD0 10 C0 CE 66 91 0A C2 77 02 3F 08 9C E4 45 9C 0D
0BE0 C2 05 00 3F 08 A5 E4 2D 98 05 05 F0 3F 08 53 3F
0BF0 0B 86 9C 0C 41 44 0F C2 CA 8E 3B F1 3B F2 98 0C
0C00 44 0F D2 D2 D2 D2 62 CC 07 EA 3B E1 0E 07 FD 0E
0C10 66 91 C3 0E 66 90 E1 18 16 EB ED 98 04 20 C1 1B
0C20 14 19 06 08 E3 A3 94 1B 0C C1 AB DC 97 1B 05 87
0C30 66 8B D5 97 03 0B D6 CF 66 91 01 CF 66 90 1F 0D
0C40 C2 3F 0C C3 0C 87 E8 E4 45 98 76 1B F2 0F 07 FD
0C50 A7 08 CB FA 0F 66 9B C2 0F 66 9C C1 0F 66 98 1A
0C60 05 0F 66 99 19 05 07 11 1F 0A 12 E4 05 9A 77 C3
0C70 E7 04 18 13 04 F0 52 52 52 52 42 46 0F 51 51 51
0C80 51 45 0F 61 C1 DB 69 CE 07 EE CD 07 EF 17 20 CC
0C90 07 F3 05 18 CD 07 F2 CD 07 EB 07 32 0D 87 F2 3F
0CA0 08 53 E5 0D 98 76 0D 87 F2 F5 FE 14 77 02 3B F0
0CB0 0E 87 F2 E9 D3 1A 65 19 0A EE 07 EF 1A 5E 19 03
0CC0 20 C8 D5 03 77 10 C3 0F 67 C0 C1 0F 27 C0 C2 A6
0CD0 01 77 08 A5 00 75 08 02 CF 67 C0 01 CF 47 C0 75
0CE0 10 17 0D 07 EB 06 FF 0E A7 E8 E4 0D 98 79 02 61
0CF0 1C 0D 6E 02 14 01 38 F9 86 03 3F 0D 5B 0D 87 E5
0D00 07 23 3F 08 53 07 25 3B FA E5 FF 98 70 02 C1 3B
0D10 F2 F9 7C 07 23 3F 0C C3 07 25 3B FA 0C 87 E3 CC
0D20 87 E5 0C 07 E4 EC 07 F3 98 69 0C 07 E3 EC 07 F2
0D30 98 61 0F 07 EE CF 87 F2 07 32 3F 08 53 0C 07 EF
0D40 CC 87 F2 3B F6 1B 03 C0 3B F1 0C 87 E8 CC 87 F2
0D50 E4 0D 14 07 28 3B E4 07 32 1B 6D 0F 07 F2 CF 07
0D60 E3 CF 07 E5 0F 07 F3 CF 07 E4 CF 07 E6 17 3B 6B
0D70 07 23 3F 08 53 0D 87 E3 E5 0D 98 76 3B F5 0C 87
0D80 E3 CC 87 E5 E4 FF 14 07 25 3B E8 07 23 1B 6D 07
0D90 32 3F 17 BF 1B 04 3B FA 3B 97 0D 87 F2 E5 FE 14
0DA0 3B 1D 3B 8D 0D 87 F2 3B 16 04 20 3F 02 B4 12 15
0DB0 3F 08 53 0C 87 F2 E4 0D 18 5C 3F 02 B4 1B 71 1F
0DC0 02 69 3F 0A 57 CF 07 FD 3F 11 A3 3F 11 D9 1F 0E
0DD0 B9 0C 07 F4 18 2A 07 28 3F 0C C3 0C 87 E8 E4 0D
0DE0 98 11 3F 08 53 0C 87 E8 3F 0E 59 0C 87 E8 CC 07
0DF0 F8 C0 C0 3B EE 05 12 06 58 CD 07 FA CE 07 FB 17
0E00 20 C8 CF CC 07 FD CC 07 F9 05 11 06 FF 1B 6A 20
0E10 CC 07 FE CC 07 FC CC 07 FF 04 01 CC 07 F4 CC 07
0E20 E9 04 18 CC 07 E8 17 04 01 CC 07 F3 04 18 CC 07
0E30 F2 17 05 00 3F 08 9C E4 53 18 09 3F 0B 89 9C 0A
0E40 10 A4 2F C1 CD 07 E2 1F 08 53 08 F9 CC 07 EB 04
0E50 01 C8 F2 17 08 F7 C8 ED 17 E4 FE 98 05 07 09 1F
0E60 0A 12 CC 07 F7 1F 08 53 20 CC 07 F4 CC 07 FD CC
0E70 07 F9 04 11 06 FF 1F 08 70 20 3F 0E 29 3B 0C 07
0E80 32 3F 08 53 0C 87 F2 E4 FF 98 76 0D 07 F2 3F 02
0E90 69 0D 07 F3 3B F9 1F 00 8A 3F 08 9C 3F 14 B3 15
0EA0 16 C2 3F 08 53 0C 87 E8 3B F3 1C 0C C3 3F 14 BB
0EB0 D0 0D 07 FD CD 26 97 C9 F9 07 3A 3B E6 1B E4 09
```

```
0EC0 F1 0D 46 98 C2 07 08 0E 26 EF CD 26 97 FB 78 C9
0ED0 E1 17 07 28 75 FF 1F 0A 12 07 11 1B 77 00 00 00
0EE0 20 C3 CF 66 C0 CF 67 00 DB 78 04 03 CC 07 E2 04
0EF0 3E CC 07 E7 1F 0E 68 0C 07 E3 0E 07 E4 1F 08 70
0F00 05 08 0D C4 40 CD 64 ED 0D E4 42 CD 64 E4 59 72
0F10 CD 04 EC 05 08 20 CD 44 F5 59 7B 17 0F 84 44 27
0F20 F0 1B 03 0F 84 44 77 0A 3B 56 CF 04 E6 75 01 0D
0F30 04 ED ED 04 E4 18 16 59 0E 04 04 05 E4 06 08 3F
0F40 11 38 3F 11 56 1B 66 04 04 05 ED 1B 70 0C 04 EE
0F50 EC 04 E5 19 62 18 04 59 60 1B 6C 0C 04 E6 EF 04
0F60 EF 98 10 3B 2C 06 09 04 04 05 ED 3F 11 38 3F 11
0F70 56 1B 1B 77 01 3B 31 1A 04 3B 3A 1B 11 CF 04 EF
0F80 06 06 0E 44 E7 AE 64 F0 94 CE 64 F0 5A 74 1F 0F
0F90 F0 75 01 05 00 06 06 51 0E 44 F0 84 66 D1 8E 64
0FA0 E7 94 CE 64 F0 5A 70 17 06 FA 0E 63 F6 EE 63 ED
0FB0 98 02 DA 76 17 77 01 06 06 0E 44 F0 AE 64 E7 94
0FC0 CE 64 F0 5A 74 17 07 0B 05 00 3B 5C 1A 07 3B 65
0FD0 85 66 95 1B 75 8D 04 FB CD 04 FB 04 04 05 ED 06
0FE0 0F 3F 10 48 FB 62 3F 11 56 07 0B 3F 10 51 FB 7B
0FF0 04 04 05 ED 06 0A 3B 14 3F 11 20 0C 04 FC 90 0E
1000 D9 07 08 0F 44 ED CF E4 46 5B 78 17 CE 04 E2 CC
1010 04 E0 CD 04 E1 07 00 05 02 75 01 0D A4 E0 98 19
1020 87 02 E5 08 98 75 C1 04 F0 CC 84 E0 04 99 CD A4
1030 E0 20 CD A4 E0 CC 04 FC 17 44 F0 98 02 87 01 03
1040 14 3B 0E 3B 21 FB 7A 17 CE 04 E2 CC 04 E0 CD 04
1050 E1 05 04 0E 04 E2 75 01 0E C4 E0 D0 CE E4 E0 E6
1060 03 98 75 F9 6E 17 0D A4 E0 0E 84 E0 18 0A 84 67
1070 94 1C 0E D9 CD E4 E0 17 58 0F EC 04 FC 98 07 06
1080 F0 CE 84 E0 1B 68 CC 04 FC A4 00 94 1B 66 07 80
1090 0E 84 42 26 F0 1B 05 07 00 0E 84 42 77 0B 3F 0F
10A0 00 03 18 06 0C 04 E7 1C 0E D2 0C 04 EF EC 04 E6
10B0 18 04 04 F0 1B 01 20 CC 04 EF EE 04 ED 98 0C 0C
10C0 04 E5 84 65 8C 04 EE 94 D1 1B 1E 0C 04 EE EC 04
10D0 E5 19 0C 98 02 06 00 0C 04 E5 AC 04 EE 1B 06 0E
10E0 04 ED AC 04 E5 94 CE 04 ED CC 04 EE CD 04 FC 75
10F0 01 04 04 05 E4 06 08 3B 3F 5F 0F C6 07 0C 04 04
1100 05 ED 06 0E 3B 32 FB 76 07 0A 3B 35 0C 04 FB 18
1110 0A A4 0F CC 04 FB 3F 0F 91 1B 71 FB 6D 1F 0F F0
1120 04 B6 C0 05 06 8D 44 F0 94 CD 64 F0 04 66 59 75
1130 B5 01 16 3B 0C 3B 1F 17 CE 04 E2 CC 04 E0 CD 04
1140 E1 05 04 06 02 0E A4 E0 50 CE E4 E0 EE 04 E2 98
1150 74 75 01 F9 6E 17 0D A4 E0 0E 84 E0 98 0E 84 67
1160 94 98 05 05 01 CD 04 FC CD E4 E0 17 58 03 CC 04
1170 FC A4 00 94 58 72 CC 84 E0 1B 6D 0D 07 FD 0D 46
1180 90 84 08 C2 07 08 0D 46 99 CE 46 F0 FB 78 C9 EC
1190 17 0B E9 20 CF 66 92 17 0B E2 0F 66 92 24 F0 CF
11A0 66 92 17 0B D7 87 98 06 06 02 C1 51 03 AD 71 C1
11B0 C3 CE 44 40 04 06 CE 44 40 5A 6E CF 04 47 CC 04
11C0 46 17 08 02 06 0F 07 F4 15 1F 0A 10 07 28 0D 87
11D0 E8 3F 08 53 E5 0D 14 1B 75 3F 0F 23 1B 0D 3F 0F
11E0 1C 1B 08 3F 10 8E 1B 03 3F 10 97 75 FF 0F 07 FD
11F0 A7 08 CB FA 0F 66 93 15 16 CF 66 90 CF 66 91 17
1200 00 8A 0A A9 32 08 8D 52 01 0B 2F 52 14 0C 4D 0C
1210 8E 0C E2 52 01 32 34 4C 49 53 D4 17 BA 52 4F 0C
```

```
1220 4D 0C 8E 52 26 0E 27 0D 8F 51 FF 0A 10 0E 4A 09
1230 76 0E 54 0E 00 32 3F 52 55 CE 09 0E 0E 0F 0D D1
1240 32 48 43 4C 45 41 D2 08 0A 34 97 4E 45 D7 08 00
1250 0B 2F 52 24 52 1E 00 00 00 32 5D 4C 45 D4 0E 99
1260 52 6C 32 2A BD 93 D5 11 7B 09 0E 0D D1 32 77 46
1270 49 D8 0E 32 09 0E 0D D1 32 89 49 C6 96 37 32 83
1280 54 48 45 CE 08 6F 15 4C 52 58 32 9A 55 4E 54 49
1290 CC 11 C5 96 37 09 0E 15 72 0D D1 32 A4 44 CF 09
12A0 0E 15 55 0D D1 32 C5 47 CF 32 B2 54 CF 93 D5 09
12B0 0E 52 BD 32 2A 53 55 C2 93 D5 09 0E 08 EA 0C 4D
12C0 0C 8E 14 DB 0D D1 32 D3 52 45 54 55 52 CE 09 0E
12D0 09 21 0D D1 32 EA 4E 45 58 D4 11 C5 0E 99 52 2A
12E0 09 0E 16 8D 93 D3 16 77 0D D1 00 33 14 46 4F D2
12F0 11 C5 0E 99 52 2A 32 2A BD 93 D5 32 2A 54 CF 93
1300 D5 33 0A 53 54 45 D0 93 D5 53 0C 15 D7 09 0E 16
1310 47 11 7B 0D D1 33 30 A4 93 D5 32 2A BD 33 25 A2
1320 0C 4D 17 0E 53 2C 32 2A A4 94 13 17 40 09 0E 0D
1330 D1 33 5D 50 D2 33 39 49 4E D4 33 40 A2 09 4F 53
1340 4D 33 49 A4 93 D5 16 F3 53 4D 93 D5 09 76 33 52
1350 AC 53 39 33 57 BB 53 59 00 8A 09 0E 0D D1 33 89
1360 49 4E 50 55 D4 15 90 11 C5 0E 99 53 7F 15 15 93
1370 D5 11 7B 15 00 33 77 AC 0E 99 53 59 15 00 48 67
1380 32 2A A4 94 13 17 38 53 59 11 33 93 53 54 4F D0
1390 14 C9 0E 00 33 A0 50 49 50 42 55 C7 00 00 00 00
13A0 00 34 A3 52 45 CD 11 CC 0D D1 93 D5 33 B2 BD 93
13B0 D5 15 1E 33 C7 BC 33 BC BD 93 D5 15 20 33 C3 BE
13C0 93 D5 15 24 93 D5 15 1C 32 2A BE 33 D1 BD 93 D5
13D0 15 1A 93 D5 15 22 33 DE AD 93 FB 11 98 53 E3 33
13E0 E1 AB 93 FB 33 EE AB 93 FB 11 A3 11 D9 53 E3 33
13F0 F9 AD 93 FB 11 A3 11 DE 53 E3 08 D6 94 13 34 08
1400 AA 94 13 11 A3 11 E8 53 FD 33 F9 AF 94 13 11 A3
1410 11 E3 53 FD 0E 99 54 1B 0E BF 08 D6 0B 2F 54 21
1420 08 D6 34 2B A8 93 D5 32 2A A9 54 31 34 33 50 C9
1430 15 D5 08 D6 34 41 41 42 D3 32 2A A8 94 24 11 91
1440 08 D6 54 54 53 49 CE 32 2A A8 94 24 04 50 08 D6
1450 08 6F 08 6F 00 34 62 49 4E D4 32 2A A8 94 24 15
1460 99 08 D6 34 71 46 52 41 C3 32 2A A8 94 24 16 07
1470 08 D6 32 2A 4D 4F C4 94 88 11 A3 11 E3 17 9F 11
1480 E8 08 D6 11 A3 11 E8 08 D6 32 2A A8 93 D5 32 2A
1490 AC 93 D5 32 2A A9 08 D6 32 58 53 49 5A C5 09 0E
14A0 0E 78 0D D1 32 2A 44 55 4D D0 17 BB 52 4F 0D D1
14B0 0D D1 D6 E4 41 16 E4 5A 15 E0 17 A6 41 D2 D2 02
14C0 17 0F 07 FD A7 08 CB FA 17 3F 08 9C 0D 87 E8 3F
14D0 08 53 E5 0D 14 01 3F 02 B4 1B 71 0C 07 EB 18 05
14E0 07 19 1F 0A 12 04 01 CC 07 F4 0C 07 F2 0D 07 F3
14F0 CC 07 E8 CD 07 E9 17 07 25 3F 0A 1B BB A5 1B 17
1500 0C 07 E8 0D 07 C0 C8 FC C9 F7 0C 07 E9 0D 07 C1
1510 C8 FC C9 F7 17 3B 69 1F 0A A9 A4 03 A4 01 A4 01
1520 A4 01 A4 01 44 07 CC 07 E3 3F 11 A3 3F 11 DE 0D
1530 07 FD A5 08 C9 FA 07 04 0D 66 9B 18 06 0D 66 9A
1540 18 02 53 53 4B E1 CF 07 F6 1F 08 D6 0B F9 15 3F
1550 11 CC 1F 0D D1 0F 07 F4 1C 0A 10 0F 07 FF E7 20
1560 9E 08 F1 0C 07 E8 CF 26 57 0C 07 E9 CF 26 57 CB
1570 EB 17 0B E8 93 05 07 2D 1F 0A 12 0C 07 F6 98 0B
```

```
1580 0F 66 57 C8 F5 0F 66 56 C8 DA 17 A7 02 CB CD 17
1590 3F 08 9C CC 07 E7 1F 08 53 77 02 0F 07 FD 0F 66
15A0 90 18 05 A7 08 1F 0A 5A 0F 66 91 18 76 E4 09 15
15B0 50 C2 44 7F 83 C3 F6 80 98 0E 0F 66 93 44 F0 CF
15C0 66 93 87 01 46 7F 86 01 20 E6 05 18 05 CF 26 92
15D0 DA 77 75 02 17 A4 01 A4 01 A4 13 D0 D0 D0 0F 07
15E0 FD C2 05 08 0E 35 EE CF 26 97 F9 78 CB F1 17 00
15F0 01 00 31 41 59 26 54 00 01 00 10 00 00 00 00 00
1600 00 00 50 00 00 00 00 00 0C 07 FD C3 84 08 C2 3B 0B
1610 CB F6 3F 15 99 3F 11 A3 1F 11 DE 05 08 0F 26 8F
1620 CE 26 8F F9 78 17 01 82 E4 01 98 05 04 30 1F 02
1630 B4 02 1C 09 B9 1F 09 A0 33 A9 A4 94 13 32 2A BD
1640 32 2A A4 94 13 17 66 0F 07 FE E7 4C 9E 08 F1 0E
1650 07 FD A6 10 CA FA A6 09 0E 66 98 CF 26 07 0C 07
1660 E8 CF 26 07 0C 07 E9 CF 26 07 05 10 0E 26 A0 CF
1670 26 07 F9 78 CB D2 17 0B CF 0E 07 F6 98 0B 0F 65
1680 F6 C8 DC 0F 65 F7 C8 DD 17 A7 13 1B 67 0D 07 FD
1690 0E 07 FE 98 05 07 35 1F 0A 12 0E 65 F5 ED 66 97
16A0 9C 16 95 CC 07 F6 3F 0E BF 07 08 0A E4 0E 25 FF
16B0 CD 26 97 FB 78 C9 D7 3F 11 A3 3F 11 D9 0A E5 09
16C0 CD 07 08 0D 26 8F CE 26 EF FB 78 C0 C0 0A C2 CD
16D0 07 FD 07 08 0E 25 F7 CD 26 8F FB 78 17 3F 0E 27
16E0 07 32 0D 87 F2 3F 08 53 E5 FE 98 76 3F 0C 97 0D
16F0 07 EB 17 3F 0C 4D 3B 65 18 05 07 31 1F 0A 12 3B
1700 E5 3B E3 0C 87 F2 E4 0D 14 3F 02 B4 1B 73 07 28
1710 06 00 0D 87 E8 3F 08 53 01 CE 25 77 E4 0D 1C 09
1720 64 E4 22 98 6D 04 0D CE 65 77 3F 15 00 3F 16 DD
1730 3F 0A E2 3F 0C E2 1B F3 3F 15 15 3F 0C 4D 1B 6D
1740 3B FA 3B EA 9C 16 FA 07 32 3F 08 53 3B FC 0C 87
1750 F2 CD 25 77 E4 0D 98 74 3B D1 3B 5F 0F 07 F4 15
1760 04 0D CC 87 E8 17 3B D4 3B C1 3B C2 3F 14 EA 3F
1770 0C 4D 3F 16 DD 3B 1E 3B 1C 20 CC 07 F6 0D 87 F2
1780 ED 87 E8 98 08 3B 0E E5 0D 98 72 C9 EE 3F 15 00
1790 1F 08 D6 C0 C0 07 32 3F 08 53 07 28 1B FA 17 0C
17A0 07 FD 84 08 C3 84 08 C2 3F 16 1B 3F 16 07 03 84
17B0 08 CC 07 FD C2 84 08 C3 1B EF 20 CC 07 F6 17 08
17C0 FB 98 03 1F 00 8A 04 0D 3F 02 B4 05 00 3F 02 A8
17D0 F9 7B 17 77 1Ø Ø6 Ø8 12 1A 7D 3F 02 AD C1 3F 17
17E0 F7 3F 02 A8 3B F9 44 80 51 61 C1 FA 74 3B F3 45
17F0 7F 76 40 01 75 18 17 12 9A 03 76 40 17 74 40 17
```