VIKY

A Language for VME Systems

Issue: May 1992
VIKY Version: 29–May –1992

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 1

An Introduction to VIKY

This chapter gives a brief insight into the workings of the VIKY programming language.

VIKY provides a means of directly manipulating VME hardware from within a high level language in a computer independent fashion. It is derived from BASIC and enjoys many of the advantages of that language – such as easy editing and modification of programs. Extensions have been provided for the hardware manipulation, interrupt support and those operations useful in such an environment.

When commanded, the source is compiled and run. It is unlike the majority of BASIC systems which interpret the source each time a statement is executed. As a result, the speed of execution is reasonably good and is ideal for VME testing and adequate for many applications using VME hardware.

This implementation is based on the recommendations contained in the publication “Language Definition of CATY 1” (published by U.K. CAMAC Association) with later enhancements, and this document is based on the format and contents of that publication.

Chapter 2

Basic Principles

This chapter explains the basic principles of the VIKY programming language.

Section 1 Introduction

A VIKY program consists of a number of lines or steps. Each line contains:

- A line number (optional).
- One or more program statements.
- A comment, introduced by a semi–colon (optional).

Each line is limited to 80 characters. Multiple statements on a single line must be separated by a space, back slash or a colon. Following a PRINT statement, however, a colon or back slash should be used.

A simple demonstration program might be:

```
1000 LET A=0
1010 LET A=A+1
1020 FOR N=1 TO A PRINT A : NEXT N
1030 GOTO 1010 ;GO BACK AND INCREMENT
```

When run, this program would cause the printing of one 1, two 2's, three 3's etc. (e.g. 1, 2, 2, 3, 3, 3, etc.).

When a statement is entered it is merged by line number such that:

- A statement with a line number that duplicates an existing line number, replaces the original line.
- A statement with a new line number is inserted or appended as appropriate.

If the statement entered was appended to the end of the program, the new step number (10 higher than the last) will be issued automatically. If a statement is entered without a line number it will be obeyed immediately.

Section 2 Notation

Throughout this document, the following abbreviations are used:

<ln>	A line number of one to five digits having a value in the range 1 to 32767.
<v>	A simple variable name i.e. a letter (A..Z) optionally followed by one or more letters or numeric characters (0..9).
<av>	As defined for <v>.
<vav>	Either <v> or <av>.
<vs>	A subscripted variable in the form A(3).
<vvs>	Either <v> or <vs>.
<n>	A constant expressed in decimal (e.g. 35) or in octal with leading apostrophe (e.g. ’43) or the ASCII code with leading dollar (e.g. $B).
<nvvs>	Either <vvs> or <n>.
<st>	A string variable name i.e. a name as defined by <v> followed by dollar e.g. TVAR$.
<sts>	A subscripted string variable in the form AB$(6).
<stsvs>	Either <sts> or <vs>.
<stav>	Either <av> or the name of a string of more than one character.
<stvav>	Either <stav> or <v>.
<ststsvvs>	One of <st>, <sts>, <v> or <vs>.
<ststsnvvs>	Either <ststsvvs> or <n>.
<ao>	One of the arithmetic operators + (addition), – (subtraction), / (division), * (multiplication), the logical operators &, EOR and IOR, or the arithmetic shift operators UP and DOWN.
<ro>	One of the relational operators: = (equal to), < (less than), > (greater than), <= or =< (less than or equal to), >= or => (greater than or equal to), # or <> or >< (not equal to).
<aan>	Alpha–numeric string starting with an alphabetic character and having at least one more alphabetic or numeric characters (e.g. ABC, A123, BO4, etc.). The string can be of any practical length.

<text>	A sequence of characters within a pair of delimiting characters. The set of delimiting characters includes +, !, %, ”, and , (the comma). The text must not include the delimiter chosen to mark the sequence. For example,
	`”THIS IS TEXT”`
	`%THIS ONE INCLUDES THE ”%`

<textst>	Either <text> or <st>.
<textsts>	Either <text> or <sts>.
<ae>	A simple expression of the form <nvvs> or <nvvs><ao><nvvs>, and whose result is an arithmetical value. For example, `A A+3 A-B(4) 2&C 1UP4`

<irq>	A VME interrupt request level in the range 0 to 7.
<c>	A VME crate number.
<cvvs>	Either <c> or <vvs>.
<m>	A VME address.
<mvvs>	Either <m> or <vvs>.
<sa>	A VME address modifier in the range 0–3F.
<savvs>	Either <sa> or <vvs>.
<cm>	A legal VIKY command of the form !XX.
<stmt>	A legal VIKY statement.
<strm>	A stream number.

<file>	A file descriptor. For example,
	`C:\FRED\TEST.VIK`
	`B:VTD1612`
	The account defaults, as normally, to the current session default account. Files written by VIKY have no protection and so can be purged and deleted at will. On all systems the file extension defaults to .VIK for VIKY source and .DAT for data. If the file name is omitted where one is expected, it defaults to VIKYDU. Note that this applies only to VIKY source files.

The terms <null> and <null command> imply an operator response of no input other than a <RETURN> line terminator.

Names in capitals enclosed in angular brackets refer to the physical keys of the terminal keyboard such as <RETURN>, <SPACE> etc..

Section 3 Numeric Variables

Variables are referenced by a single letter name (A to Z), or a letter followed by one or more letters or numeric characters (e.g. B5). Each may be a single variable (Scalar) or an array (Vector), but a given name cannot be used for both. All variables are 32 bit signed integers, 16 bit unsigned integers, 32 bit floating point variables or a character string. There is no checking for overflow or underflow of numbers. The numerical result of an overflow or underflow is undefined. Simple 32–bit integer and string variables need not be explicitly declared but may be simply used. For example,

```
1000 LET C=20
1010 INPUT A$
```

Floating point or real numbers must be declared with a statement similar to

```
1020 REAL C,FA,G9
```

Computer word length numbers (in this case 16–bit numbers) must also be declared. For example,

```
1040 COMPW E,H
```

Strings and arrays of integers and real numbers must be explicitly declared as dimensioned with a DIM statement. Thereafter they are normally referenced with a subscript. For example,

```
A(7)
D$(2)
E(T)
```

The first element of an array has subscript 1.

NOTE: Type declaration and dimensioning of an array requires two separate statements:

```
REAL AA
DIM AA(10)
```

The maximum array dimension is 65536. If a variable or an array element appears on the right hand side of an assignment (LET) statement without first having a value assigned to it, its value is indeterminate. This facilitates debugging since if the program is not altered, running it again will find the variables with the values reached at the time of exit.

Section 4 Text Variables

ASCII characters may be stored in VIKY text variables. A text variable contains only one character and a text string may be stored in an array which is dimensioned. For example,

```
1000 DIM T$(10)
.
.
2000 LET T$(1)="TEXT"
```

The letters that form the word **TEXT** will be stored in the first four bytes of the array T$, with the fifth byte Variables containing the value 0 to indicate the end of the string. Routines are available to manipulate strings and strings may be used for:

- File names in assignments.
- As parameters in subroutine calls.
- In INPUT and PRINT statements.

Section 5 VME Interfacing and Supported Controllers

VIKY is available for use with Microsoft Windows 95/98/dos operating systems and Hytec Electronics Ltd VME 3331 controller.

5.1 VIKY

This version uses the Hytec 3331 with direct mapping to the I/O registers. The base address of the registers may be selected from 200,280,300 or 380 with 300 as default. Interrupts are polled at every GOTO, or equivalent, instruction.

```
> VIKY
```

Chapter 3

VIKY COMMANDS

This chapter gives a view of the commands available in the VIKY programming language.

Section 1 Introduction

Commands are acted upon by VIKY immediately and do not form part of the stored program. Commands may be entered whenever the system is in command mode. A dot prompt (.) is issued to indicate readiness for command mode input.

Commands may also be entered following the system's automatic printout of a new line–number, if a different number is required, simply delete the characters issued as a prompt and type the required line number followed by the command.

All commands are distinguished by the presence of an exclamation mark (!). They may optionally be preceded by a step number <ln> or a range of step numbers entered as <ln> – <ln>. For a command <cm>, the forms are:

```
!<cm>
<ln>!<cm>
<ln1>-<ln2>!<cm>
<ln>!<cm><n>
```

Examples:

```
1000-1250!DU
6000!RE10
!PA5
```

The individual commands are described below. If a command does not take a line number, any supplied line numbers are ignored. If a range of lines is specified for a command that only requires one, the first line number <ln1> is used and the second <ln2> is ignored. Line numbers must actually exist except that if <ln> is greater than the highest in existence it is ignored. <ln2> must not be less than <ln1>.

Section 2 Immediate Commands

Following is a list of the immediate commands available in VIKY. The list is in three columns. The first column shows the command. The second column shows the format(s) that the command can take. The third column explains the function of the command.

2.1 !AB

```
!AB
```

ABORT VIKY. This command is used to terminate VIKY and to return to the operating system.

2.2 !AS

```
!AS<strm> <file>
```

ASSIGN STREAM. This command allows the entering of input and output filenames for stream I/O instructions. Stream numbers must lie in the range 1 to 4. For example,

```
!AS4        AUX1:
!AS1        D:\HTEST\VIKY.DAT
```

Streams can also be opened dynamically, by use of the internal subroutine CALL ASSIGN. They may be closed dynamically by using the CALL STCLOSE statement. Device and directory will be set to the default values if omitted and applicable (normal DOS actions).

2.3 !CO

```
!CO <file>
```

COMPILE. This command is used to generate a file with the same name as that following the !CO command but with extension ".DLT". This file is used with the !GO command – see below. NOTE: no extension must be included on the filename following !CO. It is assumed to be .VIK by default.

2.4 !DL

```
<ln>!DL
<ln1>-<ln2> !DL
```

DELETE LINES. Program line <ln> or lines <ln1> to <ln2> inclusive are deleted. The line(s) specified MUST exist or no deletion will take place.

2.5 !DU

```
<ln1>-<ln2>!DU <file>
```

DUMP PROGRAM. The command !DU will dump the entire source program on to the specified file. If <file> is not specified, the filename will default to VIKYDU. If the filename already exists, a new version will be created and the original will be destroyed.

If <ln1> is specified, the dump commences from that statement number and if both <ln1> and <ln2> are specified, only that range of lines is dumped, permitting the saving of GOSUB or similar sections for use with later programs.

2.6 !ED

```
<ln>!ED
```

EDIT LINE. The line specified by <ln> is displayed ready for editing. The line may be edited using the PC's <left> and <right arrow>, <home>, <end> and <insert> keys as well as the normal characters. It is not necessary for the cursor to be at the end of the line when it is accepted by hitting <RETURN>.

When in insert mode, the cursor is a narrow line and when in overwrite mode, the cursor is a solid block.

If the line number is altered during the edit, the edited line is merged as appropriate and the original line remains unaltered.

Having displayed a line for edit it is not essential to carry out such an edit. A new command or statement may be entered as usual.

2.7 !GO

```
!GO <file>
```

START PRECOMPILED PROGRAM. Each time that a program is started with the !ST or !RU command, it is compiled. For big programs this is sometimes quite time consuming. It is useful to convert stable programs to .DLT form with the !CO command. They may then be started much more quickly using !GO <file>.

2.8 !LS

```
!LS
```

LIST SUBROUTINES. Subroutines incorporated within VIKY are listed. Included are the subroutines such as VIN (...) internal to VIKY and FORTRAN and C subroutines included at Task–Build and also any subroutines incorporated by use of the !SR command. The names of subroutines, other than FORTRAN subroutines, listed by the command !LS are followed by a number indicating the expected number of parameters for that routine.

2.9 !NE

```
!NE
```

This command deletes all of any previously entered source program. It does not remove any subroutines entered by an !SR command.

2.10 !NS

```
!NS
```

This command removes from the VIKY system area any subroutines compiled by !SR commands.

2.11 !PA

```
!PA<n>
<ln>!PA
<ln>-<ln>!PA
```

PRINT ALL. The program is divided into 19 line pages for listing. The first form of this command causes page <n> to be listed on the terminal. If <n> is omitted, the first page is assumed (i.e. page 1). The second form causes a page starting with line <ln> to be listed. Following these a <null command> will list the next page, and so on.

The last version allows the specification of two line numbers. The statements from the first given through the last given will be listed on the terminal without any division into pages.

2.12 !RD

```
!RD <file>
```

READ FILE. A VIKY program is read from the storage media and merged with any existing source. The file name will default to VIKYDU if no name is specified. See Section 2.2 for the other default rules. The merging of read statements permits the use of pre–stored GOSUB or DATA statements which can be added to user programs whenever required. Caution must be exercised, however, as the merging rules are the same as for entering source, and a repeated line number will cause the previous line with that number to be deleted.

2.13 !RE

```
!RE
<ln>!RE
<ln>!RE<n>
!RE<n>
```

RENUMBER. The program statements are renumbered in original order with the first statement as <ln> and rising in steps of <n>. Blank statements are not removed. Destinations of GOTO, GOSUB, etc., are corrected. Both <ln> and <n> default to 10.

2.14 !RU

```
!RU <file>
<ln>!RU
!RU
```

RUN. The source program is compiled and run. PEEP statements are ignored. If a file name is specified, the source in that file is compiled but only the binary is loaded into memory. If no file name is specified the source code currently loaded into the source buffer is compiled.

If <ln> is specified execution commences from that statement; otherwise at the first executable statement. If <ln> is greater than the highest statement number of the program, execution starts at the beginning of the program.

2.15 !SR

```
!SR <file>
```

A subroutine previously stored in source format on file with name <file> is compiled and added to the VIKY system area. The line numbers and variables of this subroutine are purely internal, parameters are then passed to it using. The operation of this call is to first read in the file specified, overwriting any existing lines if the line numbers are the same, then compiling the resulting code before deleting the source. Thus, any existing VIKY source before the !SR command might be included in the subroutine and will be deleted. Thus, care should be taken to either issue a !NE command before !SR or issue the !SR command immediately after starting VIKY.

2.16 !ST

```
!ST <file>
<ln>!ST
```

START. This behaves as for !RU except that PEEP lines are obeyed.

Section 3 Output Radix Commands

```
!OC          (octal radix)
!DC          (decimal radix)
!CH          (character output)
!BI          (binary output)
!HX          (hexadecimal output)
!RA n        (set radix)
```

Sets the radix for all integer outputs, that is for all PRINT, PRINT@, and PEEP statements, until another !xx command is issued. On first loading, VIKY output will be in OCTAL radix until altered by an !xx command. !CH will be in OCTAL radix until altered by an !xx command.

!CH will cause all simple variables to be output as 4 packed ASCII characters. The character in the most significant byte will be output first, followed by the middle ones, and the least significant last.

The output radix may also be changed dynamically, see CALL RADIX().

The !RA command must be followed by a number from the following range:–

```
0            equivalent to           !CH
2            equivalent to           !BI
8            equivalent to           !OC
10           equivalent to           !DC
16           equivalent to           !HX
```

Floating point variables are always printed in ”E” format. Character string variables are always output as a characters.

Section 4 Keyboard Interrupt

If VIKY is running in a program requiring no operator intervention, or in an inadvertent loop, the user may type the <ESC> key. VIKY will return to command mode and issue the message,

```
KEYBOARD AT <ln>

      <ESC>
```

Chapter 4

Non–VME Statements

This chapter details the statements available in the VIKY programming language that are non–VME statements.

Section 1 Control and Arithmetic Statements

Following is a listing of the control and arithmetic statements used in the VIKY programming language.

1.1 Blank Lines

Blank lines are indicated by a statement number with no statement. These are ignored. They may be referenced by a control statement (GOSUB, GOTO, etc.) in which case the control passes to the next executable statement following.

1.2 CALL

```
FORMAT(S):  CALL <aan> (ststsnvvs, ..., <ststsnvvs>)
```

Allows entry to subroutines with parameters. The subroutine entry has dummy parameters for which the parameters given in the CALL are substituted when the subroutine is called.

Array names may be passed as a parameter, but a subscript must be given (i.e. for array A the parameter A(n) must be passed). If the corresponding parameter is dimensioned as an array within the subroutine (i.e. DIM Z(50)) then a reference in the subroutine to Z(m) will actually reference the users variable A(m+n–1).

Variables used within a subroutine are local to that subroutine. Only the variables passed as parameters in the call to the subroutine and global variables are external. Thus the program:

```
150 LET B=8
160 LET A=5
170 CALL ADD(A,C)
180 PRINT A,C
190 STOP
200 SUBROUTINE ADD(X,Y)
210 LET A=X+6
220 LET Y=A
230 RETURN
```

Will print out:

```
5          11
```

Subroutines may have been entered as part of a user program, compiled from prestored programs and added to the VIKY system area, or be actual VIKY internal subroutines.

1.3 COMPW

```
FORMAT(S):  COMPW <vav>, <vav>, ....
```

This statement declares that the variable <v> or <av> is of “computer word length” (16 bits) and not 32 bits. Declaration of large arrays as COMPW will potentially save memory and increase throughput.

One or more variables or array variables may be defined in a single COMPW statement, and several COMPW statements may be used as required.

References to real variables or arrays are ignored and references to string variables are errored.

***Note**: the statement INTEGER*2 is equivalent to COMDW and preferred.*

1.4 DATA

```
FORMAT(S):  DATA <n>, <n>, ....
```

A series of values to be read sequentially by READ statements.

1.5 DIM

```
FORMAT(S):  DIM <stav>(<n>), <stav>(<n>), ....
```

This statement declares that the variable <stav> is an array of dimension <n>. This must occur as the first reference to <stav>. For example:

```
10 DIM A(1000),BH$(20)
```

A given variable can only be dimensioned once and only have a single dimension. Subscripts begin numbering at 1.

One or more arrays may be defined in a single DIM statement, and several DIM statements may be used as required. The contents of the array elements are undefined.

1.6 FOR

```
FORMAT(S):  FOR <v> = <nvvs> TO <nvvs> STEP <nvvs>      NEXT <v>
```

These two statements define the start and finish of a program loop. Initially, <v> is set to the value of the first <nvvs> and the statements up to the corresponding NEXT <v> are performed. The value STEP <nvvs> is added to <v> and if the new value is not greater than the value TO <nvvs>, the loop is executed again.

If the STEP <nvvs> value is 1, the word STEP and the value may be omitted. STEP may be a negative number for loops stepping from the first <nvvs> down to the second <nvvs>.

When <v> passes the TO <nvvs> value, control passes to the statement following the NEXT statement. <v> will contain the last used value.

Branching out of a FOR ... NEXT loop with a GOTO statement will terminate the loop, but leaving with a GOSUB will return within the loop. Passing control into the range of a FOR loop from outside is illegal, but the variable <v> may be changed by the program within the loop.

FOR loops may be nested, each depth of nesting using a different variable, but may not partially overlap.

Only integer variables may be used for the FOR loop control and only the integer parts of <nvvs> are used.

1.7 GLOBAL

```
FORMAT(S):  GLOBAL <vav>, <vav>, ....
```

This statement declares that the variable <v> or <av> is “global” and that all references in a subroutine to this variable name refer to the same variable.

1.8 GOSUB

```
FORMAT(S):  GOSUB <ln>
```

Causes a subroutine jump to the specified statement number. When a corresponding RETURN statement is found, a return is made to the statement following the GOSUB.

GOSUBs may be nested (GOSUB within other GOSUBs) and may be recursive (GOSUB to themselves). No parameters may be passed and variables are shared with the calling program. For example,

```
1000 LET A=1
1010 PRINT A
1020 GOSUB 1050
1030 GOTO 1010
1040
1050 REM THIS GOSUB ADDS ONE TO A
1060 LET A=A+1
1070 RETURN
```

results in a printing of A with values 1, 2, 3, etc.

1.9 GOTO

```
FORMAT(S):  GOTO <ln>
```

Causes an unconditional branch to line <ln>, or to the first executable statement following if <ln> is a comment or blank line. The statement <ln> must, however, exist. Note that it is not possible to transfer control to other than the first statement of a multi–statement line.

1.10 IF

```
FORMAT(S):  IF <nnvs> <ro> <nnvs> <stmt> ....
            IF <textst> <ro> <textst> <stmt> ....
```

The two values <nvvs> or <textst> are compared and if the relation is true, the statement(s) <stmt>... are executed.

1.11 INPUT

```
FORMAT(S):  INPUT <ststsvvs>, <ststsvvs>, ....
```

Except as described below, this causes typing of the variable name followed by a question mark (e.g. A?).

VIKY then waits for the user to supply the value of the variable. Integer variables may be entered as either a decimal integer or an octal number if proceeded by an apostrophe (’) or as a single ASCII character code if proceeded by a dollar character ($). Real variables may be entered as either an integer or a fractional number with a decimal point or in “E” format. If multiple values are requested in a single INPUT statement, the first will be prompted in the usual way. It is possible to enter any number of values, separated by spaces or commas, and when <RETURN> is entered they will be read into the variables listed. If too many values are entered, the excess values will be rejected with the following warning,

```
EXCESS DATA IGNORED
```

If there were too few values supplied, the prompt will appear for the next value required. For example,

```
100 INPUT A,B,C,D
A?100 200 300
D?
```

Octal, binary and hex values may be entered by prefixing the numerical string by an apostrophe for Octal, by apostrophe B for binary and apostrophe X for hexadecimal (’,’B or ’X) eg:–

```
’17776
’XFFFF
’B1011011
```

If a value is preceded by a minus sign (–) the value will be stored as its twos complement:

```
100 INPUT A
110 PRINT A
!RU
A? -3
 37777777775
```

Subscripted Sting variable names <sts> may be entered, eg. A$(1). Characters may be typed to fill subsequent elements of the variable. eg:

```
100 DIM A$(10)
110 INPUT A$(1)
120 PRINT A$(1)
!RU
A$(1)? HELLO!
HELLO!
```

(see chapter 2, section 4).

If the last print statement before an input statement has ended with a comma, so that the cursor position is not at the start of a new line, VIKY will suppress the printing of the prompt. This permits the inclusion of user prompts,

```
1000 PRINT ”ENTER A VALUE”,
1010 INPUT A
```

will result in the printout,

```
ENTER A VALUE
```

instead of the default prompt.

For integer and real numbers VIKY will normally expect a numeric value to be entered. An invalid input will cause VIKY to print a diagnostic and prompt for the value.

When a PRINT statement which ends with a comma, is followed by an INPUT statement asking for a single variable which was the last variable printed in the PRINT statement, then the <CR> (<RETURN>) key will cause the value of the variable to be unaltered. Typing <ESC> key in answer to an INPUT prompt will return VIKY to command mode (see chapter 3, section 4).

1.12 INPUT @

```
FORMAT(S):  INPUT @<n> <ststsvvs>, <ststsvvs>, ....
```

This behaves as for the INPUT statement, except that the input is taken from the stream <n>. The number <n> must have been equivalenced to a device or a file using an !AS command before running the program.

Note that INPUT@ reads numbers and character strings in exactly the same way as the INPUT statement. Leading spaces are legal, but any unexpected character will cause VIKY to abort with the error message,

```
END OF FILE AT LINE xxxx
```

VIKY will read data written by FORTRAN into integer variables if the FORTRAN program uses,

```
FORMAT (8(I10,3X))
```

when the data format will appear identical to that produced by VIKY.

1.13 INTEGER*2

```
FORMAT(S):  INTEGER*2 <vav>,<vav>
```

This is a modern version of the COMPW statement.

1.14 INTR KEYBOARD

```
FORMAT(S):  INTR KEYBOARD
```

This prefixes a section of code which is entered whenever a keyboard character is pressed. The actual character pressed can be read using CALL QKBIN(<ststsvvs>) or CALL VIN(ststsvvs) (see below). Return is made to the main program via an EXIT statement.

1.15 LET

```
FORMAT(S):  LET <vvs>=<nvvs> <ao> <nvvs>
            LET <vvs>=<nvvs>
            LET <st>=<textst>
```

Set the variable specified to the result of the expression. All arithmetic operators are legal except for string operations.

VIKY also provides LOGICAL and SHIFT operators for integers. The first group comprises &, EOR, and IOR. The arithmetic shift operators UP and DOWN cause the value of the first variable to be rotated the second number of binary places left (up) or right (down). Bits rotated out do not reappear in a subsequent rotate in the reverse sense, and zeroes are rotated in every case.

1.16 ON ... GOSUB

```
FORMAT(S):  ON <vvs> GOSUB <ln1>,<ln2>...
```

The specified variable is examined. If its value is 1, execution passes to the line <ln1>. If its value is 2, execution passes to line <ln2> and so on. On meeting a RETURN statement, execution returns to the first line after the ON ... GOSUB ... statement. If the value of <vvs> does not correspond to a given statement <sn>, execution continues in unaltered sequence.

1.17 ON ... GOTO

```
FORMAT(S):  ON <vvs> GOTO <ln1>,<ln2>....
```

The specified variable is examined. If its value is 1, execution passes to the first statement on line <ln1>. If the value is 2, execution passes to the first statement on line <ln2>, and so on. If the value of <vvs> does not correspond to a given statement <ln>, execution continues in unaltered sequence.

1.18 PAUSE

```
FORMAT(S):  PAUSE
```

Execution of this statement causes the message

```
PAUSE AT <ln>
```

to be displayed, and VIKY waits for input from the keyboard. If this is a <null command>, the program continues. Any other input is disregarded and VIKY returns to command mode.

1.19 PEEP

```
FORMAT(S):  PEEP
```

If the program was started with a command !RU or !RUZ, PEEP statements are ignored.

If the command !ST or !STZ was used, the execution of a PEEP statement causes the message

```
PEEP AT <ln>
```

to be printed. This is followed by the values of all undimensioned variables in the current radix. The system then waits for keyboard input. If a <null command> is entered, execution will continue with the instruction following the PEEP statement.

A <SPACE> character will cause up to one page of elements of the first array to be displayed (two elements per line). Further <SPACE> characters, cause further pages of this array and of the remaining arrays in sequence to be displayed. If all elements of all arrays have been displayed, the <space> command causes the message

```
END
```

to be output.

Entering a variable name in place of the <SPACE> will cause just the specified array to be displayed. If, however, the name specified is a simple variable, the message NOT ARRAY will be displayed.

1.20 PRINT

```
FORMAT(S):  PRINT <textsts nvvs>, <textsts nvvs> ....
```

All the items after the word PRINT are displayed as a single line on the console terminal. Numbers are displayed in the current radix. If the items are followed by a trailing comma, the carriage return linefeed function between this line and the next following line is suppressed, permitting printing on one line with more than one print statement.

If a string variable name <sts> is used, the first element should be entered, ie. as a <sts>, thus A$(1). This will cause the whole sting array to be output.

1.21 PRINT @

```
FORMAT(S):  PRINT@<n> <nvvs> ... <nvvs>
```

This behaves exactly as for the PRINT statement except that the output is directed to the stream <n>. <n> should be previously declared as a string equivalent to a number using !AS.

Great caution should be taken to ensure that output to a file contains only decimal or octal data if the file is later to be re–read using INPUT@ statements, since PRINT@ is able to print numbers in any radix and may also output text sequences.

If a string variable name <sts> is used, the first element should be entered, ie. as a <sts>, thus A$(1). This will cause the whole sting array to be output.

Output to the line printer may be spooled and therefore not printed until the program has stopped.

1.22 READ

```
FORMAT(S):  READ <vvs>, ... , <vvs>
```

Operates similarly to an INPUT statement except that all the DATA statements comprise a serial data string which is read to the specified variables by one or more READ statements. If there is more DATA than variables specified by READ statements, the remaining data are ignored. If there are more variables than data available, an error condition will arise.

1.23 REAL

```
FORMAT(S):  REAL <vav>, <vav> ....
```

This statement defines which variables will be floating point.

1.24 REM

```
FORMAT(S):  REM ....
```

Lines of this form are treated as comments (remarks) and ignored when the program is compiled.

1.25 RESTORE

```
FORMAT(S):  RESTORE
```

Resets the data pointer so that the next READ statement will start accepting data at the first item of the first data statement.

1.26 RETURN

```
FORMAT(S):  RETURN
```

Causes return from a GOSUB subroutine. There may be more than one return in a GOSUB routine. Control passes to the statement after the GOSUB statement.

1.27 STOP

```
FORMAT(S):  STOP
```

It is often possible to exit from a program at intermediate points. This statement returns VIKY to command mode after the message,

```
      STOP AT <ln>
```

1.28 SUBROUTINE

- See Section 3.

1.29 WAIT

```
FORMAT(S):  WAIT
```

WAIT is similar to PAUSE except that no message is displayed. This allows programmed prompts in place of the PAUSE message.

Section 2 Internal Subroutines

The following “internal” subroutines can be accessed by the CALL statement.

2.1 **ALPHA** (this requires a Tektronix emulation driver)

```
FORMAT(S):  CALL ALPHA()
```

This call, with no arguments, sets the terminal into character mode. Characters output using the standard VIKY character output calls will appear on the screen from the last position of the graphics pointer.

2.2 APPEND

```
FORMAT(S):  CALL APPEND (<strm>)
```

This call will cause the stream defined to be opened such that data written to the file will be appended to the existing contents of the file. The call must be made after the ASSIGN call.

2.3 ASSIGN

```
FORMAT(S):  CALL ASSIGN (<strm>)
            CALL ASSIGN (<stsvs>)
            CALL ASSIGN (<text>)
```

This subroutine is functionally the same as the !AS command in associating a stream number with a file. If an integer array is used to define the filename, only the least significant byte of each element is used. The normal defaults apply to generate the full filename (see Section 2.2).

2.4 ATAN

```
FORMAT(S):  CALL ATAN (A,C)
            CALL ATAN (A,B,C)
```

The parameters to this subroutine are expected to be real. The arctangent of A or A/B is calculated and stored in the variable C.

2.5 BCD

```
FORMAT(S):  CALL BCD (<nvvs>,<vvs>)
```

This routine assumes that the first parameter is a binary number, converts it to a binary coded decimal number, and returns the least significant 8 digits in the second parameter.

2.6 BIN

```
FORMAT(S):  CALL BIN (<nvvs>,<vvs>)
```

This routine provides the reverse function to BCD. The first parameter is assumed to be a BCD number of up to 8 digits, and the routine converts it to its binary equivalent and returns the result as the second parameter.

2.7 BINASC

```
FORMAT(S):  CALL BINASC (<vvs>,<sts>)
```

This subroutine converts the input integer into an ASCII string and places the string in the output string starting at the character defined.

2.8 CHKFIL

```
FORMAT(S):  CALL CHKFIL(<strm>,<stsvs>or<text>,<vvs>)
```

This routine checks to see if the specified file exists. The first parameter specifies a stream number, the second parameter defines the file name, with the usual defaults applying. The third parameter returns the result, and it is set to 1 if the file exists, and a 2 if it does not.

2.9 COPY (not yet implemented)

```
FORMAT(S):  CALL COPY(<sts>,<textsts>)
```

This routine copies the second text string into the first text string, which must have sufficient length. If, in doing the copy, the string terminator is overwritten in the target string, a string terminator is placed at the end, otherwise no string terminator is written.

```
1000 CALL COPY(X$(1),"HELLO")
```

leaves the string X$ containing the text string “HELLO”, but,

```
3000 LET T$(1)="NOW IS THE HOUR"
3100 CALL COPY(T$(12),"TIME")
```

would replace “HOUR” with “TIME” in the string T$.

2.10 COS

```
FORMAT(S):  CALL COS(A,B)
```

The parameters to this subroutine are expected to be real. The cosine of A is calculated and stored in the variable B.

2.11 **DRAW** (this requires a Tektronix emulation driver)

```
FORMAT(S):  CALL DRAW(<nvvs>,<nvvs>)
```

This call draws a vector from the current graphics pointer to the new position defined in the call. This also becomes the new graphics pointer position. The two arguments define the new X and Y.

2.12 **ERASE** (this requires a Tektronix emulation driver)

```
FORMAT(S):  CALL ERASE(<nvvs>)
```

This call will cause the screen to be erased. The argument defines the number of <NULL> characters which will be outputted in order to compensate for the finite time the erase takes.

2.13 EXP

```
FORMAT(S):  CALL EXP(A,B)
```

The parameters to this subroutine are expected to be real. The exponent of A is calculated and stored in the variable B.

2.14 LOG

```
FORMAT(S):  CALL LOG(A,B)
```

The parameters to this subroutine are expected to be real. The Napierian logarithm of A is calculated and stored in the variable B.

2.15 LOG10

```
FORMAT(S):  CALL LOG10(A,B)
```

The parameters to this subroutine are expected to be real. The logarithm to base 10 of A is calculated and stored in the variable B.

2.16 MAPIIR

```
FORMAT(S):  CALL MAPIIR (<vvs>,<vvs>,<vvs>)
```

The first two integers are copied as is to the third argument which must be a real number. The first argument is copied to the most significant 16 bit word of the third argument. Only the least significant 16 bits of the input integers are used. This routine needs to be used where, for example, a real number is read from a VME plug–in. Note that this routine is machine family dependant.

2.17 MAPRII

```
FORMAT(S):  CALL MAPRII (<vvs>,<vvs>,<vvs>)
```

This routine is the reverse of MAPIIR. The first argument is a real number the most significant 16 bits of which are copied by this routine as is to the least significant 16 bits of the second argument, an integer. The least significant 16 bits of the real number are similarily copied to the third argument.

2.18 MOVE (this requires a Tektronix emulation driver)

```
FORMAT(S):  CALL MOVE(<nvvs>,<nvvs>)
```

This call sets the Tektronix 4010 into graphics mode and moves the graphics pointer to the position specified. Nothing is drawn on the screen as a result of this call. The first parameter is the X position and the second parameter is the Y position. The instruction manual for the terminal should be consulted for other details. Both parameters must be integers.

2.19 QKBIN

```
FORMAT(S):  CALL QKBIN (<ststsvvs>)
```

An internal variable is zeroed by every !ST or !RU command. If during the execution of the program any key except <ESC> or <CNTL–C> is typed, its 7 bit ASCII value is loaded into this variable.

This internal variable is read into the variable by the call to QKBIN and the internal variable is reset to 0.

The variable <ststsvvs> may now be tested to see if a character has been typed since the last call to QKBIN. It will either contain the ASCII value of the character, or 0 if none had been typed. If the variable is a character string variable it can be tested for 0 by testing against a null string (””).

2.20 RADIX

```
FORMAT(S):  CALL RADIX(n)
```

Dynamically changes the current output radix to be used for all PRINT, PRINT@ and PEEP statements. The single parameter is the radix to be used, and the legal values are:

- 0 Packed ASCII (4 characters per word)
- 2 Binary
- 8 Octal
- 10 Decimal
- 16 Hexadecimal

It may be important to check the current radix prior to any PEEP statements in a program which uses dynamic radix change.

2.21 READ

```
FORMAT(S):  CALL READ (<strm>,<stsvs>,<nvvs>)
```

The counterpart of the WRITE statement. The three parameters indicate the stream to be read from, the first element to be loaded, and the number of elements to load. The stream must already be opened by an !AS command or CALL ASSIGN statement. For example,

```
      00100 CALL READ (2,A(50),50)
```

2.22 SECNDS

```
FORMAT(S):  CALL SECNDS(<vvs>)
```

This call returns the time, in seconds, since midnight.

2.23 SIN

```
FORMAT(S):  CALL SIN(A,B)
```

The parameters to this subroutine are expected to be real. The sine of A is calculated and stored in the variable B.

2.24 SQRT

```
FORMAT(S):  CALL SQRT(A,B)
```

The parameters to this subroutine are expected to be real. The square root of A is calculated and stored in the variable B.

2.25 STCLOSE

```
FORMAT(S):  CALL STCLOSE (<strm>)
```

A stream previously opened by the !AS Command or CALL ASSIGN statement can be closed dynamically by using the command CALL STCLOSE (<strm>).

2.26 STRIN

```
FORMAT(S):  CALL STRIN (<strm>,<ststsvvs>)
```

Identical to VIN except that the character is read from the defined stream. Like VIN characters are not packed, but placed in the least significant byte of the second parameter.

2.27 STROUT

```
FORMAT(S):  CALL STROUT (<strm>,<ststsnvvs>)
```

Identical to VOUT but places the contents of the variable in the stream output buffer specified by the parameter <strm>.

2.28 VIN

```
FORMAT(S):  CALL VIN (<ststsvvs>)
```

Waits for a character to be typed on the keyboard and places it in the variable as a 7 bit ASCII value.

Note that VIN does not convert characters and that lower case characters may appear when typing. Thus ASCII 141 octal may appear for an A when 101 octal was expected. VIN does not pack characters.

2.29 VOUT

```
FORMAT(S):  CALL VOUT (<ststsnvvs>)
```

Places the lower 8 bits of the variable <ststsnvvs> in VIKY's output buffer. Bit–8 of the VIKY variable (octal 200) should be set if the character is a control character, otherwise it should be clear. The buffer will be sent to line only if the character is a <RETURN> (octal 15) or if the buffer is full. It may be forced to print by setting bit–8 of the VIKY variable (octal 200) or will automatically print before accepting input with the subroutine VIN.

VOUT does not unpack characters.

2.30 WAIT (not yet implemented)

```
FORMAT(S):  CALL WAIT(<nvvs>,<nvvs>)
```

This routine institutes a wait call to the operating system. The first element is the number of units to wait and the second parameter defines the units. For the second parameter, 1 is milliseconds, 2 is seconds, 3 is minutes, and 4 is hours. Any number of milliseconds can be specified, but it will be rounded to a 60Hz value.

2.31 WRITE

```
FORMAT(S):  CALL WRITE (<strm>,<stsvs>,<nvvs>)
```

Writes in binary format to the stream specified in the first parameter, starting from the array element specified by the second parameter, a number of array elements specified by the third parameter. The file must previously be opened with an !AS command or dynamically with a CALL ASSIGN statement. For example,

```
00100 CALL WRITE (2,A(50),50)
```

NOTE: It is possible to mix PRINT@, CALL WRITE, and CALL STROUT formatted data to the same stream. It is, however, almost impossible to read such a file. Files using CALL WRITE should not contain any ASCII data.

Section 3 User Subroutines

VIKY allows the writing of subroutines in a manner similar to FORTRAN. Internal dummy variables are replaced at run–time by real variables referenced in the subroutine call.

Subroutines may be incorporated in the program in the same manner as a GOSUB but in addition, VIKY includes a facility to compile user subroutines and add these to the system area of VIKY. To use this facility, the subroutine is first written in the normal way, tested and debugged. It is then stored in source format using a !DU command.

Whenever this subroutine is recalled using an !SR command it will be compiled and added to the system area of VIKY where it may be called by any user program. It is unaffected by !RD or !NE commands, it stays resident until VIKY exits, or until all user subroutines are removed by a !NS command.

The line numbers of a compiled subroutine are discarded at the time of compilation and will not duplicate line numbers used in other subroutines whether these are compiled or not.

Subroutines are called and written using one of the following statements.

```
FORMAT(S):  CALL <aan> (<ststsnvvs>, ....)
```

This statement invokes a subroutine. See under 4.1 above. There is no distinction made between types of subroutines.

```
FORMAT(S):  SUBROUTINE <aan>(<stvav>...,<stvav>)
            RETURN
```

These two statements precede and follow the section of code to be executed as a subroutine. In the SUBROUTINE statement, arrays are referred to unsubscripted and a DIM statement must be included within the subroutine.

```
SUBROUTINE EXAMPL (Z)
DIM Z(50)
:
```

Other variables included in the subroutine but not appearing in the CALL are internal and are not the same as any with the same name in the user program.

Section 4 Subroutines

It is possible to extend the subroutines built into VIKY by writing these routines in Microsoft C. All parameters must be long *. Compile with the command line:–

```
cl /AL /c /Gs file.c
```

Edit VIKYLINK.BAT to include this object module. Edit VIKCSR.TXT: add the new routine name to the end of the file, followed by the number of parameters it expects in parenthesis. Run VIKMSC to generate a new VIKCSR.OBJ. LINK @VIKYLINK.BAT to generate a new VIKY.

Chapter 5

VME and I/O Statements

This chapter briefly describes the statements and functions available in the VIKY programming language.

Section 1 VME Functions

The possible VME functions are:

Type	Mnemonic	
READ	VR8	8 bit data
	VR16	16 bit data
	VR32	32 bit data
	VRST	status register

Type	Mnemonic	Value
WRITE	VW8	8 bit data
	VW16	16 bit data
	VW32	32 bit data
	VWST	status register

Statements to execute a VME function are of the form,

```
VR*         <cvvs><mvvs><savvs><vvs>
VRST        <cvvs><vvs>
VW*         <cvvs><mvvs><savvs><nvvs>
VWST        <cvvs><nvvs>
```

where * may take any value of 8, 16 or 32.

In other words, to perform a VME operation it is necessary to specify the function, the crate, address and address modifiers, and the data to be transferred.

For example, to write the value ’23 (23 octal) to crate 2, address 3 at address modifiers 4 using function 16, the following statement could be written.

```
1000 VW16 2,3,4,’23
```

1.1 Strings

VME addresses and, optionally, address modifiers and even data can be declared as strings for easy reference. Declarations take the form,

```
100 MODULE=1,2,3
.
1000 VW16 MODULE,’23
```

Strings may be declared in the terms of a shorter string, but this is limited to a depth of two (i.e. a string defined partly in terms of another may not be used to define a third).

The example could have been written,

```
100 ADDR= 2,’X1200
110 ADMOD= ADDR,’X09
.
```

1.2 If xxxVTO

```
FORMAT(S)    IF VTO <stmt>
             IF NOTVTO <stmt>
```

These statements allow a statement conditional on the state of the timeout response of the last VME operation performed by VIKY.

1.3 !OF

FORMAT(S) `!OF <c>`

This immediate command will set the offline bit in the crate.

1.4 !ON

FORMAT(S) `!ON <c>`

This immediate command will clear the offline bit in the crate.

1.5 I/O Statements

```
FORMAT(S)   IN8         <nvvs>,<vvs>
            IN16        <nvvs>,<vvs>
            OUT8        <nvvs>,<nvvs>
            OUT16       <nvvs>,<nvvs>
```

These provide access to the IN and OUT instructions of the PC. The first parameter is the address and the second either a variable to hold the reading (if an IN command) or the value to be written.

Chapter 6

Interrupts

Section 1 Operator Interrupts

See Chapter 3, Section 4.

Section 2 VME Interrupts

Interrupt signals are the normal way for a module to demand attention.

Generally a section of program will be written as an interrupt service routine, by heading it with an INTR statement,

```
INTR <c>,<irq>,<nvvs>,<vvs>
INTR <c>,<irq>,*,<vvs>
INTR <aan>
```

An interrupt interrupt will cause a branch to the statement following the INTR statement and when reaching an EXIT statement will resume at the point reached prior to the interrupt. It is the program's responsibility to clear the interrupt in the module before performing the EXIT. Failure to do this may result in repeated execution of the interrupt routine. Clearing the interrupt may be done explicitly or implicitly by some VME function, depending on the module specification.

At an interrupt, the status/id register is read in order to determine which interrupt routine to obey. If one is found with a matching <nvvs> (or an INTR with * as its third parameter exists) then the value of the status/id is written to the variable <vvs> and the interrupt routine is entered.

Interrupts are not enabled until the first GOTO, RETURN, or NEXT statement is obeyed. This allows the user to clear interrupts in his modules at the start of his program.

Modules must be referenced by an INTR statement if that module is to interrupt the program.

Interrupts are not actually used to interrupt VIKY. The INTR bit in the status register is checked at each GOTO, RETURN or NEXT instruction obeyed.

Section 3 EXIT

EXIT Indicates end of an interrupt service routine. After an EXIT statement, operation continues from the point where the main program was broken into.

Chapter 7

VIKY Errors

This chapter briefly describes the errors you may receive while programming in the VIKY programming language.

Section 1 VIKY Errors

1.1 Command Mode Errors

1.2 Program Errors.

VIKY issues error messages at compile time and at run time in an English language format. These are of the form SYNTAX ERROR AT LINE <ln> and are generally self explanatory.

1.3 Run–time Errors.

Run–Time messages may often be FATAL causing the program to terminate and return to the command mode. The special error message EH? will appear whenever a value is omitted in an input stream by striking <RETURN> without a valid input.

1.4 VME Errors

VME errors are generally decoded by VIKY into English language form.