FACTOR
the
S-200 PROGRAMMING
LANGUAGE

Part Number 67095499
Issued: April, 1974

FAIRCHILD SYSTEMS TECHNOLOGY
3500 Deer Creek Road
Palo Alto, California 94304

FAIRCHILD
SYSTEMS TECHNOLOGY

Table of Contents

Table of Contents (Continued)

Table of Contents (Continued)

Table of Contents (Continued)

List of Appendices

List of Appendices (Continued)

List of Appendices (Continued)

List of Tables

Section I
General Information

1.0 INTRODUCTION

This manual discusses the S-200 Programming Language FACTOR (commonly referred to in this text as S-200 FACTOR or simply FACTOR), which is used with the Sentry 200 Test System. FACTOR is a procedural programming language that consists of control statements for the Sentry Test System. The term FACTOR is an acronym derived from 'Fairchild Algorithmic Compiler-Tester ORiented'.

FACTOR provides two basic types of statements: (1) arithmetic and logical control statements, such as those that normally comprise procedural languages; (2) test control statements, which set up and execute functional/parameter tests on electronic elements or devices.

The sections in this manual discuss such subjects as the codes and symbols used by FACTOR for the Sentry 200 Test System and the test statements that are part of FACTOR; its operating procedures, expressions, and control statements.

Section II
Elements Of Factor

2.0 INTRODUCTION

This section defines the basic statements used in the S-200 FACTOR Programming language, and the syntax information used to prepare programs for input to the FACTOR compiler.

2.1 CHARACTER SET

Letters	A through Z and $ #
Digits	0 through 9
Special	( ) * + -/ , . : ; = @ [ ] space ↑ &
Other	! " % ' < > ? ←

All of the above characters may be printed on a teletype (TTP) or may be displayed on a video keyboard/display terminal (VKT). All characters may be entered from either a TTY or VKT keyboard, or card reader, however, the two TTY keys for [\] are unlabeled; they are shift K and M, respectively. The "special" characters have meanings in FACTOR, the "other" characters may be used in a FACTOR program only in a REMARK statement. The meaning of a special character depends on the context in some cases. For example, the colon is used to define the immediately preceeding identifier as a statement label; it also may be used in the binary pin pattern definition in a functional test statement. The correct meaning is chosen by the compiler from adjacent information in the statement. The meaning of all special characters is discussed in the text of this manual. Appendix A tabulates the internal code for the character set. Appendix A also shows where the 029 card punch character set differs from the TTY character set. In the case of the 029 some substitutions are required.

2.2 FACTOR STATEMENTS

A FACTOR statement is the basic functional entity in a FACTOR program. Except for the IF statement, a statement is terminated by a semicolon.

called the binary test program or test plan. The compiler input is called the source version of the test plan.

A program is executed in the order written unless a specific statement alters the flow of control.

2.3.1 Record Format

A record is an arbitrary amount of data read from or written into an input/output device. A record typically contains from 1 to 80 characters, depending on the input/output device type. For a TTY or VKT, a record is the amount of data from one carriage return character until the next carriage return character. For a line printer, a record is one line. Character position within the record is frequently called a column; independently of the medium on which the record is written - even if it is not punched cards. Only the first 72 columns of a record are used for FACTOR input. The next 8 columns are reserved for sequence numbers.

FACTOR provides free field input. That is, there is no implied correspondence between the end of the record and the end of the statement. Also, wherever one space is legal, as many spaces as desired can be used. Hence, a statement may start in the middle of a record and continue for as many records as desired. Conversely, more than one statement can be placed in one record. Statement labels need not start in a fixed field such as column 1, and so on. The restrictions are: (1) individual terms, such as variable names, reserved words, and noise words cannot be divided between two records, and the record may be terminated at any point where a space is a legal character; (2) only the first 72 columns of the record may be used for statements.

2.3.2 Cards

When the user prepares his card deck of source statements there are several options: Up to 72 columns of the card may be used for one or more statements, providing a semicolon delimits each statement. Also, a FACTOR statement may be started on one card and be carried over to the next, provided that individual words of tester instructions are not divided between two cards.

Cards can be sequenced either alphabetically or numerically, or both. The sequence characters are placed in columns 73 through 80. This is why only columns 1 through 72 may be used for statements, since otherwise a portion of the statement would be interpreted as a sequence symbol.

The normal form of sequence numbers is a fixed alpha identifier in (say) columns 73-75, followed by numeric digits in the remaining columns through column 80. These (five) digits will ascend in sequence through the program by a convenient increment, one, ten, etc. Sequence symbols are checked for progression. Gaps (e.g. sequencing by tens) in the sequence may be left, so that program corrections and additions may be made without changing every sequence number in the deck. If a single deck contains more than one alpha identifier, these identifiers must be chosen so that the TRASCII ascending collating sequence is maintained, otherwise a sequence error will be produced.

When an error is detected, the compiler will always type the full current record and a message "SEQUENCE ERROR".

Examples:

	Legal Sequences							Illegal Sequences					
NUMERIC	1	3	3	10	20	99999999		0	1	3	2	5	4
COMBINED	1LB	2LB	3LB					1A	0A	3A	1A		
ALPHA	A	C	D	E	E	E	F	A	B	P	E	C	

2.3.3 Disc

Disc files may be used as a source program input to FACTOR. They must be type "string" and are loaded onto the disc as discussed in the User's manual.

2.3.4 Paper Tape

Source programs may be prepared on punched paper tape. The format rules are the same as previously described for cards. Sequence numbers are not normally used when preparing the tape via a teletype, since paper tape is restricted to 72 readable characters per line. The end of the input line is signaled by "carriage return"-"line feed".

Two editing characters may be punched on the paper tape for correcting punch errors. A character back space is obtained by typing the teletype "control" and "B" keys simultaneously. The number of times this key combination is typed corresponds to the number of previously entered characters which are to be ignored. A line delete is obtained by typing "control" and "L" simultaneously. This character deletes the current line only. It is necessary to type "carriage return" and "line feed" after the last END statement in the program.

The file is terminated by a // followed by "carriage return" and "line feed".

At least 20 characters of blank tape should be provided for leader/trailer.

2.3.5 Teletype/Video Keyboard Terminal

Source programs may be entered directly via the keyboard using the format and rules described in Section 2.3.4, describing paper tape input. In this mode of operation the statements in each input record are compiled as they are entered. The two editing characters discussed above for paper tape apply here also, viz: "control"-"B" and "control"-"L".

2.4 SYNTAX

Many FACTOR statements have numerous possible forms. Syntax notation provides a convenient method of identifying all options precisely and succintly. Special syntatical characters are used to identify alternative forms of the statement. These characters are not entered as part of the statement; they simply define the statement's structure. Throughout this manual, the term "General Form", will be used as notification that the next term is presented in syntax notation. (Refer to Appendix K, FACTOR Syntax Table.)

2.4.1 Constant Parameters

Any word shown in upper case in the general form is a constant parameter and is always entered exactly as shown. The general form of a statement required to disconnect the precision measuring unit is as shown below.

Example:

```
XPMU  PIN; Disconnect the Precision Measuring Unit.

This statement has no options.
```

2.4.2 Variable Parameters

Any word shown in lower case in the general form is a variable parameter; the word used indicates what kind of information is

required. The limits on the value of the variable depend on the statement in which it is used.

Example:

```
CPMU PIN expression;  Connect the PMU to the pin.
```

The word expression indicates a number or variable must be entered and not the ten characters that comprise the word expression. CPMU PIN 3; is a legal statement.

2.4.3 Required Parameters

Brackets are used to enclose a set of parameters where one, and only one, of the parameters in the set must be used. The parameters in the set are separated by the slash character. An underlined parameter identifies the default case if the entire statement is omitted.

Examples:

1. ON [FCT/DCT/TRIP], label;

   Either FCT, DCT or TRIP must be specified.

2. ENABLE/DISABLE RELAY;

   Either ENABLE or DISABLE must be used in this statement. If the entire statement is omitted, however, the DISABLE RELAY state is assumed.

2.4.4 Optional Parameters

Parentheses are used to enclose a set of optional parameters where, at most, one of the parameters in the set may be used. A slash is used to separate parameters in the set. An underlined parameter identifies the default case if only that parameter is omitted.

Examples:

1. MEASURE VALUE (,LOG);

   The ", LOG" is optional in this statement. Note that if LOG is used, the comma must be used and vice versa.

2. ... (, RNG2/, RNG3);

If the range is programmed, either ,RNG2 or ,RNG3 is used. If the range parameter is omitted, RNG3 will be assumed.

2.4.5 Syntax Characters

The brackets and parentheses characters can be used to define binary pin patterns for functional test statements (paragraph 11.4.1). In those patterns where the brackets and parentheses are shown, they are required and are not part of syntax notation. Parentheses are a required part of the input/output statements (Section IX); they are not part of the syntax definition. Brackets are required around the SIZE in array declaration statements.

Section III
Factor Operating Procedures

3.0 INTRODUCTION

A program written in FACTOR must be compiled before it can be executed in the Sentry 200 Test System. The compiler converts the FACTOR English-like statements into a program file of object codes. The object code is then used by the TOPSY system, which interprets and executes the program.

3.1 PROGRAM INITIATION

To initiate the FACTOR compiler, using the DOPSY monitor, the user must type:

// COMPILE [TTK/TTR/CR/] 'file name' [TTP/LP] [LIST/OBJ/LISTOBJ]

This command format is general and includes all possible options.

3.1.1 Input

The first group of enclosed options in the above command indicates that the compiler input may be specified from the keyboard (TTK), from paper tape via the teletype paper tape reader (TTR), from cards via the card reader (CR), or from a file on the disc ('file name') which was previously created as outlined in the User's Manual.

Not more than one of these options may be specified. The user may, however, elect not to specify any option, in which case the compiler will expect its input from the current principal input device (PID) assigned to DOPSY.

3.1.2 Output

The second and third groups of options, in the general format command, are used to specify the output desired.

The listing may be specified for either the teletype or the line printer. Again, no more than one option may be entered with the command. If no entry is made, the output, if any, will go to the

principal output device which is currently assigned to DOPSY. If LIST is selected, then source statements only are listed; if LISTOBJ is selected, then both source statements and their resulting object code will be listed. If either OBJ or LISTOBJ is selected, then the compiler places its translated program in working storage on the disc. Note the distinction between 'LISTOBJ' and 'LIST OBJ'.

A typical initiation command might be:

```
// COMPILE CR LP LISTOBJ
```

followed by a carriage return, if entered at the teletype. This command would read its source program from the card reader and produce both an object program in working storage and also a listing of the source statements and the interleaved object code on the line printer.

If a specified input and/or output device is not available, an error halt will be taken to 100B with the I/O device address in the accumulator.

When a program error is detected, one of two procedures is taken: (1) if the error is recoverable, i.e., if the compiler can continue, FACTOR will continue to compile and notify the user of further errors, (2) if the error is not recoverable, the DOPSY monitor will be called and an asterisk will be typed to notify the user that DOPSY is in control again.

Note that when parentheses or brackets are missing the up-arrow error position indicator may be placed within the error message text. Two parentheses or brackets are used in the text of the error message, since a single symbol might be obliterated by the up-arrow, making the message illegible.

3.2 INTERPRETER INTERFACING

FACTOR produces an object program which must be saved by the user if it is to be executed. Once a compilation has been completed return is made to the DOPSY system monitor with the compiled program in working storage. The user then has the option of correcting any errors in the source program and redoing the compilation, or if the program compiled error free, he may save the object program by creating a type "DATA" file on the disc:

```
// CREATE DATA 'file name'
```

(Refer to the S200 User's Manual for a description of the CREATE command). The user program may now be executed under the control of the TOPSY interpreter. TOPSY is called by typing // TOPSY,

followed by a carriage return. The operation of the program from this point, using the /. LOAD command, etc., is described in the Sentry 200 User's Manual.

3.3 ERROR MESSAGES

Most of the error messages issued by FACTOR are self-explanatory. They are listed below with some comment for clarification. The error messages are accompanied by an up-arrow "↑", where appropriate, to indicate the position in the statement text where the error was detected. The total number of errors is output to the POD at the end of compilation.

TEXT	DESCRIPTION
"VARIABLE NAME" ALREADY DEFINED	Notifies the user of a duplicate label definition within the same block, or a mistake in logic has been detected, i.e., using a variable as a scalar when it has been defined as an array.
DOUBLE DEFINED -- "VARIABLE NAME	Same as above.
SEQUENCE ERROR	An error has been found in the sequence numbers punched in columns 73-80 of the source card deck.
SS FULL	The compiler's capacity for the storage of symbols has been exceeded. (Reduce the number of symbols used.)
NW FULL	There are too many noise words.
WORK FULL	The program has a compound tail too large to be processed as one statement.
EXIT FULL	Same as WORK FULL.
DISC OVERFLOW	There is not enough space on the disc for further object program to be built up in working storage.
EXCESS BLOCK	The allowable maximum number of nested blocks has been exceeded. Blocks may be nested to a depth of 8 (including Block 0).
SYSTEM 2 ERROR	"Never-happen" error-return (e.g. PUTW E-0-F).

ERROR MESSAGES (Continued)

TEXT	DESCRIPTION
PROGRAM TOO BIG	The program exceeds FACTOR capabilities.
MISSING))	A left or right paren has been left out.
EXPRESSION SYNTAX	An expression has been written incorrectly.
MISSING]]	A left or right bracket has been left out.
MISSING NAME	An identifier should have been specified in this syntactical position.
MISSING NUMBER	A number should have been specified.
STATEMENT SYNTAX	A statement has been incorrectly written.
USE ERROR -- DEFINED USAGE -- [SCALAR/FOR/PAR/ ARRAY/FUNCT/SUBR/ LABEL/SS]	Incorrect usage of the applicable variable.
NUMBER SYNTAX	A number has been specified incorrectly.
INVALID TERMINATOR	An expected terminator or delimiter is incorrectly specified or missing.
I/O SPECIAL ERROR	The I/O control word has indicated an error.
END OF FILE INPUT	The input file has been exhausted without finding an END statement.
EXCESS VARIABLES	The allowable maximum number of variables per block has been exceeded or too many parameters in an EXEC statement. (Maximum number of variables/block is 127, except Block 0 which is 104. The maximum number of parameters in an EXEC statement is 63).
NUMBER EXCEEDS LIMIT	Number exceeds hardware capabilities.
ILLEGAL INSTRUCTION	Instruction is not applicable to S200.

ERROR MESSAGES (Continued)

TEXT	DESCRIPTION
LABEL NOT IN BLOCK 0	ON FAIL BRANCH label is not in Block 0.
FILE NAME ERROR	Incorrect file name was used with an INSERT statement.

Section IV
Expressions

4.0 INTRODUCTION

An expression is a grouping of one or more numbers, variables, and functions combined with arithmetic or Boolean operators and parentheses so as to represent a quantity or an operation. Note that a single number or variable is considered an expression by this definition.

4.1 NUMBERS

FACTOR accepts numbers in any of the three forms discussed below, viz: integers, decimal fractionals or exponentials. In all cases, numbers are converted to a floating point internal representation for manipulation in the computer (Refer to Appendix L). The range of allowable decimal numbers is:

$$2.7105 * 10^{-20} \leq |N| \leq 9.2228 * 10^{18}$$

where |N| means the "magnitude of N".

4.1.1 Integers

An integer is defined as a whole number, including zero. It will be interpreted as octal if it is immediately followed by a B. It will be interpreted as decimal if it is not followed by a B. It may be either signed (preceded by a + or -) or unsigned. If unsigned, it will be interpreted as positive.

The limits for decimal and octal integers are:

Form	Limits
decimal integer	-8388607 ≤ n ≤ 8388607
octal integer	-40000000B ≤ n ≤ 37777777B (in two's complement form)

The following integers

1) are acceptable:

```
0
4000000
+2361
-5
6B
```

2) are unacceptable:

```
4,000,000 (Commas not allowed)
10000000000000000000 (Too large)
125 B (Imbedded space between number
       and octal specifier)
```

4.1.2 Decimal Fractionals

A decimal fractional is any decimal number with a fractional part preceded by a period. These numbers cannot have a B (octal) notation. An attempt to use octal notation in combination with a decimal fractional results in an error message. Decimal fractionals may be signed or unsigned.

The following fractional numbers

1) are acceptable:

```
4.0
0.0
 .671
+.734650
-42.0
0.734650
```

2) are unacceptable:

```
4.      (A number cannot end with a
         period)
1.234B (A fractional number cannot be
        specified octal)
```

4.1.3 Exponentials

Exponentials are either decimal integers or decimal fractionals, followed immediately by an E and a decimal integer. They also may be signed or unsigned.

The following exponential numbers

1) are acceptable:

```
 0.1E2
+1.23E-5
 7E-3
-1.0E+5
-5E+2
```

2) are unacceptable:

```
0.1E2+   (The sign must come between E
          and its integer)
1E       (The exponent must have an  E
          number)
.234 E5  (Imbedded spaces are illegal)
2.BE2    (Octal numbers may not be ex-
          ponentially specified)
```

4.2 VARIABLES

In FACTOR, a variable denotes any quantity which is referred to by a name rather than by an explicit value. A variable may take on many values, one at a time, rather than being restricted to only one value. The values which are assigned to a variable may be any of the forms as discussed above. Also, variables may be either scalar or Boolean. A variable identifier may reference either a single variable or a set of variables considered as an array.

Two general classes of variables may be referenced by the Sentry 200 Test System FACTOR program; system variables and user variables. The values of the system variables are retained from one execution of the program to the next. The values of all user variables are lost at the end of the test sequence and are reset to zero at the start of the next sequence. The system global variables are saved for each station or test position. System global variables are automatically declared.

4.2.1 System Variables

The names and special uses of system global variables are:

Name	Use
SWITCH	May be set by a system command for program control.
VALUE	Contains the last value obtained by executing the statement MEASURE VALUE/NODE/VARIABLE
GLOB1 through GLOB20	General purpose.

4.2.2 User Variable Identifiers

Variable identifiers are names given to variables by the programmer. There are no restrictions imposed, except that the identifier must begin with a letter or the dollar sign and contain only letters, dollar signs, and digits. Identifiers can be of any length; note, however, that FACTOR retains only the first 8 characters. Consequently, the user must ensure that the first 8 characters are unique, or else an error message will be produced. Furthermore, system variable names or reserved words must not be used as identifiers. The following is a list of reserved words in the general FACTOR program language:

AND	INSERT
ASSIGN	LEQ
AT	LT
BEGIN	MEASURE
BLOCK	NEG
BRANCH	NEQ
BY	NOISE
CALL	NOT
CGEN	ON
CONF	OR
CONN	PAUSE
CLEAR	PGEN
CPMU	RD
DCL	READ
DISABLE	REM
DO	RESET
ELSE	SELECT
ENABLE	SET
END	SOCKET
EOR	SUBR
EQ	THEN
EXEC	THRU
FOR	WR
FORCE	WRITE
FUNCT	XCON
GE	XCONF
GOTO	XPMU
GT	
IF	

It is good practice, since the usage of identifiers is totally determined by the programmer, to use names (identifiers) that represent the meaning or use of a variable. TEMP, for instance, could be the name given to a working variable. COUNTER might be the name given a variable that is used as a general purpose counter, and so on. (This does not imply, however, that FACTOR attaches any significance to these names. They are purely artifical devices that aid the user's memory and make a program more intelligible.) Variables are assigned an initial value of zero. Arrays must be declared before the array is referenced. Refer to Section VI for variable and array name declaration information.

The following are acceptable variable identifiers:

```
A
CHISQUARE
ALARGEIDENTIFIER
A1B2C3D4
PHOENIX
```

The following are not acceptable variable identifiers:

123	(Identifiers may not start with a digit)
AB C	(Special characters, including blanks, are not allowed)
END	(Reserved words are illegal)

This is a general definition of identifiers, which holds for the other several types of identifiers used in FACTOR.

4.2.3 Scalar Values

The FACTOR variable in its simplest form is scalar. Scalars are defined as being nonarrayed, nonBoolean quantities. (Note that, as defined below, an array element may be a scalar value and/or a Boolean value). In addition, scalars may take on any legal numbered values. To use the scalar value which is currently assigned to a variable, the user writes the variable's identifier in his statements or expressions.

4.2.4 Boolean Values

Boolean values are quantities which when evaluated have a value of either one (true) or zero (false). Expressions involving Boolean operators (paragraph 4.6) can only take on a true or false value, i.e., a one, or a zero; thus expressions are not evaluated for any other absolute value. Whenever the user references the variable identifier the current Boolean value will be returned.

4.2.5 Array Values

An array is an ordered series of values which are grouped together positionally with respect to some variable identifier (usually the first array element identifier). The elements of the array are restricted to either signed or unsigned numbers (i.e. alpha values are illegal). FACTOR arrays are restricted to one dimension.

To obtain an array value, the user must follow the array identifier with an expression which is enclosed in brackets. The value of the expression is the subscript which tells FACTOR which element of the array is wanted. If the subscript is zero, i.e., A [0] for instance, FACTOR returns the array size. Any other value of the subscript will refer to the appropriate element in the array of values. For example, A[2] would reference the second element in the array A.

NOTE

The value of 'B' is limited to positive or zero, since the general result for a negative quantity raised to a power is a complex number. 'C' may be positive, negative or zero.

4.5 LOGICAL EXPRESSIONS

4.5.1 Logical Operators

The logical operators defined for the Sentry 200 Test System operate on full word integers. The logical operators are:

Symbol	Operation
AND	logical and
OR	inclusive or
EOR	exclusive or
NOT	one's complement

With two expressions or constants P and Q, for each bit of P matched with the corresponding bit of Q, the following truth table holds:

P	Q	P AND Q	P OR Q	P EOR Q	NOT P
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

The precedence order of the logical operators is:

NOT
AND
OR, EOR

Example:

Calculate a percentage and form an integer before printing -

```
PC = 100*X/Y;
PC = PC AND 177B;
WRITE 'PERCENT GOOD=', PC;
```

Prior to performing the logical operation the expression is evaluated and then fixed as an integer. The integer form is limited to 24 bits, therefore conversion underflow or overflow from floating point to fixed format must be considered by the programmer.

4.5.2 Logical Expression Evaluation

Logical expressions are evaluated in order of operator precedence and from left to right when two or more operators of the same precedence exist.

Logical expressions are useful for specifying more then one option in a single variable. Using octal notation, suppose that the "hundreds" digit of the system variable SWITCH is used to select ABORT on first fail if the digit is non-zero, the tens digit is used to select a device grade, and the units digit is another quantity of interest. The logical AND operator can then separate this information as follows:

```
SWITCH AND 700B extracts the "hundreds" octal digit.
SWITCH AND  70B extracts the "tens" digit.
SWITCH AND   7B extracts the "units" digit.
```

Using statements from Section VII and XI, this could be applied as follows:

```
IF SWITCH AND 700B NEQ 0 THEN BEGIN

      ON FCT, ABORT;
      ON DCT, ABORT;
      ON TRIP, ABORT;

                                END;
```

where ABORT is a label at the end of the program.

4.6 BOOLEAN EXPRESSIONS

4.6.1 Relational Operators

Relational operators deal with the comparison of two logical expressions, arithmetic expressions, variables, or constants in any combination. The result of the comparison is either true or false. The relational operators are:

Symbol	Operation
EQ	equal
GE	greater than or equal
GT	greater than
LT	less than
LEQ	less than or equal
NEQ	not equal

For example, consider the following comparison:

A LT B

If the values of the two variables are 16 and 25, respectively, the comparison is effectively:

16 LT 25,

which is a true statement.

Consider now:

B LT A, i.e. (25 LT 16)

This is of course a false condition. Similar examples could be given for EQ, LEQ, GT, NEQ and GE.

All relational operators have the same precedence level.

4.6.2 Evaluation of Boolean Expressions

A Boolean expression uses logical and relational operators and defines whether a true or a false condition exists.

The order of operations for Boolean expressions depends upon the precedence values of the operators. The precedence order is:

a) relational operators (LT, LEQ, EQ, GE, GT, NEQ),
b) NOT,
c) AND,
d) OR and EOR

The following are examples of Boolean expressions:

1. A (where A is either true or false)
2. A OR B EOR C
3. A GE B OR A LT C

In example 2, the expression is evaluated from left to right. A is ORed with B and then the result is EORed with C. In example 3, the expression A GE B is evaluated for a true or false condition: the expression A LT C is evaluated: the results of these two operations are ORed together.

4.7 MIXED EXPRESSIONS

FACTOR allows free mixing of arithmetic and Boolean expressions, without adhering to pure Boolean values. It is the responsibility of the programmer to ensure that values in mixed expressions are valid integers when they are involved in a Boolean expression. Further, arithmetic operators take precedence over Boolean operators in mixed expressions.

Section V

Block-Command And Program Concepts

5.0 INTRODUCTION

Blocks are groups of program statements between the delineators, BLOCK (or BEGIN) and END. Local variable storage and local labels do not exist outside of the block which contains them; in other words, local variables or labels cannot be referenced (accessed) outside of their parent block. A block, then, is an independent compilation, since a program can consist of several completely independent blocks.

5.1 LABEL

A label is an identifier similar to a variable identifier except that it is always followed by a colon and it refers to a statement. The complete definition and all restrictions which apply to labels can be found in paragraph 4.2.2, which describes variable identifiers. The following is an example of a label identifying an assignment statement.

```
TOTAL:  A = 1 + 2 + 3 + C/D;
```

5.2 BLOCK CONCEPT

A block must have a beginning and a closing statement. In addition, a block can be either independent or dependent. The following information will describe the block and its function.

5.2.1 Establishing the Block

A block is established in two ways:

1) It may be opened directly by writing the command BLOCK and closed by the command END; Compare this with the command BEGIN (Paragraph 7.4) which is very similar in concept.

2) A block will also open following the FUNCT and SUBR commands. These commands are discussed in Section VIII.

The initial BLOCK declaration need not be specified bacause FACTOR will assume a BLOCK 0. For clarity, the user may spell out the initial block in his listing. It must have an END statement, however.

5.2.2 Nesting Blocks

Blocks do not need to be completely independent. One of the easiest methods of introducing block dependence is by "nesting" one block within another. This results in the execution of the inner block being dependent on the execution of the outer block. Nesting can occur up to eight levels on the Sentry 200 implementation. Nesting is illustrated in the following example:

```
BLOCK
     BLOCK
     END;
     BLOCK
          BLOCK
          END;
     END;
END
```

The inner block of a nested set is considered part of the enclosing blocks. Another form of dependence is that of global variables. A global quantity is one that is accessible to a block, but is not necessarily contained in (i.e., is not local to) that block. Variables and labels can be either local or global. This is illustrated in the following example:

```
BLOCK
L: DCL A, B/10/;
     BLOCK
     DCL A,C;
     END;
END
```

Each block in the above example contains the local variable A. The A in the inner block cannot be accessed from the outer block and vice versa. The variable B in the outer block is accessible from either block, but the variable C can be accessed only from the inner block. In this example, then, B is a global variable, but C and the two variables A are all local.

Note, that if there had been a label L in the inner block, any reference to it within the inner block would have used that L rather than the one in the outer block. Any nested set of blocks establishes a block context; i.e., a relationship of local and global variables. From the example, it can be seen that a

reference to a variable or label is associated with the occurrence of that identifier or label in the same block, if it is present. If it is not, then the next outer block is examined, etc.

It should be noted that it is possible to make variables global from within a nested block in FACTOR by simply never declaring the variable as local. When the nested block is closed, the variable, and any residual value is relocated to the next outer block, where it may now be considered as global to any further nesting. When this outer block is closed, if it was nested, the variable will again be relocated to the next outer block and so forth until block 0 is closed.

The fact that the declaration of a variable within a block makes it local has important implications for the FACTOR user. After leaving a block, i.e., closing it with an END command, the values of all variables declared within the block, and thus made local, are lost. Upon re-opening the block, the values of these variables are initialized to 0.

Section VI
Variable Declaration And Value Assignment

6.0 INTRODUCTION

As described in paragraph 4.2, variables may be used in expressions without giving them initial values or by declaring them. If they are not declared, they will be assumed to be scalar. If they are not given an initial value, they will be automatically given an initial value of zero. Variables may be declared and assigned values at any point in the program.

A variable may also be used as an array reference, but then it must be declared. Thus, a declaration (DCL) statement must always be executed before any references are made to the declared arrays. If this rule is violated, TOPSY will indicate the programming error with a terminal error at run time (see Appendix F).

If the DCL statement is executed more than once in a currently open block, all but the first execution will be ignored; however, a value assignment will always occur at every execution. This point must be emphasized. It means that the evaluation of an array size will occur once only: at the first DCL for that array. However, values will be assigned for every DCL specified.

6.1 DCL

The DCL command is used to reserve storage for variables, assign initial values, and to make a variable local to the block in which it is declared.

Two types of variables may be declared: scalars and arrays.

6.1.1 Scalar Declaration

The general formats of the scalar declaration are as follows:

```
DCL   V1;
DCL   V1, V2,.....VN;
DCL   V1/VALUE 1/,V2/VALUE 2/,...VN/VALUE N/;
DCL   V1, V2/VALUE 2/,V3...VN;
```

V1...VN stand for variables number 1 through N. VALUE 1...VALUE N stand for single signed or unsigned numbers which declare the value of a variable. When declared without a value, the variable is set equal to 0. Multiple declaration and assignment can be made with one statement. As shown in the last two examples, each variable of a multiple declaration can be optionally assigned an initial value.

6.1.2 Array Declaration

The general formats of the array declaration are as follows:

```
DCL   V1[A1SIZE];
DCL   V1[A1SIZE],...VN[ANSIZE];
DCL   V1[A1SIZE]/AE1,..AEM/,...,VN[ANSIZE]/AE1,..AEM/;
DCL   V1[A1SIZE],V2[A2SIZE]/AE1...AEM/,...,VN[ANSIZE];
```

The formats are similar to those for scalar declaration; however, the array identifier, V1, requires an argument to specify the number of elements, i.e., the size of the array. This quantity is enclosed in square brackets. The size is specified by an expression which allows it to be variable or fixed. The evaluation of array size and allocation of storage is performed by TOPSY at run time. The array size of necessity is automatically truncated to the nearest integer.

The elements of an array may be optionally assigned initial values when declared. The assignment is specified by the terms AE1 through AEM as shown above; M is the size of the array. The initial values of the elements are restricted to signed or unsigned numbers. (Alpha is illegal.) If the size and number of initial value assignments do not agree, the missing (trailing) elements are set to zero. If too many elements are specified, a run time error will occur.

NOTE

> The distinction must be drawn between the value in square brackets used in a DCL statement, where it represents the array size, and the value in square brackets used in a non-DCL statement, where it specifies the array element desired.

Examples:

```
DCL   ARR[10]; REM ARRAY SIZE = 10 ELEMENTS;
FOR J = 1 THRU 10 DO
ARR[J] = 2*J; REM COMPUTE ARRAY ELEMENT VALUES;
FIFTH = ARR [5]; REM ASSIGN VARIABLE FIFTH
        THE VALUE OF THE FIFTH ARRAY ELEMENT;
```

6.2 ASSIGNMENT STATEMENT

The assignment (or replacement) statement is the most fundamental of all FACTOR statements. It takes the general form:

```
variable = expression;
```

This command results in the replacement of the value of the variable on the left by the value of the expression on the right. (In general, it is not an equation, since the variable on the left may form part of the expression on the right).

Thus the statement:

```
A = A+B;
```

means: take the value of the variable A, add the value of the variable B to this value and replace the value of the variable A with the result.

Section VII
Control Statements

7.0 INTRODUCTION

Control statements are used to direct the flow of the program by a transfer of control to different part of the program. Such a transfer may be imperative (e.g., GOTO) or conditional (e.g., IF).

The control statements to be discussed in this section are PAUSE; GOTO; IF; BEGIN; and FOR.

7.1 PAUSE

The PAUSE statement is used to stop the execution of further statements until START is depressed. The format for this statement is:

```
PAUSE expression;
```

Prior to halting, the value of the expression is evaluated and printed on the Primary Output Device.

This statement can be used to provide a programmed halt when debugging new FACTOR programs.

NOTE

Refer to the S200 User's Manual for instructions for using the monitor mode PAUSE command.

7.2 GOTO

A program is essentially a series of statements which, in general, are executed sequentially, and thereby accomplish a particular task. The computer thus operates one step at a time. However, it is essential to be able to enter or leave the sequence of instructions at any desired point.

This is the function of the GOTO statement. When executed, a GOTO statement always changes the program flow from the statement

immediately following it to the one specified in the GOTO statement.

The simplest form of the GOTO statement is:

```
GOTO        label;
```

where label as defined in paragraph 5.1 specifies the statement to be executed next.

The label must be in the same block or a lower block. In other words, it is not permissable to jump into a subroutine from the main block or from another subroutine.

7.3 IF

The GOTO statement provides one method for altering the sequence of statement executions. It is also essential to be able to change the sequence of execution based on what happens as the program executes, i.e., a conditional change of execution. This is the principal use of the IF statement.

The simplest form of the IF statement is;

```
IF relation THEN Statement-1;
Statement-2;
```

Upon execution of the IF statement, if the relation is true, statement-1 is executed followed by statement-2 (unless statement-1 carries control away from statement-2). If the relation is false, statement-2 is executed instead.

For instance:

```
IF A EQ 3 THEN GOTO LABEL;
```

will, if it is true that A is equal to 3, cause the sequence of execution to change to the point in the program having a statement labeled LABEL. If A is not equal to 3 the next sequential statement, after the IF statement, will be executed.

The true-false nature of the above relation gives a clue to the second general form of the IF relational clause. It is:

```
IF Boolean-expression THEN statement;
```

where Boolean-expression is any legal expression as defined in paragraph 4.6.

Suppose, for example, that we wish to continue doing something until the value of A and B, two variables we are manipulating, both become less than some terminal value, 0. We could make this decision and monitor the values of A and B with one IF statement as follows:

```
IF A LT 0 AND B LT 0 THEN GOTO DONE;
```

The process we wish to continue doing immediately follows the IF statement.

In all cases of IF statement usage, the following THEN can introduce any type of FACTOR statement.

7.3.1 The Conditional ELSE

The simple IF statement is one which causes a statement to execute if a relation or Boolean expression is true and skips statement execution if the relation or expression is false. A complete conditional statement does more. It specifies a second statement to be carried out if, and only if, the relation or expression is false.

The general forms are:

```
IF relation THEN S1 ELSE S2;
IF Boolean expression THEN S1 ELSE S2;
```

where S1 and S2 are any two statements. When the result of the IF operation is true, S1 will execute and S2 will be ignored. When the result is false, S1 will be skipped over and S2 executed. S2 may be any statement, including another IF statement. This nesting of conditionals can go to any depth.

Example:

```
IF relation THEN S1 ELSE IF relation THEN S2 ELSE S3;
```

7.4 BEGIN

FACTOR allows the grouping of a series of statements within the bracket commands BEGIN and END;. The command, END;, must immediately follow the last command executed. Note that the semicolon is an integral part of the END; bracket. The purpose of this command is to allow a compound statement to follow the THEN of the IF command.

For example:

```
IF relation THEN
    BEGIN
        statement;
        statement;
        statement;
    END;
```

The above example is an example of a compound statement, and is an acceptable method of writing the IF statement. The statements between BEGIN and END; are legal and, as far as the IF statement is concerned, are considered to be one statement. In other words, if the relation is false the statement after the BEGIN-END; block is executed next. Any FACTOR statement may be part of the compound statement, including another IF statement.

7.5 FOR

One of the techniques most widely used in programming is that of the program loop. This is the repetition of some program statement or statements over and over with different parameters. The FOR statement is the looping mechanism within FACTOR.

The general format of the FOR statement is:

```
FOR variable = expression THRU expression DO statement;
```

where variable, expression and statement may be in any legal form defined in this manual.

Several statements may be included in the DO loop portion of the FOR statement by specifying a compound statement with BEGIN and END;

An example of a typical loop is one designed to solve the following problem. Suppose it is desired to set the elements of an array to zero. This can be achieved with the IF statement sequence of statements, in the following

```
      I = 1
NEXT: A[I]=0;
      I = I+1
      IF I LE A [0] THEN GOTO NEXT;
```

but it is better accomplished with the statement:

```
FOR I = 1 THRU A[0]DO A[I]=0;
```

The simple FOR statement provides an index value which has three important features:

1) an initial value,
2) an (assumed) increment of +1,
and 3) a limit

In the above example, I takes on the values 1, 2, 3, ..., A[0], where A[0] is the last value corresponding to the size of the array.

The implementation of the FOR causes the address of the index, the increment and the limit to be evaluated each time the loop is executed. Therefore, caution must be exercised within the loop when changing values that might affect this evaluation.

The loop will be executed the number of times specified by the initial value, limit, and increment. (This may be zero.) Also, there is no restriction on transfers of control into or out of the loop. When the loop has finished its specified number of executions, control will pass to the next sequentially executable statement, unless this sequence is interrupted by a statement in the DO loop.

In the above discussion an automatic increment of +1 from the initial value to the final value was assumed. There is a second form of the FOR statement which uses BY; this allows the user to specify some value which will be used as the increment.

It should be pointed out that because the values may be all positive, all negative, or mixed positive and negative, the user should consider the range of possible values he expects. It makes sense to go from a negative number to a more negative number in negative increments or from a positive number to a negative number by negative increments. Going from positive to more positive or negative to positive, the increment must be positive. Going from -2 to +6 in increments of -2, as from +8 to +2 in increments of +2 is not logical and will be flagged as errors.

Caution must be exercised when using fractional values for the index, since it is possible to introduce a step error. For example, a statement such as:

```
FOR I = 0 THRU 1000 BY 0.1 DO I = I + 1;
```

may operate the DO statement more than 10,000 times because of a rounding error in the floating point conversion of 0.1.

Section VIII
Subprograms

8.0 INTRODUCTION

This section discusses the function and operation of subprograms, the calling of subroutines, and how to make a function out of a subprogram.

8.1 SUBPROGRAMS

Programs frequently have groups of statements which can be used several times with different parameters. Of course, the required statements could be duplicated wherever they are needed in the program, but to do so is error prone, and wastes user time and machine storage. Therefore, it is desirable to be able to write statements so that they may be executed from any point in the program with a different set of parameters each time they are executed. The subroutine statement makes this possible.

8.1.1 SUBR

The general formats of the subroutine declaration are as follows:

Format One:

```
SUBR        Identifier;
            statement 1;
            statement 2;
                .
                .
                .
            statement n;
END;
```

Format Two:

```
SUBR        Identifier (VI1, VI2,...,VIN)
            statement 1;
                .
                .
                .
            statement n;

END;
```

The identifier after the SUBR command is used to reference the subroutine from the main program. The statements within the subroutine will not be executed until the subroutine is called from the main program by the SUBR identifier. Any number of statements are allowed within the subroutine.

The END; statement is necessary because the SUBR command effectively opens a new block. When it is completed, it must be closed. The END; indicates the last statement in the subroutine.

Format Two indicates another important feature of the SUBR statement. The terms VI1 through VIN represent variable identifiers 1 through N. They are enclosed in parentheses and indicate to FACTOR that whenever a reference is made to this subroutine, the reference will specify actual values which are to be substituted at specific places within the subroutine body. These identifiers are called formal* parameters. There is a one for one correspondence between the position of the formal parameters and the position of the parameters or values used in the call. For reference and further explanation, see the next section on the CALL statement. The manner in which values transferred to the subroutine are used in the subroutine's statements is illustrated in the following example:

```
SUBR TOTAL (VI1, VI2, VI3);
    VI1 = VI2 + VI3;
END;
```

When the above subroutine is referenced:

```
CALL TOTAL (A1, A2, A3);
```

the values passed to it, obtained from the actual parameters A1, A2 and A3, will positionally replace VI1, VI2, VI3 and they will be used in the arithmetic expression and assignment. The value of the variable represented by VI2 will be added to that represented by VI3 and the total will be assigned to the variable represented by the formal parameter VI1.

As an example of a subroutine with no formal parameters specified, we will use a similar statement as follows:

```
SUBR TOTAL2;
    A = B+C;
END;
```

When TOTAL2 is called, the current values of the variables B and C are added and the total is assigned to the variable A. In this case A, B and C are not formal parameters. They are working

variables with current values in the outer blocks to the SUBR statement block.

Because the subroutine forms a new block, it must be remembered that any variables which are declared in the subroutine will be local.

NOTE

The word 'dummy' is often used in this context interchangeably with the word 'formal'.

8.2 CALL

A subroutine is executed by using a CALL statement, which can be placed at any point in the program where the programmer can legally place a statement. The general formats are as follows:

```
CALL SI;
CALL SI (expression 1, expression 2,...,expression N);
```

SI is the identifier of the subroutine block to be activated. The values, changed by the subroutine statements and by any other task executed, will be accomplished as if the subroutine's body of statements has been placed at the point of the CALL statement. Then, the next sequential statement, following the CALL, will be executed.

The expressions are evaluated at the time of the execution of the call and, therefore, will remove many constraints which are ordinarily placed on the CALL values. As the subroutine statements are executed, the value of expression 1 will be used wherever formal parameter 1 was used. The same will hold true for other formal parameters and expressions. The only restriction is that when a formal parameter receives a result, the corresponding actual parameter should not be an expression but a single variable identifier.

8.3 FUNCT

The subroutine call, when encountered in the program execution, brings the subroutine statements into action to accomplish whatever processing is specified (ordinarily assigning new values to outer block variables). Control then usually passes to the next sequential statement.

When only one variable is assigned a new value, as a result of executing a subprogram, the call can be simplified by making the

subprogram a function. When FUNCT is used, simply writing the identifier of the function will cause its statements to execute. However, the identifier now represents a value that may be used wherever a variable is legal. Thus, it is as though the function call represents a variable of the same name.

The general form of the function statement is:

```
FUNCT      identifier (VI1,...,VIN);
   statement 1;
        .
        .
        .
   statement n;

END;
```

The format, except for the FUNCT command portion, is exactly like the SUBR statement. It also defines a new block and it will be called by the identifier, following the FUNCT. The difference is in the way the function is activated and also that it always returns a value for the function identifier. If no assignment of a value to the function identifier is made within the statements following FUNCT and before END; is encountered, a value of zero will be returned. Assignments of values to a function, which are external to the function declaration statement are illegal, i.e., the function name must not be used on the left hand side of an assignment statement.

The function identifier is not local to its function block and therefore must not be declared within the function statements.

As with a subroutine call, the FUNCT statement, with its compound tail of statements, is not executed until the function identifier is used in an expression.

8.3.1 Function Call

As stated, using the function identifier as a variable identifier in an expression will cause the function statement block to activate and to return a value for the identifier. Note the following example:

```
FUNCT TOTAL3 (VI1,VI2,VI3);
      TOTAL3 = VI1+VI2-VI3;
END;
```

When it it desired to reference a value by executing the above function, its identifier is used as follows:

```
NEWTOTAL = TOTAL3(A,2,B+C)+(A-B);
```

The function TOTAL3 will be evaluated using the current value of A for formal parameter VI1, 2 for formal parameter VI2 and the current value of B+C for parameter VI3. This overall value will then be added to the value calculated for the sub-expression, A-B, using current values for A and B. Finally, the end value arrived at will be assigned to the variable, NEWTOTAL.

The same symbol must not be used for formal and call parameters.

It is worth considering a further example to illustrate the range of versatility of a function. Assume there are two sums to evaluate:

Sum 1:

$$A = \sum_{B=1}^{10} B^2$$

Sum 2:

$$C = \sum_{D=1}^{E+H} E+F[G]/D$$

Because expressions are allowed in the function call, one general function statement could be set up to handle both sums as shown below:

```
FUNCT     TOTAL (W,Z,Y,X,R);
    W = 0;
    FOR Z = Y THRU X DO W = W + R;
    END;
```

To evaluate Sum 1, the call is written as:

```
TOTAL (A,B,1,10,B*B)
```

Sum 2 could also be evaluated by the same function with the call;

```
TOTAL (C,D,1,E+H,E+F[G]/D)
```

The effective result of the two function calls is shown below:

```
Sum 1:
    A = 0;
    FOR B = 1 THRU 10 DO A = A+ B*B;
    END;

Sum 2:
    C = 0;
    FOR D = 1 THRU E+H DO C = C+E+F[G]/D;
    END;
```

The parameters of a function call can also include other function calls. In fact, a function may even call itself recursively. For instance, a factorial could be calculated as follows:

```
FUNCT   FACTORIA (N) ;
    IF N = 1 THEN FACTORIA = 1
      ELSE FACTORIA = N * FACTORIA (N-1);

    END;
```

This is an example of recursion, the use of a function within the same function. The user should remember that the number of recursive calls is determined (and limited) at run time by the size of core. Also, because new blocks are opened whenever a call activates a function, this recursion of the function causes nesting. Since nesting can go only 8 blocks deep, this example could easily exceed this depth in most cases.

8.4 EXEC

An assembly language program is executed by using an EXEC statement. The general formats are as follows:

```
EXEC PROGRM;

EXEC PROGRM (parameter 1, parameter 2 .. parameter N);
```

PROGRM is a FACTOR identifier which is also the name of a COREIMAGE file on disc. Each parameter is evaluated at the time of the EXEC, and may be a global variable, variable, array name, array-element, function, formal parameter or arithmetic expression. A maximum of 63 parameters is allowed.

After the assembly language program has been written, it is assembled and an object file is created from which a COREIMAGE file is created at a particular origin. This must be done either under the system job or under the same job used to run the FACTOR

program. The origin, (which must also be the entry point of the program) must be greater than 27770B. The difference between the location of the top of core and the origin, is considered to be the length of the program.

The A.L. program is brought into core at its origin location. All index register values are saved, and then the index registers are set to the values described below. Control is passed to the A.L. program. The A.L. program may return normally after executing, or it may take the ABORT exit causing a terminal error message to be printed. After a normal return the index registers are restored and the state switches reset. What occurs next is a function of the length of the A.L. program, which is discussed in the next paragraph.

It is highly desirable to keep a short A.L. program in core after execution so that it may be repeatedly executed without having to access the disc each time. However, since a long A.L. program occupies the same core space as the FACTOR program, it is desirable to remove it from core to minimize the number of disc accesses required for executing the FACTOR program.

Therefore, there exists a system constant, (currently set at 10000B) which can be easily modified to fit the needs of each installation. If an A.L. program is longer than the system constant a disc access will be made after execution, allowing the core area (to the top of core) to be used to store the FACTOR program. If an A.L. program is shorter than the system constant it will remain in core until a different A.L. program is EXECuted, and its space will not become available for FACTOR program storage.

8.4.1 Writing the A.L. Program (See also the FST-1 Assembler Manual (Part #67094951) and Appendix L of this manual)

The entry point must be the first location in the program.

```
ENTRY PROC  0
```

Normal return is

```
BRU* ENTRY
```

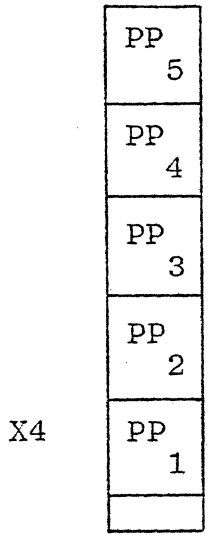
An error exit is provided and should be coded as follows:

```
AOM ENTRY

LDA mess. no.

BRU* ENTRY
```

where 0 $\leq$ mess. no < 900. This will cause a terminal error message (mess. no. + 100) to be printed, and the FACTOR test program will be aborted.

8.4.2 Referencing Parameters

In order to reference the A.L. program parameters, the working stack contains pointers to the values, with Index Register 4 (X4) pointing in front of the first parameter location and Index Register 3 containing the parameter count. Thus, for a routine with five parameters the working stack is as follows:




```
LDA* 1, 4 will get the value of the 1st parameter

STA* 1, 4 will store a new value into the 1st
          parameter
```

If the second parameter is an array of size 10

```
LDA* 2, 4

LXA 5

LDA 10, 5  will get the value of the 10th element of
           the array.
```

8.4.3 Changing the FACTOR Program

On entry to the A.L. program, Index Register 5 contains the core location of the next word of the FACTOR program, and Index

Register 2 contains the location of the PMF header for the FACTOR program.

The EXEC PROGM statement could be preceded by an ENABLE ACCESS statement allowing a maximum number of statements to be changed. In order for any statement modification to be effective PROGM must be a "short" A.L. program.

Since this technique is potentially dangerous, great care should be taken when utilizing it.

Index Register 6 contains the address of a table of eight (8) words. These eight words are the starting addresses of the block headers for the eight (or fewer) active block levels.

Index Register 7 contains the entry point address of the A.L. program.

8.4.4 Input/Output

I/O may be accomplished in the A.L. program using the system I/O drivers. However, the system I/O drivers may not be called directly, but must be referenced through their GLOBAL addresses.

In order to:	Code must be written:
CALL TTRIO	BSM* 520B
CALL TTPIO	BSM* 512B
CALL DISCIO	BSM* 554B
CALL LPIO	BSM* 543B
CALL CRIO	BSM* 553B
CALL MTIO	BSM* 552B
CALL TAPIO (FACTOR tape program)	BSM* 541B

Section IX
Input/Output Statements

9.0 INTRODUCTION

In Section III the input of source statements to the compiler and the output of compiled data statements from the compiler were discussed in detail. Section IX deals with READ and WRITE statements that control data-flow in and out of the computer during execution.

The statements READ and WRITE described in this section use the following syntax notation.

1) Outer parentheses '(',')' are used to enclose items that are optional; at least one item must be selected; items are separated by slashes; '/'.

2) A 0 indicates that none of the elements of the set need be chosen. When none are selected, the system assigns the current DOPSY Primary input or output device.

3) A file name identifier is shown in lower case letters and is enclosed by double quotation marks.

9.1 READ

The general formats of the input statement are:

```
READ( (CR) / (PID) / (FDIF) /(EIR) / (TTK) / (TTR) / (MTR)
"name"/0)V1,V2,...Vn;

READ (BCR) V1, V2,...Vn;

READ( (CR) / (TTK) / (TTR) / (PID) / (FDIF) / 0) Vi, &Vj ..&Vn;
```

The items enclosed by parentheses are peripheral devices defined as follows:

Primary Input Device	(PID)
Card Reader (Binary mode)	(BCR)
Card Reader	(CR)
Teletype/Video Terminal Keyboard	(TTK)

Teletype Reader	(TTR)
Magnetic Tape	(MTR)
External Interface Reg	(EIR) -See Paragraph 11.8
Disc Input File	(FDIF)

When magnetic tape, MTR, is specified the statement must include a file name which is enclosed by double quotation marks. The file name syntax is defined in the same manner as Identifiers (see paragraph 4.2.2).

Magnetic tape file "names" are used to uniquely identify data segments on the tape. These names are assigned with the WRITE statement (Appendix J).

The terms V1 through Vn may be any legal variable identifier, including arrays. As the input numerical data is read from the peripheral, it is assigned to the specified variable(s).

The input data or literal variables (variables preceded by '&') must be of the form:

$C_1C_2C_3C_4$ for a simple variable

$C_1C_2C_3C_{4n}..C$ for an array of size n

where the

C belongs to the FACTOR character set.

All of the characters must fit on one card (or one record of a different media). The characters are converted to TRASCII and stored into the variable without further conversion. C must appear in the first column of the card. This capability has not been implemented for magnetic tape.

When values of an array are to be read, they must be separated by at least one space (for TTK, TTR, CR, FDIF, PID). More than one card may be used to enter these values. All numbers following the last array element number on a card are ignored. Note that the first value for each new variable identifier must start on a new card.

When BCR is specified as the input device, the variables must be arrays of size 80 or larger. Each column of the card is interpreted to be a binary number between 0 and 7777. The value is converted into floating point format and assigned to the corresponding element of the array. Only 80 values will be read into the array regardless of its size. The remaining elements of the array (if larger than 80) may be used for any other reason desired.

When the input peripheral is the magnetic tape unit, the tape is searched forward until the file "name" is located. The numerical data from this file is read and assigned to the variables V1,...etc. as specified by the READ statement. For magnetic tape, the variables must be arrays which have no less than 7 elements. The maximum array size is limited by the amount of core memory available when the array is declared. It is recommended that arrays be no larger than 512 elements. Appendix J gives a detailed description of the magnetic tape operation and responses to the READ (MTR) and WRITE (MTW) statements.

When the input is a disc input file, the information is sequentially read from the disc and stored in variables V1, through Vn. (No formatting will occur.) The Disc Input file (FDIF) must have been opened and each READ continues processing the file where the previous READ left off. The assumption is made that the file is composed of records written by a FACTOR WRITE (FDOF) statement, and therefore consists of floating point numbers and alphanumeric text.

If an attempt is made to read beyond the information written in the file, a flag will be set, and control will be transferred to the statement label, which appears in the ON DIFEOF statement. If no ON DIFEOF statement has been encountered, a terminal error message #68 will be issued. If the DIF is not open, terminal error message #67 will be issued.

9.2 WRITE

The formats for output statements are:

```
WRITE ((EIR)/(POD)/(FDOF)/(TTP)/(LP)) Vi, 'Si',&Vj,'Sj',
      col V1, col 'S1', ...Vn;

WRITE (MTW) "name" V1, V2, V3, V4;
```

Where: Vi...Vj...Vn...V1...V4 are legal variable identifiers including arrays which may occur in any sequence. Si, Sj...are strings of alphanumeric characters,

col is a numeric column number between 1 and 80, enclosed by slashes, e.g., /10/

The items enclosed by parentheses are peripheral devices defined as follows:

Primary Output Device	(POD)
Teletype Printer/Punch,VKT	(TTP)

Line Printer...	(LP)	
Magnetic Tape...	(MTW)	
External Interface Register...	(EIR)	-See paragraph 11.8
Card Punch	(CP)	
Disc Output File	(FDOF)	

When magnetic tape (MTW) is specified the statement must include a file segment name which is enclosed by double quotation marks. When writing to magnetic tape, the variables Vi must be arrays which have no less than seven (7) elements and are recommended to be no larger than five hundred and twelve (512) elements as described in paragraph 9.1. Appendix J gives a detailed description of the magnetic tape operation and responses to the WRITE (MTW) statement.

When the teletype or line printer is specified as the output device there may be one or more strings of alphanumeric characters and one or more variables in a single WRITE statement. All strings must be enclosed by single quotes and must not contain semicolons (;). Multiple variables are separated by commas as are intermixed combinations of strings and variables.

When the output is to a disc output file, the information stored in the variables and the string will be output to the disc. (No formatting will occur.) As each word is output, it is added to that file which has been previously specified to be the Disc Output File (DOF). If an attempt is made to write beyond the end of the file, a terminal error message #69 will be issued and the test program will be aborted. If the DOF is not open, terminal error message #67 will be issued.

Numeric variables are output in one of three forms. If the numeric value of the variable is a positive integer whose magnitude is less than ten thousand (10000), it will be printed in the form 9999.

If the value is negative and of magnitude less than one thousand (1000), it will be printed in the form S999.

S is the sign (-) and the '9' terms are decimal digits. Leading zeroes in a positive number print as spaces.

Integers and non-integers whose magnitudes exceed nine-hundred and ninety-nine (999), or (9999), print in the following format:

S9.999EP99

where S again is the sign of the value and the '9.999' represents the decimal digits of the mantissa, the '99' represents the decimal digits of the exponent of the value and P is the sign (+

or -) of the exponent. The character 'E' prints as shown. For example, 8.979×10^{-6} prints as '8.979E-06'.

Numeric values as described above occupy a field of twelve (12) characters and are left justified within this field.

Literal variables (variables preceded by '&') are output as a string of characters. Four characters are output for a simple variable, 4n characters for an array of size n. The string of characters is followed by four blanks.

Strings of characters are printed as they appear in the enclosed quotes. The characters may be any of those in the character set (paragraph 2.1) excluding single quotation marks and semicolons(;). Leading spaces are printed according to the number of spaces following the single quote of a string. The total number of printed characters will be an integral multiple of four (4). (The restriction is automatically imposed at run-time with the addition of no more than three (3) spaces following the character preceding the trailing quote of a string.)

NOTE

> Where a 'data string' or a literal variable array is written under column format control, the entire string or array must fit on one line.

The maximum number of variables printed per line is five (5) with the first character field left justified, unless column formatting is specified.

When a variable is an array, its current values are printed five (5) per line beginning with array element one (1) left justified on the line.

More than five (5) variables can be specified per WRITE statement with the result that five values per line will be printed on all lines including the last, unless there are fewer than five values to fill the last line.

A single string of seventy-two (72) characters may be printed on a single line when the teletype/VKT is the output device. When the line printer is the output device, a string of eighty (80) characters* may be printed on one line. Single strings which extend beyond column seventy-two (72) of a punched card can be continued beginning with column one (1) of the next card, etc. When the single string exceeds the character counts described

above, the excess characters are printed on the following line. The teletype/VKT will ignore characters between column seventy-three (73) and eighty (80).

When variables and strings are intermixed in a single statement, without column formatting specified, the following output rule holds:

> If the count of characters printed on the current line exceeds fifty-six (56), then the first character of the next entity (either a variable or string) will be printed left justified beginning on the next line. Otherwise, it will be printed beginning on the current line and character position. Overflow to the next line will occur whenever the character count of a string exceeds the number of available characters on the line.

Example:

(FACTOR Code):

```
WRITE 'DATALOG';
WRITE '  ';
WRITE 'TEST#=',N,'          VALUE=',VALUE;
WRITE 'NODE=',PINN,         EXPECTED VALUE=',EV;
```

(Output Data):

```
DATALOG

TEST#= + 6                  VALUE=+1.200E-06
NODE=  - 0                  EXPECTED VALUE= + 2
```

In this example, the variables are N, VALUE, PINN, and EV. At the time the WRITE statements are executed, these variables had the following numeric values; 6, 1.2×10^{-6}, 0, 2, respectively.

Whenever column formatting is specified, the fourth character of the output will be right justified in the specified column. This capability is primarily oriented toward outputting integer values, which will always be right justified in the specified column. By specifying columns, the programmer can exceed 56 characters per line and 5 variables per line, and can concatenate values.

*Standard line printer - other line printers are available.

9.3 FACTOR DISC I/O

9.3.1 ON DIFEOF, Label

When a READ (FDIF) statement encounters the end of the written file, control is transferred to the statement Label.

9.3.2 RESET FDIF

This initializes the DIF pointer to the beginning of the DIF file.

9.3.3 Programming Conventions for Use With FACTOR Disc I/O

The following conventions are suggested to simplify use of disc I/O. It should be kept in mind that several different programs on different stations can be writing to the disc file and that the records from each station will be intermixed.

1) All WRITE FDOF statements in all programs should write the same number of words to the disc (ie. the record size should be constant).

2) Each WRITE FDOF statement should output at least 3 words of identifying information at the beginning of the record.

   Words 1 and 2 - Device name
   Word 3 - Station number

   Other identifying information could be included, such as the current date.

3) The READ/WRITE &Var capability can be used to read in identifying alpha information such as the device name, the station number, or the date. This information may then be output to the disc in a WRITE statement, or may be used in a data reduction program to compare the desired device name, station number, date, etc., against the corresponding characteristic read from the disc file.

EXAMPLES OF PROGRAMS THAT READ AND WRITE TO DISC

```
REM PROGRAM THAT WRITES TO DISC;

FACT1:      DCL DEVNAM [2], ARRAY [10];

    READ (CR) &DEVNAM, &STAT;
    .
    .
    .
    MEASURE VALUE;

    X1 = VALUE;
    .
    .
    .
    MEASURE VALUE;

    X2 = VALUE;
    .
    .
    .
    WRITE (FDOF) &DEVNAM, &STAT, X1, X2, ARRAY;
    .
    .
    .
    END;


REM DATA REDUCTION PROGRAM;
FACT2:      DCL DEV[2] DEVNAM[2], ARRAY [10], SUM;
    RESET FDIF;
    ON DIFEOF, AVER;
    I = 0
    READ(CR) &DEV;
    WRITE (LP) 'DEVICE IS' &DEV;
LOOP:  READ(FDIF) &DEVNAM, &STAT, X1, X2, ARRAY;

REM ONLY PROCESS DATA FOR CURRENT DEVICE;

    IF DEV[1] NEQ DEVNAM[1] THEN GOTO LOOP;

    IF DEV[2] NEQ DEVNAM[2] THEN GOTO LOOP;

    WRITE (LP) '    ',X1, X2;
    SUM=SUM +ARRAY[1];
    .
    .
    .
    GO TO LOOP;
```

EXAMPLES OF PROGRAMS THAT READ AND WRITE TO DISC (Continued)

```
AVER:   SUM=SUM/I;
        WRITE (LP) 'AVERAGE IS', SUM;
        .
        .
        .
        END;
```

Section X
Notational Statements

10.0 INTRODUCTION

Notational statements are used to enhance the readability and clarity of programs. Notational statements discussed in this section are NOISE and REM.

10.1 NOISE

The NOISE statement is used to define words that will make the FACTOR statements read like English sentences. Its general forms are:

```
NOISE WORD1;
NOISE WORD1, WORD2,...WORDn
```

The command NOISE is followed by at least one space and the noise word, or words which are separated by commas. After defining the noise words they are ignored by the FACTOR compiler. This provides a means for adding clarity to FACTOR statements. An example is shown below:

```
NOISE VOLTS, AMPS;

FORCE VF1 5.0 VOLTS;
```

Noise words are restricted to the format of identifiers, however, the reserved words listed in paragraph 4.2.2, as well as user-declared identifiers, are not allowed as noise words.

10.2 REM

The REM (remark) statement provides a means for adding commentary to a program listing. It is not executable and it can occur anywhere in the context of a program provided its format rules are followed. The general format is:

```
REM text;
```

where text is anything but a semicolon, END or ELSE.

For example:

```
REM  THIS SECTION PERFORMS VOL MEASUREMENTS;
```

As noted above, anything except a semicolon, END or ELSE is legal in the REM statement. All other elements of the character set will be positionally listed as located in the REM statement at compile time.

Section XI
Test Statement Formats

11.0 INTRODUCTION

Within this section, statements available for testing digital devices are described. Each statement description consists of a general form that gives the following information: time delay, definition, and, in most cases, an example.

The five major statement types discussed in this section are: Setup Statements, Programmable Power Supply Statements, Functional Test Statements, Precision Measuring Unit Statements and Miscellaneous Control Statements. The general forms of the statements use the syntax notation defined in paragraph 2.4.

11.0.1 Voltage and Current Ranges

Programmable voltage and current modules of the Sentry 200 Test System have from two to four ranges of operation. There are four possible ranges specified as follows:

RNG0
RNG1
RNG2
RNG3

NOTE

Current ranges depend on the hardware module: 102.3 milliamps is full scale in range 2 for power supplies, but is full scale in range 3 for the PMU (Refer to Appendix C).

The full scale value of voltage or current corresponding to each range depends on the statement and hardware module involved. Appendix C summarizes range numbers and their correlation to full scale value, resolution, statement and module.

11.0.2 Time Delay Dependent Instructions

Execution of a time delay dependent instruction is automatically delayed so as to allow the preceding test time dependent conditions to stabilize. For example, a DC measurement is

executed until all previously programmed power and voltage supplies have stabilized. The value of the time delay is determined by the type of tester instructions previously executed. Time delay dependent instructions wait for the status Tester Busy to be null.

Instructions that are not time delay dependent are executed without waiting for stabilization, even though they may initiate a Tester Busy status. This allows a series of programmed responses to essentially stabilize during the same time period, rather than in a sequence. Statements that are time delay dependent are listed in Appendix E.

11.1 SETUP STATEMENTS

This section describes statements that are typically used to initialize the tester prior to performing actual tests.

11.1.1 Set Delay

General Form:

SET DELAY expression (,DC);

Time Delay Generated:

0

Description:

A. Without DC Modifier

The value of the expression is loaded into the tester time delay register TD. This presets the delay time for subsequent executions of functional tests of the form SET F. That is, the expression plus 0.7 microseconds will be the value of the time delay between input stimulus execution and output comparator strobing (a SET F instruction). The 0.7 usec offset is equal to the typical driver response time to 62% of its final value from initial value. The delay used should allow for device under test response time plus 0.4 usec comparator response time. The Set Delay statement has a resolution of 0.35 microseconds and a maximum value of 5.734 milliseconds.

Example: Set the functional test delay for 350 microseconds.

SET DELAY 350E-6;

B. With DC Modifier

The value of the required time delay is scaled by FACTOR and loaded in the tester time delay register. The resolution is 0.35 milliseconds with a maximum value of 5.734 seconds. When a FORCE PMU, FORCE (VOLTAGE/CURRENT) or FORCE DELAY instruction is executed then the 'tester busy' status will remain "on" for the amount of time defined by the Set Delay, DC statement. Even if this instruction is not used a time delay generated by a fixed delay generator of 1.75 msec, if a range is changed, or 0.54 msec if no range is changed, will occur.

Example: Set the D.C. delay for 5 milliseconds

```
SET DELAY 0.005, DC;
```

11.1.2 Set Clamp

General Form:

```
SET CLAMP [POS/NEG/SYM/OFF]  number;
```

Time Delay Generated:

0.56 millisecond

Description:

The purpose of the SET CLAMP statement is to define a limited range within which PMU voltages may occur when the PMU is forcing a current into a load.

The PMU voltage CLAMP may be used when forcing voltage (to protect a device from a programming error) or when forcing current and sensing voltage.

POS means no voltages less than -0.7 volts will be allowed and NEG means that no voltages greater than 0.7 volts will be allowed. SYM allows both positive and negative voltages to occur. At the other limit, number defines the absolute value of the maximum PMU voltage which will be allowed.

There are 16 values at which the PMU may be clamped (positive or negative). They are:

1.5, 4.5, 7.5, 10.5, 13.5, 16.5, 19.5, 22.5, 25.5, 28.5, 31.5, 34.5, 37.5, 40.5, 43.5, 46.5, volts.

Any value will be accepted and the next higher (if not equal) clamp value will be selected. For example, if SET CLAMP POS 8.2; is given, the allowed voltages will be between -0.7 volts and 10.5 volts.

The actual clamp values produced by the hardware will be ± 1 volt ± 10% of the value specified. When sensing voltage, extra delay should be provided for clamp stabilization for each FORCE CURRENT statement executed.

11.1.3 On Program Branch Control

General Form:

ON [DCT/FCT/TRIP], label;

Time Delay Generated:

0

Description:

These statements establish program branch control on DC test failures (DCT), functional test failures (FCT) and power supply trip failures (TRIP). The label specifies the branch location. The label must be in the outermost block of the test program in all cases, i.e., the label must be in the main program, not in a subroutine. The branch instruction is canceled after the branch is executed.

The three branch conditions are programmed independently. Once a branch is taken, the ON statement is cancelled; a second failure of the same type will not alter program control unless the branch is reprogrammed. In the event of multiple fails on the same test, a power supply trip branch has priority over a DC or functional fail branch.

When a trip, functional or DC failure occurs at statement n and, if the corresponding ON statement has not been processed or if the branch has been used and not reprogrammed, program control will resume at statement n+1.

Example:

Set branch points to alter program control when functional, DC and power supply trip failures are detected.

```
ON DCT , DFAIL1;
```

```
ON FCT , FFAIL;
ON TRIP, CFAIL;

DFAIL1:  ON DCT, DFAIL2 ;
```

11.1.4 Enable Limits

General Form:

DISABLE/ENABLE [ILO/IHI/VLO/VHI] [GT/LT] number;

Time Delay Generated: 0

Description:

These instructions enable limit comparisons to be made on all programmed current/voltage operands prior to an instruction execution. If the operand fails to be within the LIMIT bounds, a system terminal error is issued. (See Appendix F.) Instructions with operands in the LIMIT bounds are executed.

The "pass" regions are established by the LIMIT pairs ILO, IHI for currents and by the LIMIT pairs VLO, VHI for voltages. The absence of ENABLE LIMIT instructions in a program allows all magnitudes less than the range limits to pass. The function of the limit comparison is to protect the device under test where source forcing parameters are calculated or may be unknown. Once the program is operational and safe parameters are known, these instructions may be removed for execution time efficiency.

NOTE

Specification of limits with these statements will increase test execution time.

Examples: Enable limits to "pass" voltages which are between the values of +5.0 and 0 volts.

```
ENABLE VHI GT 5;
ENABLE VLO LT 0;
```

Enable limits to pass currents which are between the values of +100mA and -5mA.

```
ENABLE IHI GT 100-3,RNG3;
ENABLE ILO LT -5E-3,RNG2;
```

DISABLE [ILO/IHI/VLO/VHI] nullifies the ENABLE limit comparisons invoked above.

11.1.5 Socket Identification

General Form:

SOCKET ID number;

Time Delay Generated:

0

Description:

This statement compares the value of the number with the identifier code of the performance board in the test socket. If the values are not equal, a system terminal error is issued and the program aborts. If the values compare, the program execution continues. The maximum Socket ID is 4095 (7777B).

Example:

Test the socket ID for a value of 4095.

SOCKET ID 4095;

11.2 PROGRAMMABLE POWER SUPPLY STATEMENTS

This section describes the statements that provide programmable control of the Sentry 200 Test System power supplies. The power supplies can force either voltage or current, monitor the resulting current or voltage, and cause a trip if programmed.

11.2.1 Force DPS Voltage Supplies

General Form:

FORCE [VF1/VF2/VF3] expression (,RNG2/,RNG3);

Time Delay Generated:

5.37 milliseconds or the programmed DC delay, whichever is greater.

Description:

This instruction forces the programmable voltage forcing supplies to the value specified. If the range is not specified, then the highest range is set. This instruction automatically connects the addressed unit to the test station load board.

If a current trip has not been programmed, an "impossible" trip of less than -1 ampere will be set, but not enabled. The trip register will be set to range 3.

The VF units will automatically disconnect under the following conditions:

(1) when the magnitude of the supply current is greater than 150 milliamps and the trip register is in range 2.

(2) when the magnitude of the supply current is greater than 1.50 amps and the trip register is in range 3.

Example:

Force units VF1, VF2, VF3, to +8, -5 and -30 volts respectively.

```
FORCE VF1 8, RNG2;
FORCE VF2 -5, RNG2;
FORCE VF3 -30;
```

11.2.2 Force DPS Current

General Form:

FORCE [IF1/IF2/IF3] expression (, RNG2/,RNG3);

Time Delay Generated:

5.37 milliseconds or the programmed DC delay, whichever is greater.

Description:

This instruction directs the programmable power supplies to force the specified currents. If the range is not specified, range 3 is used. The direction of positive current is out of the supply.

If a voltage trip has not been programmed, an "impossible" trip of less than -40 volts will be set, but not enabled.

Examples:

```
FORCE IF1 100E-3, RNG3;
FORCE IF3 -5E-3, RNG2;
FORCE IF2 30E-2;
```

11.2.3 Enable Current Trip

General Form:

```
ENABLE [TRIPI1/TRIPI2/TRIPI3] [LT/GT] expression
(,RNG2/,RNG3);
```

Time Delay Generated:

5.37 milliseconds or the programmed DC delay, whichever is greater.

Description:

These instructions enable the current-trip detector of the corresponding voltage forcing unit (VF1, VF2, VF3). If the source/load current of the forcing unit (VF) exceeds the enabled trip value during a test sequence, indicating a DC failure, then program control is transferred to the instruction as specified by the ON TRIP instruction, if that instruction was given. If an ON TRIP has not been executed prior to the trip, the program proceeds normally. At a pause or end-of-test, the PARAMETER FAIL indicator will be 'ON' if a trip occurred. If the DATALOG ON TRIP is set, the value specified by this instruction will be written on the output device.

The trips are ignored while the Tester Busy status is 'ON', i.e., until the time delay generated has expired. This feature allows surge currents without setting the trip when the VF supplies are driving capacitive loads. For additional time without increasing the value programmed in the SET DELAY, DC statement, the ENABLE TRIP may be programmed several statements after the FORCE.

The VF units will automatically disconnect under the following conditions:

(1) when the magnitude of the current is greater than 150 milliamps and the trip register is in range 2.

(2) when the magnitude of the current is greater than 1.50 amps and the trip register is in range 3.

The automatic disconnect is a safety feature that protects both the device under test and the DPS units. The trips will be processed, provided they have been enabled, even though the automatic disconnect occurs.

Example:

Enable the voltage forcing unit VF1 so that it will trip on load currents exceeding 100 milliamps and VF2 to trip on currents more negative than -50 milliamps.

```
ENABLE TRIPI1  GT 100E-3, RNG3;
ENABLE TRIPI2  LT -0.05,  RNG2;
```

NOTE:

If any trip is enabled, then all trips are enabled by implication. The trip specified will be enabled for the particular value; the others will be enabled for less than -1 ampere if the DPS is in voltage force mode or for less than -40 volts if the DPS is forcing current.

11.2.4 Enable Voltage Trip

General Form:

```
ENABLE [TRIPV1/TRIPV2/TRIPV3] [LT/GT] expression
(,RNG2/,RNG3);
```

Time Delay Generated:

5.37 milliseconds or the programmed DC delay, whichever is greater.

Description:

These statements enable the voltage trip detector of the corresponding power supply in the current forcing mode. Voltage

trips are processed in the same fashion as current trips (Refer to paragraph 11.2.3). This statement automatically connects the addressed unit to the test station load board.

Example:

Enable trip interrupts occur if the voltage on DPS1 is more negative than -10 volts or if the voltage on DPS3 is more positive than 30 volts.

```
ENABLE TRIPV1  LT -10.0, RNG2;
ENABLE TRIPV3  GT +30;
```

11.2.5 Disable Trips

General Form:

DISABLE TRIPS;

Time Delay Generated:

0 delay

Description:

This statement causes all voltage and current trips to be disabled.

11.2.6 Disconnect DPS Unit

General Form:

XCON [VF1/VF2/VF3];

Time Delay Generated:

5.37 milliseconds or the programmed DC delay, whichever is greater.

Description:

This statement disconnects the specified voltage forcing unit from the test head. If current forcing, the magnitude of the specified unit is automatically set to 0 in the low range prior to disconnecting. When forcing a voltage, the user should force a value which will minimize current flow before disconnecting. After execution of this statement, the DPS will automatically be

set to voltage force mode and an impossible current trip condition. The trip function for the specified unit is disabled. The DPS units will automatically disconnect from the test head at end-of-test without this instruction. Also, the trip function for the specified unit is disabled.

Example:

```
Disconnect all VF units

XCON  VF1;
XCON  VF2;
XCON  VF3;
```

11.2.7 DPS Programming - User Rules

Mode Change

Changing from voltage force to current force implies a disconnect. The user should minimize current flow with a FORCE VFx before a FORCE IFx or ENABLE TRIPx is programmed.

Programmed Delays

A hardware delay is initiated when necessary for relay changes. Additional delay may be desirable in some cases. For example, when forcing voltage into a capacitive load and a trip is programmed, additional delay may be used to prevent a trip while the capacitance is being charged. The trip can then be used for the steady state load.

The first time the DPS is programmed to a certain pin the time delay generated is 5.37 milliseconds or the programmed DC delay, whichever is greater. Subsequent DPS programming to the same pin results in a delay of 0.56 milliseconds or the programmed DC delay, whichever is greater.

Programmed Waits

When no mode change is required, delays will be initiated by the system but will not be timed out before the next DPS statement is processed. The delay will be executed before a measurement, however. This approach permits optimum throughput. If it is desired to ramp a DPS while waiting for a trip, this does require the user to program a FORCE WAIT in the ramping loop.

Initialization

It is not necessary to program a dummy trip to get maximum current capability while forcing voltages or to prevent one DPS from tripping on less than zero current if another DPS trip is enabled.

Trips

Enabling a trip for any DPS will enable trips for all DPS's. If trips are not programmed for remaining power supplies, an "impossible" value of less than -1 ampere or -40 volts will be used. Note that this enables the supply to deliver maximum power. Disconnecting any or all supplies does not disable the general trip interrupt in all cases. The disconnected supply can not trip until it is reprogrammed. (Actually, not until its DPS register and any DPT register are reprogrammed.) The general trip interrupt enable can be turned off only with a DISABLE TRIP statement.

Also, note that programming an ENABLE TRIPV will also cause the supply to be connected.

11.3 SET LOGIC

General Form:

SET LOGIC [POS/NEG];

Time Delay Generated:

0 delay

Description:

These statements initialize the functional test comparator logic pass conditions for either positive or negative voltage logic for the device under test (DUT). If this instruction is not used, the positive logic condition is assumed.

For positive logic the pass conditions are defined as follows:

(a) F(i) = 1 (expected output function for pin (i) = 1)
Pass = DUT OUTPUT SIGNAL>S1, otherwise fail.

(b) F(i) = 0 (expected output function for pin (i) = 0)
Pass = DUT OUTPUT SIGNAL<S0, otherwise fail.

For negative logic the pass conditions are defined as follows:

(a) F(i) = 1 (expected output function for pin (i) = 1)
Pass = DUT OUTPUT SIGNAL<S1, otherwise fail.

(b) F(i) = 0 (expected output function for pin (i) = 0)
Pass = DUT OUTPUT SIGNAL>S0, otherwise fail.

Note that all voltages, positive or negative, are treated algebraically, thus -10 is less than -1.

Example:

```
SET LOGIC NEG;
```

11.3.1 Force Voltage Conditioner References

General Form:

```
FORCE [E0/E1/EA0/EA1/EB0/EB1/EC0/EC1] expression
(,RNG2/,RNG3);
```

Time Delay Generated:

0.56 millisecond or 0.3 millisecond per volt, based on the difference between the previous voltage and the programmed voltage. 0.56 millisecond for voltage swings less than 5 volts.

Description:

These instructions force the voltage conditioner reference supplies to the programmed values. If the range is not specified, the lowest range, consistant with the programmed value, is automatically selected.

E0, E1, EA0, and EA1 are voltage conditioner references. The truth table below shows the relationship of F, S and these supplies. The supplies EB0, EB1, EC0 and EC1 are optional.

Example:

Force the standard reference pair to 3.5 and .5 volts respectively for the "1" and "0" levels.

```
FORCE E1 3.5, RNG2;
FORCE E0  .5, RNG2;
```

The following combinations of F and S bits per pin determine which reference voltage is applied to the pin.

F	S	
0	0	E0
1	0	E1
0	1	EA0
1	1	EA1

11.3.2 Set Reference Supplies for Functional Test Comparators

General Form:

SET [S1/S0] expression (,RNG2/,RNG3);

Time Delay Generated:

0.56 millisecond or 0.3 millisecond per volt, based on the difference between the previous voltage and the programmed voltage. 0.56 milliseconds for voltage swings less than 5 volts.

Description:

S1 and S0 are reference supplies for the functional test comparators. S1 is the reference level for the expected logic "1" levels and S0 is the reference level for the logic "0" levels. The programmed value is loaded into the functional test comparator reference voltage supply register.

For testing positive logic, S1 S0. For testing negative logic, as defined by the instruction SET LOGIC NEG, S1 S0. The following table shows pass/fail decisions made by the comparators.

POS LOGIC:		NEG LOGIC:	
S1	F=1 pass / F=1 fail	S0	F=0 pass / F=0 fail
S0	F=0 fail / F=0 pass	S1	F=1 fail / F=1 pass

Example: Set S0 comparator reference voltage to -5 volts:

```
SET S0 -5, RNG2;
```

11.3.3 Set Voltage Offset

General Form:

SET VOFFSET number;

Time Delay Generated:

0

Description:

This statement allows the specification of a voltage offset value which will be automatically added to values programmed by all voltage forcing statements.

The VOFFSET level can be used as a reference level for all other forcing voltages and hence reduce the programming task and improve listing readability.

Example:

If a supply absolute voltage range is +6 to -30 volts and a device requires greater than +6 volts with voltage swings within the 36 volt range, the offset value and supplies can be programmed as:

```
SET VOFFSET -18;
VSS = 20;
FORCE VF1 0;
FORCE VF2 VSS;
```

Here the actual voltages programmed are: VF1 = -18 volts and VF2 = +2 volts.

The voltage forcing statements affected by the voltage offset value are:

```
FORCE [VF1/VF2/VF3] expression (,RNG2/RNG3);
FORCE [E0/E1/EA0/EA1/EB0/EB1/EC0/EC1] expression
(,RNG2/RNG3);
FORCE PMU expression;
FORCE VOLTAGE expression (,RNG1/RNG2/RNG3);
SET [S1/S0] expression (,RNG2/RNG3);
ENABLE [TRIPV] expression (,RNG2/RNG3);
SET DCT [LT/GT] expression;
ENABLE [VL0/VH1] [LT/GT] expression;
ENABLE [DCT0/DCT1] [LT/GT] expression;
```

The resultant voltage value is autoranged for best resolution or a default range of 3 is used unless a range specifier is included in the forcing statement.

11.4 FUNCTIONAL TEST STATEMENTS

This section describes programming the long registers and functional testing.

11.4.1 Long Registers

The long registers are used to set up hardware conditions for functional testing. They include the following:

1) I/O pin definition, Set D (11.4.2)

2) Mask pattern definition, Set M (11.4.4)

3) Input reference power supply selection, Set S (11.4.5)

4) Utility Relay control for each pin Set R (11.4.6)

5) Define the functional test pattern for logical input states and expected output states, Set F (11.4.3)

The general form of the statements for programming functional test conditions is:

SET register or function name (*) binary pin pattern;

SET F (*) binary pin pattern (, binary pin pattern..);

Description:

This statement controls the functions that can be used to control the programmed functional test for each pin. The functions are each described as follows:

D — This sets the DEFINITION pattern for the input and output pins. A binary 1 defines the pin as an input and the corresponding driver amplifier will be connected. A binary 0 defines an output pin. In either case, the level detectors are connected to the pin. Time Delay Generated: 0.56 millisecond.

M — This control condition sets the MASK pattern that defines the pins on which test results will be measured. A binary 0 means a don't care for that pin. A binary 1 means a care and the comparators will be enabled for that pin. Time Delay Generated: 0.

S — This control condition sets the SELECT pattern that specifies the alternate or standard forcing logic level pair for each pin defined as an input. A binary 0 selects the standard reference pair E1 or E0. A binary 1 selects the alternate reference pair EA1 or EA0. Time Delay Generated: 0.28 millisecond.

R — This control condition sets the UTILITY RELAY pattern that specifies which relays are closed and which relays are open. A binary 1 closes the corresponding utility relay. Time Delay Generated: 1.75 milliseconds.

F — This control condition sets the binary FUNCTION pattern for the logical input states and the expected output states. For an input pin, a binary 1 specifies that the input level will be that specified by a 1 reference (E1 or EA1). For an output, a binary 1 means the output voltage is compared to S1 reference. Time Delay Generated: determined by Set Delay statement.

The binary pin pattern is defined by a binary value which has a one to one correspondence between pin and pattern bit location. Only pins that change from the previous state need be specified by the SET statement. Even if all pins are specified and the asterisk is omitted, only codes for the ranks (Refer to Appendix B) in which pin data has changed are generated for maximum test rate. In the first statement of a type executed by the program, all pin states not specified will be assumed to be zero.

The binary pin pattern can be programmed by either specifying each bit of the pattern or by use of the following operators: n = pin origin and (:) = pattern replicator. These two operators are defined as follows:

[n] = pin origin operator
where: n is an integer

$(m:b_p)$ = binary pattern replicator operator
where: m is an integer
and b_p is the binary pattern

All non-addressed pins will take on the previously specified values. The absence of a previous specification is interpreted as a binary 0. Furthermore, non-addressed ranks will not be affected, i.e., no code will be generated and hence, at runtime, the nonaddressed ranks will not be modified.

A sequence of binary patterns, separated by commas represent a series of functional tests. To illustrate the use of these statements, two examples are shown below. The first example uses the origin and replicator operators. The second example yields the same binary pin patterns, but does not make use of the operators.

Example (1)

```
SET F       (3:0)    (13:1)   (2:0),
            (3:101)    010    (6:1),
            [8]      (2:1)    (9:0);
```

Example (2):

```
SET F       000 111 111 111 111 100,
            101 101 101 010 111 111,
            101 101 111 000 000 000;
```

Explanation:

(3:0) means three 0 values specified, (13:1) means thirteen 1 values (2:0) means two 0 values, (3:101) means 3 sets of 101 values, 010 means set the next three pins to this pattern, etc., etc. The [8] means: preserve the previous pattern and start at pin 8 with the following specifications. (Note pins are numbered from 1.)

It should be noted that blanks are ignored within the binary pin pattern.

The asterisk form of the instruction forces data codes to be generated for all ranks in which pins are specified. It should be used whenever the flow of control is altered.

```
Example:                Instruction          Compiled Ranks

   1                    SET S (30:1);           1 and 2
   2                    SET S (29:1) 0;         2
   3          X:        SET S*(30:1);           1 and 2
                              .
   .                          .
   n                    SET S (30:0);           1 and 2
                        GOTO X;
```

Statement 1 is included in order to get the S register in a known state. Since pins 1 to 15 are already 1's, only rank 2 is generated for statement 2. Without the asterisk in statement 3, only rank 2 would be regenerated for the same reason; however, since the first 15 pins are 0's at statement n, execution of statement 3 as the result of a branch from statement n+1 would be an error. Using the asterisk form of the SET S statement corrects the problem in this example.

The asterisk form should also be used after a subroutine call.

Example:

Instruction		Compiled Data
SET F	(60:1);	06077777 06177777 06277777 26377777
SET F	(59:1)0	26337777
CALL X;		
SET F*	(60:1);	06077777 06177777 06277777 06377777

In this example, note that the second SET F generated only one word. Normally, words are generated only for ranks with data that changes. However, the fourth statement follows a subroutine call and, at run time, subroutine X may alter the F register. To insure that all 60 pins were returned to "one", the programmer used the asterisk. This asterisk forced the compiler to generate data for all ranks explicitly mentioned in the statement.

11.4.2 Set D

General Form:

SET D binary pattern;

Time Delay Generated:

0.56 millisecond

Description:

This instruction loads the D I/O pin definition register. A binary 1 in the pattern field specifies the corresponding test pin to be used as an input, thus connecting the pin to a voltage driver via a reed relay. A binary 0 in the pattern field specifies the corresponding test pin to be an output, thus disconnecting this pin from the voltage driver. In both cases, the test pin remains connected to the corresponding pin voltage level detector.

11.4.3 Set F

General Form:

SET F binary pattern (,binary pattern,...);

Time Delay Generated:

Specified by SET DELAY statement.

Description:

This instruction loads the F register. A binary 1 in the binary pin pattern specifies a logic 1 level and a binary 0 specifies a logic 0 level. The bits in the binary pin pattern corresponding to inputs, as specified by (D), are the forcing function. The bits in the binary pin pattern corresponding to outputs, as specified by (D), are the expected outputs. The input forcing analog voltage levels and the expected output comparison thresholds are determined by the contents of the programmable reference supplies S0, S1, E0, E1, EA0, EA1, and by the S register.

11.4.4 Set M

General Form:

SET M binary pattern;

Time Delay Generated:

0

Description:

This instruction loads the M register. A binary 1 in the binary pin pattern enables the associated pin level detector. A binary

0 in the binary pin pattern disables the associated pin level detector. These two states are referred to respectively as the "care" and "don't care" conditions for pins on which functional test results will be measured. The functional test results are strobed from the level detector outputs to the comparison register C, for "care" pins. The strobe is inhibited for "don't care" pins.

11.4.5 Set S

General Form:

SET S binary pin pattern;

Time Delay Generated:

0.28 millisecond

Description:

This instruction loads the S register. A binary 0 in the binary pin pattern selects the standard logic level pair E1/E0 as input forcing voltages and the standard comparator reference pair S0/S1 for the corresponding test pin. A binary 1 selects the optional alternate logic level pair EA1/EA0, and the optional comparator reference SA0/SA1. When switching from one comparator reference pair to the other, the user must program a one millisecond delay to allow switching of the relay multiplexer for the references.

11.4.6 Set R

General Form:

SET R binary pin pattern;

Time Delay Generated:

1.75 milliseconds

Description:

This instruction provides the control for opening or closing the utility relays. There is one utility relay associated with each tester pin. A binary 1 in the binary pin pattern will close the utility relay and a binary 0 will open the relay.

11.4.7 Force Strobe

General Form:

FORCE STROBE;

Time Delay Generated:

0

Description:

This instruction forces a single functional test strobe, thus transferring the functional comparator output states to the C register. The strobe is generated, even though the COMPARATORS may be disabled (11.4.9). The detection of a failure is processed in the same manner as is normally done during functional testing.

Example:

FORCE STROBE;

11.4.8 Enable Latches

General Form:

[ENABLE/DISABLE] LATCHES;

Time Delay Generated:

0

Description:

The DISABLE instruction initializes the functional test control so that the C register is cleared prior to strobing the functional test comparators for each functional test.

If no latch statement is made, the disable latch mode is assumed.

The ENABLE instruction initializes the functional test control so that the C register is not cleared prior to strobing the functional test comparators. In this mode all functional failures are retained in the C register throughout a sequence of tests.

Examples:

Enable the comparison register to retain a history of all functional failures throughout a sequence of functional tests.

```
ENABLE LATCHES;
```

Disable the comparison register latches so that the C register contains only the current functional test failures.

```
DISABLE LATCHES;
```

11.4.9 Enable Comparators

General Form:

[ENABLE/DISABLE] COMPARATORS;

Time Delay Generated:

0

The ENABLE instruction initializes the functional test control logic so that the comparator outputs will be strobed to the C register for each functional test. The Enable comparator state is assumed if no statement is given.

Also the DISABLE instruction initializes the functional test control logic so that the comparator output will not be strobed to the C register for each functional test.

The disable comparator instruction should be issued when a series of functional patterns are to be executed but the device under test response is not defined.

Example:

Enable the comparators.

```
ENABLE COMPARATORS;
```

11.4.10 Enable Strobe

General Form:

ENABLE STROBE binary pattern;

Time Delay Generated:

0

Description:

This instruction enables the functional test comparator strobe to be controlled by the contents of F (1-4) and the "binary Pattern" (or equivalently the 4 bit "value"). The "binary pattern" for this instruction must be a 4-bit binary number and it is defined as follows:

binary pattern = $b_1b_2b_3b_4$,

where the subscripts refer to the tester pin number and b is either a 0 or a 1.

After executing the ENABLE STROBE instruction, all subsequent functional test results will be strobed to the C register according to the following logical condition ('.' AND, '+' OR):

$$STROBE = b_1 \cdot F(1) + b_2 \cdot F(2) + b_3 \cdot F(3) + b_4 \cdot F(4)$$

Example:

Enable the strobe to be controlled by F(1).

```
ENABLE STROBE 1000;
```

Note: This statement automatically DISABLES the normal comparator strobe, i.e., DISABLE COMPARATORS

11.5 AUXILIARY CLOCK STATEMENTS

This section describes the statements which control the auxiliary clock functions of the Sentry 200. The clock functions allow a string of up to 255 clock pulses to be generated for each functional test. In addition, these clocks can be selectively enabled or disabled for each functional test.

11.5.1 Set Clock

General Form:

```
SET CLOCK expression;
```

Time Delay Generated:

0

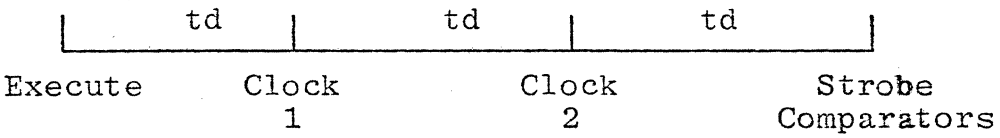
Description:

The value of the expression is loaded into the clock burst count register of the tester. The count specifies the number of clock signals which the tester will output for each functional test. The time delay between clocks is equal to the current contents of the time delay register. The maximum clock count is 255.

Example:

SET CLOCK 2;

The resultant timing is explained in the following diagram:



td = contents of time delay register +0.7 usec.

11.5.2 Enable Clock

General Form:

ENABLE CLOCK binary pattern;

Time Delay Generated:

0

Description:

This instruction enables clock signals to be connected to tester pins 1, 2, 3, 4, according to the "binary pattern" (or equivalently the 4 bit "value"). The binary pattern for this instruction must be a 4 bit binary number and it is defined as follows.

binary pattern = $b_1b_2b_3b_4$,

where the subscripts refer to the tester pin number and b is either a 0 or a 1.

Four clock sync lines for the first four pins are brought to the load board. These are used to drive an external clock generator. The clock generator may be as simple as an IC gate mounted on the load board or may be a complex 4-phase clock. The clock may be connected to the device-under-test (DUT) via a utility relay pin on the load board or with a separate relay to the DUT directly. If a separate relay is used, it may be driven by the clock relay (CRLY) signals at the load board. Whenever the PMU is addressed to one of the clock pins, the clock relay will automatically open.

Enabled clock signal pins are disconnected from their corresponding functional test circuits. In addition to connecting the clock lines, this instruction enables the sync signals. Sync signals are programmed by the logical "and" condition of the enabled clock pins. The contents of F(1-4) is as follows:

$$SYNC_1 = b_1 \cdot F(1)$$

$$SYNC_2 = b_2 \cdot F(2)$$

$$SYNC_3 = b_3 \cdot F(3)$$

$$SYNC_4 = b_4 \cdot F(4)$$

The number of sync pulses and their periods are specified by:

SET CLOCK number; (clock burst counter)

Period = 0.7 microseconds + td, where td is the time delay specified by the SET DELAY instruction.

Example:

Enable clock signals to pins 1 and 4.

```
ENABLE CLOCK 1001;
```

11.5.3 Force Clock

General Form:

```
FORCE CLOCK;
```

Time Delay Generated:

- as specified by the SET DELAY instruction.

Description:

This instruction forces a single clock pulse to occur at each of the 4 SYNC lines. The width of the SYNC pulse is equal to the value specified by the SET DELAY instruction.

Example:

```
FORCE CLOCK;
```

11.5.4 Programming Cautions

Executable code for DMA modes are directly generated by the single-pass compiler and several points should be kept in mind while programming a FACTOR test plan.

The compiler sees the event in the order of statements, not in the order of execution. Therefore, whenever a branch, a subroutine call, or a conditional statement is involved in a program, care must be taken in order to maintain the proper long register data (S, M, R, etc.) and the proper F data.

The compiler automatically generates F data for all ranks in which the pins are specified at the first SET F even if the SET F* form is not programmed.

The compiler keeps the previous state of all long register ranks* and compares them with the programmed pattern. Long register data are generated only for ranks which have been changed.

The decision to change long register data is strictly done in the statement order the compiler sees. Here again a branch, a subroutine call or a conditional statement affects the status of data and care must be taken.

Example:

```
IF A NEQ B, THEN SET M ---
ELSE SET M ----
```

The compiler sees the SET M in the THEN clause first and processes the rank change. And it sees the SET M in the ELSE clause and processes the change against the previous state. Therefore, the decision whether the M data in the ELSE clause has changed or not is dependent on the M data in the THEN clause.

When the SET register* form is programmed, the compiler generates data for all ranks in which the pins are specified, even if the ranks are not changed.

```
SET F* (60:0);  generates rank 1, 2, 3, 4
SET F* [16] 1;  generates rank 2 only.
```

*A long register rank data memory is maintained in the compiler for registers programmed by the following statements:

```
SET  F/D/M/S/R
```

11.6 PRECISION MEASURING UNIT STATEMENTS

This section describes the statements that control the operation of the Sentry 200 Test System precision measuring unit. This unit is used for measuring device under test DC parameters and for system self-check. DC test macro statements are provided to reduce programming and increase test throughput (Refer to paragraph 11.9).

11.6.1 Set PMU Ranges

General Form:

```
SET PMU [SENSE/FORCEV/FORCEI] (,RNG0/,RNG1/,RNG2/
,RNG3/,AUTO);
```

Time Delay Generated:

Set PMU SENSE - 0.56 millisecond with a mode change or voltage range change or 4 milliseconds (± 1 millisecond) with a current range change.

Set PMU FORCE - The value programmed in the SET DELAY, DC statement or the appropriate value shown for SET PMU SENSE, whichever is greater.

Description:

This instruction initializes the precision measuring unit (PMU). The force and sense functions are complimentary with respect to voltage and current; i.e., when the unit is set to force voltage (or current) it is automatically initialized to sense current (or voltage). When forcing or sensing voltage, the legal ranges are 1, 2 and 3. When forcing or sensing current, the legal ranges are 0, 1, 2 or 3.

When forcing the PMU and AUTO is specified, the hardware range will be selected to give best resolution. When measuring and AUTO is requested, the initial measurement is made in the highest range (current/voltage range 3) and then the range is decreased if necessary to provide the best measurement resolution.

Example:

```
SET PMU SENSE, RNG2;
SET PMU FORCEI, RNG3;
```

In this example the statements initialize the PMU to force current in range 3 and sense voltage in range 2.

11.6.2 Force Voltage/Current

General Form:

```
FORCE CURRENT  expression  (,RNG0/,RNG1/,RNG2/,RNG3);

FORCE VOLTAGE  expression  (,RNG1/,RNG2/,RNG3);
```

Time Delay Generated:

Same as SET PMU FORCE.

Description:

This instruction is used to force a programmed voltage or current via the precision measurement unit. Upon execution of this statement, the output of the precision measurement unit will begin to slew to the desired value. If range is not specified; the highest range will automatically be set. If the expression contains only a constant, this statement will be executed in DMA mode.

Example:

Force the output of the precision measurement unit to -1 microamp.

```
FORCE CURRENT -1E-6 , RNG1;
```

11.6.3 Force PMU

General Form:

FORCE PMU expression;

Time Delay Generated:

Same as SET PMU FORCE

Description:

The expression in this statement is scaled according to the mode, V or I, and range as preset by the SET PMU SENSE/FORCE... statement (see paragraph 11.6.1). If AUTO ranging has been preset or if the expression contains a variable, then the range which gives best resolution will be automatically determined (at run time) prior to loading the PPS register. This instruction is especially useful when the PMU is to be forced to several values in the saME range, but it has a slightly longer execution time than FORCE VOLTAGE/CURRENT.

Example:

```
Force the output of the PMU to -1 microamps:
SET PMU FORCEI, AUTO;
A=5E-6; B=5;
FORCE PMU -A/B;
```

11.6.4 Connect PMU

General Form:

CPMU PIN expression;

Time Delay Generated:

0.56 millisecond

Description:

This statement connects the precision measurement unit to the pin number specified by the expression (or equivalent value). When the expression is a constant, the PMU will be connected in DMA mode.

Internal nodes are specified by numbers in the range 128 through 145. The node numbers and descriptions are summarized in Appendix H.

NOTE

The PMU must be programmed to force 0 current in range 2 before the PMU is connected to an internal node.

11.6.5 Measure Pin

General Form:

```
MEASURE PIN;
```

Time Delay Generated:

.56 millisecond

Description:

MEASURE PIN allows fast go/no-go DC parameter tests. It is similar to the MEASURE VALUE statement (paragraph 11.6.6) except that go/no go comparisons are made against the SET DCT limit. No floating point conversion is made, nor is the result stored in VALUE.

Since the comparison is made by the hardware, SET DCT must be executed after the PMU sense and force conditions are specified, so that the DCT limit can be scaled properly.

No autoranging will occur when this instruction is used.

If any datalogging conditions are specified and met (Datalog DCT or MEASURE), the resultant measurement will be determined by successive approximation.

Example:

```
SET PMU SENSE, RNG1;
CPMU PIN 5;
FORCE VOLTAGE 4.5, RNG2;
SET DCT GT 6.0E-6;
MEASURE PIN;
```

11.6.6 Measure Value/Node

General Form:

MEASURE [VALUE/NODE number] (,LOG);

Description:

This statement initiates a measurement within the precision measurement unit (PMU).

When the VALUE option is specified, the measurement is made according to the existing state of the PMU. The measured value is converted and scaled to floating-point, using a successive approximation algorithm, and stored in the system global variable, VALUE. If d.c. trips are enabled (see paragraph 11.6.9), VALUE is tested to determine if it falls within the pass window (defined by the previous ENABLE DCT0/1 statements, regardless of the current DCT register contents). If it passes, the "D.C. Pass' indicator is set. If it fails, the "D.C. Fail" indicator is set. Conditions required for datalogging the resulting value are described in the note at the end of this section.

If AUTO ranging has been specified (see paragraph 11.6.1), the system will automatically select the measuring range which gives best resolution. Autoranging begins with the highest range and ranges downward until best resolution is obtained or until the lowest range is reached.

Use of the NODE option provides a means for measuring a parameter at a system internal monitor node. The node numbers and their descriptions are listed in Appendix H. Ranging and measuring conditions are automatically controlled.

For internal nodes, the measured value, logging, and pass/fail conditions are the same as previously described. At the conclusion of an internal node measurement cycle, the PMU is automatically disconnected and initialized to force 0 current in Range 1.

Values measured at internal nodes 202B to 205B for the E reference supplies are divided by 8. That is, when a reference supply (E1, E0, etc) is programmed to V volts, the measured value will be V/8 volts since the buffer amplifier at each pin has a gain of eight.

Example:

Measure and log the current from VF1:
MEASURE NODE 217B, LOG;

NOTE

Datalogging options available via the TOPSY monitor are:

1) DATALOG MEASURE: All MEASURE VALUE statements are logged.
2) DATALOG LOG: All MEASURE VALUE, LOG statements are logged.
3) DATALOG DCT: All measurements which fail to meet the limits specified by the ENABLE DCT statements are logged.
4) Any combination of the above.

11.6.7 Disconnect PMU

General Form:

XPMU PIN;

Time Delay Generated:

0.56 millisecond

Description:

This statement disconnects the PMU from its present pin connection and resets it to Pin 0.

11.6.8 Set DC Parameter Limit

General Form:

SET DCT [LT/GT] expression (,RNG0/,RNG1/,RNG2/,RNG3) ;

Time Delay Generated:

56 microseconds

Description:

The SET DCT forms a pass or fail threshold.

When a MEASURE PIN does not pass the level specified by the DCT function, a DC fail is indicated and program control is transferred to the instruction specified by the "ON DCT"

instruction. If an "ON DCT" has not been previously executed, then the next instruction following the MEASURE is executed. A failure will cause the PARAMETER FAIL indicator to go 'ON' at the next pause or at end-of-test. The range specified in this instruction will override that specified in the SET PMU SENSE statement.

Example:

Enable DC trip limits which will pass all measured values between -2 milliamps and +2 microamps.

```
SET DCT GT 2E-6, RNG1;
MEASURE PIN;
SET DCT LT -2E-3, RNG1;
MEASURE PIN;
```

This statement sequence will be executed at DMA speed.

11.6.9 Enable DC Parameter Limits

General Forms:

[ENABLE/DISABLE] [DCT0/DCT1] [LT/GT] expression;

Time Delay Generated:

0

Description:

Execution of ENABLE DCT0/DCT1 forms a pass or fail threshold for level DCT0 and/or DCT1. Either one or both DCT thresholds may be specified.

Using both DCT functions specifies a "pass window" for subsequent DC measurements (see MEASURE VALUE, Paragraph 11.6.6). The pass region may be specified by the operators LT/GT and by the value of the expression in the ENABLE statement.

When a measurement caused by the statement MEASURE VALUE does not fall within the "pass window" specified by the DCT function, a DC fail is indicated and program control is transferred to the statement specified by the "ON DCT" statement (paragraph 11.1.3). If an "ON DCT" has not been executed, then the statement following the MEASURE will be executed. At a pause or end-of-test, if a failure has occurred, the parameter fail light will be lighted.

The DISABLE statement disables the comparison limits and inhibits the DC FAIL regardless of the measured value.

NOTE

This statement provides a measurement limit comparison for the statement MEASURE VALUE only.

Example: Enable DC trip limits which will pass all measured values in the range from -2 milliamperes to +2 microamperes.

```
ENABLE DCT1 GT -2E-6;
ENABLE DCT0 LT -2E-3;
```

11.6.10 Enable Relay - Connect PMU to Functional Circuitry

General Form:

[ENABLE/DISABLE] RELAY;

Time Delay Generated:

0.56 millisecond

Description:

The DISABLE instruction initializes the pin address control logic such that the voltage conditioner for pin (n) will be automatically disconnected when the precision measurement unit is connected to pin (n). If no relay statement is made, the disable mode is assumed.

The ENABLE instruction initializes the pin address control logic such that the voltage conditioner for pin (n) will remain connected, even though the precision measurement unit is connected to pin (n). After connecting the precision unit to pin (n), the voltage conditioner can be disconnected by executing the instruction: Disable Relay; i.e., allowing a make-before-break sequence to maintain bias on a pin.

Example: Connect the precision measurement unit to pin (10) with the voltage conditioner connected and then disconnect the voltage conditioner.

```
ENABLE RELAY;
CPMU PIN 10;
DISABLE RELAY;
```

11.6.11 Measure Variable

General Form:

```
MEASURE VARIABLE variable (,LOG);
```

Time Delay Generated:

0

Description:

The value of the variable, in floating point format, is stored in the system global variable: VALUE. Variable may be a variable identifier or an array element.

If DC trips are enabled (see Section 11.6.9), VALUE is tested to determine if it falls within the pass window. If it passes, the "DC PASS" indicator is set. If it fails, the "DC FAIL" indicator is set. The value of the variable will be logged according to the Monitor logging command conditions. Tester pin number 0 will be logged for the statement.

Datalogging options specified by the TOPSY Monitor are:

1. Datalog MEASURE: All 'MEASURE VARIABLE' statements are logged.

2. Datalog LOG: 'MEASURE VARIABLE, LOG' statements are logged, i.e. only those having the 'LOG' modifier.

3. Datalog DCT: All 'MEASURE VARIABLE' statements that fail to pass the limits specified by 'ENABLE DCT' statements are logged.

Example:

Measure and log the value of variable XVAL,

```
MEASURE VARIABLE XVAL,LOG;
```

11.7 MISCELLANEOUS CONTROL STATEMENTS

This section describes statements that control the timing as well as initialization of the Sentry 200 Test System.

11.7.1 Force Reset

General Form:

```
FORCE RESET;
```

Time Delay Generated:

Programmed DC delay or 5.37 milliseconds, whichever is greater.

Description:

This instruction forces the test system into the reset state, thus clearing all programmable test conditions.

Example:

Clear the test system.

```
FORCE RESET;
```

11.7.2 Force Delay

General Form:

```
FORCE DELAY;
```

Time Delay Generated:

As specified by SET DELAY, DC statement.

Description:

This instruction forces a time delay to occur, prior to executing the next delay dependent instruction.

Example:

Provide a delay after programming the VF1 unit to provide settling time for the change of programmed voltage prior to executing a MEASURE VALUE.

```
FORCE VF1 10.0;
FORCE DELAY;
MEASURE VALUE;
```

11.7.3 Force Wait

General Form:

FORCE WAIT;

Time Delay Generated:

The time required for the tester to become not-busy.

Description:

This instruction forces the tester to wait until the tester status is not-busy before processing the next instruction.

Note: The tester goes busy after executing those instructions with a non-zero time delay.

Example:

Provide a delay until the tester is not busy after forcing a power supply voltage.

```
FORCE VF1, 5.0 RNG2;
FORCE WAIT;
```

11.7.4 Clear Fail

General Form:

CLEAR FAIL [FCT/DCT/TRIP] ;

Time Delay Generated:

0 delay

Description:

This instruction clears the system software fail flags. When a fail is normal e.g., for device pre-conditioning, this instruction may be used to inhibit this fail from being displayed at end of test. More than one option may be specified, separated by commas.

11.7.5 Enable Access

General Form:

ENABLE ACCESS;

Time Delay Generated:

30 - 75 millisec (one disc access)

Description:

This instruction forces a disc access to reload the core memory buffer. Normally TOPSY dynamically allocates the amount of data in the core memory buffer. With a 16K core memory, the maximum buffer size is about 4000 words. A disc access and reload of this buffer takes place at the end of the core buffer and takes about 70 milliseconds. When testing dynamic MOS devices with large programs, it may be desirable to control when the core buffer is refreshed so some initializing data for the device under test can follow the disc access period. This may be done with the Enable Access instruction.

11.7.6 Insert

General Form:

INSERT string file name;

Description:

This statement causes the FACTOR statements of the referenced string file name to be compiled, with the resultant object code inserted into working storage, along with the current FACTOR program compilation.

NOTE

The string file name must be a file under the current job name. In addition, the string file must not contain an END statement as a block end. The inclusion of one will cause the compilation to terminate incorrectly at that point.

11.8 EXTERNAL INTERFACE REGISTER READ/WRITE

General Form:

```
WRITE (EIR) expression;
READ (EIR) variable;
```

Time Delay Generated:

0 delay

Description:

This fifteen bit register is used to display test results and control external device handlers. Bits 0-9 are available to the programmer to use in any form, such as to define various pass categories. Bits 10-14 are defined by system software. All bits are read/write. If the user wishes to use some bits to read the status of external equipment, then a simple hardware modification can be made to the register by disconnecting the bit storage device from the register. Consult your field service representative if this is desired.

Example:

Set bit 7 of the EIR without modifying other bits -

```
READ (EIR) X;
WRITE (EIR) X OR 100B;
```

11.9 FACTOR TEST MACROS

11.9.1 Introduction

Up to this point, the set of FACTOR language statements have been constructed to provide a one for one correspondence between statement and tester function. There are standard measurements that are common to most devices that always require a combination of several FACTOR statements. It then becomes advantageous to provide a macro statement that combines these functions in order to perform a measurement. The use of macros to perform measurements also induces other beneficial side effects.

The most significant advantage is that its use reduces the burden of device test programming and in turn reduces debug time. This reduction is achieved by using assured good device programming techniques. These techniques include insuring that PMU range or

mode (current or voltage) changes are not made while the PMU is connected to the device to prevent incorrect readings or possible harmful transients. These techniques also guarantee that tester relays are switched cold thereby prolonging the life and reliability of the test system.

Another advantage is the provision for precharging the PMU. For some devices whose pins feedback to internal gates, it may be necessary to precharge the PMU to a voltage that will not disturb the functional state of the device when the PMU connection is made. eg. The 9328 TTL dual 8 bit shift register with unbuffered outputs required precharging the PMU to some voltage in the logic area (Vcc).

Because macro's are combinations of single functions, overhead time is reduced which results in shorter test time and higher device thruput along with a reduction in the total test program size.

11.9.2 Macro Factor Statements

General Form:

```
SET TEST number;

MEASURE PIN number (,LOG);
```

11.9.3 Macro Definition and Description

The statement, SET TEST number, together with an immediately proceeding setup procedure provides a macro definition for performing a particular type of DC measurement. The DC measurement sequence is then initiated with the statement 'MEASURE PIN number'. Depending on the test type, the setup procedure may contain the PMU connected to a precharge pin, open, or short circuit. The PMU is forced to the desired current or voltage value in the desired sense range.

An example setup for VOH test is as follows:

```
ENABLE RELAY;
CPMU PIN 16;                REM PIN 16 = Vcc for precharge
FORCE CURRENT -360E-6, RNG2;
SET PMU SENSE, RNG2;
SET TEST 1;
ENABLE DCT1 GT 3.5;
ENABLE DCT0 LT 2.7;
MEASURE PIN 12;
```

The execution of the SET TEST 1 statement causes the connected test pin, pin 16, and the forcing mode and value, current -360E-6, RNG2, to be saved. In addition, when current is being forced, the PMU is reset to force zero current on the same range whenever relay switching occurs. The statement MEASURE PIN 12 will cause the following measurement sequence to be performed.

CPMU PIN 12 DISABLE RELAY* FORCE CURRENT -360-6, RNG2	The PMU is switched from the prechange pin to the pin under test.
MEASURE VALUE	The measurement is made.
FORCE CURRENT 0, RNG2 ENABLE RELAY* CPMU PIN 16	The PMU is reconnected to the rest pin while the PMU is forcing 0 current.

NOTE

The FACTOR statement MEASURE PIN is different from MEASURE PIN number. MEASURE PIN is retained for performing a GO/NO-GO comparison on the currently connected tester pin. The MEASURE VALUE performed by the macro does not update the variable VALUE.

* The Enable Relay-Connect-disable relay sequence is used to allow the local pin voltage driver to clamp the current forcing PMU when initially connected to the pin. That is, a make-before-break sequence between the PMU relay and D relay occurs. To make use of this feature, the pins must be previously defined as input by the SET D instruction.

11.9.4 Available Macro DC Measurements

SET TEST 1 = VOH	Voltage output high
SET TEST 2 = VOL	Voltage output low
SET TEST 3 = IIH (IR)	Input leakage
SET TEST 4 = ICEX	Output leakage
SET TEST 5 = IIL (IFX)	Input low current
SET TEST 6 = VF3	Input diode clamp
SET TEST 7 = ISC	Short ckt. current with Outputs high
SET TEST 9 = IOL	Output current with Outputs low
SET TEST 10 = VBD	Voltage breakdown

1. SET TEST 1; VOH

Set-up Procedure:

```
ENABLE RELAY
CPMU PIN r                REM connect to rest pin - precharge
FORCE CURRENT i , RNGx
SET PMU SENSE, RNGy
ENABLE DCT0 LT lower limit
ENABLE DCT1 GT upper limit
SET TEST 1                REM select test macro

MEASURE PIN t             REM executes the test measurement
                          macro on pin t.
```

Macro Measurement Sequence:

```
CPMU PIN t
DISABLE RELAY
FORCE CURRENT i, RNGx
MEASURE VALUE             REM perform measurement
FORCE CURRENT 0, RNGx
ENABLE RELAY              REM reconnect the rest pin
CPMU PIN r
```

2. SET TEST 2; VOL

Set-up Procedure:

```
ENABLE RELAY
XPMU PIN
FORCE CURRENT i, RNGx
SET PMU SENSE, RNGy
```

```
ENABLE DCT1 GT limit
ENABLE DCT0 LT limit
SET TEST 2

MEASURE PIN t              REM the same measurement sequence
                           as SET TEST 1 is performed.
```

3. SET TEST 3; IIH (IR)

Set-up Procedure:

```
DISABLE RELAY
XPMU PIN
FORCE VOLTAGE v, RNGx
SET PMU SENSE, RNGy
ENABLE DCT0 LT limit
ENABLE DCT1 GT limit
SET TEST 3

MEASURE PIN t              REM execute test measurement macro.
```

Macro Measurement Sequence:

```
      CPMU PIN t
      FORCE DELAY
      MEASURE VALUE        REM perform measurement.
```

4. SET TEST 4; ICEX

The setup procedure and measurement sequence is identical to SET TEST 3; IIH (IR)

5. SET TEST 5; IIL (IFX)

Set-up Procedure

```
DISABLE RELAY
XPMU PIN
FORCE VOLTAGE v, RNGx
SET PMU SENSE, RNGy
ENABLE DCT0 LT limit
ENABLE DCT1 GT limit
SET S1 voh, RNGx           REM Set Ref S1 slightly > device
SET TEST 5                 VOH on same range as PMU FORCE

MEASURE PIN t                    REM execute test measurement
                                 macro.
```

Macro Measurement Sequence:

```
CPMU PIN t
FORCE VOLTAGE v, RNGx
MEASURE VALUE
FORCE VOLTAGE voh, RNGx
XPMU PIN                    REM if XPMU PIN was given in
                            the set-up procedure, other-
                            wise the last pin number.
```

6. SET TEXT 6; VF3

Set-up Procedure

```
ENABLE RELAY
XPMU PIN
FORCE CURRENT i, RNGx
SET PMU SENSE, RNGy
ENABLE DCT0 LT limit
ENABLE DCT1 GT limit
SET TEST 6

MEASURE PIN t               REM execute test macro.
```

Macro Measurement Sequence:

```
CPMU PIN t
DISABLE RELAY
FORCE CURRENT i , RNGx
MEASURE VALUE
FORCE CURRENT 0 , RNGx
ENABLE RELAY
```

7. SET TEST 7; ISC

Set-up Procedure:

```
DISABLE RELAY
XPMU PIN
FORCE VOLTAGE v, RNGx
SET PMU SENSE, RNGy
ENABLE DCT0 LT limit
ENABLE DCT1 GT limit
SET S1 voh, RNGx            REM Set Ref. S1 slightly > device
SET TEST 7                  VOH on same range as PMU force

MEASURE PIN t               REM execute test macro.
```

Macro Measurement Sequence:

```
CPMU PIN t
FORCE VOLTAGE v, RNGx
MEASURE VALUE
FORCE VOLTAGE voh, RNGx
XPMU PIN                    REM if XPMU PIN was given in
                            the setup procedure, otherwise
                            the last connected pin number.
```

8. SET TEST 8; Not Available

9. SET TEST 9; IOL

Set-up Procedure

```
DISABLE RELAY
XPMU PIN
FORCE VOLTAGE v , RNGx
SET PMU SENSE, RNGy
ENABLE DCT0 LT limit
ENABLE DCT1 GT limit
SET S0 vol, RNGx          REM Set Ref S0 slightly < device
SET TEST 9                VOL on same range as PMU force.

MEASURE PIN t                   REM execute test macro.
```

Macro Measurement Sequence:

```
CPMU PIN t
FORCE VOLTAGE v, RNGx
MEASURE VALUE
FORCE VOLTAGE vol, RNGx
XPMU PIN
```

10. SET TEST 10; VBD

The setup procedure and measurement sequence is identical to SET TEST 6, VF3.

FACTOR STATEMENT	TEST	SETUP PROCEDURE	MEASUREMENT SEQUENCE	STATE OF PMU AFTER MEASURE PIN t: FORCING	STATE OF PMU AFTER MEASURE PIN t: PIN
		ENABLE RELAY			
SET TEST 1	VOH	CPMU PIN r	CPMU PIN t		
MEASURE PIN t		FORCE CURRENT i, RNGx	DISABLE RELAY		
		SET PMU SENSE, RNGy	FORCE CURRENT i, RNGx		
		SET TEST 1	MEASURE PIN		
		SET DCT Limit	FORCE CURRENT 0, RNGx		
			ENABLE RELAY	0 CURRENT	Set up pin
			CPMU PIN r		
		XPMU PIN			
SET TEST 2	VOL	ENABLE RELAY	CPMU PIN t		
MEASURE PIN t		FORCE CURRENT i, RNGx	DISABLE RELAY		
		SET PMU SENSE, RNGy	FORCE CURRENT i, RNGx		
		SET TEST 2	MEASURE PIN		
		SET DCT Limit	FORCE CURRENT 0, RNGx		
			ENABLE RELAY	0 CURRENT	Set up pin
			CPMU PIN r		
		XPMU PIN			
SET TEST 3	IIH (IR)	DISABLE RELAY	CPMU PIN t		
MEASURE PIN t		FORCE VOLTAGE v, RNGx	FORCE DELAY		
		SET PMU SENSE, RNGy	MEASURE PIN	v VOLTAGE	PIN t
		SET TEST 3			
		SET DCT Limit			

FACTOR STATEMENT	TEST	SETUP PROCEDURE	MEASURE SEQUENCE	STATE OF PMU AFTER MEASURE PIN t: FORCING	STATE OF PMU AFTER MEASURE PIN t: PIN
SET TEST 4	ICEX	XPMU PIN	CPMU PIN t		
MEASURE PIN t		DISABLE RELAY	FORCE DELAY		
		FORCE VOLTAGE v, RNGx	MEASURE PIN	v VOLTAGE	PIN t
		SET PMU SENSE, RNGy			
		SET TEST 4			
		SET DCT Limit			
SET TEST 5	IIL(IFX)	XPMU PIN	CPMU PIN t		
MEASURE PIN t		DISABLE RELAY	FORCE VOLTAGE v, RNGx		
		FORCE VOLTAGE v, RNGx	MEASURE PIN		
		SET PMU SENSE, RNGy	FORCE VOLTAGE voh,RNGx		Set up pin
		SET S1 voh , RNGx	XPMU PIN	voh VOLTAGE	
		SET TEST 5			
		SET DCT Limit			
		XPMU PIN			
SET TEST 6	VF3	ENABLE RELAY	CPMU PIN t		
MEASURE PIN t		FORCE CURRENT i, RNGx	DISABLE RELAY		
		SET PMU SENSE, RNGy	FORCE CURRENT i, RNGx		
		SET TEST 6	MEASURE PIN		
		SET DCT Limit	FORCE CURRENT 0, RNGx	0 CURRENT	PIN t
			ENABLE RELAY		

FACTOR STATEMENT	TEST	SETUP PROCEDURE	MEASUREMENT SEQUENCE	STATE OF PMU AFTER MEASURE PIN t: FORCING	STATE OF PMU AFTER MEASURE PIN t: PIN
SET TEST 7	ISC	XPMU PIN	CPMU PIN t		
MEASURE PIN t		DISABLE RELAY	FORCE VOLTAGE v, RNGx		
		FORCE VOLTAGE v, RNGx	MEASURE PIN		
		SET PMU SENSE, RNGy	FORCE VOLTAGE ovh, RNGx		
		SET S1 voh, RNGx	XPMU PIN	voh VOLTAGE	Set up PIN
		SET TEST 7			
		SET DCT Limit			
SET TEST 9	IOL	XPMU PIN	CPMU PIN t		
MEASURE PIN t		DISABLE RELAY	FORCE VOLTAGE v, RNGx		
		FORCE VOLTAGE v, RNGx	MEASURE PIN		
		SET PMU SENSE, RNGy	FORCE VOLTAGE vol,RNGx		
		SET S0 vol, RNGx	XPMU PIN	vol VOLTAGE	Set up PIN
		SET TEST 9			
		SET DCT Limit			
		XPMU PIN			
SET TEST 10	VBD	ENABLE RELAY	CPMU PIN t		
MEASURE PIN t		FORCE CURRENT i, RNGx	DISABLE RELAY		
		SET PMU SENSE, RNGy	FORCE CURRENT i, RNGx		
		SET TEST 10	MEASURE PIN		
		SET DCT Limit	FORCE CURRENT 0, RNGx		
			ENABLE RELAY	0 CURRENT	PIN t

Appendix A
Character Coding (TRASCII)

Code	Char.	029 Special Character	Code	Char.	029 Special Character
00	SPACE	BLANK	40	@	4-8
01	!	11-2-8	41	A	12-1
02	"	7-8	42	B	12-2
03	#	3-8	43	C	12-3
04	$	11-3-8	44	D	12-4
05	%	0-4-8	45	E	12-5
06	&	12	46	F	12-6
07	'	5-8	47	G	12-7
10	(	12-5-8	50	H	12-8
11	)	11-5-8	51	I	12-9
12	*	11-4-8	52	J	11-1
13	+	12-6-8	53	K	11-2
14	'	0-3-8	54	L	11-3
15	-	11	55	M	11-4
16	.	12-3-8	56	N	11-5
17	/	01	57	O	11-6
20	0	0	60	P	11-7
21	1	1	61	Q	11-8
22	2	2	62	R	11-9
23	3	3	63	S	0-2
24	4	4	64	T	0-3
25	5	5	65	U	0-4
26	6	6	66	V	0-5
27	7	7	67	W	0-6
30	8	8	70	X	0-7
31	9	9	71	Y	0-8
32	:	0-8-2	72	Z	0-9
33	;	11-6-8	73	[	12-4-8 <
34	<	12-0	74	\	11-7-8 ¬
35	=	6-8	75	]	0-6-8 >
36	>	11-0	76	↑	12-7-8 |
37	?	0-7-8	77	←	0-5-8 —

Appendix B
Reading And Writing Of Long And Short Registers

B.1 INTRODUCTION

The Sentry 200 Test System READ and WRITE capabilities reference the system's long and short registers. (The long and short registers consist of one bus each). The following is a description of the operation of the long and short registers.

B.1.1 Long Registers

The long registers are interfaced to the memory interface unit, called the Instruction Register, which sends information to and from the test station. It has a bus, which is used primarily for transmitting functional test data and PMU control data to the test station. The S, F, D, M, C and R registers are the only registers programmable with functional test data. There are other registers that are essentially like the short registers, but since the hardware resides in the test station, they are interfaced to the long register data bus and use rank address bits for identification.

B.1.2 Short Registers

The short register is interfaced to the computer accumulator. The register is used for controlling the digital to analog converter subsystems and for communicating the tester status, mode and interrupt information. Therefore, the E1, E0, EA1, EA0, etc., registers, that contain data for reference supplies are interfaced to the short register data bus.

B.2 ADDRESSING SHORT REGISTERS

Each register can be addressed by the three computer Select Peripheral Unit (SPU) commands: READ, WRITE and SPECIAL. They are each discussed as follows:

READ

To examine the contents of a register, the register is first addressed with an SPU READ command. The contents of the register are then read into the CPU accumulator.

WRITE

To write a bit pattern into a register, the register is first addressed with an SPU WRITE command. The command is followed by the bit assignment for that register.

SPECIAL

An SPU SPECIAL command is defined as an instruction that executes some function, but no read or write data transfer is involved, e.g., Increment IND Counter or Disconnect DPS.

Each register command consists of an 8 digit octal code. The code, in effect, identifies a specific register in a specific unit and informs this register that it is about to either receive or transmit data. The data will be either read out of the register, written into the register or the register will perform a special function. The command format is shown in Table B-1.

Consider an example of an SPU command so that its mode and function within the system can be understood. The following table shows the MODE register SPU READ command and its octal and binary equivalents:

TABLE B-1. SPU COMMAND FORMAT

Bit Location:	23 22 21	20 19 18	17 16 15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
Octal Value :	0	6	6	0	0	5	2	0
Binary Value:	0 0 0	1 1 0	1 1 0	0 0 0	0 0 0 1	0 1 0	1 0 0	0 0 0
	OP CODE (SPU)		6=Read 4=Write 2=Special 0=No-op		TESTER REGISTER		UNIT ADDRESS	

Starting from the left, the high-order six binary bits (23-18) represent the octal code 06. This octal code is the SPU op code for the READ, WRITE, or SPECIAL command functions. The op code

informs the system that it is about to address a register in a unit with either READ, WRITE or SPECIAL information.

The 3-bit value in bits 17-15 defines the command as READ, WRITE, or SPECIAL transfer. Octal 6 = READ: 4 = WRITE; 2 = SPECIAL and 0 = No op.

The six bits shown under tester register (13-8) specify one of 64 unique registers. The remaining bits (7-0) are used to form the unit address. The tester is unit 120B.

B.2.1 Short Register Descriptions

The register number is the octal equivalent of bits 13-8 of the SPU command. The following paragraphs summarize the short registers, their addresses and special functions.

B.2.1.1 Mode Register (MR) Address 01

The mode register controls modal functions affecting the total test system as shown in Table B-2.

TABLE B-2. MODE REGISTER

BIT	FUNCTION	Read	Write
0	Reset Tester Short Reg.		x
1	Reset Tester Long Reg.		x
2	Monitor Mode	x	
3	Auto Mode	x	
4	Negative Logic Mode	x	x
5	Latch C Mode	x	x
6	Strobe Inhibit Mode	x	x
7	Force Strobe		x
8	Force Sync Pulse		x
9	DMA Mode		x
10	DMCRS Last function test register used	x	x
11	Function Test Suspended	x	x
12	Trip Fail	x	x
13	Functional Fail	x	x
14	Pass (Cleared by 2 or 13)	x	x
15	Spare	x	x

NOTE: A SPECIAL command clears the mode register.

B.2.1.2 Status Register (SR) Address 02

The status register contains interrupt information as shown in Table B-3.

TABLE B-3. STATUS REGISTER

BIT	FUNCTION	Read	Write
0	Instruction Number Compare Interrupt Enable	x	x
1	Instruction Number Compare Interrupt	x	x
2	Delay Complete Int. Enable	x	x
3	Delay Complete Int.	x	x
4	Trap Interrupt Enable	x	x
5	Trap Interrupt	x	x
6	Fail Interrupt Enable (See B.3.3.18)	x	x
7	Fail Interrupt (See B.3.3.18)	x	x
8	Trip Interrupt Enable	x	x
9	DPS #1 Trip (8 must be on)	x	x
10	DPS #2 Trip (8 must be on)	x	x
11	DPS #3 Trip (8 must be on)	x	x
12	Stop Interrupt Enable	x	x
13	Stop Interrupt	x	*
14	Interrupt in Process	x	
15	Spare	x	x
16	Time Fail Interrupt Enable	x	x
17	Time Fail Interrupt	x	x
	*Set by short register reset.		
NOTE: A SPECIAL command clears the status register. (B0-B13)			

B.2.1.3 Instruction Register (IR) Address 03

The instruction register is a buffer between B memory and the long registers via the accumulator. It contains the information listed in Table B-4.

TABLE B-4. INSTRUCTION REGISTER

BIT(S)	FUNCTION	Read	Write
0	Data	x	x
.	.	.	.
.	.	.	.
.	.	.	.

TABLE B-4. INSTRUCTION REGISTER (Continued)

BIT(S)	FUNCTION	Read	Write
14	Data	x	x
15-18	Rank Address		x
19-21	Register Address		x
22-23	Long Reg. Read/Write Control		x

00 = WRITE & HOLD BITS 0-14
01 = WRITE & EXEC BITS 0-14
10 = READ BITS 0-14

WRITE & EXEC in DMA mode advances the Instruction Number Counter (IND) and waits for 'tester not busy;

B.2.1.4 Memory Address Register (MAR) Address 04

The memory address register contains the memory address for the tester DMA mode. The fourteen bits are all read and write. When the tester is in the DMA mode, phase loop control automatically increments MAR.

B.2.1.5 Test Station Control Register (TSC) Address 05

The test station control register controls four channel multiplexing as listed in Table B-5.

TABLE B-5. TEST STATION CONTROL REGISTER

BITS	FUNCTION	Read	Write
0-1	Station Address (Write places station on-line)	x	x
2-5	Start Requests from Stations 1 to 4	x	
6-9	Manual Mode from Stations 1 to 4	x	
10-13	Reset Request from Stations 1 to 4	x	

Reset and Start are writable only by addressing the associated station.

B.2.1.6 Clock Burst Count Register (CBC) Address 10

The clock burst count register consists of eight bits, all read/write, which contain the count of the number of clock syncs generated per function test.

B.2.1.7 Time Delay Register (TD) Address 11

The time delay register consists of fourteen bits, all read/write, representing the value of a functional or DC time delay to be generated by certain tester instructions. For function tests, the least significant bit represents 0.35 microsecond and full scale is 5.734 milliseconds. The phase loop counter triggers the time delay counter when a SET F instruction is executed. For DC tests, the least significant bit represents 0.35 millisecond and the full scale value is 5.734 seconds. An SPU SPECIAL command triggers the DC time delay.

B.2.1.8 Instruction Number Display Register (IND) Address 14

The instruction number display register is a sixteen bit register, all sixteen bits read/write, representing the test instruction being executed. An SPU SPECIAL command increments the contents of the register by one. Also incremented in DMA mode when bits 23 and 22 = 01, WRITE and EXECUTE, are set as shown.

B.2.1.9 Instruction Number Compare Register (INC) Address 15

The instruction number compare register consists of sixteen bits, all read/write, representing the test instruction number at which a pause or external sync pulse occurs. A compare interrupt is generated if the INC Interrupt is enabled, else a sync pulse occurs.

B.2.1.10 Digital Programmable Power Supply Registers DPS1, DPS2, and DPS3 Addresses 21, 22, 24

Registers DPS1, DPS2 and DPS3 contain the range, polarity and magnitude of the DPS voltage being forced or the voltage trip point (Refer to Table B-6).

TABLE B-6. DIGITAL PROGRAMMABLE POWER SUPPLY REGISTERS

BITS	FUNCTION	Read	Write
0-9	Voltage Magnitude (Forced or Trip Value) LSB = 0.01 volt in low range = 0.04 volt in high range	x	x
10	Polarity 0 = Pos 1 = Neg	x	x
11	Range 0 = low 1 = high	x	x
NOTE:	An SPU SPECIAL command disconnects the corresponding supply. A DPS write connects the unit to the load board.		

B.2.1.11 DPS Trip Registers - DPT1, DPT2 and DPT3 Addresses 23, 25, 26

Registers DPT1, DPT2 and DPT3 contain the current trip point or the current being forced and the trip greater than or less than control plus the DPS forcing mode control (Refer to Table B-7).

TABLE B-7. DPS TRIP REGISTERS

BIT(S)	FUNCTION	Read	Write
0-9	Current Magnitude (Forced or Trip Value) LSB = 0.1mA in low range = 1. mA in high range	x	x
10	Polarity 0 = Pos 1 = Neg		
11	Range 0 = low 1 = high	x	x
13	GT or LT:1 = GT 0 = LT	x	x
14	Voltage/current 0 = voltage force 1 = current force	x	x

B.2.1.12 Reference Voltage Supply Registers S0, S1, E0, E1, EA0, EA1, EB0, EB1, EC0, EC1, SA0, SA1 Addresses 32-37, 42-47

The reference voltage supply registers contain the range, polarity and magnitude of the reference voltage supply (Refer to Table B-8).

TABLE B-8. REFERENCE VOLTAGE SUPPLY REGISTERS

BIT(S)	FUNCTION	Read	Write
0-9	Voltage Magnitude LSB = 0.01 volt in low range LSB = 0.04 volt in high range	x	x
10	Polarity 0 = Pos 1 = Neg	x	x
11	Range 0 = low 1 = high	x	x

B.3 LONG REGISTER DESCRIPTION

The registers associated with the long register are divided into two groups. The first group consists of the D, M, S, R, F and C registers. The second group consists of Pin Address, Statement Number Display, Precision Power Source, Precision Sense Level, External Interface, Slave TSC, DC Trip, Status and Mode A and B registers, and Long Register Extend register.

B.3.1 The D, M, S, R, F and C Registers

The six registers of the first group are discussed below.

B.3.1.1 D Register (Address 02)

The D register is termed the input/output register. If the D register is programmed as a binary 1, the associated pin is defined as an input pin. If the D register is programmed as a binary 0, the associated pin is defined as an output pin. When a pin is defined as an input pin in the D register, a relay is energized to connect the output of the driver to the pin.

B.3.1.2 M Register (Address 04)

The M register is the care/don't care or "mask" register. If the programmer is interested (care) in knowing the output level of a pin, the M register is programmed as a binary 1. If the programmer is not interested (don't care) in the output level of a pin, the M register is programmed as a don't care, or a binary 0. An input pin, or an output pin with an undefined state, would normally be programmed as don't care to prevent false failure indications on that pin. The don't care condition inhibits any fail indication from the output of the detector.

B.3.1.3 S Register (Address 10)

The select reference register selects which set of reference supplies are to be used by the functional test driver for each tester pin. A binary 0 selects the primary reference pairs, while a binary 1 selects the alternate reference pairs.

B.3.1.4 R Register (Address 14)

The utility relay register controls the utility relays, one relay per tester pin. A binary 1 indicates a closed relay and a binary 0 indicates an open relay. The utility relays can be used for such functions as connecting a load resistor for an output pin to a programmable power supply.

B.3.1.5 F Register (Address 06)

The F register contains the logic patterns 1 or 0 to be applied to those pins which are defined as input pins by the D register. If the F register is programmed as a high level (1), the F register will cause the high output of the driver to be applied to the associated pin. If the F register is programmed with a low level (0), the low output of the driver will be applied to the pin. The F register also contains the expected logical output of those pins which are defined as output pins.

B.3.1.6 C Register (Address 12 (Read Only))

The C register stores the go/no-go results of a comparison between the actual output of a device and the expected output. A binary 1 represents a comparison failure and a binary 0 represents a pass condition.

B.3.2 Format of Functional Test Word

The primary difference between the long and short registers is that the short registers consist of fixed bit test words. The long registers are variable length words. The format of a 24-bit function test word is discussed below and it is illustrated in the accompanying Figure B-9. The format for all function test registers is the same, except that the address for each register is different.

TABLE B-9. TEST WORD FUNCTION FORMAT

24 Bit Functional Test Word

23 22	21 20 19	18 17 16 15	14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
Control	Register Address	Rank Address	Pin Data Field

Rank Address

3 Bits = 1 to 8 Ranks
Maximum Register Length = 8 x 15 = 120 Bits

Register Address

Bits: 21	20	19	18	Meaning
0	0	1	0	D DEFINE I/O PINS
0	1	0	0	M MASK
0	1	1	0	F FUNCTIONAL PATTERN
1	0	0	0	S SELECT ALTERNATE REFERENCE
1	0	1	0	C COMPARE (FAIL PATTERN)
1	1	0	0	R UTILITY RELAY
1	1	1	X	TEST STATION REGISTERS

Control

Bits: 23	22	Meaning
0	0	WRITE AND HOLD
0	1	WRITE AND EXECUTE
1	0	READ
1	1	TRAP

Register Load Times

1.75 (1 + N) microseconds
where: N = Number of Ranks Changing

Starting from the right, the 0 bit represents pin 1, the 1 bit represents pin 2, etc., up to bit 14 which represents pin 15.

Rank Address:

Bits 15 through 17 represent the rank to which the first 15 bits have been assigned. The ranks are determined by the normal 4-2-1 binary code minus one. 000 inserted in bits 15 through 17 represent rank 1; 111 represent rank 8. In this manner, 8 ranks of 15 pins can be programmed, thus providing a capacity of 120 pins.

Register Address:

Bits 18, 19, 20 and 21 determine the register to which the rank of 15 bits is to be sent.

Control:

Bits 22 and 23 control the read/write function. Assume the device under test is a 15-pin device; all 15 pins will be programmed on one line and will be assigned to rank 1. The correct address code is inserted and bits 22 and 23 are programmed as 01 (write and execute), i.e., there are no more pins to program. If the device is a 45-pin device, the first 15 pins are programmed as described above, except for bits 22 and 23 which are programmed as 00 (write and hold), i.e., there are more pins to program. The second group of 15 pins is assigned to rank 2 and, again, bits 22 and 23 are programmed as 00, i.e., there are more pins to program. The third group of 15 pins is assigned to rank 3 and bits 22 and 23 are programmed as 01 (write and execute), i.e., there are no more pins to program-execute the command. The F and S registers are MASTER/SLAVE so that all bits execute together.

B.3.3 Special Test Station Registers

The registers of the second group are discussed below.

B.3.3.1 Pin Address Register (Address 160)

The pin address register addresses the precision measuring unit to one of the pins of the device under test or to an internal node (Refer to Table B-10). The reset state of this register is 0.

TABLE B-10. PIN ADDRESS REGISTER

BIT(S)	FUNCTION	Read	Write
0-3	Pin Number 1-15 0000 = disconnect 0001 = pin 1	x	*
4-6	Rank Number 1-8 000 1111 = pin 15 001 0001 = pin 16	x	*
7	Internal Node Address	x	*
8	Connect Voltage Conditioner (Enable Relay FACTOR Instruction)	x	**
14	Write Protect Bit		x
	*Write Protected if bit 14 is a 1. **Write Protected if bit 14 is a 0.		

B.3.3.2 Socket ID (Address 161)

The socket ID reads a hard wired address on the load board so that a FACTOR program can compare the load board ID with the program ID.

The register is 12 bits, read only.

B.3.3.3 Statement Number Display Register (Address 162)

The statement number display register contains the statement number to be displayed on the test station control panel. This register is interfaced to the IND register with software in TOPSY. Whenever the test sequence pauses, the software updates SND. The register is 15 bits, both read and write.

B.3.3.4 Clock and Strobe Register (Address 163)

The clock and strobe register is actually two registers under one address. The first part, clock address, is bits 0-3. The clock address bits are ANDed with the first four bits of the F register to generate clock sync signals, (see Enable Clock Description, 11.5.2). The second part, Enable Strobe, is bits 4-7, (see the Enable Strobe Description, 11.4.7). All eight bits are read/write.

B.3.3.5 Precision Power Source Register/Precision Measurement Unit Forcing Register (Address 164)

This register contains the magnitude, polarity and range information of the PMU forcing value. It also contains the voltage or current force mode bit (Refer to Table B-11).

TABLE B-11. PPSR/PMUF REGISTER

BIT(S)	FUNCTION			Read	Write
0-9	Magnitude			x	x
	LSB	Range	Full Scale		
	1mV	1	1.023V		
	10mV	2	10.23V		
	40mV	3	40.92V		
	1nA	0	1.023uA		
	100nA	1	102.3uA		
	10uA	2	10.23mA		
	100uA	3	102.3mA		
10	Polarity		0 = POS 1 = NEG	x	x
11-12	Forcing Range		00 = Range 0 (Current Force only) 01 = Range 1 10 = Range 2 11 = Range 3	x	x
13	VF/IF		1 = Voltage Force 0 = Current Force	x	x

B.3.3.6 Precision Sense Level Register (Address 16)

This register contains the PMU voltage clamp levels and the measuring range (Refer to Table B-12).

TABLE B-12. PRECISION SENSE LEVEL REGISTER

BIT(S)	FUNCTION	Read	Write
0-3	Voltage Clamp Magnitude	x	*
4	Clamp Control (1 = on, 0 = off)	x	*
5	Clamp Range	x	*
	Range Clamp Voltage Range of Values		
	0 (1.5+3n)V $0 \leq n \leq 15$ 1.5 to 46.5		
6	Allow Negative Voltages (SET CLAMP NEG)	x	*
7	Allow Positive Voltages (SET CLAMP POS)	x	*
8-9	Not used.		
10	Write Protect Bit		
11-12	Sense Range	x	**
13	VM/IM (complement of PPS bit 13)	x	
	*Write Protect if bit 10 is a 1.		
	**Write Protect if bit 10 is a 0.		

B.3.3.7 External Interface Register (Address 166)

This fifteen bit register is used to display test results and control external handlers (Refer to Table B-13). Bits 0-9 are available to the programmer to use in any form, such as to define various pass categories. Bits 10-14 are defined by system software. All bits are read/write. If the user wishes to use some bits to read the status of external equipment, then a simple hardware modification can be made to the register by disconnecting the bit storage device from the register. Consult your field service representative if this is desired.

TABLE B-13. EXTERNAL INTERFACE REGISTER

BIT(S)	FUNCTION	Read	Write
0-9	Defined by User		
	Displayed on Station Control Panel	x	x
10	D.C. Fail Lamp*	x	x
11	D.C. Pass Lamp*	x	x
12	Functional Fail Lamp	x	x
13	Functional Pass Lamp	x	x
14	End-of-Test	x	x
	*If both DC fail and DC pass are set, the Terminal Error lamp will go 'ON'.		

B.3.3.8 Slave Test Station Control (Address 167)

This register is a copy of the TSC short register and is used at each station equipped with relay mux. It is used to put the head on line; and contains Reset Manual and Start data from each of the sub station test heads.

B.3.3.9 DC Trip Limit Register (Address 171)

The DC trip limit register holds a go/no-go limit for PMU tests made in DMA mode (MEASURE PIN). It is also used to hold trial limits in the software A to D conversion routine. (Reference Table B-14).

TABLE B-14. DC TRIP LIMIT REGISTER

BIT(S)	FUNCTION	Read	Write
0-9	Magnitude (See PPS register)	x	*
10	Polarity (0 = POS, 1 = NEG)	x	*
11-12	Sense Range (Set by PSL register)	x	
13	LT/GT Bit (0 = LT, 1 = GT)	x	*
14	D.C. Fail	x	
14	D.C. Strobe		x
	*Write Protect if DC strobe is 1.		

NOTE:

A DC strobe is generated either by writing bit 14 or reading the register. A comparator fail on a strobe sets on a latch on bit 14; a comparator pass during a strobe resets the latch.

B.3.3.10 Status and Mode Register A (Address 1730)

The Status and Mode A Register contains control and status information to supplement the functional/DC fail interrupt in the status register (Reference Table B-15).

TABLE B-15. STATUS AND MODE REGISTER A

BIT	FUNCTION	Read	Write
4	Functional Fail	x	
5	Parametric Fail	x	
6	Functional Fail Enable	x	x
7	Parametric Fail Enable	x	x

B.3.3.11 Status and Mode Register B (Address 1734)

The Status and Mode B Register allows control of Interrupt Enable for a DC pass. This register may not be read or written from a FACTOR program. (Reference Table B-16.)

TABLE B-16. STATUS AND MODE REGISTER B

BIT	FUNCTION	Read	Write
7	D/L Measure Interrupt Enable	x	x

B.3.3.12 Long Register Address Extend (Address 1737)

The Long Register Address Extend register is used to control long register address bank switching. This register is used only with the Time Option.

B.4 FORMATTING OF FACTOR WRITE AND READ STATEMENTS

The format for programming the short or long registers is:

```
WRITE (XXXXB) expression;
```

where: XXXX is any register number, and
B is the octal indicator.

Reading information from a short or long register is:

```
READ (XXXXB) Z;
```

Tables B-17 and B-18 provide the necessary information for reading from or writing to a specific register for a specific function on the short and long registers.

TABLE B-17. SHORT REGISTER READING AND WRITING CODES

A=2= SPECIAL
A=4= WRITE
A=6= READ

Reg. No.	Register	X X X X	SPECIAL Function
0	No-op		
1	MODE	A 0 0 2	Clear Status Reg
2	STATUS	A 0 0 4	Clear Status Reg
3	Instruction	A 0 0 6	
4	Memory Address	A 0 1 0	
5	TSC	A 0 1 2	
10	Clock Burst Count	A 0 2 0	
11	Time Delay	A 0 2 2	Start D.C. Delay
14	Instruction Number Display	A 0 3 0	
15	Instruction Number Compare	A 0 3 2	
21	DPS1	A 0 4 2	Disconnect DPS1
22	DPS2	A 0 4 4	Disconnect DPS2
23	DPT3	A 0 4 6	
24	DPS3	A 0 5 0	Disconnect DPS3
25	DPT2	A 0 5 2	
26	DPT1	A 0 5 4	
32	E1	A 0 6 4	
33	E0	A 0 6 6	
34	S1	A 0 7 0	
35	S0	A 0 7 2	
36	EA1	A 0 7 4	
37	EA0	A 0 7 6	
42	EB1	A 1 0 4	
43	EB0	A 1 0 6	
44	EC1	A 1 1 0	
45	EC0	A 1 1 2	
46	SA1	A 1 1 4	
47	SA0	A 1 1 6	

TABLE B-18. LONG REGISTER READING AND WRITING CODES

Register (Pins)		Register No.	Write X X X	Read X X X
D	1-15	020	220	420
D	16-30	021	221	421
D	31-45	022	222	422
D	46-60	023	223	423
D	61-75	024	224	424
D	76-90	025	225	425
D	91-105	026	226	426
D	106-120	027	227	427
M	1-15	040	240	440
M	16-30	041	241	441
M	31-45	042	242	442
M	46-60	043	243	443
M	61-75	044	244	444
M	76-90	045	245	445
M	91-105	046	246	446
M	106-120	047	247	447
F	1-15	060	260	460
F	16-30	061	261	461
F	31-45	062	262	462
F	46-60	063	263	463
F	61-75	064	264	464
F	76-90	065	265	465
F	91-105	066	266	466
F	106-120	067	267	467
S	1-15	100	300	500
S	16-31	101	301	501
S	31-45	102	302	502
S	46-60	103	303	503
S	61-75	104	304	504
S	76-90	105	305	505
S	91-105	106	306	506
S	106-120	107	307	507
C	1-15	120		520*
C	16-30	121		521*
C	31-45	122		522*
C	46-60	123		523*
C	61-75	124		524*
C	76-90	125		525*
C	91-105	126		526*
C	106-120	127		527*

C REGISTER:Read Only			
R 1-15	140	340	540
R 16-30	141	341	541
R 31-45	142	342	542
R 46-60	143	343	543
R 61-75	144	344	544
R 76-90	145	345	545
R 91-105	146	346	546
R 106-120	147	347	547
Pin Address	160	360	560
Socket ID	161	-	561
Statement Number Display	162	362	562
Clock and Strobe	163	363	563
Precision Power Source	164	364	564
Precision Sense Level	165	365	565
External Interface Register	166	366	566
Slave Test Station Control	167	367	567
DC Trip	171	371	571
Status & Mode A	173	373	573
Status & Mode B	1734	These registers cannot be	
Long Register Address Extend	1737	read or written from FACTOR	

Appendix C

Voltage And Current Range Definitions

MODULE	STATEMENT	PROGRAMMABLE VALUE/RESOLUTION			
		RANGE 0	RANGE 1	RANGE 2	RANGE 3
PMU	Force Voltage		0 to 1.023v/1 mV	0 to ±10.23V/10mV	0 to ±10.92V/10mV
PMU	Set PMU ForceV		0 to 1.023V/10mV	0 to ±10.23V/10mV	0 to ±10.92V/10mV
PMU	Force Current	0 to ±1.023uA/1nA	0 to ±1023mA/.1uA	0 to ±10.23mA/10uA	0 to ±102.3mA/.1mA
PMU	Set PMU ForceI	0 to ±1.023 A/1nA	0 to ±1023mA/1uA	0 to ±10.23mA/10uA	0 to ±102.3mA/.1mA
PMU	Set PMU Sense Voltage		0 to ±1.023V/1mV	0 to ±10.23V/10mV	0 to ±10.92V/10mV
PMU	Set PMU Sense Current	0 to ±1.023uA/1nA	0 to ±1023mA.1uA	0 to ±10.23mA/10uA	0 to ±102.3mA/.1mA
DPS	Force VF			0 to ±10.23V/10mV	0 to ±10.92V/10mV
DPS	Force IF			0 to ±102.3mA/.1mA	0 to ±1.023A/1mA
DPS	Enable Trip			0 to ±102.3mA/.1mA	0 to ±1.023A/1mA
DPS	Enable TripV			0 to ±10.23V/10mV	0 to ±10.92V/10mV
RVS	Set (S0/S1)		0 to ±10.23V/10mV	0 to ±30.0V10mV	
RVS	Force F		0 to ±10.23V/10mV	0 to ±30.0V/10mV	

Appendix D
DMA Mode Statements

S-200 FACTOR statements that cause a value to be loaded in a tester long register are normally executed in direct memory access (DMA) mode. Briefly, in this mode, the system software determines the start address of a sequence of these statements, loads the MAR register, and initiates DMA mode. The hardware then executes the test program directly until an instruction that can not be processed in this mode is encountered. Such an instruction may require several operations to be performed; these instructions are executed interpretatively by the system software. Execution in DMA mode is more efficient, particularly if the programmer structures his program so that long DMA sequences are not broken by interpretative statements or statement labels.

The following table indicates which FACTOR instructions are executed in DMA mode and which long registers may be affected.

TABLE D-1. STATEMENTS EXECUTED IN DMA MODE

INSTRUCTION	LONG REGISTER NUMBER	
SET D SET M SET F SET S SET R	020 through 027 040 through 047 060 through 067 100 through 107 140 through 147	Functional test instructions load from 1 to 8 ranks per statement depending on the specified pins.
CPMU PIN * XPMU PIN ENABLE/DISABLE RELAY	160 160 160	
SET PMU FORCEV/FORCEI* FORCE VOLTAGE/CURRENT*	164 164	

TABLE D-1. STATEMENTS EXECUTED IN DMA MODE (Continued)

INSTRUCTION	LONG REGISTER NUMBER
SET PMU SENSE	165
SET CLAMP number	165
MEASURE PIN	171, 173
SET DCT	171, 165

*Only when the expression is a simple constant. (If the expression must be evaluated at execution time, the statement is executed interpretively.

Appendix E
Time Delay Related Statements

FACTOR statements that start a software delay, i.e., cause the time delay register countdown to be initiated, must also be time delay dependent. This is necessary in order to avoid disturbing a previous countdown in progress, if any, which could have a longer net value than the countdown to be started. The following table lists all statements that are time delay dependent; the execution of these statements must be delayed until any previous hardware or software delay has expired. Instructions which generate time delays (a fixed number if hardware initiated or a variable if software initiated) are also listed. Note that x implies a DPS number 1 through 3 and y implies a reference voltage supply (RVS) level of 0 or 1.

TABLE E-1. TIME DELAY DEPENDENT STATEMENTS

FORCE WAIT
SET DELAY
ENABLE TRIPVx (if changing to current force mode)
ENABLE TRIPIx (if changing to voltage force mode)
FORCE VFx (after a FORCE IFx)
FORCE IFx (after a FORCE VFx or FORCE RESET)
FORCE VOLTAGE
FORCE CURRENT
FORCE PMU
SET PMU FORCEI/FORCEV/SENSE
SET F
SET D
SET M
SET R
SET S
CPMU PIN expression;
XPMU PIN
MEASURE PIN
MEASURE VALUE/NODE/PIN number
The first statement of any DMA mode sequence.*
*Refer to Appendix D.

TABLE E-2. TIME DELAY GENERATING STATEMENTS

FORCE DELAY SET S	Programmed DC Time Delay 0.28 millisecond
SET D, SET CLAMP CPMU PIN, XPMU PIN, ENABLE RELAY, DISABLE RELAY	0.56 millisecond
SET R	1.75 millisecond
SET PMU FORCEV, FORCEI, FORCE VOLTAGE, FORCE CURRENT FORCE PMU	Programmed DC Time Delay or 0.56 millisecond with no current range change or 4 millisecond (+1 millisecond) with current range change, whichever is greater.
SET PMU SENSE	0.56 millisecond with no current range change or 4 millisecond with current range change.
FORCE VFx, FORCE IFx, ENABLE TRIPx, ENABLE TRIPVx, XCON VFx, FORCE RESET	Programmed DC Time Delay or 5.37 milliseconds, whichever is greater.
SET Sy, SET SAY, FORCE Ey, FORCE EAy, FORCE EBy, FORCE ECy	Approximately 300 microseconds per volt of change or 0.56 millisecond, whichever is greater.
SET DCT, MEASURE PIN	56 microseconds
SET F	Programmed functional delay.

Appendix F
Execution Terminal Error Numbers

Certain set up and programming errors cannot be detected at compilation time; these errors are discoverable only while testing. The Terminal Error lamp will go 'ON' for errors described in the following table. The error number will be logged. Both the Parameter FAIL and Parameter PASS lights are 'ON', but the EOT light is 'OFF'. The error number is displayed (in binary format) in the EIR register, bits 5-10; least significant bit to the left (Bit 5).

TABLE F-1. TERMINAL ERRORS

ERROR NUMBER	MEANING
1	program not loaded
2	station disabled (power off)
3	magnitude or polarity error in pin number, or time delay
4	DMA did not start, - hardware fault
5	magnitude error in voltage, current or time (exceeds hardware limitations)
6	magnitude error in a programmed time value
21	current value not within set limits
22	voltage value not within set limits
23	improper pin address
24	voltage value exceeds limit
26	undefined opcode
31*	read (file skip forward executed)
33*	write (file skip backward executed)
35*	EOT tape on write (tragic error)
36*	EOT tape on read
37*	memory protect on tape read
40*	data count error less than 7 or greater than assigned array
42*	irrecoverable error
50	improper vector declaration
51	number of formal and actual parameters do not agree
52	subscript violation
53	empty stack

TABLE F-1. TERMINAL ERRORS (Continued)

ERROR NUMBER	MEANING
54	program too big
55	EOF on test program
56	illegal op code
57	improper vector initialization
58	I/O error
59	improper 'FOR' loop constants
60	assembly language program not found on disc
61	assembly language program too long
62	arithmetic overflow (illegal arithmetic computations)
67	DIF (disc input file) or DOF (disc output file) not open.
68	trying to read beyond EOF (end of file) of DIF (disc input file).
69	trying to write beyond EOF (end of file) of DOF (disc output file).
70	SENTRY 600 program on S200/400 test station or S200/400 program on S600 test station
71	local memory size requested S600 hardware
72	programmed timing generator width or delay programmed test period
74	programmed local memory address out of bounds as specified by 'SET PAGE' statement
75	10 MHz program on a 5 MHz station
80	array too small for binary card punch or binary card read

On terminal error 31, the tape is moved to the next tape file. On terminal error 33, the tape is moved back to the start of the last file. When start is pressed, the program will continue execution from these tape locations.

*Tape status issued is in octal. On terminal errors 35 or 36 a tape rewind is executed. The program is aborted on terminal error 35.

Appendix G

TABLE G-1. CALIBRATION RESISTOR TABLE

TVFY LOAD BOARD PIN	CALIBRATION RESISTANCE	VOLTAGE RANGE	CURRENT RANGE
1	10	1 (1V)*	3 (100mA)*
3	100	1	2 (10mA)*
11	10 K	1	1 (100 a)*
17	1 M	1	0 (1 a)*
3	100	2 (10V)*	3
7	1 K	2	2
13	100 K	2	1
19	10 M	2	0
5	400	3 (40V)*	3
9	4	3	2
15	400 K	3	1
21	40 M	3	0

* Full scale.

Appendix H
Internal Node Measurement

Internal nodes are listed in the table below. The PMU must be programmed to force zero current in range 2 before the PMU is connected to any internal node, including the load current nodes (143-145). The voltage sensing range and the limit is programmed according to the expected value.

TABLE H-1. INTERNAL NODES

NODE NUMBER			MEASURED PARAMETER
DECIMAL	OCTAL	NAME	DESCRIPTION
128	200	S1	COMPARATOR S1 REF. VOLTAGE
129	201	S0	COMPARATOR SO REF. VOLTAGE
130	202	E1	FORCING LEVEL E1 REF. VOLTAGE
131	203	E0	FORCING LEVEL E0 REF. VOLTAGE
132	204	EA1	FORCING LEVEL EA1 REF. VOLTAGE
133	205	EA0	FORCING LEVEL EA0 REF. VOLTAGE
134	206	EB1	FORCING LEVEL EB1 REF. VOLTAGE
135	207	EB0	FORCING LEVEL EB0 REF. VOLTAGE
136	210	EC1	FORCING LEVEL EC1 REF. VOLTAGE
137	211	EC0	FORCING LEVEL EC0 REF. VOLTAGE
140	214	VF1	VOLTAGE FORCING UNIT 1 OUTPUT VOL.
141	215	VF2	VOLTAGE FORCING UNIT 2 OUTPUT VOL.
142	216	VF3	VOLTAGE FORCING UNIT 3 OUTPUT VOL.
143	217	TRIP1	VF1 LOAD CURRENT
144	220	TRIP2	VF2 LOAD CURRENT
145	221	TRIP3	VF3 LOAD CURRENT

Load currents are proportional to the voltage drop across an internally connected resistor chosen such that the full scale measurement value is 1.023 volts. If the power supply is in range 3, 1 millivolt of voltage drop corresponds to 1 milliamp of load current. If the power supply is in range 2, 1 millivolt of voltage drop corresponds to 0.1 milliamp of load current.

Assume that a load current measurement with a limit of 100 milliamps is to be programmed. With the DPS in range 3, the limit is converted to a value of 100 ma. Then, the measurement may be made with the PMU sensing in range 1 and the correct magnitude of the current would be logged. If the DPS were in range 2, 100 milliamps of load current is converted to a limit value of 1000 millivolts. Hence, the programmed limit must be multiplied by 10 and the resultant logged value divided by 10 to convert the voltage measured back to load current.

Appendix I
Instruction List

The following instruction forms are allowed in programming the S-200.

BASIC STATEMENT FORMS

BLOCK

Creates groups of program statements.

```
SUBR  identifier;
SUBR  identifier (parameters);
FUNCT identifier (parameters);
```

Delineates a group of statements which can be repeated with a call statement.

END;

Closes BLOCK or subroutine or BEGIN.

```
CALL identifier;
CALL identifier (parameters);
```

Subroutine is executed and at completion control is returned to the calling routine.

INSERT filename;

Allows inclusion and compilation of the named source file at point specified.

NOISE XXX;

Words listed as noise may be used in any statement but will be ignored by the compiler. (Must not include reserved words.)

REM ------;

Allows user to give documentation which will be ignored by the compiler.

```
DCL V1, V2, VN;
DCL V1/VALUE1/, V2/VALUE2/;
DCL V1 [A1SIZE]/AVALUE1,AVALUE2,.../;
```

Declares variables which may be assigned values.

```
GOTO label;
```

Causes unconditional branch.

```
LABEL: - - - ;
```

An address is assigned to label to allow branching to label.

```
IF relation THEN statement;
IF relation THEN BEGIN - - - - - END;
IF relation THEN statement1 ELSE statement2;
```

Statements are executed if the 'if' condition is met.

```
FOR variable = expression THRU expression
BY increment DO statement;
```

Allows looping under control of a variable.

```
PAUSE expression;
```

Program pauses -- value of expression printed on POD

FORMS OF ARITHMETIC STATEMENTS

```
variable =  integer/integer-expression ;
variable = expression;
```

Variable is assigned a value.

ARITHMETIC EXPRESSIONS:

With parenthetical expressions:
Read from left to right only:

Arithmetic replacement statements may use the following operators:

+ ADDITION
- SUBTRACTION
* MULTIPLICATION
/ DIVISION
↑ EXPONENTIATION

Boolean replacement statements may use the following operators:

LT	LESS THAN
EQ	EQUAL TO
LEQ	LESS THAN OR EQUAL
GT	GREATER THAN
NEQ	NOT EQUAL
GE	GREATER THAN OR EQUAL
OR	INCLUSIVE OR
EOR	EXCLUSIVE OR
AND	LOGICAL AND
NOT	NEGATION

INPUT/OUTPUT STATEMENT FORMS

READ ((CR)/(TTK)/(MTR) "name"/(FDIF)/(PID)) V1, V2, .. VN,&V1,&V2, .. &VN;

Read numerical and literal data from specified device and assign to variables.

WRITE ((TTP)/(MTW) "name"/(FDOF)/(LP)/(POD)) V1,V2, .. VN,S1,S2, .. SN, &V1,&V2, .. &VN;

Write variables (V1 through VN), strings of alphanumeric data (S1 through SN) and literal variables (&V1 through &VN) to specified device.

WRITE ((TTP)/(MTW) "name"/(FDOF)/(LP)/(POD))/COLV1/V1,/COLV2/V2, .. /COLVN/VN,/COLS1/S1,/COLS2/S2, .. /COLSN/SN,/COL&V1/&V1,/ COL&V2/&V2, .. /COL&VN/&VN;

Write variables (V1 through VN), strings of alphanumeric data (S1 through SN) and literal variables (&V1 through &VN) to specified device (column formatted).

NOTE

> When reading data, literal variables may be specified freely mixed with numeric variables. When writing, alphanumeric strings may also be included (enclosed in quotes), and any variables and strings may be column-formatted.

ON DIFEOF, label;

Causes program branch to label when disc EOF (end of file) is read.

RESET DIF;

Re-opens disc input file (DIF), i.e. resets pointer to beginning of file.

WRITE (XXXXB) expression:

Write to long or short register.

READ (XXXXB) variable;

Read long or short register.

TESTER STATEMENTS

SET FORMS

SET DELAY expression (, DC);

Time delay register is loaded with value.

SOCKET ID number;

Number is compared to code on load board and terminal error is caused if they are not equal.

SET DCT [LT/GT] expression (,RNG0/1/2/3);

Sets a hardware pass-fail limit for one DCT threshold at a time, for 'MEASURE PIN'.

SET CLAMP [POS/NEG/SYM/OFF] number;

Sets limit on voltage allowed when current is forced.

SET LOGIC [POS/NEG] ;

Initializes functional test comparator logic for either positive or negative logic.

SET [S1/S0] expression (,RNG2/RNG3);

Reference supplies are set to value specified.

SET PMU [SENSE/FORCEV/FORCEI] (, RNG0/RNG1/RNG2/RNG3/AUTO);

Initializes PMU.

SET CLOCK expression;

Value is loaded into the clock burst counter register. (8 bits: maximum value = 255).

SET [D/M/S/R/F] (*) binary-pin-pattern ;

Definition, mask, select, relay, or function registers are set to pattern.

SET M binary-pattern ;

Sets pin mask register.

SET D binary-pattern ;

Sets I/O pin definition register.

SET VOFFSET number

Specifies an offset voltage to be added to all tester statements which control a voltage level.

ENABLE FORMS

ENABLE [ILO/IHI/VLO/VHI] [GT/LT] number;

Enables limit comparisons to be made on all programmed current/voltage operands prior to an instruction execution.

ENABLE [TRIPV1/TRIPV2/TRIPV3] [LT/GT] expression (,RNG2/RNG3);

Enables the voltage-trip detector of the corresponding current forcing unit.

ENABLE [TRIPI1/TRIPI2/TRIPI3] [LT/GT] expression (,RNG2/RNG3);

Enables current-trip detector of the corresponding voltage forcing unit.

[ENABLE/DISABLE] LATCHES;

Determines if C register is to be cleared prior to strobing functional test comparators.

ENABLE ACCESS

Forces a disc access to reload the core memory buffer.

[ENABLE/DISABLE] RELAY;

Determines if voltage conditioner will remain connected to a pin when the PMU is connected.

DISABLE TRIPS;

Clears trip limits set up with Enables.

ENABLE [DCT0/DCT1] [LT/GT] expression;

Forms a software pass-fail threshold, or if both DCT0 and DCT1 are specified, a pass-fail window, for 'MEASURE VALUE'.

DISABLE [DCT0/DCT1];

Disables comparison limits.

[ENABLE/DISABLE] COMPARATORS;

Determines if comparator outputs will be strobed to C register.

ENABLE STROBE;

Enables comparator strobe to be controlled by contents of F(1-4) and binary pattern.

ENABLE CLOCK [binary-pattern];

Enables clock signals to be connected to tester pins 1-4. Clock burst occurs when F register contains a corresponding '1' bit.

FORCE FORMS

FORCE [VF1/VF2/VF3] expression (,RNG2/RNG3);

Forces DPS voltage supply to value specified.

FORCE [IF1/IF2/IF3] expression (,RNG2/RNG3);

DPS unit is to force current specified.

FORCE [E0/E1/EA0/EA1/EB0/EB1/EC0/EC1] expression (,RNG2/RNG3);

Forces voltage conditioner reference supplies to programmed value.

FORCE PMU expression

Forces output of PMU to value specified.

FORCE VOLTAGE expression (,RNG1/RNG2/RNG3) ;

Forces PMU to voltage speciifed.

FORCE CURRENT expression (,RNG0/RNG1/RNG2/RNG3);

Forces PMU to current specified.

FORCE RESET;

Clears all programmable test conditions and causes a hardware reset.

FORCE DELAY;

Forces the time delay to occur and to wait until tester not busy.

FORCE WAIT;

Forces tester to wait until 'tester not busy'.

FORCE STROBE;

Forces a single strobe, transferring comparator output states to C register.

FORCE CLOCK;

Forces a single clock pulse at each of the 4 sync lines.

MISCELLANEOUS FORMS

ON [DCT/FCT/TRIP], label;

Causes program branch to label on failure.

XCON [VF1/VF2/VF3] ;

Specified voltage forcing unit is disconnected from the test head.

CPMU PIN expression ;

PMU is connected to pin specified.

XPMU PIN;

Disconnects PMU.

MEASURE PIN ;

Pass-fail comparison is made with programmed limit. No floating point conversion.

MEASURE [VALUE/NODE number] (,LOG);

Measurement is made and a software analog-to-digital conversion takes place, with result stored in global variable 'VALUE'.

CLEAR FAIL [DCT/FCT/TRIP];

Previous fail indicator is cleared.

KEY

[X/Y/Z]	one of options is required.
(X/Y/Z)	one of options may be used but none is required.
integer	user must select appropriate expression or number.
number	any floating point number but may not be a variable.
expression	any floating point number or variable or arithmetic combination of numbers and variables.

Appendix J
Read/Write Magnetic Tape Statements

J.1 DEFINITION

The FACTOR READ (MTR) and WRITE (MTW) statements are defined as follows:

```
READ (MTR) "name" V1, V2, V3, V4;

WRITE (MTW) "name" V1, V2, V3, V4;
```

The terms V1 through V4 represent array identifiers which have been declared prior to executing the READ/WRITE statements. There may be one to four arrays per statement. The term "name", enclosed by double quotations specifies the file name of the data to be written on magnetic tape.

Execution of the WRITE (MTW) statement causes the Array Data Segment(s) to be written on magnetic tape at the tape's current position. Figure J.1 gives the format specification of an Array Data Segment.

An EOF (End of File) tape mark is written under the following conditions:

a) When End of Test occurs and the tester is in automatic mode, and at least one WRITE (MTW) statement has been executed.

b) At the completion of each WRITE (MTW) statement when the tester is in manual mode.

c) When the tester pauses as the result of a TOPSY "PAUSE" command or a FACTOR "PAUSE" and at least one WRITE (MTW) statement has been executed.

Only one magnetic tape unit may be used with the SENTRY-200 even though the system may have more than one test station. Any of the four stations which execute programs containing READ (MTR) and/or WRITE (MTW) statements will have access to the magnetic tape unit. To avoid having read/write conflicts which could destroy valid data, only one station of a multiple station system should execute programs which utilize magnetic tape.

TABLE J-1. ARRAY DATA SEGMENT

Physical Record Number	Word Number	Contents
1	1	8 character TRASCII code word for the file "name".
	2	
	3	Data record length = N (integer)
	4	0
	5	0
	6	0
2	1	Words 1 to N are the contents of one variable length FACTOR array. (FST-1 floating point)
	2	
	.	
	.	
	.	The maximum number of words per each record is limited to 512, (but must not be fewer than 7).
2	N	

J.2 READ ERRORS

J.2.1 Array Element Count Error

If the word count of the tape data exceeds the number of elements in the specified array(s) or if the declared array has less than seven (7) elements, the system issues terminal error 40. If the array size is less than 7 elements, the tape is not advanced. When the tape data word count exceeds the array size, the tape will have advanced to the end of the excessive tape segment prior to accepting the next station "START".

J.2.2 Data Transfer Error

If a data transfer error is detected, terminal error 31 is issued and the tape is positioned forward to the beginning of the next file. The TOPSY program statement counter is reset such that when station "START" is depressed, the loaded program will begin execution at statement one (1).

J.2.3 End of Tape Error

If the End Of Tape (EOT) mark is encountered before the specified segment is found, the tape is rewound to the Beginning Of Tape (BOT) mark and terminal error 36 is issued. The TOPSY program statement counter is reset such that when station "START" is depressed, the loaded program will begin execution at statement one (1).

J.2.4 Memory Protect

If the memory protect switch located on the tape controller is enabled, the system issues terminal error 37 and the TOPSY program counter is reset such that when station "START" is depressed, the loaded program will begin execution at statement one (1).

J.3 WRITE ERRORS

J.3.1 Data Transfer Error

If a data transfer error is detected, terminal error 33 is issued and the tape is positioned backwards to the start of the current file. The TOPSY program counter is reset so that when station "START" is depressed, the loaded program will begin execution at statement one (1).

J.3.2 End of Tape Error

If the End Of Tape (EOT) mark is encountered prior to completion of a WRITE operation, the tape is rewound and the system issues terminal error 35. The station is unloaded so that it cannot be restarted by pushing station "START". This avoids accidental writing over good data at the beginning of the tape.

J.3.3 Array Element Count Error

If an array of size less than seven (7) appears in the WRITE statement, terminal error 40 is issued, (see below J.3.4).

J.3.4 Unrecoverable Errors

Any errors other than those described above are considered to be unrecoverable and the system issues terminal error 40. The TOPSY program statement counter is reset.

J.4 STANDARD MAG TAPE OPERATION IN TOPSY

J.4.1

Before executing a program employing mag tape read or write statements, the operator must set the tape at the BOT marker of the tape file the program is to read or write.

J.4.2

The instructions relating to the periodic maintenance of the mag tape should be attended to if error free operation is desired.

J.4.3

Before executing the TOPSY program, the REMOTE switch on the mag tape unit must be enabled.

J.4.4

After steps J.4.1 through J.4.3 it is only necessary to execute the TOPSY program from the tester station. All mag tape controls are performed by TOPSY.

J.5 UNUSUAL MAG TAPE OPERATION IN TOPSY

J.5.1 Catastrophic Errors

If a 'catastrophic' error occurs during mag tape operation and the user desired to make some attempts to recover then the following course of action is recommended as a desperation procedure.

J.5.1.1 Write Operation

Go back to DOPSY manually and execute two tape mark writes, viz:

```
// MTAP TMARK (twice),
```

followed by:

```
// MTAP SKIP BACK 1 RECS
```

J.5.1.2 Read Operation

Go back to DOPSY manually. Rewind the tape via the tape transport REWIND switch and restart TOPSY.

J.5.1.3 Warning

The user should be aware that these recovery actions bypass the normal TOPSY-DOPSY return and, consequently, do not update the present state of TOPSY. When TOPSY is reentered, it is initialized to the state prior to the last return to DOPSY.

Appendix K
Factor Syntax Table

This appendix provides a Factor Syntax Table. To use the table, locate the Syntactical Entity that is to be defined, and read the applicable references in the Definition column. To define each reference listed in this column, simply locate the applicable reference in the Reference column, and read the definition in the Definition column.

Metasymbols used within the table are defined below:

METASYMBOL	MEANING
|	The vertical line means 'or'.
θ	Theta means the previous entity for that applicable definition referenced.
<>	Entities enclosed within angle brackets are optional.
...	Three periods mean continuous, but self-evident entities.
{}	Braces mean specify one and only one of the enclosed entities.

NOTE

Brackets, parentheses, periods, commas, colons, single quotes, the slash, and semi colons are required and are not syntax. Arithmetic symbols in the table follow conventional Factor meanings.

Syntactical Entity	Definition	Reference
Octal digit	0|1|2|3|4|5|6|7	O
Decimal digit	0|8|9	d
Letter	A|B|C|...|X|Y|Z|$	ℓ
Identifier	ℓ|θd|θℓ	1
Variable	1|1 [19]	2
String	'(Character set-quote)'	3
Octal integer	OB|Oθ	4
Decimal integer	d|dθ	5
Decimal fraction	.5	6
Decimal number	5|6|5 6	7
Unsigned number	7|7 E 5|7 E{+|-} 5|4	8
Actual parameter list	19|θ, 19	9
Subroutine reference	1 (9) | 1	10
Function reference	1 (9)	11
Primary term	2|8|11|(19)	12
Complement term	12|NOT 12|-12	13
Power term	13|θ↑13	14
Multiplying term	14|θ{*|/} 14	15
Adding term	15|θ {+|-} 15	16
Relation term	16|θ{LT|LE|EQ|NEQ|GE|GT} 16	
Intersection term	17|θ AND 17	18
Expression	18|θ{OR|EOR} 18	19
NOISE statement	NOISE 1|θ,1	21
REM statement	REM followed by anything but END, ELSE OR ';'	22
Assignment statement	2 = 19	23
GOTO statement	GOTO 40	24
CALL statement	CALL 10	25
PAUSE statement	PAUSE 19	26
IF statement	IF 19 THEN 39<ELSE 39>	27
FOR statement	FOR 2 = 19 THRU 19 <BY 19> DO 39	28
Output statement	WRITE (35) 36| WRITE 36	29
Input statement	READ (37) 38| READ 38	30
DCL statement	DCL 34|θ, 34	31
Tester statement	Res word <arg><exp><modify>;	32
DCL Initialization part	8| {+|-} 8|θ,θ	33
DCL element	2|2/33/	34
Output device	{TTP|LP|MTW 3|POD|EIR|DOF}|8	35

FACTOR SYNTAX TABLE (Continued)

Syntactical Entity	Definition	Reference
Output list	3|2|θ/5/|θ,θ	
Input device	{TTK|TTR|CR|MTR 3|PID|DIF|EIR| BCR}|8	37
Input list	2|θ, 2	38
Basic statement	22|23|...|32|43|44|46|47	39
Label	1:	40
Statement	39|40: 39	41
Program	<43|44> 41 49	42
Compound state	BEGIN 41 49	43
BLOCK	BLOCK 41 49	44
Argument	1|θ, 1	45
Function block	FUNCT 1 (45) 41 49	46
Subroutine block	SUBR 1;41 49|SUBR 1 (45) 41 49	47
Terminator	END;	49

Appendix L
Floating-Point Package

Calling sequences and timings for the individual subroutines of the floating-point package, and the internal format and range of floating-point values are discussed below.

Calling Sequences:

Type 1 Subroutines:

FCAM, FMUL, FDIV, FSUB, FADD, FAND, FOR, FEOR

All of these subroutines require two floating-point parameters. Value 1 must be in the A-register, value 2 is obtained indirectly, (the floating-point result is returned in the A-register):

```
CALL FPSUBR  call each subr by name with value 1 in A-reg
NOP  ADDRV2  address of value 2 in address field
```

Type 2 Subroutines:

(FNOT, FNEG, SQRTF, FLOG, FEXP):

All of these subroutines require one floating-point value in the A-register. (The floating-point result is returned in the A-register):

```
CALL FPSUBR  call each subr by name with value in A-reg
```

Type 3 Subroutine (One-Word Fixed-to-Floating):

On entry the A-register has a signed integer value. On exit the A-register has the required floating-point value:

```
CALL FFLT
```

Type 4 Subroutine (One-Word Floating-to-Fixed):

On entry the A-register has a floating-point value. On exit the A-register has the one-word (signed) integer value:

```
CALL FFIX
```

Type 5 Subroutine (Two-Word Fixed-To-Floating):

On entry the A-register has a (signed) integer value and the E-register has the power of 10 multiplier (+ or - power). On exit the A-register has the required floating-point value:

CALL FFLTS

Type 6 Subroutine (Two-Word Floating-To-Fixed):

On entry the A-register has the floating-point value to be fixed. On exit the A-register contains a signed integer and the E-register contains the power of 10 multiplier:

CALL FFIXS

Timings:

FNEG: 3 cycles (constant)

FCAM: 24 cycles (constant)

FDIV: 103 cycles (maximum)..if either or both arguments negative
101 cycles for both arguments positive
96 cycles for overflow: both arguments negative
95 cycles for overflow: either argument negative
94 cycles for overflow: both arguments positive
89 cycles for underflow (0 result): both arguments negative
88 cycles for underflow (0 result): either argument negative
87 cycles for underflow (0 result): both arguments positive
31 cycles for 0 divisor (dividend negative): overflow bit set
30 cycles for 0 divisor (dividend positive): overflow bit set
3 cycles for 0 dividend

FMUL: 101 cycles (maximum)..if either or both arguments negative
99 cycles for both arguments positive
94 cycles for overflow: both arguments negative
93 cycles for overflow: either argument negative
92 cycles for overflow: both arguments positive
87 cycles for underflow (0 result): both arguments negative
86 cycles for underflow (0 result): either argument negative

```
          85 cycles for underflow (0 result): both arguments
             positive
          26 cycles for 0 multiplier (0 result): multiplicand
             negative
          25 cycles for 0 multiplier (0 result): multiplicand
             positive
           3 cycles for 0 multiplicand

FFLT:     36 cycles (maximum): values -1 through -7
          35 cycles: -10 thru -377 octal, (-8 thru -255 decimal)
          34 cycles: -400 thru -17777 octal
          34 cycles (maximum for positive values): 1 thru 7 octal
          33 cycles: -20000 thru -37777777 octal
          33 cycles: 10 through 377 octal
          32 cycles: 400 through 17777 octal
          31 cycles: 20000 through 37777777 octal
           3 cycles: value 0

FFIX:     41 cycles (maximum): 53100000 (-1000000) to 52200001
             (-17777700)
          40 cycles: 54400004 (-20000) to 53200001 (-777774)
          39 cycles: 55600200 (-400) to 54400005 (-17777)
          39 cycles: 24700000 (1000000) to 25577777 (17777700)
          38 cycles: 57500000 (-1) to 55600201 (-377)
          38 cycles: 23377774 (20000) to 24577777 (777774)
          37 cycles: 22177600 (400) to 23377773 (17777)
          36 cycles: 20300000 (1) to 22177577 (377)
          24 cycles: negative overflow: threshold: 52100000
          23 cycles: positive overflow: threshold: 25700000
          17 cycles: negative underflow: threshold: 57600001
          16 cycles: positive underflow: threshold: 20177777
           3 cycles: value 0

SQRTF:  160 cycles (maximum) .. exponent even
        156 cycles if exponent odd
         10 cycles for negative argument: overflow bit set
          3 cycles for 0 argument
```

Note 1: By 'signed' is meant that a negative value is the internal machine representation for negative values (two's complement of the corresponding positive value).

Note 2: Any error results in the overflow bit being set on exit. CAUTION: IT is the programmer's own responsibility to clear this bit before a subsequent subroutine call.

Note 3: Internal floating-point format:

Bit 23: Sign (of value, ie, sign of 'MANTISSA')
0: value is positive,
1: value is negative (TCA of positive value)

Bits 22--16: Biased characteristic (7 bits)
(100) 8 = BIAS, ie, represents 0 characteristic
(177)8 represents (63)10 characteristic
(000)8 represents (-64)10 characteristic

Bits 15--0: Fractional mantissa, ie, octal point is to the left of bit 15. Thus bit 15 = 1/2,
bit 14 = 1/4, etc.

Examples: Representation of (23.5)10:
(23.5)10 = (27.4)8 = (10111.1)2
= (0.101111)2 *2↑5
Thus to create the floating-point number:-
number is positive, therefore sign = 0
characteristic = (105)8 = (1000101)2
ie, bias + 5 --(100 + 5 from 2↑5 above)
mantissa = (1011110000000000)2
ie, the (0.101111)2 from above left-justified in 16 bits.
Putting the sign, characteristic and mantissa together: 0 1000101 1011110000000000
ie: (01000101101111000000000)2
ie: (2 1 3 3 6 0 0 0)8
Thus (21336000)8 is the required value.

Representation of (-23.5)10:
Take the two's complement of the floating-point value for (23.5)10, ie, 2's complement of (21336000)8
ie, (56442000)8. This is the required value.

Note 4: Range of Floating-Point Values:

The smallest positive floating-point value which FPP can handle is represented by (00100000)8. The value of this is (.1)2 *2↑(-64) = (.5)10 *2↑(-64) = 2↑(-65) = 2.711E-20 (approximately).

The largest positive floating-point value which FPP can handle is represented by (37777777)8. The value of this is (.777774)8 *2↑(63) = 2↑(62) + 2↑(61) + 2↑(60) + 2↑(59)..... + 2↑(47) = 2 ↑(63) - 2↑ (47) = 9.445E18 (approximately).

The negative range exactly parallels the positive range.