



# EUROTHERM PROCESS AUTOMATION

## PROGRAMMABLE INSTRUMENTS PROGRAMMING MANUAL

Prepared by: DP/RWP

Issue D

January 1986

### EUROTHERM PROCESS AUTOMATION LIMITED

SOUTHDOWNVIEW WAY, WORTHING, W SUSSEX BN14 8NN TELEPHONE: 0903 205277 FAX: 0903 233902 TELEX: 87437

DIRECTORS: J L LEONARD MSc PhD CHAIRMAN, F J PAGET MANAGING, M K ELLIOTT BCOM CA.

R W FREW CENG, P V WYLD BSc(Eng) CEng MIEE MIMgt

Registered number: 1180758 England. Registered office: Leonardslee, Lower Beeding, Horsham, West Sussex RH13 6PP

Turnbull Control Systems Ltd. reserves the right to make specification changes at any time without notice, in order to improve design and supply the best equipment possible.

Turnbull Control Systems Ltd. cannot assume any responsibility for any circuits or system schematics shown. All applications information contained herein is intended solely for general guidance and use of information for users' specific applications is entirely at the users own risk.

PROGRAMMABLE INSTRUMENTS 6433, 6356/66 and 6445

PROGRAMMING MANUAL

CONTENTS

\*\*\*\*\*

1. GENERAL INTRODUCTION

- 1.1 The Manual
- 1.2 The Language
- 1.3 The Instruments

2. SOFTWARE STRUCTURE

- 2.1 6433 Functional Overview
- 2.2 6356/66 Functional Overview
- 2.3 6445 Microsupervisor
- 2.4 Programming Considerations
- 2.5 Program Memory
- 2.6 Selecting a Program

3. GETTING STARTED

- 3.1 Connecting a Terminal
- 3.2 Typing Conventions
- 3.3 Understanding the Stack
- 3.4 Keep Note of the Stack

4. PROGRAMMING THE INSTRUMENT

- 4.1 Introduction
- 4.2 First Steps
  - 4.2.1 Logging-on
  - 4.2.2 Logging-off
  - 4.2.3 Escape Key
  - 4.2.4 Single Line Editor
  - 4.2.5 Have a Go
  - 4.2.6 Summary
- 4.3 Interaction and the Interpreter
  - 4.3.1 Arithmetic Calculations
  - 4.3.2 Try Some Arithmetic
  - 4.3.3 Summary

- 4.4 Terminal Input Output Operations
  - 4.4.1 Numbers and Print Routines
  - 4.4.2 Character Output
  - 4.4.3 Summary
- 4.5 Writing a Program
  - 4.5.1 Creating a Word
  - 4.5.2 Editing a Word
  - 4.5.3 Dictionary Management Words
  - 4.5.4 Summary
- 4.6 Stack Manipulations
  - 4.6.1 Local Variables
  - 4.6.2 Summary
- 4.7 More Arithmetic and Trigonometric Words
- 4.8 Logical Words
  - 4.8.1 Examples
  - 4.8.2 Summary
- 4.9 Comparison Words
- 4.10 Variable and Timer Words
  - 4.10.1 Variables
  - 4.10.2 Timers
  - 4.10.3 Summary
- 4.11 Terminal Input
- 4.12 Conditional Branches
  - 4.12.1 IF ... ELSE ... ENDIF Structures
  - 4.12.2 CASE ... ENDCASE Structures
  - 4.12.3 Summary
- 4.13 Loop Structures
  - 4.13.1 Definite Loops
  - 4.13.2 Indefinite Loops
  - 4.13.3 Summary
- 4.14 Instrument Specific Words
  - 4.14.1 Special 6433 Words
  - 4.14.2 Summary of Special 6433 Words



- 4.14.3 Special Advanced Controller Words
- 4.14.4 Summary of Special Advanced Controller Words
- 4.14.5 Special Microsupervisor Words
- 4.14.6 Summary of Special Microsupervisor Words

#### 4.15 Debugging Facilities

#### 4.16 Reserved Words

- 4.16.1 The 6433 MAIN Program
- 4.16.2 Advanced Controller Programs
- 4.16.3 6445 Microsupervisor
- 4.16.4 The ERROR Program

#### 4.17 Error Messages

### 5. PROGRAMMING TERMINAL UTILITIES

#### Appendices

- |            |                                                   |
|------------|---------------------------------------------------|
| Appendix 1 | Interconnection Cables                            |
| Appendix 2 | Binary Protocol Parameter Numbers for the 6433    |
| Appendix 3 | The Functional Blocks for the Advanced Controller |



## 1. GENERAL INTRODUCTION

### 1.1 The Manual

This manual introduces the programming techniques associated with the TCS System 6000 programmable instruments:

6433	Programmable Signal Processor
6356/66	Programmable Advanced Controller

The text assumes that the reader has a working knowledge of the System 6000 intelligent instruments, specifically

6432	Signal Processor
6350/60	Process Controller

Before attempting to create programs for the programmable instruments, it is recommended that the reader refers to the 6432, 6350/60 Technical Manuals.

The Programming Manual is written in two sections:-

- Firstly - A tutorial to allow a beginner to start to perform simple routines.
- Secondly - A reference section to provide the more experienced programmer with a check list of the instrument functions.

### 1.2 The Language

The System 6000 programmable instruments are programmed in a version of the "FORTH" \* language developed by TCS specifically for use with these instruments.

TCS FORTH differs from other standard FORTH implementation in that:-

- (a) It is considerably simpler and does not require features such as disk file I/O.
- (b) It handles numeric data exclusively in floating point format, thus removing the need for the programmer to consider binary point positioning.

The manual describes the TCS FORTH implementation and provides examples relating to the use of the language. Appendices show examples of a documented applications program for a typical process control problem.

\* FORTH is the registered trademark of Forth Inc.

Readers who already have programming experience (not necessarily in FORTH) should find the manual sufficient. Readers who have no previous software knowledge will probably find it beneficial to refer to a standard introductory text on FORTH. This will provide an introduction to the programming concepts of the language making the implementation described in this manual more meaningful.

### 1.3 The Instruments

The programmable instruments are based on the hardware of the existing configurable instruments as follows:-

6433 based on 6432 Signal Processor hardware

6356/66 based on 6350/60 Single Loop Process Controller

6445 Microsupervisor - an extension of the 6433 Signal Processor

The programmable instruments include all of the features of the configurable instruments plus the TCS FORTH package that allows arithmetic and boolean computation and display assignment.

The programmable units extend the monitoring and control facilities of the configurable instruments by providing features necessary for batch and sequence control and interlocks. As well as "analogue" calculation and logical manipulation, timing and counting functions suitable for sequencing applications are included. Timing functions are carried out to a precision of 2 milliseconds under control of a real-time clock.

The input/output capability of the instruments is as follows:-

6433 - 4 blocks of 8 channels where the blocks may be analogue or digital inputs or outputs.

6356/ 3 analogue inputs, 3 analogue outputs,  
6366 - 8 digital inputs, 8 digital outputs.

6445 - 5 serial lines, 8 digital inputs,  
8 digital outputs.

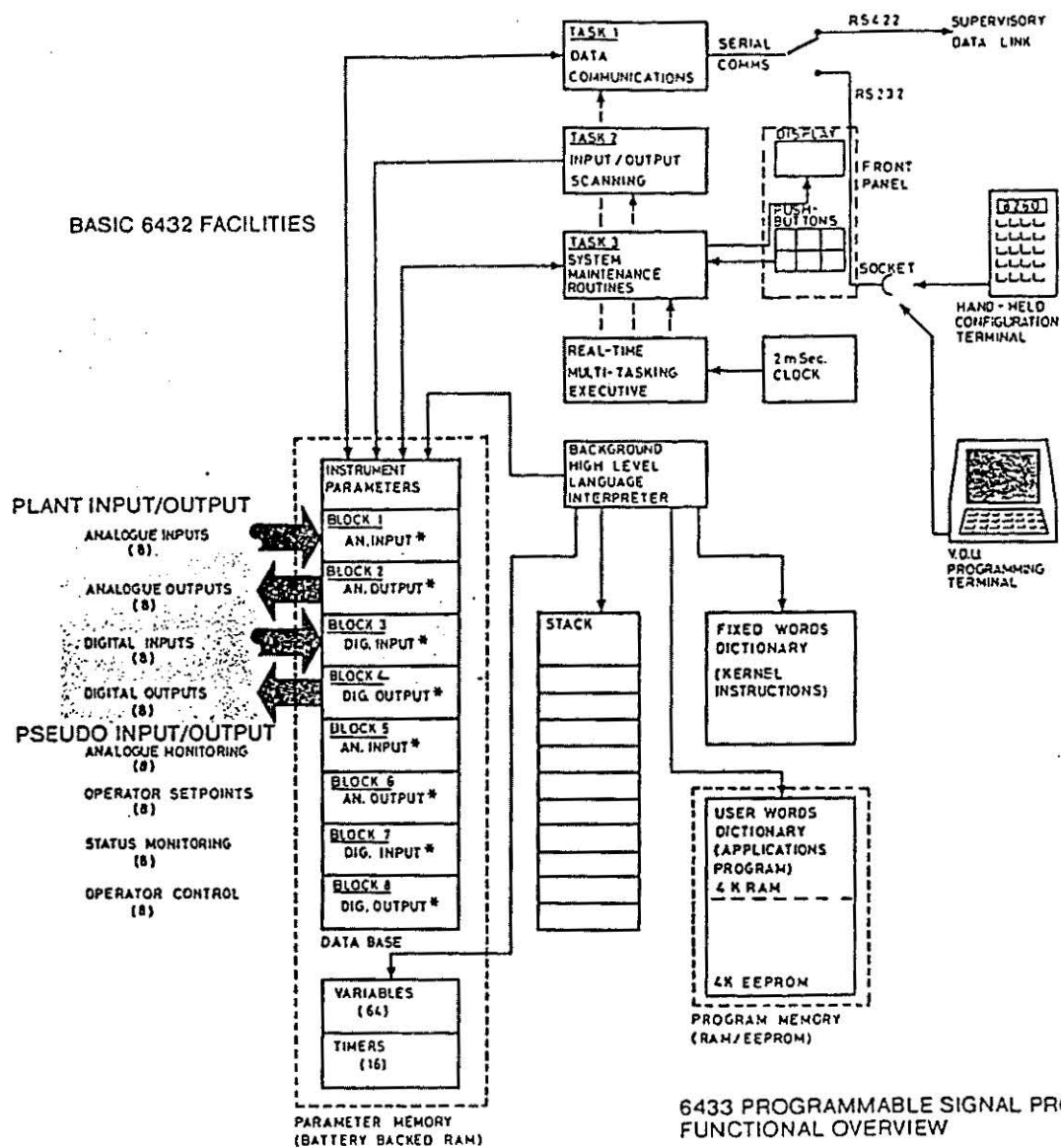
The 6433 provides an additional 4 blocks of 8 pseudo channels which may be used for internal derived variables. These are accessible from the front panel and via the hand-held programming terminal or via the serial link in the same way as the real input/output channels, and would typically be used to display the results of calculations, status and to set internal constants.

The 6356/66 include 8 constants plus two 8 bit status words that may be used to store intermediate values as above. The constants may be assigned to the front panel display and are accessible via the hand-held programming terminal or the serial link.

Programs are developed in RAM using an RS232 "Teletype" compatible device such as a VDU plugged into the front panel socket.

A program developed in RAM can be "fixed" into ROM from the programming device. The program is, however, always run in RAM and this allows programs to be loaded in RAM, edited and then debugged before the original program is replaced.

Full program edit and load/save facilities may be obtained using the BBC 'B' or Epson PX8 computers.



\*NOTE: ANALOGUE AND DIGITAL INPUT/OUTPUT CARDS ARE SHOWN IN A TYPICAL CONFIGURATION: ANY COMBINATION MAY BE SPECIFIED.

Fig 2.1

## 2. SOFTWARE STRUCTURE

### 2.1 6433 Functional Overview

The heart of the 6433 is the data base as shown in the functional overview of Fig. 2.1. This holds the instrument parameters and channel parameters for the 4 blocks of 8 real input/output channels as for the 6432. In addition it holds the parameters for the 4 blocks of 8 internal or pseudo channels.

This data base is scanned continuously and up-dated with the values of real inputs or computer values; output blocks are up-dated with new values if a change has taken place to the relevant values in the data base. All values are available for display on the front panel under the control of the pushbuttons.

As in the 6432 instrument, values may also be accessed via the data communications task. In normal on-line operation this services the RS422 serial link. However, plugging in a programming terminal disables the RS422 link and transfers control to the RS232 front panel socket which may be used for either the hand-held terminal or a teletype compatible console such as a VDU.

Also held in the data base are current values of the 64 variables and 16 timers. Variables are stored in 32 bit floating point format. Timer values are stored as 32 bit numbers with 1 bit corresponding to 2 milliseconds, giving a range of + 4,294,960 secs. or about 7 weeks.

User programs are written in a high level interpretive language based on a version of FORTH. Program statements are called "Words" and these loosely correspond to sub-routines. A kernel set of the most common functions are resident in ROM in the "Fixed Word Dictionary".

These comprise the common arithmetic and boolean operators, input/output routines, etc. The user then builds up a program by creating a hierarchy of user words in which both user-created and fixed dictionary words may be nested and strung together. The user words are added to the "User Word Dictionary".

Program words operate on one or more values on a data "stack". New values are put on to the top of the stack and push down old values. They are picked off on a first-in last-out basis so that program statements operate in a "Reverse Polish" format.

The front panel RS232 socket supports two terminal modes. The first is Command Mode which provides the normal access to configuration parameters via 2 character mnemonics. The second is Programming Mode which provides access to the FORTH editor for entering and modifying programs. This mode is protected by a security number associated with a user name.

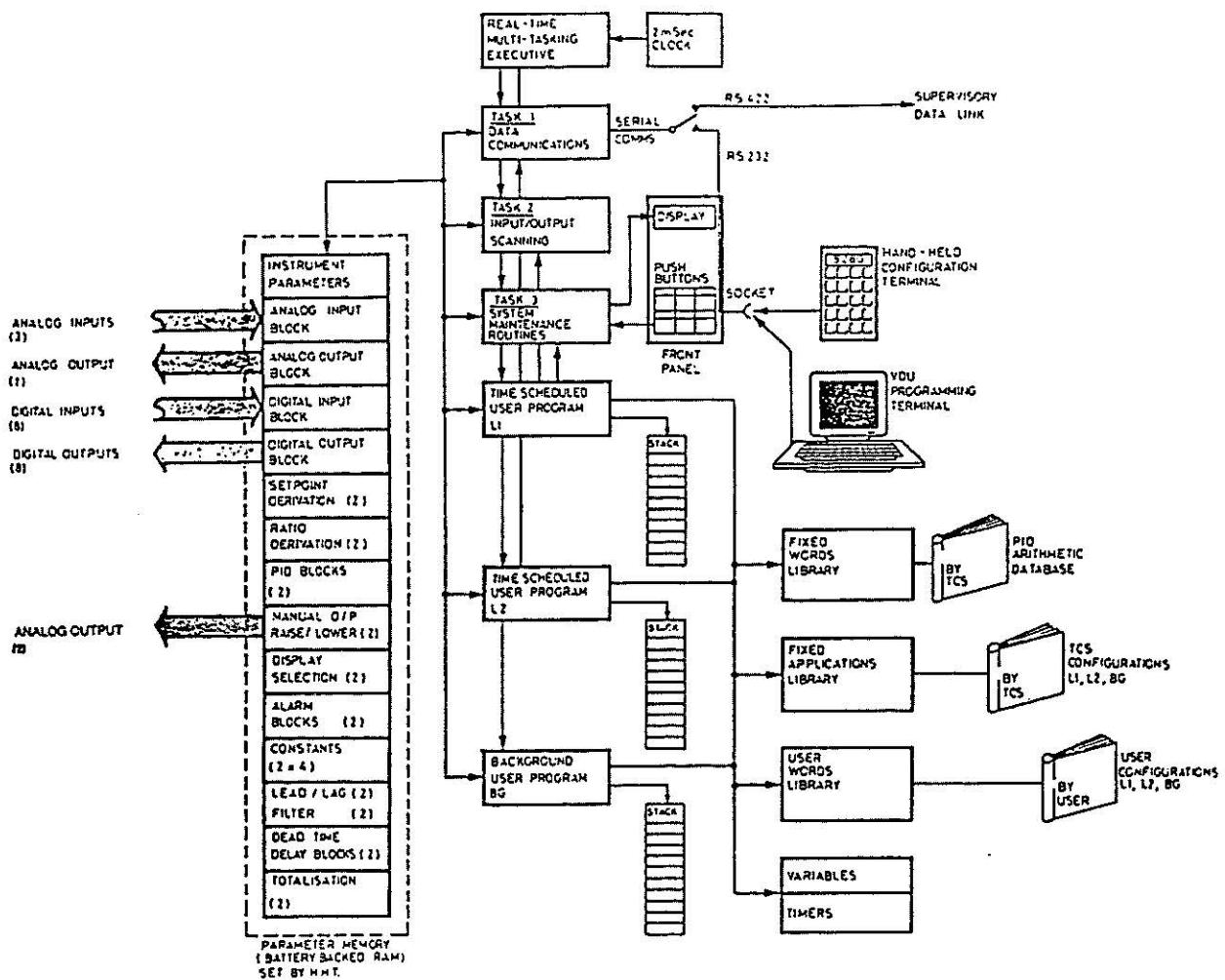
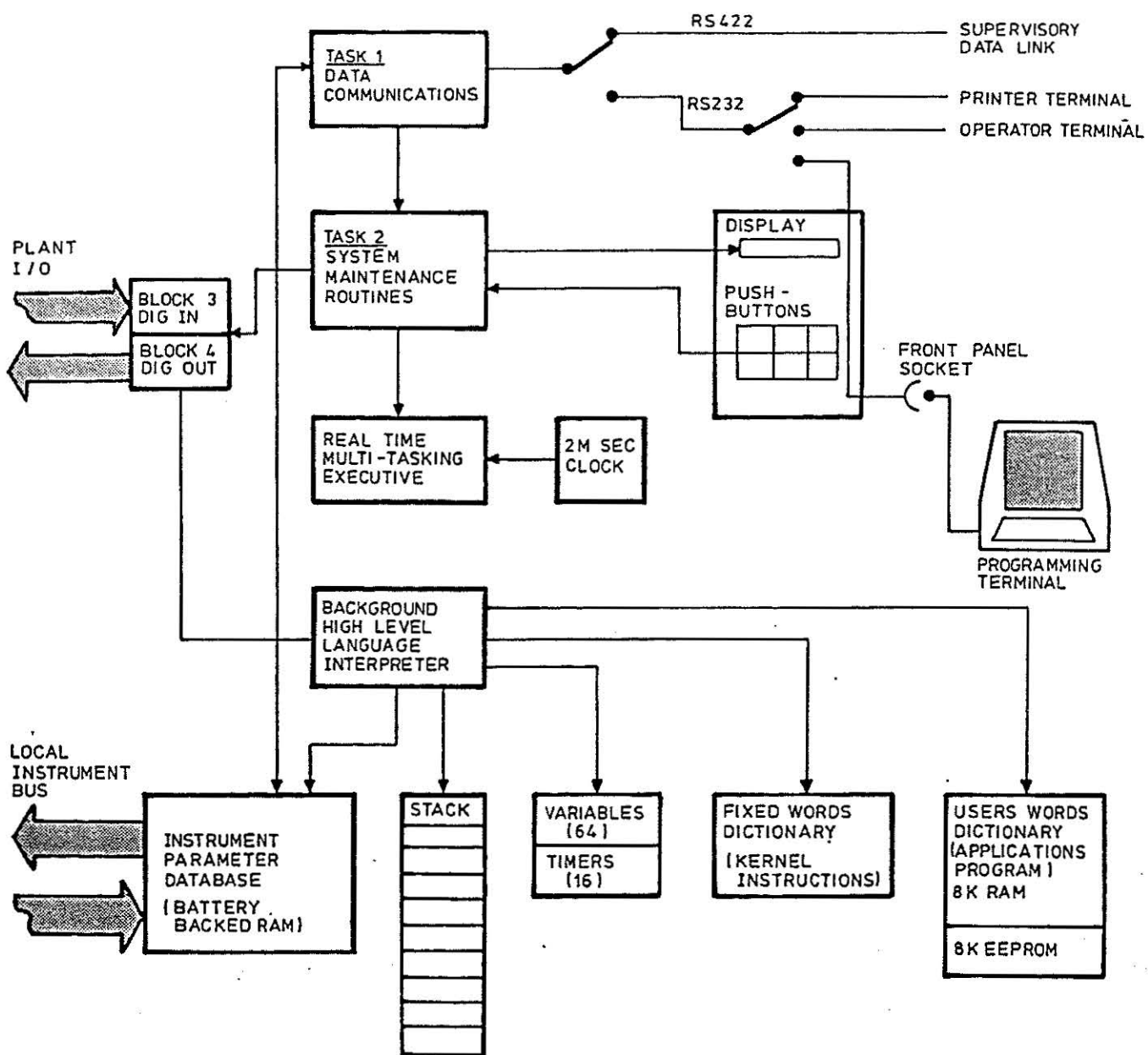


Fig. 2.2





6445 MICROSUPERVISOR  
FUNCTIONAL OVERVIEW

Fig 2.3

## 2.2 6356/66 Functional Overview

The structure of the 6356/66 Controllers is illustrated in Fig. 2.2. It differs from the 6433 in that it includes 3 time scheduled programs instead of 1. This allows separate programs to be run for each of the available PID loops plus a background program area that sets up I/O interlocks, etc.

The instrument communications and tasks are the same as the 6433.

The "Fixed Word Dictionary" has been extended to include a number of additional words such as PID, MSCONT, RATIO, REMOTE, ALARM, etc.

Two further Libraries containing the TCS fixed applications plus an area to store customer applications programs are also included.

The program structure and the use of the "stack" are identical to the 6433 above.

## 2.3 6445 Microsupervisor

The structure of the 6445 Microsupervisor is illustrated in Fig. 2.3.

The 6445 is basically an extension of the 6433 Programmable Signal Processor that allows four important advantages.

### i) Input / Output Expansion

The 6433 I/O is limited to the 32 channels of the four I/O cards.

The 6445 I/O depends on the I/O of the instruments connected to the instrument bus, which may be in excess of 60 analogue channels or 150 digital channels or combinations.\*

\* N.B. These figures are based on worse case conditions see calculation in section 3.

### ii) Local Task Control / Softwired Configuration

The 6445 provides a low cost and convenient means of self-wiring interaction between small numbers of instruments. The local "task" control has advantages over the supervisory computer tasking package where speed, control security and cost are important.

### iii) Increased Program Memory

The 6445 has 8K RAM and 8K EEPROM, twice the memory size of the 6433.

#### iv) Local Operator Displays and Logging

Features (i), (ii) and (iii) are direct extensions of the 6433 Programmable Signal Processor and consequently we now have considerable experience in assessing and costing engineering involvement.

Feature (iv) is - in the first release of the 6445, relatively undeveloped and consequently care must be taken when making quotations.

The two additional RS232 ports allow:-

- (a) an additional programming terminal or operator VDU/keyboard to be connected:

- (b) a printer/data logger to be connected.

These ports together with the additional string statements allow data to be buffered and presented to the display and/or printer by the FORTH Program.

### 2.4 Programming Considerations

A stack-oriented language imposes a structured approach to applications programming. Program statements or "Words" can only call other routines that have already been defined and are recognised by the interpreter. This implies "top down" formulation of the problem with "bottom up" implementation of the program.

The first step is to define the data base. This will comprise real inputs and outputs according to the hardware configuration, and then internal constants or "pseudo" inputs and outputs for derived values and constants requiring access from the front panel. It should be noted that the 6433 derived values for display should be set up as pseudo inputs because this allows the facility of setting high and low alarms as for real inputs. Conversely operator-set constants should be set as outputs so as to allow use of the raise/lower buttons or hand-held terminal for changing values.

Intermediate results used in more than one place in the program but not required for display are conveniently set as variables. This has the advantage of reducing dependence on the stack and can allow individual program statements to be more self-contained.

The next step is to draw a flow chart for each independent task. All tasks residing in the instrument must then be incorporated in a master loop which performs scheduling; usually it is sufficient to cycle sequentially through all the tasks and enclose them in a BEGIN .... REPEAT structure. Two preliminary tasks are also generally required to set up initial values of constants and to reset timers, counters and flags, etc.

The third step is to divide the flow chart into sub-routines of a suitable size to be defined as program words. It may be desirable for clarity to restrict word length to one line on the terminal device, to a length set by the WINDOW operator. Longer statements can be achieved by stringing together several words. Words should as far as possible be self-contained.

The final step is to translate the flow chart into code and enter the program, noting that words lowest in the hierarchy must be entered first.

For simple analogue computations a block diagram approach is often more appropriate than a flow chart. An example is shown in the Appendix.

Individual parts of the program may be tested at any stage by executing in immediate mode, simply by typing in the appropriate Word name.

## 2.5 Program Memory

The TCS range of programmable instruments uses two areas of memory for programs.

The first area comprises Random Access Memory (RAM) that is used for developing and running user programs.

The second area is Electrically Erasable Programmable Read Only Memory (EEPROM) which is used for archiving programs. This memory provides a secure method of storing programs and does not require a battery when power is removed from the instrument. In addition, switches or jumpers are provided on the boards to prevent accidental erasure of the EEPROM.

When the instrument is powered-up, the program in EEPROM is loaded into the RAM area and this program is then executed. The user programs in memory are checksummed for security and this is monitored whilst the program is running.

The EEPROM is also used to store a security name, security code and a window size that are required whilst programming.

## 2.6 Selecting a Program

When an instrument is powered up, it must decide which program to run. The selection of programs varies slightly between instruments.

### a) Program Selection for the 6433

The power-up procedure initiates a search through the dictionaries for a program called MAIN. If this program is found, it will then be run, otherwise the instrument is considered to be HALTED and this diagnostic should appear on the front panel.

b) Program Selection for the Advanced Controller

The Advanced Controller allows up to three programs to be run. Two of these programs are scheduled at times selected within the program and the third program runs in the background.

To allow flexibility, the user can select which programs are run using a standard hand-held terminal. This is achieved by setting the required program names into the parameters L1, L2 and BG in the general purpose data block. The programs specified in L1 and L2 are the time scheduled programs, where L1 has the highest priority. The program specified in BG runs in the background.

At power-up, the dictionaries are searched for these programs and if they are found, they are then installed and run.

Changing the names of the required programs in L1, L2 and BG has no effect on the programs that are running since these parameters are only inspected at power-up or when invoked from a programming terminal.

c) Program Selection for the Microsupervisor 6445

The power-up procedure initiates a search through the dictionary for a program called BGRND. If this is found, it will then be run, otherwise the instrument is considered to be HALTED and this diagnostic should appear on the front panel status display



## Section 3

### Getting Started

This section describes the equipment needed to program the TCS instruments, and outlines some of the documentation conventions used.

#### 3.1 Connecting a Terminal

Initially a programmer requires a programmable instrument and a programming terminal. The simplest programming terminal is a standard VDU with an RS232 interface. The examples included in this section assume the user simply has a standard VDU.

In addition TCS have written applications software to allow a BBC model B computer or an EPSON PX-8 portable computer to be used as programming terminals, these computers provide additional facilities to archive user programs and instrument data bases. (Ref ????)

The first step is to connect the VDU to the Hand Held Terminal (HHT) socket on the front of the instrument, this should be done using the lead described in Appendix ?.

The second step is to select the instrument baud rate to match that of the VDU. The baud rate is selected using Switch Bank 1. With switch 1 off the baud rate is set to 300 baud, however when switch 1 is on the programmer can select the baud rate using switches 2,3 and 4 in the normal way (see Table 3.1). Normally whilst programming the baud rate would be set to 9600 by setting switches 1,2 3 and 4 on.

Finally, check that the terminal data format is set correctly. This should be:

- 1 start bit
- 7 data bits
- 1 parity bit (even)
- 1 stop bit (2 stop bits at 110 baud)

#### 3.2 Typing Conventions

In common with many other manuals the following conventions are used in this section.

1. Special function keys such as Return and Delete are enclosed in angle brackets. This indicates that you should only type a single key, eg if you are asked to type the line

1 2 <Return>

then type the two numbers separated by a space and then the Return key.

2. Some characters on VDU terminals require the use of more than one key, eg to type capital letters you would hold down the shift key whilst typing the required letter. When an operation requires more than one key it will be written thus:

eg SHIFT-X    Hold SHIFT key whilst typing X  
CTRL-P       Hold CTRL key whilst typing P

3. To avoid confusion between text typed by a programmer and messages from the instrument, all messages from the instrument will be underlined.

### 3.3 Understanding the Stack

To understand and use the Forth programming language, a programmer must learn the concept of a "data stack", (usually abbreviated to "stack").

In simple terms a stack is an area of computer memory. Data can be added or "pushed" on to the top of the stack, when this happens, existing data is moved down one location. A further feature of the stack is that data can only be removed from the top of the stack.

For this reason the stack is often described as "last-in, first-out" (LIFO). As data is pushed on top of the stack, existing data is pushed further down. As data is removed from the top of the stack, the last data in is the first data out.

The Forth programming language uses Reverse Polish Notation (RPN) or post-fix arithmetic, rather than the standard algebraic format.

This has several advantages:

- a. The rules are very simple.
- b. There is no need for parenthesis.
- c. This is the way computers do arithmetic, and it makes the programs more efficient.

Reverse Polish Notation is simply a way of presenting data and operations to a computer in the correct order for computation.

Whenever a number is introduced in a calculation, it is automatically pushed onto the stack. Other operations or programs can modify, add or remove data on the stack. In the examples below, the contents of the stack are shown as a list of data, and the top of the stack item is marked with an arrow.



### Example 1

Consider the sum  $(4+5)$  , this would be written in Reverse Polish as:

4 5 +

The computation steps would be:

Step No	Description	Stack
1	Stack the number 4	Top-> <u>4</u> : : :
2	Stack the number 5	Top-> <u>5</u> : : 4 : : :
3	+ (Remove numbers and replace with the sum)	Top-> <u>9</u> : : :

### Example 2

Consider the sum  $4*(5+6)$ , in standard algebraic format this requires parenthesis to ensure that the addition is computed before the multiplication. This would be written in Reverse Polish as:

4 5 6 + \*

The computation steps would be:

Step No	Description	Stack
1	Stack the number 4	Top-> <u>4</u> : : :
2	Stack the number 5	Top-> <u>5</u> : : 4 : : :
3	Stack the number 6	Top-> <u>6</u> : : 5 : : 4 : : :
4	+ (Remove top two numbers and replace with the sum)	Top-> <u>11</u> : : 4 : : :
5	* (Remove numbers and replace with the product)	Top-> <u>44</u> : : :

### 3.4 Keep Note of the Stack

It is important to know what is on the stack, and to maintain a discipline of tidying up the stack when a program has finished.

If you have nothing on the stack, and you try and remove something, then you will get a "Data stack underflow" error.

There is also a limit to how much data can go on the stack. If you do exceed this limit you will get a "Data stack overflow" error, (although it is difficult to do this unless you make some mistakes).

The following notation is often used with the description of a program to describe the stack status before and after a word is executed.

(before -- after)

The dashes separate the data that should be on the stack before the operation from the data after the operation.

When there is more than one item on the stack, the item on the right of the list is the top of stack item.

Eg the notation for the operation + in the previous examples is:

+ (n1 n2 -- sum)

where n2 is above n1 on the stack before the operation.

## Section 4

### Programming the Instrument

#### 4.1 Introduction

This section provides an introduction to TCS programmable instruments.

The information provided here is intended to guide an individual with some experience of computing through the first steps in working with the instruments. As such the approach is intended to be relatively simple, however, most sections are followed by a summary so that they can also be used as a reference manual at a later date.

#### 4.2 First Steps

The first step is to find somewhere quiet, connect your terminal to the instrument and gain some confidence with the programming techniques.

#### 4.2.1 Logging-on

When the VDU is plugged into the front of the instrument the normal HHT prompt ?? CMD should be printed on the screen. If it does not, try typing the letter Z. If the ?? CMD prompt still does not appear then:

N.B. There is no prompt for the 6445 Microsupervisor, simply type CTRL-P.

1. Check the leads and baud rate on the instrument and the VDU.
2. Unplug the lead from the front of the instrument and then plug it in again.

When the correct display is shown on your VDU type

CTRL-P

The terminal will now display one of the following two messages:

#### 1. \* Please enter your security number \*

The programmer should then enter a 3 digit security number. Entering the correct code will result in the response:

\* Hit space bar to change name \*

If you want to change the name and security code type a space and then proceed as outlined in part 2 below. Otherwise in response to any other character the display will show:

\* TCS Forth Version 1.(1) \*

\* Have a nice day \*

\*Free nnn\*

Where nnn represent the amount of memory available for programs.

#### 2. \* Please Log-on \* USER

You should now enter a 7 character user name (including spaces). When you have done this the instrument will print:

\* Your security number is yyy Remember it \*

\* Hit space bar to change name \*

The number yyy is your security code, make a note of this number as you will need it next time you log-on to the instrument.

If you wish to change or correct the user name type a space and have another go, otherwise in response to any other character the display will show:

\* TCS Forth Version 1.(1) \*

\* Have a nice day \*

\*Free nnn\*

Where nnn represent the amount of memory available for programs.

#### 4.2.2 Logging-off

The procedure to return to the normal HHT operating mode depends on the issue of software in use.

Type CTRL-Q to log-off when using TCS Forth Version 1

Type CTRL-X to log-off when using TCS Forth Version 2 & 3

The display should now print the standard HHT response:

?? CMD

#### 4.2.3 Escape Key

Hitting the ESCAPE key whilst programming the instrument will abort the current operation and print the standard message:

\*Free nnn\*

#### 4.2.4 Single Line Editor

The programs are normally entered by typing a line of text at the terminal. The editor is always in 'insert' mode, this means that any characters typed are inserted into the text before the character pointed to by the cursor.

Naturally everybody makes mistakes, so to alter the text the user can move the cursor forwards or backwards along the line of text using the TAB or BS (BACK SPACE) characters respectively. To delete a character move the cursor over the character, and hit the DELETE key.

If you type an illegal character this will ring the 'bell' on your VDU and the character is ignored.

To terminate a line, or force a statement to be executed, hit the RETURN key.

Finally only a limited number of characters can be displayed on one line of a VDU screen, however it is possible that your program could contain more characters than this. To allow you to edit the line you will find that the editor shuffles the program across the screen when you try and move the cursor beyond the edges of the screen.

You can alter the length of the displayed string with the WINDOW routine. You should normally set the window to be one less than the number of characters displayed on a line, this avoids confusion when the cursor attempts to move off the edge of the screen.

Eg a standard VDU will display 80 characters on a line, in this case you should select a window of 79 characters by entering:

79 WINDOW <Return>

#### 4.2.5 Have a Go

At this stage it may be useful to try typing some text at the terminal, this should not cause any problems as you are not actually connected to any real plant!

Try the effect of reducing the window size (you cannot make it smaller than 6).

Try modifying text on the screen using the TAB, BS and DELETE keys.

If anything goes wrong, hit the ESCAPE key and the terminal should display:

\* Free nnn\*

#### 4.2.6 Summary

Table 4.2.6 describes the single character commands introduced in this section.

---

CTRL-P	Initiate log on sequence (Halt all user programs)
CTRL-Q	Log-off for Version 1
CTRL-X	Log-off for Version 2 & 3
ESCAPE	Halt background program and print standard message
BS	Back space single line editor
TAB	Forward space single line editor
DELETE	Delete character in single line editor

---

Table 4.2.6 Single Character Commands

### 4.3 Interaction and the Interpreter

One of the major advantages of the FORTH language is that it is both interactive and interpretive. This makes it easy to learn the language since you can pick it up in easy stages.

An interpretive language also means that programs can be modified very easily, simply by editing them. You don't have to worry about compiling and linking programs as this is done automatically.

A further advantage of FORTH with respect to many other interpretive languages, such as BASIC, is that the interpreter actually compiles the program as you enter it, and NOT at run time. This means that programs are very efficient, both in size and speed.

When you enter a program, the text is scanned for syntax errors (typing mistakes) and valid references (that any programs you call already exist). If there are any errors, a message will be printed on your terminal. A list of the error messages and a more complete explanation of their meaning is given in \*\*\*\*\*.\*\*.

This section will introduce you to the language by performing some calculations directly at the VDU. In fact you will be using the FORTH language as a rather sophisticated calculator.

#### 4.3.1 Arithmetic Calculations

Arithmetic is performed in Reverse Polish Notation (RPN) or postfix arithmetic, rather than the algebraic notation used in written arithmetic.

The rules of RPN are very simple, and will be familiar to those who have used a Hewlett Packard calculator. The first process is to put data onto the stack, then when the program reaches an operator (eg Add) this will modify the data on the stack.

The most commonly used operators for arithmetic are:

+ Add  
- Subtract  
\* Multiply  
/ Divide

Consider an example to add the numbers 4 and 5

this would normally be written (4+5)

however in RPN it would be written 4 5 +

Taken step by step the following table shows what the computer would do

: Step No : Operation : Stack position :				
:	:	:	2nd	Top
:	:	:	:	:
: 1	: 4	: -	:	4
: 2	: 5	: 4	:	5
: 3	: +	: -	:	9
:	:	:	:	:

A more complicated example that requires the use of brackets is the expression:- 4\*(5+6)

This would be written in RPN as:- 4 5 6 + \*

The following table shows what happens step by step

: Step No : Operation : Stack position :					
:	:	:	3rd	2nd	Top
:	:	:	:	:	:
: 1	: 4	: -	:	-	4
: 2	: 5	: -	:	4	5
: 3	: 6	: 4	:	5	6
: 4	: +	: -	:	4	11
: 5	: *	: -	:	-	44
:	:	:	:	:	:



#### 4.3.2 Try Some Arithmetic

You are now ready to try some examples at the terminal, naturally you will want to see the results of your calculations, this can be accomplished by printing the top value on the stack using the . (Dot) function.

Type the following example at your keyboard:

```
<Escape>
*Free nnn*
4 5 + . <Return>
9
*Free nnn*
```

Try a few sums of your own and confirm that you are getting the correct answers.

#### 4.3.3 Summary

Table 4.3 describes the arithmetic functions introduced in this section.

The notation follows the form described in section 3.4, and the description column provides a concise explanation of the operation.

Operator	Stack notation	Description
+	(n1 n2 -- sum)	Adds (n1+n2)
-	(n1 n2 -- diff)	Subtracts (n1-n2)
*	(n1 n2 -- prod)	Multiplies (n1*n2)
/	(n1 n2 -- quot)	Divides (n1/n2)

Table 4.3 Arithmetic Operators

#### 4.4 Terminal Input Output Operations

This section describes the operators used for printing results or messages on the terminal. These operators would normally be used during program development to check results, and monitor the progress of a program.

##### 4.4.1 Numbers and Print Routines

All data used in programs is stored in the same format as floating point numbers, however in some cases it is useful to present data in either Hexadecimal or ASCII formats.

Data can be entered in the following three formats:

- a) Standard floating point numbers, eg 56 1000 1.2E-12 -5.678 .
- b) Hexadecimal format can be used to represent a 16 bit number. This number is preceded by the # character, eg #001C #CDEF .
- c) ASCII format can be used to provide a two character number. This number is preceded by the \$ character, eg \$AB \$XY .

Note that in TCS Forth Version 1 data can only be presented in standard decimal format.

Numerical data on the top of the stack can be printed in any of these three formats.

##### Operator Description

- . This operator (introduced in section 4.3.2) takes the number off the top of the stack and prints it on the terminal.
- # This operator takes the number off the top of the stack and prints it in Hexadecimal format. The number is preceded by a # for identification purposes.
- \$ This operator takes the number off the top of the stack and prints it as two ASCII characters. The characters are preceded by a \$ for identification purposes.

The examples below show how a single number can be printed in the differing formats:

```
<Escape>
*Free nnn*
16706 . <Return>
16706
*Free nnn*
16706 # <Return>
#4141
*Free nnn*
16706 $ <Return>
$AB
*Free nnn*
```

#### 4.4.2.Character Output

It may be useful to print messages on the screen whilst programming, or to provide some type of format when printing results. This can be achieved with the following operators:

##### Operator Description

." This word defines the start of a print string or message to be printed on a terminal. The string must be terminated by a double quote character. Try the example:

```
." THIS IS A STRING" <Return>  
THIS IS A STRING  
*Free nnn*
```

This word is only available in TCS Forth Version 3 and cannot be used in "immediate mode". The word defines the start of a text string which is terminated by another " character. When the word is executed, it stacks the address of the string as a floating point number. The printing words . or \$ will now print this number as a text string rather than as a standard floating point number.

SPACE This transmits the standard ASCII space character to the terminal.

CR This transmits the standard ASCII carriage return character to the terminal.

LF This transmits the standard ASCII line feed character to the terminal.

BS This transmits the standard ASCII backspace character to the terminal.

NL This forces a new line on a terminal, it is a shorthand version of entering CR followed by LF.

EMIT In some cases it is useful to be able to transmit special control characters to the terminal. The EMIT operator takes a value in the range 0 to 127 off the top of the stack and transmits it to the terminal. Eg to send one character to the terminal to ring the bell type:

```
7 EMIT <Return>
```

If the value is in the range 128 to 255 it is transmitted as 3 ASCII characters:

```
ESC C1 C2
```

where ESC is the normal Escape character (Decimal 27), and C1 and C2 are the characters of low and high significance in the Hexadecimal representation of the number.

Eg 128.10 = 80.16 is emitted as ESC O 8  
254.10 = FE.16 is emitted as ESC E F

Try to use a few of these operators and test their effect. You should now be getting the hang of using the interpreter in what is called "Immediate Mode". This term means that your operations are executed immediately, unlike in a program where you define a set of operations that can be invoked at a later time.

The Immediate mode is a useful facility since it allows you to test programs at your terminal.

#### 4.4.3 Summary

Table 4.4 provides a concise list of the words introduced in this section.

---

Operator	Stack notation	Description
.	(n1 -- )	Print the value n1 as a decimal number.
	(n1 -- )	Print the value n1 in hexadecimal format. (Not available in TCS FORTH Version 1)
\$	(n1 -- )	Print the value n1 as 2 ASCII characters. (Not available in TCS FORTH Version 1)
." abcd"	( -- )	Prints the string abcd on the terminal. The " character terminates the string.
SPACE	( -- )	Transmit a space to the terminal.
CR	( -- )	Transmit a carriage return to the terminal.
LF	( -- )	Transmit a line feed to the terminal.
BS	( -- )	Transmit a backspace to the terminal.
NL	( -- )	Transmit carriage return and line feed to the terminal.
EMIT	(n1 -- )	Transmit n1 as an ASCII character
WINDOW	(n1 -- )	Set the terminal line length (see section 4.2.4)
" abcd"	(-- add)	Stacks address of the string abcd. The " character terminates the string. Not available in TCS Forth Version 1 and 2.

---

Table 4.4 Number Printing and Terminal Control

#### 4.5 Writing a Program

The Forth programming language is unusual with respect to many high level languages. Its power comes from the ability to build up powerful programs, whilst allowing a programmer some access to the primitive functions of the computer when required.

A Forth program comprises a series of "words", these correspond to the functions or subroutines of other programming languages.

The words are stored in a dictionary. This dictionary already contains a set of words that are fixed in the instrument. When you write a word this is added to the end of the dictionary. The order of the words in the dictionary is defined by the order in which the words are entered.

There is no difference between using the words already in the dictionary and the words you add to the dictionary, although for convenience the words are normally listed separately. The only restriction is that words can only call words that already exist below them in the dictionary.

This means that a programmer would define a problem from the top down, but must write the program from the bottom up. For example the main program may be intended for boiler supervision - the top level. This would be broken down into elements such as adjusting setpoints and checking for safe operating conditions. This process continues downwards to the basic functions of opening valves or triggering alarm signals.

The programming phase now starts from the bottom level writing the basic functions which will be used by other routines. In this way the low level words can be written and tested before inclusion in the main program.

#### 4.5.1 Creating a Word

A word is created by the user at the terminal. The general format of a word is:

: Name Program-code ;

This can be broken down into four parts.

- a. The colon character : - This is a special word that defines the start of a word definition.
- b. The Name - This provides a unique identifier for each word in the dictionary. The name can contain any standard printing ASCII characters. The compiler checks and stores the first 7 characters in the name. If a name contains more than 7 characters a check is made of the length of the string and the values of the first 7 characters.
- c. The program code - This part contains the actual program, it consists of other words from the dictionary or literals (numbers).
- d. The semi-colon character ; - This is a special word that terminates the program. Every colon at the start of a word must be matched by a semi-colon at the end of the word.

It is useful to see if any programs exist in the dictionary before entering a program. This can be done using the words FWORDS and UWORDS. A more complete description of these words is given later in this section.

Type the following line:

FWORDS <Return>

This should list the names of all the fixed words on your terminal. You should recognise some of these from earlier sections.

Type the following line:

UWORDS <Return>

This should list the names of any words that have been added to the user section of the dictionary. If there are no words in the user dictionary an error message is printed.

Before writing any new words for the instrument, check that nobody else has any programs in the instrument that they want to keep. Then type the line:

NEW <Return>

This will delete all the user words in the RAM area of memory.

Write the following word and type <Return> when you have finished. Since you are getting the hang of the language, the <Return> command

will be left out of the following examples. You can tell when the key is required because the instrument's response is underlined.

```
: TEST 4 5 + . ;  
*Free nnn*
```

If you make a mistake the instrument will print an error message and retype the word so that you can modify it. When you are happy type the <Return> key. Note you do not need to move the cursor to the end of the line before typing the <Return> key.

If you get lost or confused, don't panic! Type the <Escape> key, and the instrument should print:

```
*Free nnn*
```

You can then start again.

This word should now be in your user word dictionary, try typing:

UWORDS

```
*  
** USER **  
*  
TEST  
*Free nnn*
```

This is the same sum that you tried in section 4.3.2, however this time it has been written as a program and stored in the dictionary.

To run the word type:

```
TEST  
2  
*Free nnn*
```

This time try writing a word to average two numbers on the stack, this word adds the top two numbers on the stack and then divides by two to calculate the average. The result is printed with a message.

```
: AVERAGE + 2 / ." The average is " . ;  
*Free nnn*
```

Try the example:

```
4 5 AVERAGE  
The average is 4.5  
*Free nnn*
```

You should now be able to see how you can build up words to do simple tasks that can be included in other words.

Try creating some words of your own, it is not difficult and with a little practice you should start to gain confidence with the language.



#### 4.5.2 Editing a Word

The editor within the instruments allows you to examine and modify words that you have written.

To examine an existing word type:

: Name <Return>

Where Name represents an existing program. If the program name does not exist you will get an error message on your terminal. This facility allows you to correct errors, or modify programs without having to start from scratch each time.

The editor also allows you to create new words that are similar to existing words. This is done by calling up an existing word on the terminal, you then modify the name of the word and the program, and store the new word in your dictionary.

#### 4.5.3 Dictionary Management Words

Whilst you are learning how to create words, it may be useful to consider other words in the fixed dictionary that are used for modifying the dictionary contents.

Word	Description
------	-------------

FWORDS	This word lists the names of all the words in the fixed dictionary on the terminal.
--------	-------------------------------------------------------------------------------------

UWORDS	This word lists the names of any words that the user has added to the dictionary on the terminal.
--------	---------------------------------------------------------------------------------------------------

AWORDS	(6356/6366 only) This word lists the names of the application programs in the fixed dictionary on the terminal.
--------	-----------------------------------------------------------------------------------------------------------------

ULIST	This word provides a complete listing of the program on the terminal by printing all the words in the user dictionary in the order in which they were entered. The listing is formatted to give some structure to the program. The listing can be suspended and restarted by typing a character on the keyboard.
-------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

FORGET	This word deletes the last word entered in the user dictionary. Words may only be deleted on a last in first out basis, to avoid conflicts between word references.
--------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------

INSERT This word allows a new word to be inserted within the current user dictionary. The user would type

INSERT Name

\*Free nnn\*

The new word can now be written and will be placed before the named word in the dictionary, ie it would appear to have been written before the named word.

NEW This word deletes all the programs stored in the RAM area of memory. It is useful when reprogramming an instrument.

STORE The STORE word is used to copy a program from the RAM area to the EEPROM area. Note that at power up the program in EEPROM is automatically copied to the RAM area, thus if the RAM area contains a different edited version it will be overwritten.

The STORE word takes up to 40 seconds to execute. Data is verified as it is written, and progress is indicated by printing an asterisk every 100 bytes. If the transfer fails then a printout occurs of the location and both the original and incorrect data.

Note that when you store a program you also store the current window size, user name and security code in the EEPROM.

RECALL The RECALL word is used to copy a program from the EEPROM area to RAM area. Note that this occurs automatically at power up, and any program in the RAM area will be overwritten.

Note that RECALL will also restore the window size, user name and security code stored in the EEPROM.

CLEAN The CLEAN word restores the EEPROM memory to the unprogrammed state. It operates in the same way as STORE by writing FF to to every location in EEPROM.

#### 4.5.4 Summary

Table 4.5 describes the words associated with modifications to the dictionary.

Operator	Stack	Description
: Namel	( -- )	Create a new program called Namel or list an existing program called Namel
;	( -- )	Terminate a : definition
FWORDS	( -- )	List the fixed words dictionary.
UWORDS	( -- )	Lists the user words dictionary.
AWORDS	( -- )	(6356/6366 only) List the application words dictionary.
ULIST	( -- )	List the user program.
FORGET	( -- )	Delete the last word entered in the user dictionary.
INSERT Namel	( -- )	Insert a new word before the word Namel.
NEW	( -- )	Delete all the programs stored in the RAM.
STORE	( -- )	Copy programs from RAM to EEPROM.
RECALL	( -- )	Copy programs from EEPROM to RAM.
CLEAN	( -- )	Clean the EEPROM memory.

Table 4.5 Dictionary and Word Management

Some examples of other stack manipulation words are given below, and a full list is given in Table 4.6.1.

Word	Description	Stack before	Stack after
DROP	Discard the top of stack item.	Top->: 2 : : 1 :	Top->: 1 : : - :
DUP	Make a copy of the top of stack item.	Top->: 5 : : 7 : : - :	Top->: 5 : : 5 : : 7 :
OVER	Move a copy of the second stack item to the top of the stack.	Top->: 3 : : 4 : : 9 : : - :	Top->: 4 : : 3 : : 4 : : 9 :
ROT	Rotate the top three stack items.	Top->: 6 : : 0 : : 8 : : 1 :	Top->: 8 : : 6 : : 0 : : 1 :
PICK	Replaces the pick argument with the required value from lower on the stack.	Top->: 3 : : 4 : : 6 : : 8 : : 0 :	Top->: 8 : : 4 : : 6 : : 8 : : 0 :
ROLL	Rotate the required number of values on the stack. Note: 2 ROLL is equivalent to SWAP 3 ROLL is equivalent to ROT	Top->: 3 : : 4 : : 6 : : 8 : : 0 :	Top->: 8 : : 4 : : 6 : : 0 : : - :

#### 4.6 Stack Manipulations

Previous sections have shown how you can do arithmetic with data on the stack, however a programmer will often find that data is not in the correct order on the stack for the required calculation.

Consider writing a word to find the reciprocal of a number on the stack. The short form description of this word could be:

INVERT (n -- 1/n) Replaces n with its reciprocal value.

The required computation would normally be written as:

1 n /

but for the general purpose word, how do you get the number 1 below n on the stack?

The solution is to use the word SWAP, which switches the order of the top two stack items. The example below shows how the stack is altered:

SWAP	Switch the order of the top two stack values.		Before		After		
		Top->:	n2	:	Top->:	n1	:
		:	n1	:	:	n2	:
		:	:	:	:	:	

Try the example:

1 2 . SPACE .  
2 1  
\*Free nnn\*

Now try the example

1 2 SWAP . SPACE .  
1 2  
\*Free nnn\*

You should now be able to write the word INVERT, this would be:

: INVERT 1 SWAP / ;

Try it and check that it really works.

#### 4.6.1 Local Variables

Note these local variables are not included in TCS Forth Version 3.

Programs requiring access to several arguments or temporary variables can result in intricate stack manipulations, which can be difficult to understand and maintain.

This problem can be alleviated by the use of local arguments within an ARG-RES (arguments-results) structure. A programmer can allocate up to 8 stack values to be used as arguments within an ARG-RES structure. At the same time space for 8 results is reserved on top of the stack. Both the arguments and results can be used as local variables within the structure. On leaving the ARG-RES structure a number of results can be left on top of the stack for use by other parts of the program.

To define an ARG-RES structure, the programmer defines the number of arguments on the top of the stack with the ARG word, and this automatically reserves space on the stack for 8 results

	Stack before	Stack after
This example shows how the stack is modified with the ARG word when three arguments are required. The value xxx indicates the initial limit on the stack.	Top->: 3 :	Top->Res8->: - :
	: 5 :	" 7->: - :
	: 6 :	" 6->: - :
	: 8 :	" 5->: - :
	: xxx :	" 4->: - :
	:	" 3->: - :
	:	" 2->: - :
	:	" 1->: - :
	:	Arg3->: 5 :
	:	" 2->: 6 :
	:	" 1->: 8 :
	:	: xxx :

To leave an ARG-RES structure the programmer defines the number of results that are to be left on the stack with the RES word. All other arguments, results and residual data on the stack are discarded.

	Stack before	Stack after
This example shows how the stack is modified with the RES word when two results are required. The value xxx indicates the initial limit on the stack when the ARG word was used.	Top->: 2 :	Top->: 0 :
	: m :	: 125 :
	Res8->: - :	: xxx :
	" 7->: - :	:
	" 6->: - :	:
	" 5->: - :	:
	" 4->: - :	:
	" 3->: - :	:
	" 2->: 0 :	:
	" 1->: 125 :	:
	Arg3->: 5 :	:
	" 2->: 6 :	:
	" 1->: 8 :	:
:	: xxx :	

The words GETARG, GETRES, SETARG and SETRES are used to access the local variables. Simple examples showing how the stack is affected are given below.

As you try to use these words, you will discover that they can only be used within an ARG-RES structure.

ARG-RES structures may be nested, but only the arguments and results associated with the current definition may be accessed.

Word	Description	Stack before	Stack after
GETARG	Copies the data from the required Argument to the top of the stack.	Top->: 2 : : 7 : : : Arg2->: 6 : : : : xxx :	Top->: 6 : : 7 : : : Arg2->: 6 : : : : xxx :
SETARG	Move data from the top of stack to the required argument.	Top->: 2 : : 4 : : 7 : : : Arg2->: 6 : : : : xxx :	Top->: 7 : : : Arg2->: 4 : : : : xxx : : :
GETRES	Copies the data from the required Result to the top of the stack.	Top->: 1 : : 7 : : : Res1->: 6 : : : : xxx :	Top->: 6 : : 7 : : : Res1->: 6 : : : : xxx :
SETRES	Move data from the top of stack to the required Result.	Top->: 2 : : 125 : : 7 : : : Res2->: 3 : : : : xxx :	Top->: 7 : : : Res2->: 125 : : : : xxx : : :

#### 4.6.2 Summary

Table 4.6.2 provides a concise list of the stack manipulation words described in this section.

Word	Stack notation	Description
DROP	(n1 -- )	Discard the top stack item.
DUP	(n1 -- n1 n1)	Duplicate the top stack item.
OVER	(n1 n2 -- n1 n2 n1)	Copy the second item and put it on top of the stack.
ROT	(n1 n2 n3 -- n2 n3 n1)	Rotate the third item to the top.
SWAP	(n1 n2 -- n2 n1)	Swap the top two stack items
PICK	(np .. n2 n1 p -- np .. n2 n1 np)	Replaces the PICK argument p with the data p elements deep on the stack. (Not in TCS Forth Version 1)
ROLL	(n1 n2 .. nr r -- n2 .. nr n1)	Rotate the top r elements on the stack and discard the ROLL argument. (Not in TCS Forth Version 1)
?DUP	(n -- n n) or (0 -- 0)	Duplicate the top stack item if it is not zero. (Not in TCS Forth Versions 1 & 2)
* ARG	(A1 .. An n -- A1 .. An R1 .. R8)	Mark n values on stack as local arguments and reserve 8 stack locations for a local results area.
* RES	(A1 .. An R1 .. R8 m -- R1 .. Rm)	Discard the local arguments and results, and replace with the first m results.
* GETARG	(n1 -- ARGn)	Stack the data from local argument n1.
* SETARG	(n1 n2 -- )	Move data n1 to local argument n2.
* GETRES	(n1 -- ARGn)	Stack the data from local result n1.
* SETRES	(n1 n2 -- )	Move data n1 to local result n2.
<hr/>		
* These words are not available in TCS Forth Version 3.		

Table 4.6.2 Stack Manipulation Words



#### 4.7 More Arithmetic and Trigonometric Words

This section describes additional arithmetic and trigonometric words available in the language. By now you should be familiar with the notation used to describe a word, and full descriptions are not given here, however it is worth trying each word out to familiarise yourself with what is available.

Word	Stack notation	Description
ABS	(n -- :n:)	Return absolute value.
MAX	(n1 n2 -- n-max)	Return maximum of two numbers.
MID	(n1 n2 n3 -- n-mid)	Return middle of three numbers. (Not available in Version 1)
MIN	(n1 n2 -- n-min)	Return minimum of two numbers.
MINUS	(n -- -n)	Negate the top of stack number.
SQR	(n -- n*n)	Square the top of stack number.
SQRT	(n -- SQRT(n))	Square root the top of stack number.
INT	(n -- n-int)	Return integer part of n.
E	( -- e)	Return e (2.71828) on stack.
PI	( -- pi)	Return pi (3.14159) on stack
SIN	(n -- sin(n))	Return sin(n), n expressed in radians.
COS	(n -- cos(n))	Return cos(n), n expressed in radians.
ATAN	(n -- atan(n))	Return atan(n), expressed in radians and in the range $-\pi/2$ to $+\pi/2$ .
ATAN2	(n1 n2 -- atan(n1/n2))	Return atan(n1/n2), expressed in radians and in the range $-\pi$ to $+\pi$ .
EXP	(n -- exp(n))	Return exp(n) on stack.
LN	(n -- ln(n))	Return natural log, ln(n) on stack.

Table 4.7 More Arithmetic and Trigonometric Words

#### 4.8 Logical Words

These words allow logical operations or boolean arithmetic to be used on data on the stack.

Since data is kept in floating point, programmers should consider the way numbers are treated before the operation.

Numbers are treated slightly differently in Version 1, 2 and 3.

##### TCS Forth Versions 1 and 2

For all the logical words except NOT the two stack entries are first rounded to the nearest integer, and limited to 16 bit integers in the range 0 to 65535 (Hex values 0 to FFFF). Negative numbers are all treated as 0. The boolean operation is then performed on the two 16 bit words, and the result is converted to the floating point result.

##### TCS Forth Version 3

For all the logical words except NOT the two stack entries are first rounded to the nearest integer. Positive numbers are limited to 16 bit integers in the range 0 to 65535. Negative numbers in the range -65536 to -1 are converted to their 16 bit 2's complement form, and numbers less than -65536 are treated as 0. The boolean operation is then performed on the two 16 bit words, and the result is converted to the floating point result.

The lists below show how numbers are converted to 16 bit values for boolean arithmetic.

##### Positive numbers Versions 1 2 and 3

Decimal number	Hexadecimal equivalent
65535	FFFF
65534	FFFE
32769	8001
32768	8000
32767	7FFF
1	0001
0	0000

##### Negative numbers Version 3 only

Decimal number	Hexadecimal equivalent
-1	FFFF
-2	FFFE
-32767	8001
-32768	8000
-32769	7FFF
-65535	0001
-65536	0000

#### 4.8.1 Examples

These examples show the effects of the logical or boolean words on the stack.

Word	Description	Stack before	Stack after
AND	Form the logical AND on the top two stack values.	Top->: 3 : : 1 : : 5 :	Top->: 1 : : 5 : : - :
OR	Form the logical OR on the top two stack values.	Top->: 2 : : 1 : : 5 :	Top->: 3 : : 5 : : - :
XOR	Form the logical XOR on the top two stack values.	Top->: 5 : : 7 : : 5 :	Top->: 2 : : 5 : : - :
NOT	Replaces a 0 on top of the stack with a 1, otherwise returns a 0.	Top->: 0 : : 5 :	Top->: 1 : : 5 :

#### 4.8.2 Summary

Word	Stack notation	Description
AND	(n1 n2 -- and)	Return the logical AND.
OR	(n1 n2 -- or)	Return the logical OR.
XOR	(n1 n2 -- xor)	Return the logical XOR.
NOT	(n -- f)	Return flag where: f=true=1 if n=0, or f=false=0 if n≠0

Table 4.8 Logical Words

#### 4.9 Comparison Words

The comparison words allow tests on data on the stack. The result is returned as a flag on the stack where:

Flag = 1 = True  
Flag = 0 = False

Since these words are reasonably simple to understand, you are left to your own devices to find ways of testing them.

Word	Stack notation	Description
>	(n1 n2 -- f)	Return true flag if n1 is greater than n2.
<	(n1 n2 -- f)	Return true flag if n1 is less than n2.
=	(n1 n2 -- f)	Return true flag if n1 is equal to n2.
0>	(n -- f)	Return true flag if n is positive.
0<	(n -- f)	Return true flag if n is negative.
0=	(n -- f)	Return true flag if n is zero.

Table 4.9 Comparison Words

#### 4.10 Variable and Timer Words

The TCS Forth provides access to 64 general purpose floating point variables, and 16 timers.

##### 4.10.1 Variables

In normal use, temporary data is stored on the stack, and modified or discarded after it has been used, however, in some cases intermediate results may need to be stored for later use elsewhere in a program. For these applications the programmer can read or modify 64 floating point variables. These are referred to as variables 1 to 64.

In addition to reading or writing to the variables, words are also provided to integrate or accumulate data into a variable, and also to form the difference between new data and the current data stored in a variable. Examples of how these words affect the stack and the variables are given below:

Word	Description	Stack before	Stack after
SETVAR	Store the second stack item in the variable given by the top stack item.	Top->: 7 : : 124 : : 3 :	Top->: 3 : : - : : - :
		Var 7 = xxx	Var 7 = 124
GETVAR	Replace the variable number with the data stored in the variable.	Top->: 7 : : 3 : : - :	Top->: 124 : : 3 : : - :
		Var 7 = 124	Var 7 = 124
SUMVAR	Add the second stack item to the variable given by the top stack item, and return the result on the stack.	Top->: 7 : : 5 : : 3 :	Top->: 129 : : 3 : : - :
		Var 7 = 124	Var 7 = 129
DIFVAR	Store the second stack item in the variable given by the top stack item, and return the difference on the stack.	Top->: 7 : : 138 : : 3 :	Top->: 9 : : 3 : : - :
		Var 7 = 129	Var 7 = 138

#### 4.10.2 Timers

The timers are generally used in sequencing operations, for example a programmer may want to measure the time period between two events, check that actions are taken within a given time period, or provide a delay. For these purposes 16 timers are provided, referred to as timers 1 to 16.

The timers are set and read in seconds, however, they are stored as 32 bit registers, where one bit represents approximately 2ms. If you do the arithmetic you will find that this represents plus or minus 4,294,967 sec or just over 7 weeks.

The timers count downwards, so that if a timer is initialised to 10, then 10 sec later it will read back as 0, and a further 10 sec later it will read back as -10. Timer values are updated continuously when the instrument is powered up, this means that when a timer overflows, below -4,294,967, it will automatically be reset to +4,294,967, and continue counting down.

Due to the floating point format used within the instrument and the 2ms update rate, the timers can be set or read to a resolution which is the larger of 2ms or 1 part in  $2^{24}$ .

Some examples of the effects of the timer words on the stack are given below:

Word	Description	Stack before	Stack after												
SETTIM	Store the second stack item in the timer given by the top stack item.	Top-> <table><tr><td>3</td><td>:</td></tr><tr><td>100</td><td>:</td></tr><tr><td>9</td><td>:</td></tr></table>	3	:	100	:	9	:	Top-> <table><tr><td>9</td><td>:</td></tr><tr><td>-</td><td>:</td></tr><tr><td>-</td><td>:</td></tr></table>	9	:	-	:	-	:
3	:														
100	:														
9	:														
9	:														
-	:														
-	:														
		Tim 3 = xxx	Tim 3 = 100												
GETTIM	Replace the timer number with the current time in the timer.	Top-> <table><tr><td>3</td><td>:</td></tr><tr><td>9</td><td>:</td></tr><tr><td>-</td><td>:</td></tr></table>	3	:	9	:	-	:	Top-> <table><tr><td>-15</td><td>:</td></tr><tr><td>9</td><td>:</td></tr><tr><td>-</td><td>:</td></tr></table>	-15	:	9	:	-	:
3	:														
9	:														
-	:														
-15	:														
9	:														
-	:														
		Tim 3 = -15	Tim 3 = -15												
+TIM	Add the second stack item to the timer given by the top stack item.	Top-> <table><tr><td>3</td><td>:</td></tr><tr><td>100</td><td>:</td></tr><tr><td>9</td><td>:</td></tr></table>	3	:	100	:	9	:	Top-> <table><tr><td>9</td><td>:</td></tr><tr><td>-</td><td>:</td></tr><tr><td>-</td><td>:</td></tr></table>	9	:	-	:	-	:
3	:														
100	:														
9	:														
9	:														
-	:														
-	:														
		Tim 3 = -22	Tim 3 = 78												

#### 4.10.3 Summary

Word	Stack notation	Description
GETVAR	(Vn -- n)	Return value stored in variable Vn.
SETVAR	(n Vn -- )	Store value n in variable Vn.
DIFVAR	(n1 Vn -- n2)	Return the result n2=n1-value in variable Vn, and store value n1 in variable Vn.
SUMVAR	(n1 Vn -- n2)	Add n1 to the value stored in variable Vn, the result is also returned as n2.
GETTIM	(Tn -- n)	Return current value in timer Tn.
SETTIM	(n Tn -- )	Store time n seconds in timer Tn.
+TIM	(n Tn -- )	Add n seconds to value in timer Tn.

Table 4.10 Variable and Timer Words

#### 4.11 Terminal Input

This section describes the words that allow a program to receive data from the programming terminal.

The words are normally used in conjunction with the printing words described in section 4.4, and are particularly useful when debugging and testing new words.

The word KEY examines the input buffer to see if any keys have been pressed, and returns immediately to the program with the result on the stack.

The word NUMBER suspends the program until a valid number has been entered at the keyboard.

The effects these words have on the stack is shown below. Naturally these words would be used in different applications, and examples of words using KEY and NUMBER are given in sections 4.12 and 4.13.

Word	Description	Stack before	Stack after
KEY	Read any characters from the programming terminal.	Top->: 5 : : - :	Top->: 0 : : 5 :
Example where no key was typed			
		Top->: 5 : : - : : - :	Top->: 65 : : 65 : : 5 :
Example where letter A was typed			
NUMBER	Suspends the program until a valid number has been entered.	Top->: 5 : : - :	Top->: 7.1 : : 5 :
Example where 7.1 was typed			

---

Operator	Stack notation	Description
KEY	( -- c c) or ( -- 0)	Return the ASCII value of the next character from the terminal and a 1 flag, or a 0 flag if no character available.
NUMBER	( -- n1)	Suspend program till a valid number is entered at the keyboard.

---

Table 4.11 Terminal Input Words



#### 4.12 Conditional Branches

A characteristic of a programming language is the ability to make decisions. This allows you to follow different routes through a program depending on various conditions.

Two types of conditional branch structures are described here:

##### 4.12.1 IF ... ELSE ... ENDIF structures

This structure allows a programmer to select a choice of two routes through a program.

The IF word tests and discards the number on top of the stack. If the number is not zero, then the statement following the IF is executed. If the number was zero then the statement after the the next appropriate ELSE would be executed.

To reduce the chance of programming errors, the TCS Forth Interpreter checks the program before it is compiled, this means that the words IF and ENDIF must occur in pairs to mark the start and end of the branch structure, and only one ELSE should be used between the IF and ENDIF.

Try the following example on your terminal, by creating the words GETNUM and T1, and then execute T1.

By this stage you should be familiar with the TCS Forth and from now on examples will show how words would be used in a program, and the intermediate responses from the instrument will not be given.

```
: GETNUM ." Enter a number" NUMBER ;
```

```
: T1 GETNUM NL 0< IF ." Negative" ELSE ." Positive" ENDIF ;
```

The ELSE word is optional within an IF .. ENDIF pair, as shown in the following example:

```
: GET+NUM ." Enter a positive number" NUMBER ;
```

```
: T2 GET+NUM NL 0< IF ." I said a positive number" ENDIF ;
```

The IF .. ELSE .. ENDIF structures can be nested, but remember this will make a program more complicated, and will require careful testing.

#### 4.12.2 CASE ... ENDCASE Structures

This structure allows one of several options to be selected depending on the initial argument on top of the stack, this can provide a very powerful structure, particularly in sequencing applications.

The following three sets of words are associated with this structure, and must always occur in pairs to mark the start and end of a section of program.

```
CASE ... ENDCASE
OF ... ENDOF
ELSOF ... ENDOF
```

##### CASE ... ENDCASE

The CASE and ENDCASE words mark the bounds of a section of program where the OF, ELSOF and ENDOF words can be used.

The CASE word leaves the original entry or 'CASE argument' on the stack unchanged, whilst the ENDCASE word will drop this argument if it has not been dropped by an OF or ELSOF word within the structure.

##### OF ... ENDOF

The OF word examines the two numbers on top of the stack, these would normally be the original CASE argument and the OF argument (which is on top of the stack). The OF condition is defined as true when the CASE argument is less than or equal to the OF argument.

If the OF condition is true then both the CASE and OF arguments are dropped off the stack, and execution continues from the word following the OF.

If the OF condition is false then just the OF argument is dropped off the stack, and execution is transferred to the word following the appropriate ENDOF.

The ENDOF word marks the end of an OF ... ENDOF section, and transfers execution of the program to the word following the appropriate ENDCASE.

##### ELSOF ... ENDOF

The ELSOF word can be used to trap CASE arguments greater than the maximum OF argument, and it is assumed that the CASE argument is still on the stack.

If the ELSOF word is executed then the CASE argument is dropped off the stack, and execution continues from the word following the ELSOF.

The ENDOF word marks the end of an ELSOF ... ENDOF section, and transfers execution of the program to the word following the appropriate ENDCASE.

Note that since the ELSOF word transfers program execution to the word following the ENDCASE, only one OF ... ENDOF or ELSOF ... ENDOF segment can be executed each time the CASE structure is executed.

The following example shows how different words can be executed depending on a number typed at the keyboard. You should be able to see how these messages could be replaced with particular operations in a sequencing application.

```
: M1 NL ." Number less than or equal to 0" ;
: M2 NL ." Number between 0 and 1" ;
: M3 NL ." Number greater than 1" ;
: T3 NUMBER CASE 0 OF M1 ENDOF 1 OF M2 ENDOF ELSOF M3 ENDOF ENDCASE ;
```

#### 4.12.3 Summary

Word	Stack notation	Description
IF	( f -- )	If f is true (non-zero) execute code following IF word. Otherwise transfer execution to the word following the ELSE (if it exists) or the ENDIF.
ELSE	( -- )	Marks the end of a block of conditional code. Transfer execution to the ENDIF word
ENDIF	( -- )	Marks the end of an IF structure.
CASE	( n1 -- n1 )	CASE marks the start of the CASE structure.
ENDCASE	( n1 -- )	Drops the case argument.
OF	( n1 n2 -- n1 ) or ( n1 n2 -- )	If n1 <= n2 then drop n1 and n2 and execute following code. Otherwise drop n2 and transfer execution to the first word <u>after</u> the ENDOF statement.
ELSOF	( n1 -- )	Drop the case argument n1 and execute following code.
ENDOF	( -- )	Transfer execution to the first word <u>after</u> the ENDCASE statement.

Table 4.12 Conditional Branches

#### 4.13 Loop Structures

The last section covered 'decision making' by conditionally executing some parts of a program.

This section covers words that allow a program to conditionally branch back to an earlier part of the program. This type of structure is called a loop and allows repetition of a segment of program.

Words are provided to allow a programmer to execute a section of code a number of times, to wait until some event occurs or simply to sit in a loop indefinitely (not as stupid as it sounds).

Loops are usually divided into two types, definite and indefinite loops. A summary of the words used in these structures together with their effects on the stack is given at the end of this section, whilst a description of how the words work and examples are given below.

##### 4.13.1 Definite Loops

Definite loops have a defined start and end point.

##### DO ... LOOP structure

The DO and LOOP words define the start and end points of a loop structure, and must always be used as a pair.

The word DO sets up a loop and takes two arguments from the stack. The top number is the initial index or starting point for the loop, and the second value is the loop limit. The DO word takes these values off the stack, and stores them for its own internal use.

The word LOOP increments the current index value, and compares this to the limit value given to the DO word. If the new index is less than the limit then the program branches back to the word following the DO. When the new index is equal to the limit the program continues with the word following the LOOP.

Since the LOOP word increments the index before comparing it against the limit, the loop is never executed with the index equal to the limit, this should become clear in the later examples. The following code would print the word Hello 10 times on your terminal--try it.

```
10 0 DO NL ." Hello" LOOP
```

### The I word

The word I stacks the index for the current DO ... LOOP structure. This index was mentioned in the previous paragraphs and describes the current status of the loop.

The following example shows how the index varies, the loop is executed 10 times with an index starting from 0, but is never executed with an index = limit.

```
10 0 DO I . LOOP
```

this should print the following on your terminal:

```
0123456789
```

### DO ... +LOOP structure

The DO and +LOOP words define the start and end points of a loop structure, however, whilst the LOOP word simply increments the index the +LOOP allows other increments. The words DO and +LOOP must always be used as a pair.

The word +LOOP takes a value off the stack and adds it to the current index value, the new index is then compared to the limit value given to the DO word. If the new index has not yet reached the limit then the program branches back to the word following the DO. When the new index is equal or passes the limit the program continues with the word following the +LOOP.

The following example shows how the index can be altered with +LOOP. Do you understand why the initial index in this example was not set to 0 ?

```
128 1 DO I . SPACE I +LOOP
```

### The LEAVE Word

The word LEAVE sets the current index value equal to the final limit. This causes the program to leave the loop the next time the word LOOP or +LOOP is encountered.

Note that DO ... LOOP and DO ... +LOOP structures can be nested, however, the words I and LEAVE only operate on the current loop index.

#### 4.13.2 Indefinite Loops

This type of loop can repeat indefinitely, or until some event occurs.

##### BEGIN ... REPEAT structure

Normally you will write a series of words that will scan the inputs and outputs and take any appropriate action. At this point the program would stop, and you would have to execute the words again from the keyboard.

The BEGIN ... REPEAT structure allows your words to be included in an infinite loop, so that your program will run indefinitely, or at least as long as the instrument is powered up.

The BEGIN word marks the start of an indefinite loop, it also clears the \*HALTED\* message or flags that can flash on the front panel of the instrument.

The REPEAT word marks the end of an indefinite loop, and branches the program back to the BEGIN word.

Note for instruments with multi-tasking. When the word REPEAT is encountered in a time scheduled task, the task will be descheduled, however, the next time the program is scheduled it will recommence at the BEGIN word.

The following example will print the letter A on your terminal indefinitely, or until you stop the program with the <Escape> key.

```
BEGIN 65 EMIT REPEAT
```

##### BEGIN ... UNTIL structure

The BEGIN ... UNTIL structure surrounds a section of program that runs indefinitely, until a condition is met.

The UNTIL word takes a number off the stack. If this number is 0 the program branches back to the BEGIN word, otherwise the program continues with the word following UNTIL.

Note for instruments with multi-tasking. When the word UNTIL is encountered in a time scheduled task and the UNTIL condition is not satisfied, the task will be descheduled; the next time the program is scheduled it will recommence at the BEGIN word. When the UNTIL condition is satisfied the program continues with the word following the UNTIL.

The following example waits until a key is typed on the terminal:

```
BEGIN KEY UNTIL ." That was " .
```

#### 4.13.3 Summary

Word	Stack notation	Description
DO	(limit index -- )	Set up a finite loop, with a start point for the index and a final limit.
LOOP	( -- )	Increment the index, and transfer execution back to the word following the DO while index < limit.
+LOOP	(n -- )	Add n to the index, and transfer execution back to the word following the DO until the index reaches or passes the limit.
I	( -- index)	Return the current loop index value.
LEAVE	( -- )	Leave the loop at the next LOOP or +LOOP.
BEGIN	( -- )	Mark the start of indefinite loop.
REPEAT	( -- )	Transfer execution back to the BEGIN word.
UNTIL	(f -- )	Transfer execution back to the BEGIN word while the flag is false (0).

Table 4.13 Loop Structures

#### 4.14 Instrument Specific Words

The TCS Forth provides a common programming style on the programmable instruments, however each instrument is designed for a different application, and there are some words that are unique to each instrument. These words cover data base access, display selection and other special functions.

Full descriptions of these words with application examples are given in the appropriate technical manuals, however, for completeness, all the words are also described here.

##### 4.14.1 Special 6433 Words

The 6433 Programmable Signal Processor is capable of sophisticated computation and sequencing functions dependent on the state of the input and output signals stored in its data base. This section describes the words used to inspect or modify these values.

Each input or output from the instrument is described by a Board number (Bn) and a Channel number (Cn). The following four words are used to access the inputs and outputs.

**GETAN** Moves the analogue data from the required board and channel to the top of the stack. The value is returned in engineering units with the full resolution available from the a-d conversion. The following example would print the current value on analogue board 1 channel 3.

1 3 GETAN .

**SETAN** Moves the data, in engineering units, to the required board and channel output. The following example would set the analogue output on board 2 channel 5 to 80 units.

80 2 5 SETAN

**GETDIG** Moves the digital status from the required board and channel to the top of the stack. The value returned is either a 1 or 0 depending on the status of the input or output. The following example would print the current status of digital board 3 channel 7.

3 7 GETDIG .

**SETDIG** Modifies the digital status of the required board and channel output. The output is set low if the data is zero, otherwise it is set high. The following example would set the digital output on board 4 channel 2 high.

1 4 2 SETDIG



A program can also access the complete data base of the instrument, including status words, ranges and limits. Each parameter in the data base is described by a Board number (Bn) and a Parameter number (Pn), where the parameter numbers are identical to those used for the binary protocol. The data is returned to "display" resolution, ie the same as it would appear on the front of the instrument or on a hand-held terminal. The following two words are used to access these parameters.

GETPAR Moves the data from the required board and parameter to the top of the stack. If board 1 of your instrument is an analogue board, the following example would print the current value of the high range of board 1 channel 3.

1 33 GETPAR .

SETPAR Moves the data to the required board and parameter. If board 2 of your instrument is an analogue output board, the following example would set the output high limit of board 2 channel 4 to 95 units.

95 2 44 SETPAR .

In some cases a programmer may want to force the front panel display to a particular board and channel. This could be useful for alarm annunciation or operator interaction. The following words allow the program to monitor or change the front panel display.

GETBCN Returns the board and channel number currently selected on the front panel display. The following example will print the current front panel board and channel numbers.

GETBCN SWAP ." You are looking at board " . ." channel " .

SETBCN Selects a specific board and channel number for the front panel display. The following example sets the front panel to board 1 channel 5.

1 5 SETBCN

TAG." This word transfers the following text string, up to 8 characters, to the tag display of the currently selected front panel board and channel. The following message would print START UP on the tag display.

TAG." START UP"

#### 4.14.2 Summary of Special 6433 Words

Word	Stack notation	Description
GETAN	(Bn Cn -- n)	Return analogue value from the appropriate Board and Channel.
SETAN	(n Bn Cn -- )	Store analogue value n to the appropriate Board and Channel.
GETDIG	(Bn Cn -- f)	Return a flag depending on the digital status of appropriate Board and Channel.
SETDIG	(f Bn Cn -- )	Set or clear digital status on appropriate Board and Channel.
GETPAR	(Bn Pn -- n)	Return value from appropriate Board and Parameter.
SETPAR	(n Bn Pn -- )	Store value n in appropriate Board and Parameter.
GETBCN	( -- Bn Cn)	Return the Board and Channel numbers currently displayed on the front panel.
SETBCN	(Bn Cn -- )	Set the front panel display to the required Block and Channel.
TAG." abc" ( -- )		Transfers the string abc to the current tag. The " character terminates the string.

Table 4.14.2 6433 Data Base Access Words

#### 4.14.3 Special Advanced Controller Words

The Advanced Controllers allow a programmer to set up a special control strategy within a stand alone instrument. This can simplify many installations that have previously required considerable analogue and digital computation units in addition to a controller.

Data in the instrument is referenced by a Block number (Bn) and a Parameter number (Pn), see Appendix 2.

To simplify access to the data base fixed words are provided to place the Block and Parameter numbers on the stack. The Block numbers are given by a 3 character word, the first two characters are the block mnemonic and the third character is the relative block number. The following example would print the absolute block number of Setpoint Block 2.

SP2 .

The parameter numbers are given by a 2 character word which is identical to the mnemonic used by the hand-held terminal. The following example would print the parameter number for PV.

PV .

The following words are used to access the Advanced Controller data base.

GET Moves the analogue data from the required block and parameter to the top of the stack. The following example would print the Process Variable from Setpoint Block 1.

SP1 PV GET .

SET Moves data to the required Block and Parameter. The following example would set the Local Setpoint in Setpoint Block 2 to 40 units.

40 SP2 SL SET

%GET This is similar to GET, however the word returns the data as a normalised value in the range -100% to +100%. The following example prints the Process Variable from Setpoint Block 1 as a percentage of its range.

SP1 PV %GET .

%SET This is similar to SET, however the data on the stack is expressed as a percentage of full range. The following example would set the Local Setpoint in Setpoint Block 2 to 20% of its range.

20 SP2 SL %SET

GETDIG Moves the digital status from the required digital input to the top of the stack. The value returned is either a 1 or 0

depending on the status of the input or output. The following example would print the current status of digital input 3.

D11 3 GETDIG .

SETDIG Modifies the required digital output. The output is set low if the data is zero, otherwise it is set high. The following example would set digital output 4 low.

0 D01 4 SETDIG

The Advanced Controllers can run two time scheduled programs in addition to a background program. The following words are used to modify and monitor the program repeat times, or to start and stop the user programs.

GETREP Moves the program repeat time (in seconds) to the top of the stack. The following example would print the program repeat time for the first time scheduled program.

1 GETREP .

SETREP Sets the program repeat time. The following example would set the second time scheduled program to run at .5 second intervals.

.5 2 SETREP

RUN This forces a dictionary search for the programs defined in the General Purpose Block parameters L1, L2 and BG. If the programs are found they are then installed and run. (This is automatically done when the instrument is powered up).

HALT This stops all user programs. Successful attempts to edit user programs automatically force a HALT.

The Advanced Controllers use a number of additional words associated with the special blocks within the data base. These words are briefly described below.

PID This word is used to compute a control output from the Process Variable on the stack. The following example takes an input of 1200 units and stores the value as the Process Variable in Setpoint Block 1. It then calculates an output dependent on the Setpoint in Setpoint Block 1, the 3-term parameters in PID Control Block 1 and the mode in Display and Control Block 1. The result is then discarded.

1200 371 PID DROP

PIDX Normally a control loop is made up of a Setpoint Block, a PID Block, a Manual Station Block and a Display and Control Block, which are automatically linked together. In some cases only one control loop is required, but with two sets of 3-term constants. The PIDX word allows the user to specify which PID block is used, and which loop it is linked to. The following

example takes an input of 1500 units and stores the value as the Process Variable in Setpoint Block 1. It then calculates an output dependent on the Setpoint in Setpoint Block 1, the 3-term parameters in PID control Block 2 and the mode in Display and Control Block 1. The result is then discarded.

1500 1 3T2 PIDX DROP

**MSCONT** Moves data from the stack to the output register of the appropriate Manual Station Block only when the loop is in an AUTO mode. The following example sets the output of Manual Station 1 to 50% when loop 1 is in AUTO.

50 MS1 MSCONT

**REMOTE** Moves data from the stack to the Remote Setpoint of the required Setpoint Block. This also configures the loop as a remote setpoint controller. The following example sets the Remote Setpoint register of Setpoint Block 2 to 65 units.

65 SP2 REMOTE

**%REMOTE** This word is similar to REMOTE, however the setpoint is expressed as a percentage of the setpoint range. The following example would set the Remote Setpoint of Setpoint Block 1 to 25%.

25 SP1 %REMOTE

**RATIO** Uses data on the stack as the ratio process variable and moves the result to the Remote Setpoint of the appropriate Setpoint Block. This also configures the loop as a ratio controller. The following example takes a ratio process variable of 800, and calculates the Remote Setpoint for Setpoint Block 1, using the values in Ratio Block 1.

800 SP1 RATIO

**ALARM** Moves the data on the stack to the appropriate Alarm Block and updates the alarm block status bits. The following example would set Alarm Block 2 Process Variable to 400, check this value against the alarm limits, and update the alarm status bits.

400 AB2 ALARM

**FILTER** Uses the data on the stack as the input to the appropriate Filter Block, and returns the resultant output on the stack. The following example prints the result of applying an input of 30% to Filter Block 1.

30 FB1 FILTER .

**SETDEL** Moves data from the stack into the buffer of the appropriate Delay Block. The following example moves the value 18 into Delay Block 1.

18 DB1 SETDEL

GETDEL Moves data from the appropriate delay line to the stack. If the following example is included in a program it will use Delay Block 1 to retransmit an analogue input delayed by 20 seconds.

AI1 AV GET DB1 SETDEL 20 DB1 GETDEL AO1 AO SET

TOTAL Takes data from the stack as the input to the appropriate Totalisation Block. The word returns a flag which is 0 if the Flow Total is unchanged, or 1 if the Flow Total has been incremented. The following example uses the value 10 as the current input to Totalisation Block 1, and prints the flag on the terminal.

10 TB1 TOTAL .

The following word allows a user program to change or lock the front panel display to a particular loop.

SETLN Takes the front panel loop number from the stack. If the loop number is negative this locks the front panel to the selected loop, and means the user cannot change the displayed loop from the front panel. The following example would set the front panel to display loop 2.

2 SETLN

#### 4.14.4 Summary of Advanced Controller Special Words

This section describes the data base access routines and special function words associated with the Advanced Controller.

Word	Stack notation	Description
GET	(Bn Pn -- n)	Return data n from the Block and Parameter.
SET	(n Bn Pn -- )	Move data n to the Block and Parameter.
%GET	(Bn Pn -- n)	Return data n from the Block and Parameter. The data is returned as a percentage of range
%SET	(n Bn Pn -- n)	Store the value n expressed as a percentage in the Block and Parameter.
GETDIG	(Bn Dn -- f)	Return a flag depending on the status of the Block and Digital channel.
SETDIG	(f Bn Dn -- )	Modify status on Block and Digital channel.
GETREP	(n1 -- n2)	Return program n1 repeat time in seconds.
SETREP	(n1 n2 -- )	Sets program n2 to run every n1 seconds.
RUN	( -- )	Search install and run the programs defined in parameters L1, L2 and BG.
HALT	( -- )	Stop all user programs.
PID	(PV Bn -- OP)	Compute the control output from the Process Variable using parameters in PID block Bn.
PIDX	(PV n Bn -- OP)	Compute the control output from the Process Variable using parameters in PID block Bn. Link PID constants to loop n
MSCONT	(OP Bn -- )	Update the OP register of the Manual Station Block Bn when the loop is in an AUTO mode.
REMOTE	(SP Bn -- )	Update the Remote Setpoint of Setpoint Block Bn.
%REMOTE	(%SP Bn -- )	Update the Remote Setpoint of Setpoint Block Bn. Setpoint expressed in percentage.
RATIO	(RPV Bn -- )	Uses Ratio Process Variable to update the Remote Setpoint of Setpoint Block Bn.
ALARM	(PV Bn -- )	Update the Alarm Block PV and ST registers.
FILTER	(PV Bn -- OP)	Update the Filter Block FI and OP registers.

SETDEL	(n1 Bn -- )	Push data n1 into the buffer of the Delay Block.
GETDEL	(n1 Bn -- n2)	Return data delayed by n1 seconds from Delay Block Bn.
TOTAL	(n1 Bn -- f)	Totalise data n1 in Totalisation Block Bn. The flag is set if the Flow Total has increased.
SETLN	(n -- )	Set front panel display to loop n. If n is negative, disable front panel loop changes.

---

Table 4.14.4 Advanced Controller Special Function Words



#### 4.14.5 Special Microsupervisor Words

The Microsupervisor provides the computation and sequencing facilities associated with the TCS range of programmable instruments.

In addition to the normal serial link to a computer supervisory system, the instrument has a serial line that allows the supervision and interaction of a group of instruments. Further serial lines are provided that can be connected to a printer or terminal to allow logging and some operator interactions.

Data from individual instruments connected to the serial bus is accessed by an Instrument number (In) and the binary protocol Parameter number (Pn). The following words are used to access the instrument data.

**GETEXT** This word returns an error flag and data from the specified point. If there is no error, the data is stored on the stack and the error flag is set to 0. When an error occurs, only a non zero error flag is returned on the stack. The following example would fetch the instrument identity of instrument 3 and print it on a terminal.

```
3 18 GETEXT IF ." ERROR" ELSE ." II=" # ENDIF ;
```

The error numbers associated with GETEXT are listed in section 4.14.6.

**SETEXT** This word moves data to the required instrument and parameter and returns a flag on the stack. The flag is 0 if there is no error. The following example shows how the proportional band of a controller with instrument number 8 could be set to 25%.

```
25 8 20 SETEXT IF ." ERROR" ENDIF
```

The Microsupervisor has 8 digital inputs, 8 digital outputs and 6 pushbuttons. The following words are used to modify or monitor these signals.

**GETDI** Moves the digital status of the required input to the top of the stack. The value returned is either a 1 or 0, depending on the status of the input. The following example would print the status of digital input 2.

```
2 GETDI .
```

**SETDO** Modifies the digital status of the required output. The output is set low if the data is zero, otherwise it is

set high. The following example will set digital output 5 low.

0 5 SETDO

GETDS Returns the status of all 8 digital inputs as a number on top of the stack. The following example would print the digital input status on the terminal.

GETDS #

SETDS Modifies all digital outputs with the data on top of the stack. The following example will set digital outputs 1,2,3 and 4, and reset digital outputs 5,6 and 7. Digital output 8 will be reset if it is not allocated to a changeover relay.

#F000 SETDS

?F Returns the status of the required front panel pushbutton. The value returned is either a 1 or 0 depending on the status of the button. The following example would print the status of pushbutton 5 on the terminal.

5 ?F .

To allow data and messages to be logged to a printer, or operator interaction at a terminal, the program can select which serial line is used for terminal input and output.

LINE Selects a serial line or display for input and output. The following line number options are available:

- 3 Alternate programming terminal serial line.
- 4 Printer serial line.
- 5 Front panel serial line.
- 6 Front panel tag display.

The following example would send a message to the printer and then switch input and output back to the front panel line.

4 LINE ." PRINTER TEST" 5 LINE

The following words allow a program to present messages or data on the front panel tag display.

TAG." This word transfers the following text string, up to 8

characters, to the tag display. This example would show WAIT KEY on the tag display.

TAG." WAIT KEY"

TAG. Prints the number on top of the stack on the tag display. This example prints a number on the tag display.

12.34 TAG.

When using the Microsupervisor in data logging applications, the standard print routines do not provide a tidy way of displaying numbers. The following words are used to format numbers in the same as they would appear on the front panel of a standard TCS instrument.

LFMT. Prints a number as 8 characters left justified, with a defined number of characters after the decimal point. This example prints a number with two characters after the decimal point.

12.5 2 LFMT.

RFMT. Prints a number as 8 characters right justified, with a defined number of characters after the decimal point. This example prints a number with three characters after the decimal point.

1.234 3 RFMT.

To enable data logging applications, a time of day clock and a calendar are included in the instrument. The words used to access these features are described below.

GETCLK Returns the time as three values on the stack. The following example would print the time in the order - hours, minutes and seconds.

GETCLK . SPACE . SPACE .

SETCLK Uses three values on the stack to initialise the clock. The following example sets the clock to 10:30:00: (half past ten).

10 30 0 SETCLK

ADJCLK Due to variations in components during manufacturing of the instruments, the clock may gain or lose time. A compensation value is stored in EEROM and this word allows a user to adjust the value. For example, if the clock were gaining 5 seconds a day, the user would enter:

-5 ADJCLK

CLK. Prints the current time as an eight character string on the terminal.

KEYCLK Waits for the user to enter the current time on a terminal and echoes the values entered. The following example prints a prompt and waits for a user reply.

NL ." --:--:--" CR KEYCLK

GETDATE Returns the date as three values on the stack. The following example would print the date in the order - day, month and year.

GETDATE . SPACE . SPACE .

SETDATE Uses three values on the stack to initialise the date. The following example sets the date to 25th December 1986. The date algorithm ignores the century value if it is entered.

25 12 1986 SETDATE  
or 25 12 86 SETDATE

DATE. Prints the current date as an eight character string in the order: day, month and year.

KEYDATE Waits for the user to enter the current date on a terminal and echoes the values entered. The following example prints a prompt and waits for a user reply.

NL ." dd-mm-yy" CR KEYDATE

#### 4.14.6 Summary of Special Microsupervisor Words

Word	Stack notation	Description
GETEXT	(In Pn -- n 0) or (In Pn -- f)	Return data n from instrument and parameter. The flag describes any error conditions.
SETEXT	(n In Pn -- f)	Store value n to instrument and parameter. The flag describes any error conditions.
GETDI	(Cn -- f)	Return flag describing the digital input status.
SETDO	(f Cn -- )	Set or clear digital output Channel.
GETDS	( -- n)	Return the status of the 8 digital inputs.
SETDS	(n -- )	Modify the status of the 8 digital outputs.
?F	(n -- f)	Return the status of front panel switch n.
LINE	(n -- )	Select serial line n for input and output.
LFMT.	(n1 n2 -- )	Print n1 as a left justified number with n2 digits after the decimal point. (0 <= n2 <= 4.)
RFMT.	(n1 n2 -- )	Print n1 as a right justified number with n2 digits after the decimal point. (0 <= n2 <= 4.)
TAG." ab" ( -- )		Prints the string ab on the tag display. The " character terminates the string.
TAG.	(n1 -- )	Print the value n1 on the tag display.
GETCLK	( -- sec min hr)	Return the time as three values on the stack.
SETCLK	(hr min sec -- )	Set the time from the three values on the stack.
ADJCLK	(n1 -- )	Compensate for clock gains of n1 seconds per day
CLK.	( -- )	Prints the time as hh:mm:ss .
KEYCLK	( -- )	Waits for the user to enter the time, and echoes the characters entered.
GETDATE	( -- yr mon day)	Return the date as three values on the stack.
SETDATE	(day mon yr -- )	Set the date from the three values on stack.
DATE.	( -- )	Prints the date as dd-mm-yy .
KEYDATE	( -- )	Waits for the user to enter the date, and echoes the characters entered.

Table 4.14.6 Special Microsupervisor Words

Error condition flags from GETEXT.

Code	Meaning
-1	Communications time out.
0	Good data.
1	Instrument number not configured in data base.
2	Pseudo instrument parameter number error.
3	Pseudo instrument bad data base error.

Error condition flags from SETEXT.

Code	Meaning
-1	Communications time out.
0	Good data.
1	Instrument number not configured in data base.
2	Pseudo instrument parameter number error.
3	Real instrument invalid reply (eg NAK).

#### 4.15 Debugging Facilities

To assist in program debugging, a trace feature can be enabled either in immediate mode, or from within a program.

When the trace is enabled, each time a user word is completed the word name is printed on the terminal, with a list of all the data on the stack.

The stack items are printed (non destructively) with the top of stack item printed first, at the left of the display.

---

Operator	Stack notation	Description
TRA-ON	( -- )	Turn trace option on.
TRA-OFF	( -- )	Turn trace option off.

---

Table 4.15 Debug Words

#### 4.16 Reserved Words

Apart from the words in the fixed dictionary, there are some words reserved that define the program installed and executed at power up.

##### 4.16.1 The 6433 MAIN Program

When a 6433 is powered up, the program stored in the EEROM is loaded into RAM. The software in the instrument will then search the user dictionary for a word called MAIN. If this word exists, the program will be executed automatically.

The same effect can be obtained by executing the words RECALL and MAIN at the terminal during a programming session.

##### 4.16.2 Advanced Controller Programs

When an Advanced Controller is powered up, the program stored in the EEROM is loaded into RAM. The Advanced Controller can then load up to three programs. The names of these programs are selected in parameters L1, L2 and BG in the General Purpose Block.

The same effect can be obtained by executing the words RECALL and RUN at the terminal during a programming session.

##### 4.16.3 6445 Microsupervisor BGRND Program

The Microsupervisor is the same as the 6433 but the stored program is executed when the word called BGRND is found in the user dictionary.

##### 4.16.4 The ERROR Program

If a run time error occurs in the background program, the software in the instrument will stop the current background program, and search the user dictionary for a word called ERROR. If this word exists the program will be run automatically.

The programs are considered to be "running" when the background program enters a BEGIN ... REPEAT or BEGIN ... UNTIL structure.



#### 4.17 Error Messages

This section lists the error messages that could occur during a programming session and gives more details about the causes.

1. Matching pairs - Some words must be used in pairs, e.g. DO ... LOOP, IF ... ENDIF, : ... ; . This indicates one half of the pair is missing.
2. Compiler output buffer overflow - This indicates there is not enough memory to compile a word.  
To overcome this, split the word into smaller modules.
3. Terminal input buffer overflow - Indicates there are too many characters in a word.  
To overcome this, split the word into smaller modules.
4. Data stack overflow - Attempting to put too much data on the stack.
5. Data stack underflow - Attempting to remove data from an empty stack.
6. Attempting to edit a FIXED word - User tried to modify a word fixed in PROM.
7. I/O board type - Incorrect data base access routine used for an input/output board.
8. LOCAL VARIABLE argument out of range - Incorrect argument used for GETARG, SETARG, GETRES, SETRES.
9. TIMER argument out of range - Attempt to access a non-existing timer.
10. ARG/RES argument out of range - Incorrect argument for the ARG or RES words.
11. I/O Bn out of range - Illegal block or board number used.
12. I/O Cn out of range - Illegal channel number used.
13. Undefined or forward referenced word - Attempt to reference a non-existent word, or a word that is not deeper in the dictionary.
14. I/O Pn argument out of range - Illegal parameter number used.
15. User dictionary empty - No words in the user dictionary.
16. Memory corruption - The user program memory is corrupted, the program may have to be re-entered, or there is a memory fault.

17. VARIABLE argument out of range - Attempt to access a non-existent variable.
18. I/O Write protected - Attempting to write to protected parameter in the data base.
19. I/O Board hardware - Attempt to access a faulty board.
20. User memory not available - Insufficient space in the dictionary for the current word.
21. Missing ARG statement - No ARG used in the current word.
22. Illegal word use - The word is used incorrectly.
23. Task argument out of range - Incorrect argument for time scheduled program.

## Section 5

### Programming Terminal Utilities

This section provides technical details of the serial interface from the front panel of the instrument. It is intended for people who want to produce a programming terminal using their own personal computer.

If you are using a computer with a TCS application VDU program, then the procedure for saving and loading a program is explained in the appropriate manual.

#### 5.1 Front Panel Electrical Specification

The port on the front panel provides a standard serial link:

Transmission standard RS232C/V24 (+12 Volts)

Character length        1 start, 7 data, 1 parity (even) + 1 stop bit  
                          (2 stop bits at 110 baud)

#### 5.2 Normal Operating Mode

A VDU can be used in place of the normal hand-held terminal to inspect and modify the parameters in the data base. When using a VDU in this mode the following keys are used to move the cursor or load the data:

L	Enter a positive or hexadecimal number
M	Enter a negative number
Q	Backspace the cursor
W	Scroll to the next parameter
Z	Reset display to the ?? CMD prompt

The instrument transmits standard ASCII printing and control characters to the VDU. In addition the following control characters have a special function:

TAB    This character is used to move the cursor forward 1 position.

US LF These two characters are used to reset the display.

When a user logs on, the instrument behaves in the normal way, echoing and transmitting characters to update the display on the screen.

### 5.3 Saving a User Program

This mode allows a user to save a program from an instrument as a text file. This text can be archived and loaded into the computer at a later date.

The program is transmitted one word at a time, and the text transmitted contains additional space, carriage return and line feed characters to provide some structure to the program.

In the save mode the instruments support the XON-XOFF protocol.

To save a program, the user must be logged on to the instrument.

The save is initiated by sending an ENQ character to the instrument.

The instrument will then send one word, preceded by an STX character and terminated by an ETB character.

The instrument will then wait for another ENQ character before sending the next word. If the instrument receives a character other than ENQ then it automatically exits the save mode.

To indicate when the complete program has been sent, the final word transmitted from the instrument is terminated by an ETX character (not an ETB).

#### 5.4 Loading a User Program

This facility allows a very flexible format for the text files containing the program.

When the file is loaded, control characters such as carriage return, tab and line feed are treated as spaces; this means that the text file can be structured to make a program more comprehensible. Comments can also be included in a file provided they are enclosed in ordinary brackets. As the instrument receives the file, it discards redundant characters and comments so that they do not waste space in memory.

To load a program, the user must have logged on to the instrument, and deleted unwanted words. Programs can be appended to existing words in the user dictionary.

The computer transmitting the program must support the XON-XOFF protocol.

The load is initiated when the instrument receives a STX character. This switches the instrument to load mode so that characters received are not echoed back to the computer.

The text file of the program can then be transmitted.

The text should be followed by the ETX character to switch the instrument out of load mode.

As the program is loaded, it is checked for errors, and compiled into the user dictionary. If any errors are found, the instrument exits the load mode, and transmits an error message.

# INDEX OF WORDS DESCRIBED IN THE MANUAL

<u>Word</u>	<u>Section</u>
"	4. 4.3
#	4. 4.3
\$	4. 4.3
%GET	4.14.4
%SET	4.14.4
%REMOTE	4.14.4
*	4. 3.3
+	4. 3.3
+LOOP	4.13.3
+TIM	4.10.3
-	4. 4.3
.	4. 4.3
."	4. 4.3
/	4. 3.3
0<	4. 9
0=	4. 9
0>	4. 9
:	4. 5.4
;	4. 5.4
<	4. 9
=	4. 9
>	4. 9
?DUP	4. 6.2
?F	4.14.6
ABS	4. 7
ADJCLK	4.14.6
ALARM	4.14.4
AND	4. 8.2
ARG	4. 6.2
ATAN	4. 7
ATAN2	4. 7
AWORDS	4. 5.4
BEGIN	4.13.3
BS	4. 4.3
CASE	4.12.3
CLEAN	4. 5.4
CLK.	4.14.6
COS	4. 7
CR	4. 4.3
DATE	4.14.6
DIFVAR	4.10.3
DO	4.13.3
DROP	4. 6.2
DUP	4. 6.2
E	4. 7
ELSE	4.12.3
ELSOF	4.12.3
EMIT	4. 4.3
ENDCASE	4.12.3
ENDIF	4.12.3
ENDOF	4.12.3
EXP	4. 7

FILTER	4.14.4	
FORGET	4. 5.4	
FWORDS	4. 5.4	
GET	4.14.4	
GETAN	4.14.2	
GETARG	4. 6.2	
GETBCN	4.14.2	
GETCLK	4.14.6	
GETDATE	4.14.6	
GETDEL	4.14.4	
GETDI	4.14.6	
GETDIG	4.14.2	4.14.4
GETDS	4.14.6	
GETEXT	4.14.6	
GETPAR	4.14.2	
GETREP	4.14.4	
GETRES	4. 6.2	
GETTIM	4.10.3	
GETVAR	4.10.3	
HALT	4.14.4	
I	4.13.3	
IF	4.12.3	
INSERT	4. 5.4	
INT	4. 7	
KEY	4.11	
KEYCLK	4.14.6	
KEYDATE	4.14.6	
LEAVE	4.13.3	
LF	4. 4.3	
LFMT.	4.14.6	
LINE	4.14.6	
LN	4. 7	
LOOP	4.13.3	
MAX	4. 7	
MID	4. 7	
MIN	4. 7	
MINUS	4. 7	
MSCONT	4.14.4	
NEW	4. 5.4	
NL	4. 4.3	
NOT	4. 8.2	
NUMBER	4.11	
OF	4.12.3	
OR	4. 8.2	
OVER	4. 6.2	
PI	4. 7	
PICK	4. 6.2	
PID	4.14.4	
PIDX	4.14.4	
RATIO	4.14.4	
RECALL	4. 5.4	
REMOTE	4.14.4	
REPEAT	4.13.3	
RES	4. 6.2	
RFMT.	4.14.6	
ROLL	4. 6.2	

ROT	4. 6.2	
RUN	4.14.4	
SET	4.14.4	
SETAN	4.14.2	
SETARG	4. 6.2	
SETBCN	4.14.2	
SETCLK	4.14.6	
SETDATE	4.14.6	
SETDEL	4.14.4	
SETDIG	4.14.2	4.14.4
SETDO	4.14.6	
SETDS	4.14.6	
SETEXT	4.14.6	
SETLN	4.14.4	
SETPAR	4.14.2	
SETREP	4.14.4	
SETRES	4. 6.2	
SETTIM	4.10.3	
SETVAR	4.10.3	
SIN	4. 7	
SPACE	4. 4.3	
SQR	4. 7	
SQRT	4. 7	
STORE	4. 5.4	
SUMVAR	4.10.3	
SWAP	4. 6.2	
TAG.	4.14.6	
TAG."	4.14.2	4.14.6
TOTAL	4.14.4	
TRA-OF	4.15	
TRA-ON	4.15	
ULIST	4. 5.4	
UNTIL	4.13.3	
UWORDS	4. 5.4	
WINDOW	4. 4.3	
XOR	4. 8.2	



APPENDIX 1

8261 Data Base Configurator

Interconnection Cables



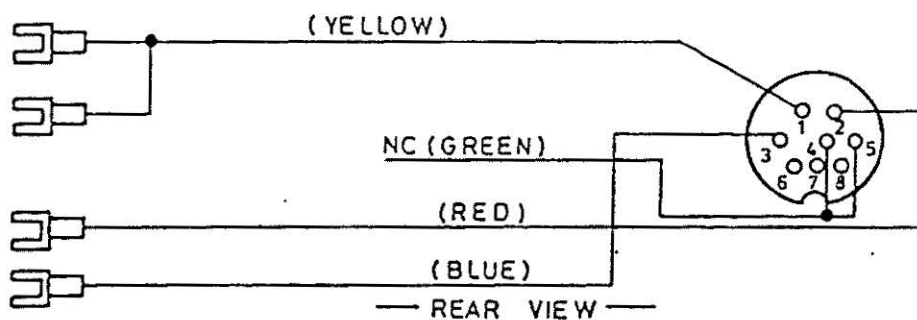
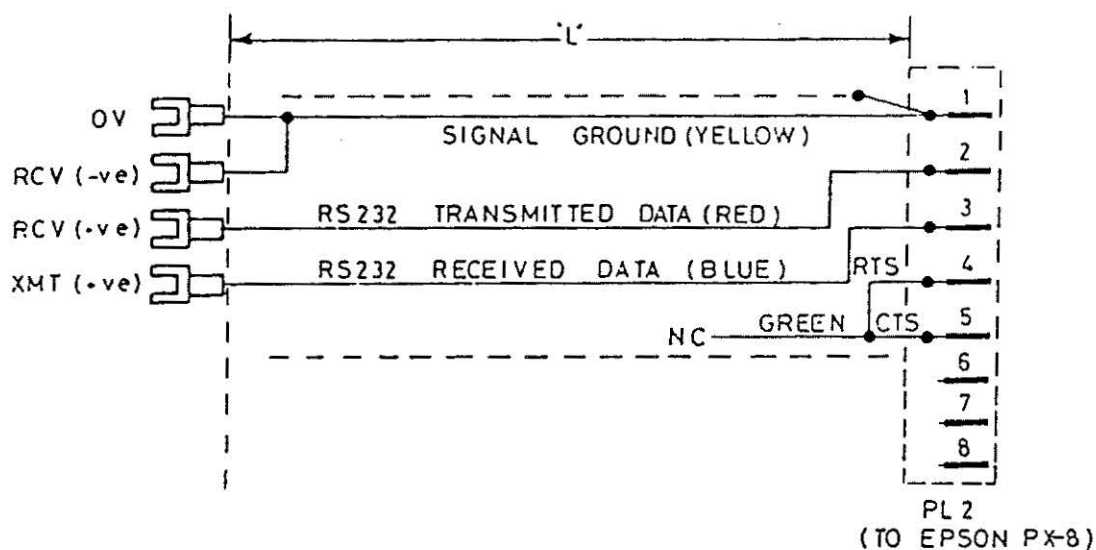
## GENERAL DRAWING PRACTICE TO BS 308/BS 3939

S<sub>S</sub> DATE

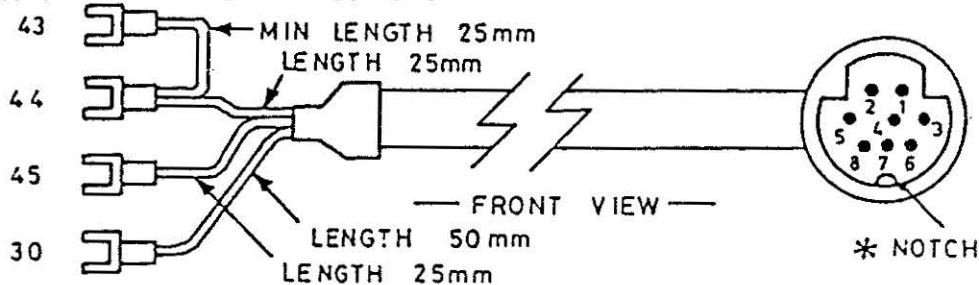
DO NOT SCALE

THIRD ANGLE PROJECTION

1 25/10/85



## 7600 BIN TERMINAL CONNECTIONS



NOTE: DIMENSION 'L' DENOTED BY SUFFIX NUMBER WHICH IS INCREMENTED BY 001 FOR EACH METRE LENGTH. DEFAULT LENGTH IS 3 METRES.

\* 'ARROW' ON OUTER PLASTIC SHROUD MUST BE IN LINE WITH NOTCH.

DRAWN JF/JM

MATERIAL

SCALE

DIMS. IN M.M. APPLY OVER FINISH (EXCEPT FOR PAINT AND LACQUER)

CHECKED AS

NTS

GENERAL X - ± 0.4  
TOLERANCE XX - ± 0.2 HOLES < Ø 7mm  
XXX - ± 0.1 - 0.02 ± 0.07

DESIGN APPROVAL

FINISH

ASSEMBLED ON

TITLE

MANF. APPROVAL

MPB

8271

INTERFACE CABLE, EPSON PX-8  
TO INSTRUMENT RS422 PORT

TCS

TURNBULL  
CONTROL  
SYSTEMS LTD



DRAWING NUMBER

SHT 1

LA 076706 C\*\*\*

OF 1 SHTS

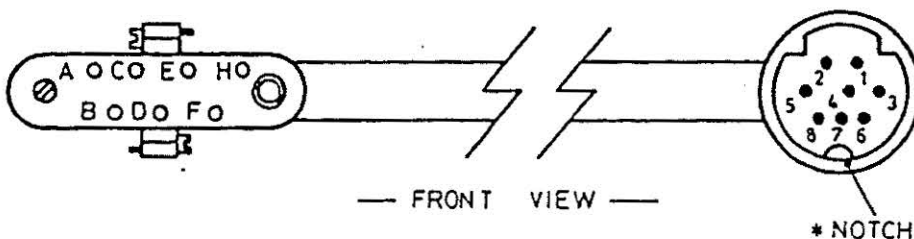
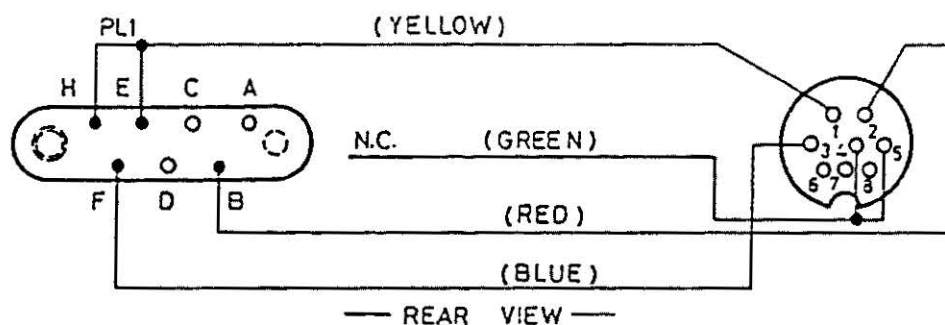
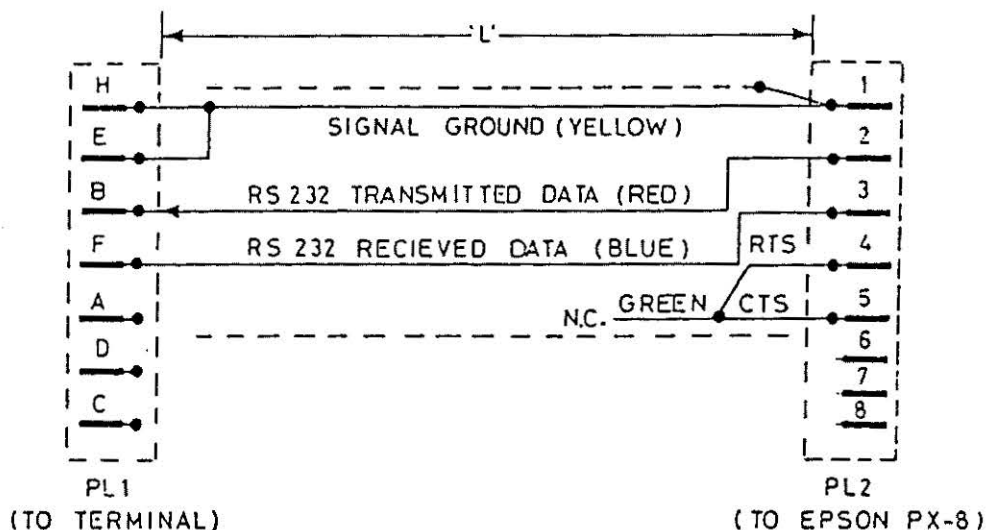
## GENERAL DRAWING PRACTICE TO BS 308/BS 3939

SS DATE

DO NOT SCALE

THIRD ANGLE PROJECTION

1 25.9.85



## NOTE:

DIMENSION 'L' DENOTED BY SUFFIX NUMBER WHICH IS INCREMENTED BY 001 FOR EACH METRE LENGTH.

\* 'ARROW' ON OUTER PLASTIC SHROUD MUST BE IN LINE WITH NOTCH. DEFAULT LENGTH IS 3 METRES.

DRAWN JF/JM	MATERIAL	SCALE N.T.S.	DIMS. IN M.M. APPLY OVER FINISH (EXCEPT FOR PAINT AND LACQUER)	
CHECKED as			$X - \pm 0.4$ GENERAL $X.X - \pm 0.2$ HOLES $< \varnothing 7mm$ TOLERANCE $X.XX - \pm 0.1$ $-0.02 + 0.07$	
DESIGN APPROVAL <i>U/S</i>	FINISH	ASSEMBLED ON	TITLE	
MANF. APPROVAL MAB			INTERFACE CABLE, EPSON PX-8 TO H.T. INPUT SOCKET	
TURNBULL CONTROL SYSTEMS LTD		EI	DRAWING NUMBER LA 076644 C / ***	SHT 1 OF 1 SHTS

## APPENDIX 2

### 6433 Instrument Parameters

II, SW, MD and those instrument parameters relating to the Input/Output Blocks can be accessed from each of the board types.

For real blocks 1 to 4 parameter numbers 1 to 8 refer to S1 to A4 inclusive.

For pseudo blocks 5 to 8 parameter numbers 1 to 8 refer to S5 to A8 inclusive.

The following Tables list the parameter types and numbers.



		0	1	2	3	4	5	6	7	
	0	II	S1 S5	A1 A5	S2 S6	A2 A6	S3 S7	A3 A7	S4 S8	(4)
	8	A4 A8	SW	MD						(4)
CHAN 1	16	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 2	24	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 3	32	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 4	40	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 5	48	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 6	56	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 7	64	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 8	72	ST*	HR*	LR*	HA*	LA*	PV*	AR*		
CHAN 1-2	80	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 3-4	88	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 5-6	96	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 7-8	104	T1	T2	T3	T4	T1	T2	T3	T4	

List of 6433 Parameter Numbers, [PNO]s,  
and their respective mnemonics for  
pseudo-Analogue Input Boards

TABLE 1

		0	1	2	3	4	5	6	7	
	0	II	S1 S5	A1 A5	S2 S6	A2 A6	S3 S7	A3 A7	S4 S8	(4)
	8	A4 A8	SW	MD						(4)
CHAN 1	16	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 2	24	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 3	32	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 4	40	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 5	48	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 6	56	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 7	64	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 8	72	ST*	HR*	LR*	OP*	HO*	LO*			
CHAN 1-2	80	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 3-4	88	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 5-6	96	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 7-8	104	T1	T2	T3	T4	T1	T2	T3	T4	

List of 6433 Parameter Numbers, [PNO]s,  
and their respective mnemonics for real  
and pseudo-Analogue Output Boards

TABLE 2



		0	1	2	3	4	5	6	7	
CHAN 1-8	0	II	S1 S5	A1 A5	S2 S6	A2 A6	S3 S7	A3 A7	S4 S8	(4)
	8	A4 A8	SW	MD						(4)
	16	ST*	AM*	DS*						
	24									
	32									
	40									
	48									
	56									
	64									
	72									
CHAN 1-2	80	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 3-4	88	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 5-6	96	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 7-8	104	T1	T2	T3	T4	T1	T2	T3	T4	

List of 6433 Parameter Numbers, [PNO]s,  
and their respective mnemonics for real  
and pseudo-Digital Input Boards

TABLE 3

		0	1	2	3	4	5	6	7	
CHAN 1-8	0	II	S1 S5	A1 A5	S2 S6	A2 A6	S3 S7	A3 A7	S4 S8	(4)
	8	A4 A8	SW	MD						(4)
	16	ST*	AM*	DS*						
	24									
	32									
	40									
	48									
	56									
	64									
	72									
CHAN 1-2	80	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 3-4	88	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 5-6	96	T1	T2	T3	T4	T1	T2	T3	T4	
CHAN 7-8	104	T1	T2	T3	T4	T1	T2	T3	T4	

List of 6433 Parameter Numbers, [PNO]s,  
and their respective mnemonics for real  
and pseudo-Digital Output Boards

TABLE 4



