



HUSKY

Operations Manual

Operation

BASIC Programming

Advanced Programming

Communications

Technical Data

CONTENTS

HUSKY OPERATIONS MANUAL

CONTENTS

PART 1	INTRODUCTION
PART 2	HUSKY OPERATION
PART 3	HUSKY BASIC PROGRAMMING
PART 4	HUSKY ADVANCED PROGRAMMING
PART 5	HUSKY COMMUNICATIONS
PART 6	TECHNICAL DATA & ACCESSORIES

HUSKY COMPUTERS LIMITED
P O BOX 135
345 FOLESHILL ROAD
COVENTRY, WEST MIDLANDS
ENGLAND CV6 5RW

USING THE MANUAL

1 INTRODUCTION

PLEASE READ THIS SECTION

1.1 Purpose

This manual is intended to fulfil several functions:

- 1) As an introduction to the programming, operation and use of **HUSKY**.
- 2) As a source of reference for technical data about **HUSKY**.
- 3) As the vehicle for achieving **HUSKY's** full potential in diverse user applications.

It is **not** intended as a guide to first time computer or Basic users. For guidance on these subjects, the reader is referred to the many excellent introductory works dealing with Microcomputer Basic, now commonly available.

1.2 How to Use This Manual

The manual is split into a number of **parts**. Part 1 is this introduction. The others are :

- 2) **Operation:**
How to operate **HUSKY**. The location and use of its screen, keyboard and power supplies.
- 3) **BASIC Programming**
A full exposition of **HUSKY's** resident BASIC interpreter, including programming data, performance details and sample programs.

A comprehensive index to BASIC functions is provided in this section. Start here if your interest is in Basic compatibility.

- 4) **Advanced Programming:**
Data specific to **HUSKY** including details of **HUSKY's** architecture and housekeeping programs. Details of CP/M compatibility. Interfacing for direct data acquisition.
- 5) **Communication:**
A full description of **HUSKY's** very flexible serial data communication facilities.
- 6) **Technical Data & Accessories:**
Information on maintenance and installation of **HUSKY**.
: Technical Specification : Details of Accessories.

Each Part has an index page, detailing subsections, immediately following the coloured divider.

1.3

Page Format

Every page in this Manual is laid out on a standard format. Sections are identified in the left Margin by section numbers, e.g. 1.3, meaning Part 1 (Introduction), Section 3.

At top left of the page is a reminder of which part is being studied, i.e. "Husky Operations Manual Introduction".

At top right, in bold type, is the subject of this page. (Not all pages have this bold type). On this page it is "PAGE FORMAT".

Just above this is the number of the first section on the page, e.g. "Section 1.3"

At bottom left is the revision of Husky Operating system to which this Manual relates. Every effort is made to keep Husky versions "upward compatible", but sometimes, very rarely, features in the operating systems undergo slight changes for improved performance or compatibility reasons.

The version of Operating system in your Husky can be found by typing "VER" (Section 3.3.2.31).

At bottom right is the page number of this page, referenced to the start of this part. New material is sometimes inserted into the page sequence, and is then labelled with page number followed by suffixes -1, -2, -3 etc.

PART 2

HUSKY OPERATION

- 2.1 INTRODUCTION TO HUSKY
- 2.2 OWNER'S INFORMATION
- 2.3 BATTERIES, POWER WARNING
- 2.4 COMMUNICATION
- 2.5 SETTING THE CLOCK

2.1

WELCOME

Welcome to **HUSKY**!

HUSKY is probably the friendliest computer you've yet seen. It's small, light very powerful and totally dependable.

This is the first true computer you can take almost anywhere, use almost anywhere and program yourself. It's resistant to moisture, dust, vibration, shock and electro-magnetic interference. But please don't abuse it. Inside the impervious cast aluminium case is some of the most advanced technology money can buy. Don't drop **HUSKY** needlessly, use it to strike other objects or pile heavy things on top of it.

If you can't find the information you need in this manual or have problems with **HUSKY**, please call us. We've done all we can to make **HUSKY** a practical and utilitarian tool, not a frustrating incumbrance. We hope you agree. After all, **HUSKY** is our baby.

The **HUSKY** team.

HUSKY COMPUTERS LIMITED
PO BOX 135
345 FOLESHILL ROAD
COVENTRY, CV6 5RW
ENGLAND

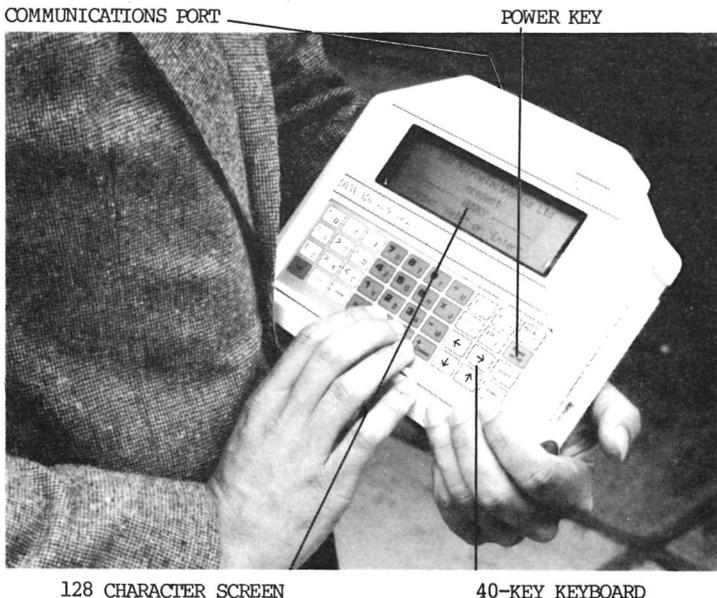
TEL: (0203) 668181

TELEX: 317450 HUSKY G

2.2 OWNER'S INFORMATION

- 2.2.1 **Layout of HUSKY**
Measuring 24.1 x 20.3 x 4.4cm (9.5" x 8" x 1.75") and weighing about 2 Kg. (4.4 lbs), **HUSKY** is a completely self contained computer system.

FIG 2.1 HUSKY LAYOUT



HUSKY is completely sealed against moisture, dust and other hazards. **NEVER ATTEMPT TO OPEN THE CASE.**

There are no user-serviceable parts inside.

When handed over to you, **HUSKY** is likely to be already **programmed** for the task you have in mind. This programming will cause **HUSKY** to ask questions, supply data and otherwise assist you.

There is likely to be a specific manual for the applications program loaded in your **HUSKY**, to which you refer. Once loaded,

SCREEN

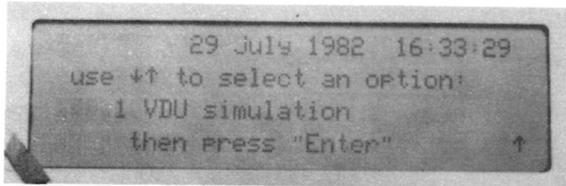
applications programs remain installed in **HUSKY** until the unit is required for another purpose.

2.2.2

HUSKY Screen

HUSKY has one of the largest LCD (Liquid Crystal Display) screens yet developed. It can present up to 128 characters on 4 lines of 32 characters each, just like a small video terminal.

FIG.2.2



The character set includes upper and lower case alphabets. Additionally, special characters and graphic symbols are used in specific applications.

The **HUSKY** display is designed for use in bright sunlight where other displays (the red 'LED' or green 'vacuum flourescent', for example) become invisible.

Because it works by contrast, rather than emitting its own light, it is perfectly visible no matter how bright the light is. But a word of caution: try to avoid leaving **HUSKY** exposed to direct sunlight for prolonged periods - the delicate chemicals in the LCD can be damaged.

The 'Cursor' shows where data entered on the keyboard will appear: The 'shift arrow' shows whether the key entries are shifted or not.

The screen is protected by a polycarbonate window, as strong as glass. Like glass, it can be scratched, so please take care. The window should only be cleaned with chamois or lens cleaning cloth.

HUSKY is built to last - please take care of it.

KEYBOARD

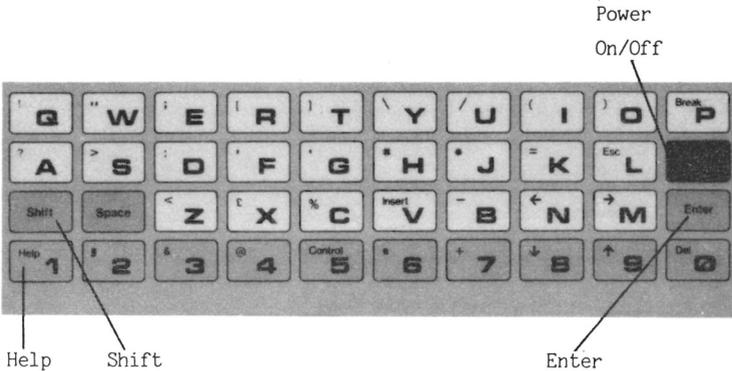
2.2.3

HUSKY Keyboard

HUSKY's keyboard has 40 keys, laid out in a typewriter style 'QWERTY' format.

HUSKY's keyboard looks like this: (you may see other versions, too. This is the standard layout)

FIG.2.3



NOTICE THAT HUSKY'S KEYS REPEAT AUTOMATICALLY WHEN YOU HOLD THEM DOWN.

There are four keys you should get to know straight away.

2.2.3.1

POWER ON/OFF

Press once to turn HUSKY on : again to turn off. Press firmly and hold the key down until HUSKY responds.

2.2.3.2

SHIFT

Press the 'Shift' key and watch the 'Shift Arrow' in the bottom right-hand corner of the screen. ↑ means shift 'UP', while ↓ means shift 'DOWN'. There are two types of shift key in HUSKY:

'Momentary Shift' (for two handed use) and 'Latched Shift' (for use with one hand only).

Momentary shift works just like a typewriter shift key, while with latched shift each time you press 'Shift', the arrow will

change, remaining where it is until you shift again.

'Shift' never affects the operation of **HUSKY** : you can change 'Shift' whenever you like.

2.2.3.3 ENTER

The 'Enter' key accepts entries from the keyboard and terminates lines. If **HUSKY** asks for a keyboard entry, always terminate your message with 'Enter'.

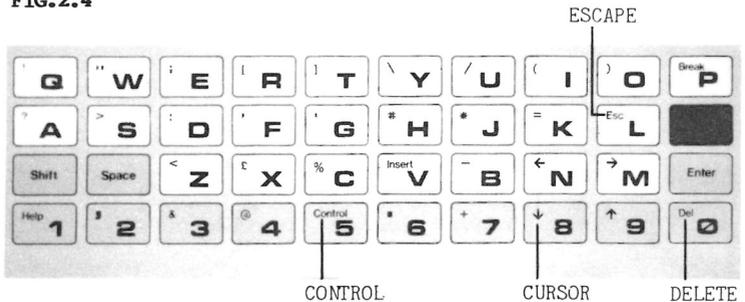
2.2.3.4 HELP

Whatever you are doing, 'Help' is always available. Pressing 'Help' doesn't destroy your program or affect entries you are making. 'Help' will display pages of instruction relevant to the operation you are carrying out: you can scan through 'Help' text using the cursor keys.

To leave 'Help', press 'Enter'. The screen will return to wherever you were before, unchanged. Note that 'Help' is a 'shifted' key : you will probably need to press shift first.

Having mastered 'Power', 'Help', 'Enter' and 'Shift', you will need to know about a second group of keys. These are:

FIG.2.4



- 2.2.3.5 Most important are the 'Cursor' keys, ← → ↑ ↓. These move the cursor (see the display section 2.2) in the direction of the arrows. They have two main uses:

Menu selection : a 'menu' is a range of choices presented by **HUSKY** for selection by the user. The choices are selected by pressing cursor keys until the desired option appears. The option is then confirmed by pressing the 'ENTER' key.

Text scrolling : use the cursor keys to step sequentially through lines of text, as in '**HELP**'. Remember that ↑ moves the

cursor **DOWN** to the **NEXT** line, i.e. the text moves **UP** the screen, **↓** moves **BACK** one line, the text scrolling **DOWN** the screen.

NOTE: Many industry standard programs, like **HUSKY**'s **BASIC**, don't recognise the cursor keys and can be confused by them. Use 'Delete' and 'Enter' instead.

The cursor keys never affect a program directly.

2.2.3.6 **DELETE**

'Delete' is used with line-by-line programs like **BASIC**, and is the equivalent of ←. Pressing Delete steps the cursor back one character and erases it, ready for a new character to be entered.

You can erase a whole line, if you want. When you get to the beginning of the line, 'Delete' will give a warning tone to tell you.

2.2.3.7 **CONTROL**

Only one key on **HUSKY** does not return a confirmation tone when pressed: 'Control'. 'Control' is a shifted key.

You can enter control characters by first pressing 'Control', and then the appropriate key without releasing 'Control'. Try control - G (Bell). Press and hold 'Control', then 'G'. **HUSKY** will give a long bleep.

As the control function is on upper shift of the standard Husky keyboard, to obtain control codes from the **HUSKY** keyboard, three simultaneous key operations would normally be required. The shift key, the control key and the key for which the code is required.

In order to simplify the number of key operations to two, the following procedure should be followed:

Example to obtain Control A

Operate the shift key and then select the control key simultaneously. Once the control key has been operated the shift key can be released as the control function has been latched, provided the control key remains operated. The key 'A' can then be pressed to generate the control A code.

2.2.3.8 **ESC**

'Escape' is an upper-shifted key used by industry standard programs to terminate execution. It is only available when running certain programs, and otherwise has no effect.

2.2.4

PHYSICAL

HUSKY's keyboard is built for extended use and long life. The contact elements (deformable metal discs that 'click' when operated) are protected by several membranes, including an outer polycarbonate overlay that is colour printed with the keyboard lettering and can be replaced if needed. The keyboard will withstand water, coffee and other assaults; but it can be damaged by direct impact or sharp objects like nails.

DON'T USE POINTED OBJECTS TO OPERATE THE KEYS

2.2.5

THE BELL

HUSKY has an audio transducer that produces a loud tone or a sharp click when keys are operated, or when required by a user program.

The sound level is maximised for use outdoors: indoors, it can be a nuisance. To suppress the bell, type control 'Delete' (See 2.2.3.6). This will suppress further sounds.

2.3

BATTERIES

HUSKY is battery powered. It runs, for a long time, on standard 'C' size cells you can buy anywhere.

HUSKY has two separate batteries:

FIG.2.5



The MAIN Battery

The LEMO
(charger) socket

The STAND-BY
Battery

The MAIN battery is made up of four 'C' cells.

The STAND-BY battery is a single mercury cell.

2.3.1 BATTERY INSTALLATION

Both battery holders are spring-loaded to ensure reliable contact. The batteries are retained by threaded plugs.

When installing batteries, follow this procedure:-

- 1 ONLY USE A COIN in the battery plug. Screwdrivers, etc., will damage the slot and HUSKY's appearance.
- 2 Insert the cells, positive terminal inwards. Don't drop the 'C' cells in vertically : they can be damaged. Instead, tilt HUSKY slightly.
- 3 Take the battery plug, and with **finger pressure only**, press into the battery compartment and turn.
- 4 Only when the thread is **started**, use a coin to screw the plug home.

The MAIN BATTERY plug should be flush with the case wall.

The STAND-BY battery plug stands proud by about 2mm.

NOTE:

Clockwise: inserts the plug

Anti-clockwise: removes the plug

2.3.2 BATTERY TYPE

2.3.2.1 Main Battery - Primary Cells

We strongly recommend the use of Alkaline-Manganese batteries similar to Mallory MN1400 or Ever Ready (Berec) MN1400LR14.

We do not recommend the use of cheaper zinc-carbon cells for routine operation, since these can suffer electrolyte leakage, have shorter lives and can introduce technical problems.

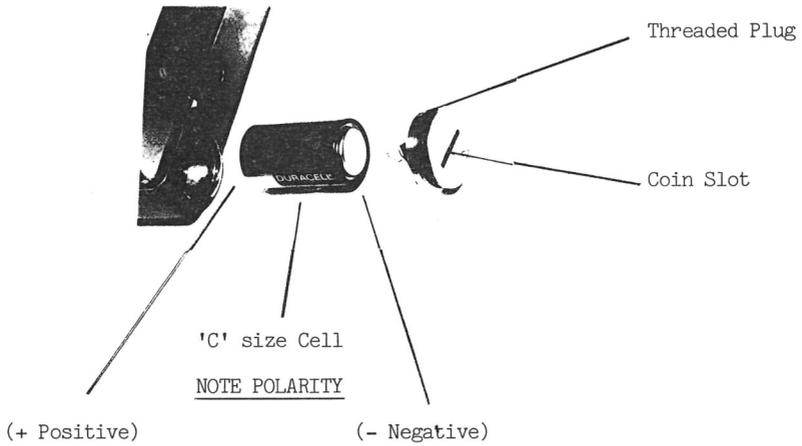
2.3.2.2 Main Battery - Rechargeable Cells

Only Nickel-Cadmium (Ni-Cd) cells are to be used. Under no circumstances must any other type of rechargeable cell be installed. We recommend Berec type NCC200 cells of 2000 mAh capacity.

2.3.2.3 Stand-by Battery

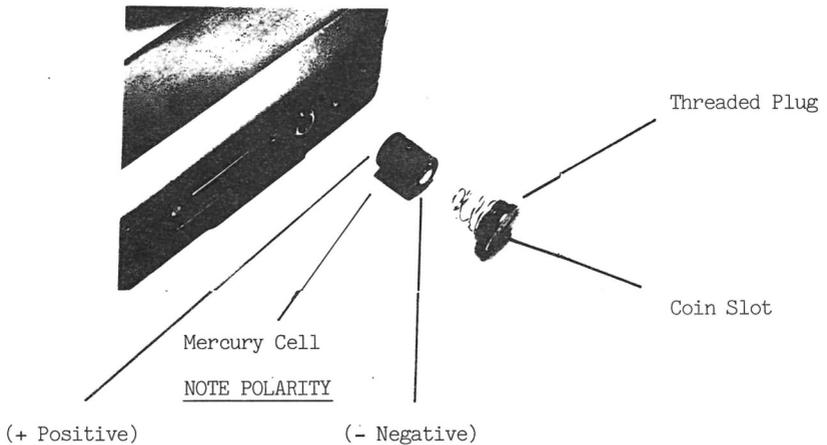
Use only a single Mallory mercury cell type PX23. It is recommended that this cell is replaced annually.

FIG. 2.6
MAIN BATTERY



Both batteries are secured by screw-in plugs and 'O' ring seals.

STAND-BY BATTERY



Alkaline cells will give 30 - 50 hours of **HUSKY** use, and almost very long storage life. There is no need to remove cells from **HUSKY** during storage or shipment but remember that capacity will reduce steadily with time.

2.3.3 **LOW BATTERY WARNING**

When **HUSKY's** batteries are nearly exhausted, a power warning message will appear on the screen. Normal operation will continue, but keyboard entries will be punctuated by warning tones and repetition of the power warning message until the battery state is rectified either by replacement of primary cells or recharging of secondary cells, if installed.

Eventually, if the warnings are ignored, the **HUSKY** will shut down and refuse to operate further.

The warning message appears on the top line of the screen and looks like this:

WARNING - BATTERIES ARE LOW

If batteries become exhausted whilst communications are in progress, then normal power warning messages will appear once the keyboard operation is resumed.

NOTE: Power warning messages only occur when keyboard entries are in progress.

2.3.4 **BATTERY CHARGING**

HUSKY is optionally available with rechargeable cells and a mains (line) powered charger. This arrangement allows cells to be recharged in the **HUSKY** and for alkaline cells to be quickly substituted if the user forgets to recharge!

But **BE CAREFUL - NEVER** connect the charger when alkaline batteries are installed. They will **NOT** re-charge, instead, they may explode, jam in the battery tube or leak corrosive chemicals.

HUSKY's charger is in the form of a moulded plug-top and incorporates an integral double insulated transformer for safety. The charger connects to **HUSKY** via the LEMO connector and simply plugs into a suitable electrical outlet. Chargers are available in different forms to suite national variations - be sure that the charger supplied with your **HUSKY** is suitable for the voltage and standards at your location.

The recommended 2000mAH Nickel-Cadmium cells cannot be over-charged. The cells will fully re-charge in about 12 hours if the **HUSKY** is not in use.

2.3.5**CONTINUOUS CONNECTION**

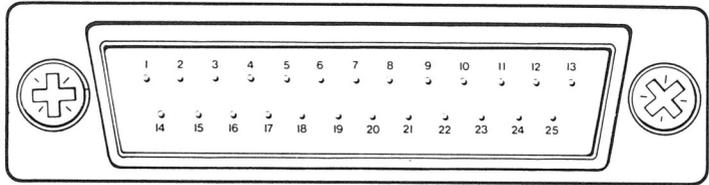
The **HUSKY** can be powered permanently from the charger by simply leaving the unit connected. In this mode, the rechargeable cells will be kept 'topped up' by the charger despite the continuous current drain. Of course, a fully discharged battery will take longer to recharge than if the **HUSKY** is powered down.

2.4

COMMUNICATION

HUSKY has an industry-standard RS-232/V24 serial communications port, and will talk directly to most computer peripherals, modems and other devices.

FIG 2.8



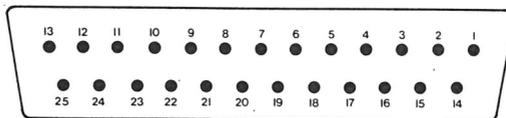
THE SERIAL PORT

The port communicates via a standard male 25-way 'D' type connector. The pin connections and other data are detailed in Part 5 'COMMUNICATIONS'.

NOTE: Never, never try to force a mating connector home. Check that the connector is the right way up (like this):

FIG 2.9

TOP



and is a **female** type

If the connector won't fit easily, it probably won't fit at all
FIND OUT WHY!

HUSKY's communication format is usually set up by its applications program for specific situations, and does not require operator attention.

Sometimes, a user program will present a 'menu' selection for user choice. In this event, selection of the desired communications mode will automatically set the communications parameters without further action.

Programming the user applications program in this fashion is described in Part 4, 'ADVANCED PROGRAMMING', Section 4.2.3.18.

Communications parameters can also be set manually. An internal program, 'COMMUNICATIONS PARAMETERS', can be accessed or called by a user application program. See PART 5, 'COMMUNICATIONS', for details.

2.5

CLOCK

HUSKY has a built-in calendar clock, keeping track of years, months, days, hours, minutes and seconds.

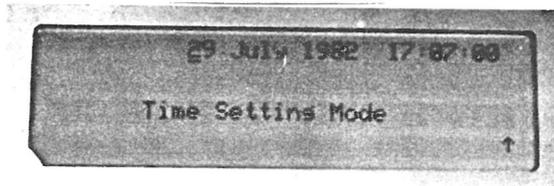
2.5.1

Occasionally, **HUSKY's** clock may need adjustment.

Most user applications programs provide access to one of **HUSKY's** internal programmes, 'INITIALISE CLOCK' mode.

HUSKY displays:

FIG.2.10



- a) Using the `<` (right arrow) and `>` (left arrow) keys, step the cursor to the item you wish to set. The seconds cannot be set and are always initialised to zero when synchronising the clock (see below).
- b) Using the `↑` (up arrow) and `↓` (down arrow) keys, step the value selected until it is correct. The `↑` key increments the selected value and loops round to the associated minimum value. The `↓` key decrements the selected value and loops round to the associated maximum value.

The months step in the following manner:-

Nov,Dec,Jan etc. ; `↓` key

Jan,Dec,Nov etc. ; `↑` key

- c) Set the 'minutes' to one minute ahead of the present time.
- d) Press 'Enter'. **HUSKY** will display:
'Press ENTER to synchronise'

- e) When the time code, pip tone, or other reference arrives press ENTER.

HUSKY's clock will run from zero seconds in exact synchronisation with the external reference.

- f) The clock automatically accounts for leap years.
- g) If an attempt is made to specify an incorrect day value, e.g. 31st April, HUSKY will display the warning:

Error - Incorrect day value

The error must be corrected before synchronising the clock.

2.5.2

ADVANCED USE

HUSKY's clock can be used to time-stamp entries, automatically label printouts or even initiate program execution automatically, like an alarm clock.

See Part 4, 'ADVANCED PROGRAMMING', Section 4.2.3.20, for details.

BASIC PROGRAMMING

PART 3

HUSKY BASIC PROGRAMMING

- 3.1 INTRODUCTION
- 3.2 **HUSKY MEMORY**
- 3.3 EXPRESSIONS AND OPERATORS
- 3.4 FUNCTIONS
- 3.5 ERROR CODES
- 3.6 POWER WARNING
- 3.7 MACHINE CODE CALLS
- 3.8 PROGRAMMING TECHNIQUES
- 3.9 OFFLINE PROGRAM STORAGE

APPENDICES

- APPENDIX I ASCII CHARACTER SET
- APPENDIX II CONTROL CODES

INTRODUCTION

3.1

3.1.1 Introduction

HUSKY BASIC is a powerful and flexible BASIC interpreter installed within **HUSKY**'s firmware, allowing user's programs to be easily written and used without the need for any additional equipment. In addition to normal programming features, **HUSKY BASIC** can also handle communications with other devices.

Because it is designed for use in the portable environment, **HUSKY BASIC** has some unique features not found in other Basics. Principal amongst these is the ability of **HUSKY BASIC** to keep both programs and data after its power has been switched off. Data can be used later by the same or another program following an intervening period when **HUSKY** is powered down and not in use at all. Similarly, user's programs written in **HUSKY BASIC** are retained and are always available for further use until deliberately cleared.

Different programs can be stored simultaneously and executed independently by simple 'GOTO' statements.

Variable definitions are common to all programs concurrently resident in **HUSKY** and can be swapped between application software without the need for global declaration.

3.1.2 Initiating **HUSKY BASIC**

HUSKY BASIC is accessed from **HUSKY**'s main menu by stepping to the 'BASIC interpreter' option using the scroll keys | | and then by pressing 'Enter'.
HUSKY will then prompt:

```
HUSKY BASIC Interpreter  
READY
```

If **HUSKY** is already loaded with an application program, 'main menu' may not appear. Instead, the application program may execute immediately. This simply means that the 'auto-start' flag (see 3.1.4) is set.

3.1.3 Entering a Program

Programs are entered into **HUSKY**'s user memory by first typing a line number and then the program line required. Unnumbered lines will execute immediately. Variables can always be inspected or changed using 'Print' or 'Equal' statements. Effects of functions and other operators can be determined empirically by simply typing unnumbered lines and observing the result.

Certain functions are not allowed to execute directly e.g. GOSUB.

AUTO POWER FEATURE

3.1.4

AUTO START - BASIC

Once a user program is written and fully debugged it may be felt desirable for HUSKY to 'power up' directly into the user program rather than going through the procedure of entering the Basic Interpreter and typing RUN.

A flag has been provided to inform the start up software to enter BASIC directly. The flag is STARTIF at location 17832 (Decimal) which should have its value POKED to 170 (Decimal). See Section 4.1.4

It has been made more difficult to ESCape from Basic under these conditions, for program security and so as not to confuse the operator.

To 'escape' from the program (back to the Basic Interpreter), type ESC followed by a 5-digit escape code.

The default code is: 56580. However, this may be changed by POKeIng a new number into ESCCODE located at 17027 to 17031 (Decimal). Ensure that the number is numeric and in ASCII (48 decimal to 57 decimal), and that the POKe statements are executed whenever the program is RUN.

The auto start feature can be disabled by resetting STARTIF to 0 by the instruction:

```
POKE 17832,0
```

3.2

MEMORY ALLOCATION

HUSKY's user memory is automatically partitioned by the BASIC interpreter according to the number of program lines entered by the user and the variable storage space required. Variable storage is reserved using 'DIM' dimension statements and otherwise as needed during the entry of a program.

As a guide, a 32K HUSKY standard model will allow user programs up to about 600 lines of BASIC and still leave enough space for about 2,000 variables to be stored. These variables can either be individually allocated or can form arrays.

NOTE: certain types of variable (strings, double precision) can only be used once array space has been declared for them using 'DIM'. Failure to do this will produce 'Array Error'.

3.2.1

Line Numbers

Line numbers are in the range 0 to 65,280. Extra lines can be inserted into the text by simply typing in a new line number between the two lines of interest. In view of this it is recommended that line numbers are separated by an increment of 10 for each new line. Line numbers are stored in binary form and occupy a constant space in memory regardless of the number chosen. Each line number requires two bytes of memory.

3.2.2

Variable Storage

HUSKY variables can be stored as either single or double precision floating point numbers. All numbers have the range $\pm 10^{126}$. All simple variables are single precision giving 6 digit accuracy. Variables are designated by one alphabetic character (A-Z) followed by an optional alphanumeric character (A-Z) or (0-9), giving up to 962 separate variables.

NOTE: The simple variable names **ON**, **TO**, and **IF** should be avoided as these are also reserved words. Use as variables can lead to execution errors.

3.2.2.1 **Arrays**

Additionally, variables may form a one-dimension array using 'DIM'. Variable arrays may optionally be designated double precision giving 14 digit accuracy but with an additional memory overhead. Single precision arrays are designated by a number in parenthesis following the variable name:

```
Al(19),Z(2),C(X),A(20*T),AD(10),AZ(62)
```

There can be 962 separately identified arrays of single precision variables.

NOTE: The number in parenthesis can be an expression. Each variable occupies 5 bytes of HUSKY's user memory when part of an array; 7 bytes otherwise.

3.2.2.2 **Double Precision**

Double precision arrays are identified by the symbol "!" followed by a number or expression in parenthesis. Double precision variables have 14 digit accuracy.
e.g:

```
Al!(19),Z!(Z),C!(X) AA!(23)
```

There are a further 962 arrays.

Each double precision element occupies 9 bytes of HUSKY memory. Remember that HUSKY variable definitions are common to all programs co-resident in user memory. If the variables are not actually common to separate programs then remember to use different names or numerical suffixes to differentiate between them. Results can otherwise be confusing.

The following are all valid and independent variables:

```
X,X0,X1,X2,X3,X4,X5,X6,X7,X8,X9,  
X(0),X0(0),X1(0),X1!(0),X0!(0), etc.
```

3.2.3 String Storage

Literal strings are stored in arrays, and are identified by the string symbol \$.

String **NAMES** are defined exactly as variable names, with one alphabetic character (A-Z) and one optional alphanumeric character (A-Z,0-9) giving a total of 962 separately identified string variables.

Because storage overflow problems in the field caused by insufficient string pool space would be unacceptable in **HUSKY**, the programmer has to declare all strings by 'DIM' statements whether used in arrays or not.

The maximum length of any string is 255 characters.

```
DIM A1$(10,15)
```

defines a string array of 11 elements, each of which can be up to 15 characters long.

The maximum size of any array depends on the memory available. Examples of String variables are:

```
A1$,Z$,C5$(22),H$(J),MM$(32)
```

String variables can be initialised using LET, just assigned using = (equate), or by READING from DATA statements.

The first element of a string array may be referenced with just the variable name, without the need for any number in parenthesis.

Example:

```
A1$,Z$ actually refer to the variables A1$(0), Z$(0)
```

A single variable may therefore be defined as:

```
ST$(0,10) and used as:
ST$
```

3.2.4 Multiple Statements

HUSKY Basic supports multiple statements in a single program line. Statements are separated by a colon (:)

Example:

```
100 A=0:B=10:PRINT A,B:B=11
```

The primary use of this feature is to reduce a program size. It may also make a program more legible.

3.2.5 PROGRAM LIMITS AND MEMORY USAGE

1. Ranges

Variables: $\pm 9.99999E\pm 126$
 String arrays: Up to 255 characters per string
 Line numbers: 0-65280 inclusive
 Program line length: Up to 96 characters including line number.

2. Precision

Single precision variables have 6 digit resolution. Double precision variables have 14 digit resolution.

3. Memory Overhead

Program lines require 4 bytes minimum, as follows:

Line number: 2 bytes
 Line length: 1 byte
 Carriage Return: 1 byte

Also each reserved word, operator, variable name character, special character and constant character requires one byte.
 Maximum program size in present versions: 48K Bytes (Page 0)

3.2.6 DYNAMIC (RUN-TIME) MEMORY ALLOCATION

1. Symbol Table

Entries occupy: 7 Bytes each.
 Maximum symbol table size: 16K Bytes

2. Single Precision Array variables

e.g. DIM(X) occupies $(X+1)*5+7$ Bytes

3. Double Precision Array variable

e.g. DIM!(X) occupies $(X+1)*9+7$ Bytes

4. String Array Elements

e.g. DIM\$(A,N) occupies $(A+1)*N+7$ Bytes

5. Array Size

Maximum number of elements in an array is 16,383.

See Section 3.8.5 for an example of a larger size array program.

3.3.

EXPRESSIONS AND OPERATORS

3.3.1

Symbolic operators for use with numerical variables:

'=' Equality or assignment of values

'+' Addition

'-' Subtraction or negation of value

'*' Multiplication

'**' Powers

'/' Division

'(' Open parenthesis

')' Closed parenthesis

'>=' Equal to or greater than

'<=' Equal to or less than

'<' Less than

'>' Greater than

'<>' Not equal to

3.3.2 Operators for use with string variables:

'=' Equality or assignment of values

'<>' Not equal to

'+' String addition or concatenation

BASIC FUNCTIONS

3.4 FUNCTIONS

HUSKY BASIC understands four kinds of function:

- :Commands
- :Statements
- :Operators
- :Functions

Commands generally initiate an action and can also control execution of application programs.

Statements form the structure of application programs and direct program flow.

Operators are used to relate data and to test for specified conditions.

Functions perform arithmetic or string computations.

Some commands and statements will function without further information; others require an argument. An argument is either a value, an expression or another function.

A complete index to functions is given overleaf.

INDEX TO BASIC FUNCTIONS

3.4.1	ABS	Function	Returns absolute value of argument
3.4.2	ADIN	I/O Statement	Returns value of A/D converter input
3.4.3	ARG	Function	Sets up argument for CALL
3.3.4	ASC	Function	Returns decimal equivalent of string
3.4.5	ATN	Function	Returns Arc-Tangent of argument
3.4.6	CALL	Statement	Calls machine-code subroutine
3.4.7	CHR\$	Function	Returns string equivalent of argument
3.4.8	CLEAR	Command	Clears all variables and pointers
3.4.9	CONT	Command	Continues execution
3.4.10	COS	Function	Returns cosine of argument
3.4.11	CRT	Command	Switches console to RS-232
3.4.12	DATA	Statement	Holds data for use by program
3.4.13	DIM	Statement	Initialises arrays
3.4.14	END	Statement	Terminates execution
3.4.15	EXP	Function	Returns e to the power of the argument
3.4.16	FOR	Statement	Starts FOR...NEXT loop
3.4.17	FRE	Statement	Returns number of bytes free
3.4.18	GOSUB	Statement	Branches to subroutine
3.4.19	GOTO	Statement	Branches to alternative line
3.4.20	HELP	Statement	Initialises HELP key text pointer
3.4.21	IF	Statement	Conditional branch
3.4.22	INCHR	I/O Statement	Returns single character from keyboard
3.4.23	INKEY	I/O Statement	Returns keyboard status
3.4.24	INP	I/O Statement	Returns value at port address
3.4.25	INPUT	I/O Statement	Returns value at port address
3.4.26	INPUT USING	I/O Statement	Fixed field INPUT
3.4.27	INT	Function	Returns integer equivalent of argument
3.4.28	JSR\$	Function	Returns fixed-field string
3.4.29	LEFT\$	Function	Returns left part of string
3.4.30	LEN	Function	Returns lengths of string
3.4.31	LET	Statement	Equates variables, strings
3.4.32	LINCHR	I/O Statement	Returns single character from RS-232
3.4.33	LINPUT	I/O Statement	Returns entry from RS-232
3.4.34	LIST	Command	Lists program at LCD
3.4.35	LLIST	I/O Statement	Lists program at RS-232
3.4.36	LLOAD	I/O Statement	Loads program from RS-232
3.4.37	LN	Function	Returns natural logarithm
3.4.38	LOG	Function	Returns logarithm to base 10
3.4.39	LOPCHR	I/O Statement	Sends single character to RS-232

3.4.40	LPRINT	I/O Statement	Prints to RS-232
3.4.41	MID\$	Function	Returns mid portion of string
3.4.42	NEW	Command	Initialises program space
3.4.43	NEXT	Statement	Concludes FOR...NEXT loop
3.4.44	ON BREAK	I/O Statement	Vectors program on BREAK key
3.4.45	ON COMMS	I/O Statement	Vectors program on COMMS failure
3.4.46	ONERROR	Statement	Vectors program on Syntax error
3.4.47	ON GOSUB	Statement	Conditional branch to subroutine
3.4.48	ON GOTO	Statement	Conditional branch
3.4.49	ON POWER	I/O Statement	Vectors program on POWER key
3.4.50	OPCHR	Statement	Outputs 1 or more ASCII characters
3.4.51	OUT	I/O Statement	Outputs to specified port
3.4.52	PEEK	Statement	Returns decimal byte value of memory locations.
3.4.53	POKE	Statement	Sets a specified memory location
3.4.54	POP	Statement	Returns a numeric value from machine code linkage/stack.
3.4.55	PRINT	Statement	Outputs to LCD
3.4.56	PUSH	Statement	PUSHES a numerical variable onto the machine code linkage stack.
3.4.57	READ	Statement	Reads data statements
3.4.58	REM	Statement	Can be followed by REMarks
3.4.59	RESTORE	Statement	Resets Read pointer
3.4.60	RETURN	Statement	Returns from subroutine
3.4.61	RIGHT\$	Function	Returns rightmost portion of a string
3.4.62	RND	Function	Produces random number
3.4.63	RUN	Command	Starts a program execution
3.4.64	SGN	Function	Returns a value for the sign of its argument.
3.4.65	SIN	Function	Returns Sin of argument
3.4.66	SQR	Function	Returns square root of its argument.
3.4.67	SRCH	Function	Returns target string array position.
3.4.68	STEP	Statement	Modifies increment IN FOR-NEXT loop
3.4.69	STOP	Statement	Terminates program execution
3.4.70	STR\$	Function	Returns a string equivalent of a numerical argument.
3.4.71	TAB	I/O Statement	Formats Print output
3.4.72	TAN	Function	Numerical argument
3.4.73	VAL	Function	Returns numerical evaluation of a character string.
3.4.74	VER	Command	Returns version number of BASIC INTERPRETER.
3.4.75	WINCHR	I/O Statement	Inputs a single character from an optical wand.
3.4.76	WINPUT	I/O Function	Input character string from optical wand.

ABS

- Function** **ABS(N)** returns the absolute value of the argument.
- Syntax** **ABS(N)** where N can be a variable, number or result of a numeric expression.
- Examples** Y=ABS(-12.345) returns Y=+12.345
 Y=ABS(-0.5) returns Y=+0.5
 Y=ABS(0.5) returns Y=+0.5
- or if V=-27
- Y=ABS(V) returns Y=+27
 Y=ABS(V+50) returns Y=+23

ADIN

Function **ADIN** is a statement for use with the optional multiplexed analogue to digital converter fitted within the Husky.

It causes an analogue to digital conversion to be executed for a channel specified in the argument.

Syntax Variable = ADIN(x)
Where x is a channel number

Examples A=ADIN(5)

PRINT "VOLTAGE INPUT"=",ADIN(5)*100

Remarks Channels are defined as 0 to 7.

The value obtained can be attributed to a variable or contained in an arithmetic expression.

If a channel is overrange, the value '9999' is returned. For sensitive applications this reading must be detected, a numeric result being returned in order not to cause mismatch or other syntax errors.

ADIN(x) returns the polarity of the applied voltage to the channel together with its magnitude.

The ADIN(x) statement executes a conversion in 600 ms.

Huskies not fitted with the analogue to digital converter will simply return meaningless data if ADIN is executed.

ARG

Function ARG is used for passing parameters to machine code subroutines.

Syntax D=ARG(N)
Where D is a Dummy Variable
N is the number to be passed.

Examples P=ARG(10) Passes 10
P=ARG(V) Passes V

Remarks ARG loads the Z80 E and C registers.
They are loaded with the higher and lower byte portions of a 16 bit representation of the ARG argument.

For example if V in the example above was 4100 then:

E=16 C=4

NOTE: $16 \times 256 + 4 = 4100$

ASC

Function **ASC** returns the decimal ASCII value of the first character of a string variable.

Syntax `ASC("STRING")`

Examples `Y=ASC(X$(3))`
 `PRINT ASC("A")`

will print 65

```
10 T$="AB"  
20 PRINT ASC(T$)
```

will also print 65.

Remarks The string argument must be enclosed in parenthesis.

ASC is particularly useful for detecting special characters like control codes, and/or distinguishing them from digits 0-9 or characters A-Z.

ATN

- Function** ATN generates the angle whose tangent is specified by expression.
- Syntax** ATN(N)
- Examples** A=ATN(0.6009)
B=ATN(V)
sets A equal to 0.541071 Radians, i.e. 31°
B equal to ARCTAN of variable V
- Remarks** The resultant angle is specified in radians.
To convert to degrees:
A=ATN(V) x 180/3.1415926
(PI=3.14159269)

CALL

Function **CALL** allows a machine code subroutine or an existing routine in **HUSKY's** housekeeping package to be called from BASIC and executed.

Syntax D=CALL(N)

where D is a dummy variable
 N is a Call Address

Examples 10 A=CALL(0)
 20 B=CALL(V) where V is the address to be called

switches execution to the program located at the location specified. This function causes a restart of the **HUSKY's** software.

Remarks Further information on the use of **CALL** is provided in section 3.6 and also in part 4, Advanced programming.

WARNING: Use **CALL** only as indicated in Part 4 Advanced Programming, Section 2. Use of unspecified calls will crash **HUSKY**, resulting in loss of data, etc.

When control has passed back to Basic from a machine code program, then D = the final number stored in the Z80 accumulator when the program returns.

CHR\$

Function CHR\$ Performs the inverse of ASC. It returns the one character string specified by the expression.

Syntax CHR\$(N)

where N is a numeric expression between 0 and 127.

Examples

```
10 DIM A$(10,10)
1000 A$ = CHR$(34)
1010 PRINT "HE SAID," ,A$, "HELLO",A$
```

prints

```
HE SAID, "HELLO"
```

```
2000 A=CHR$(B)
```

```
IF B=7 then
```

```
2000 PRINT CHR$(B)
```

HUSKY bleeps.

```
2020 PRINT CHR$(1)
```

This clears the HUSKY screen.

Remarks

The expression must have the value 0 to 127. The expression must be enclosed in parenthesis.

(The parenthesis character " cannot be imbedded in PRINT strings since it acts as a delimiter).

CLEAR

- Function** **CLEAR** will set all variables defined in the applications program to zero.
- Syntax** **CLEAR**
- Examples** **CLEAR**
resets all variables.
- Remarks** Note that the variables are also set to zero if at any time program lines are altered or added. Variable contents are otherwise maintained indefinitely by **HUSKY**, as described earlier.

CONT

Function	CONT continues Basic program after ESC.
Syntax	CONT [LINE] Line is optional and allows the program to continue from a different line number.
Examples	CONT continue from next line or CONT 1000 continue from line 1000
Remarks	Allows a program to continue execution from the line following the occurrence of: a) detection of the break key b) execution of a 'STOP' statement c) execution of an 'END' statement d) detection of a syntax error By specifying a line number argument continuation can occur from that line number. In the case of multiple statement lines, execution will occur from the next line.

COS

Function **COS** returns the Cosine of the argument.

Syntax **COS**(ANGLE)
where angle is in radians.

Examples $X = \text{COS}(Y)$ sets X equal to the cosine of Y.
 $R = \text{COS}(5)$
 R = COS of 5 radians

Remarks **NOTE:** The argument is expressed in radians.

For angles in degrees, then use:

$X = \text{COS}(A \times 3.1459269/180)$

(PI=3.4159269)

CRT

Function CRT causes the HUSKY to treat the serial port as the display and keyboard rather than the internal LCD display and keyboard.

Syntax CRT

Examples CRT

sets keyboard and display functions to the RS-232 port.

Remarks The command is useful for entering programs quickly; also characters not on the HUSKY keyboard may be used, e.g. lower case alphabetics.

Any standard VDU may be used. Baud rates and other parameters are set on the HUSKY using 'Initialise Communications'.

Remember that a 'crossed' RS-232 lead is needed with GND,RXD and TXD circuits installed.

DATA

Function **DATA** statements hold constants for use in subsequent **READ** statements.

Syntax **DATA** item 1, item 2....., item n.

Examples 100 **DATA** 12,A1,25,SIN(0.5),MID\$(A\$,3,4),99
 110 **DATA** 5,6,7,8,6*5,"HELLO"

After reading the value 99 the **READ** will take 5, and so on, until every data value is used. This form of data storage is not affected by **CLEAR** or program line changes which will erase variables. To return to the first value in a **DATA** list use the **RESTORE** statement.

Remark The arguments for **DATA** statements can be either numerical values, variables, or expressions. Multiple arguments are separated by commas.

String variables may be used in **DATA** statements. Direct text must be enclosed in quotes.

NOTE: Care must be taken that variables in the corresponding **READ** statement are of the same type as the **DATA**.

DIM

Function

The **DIM** statement allocates arrays within the **HUSKY**.

Syntax

DIM variable name (array size)

DIM double precision variable name (array size)

DIM string variable name (array size, element size)

where array size and element size are numeric expressions.

Examples

```
DIM A (25)
```

defines an array variable **A** of 26 elements, including the element **A(0)**. The array size must be a positive integer.

```
DIM J4(100), J5(100),Q(1000),J(N)
```

defines multiple arrays. **N** is a previously defined variable.

Remarks

There are three kinds of **DIM** statement in **Husky Basic** : Simple variables, double precision variables and strings.

Double precision and string variables can only be used once space has been reserved by **DIM**.

The maximum array size is limited by available memory, and is typically about 28,000 single precision variables for a 144K **HUSKY**, subject to program size.

NOTE: The maximum number of elements in a single array is 16,384.

All array types have a range from 0 (zero) to the limit defined in the **DIM** statement.

Arrays can be allocated to any combination of variable names.

Double precision arrays are defined in the same way except for the use of "!". Each element has a 14-digit size.

```
DIM A!(25)
DIM AZ!(32)
```

define double precision arrays.

Space for string arrays is reserved in **HUSKY** memory using a **DIM** statement followed by string name, the number of elements in the array, and the maximum string length

required.

```
DIM A$(25,10)
```

defines a string array with 25 elements each of maximum length of 10 characters.

Each character in a string requires 1 byte of storage. Since HUSKY retains variable definitions indefinitely, DIM statements should not be included in normal user program sequences. Instead, arrays should be initialised by a separate routine or defined manually, i.e. without line numbers.

A useful technique is to set in the application software a switch indicating whether the program is being used for the first time or not. If the switch is a variable which is made non zero when the program is first run, the fact that any modifications or use of the CLEAR statement will clear the value to zero can be used to determine whether the arrays should be redefined or not.

```
10 If A = 1 THEN GO TO 100
20 DIM J(50)
   .
   .
   .
90 A = 1
100 REM user program starts here.
```

NOTE: That the variable A could be used as a counter to indicate the number of times that the program has been run.

END

Function **END** terminates user program execution.

Syntax **END**

Examples 1000 **END**

when executed, displays:

STOP IN LINE 1000

and returns control to the Basic interpreter.

Remarks **END** does not require an argument.

EXP

Function	EXP(X) generates the value of e raised to the power of X.
Syntax	EXP(N)
Examples	A=EXP(3) A=e to the third power B=EXP(N) B=e to the power of N
Remarks	X must be in the range -290 to +290. If X is not in this range then a 'Magnitude Error' will occur. X may be an expression. e is defined as: 2.7182818284590

FOR

Function FOR executes a series of instructions in a loop a given number of times.

Syntax FOR N = A TO B STEP C
NEXT N

Where N is a variable, A is a numeric start value, B is a numeric end value and C is an increment/decrement value.

A, B and C can be numbers, variables or expressions.

Examples 10 FOR A = 1 TO 100 STEP 10
.
.
.
100 NEXT A

causes the value of A to equal 1,11,21 etc., for each execution of the loop.

To obtain a **decrementing** count negative values of STEP are used.

1000 FOR Z1=91 TO 10 STEP -.1
.
.
.
1050 NEXT Z1

Remarks When followed by a NEXT statement, FOR will execute the intervening parameters for the number of times indicated by the values A and B divided by N to the formula:

$$\frac{B-A+1}{N}$$

Implements the loop 20 times.

When terminating FOR...NEXT loops it is important to end using NEXT and **not** GOTO. Alternatively, exiting via a RETURN when in a subroutine will not cause a build up of the 'control stack'.

Failure to do this will result in a control stack error.

NEXT requires as an argument the same variable name used in the corresponding FOR statement.

STEP may be used to modify a FOR NEXT loop for increments other than 1.

FRE

- Function** **FRE** returns the number of free bytes left in memory.
- Syntax** **FRE**(0)
 0 is a dummy argument.
- Examples** `PRINT FRE(0)`

 displays amount of free space.

 The argument (0) is a dummy.

 or

 `A=FRE(0)`

 Sets A = amount of free space.
- Remarks** Note that if a program has not been RUN then the variables will not have been assigned space. This enables easy display of program size and, after RUNing, the total memory usage.

 The latter examples allows the AUTO SIZING of arrays to their maximum size.

 e.g. `DIM A((FRE(0)-10240)/5)`

 This autosizes A leaving 10K (10240 byte) for other variables. "5" is the number of bytes used for each member of A. Remember to leave enough space for all simple variables.

GOSUB

Function GOSUB causes execution to jump to a BASIC subroutine located elsewhere in the applications program.

Syntax GOSUB line number
GOSUB numeric expression

Examples GOSUB 150

causes program execution to jump to a subroutine located at 150.

The argument can also be an argument or expression, for example:

```
GOSUB SI  
or  
GOSUB SI+100*A
```

This is illustrated by the program sequence:

```
10 INPUT A  
20 SI=1000  
30 GOSUB SI+100*A  
40 STOP  
  
1000 PRINT "THIS IS SUBROUTINE 1"  
1010 RETURN  
1100 PRINT "THIS IS SUBROUTINE 2"  
1110 RETURN  
etc.
```

which branches to the subroutine specified in A.

Remarks

A subroutine must always be terminated by a 'RETURN' statement. Program execution then continues from the statement immediately after the GOSUB. GOSUB always requires a line number as an argument.

WARNING: Use this feature with care : ensure that only valid argument values can exist under all circumstances.

GOTO

Remarks	GOTO branches unconditionally to a specified line number.
Syntax	GOTO line number GOTO numeric expression
Examples	<p>GOTO 50</p> <p>causes program execution to commence immediately at line 50.</p> <p>10 GOTO 100</p> <p>causes program execution to skip from line 10 to line 100.</p> <p>The line number may also be a variable or an expression. Examples:</p> <p>GOTO A</p> <p>or</p> <p>GOTO A*100</p> <p>are equally valid.</p> <p>The program sequence:</p> <pre>20 ZB=150 30 GOTO ZB 150 PRINT "THIS IS LINE",ZB</pre> <p>would give:</p> <p>THIS IS LINE 150</p>
Remarks	<p>GOTO is always used with an argument (line number). If used as an immediate command, GOTO will cause program execution to commence from the line number specified. If used as part of a program, GOTO will cause program execution to skip to the line number specified. If the line does not exist execution will skip to next line following.</p> <p>If neither the line specified or a higher line is present in memory, HUSKY will display:</p> <p>LINE NUMBER ERROR IN LINE.....</p>

If the GOTO line was unnumbered, the error message line number will be meaningless.

HELP

Function **HELP** displays lines of text on HUSKY's screen when the **HELP** key is pressed.

Syntax **HELP** line number
 HELP numeric expression

Examples

```
.
1010 REM THIS IS A PIECE OF HELP TEXT
1020 REM INTENDED TO ASSIST THE OPERATOR
1030 REM IN THE USE OF HUSKY
1040 REM PLEASE PRESS ENTER TO RETURN
1050 REM TO PROGRAM
.
.
10000 HELP 1010
```

Execution of line 10000 will cause the block of remarks to be used as the **HELP** text.

Remarks

HELP requires an argument which must be a valid line number. The referenced line must be a **REM** statement. When the operator presses **HELP**, text from the referenced **REM** is placed on the screen. Use of the | | cursor keys causes scrolling through a contiguous block of **REM** statements.

The **HELP** text start may be changed by executing another **HELP** statement.

It is recommended that all the **HELP** text be written contiguously so that the operator may scroll through all of it. Scrolling stops when any non-**REM** line is encountered, so blocks of text can be easily partitioned if required. The **HELP** display is terminated by pressing the 'ENTER' key.

IF

Function	IF selectively executes program statements dependant on result of an expression.
Syntax	<p>IF A (op)B THEN C</p> <p>Where A is an expression, variable or constant and (op) is an operator =, <, > etc.</p> <p>B is an expression, variable or constant</p> <p>C is either a line number or program statement sequence.</p>
Examples	<pre>IF A = B THEN 100 IF A > B THEN PRINT "NO"</pre> <p>C may be a multiple statement which will be entirely executed if the condition is true, but skipped if not.</p> <p>HUSKY also permits nesting of IF statements. Example:</p> <pre>IF A = B THEN IF C = D THEN PRINT "FINISHED":GOTO 1000</pre> <p>String expressions may be tested for equality or inequality: Example:</p> <pre>IF A\$ + B\$ = "ABCDEAB" THEN 100</pre>
Remarks	<p>For strings of unequal length the equality will be true if the string on the left of the equates is equal to or part of the string on the right. Example:</p> <pre>IF A\$ = "ABC" B\$ = "ABCDE" A\$ = B\$ is true B\$ = A\$ is not true</pre>

INCHR

Function INCHR inputs a single character from the keyboard.

Syntax INCHR variable name
INCHR "prompt string", variable name

Examples

```
10 INCHR "DECIMAL VALUE",A
20 PRINT A
30 GOTO 10
```

displays:

```
DECIMAL VALUE A 65
DECIMAL VALUE B 66
etc.
```

Remarks INCHR does not wait for ENTER to be pressed. A prompt message is displayed as with INPUT. Only one variable may be entered. The variable is set to the decimal value of the character, as detailed in Appendix I.

A comma following INCHR or the prompt message will suppress the prompt character '?'

NOTE: No values are returned for SHIFT, HELP, CONTROL or POWER keys, although these function normally during INCHR.

'Escape' cannot be used to return to Basic from an INCHR loop, since it simply returns the value 27!

INKEY

Function	INKEY Checks the keyboard for the operation of a key.
Syntax	INKEY A
Examples	<pre>INKEYQ=3 THEN STOP</pre> <p>detects control C and stops.</p>
Remarks	<p>The variable A is attributed the value of the key pressed. If no key is detected then the variable is equal to zero.</p> <pre>INKEY A = 65 THEN 1000</pre> <p>This variation allows a variable to be attributed the value of the key pressed, and also its value to be tested as in 'IF' statements. If the result is true, execution branches to the line indicated.</p>

INP

- Function** INP inputs a byte from the designated port.
- Syntax** INP(PORT)
where PORT is numeric Port Number.
- Examples** A = INP(192)
sets A equal to the decimal value (0-255) of the binary data present at the 8-bit port.
- Remarks** Refer to the Port Map in Part 4 (Advanced Programming) Section 10 for details of Port addresses and functions.
INP can also be used with the optional parallel input Port.

INPUT

Function INPUT obtains a line of data from HUSKY's keyboard.

Syntax

```
INPUT variable name
INPUT, variable name
INPUT. variable name
INPUT "prompt string" variable name
INPUT "prompt string", variable name
INPUT "prompt string". variable name
```

Examples

```
INPUT A

Inputs the variable A

INPUT "PARAMETER VALUES" A,B,C

Input the values of A, B and C with the prompt message:

PARAMETER VALUES
?
```

Remarks Note that input lines are always terminated by pressing the 'ENTER' key.

INPUT cannot be used as a direct statement.

A prompt message can be displayed by the INPUT statement. A string variable name or a literal string is required as an argument. Multiple variables can be strung together, separated by commas.

A comma following INPUT (or prompt message) will suppress INPUT's '?' prompt.

A period following INPUT (or prompt message) suppresses both "?" and an echo to the terminating ENTER. This can be useful on formatted displays, to prevent scrolling.

Input will function with any type of variable, but remember that String and Double precision variables **must** have previously been defined as a 'DIM' statement.

NOTE: If INPUT's argument is a simple variable and an invalid entry is made, INPUT will simply re-prompt without an error message. An example is:

```
10 INPUT A
```

The user types 'HELLO'. After 'ENTER', INPUT generates a loud 'bleep' and re-prompts on the next line.

INPUT USING

Function **INPUT USING** obtains a line of data from HUSKY's keyboard and validates this data with a user defined mask.

Syntax **INPUT USING** (<validation string>,min,max,E)<input string>
INPUT USING (<validation string>,min,max,E)VARIABLE

where min, max are numeric expressions and E is a special command.

Examples **INPUT USING**(A\$,A,B)\$
INPUT USING(B\$,,)A
INPUT USING ("A999",2,4)\$

Causes ? prompt

A minimum of 2 characters must be entered up to a maximum of 4.

The inputs are stored in string expression S\$. The first entry must be a letter in the range A-Z, the remaining entries must be numbers in the range 0-9.

INPUT USING ("A999",,4)\$

The same as above, but minimum defaults to 1.

INPUT USING ("A999",2)\$

A minimum entry of 2 and maximum defaults to 95. Any entries after the fourth are validated against WILD.

INPUT USING ("A999")\$

Minimum and maximum default to 1 and 95 respectively.

INPUT USING (M\$,3,6)\$

A minimum of 3 and up to a maximum of 6 entries are allowed. The entries are validated against the string expression M\$. The user will receive a '?' prompt.

INPUT USING (M\$,3,6)"ENTER",S\$

Same as above, but with an 'ENTER' prompt.

INPUT USING (M\$,3,6,E)"ENTER",S\$

Same as above, but after the 6th character has been entered HUSKY will assume that the enter key has been

pressed.

Remarks

Entry fields can be made numeric, alphanumeric, etc., in flexible configurations. Entries that do not fit the mask are rejected with clear warning indications.

INPUT USING is an extension of the standard INPUT statement (see section 3.3.2.11). However, as each entry is made by the operator, the character is checked against the defined check string. A value which is out of range will cause an audible bleep to occur and a special character to be momentarily echoed on the screen, rejecting the invalid entry.

The input validation string consists of defined validation characters either in the form of a string expression e.g. A\$,B4\$ etc., or a direct string enclosed in quotations. Input validation string may **NOT** be omitted.

The optional min and max expressions define the minimum number and maximum number of entries required. If neither min nor max are specified then default values of min 1 and max 95 are assumed. If maximum exceeds the number of characters in the input validation string, then the entries for which there are no validation characters are validated automatically with WILD characters.

The values min and max can be variables or arithmetic expressions. Min must not exceed max or a syntax error will occur. Min must not be negative or a magnitude error will occur. Min and max must not exceed 95 or a syntax error will occur.

'Enter' terminates entry. 'Enter' will not be accepted if the number of characters is less than minimum.

When the number of characters exceeds maximum no more characters will be accepted. A warning tone will sound. The cursor control keys — and — are available in INPUT USING fields to edit or correct data prior to pressing 'ENTER'.

The E parameter is optional. If it is specified then the 'enter' key does not have to be pressed to terminate the input of the characters allowed by the validation mask.

CAUTION: If the input field extends over more than one Husky screen line confusing results may occur in subsequent attempts to delete or edit characters.

A comma following the validation expression will suppress

the prompt character "?". However, if a prompt string is being used then the comma must follow this prompt string to suppress the prompt character "?"

A string, enclosed by quotes, immediately following the validation expression will cause that string to act as the prompt.

NOTE: Remember that string variables (A\$,S\$, etc) must have previously been defined by a DIM statement. See Section 3.3.2.2.

The scroll left and right keys are enabled in order to allow the user to modify incorrect entries by simply over-typing them. The delete key will delete the last character entered.

VALIDATION CHARACTER TABLE

A	A-Z only
B	A-Z space
C	A-Z 0-9
D	A-Z 0-9 space
N	0-9 + - . ,
9	0-9 only
.	Decimal point only
*	WILD

NOTE 1: WILD allows a character in the range of space - del which includes all letters and numbers. See Fig.3.8.2

NOTE 2: Control characters are not accepted. Other characters than those in the table will give a Syntax error when the statement is RUN.

INT

Function **INT** returns the truncated equivalent value of the argument.

Syntax **INT** (N)
 where N is a numeric expression

Examples A = **INT**(123.456)
 sets A equal to 123.

A dark rectangular box containing the text "JSR\$" in a white, stylized, serif font.

Function **JSR\$(A,N)** This function creates a string of given length where the numeric argument is right justified and padded with leading zeroes.

Syntax JSR\$(A,N)
 A=Numeric argument
 N=length

Examples A = -12.34

 JSR\$(A,8) = -0012.34
 JSR\$(A,5) = -12.3
 A = 12.34

 JSR\$(A,8) = 00012.34
 JSR\$(A,5) = 012.3

Remarks For positive numbers the '+' signs are replaced by '0'.
 Sign (0 and -) and decimal point each count as one
 character.

LEFT\$

Function **LEFT\$(string,n)** Returns the first n characters from a designated string. Either the string or n may be expressions.

Syntax LEFT\$("STRING",N)

where string is a string expression N is the number of left most characters.

Examples 10 A\$ ="ABCDEFGH"
 20 B\$ = LEFT\$(A\$,3)
 30 PRINT B\$

prints ABC.

Remarks LEFT\$(A\$,0) = Null String i.e. " "
 LEFT\$(A\$,50) = ABCDEFG

LEN

Function **LEN(string)** returns the length of a string.

Syntax **LEN ("STRING")**
where string is a string expression.

Examples 10 A\$ = "ABCDEFGG"
 20 PRINT LEN (A\$)

 prints 7.

Remarks The string may be an expression (non-null characters).
 A null string has zero length.

LET

Function **LET** assigns a value to a variable.

Syntax **LET** variable name = numerical expression
 LET string variable = string expression

Examples

```
LET C =5
LET D3 = B
LET J = SIN(X)
LET A$ = "HELLO"
LET A$ = B$ (see note)
```

Entry of the **LET** statement is optional. The formats:

```
C = 5
D3 = B
J = SIN(X)
A$ = "HELLO"
A$ = B$ (see notes below)
```

are equally acceptable.

Remarks **NOTE 1:** A string such as **A\$** must be defined as a single dimension array. In reality **A\$** is equivalent to **A\$(0)** but either definition is accepted.

NOTE 2: When equating string arrays of unequal length the string array on the right side of the equates will be truncated on the left side.
 Example:

```
A$ = "HELLO"
B$ = "GOODBYE"
A$ = B$
```

then **A\$** = "GOODB"

Strings can be created by multiple additions.
 Example:

```
A$ = A$ + MID$(C$,4,3)+CHR$(65)
```

String expressions can also be created from complex string expression functions. The derivation of the function requires the creation of an expression stack which can handle a maximum of ten functions. If the stack is exceeded a 'string complexity' error will result.
 Example:

```
A$ = LEFT$(RIGHT$(MID$(A$,3,N),A),B)
```

LINCHR

Function LINCHR inputs a single character from the RS-232 port.

Syntax LINCHR variable name
LINCHR "prompt string", variable name

Examples

```
10 LINCHR "DECIMAL VALUE",A
20 PRINT A
30 GOTO 10
```

displays:

```
DECIMAL VALUE A 65
DECIMAL VALUE B 66
etc.
```

Remarks LINCHR does not wait for carriage return to be received. A prompt message is displayed on the LCD screen as with INPUT. Only one variable will be received. The variable is set to the decimal value of the character, as detailed in Appendix I.

A comma following LINCHR or the prompt message will suppress the prompt character '?'

LINPUT

- Function** LINPUT obtains a line of data from HUSKY's RS-232 port.
- Syntax** LINPUT variable name
 LINPUT, variable name
 LINPUT "prompt string" variable name
 LINPUT "prompt string", variable name
- Examples** INPUT A
- Inputs the variable A
- LINPUT "PARAMETER VALUES" A,B,C
- Input the values of A, B and C with the prompt message:
- ```
PARAMETER VALUES
?
```
- Remarks** Note that input lines are always terminated on reception of a carriage return (CR) character.
- LINPUT cannot be used as a direct statement.
- A prompt message can be displayed by the LINPUT statement. A string variable name or a literal string is required as an argument. Multiple variables can be strung together, separated by commas.
- A comma following LINPUT (or prompt message) will suppress LINPUT's '?' prompt.
- LINPUT will function with any type of variable, but remember that String and Double precision variables **must** have previously been defined as a 'DIM' statement.
- NOTE:** If LINPUT's argument is a simple variable and an invalid entry is made, LINPUT will simply re-prompt without an error message. An example is:
- ```
10 LINPUT A
```
- Husky receives 'HELLO'. After 'CR', LINPUT generates a loud 'bleep' and re-prompts on the next line.

LIST

Function **LIST** causes **HUSKY** to list one program line on the LCD screen.

Syntax **LIST** line number
 LIST numerical expression

Examples **LIST** 100

lists line 100 on the screen. If there is no line 100 it will **LIST** the next line in sequence.

Remark **LIST** can be followed by a line number. **LIST** will only display the first line.

Following the listing of the specified line, pressing | will list the next line unless the end of the program has been reached. Any other key will abort the **LIST** feature and return to **BASIC**'s input mode.

LLIST

Function LLIST causes HUSKY to list the applications program to the serial port.

Syntax LLIST line number

Examples LLIST 100

will cause a listing to start at line 100 in the program.

Remarks If LLIST is followed by an argument (a line number) then LLIST will begin listing to the serial port at the line specified in the argument or the first line following if line number does not exist. Otherwise, listing begins at line 0.

LLIST can be terminated at any time by the escape ('ESC') key being depressed and held down until HUSKY responds with a confirmatory 'beep'. Because a transmission buffer is used, up to 256 characters may remain for transmission after 'ESC' is confirmed. Power OFF will also abort a listing at any time without loss of stored data or program.

It is advisable to check HUSKY's Communication Parameters (Section 5.3) before using LLIST for compatibility with the external device receiving the data. When a printer is being used, remember to check for baud rate and line feed requirements.

NOTE 1 LLIST increases HUSKY's battery consumption by turning on the communications power supply. This is left on at the end of the listing, so it is advisable to shut it down either by typing OUT 132,0 (Section 4.10) or by power off.

NOTE 2 Because the communications power supply is powered up by LLIST, a single high-to-low character will appear at the start of the record. Many systems will see this transition as a single, spurious, character. To avoid this problem, power up the interface first with either:

LPRINT " " or OUT 132,1

Otherwise, it may be necessary to edit out this character on the host system.

LLOAD

Function	LLOAD command allows the HUSKY to re-load programs through its serial port.
Syntax	LLOAD 'Control Enter' LLOAD.
Examples	LLOAD loads Basic source from the RS-232
Remarks	<p>Programs which were output using LLIST onto a storage medium, e.g. disk or another HUSKY, may be reloaded using this command. LLOAD is terminated by typing ESC. See Section 3.9 for a detailed description of program loading/unloading options.</p> <p>Be sure to check HUSKY's receiving parameters (Section 5.3) for compatibility with the device transmitting data.</p> <p>LLOAD data goes into memory via the Basic interpreter input routine and is syntax checked on the way - if errors or spurious data are found in the incoming data, HUSKY will stop loading and display:</p> <p style="padding-left: 40px;">* Syntax Error</p> <p>If this occurs at the start of a file, it is likely that some header data, not in Basic syntax, has caused the problem.</p> <p>Simply type LLOAD again, quickly, as HUSKY's input buffer will still be receiving data.</p> <p>To facilitate high speed loading the command:</p> <p style="padding-left: 40px;">LLOAD.</p> <p>may be used. This will not echo incoming text on the screen and cannot be stopped with ESC.</p> <p>NOTE: A very useful shorthand for LLOAD permitted in Husky Basic is 'Control Enter'. Simply press 'Control', followed by 'Enter', together.</p>

LN

Function **LN(X)** generates the NATURAL logarithm of X, i.e: logarithm to base e of X.

Syntax LN(N)
where N is a numeric expression.

Examples LN(8)
generates the logarithm to base e of 8.
If S=9.12 then:
LN(S)
generates the logarithm to base e of 9.12.44
LN(-91)
generates a 'Magnitude Error' and programme execution will stop.

Remarks e is defined in section 3.3.4.8.
X must be greater than zero.
If X is negative or equal to zero then a 'Magnitude Error' will occur.
X may be an expression.

LOG

Function LOG(X) generates the logarithm to base 10 of X.

Syntax LOG(N)
where N is a numeric expression.

Examples LOG(32)
generates the logarithm to base 10 of 32.
If S=1.319 then:
LOG(S)
generates the logarithm to base 10 of 1.319.

LOG(-6)
generates a 'Magnitude Error' and programme execution will stop.

Remarks X must be greater than zero.
If X is negative or equal to zero then a 'Magnitude Error' will occur.
X may be an expression.

LOPCHR

Function	LOPCHR outputs characters to the RS-232 port which have their ASCII values defined as decimal argument(s).
Syntax	LOPCHR numerical exprssion 1, numerical expression 2...
Examples	LOPCHR 61,62 transmits =>
Remarks	This permits computing of characters and also control of special functions in receiving devices, such as printers. The argument should be 0 - 127. Arguments greater than 255 will give an argument error.

LPRINT

- Function** **LPRINT** causes **HUSKY** to output to the serial port the value of the number, expression or function supplied as an argument.
- Syntax** **LPRINT** [%] **STRING-OF-VARIABLES** [%]
- Where **STRING-OF-VARIABLES** is a list of variables, each variable being separated by a comma.
- Remarks** **LPRINT** 10
- outputs the number 10 to the serial port.
- LPRINT** SIN(X)
- outputs the value of sine of X to the serial port.
- Several arguments can be strung together, separated by commas.
- A comma following the last argument will suppress the carriage return following **LPRINT** and will allow further **LPRINT** statements in the program to output on the same line.
- Examples:
- LPRINT** 10,X,COS(T),
- displays 10 0 .9999999999777
- If X and T both equal to zero.
- Literal strings can be displayed by using quotes as a string delimiter.
- Example:
- LPRINT** "THIS IS A STRING"
- outputs 'THIS IS A STRING' to the serial port.
- LPRINT** formats can be altered by the following statements:
- LPRINT** % N %
- sets the number of decimal places equal to N, a number in the range 0-9. Rounding occurs up or down as appropriate.
- LPRINT** % Z N %

sets the number of trailing zeroes (or decimal places) equal to N , a number in the range 0-9.

PRINT % E %

sets all following LPRINT formats to scientific notation.

LPRINT % %

restores LPRINT format to normal.

MID\$

Function MID\$(string,x,y) returns a substring of given length starting at a given position.

Syntax MID\$("STRING",POS,LEN)

where STRING is a string expression POS is its start position, LEN is its length.

Examples 10 A\$="ABCDEFGH"
20 PRINT MID\$(A\$,3,4)

prints CDEF

Remarks The string, x or y may be expressions.

MID\$(A\$,0,3) = ABC
MID\$(A\$,4,0) = null string
MID\$(A\$,20,5) = null string

NEW

Function	NEW will re-initialise all HUSKY 's user memory and reset all program pointers.
Syntax	NEW
Examples	<pre> READY NEW are you sure (Y/N)? Y 94636 bytes available READY </pre>
Remarks	<p>All program lines are erased!</p> <p>NEW should, therefore, not be repeated if several programs are to co-exist in the same HUSKY.</p> <p>Always use NEW before loading a new user program into HUSKY to ensure the maximum use of space.</p> <p>NEW replies with:</p> <pre> Are you sure? (Y/N) </pre> <p>If any response to the confirmation message other than YES (Y) is made, then the function is aborted and existing programs are not affected.</p> <p>Following the 'Y' response, NEW displays the number of free bytes available before returning to BASIC.</p>

NEXT

FUNCTION **NEXT**

Syntax **NEXT [V]**
V is optional and is a variable name that corresponds to the original FOR statement.

Examples **NEXT I** or **NEXT**

Remarks It is quicker without the variable.

ON BREAK

- Function** **ON BREAK** allows the user to interrupt processing.
- Syntax** **ON BREAK GOTO LINE**
 where line is a line number,

 or

 ON BREAK OFF
- Examples** **ON BREAK GOTO 1000**
 ON BREAK OFF
- Remarks** This statement operates in a similar manner to the **ON POWER** statement, except that the break key is monitored instead of the power key.

ON COMMS

Function	ON COMMS On Comms allows communication errors (typically a protocol failure) to be handled by the user's application program.
Syntax	ON COMMS GOTO LINE where line is a number or ON COMMS OFF
Examples	ON COMMS GOTO 1000 ON COMMS OFF
Remarks	<p>The statement operates in a similar manner to ON POWER.</p> <p>When a communications error is detected, control is returned to the Basic interpreter. Execution continues from a defined line number specified in the Basic program.</p> <p>In order to allow normal communication error messages the statement ON COMMS OFF disables this mode. See Section 5.9 for details of error messages. The byte of memory COMERR (4540H - decimal 17728) is available to the programmer and contains a code which represents the type of error which has occurred in the communications. The available codes and their meaning are described more fully in Section 5.9.</p> <p>In a typical application, ON COMMS would be used to present the Operator with a plain language statement of the form:</p> <p>"Communications have failed. Please re-dial and try again".</p> <p>Re-trys or repeat blocks can also be controlled by the user program, as required.</p>

ON ERROR

Function **ONERROR** is used to catch errors which would normally give a Basic Error message.

Syntax **ONERROR** LINE

where line is a line number.

ONERROR OFF

Examples

```
10 ONERROR 1000
20 INPUT A
30 A = SQR(A)
40 PRINT A
50 GOTO 20
1000 A =ABS(A)
1010 PRINT "NEGATIVE NUMBER CORRECTED"
1020 ONERROR 1000
1030 GOTO 30
```

This example will trap negative numbers and correct the condition.

Remarks **ONERROR** causes program execution to be diverted to an error handling subroutine.

Any error can be trapped. However, **ONERROR** is normally used for correctable errors e.g. magnitude errors which could be caused by operator input etc.

Each **ONERROR** line number is used only once and a previously defined **ONERROR** line will be used with the next error. **ONERROR** may be used within the error handler.

Line 1030 of the above program illustrates the necessity of restoring the **ONERROR** trap after it has been used.

NOTE: **ONERROR** line numbers are stored in one of **HUSKY**'s Basic stacks. Repetitive **ONERROR** definitions (in a **FOR...NEXT** LOOP, for instance) will result in stack overflow.

ON..GOSUB

- Function** ON...GOSUB... allows conditional subroutine calls.
- Syntax** On expression GOSUB line number 1, line number 2,line number n.
- Examples**
- ```
10 ON A GOSUB100,180
20 PRINT "HELLO"
```
- If A=1 then program execution will go to line 100 and will then go to line 20 when a RETURN is encountered.
- If A=2 then program execution will go to line 180 and will then go to line 20 when a RETURN is encountered.
- ```
10 B=100:C=300:D=350
20 ONAGOSUB B,C,D
30 PRINT "HELLO"
```
- If A=1 then program execution will go to line 100 and revert back to line 30 when a RETURN is encountered.
- If A=3 then program execution will go to line 350 and revert back to line 30 when a RETURN is encountered.
- Remarks**
- ON GOSUB will cause program execution to go to a line number chosen from the list of line numbers positioned immediately after GOSUB. The selection of a particular line number in this list is determined by the value of [expression].
- Once program execution has been forced to go to one of the selected line numbers, each statement following that line number will be executed in order until a RETURN is encountered. At this point program execution will revert back to the statement following the list of line numbers.
- The integer part of [expression] is used so that if A=2.9 program execution will go to line 180 in the example.
- NOTE:** The value of [expression] is **NOT** rounded up.
- If the value of [expression] is zero or greater than the number of line numbers in the list, then program execution will continue with the next statement.
- In the above example, if A=3 then "HELLO" will be printed **IMMEDIATELY**.
- The maximum value of [expression] allowed is 255.

If the value of [expression] is greater than 255 or negative then a 'Magnitude error' will occur.

The line numbers in the list of line numbers may themselves be expressions.

ON..GOTO

Function	ON...GOTO... allows conditional jumping.
Syntax	On expression GOTO line number 1, line number 2,line number n.
Examples	<pre>10 ON A GOTO100,200,300 20 END</pre> <p>If A=1 then the program will go to line 100 If A=2 then the program will go to line 200 If A=3 atthen the program will go to line 300</p> <p>If A=1.3 then the program will go to line 100.</p>
Remarks	<p>ON GOTO will cause program execution to jump to a line number chosen from the list of line numbers positioned immediately after GOTO. The selection of a particular line number in this list is determined by the value of [expression].</p> <p>The line numbers may themselves be expressions.</p> <p>The value of [expression], in the above example A, is truncated to an integer.</p> <p>It must be noted that the value is truncated to an integer and NOT rounded up. So if the above example A=1.9 then the program will go to line 100.</p> <p>If the value of [expression] is zero or greater than the number of line numbers in the list, then program execution will continue with the next statement.</p> <p>In the above example, if A=4, or if A=0 then the program will automatically go to line 20, and an END will be executed.</p> <p>The maximum value of [expression] allowed is 255.</p> <p>If the value of [expression] is greater than 255 then a 'Magnitude Error' will occur and program execution will stop. If in the above example A=256 then an error will occur. A 'Magnitude Error' will also occur if the value of [expression] is negative.</p> <p>Example:</p> <pre>10 ONAGOTO 100,200,300:PRINT "HELLO" 20 END</pre>

If $A > 3$ or $A = 0$ then the program will not go to any of the lines but will immediately print "Hello", and then END will be executed.

ON POWER

Function ON POWER allows the programmer to trap and vector program execution if a user presses the power off key.

Syntax ON POWER GOTO LINE
where line is a line number,

or

ON POWER OFF

Examples ON POWER GOTO 1000

Remarks This statement causes the OFFVECT (See Section 4.1.4) locations to be loaded with a vector so that if operation of the power down key is detected, program execution occurs from the line number specified.

Operation of the break key, or a syntax error, will cause the vector to be re-initialised to enable the power on/off switch to function normally.

ON POWER OFF

This statement also re-initialises the OFFVECT location.

OPCHR

Function	OPCHR outputs characters to the screen which have their ASCII values defined as decimal argument(s).
Syntax	OPCHR numerical exprssion 1, numerical expression 2...
Examples	OPCHR 61,62 displays => OPCHR1 clears the screen.
Remarks	This permits computing of characters and also advanced functions such as cursor addressing. The argument should be 0 - 127. Arguments greater than 255 will give an argument error.

OUT

- Function** **OUT** outputs a byte of data to a specified port.
- Syntax** **OUT** address, data 1, data 2....data n.
- Examples** **OUT** 192,5,10
 Outputs binary 5 and 10 to the parallel ports 192 and 193 respectively.
- Remarks** **OUT** is used to control internal **HUSKY** functions like power control. See part 4 Advanced Programming, section 10 for details. With the optional parallel port, **OUT** is used to write 8-bit data to external devices.

PEEK

- Function** PEEK fetches the value of a designated byte in HUSKY's memory.
- Syntax** PEEK (ADDRESS)
where ADDRESS is a numeric expression.
- Examples** PRINT PEEK (17933)
prints the value of location 17933 (in decimal) on the LCD screen.
- Remarks** PEEK requires the decimal value of the required memory location as an argument.
See Part 4, of Advanced Programming, Section 1 for details of PEEK-able locations.

POKE

Function POKE allows the user to write to a specified location in HUSKY's memory.

Syntax POKE address, data 1, data 2,....data n.

Examples POKE 17724,1

Pokes the value 1 into location 177224 (453C Hexadecimal).

Remarks This is the address of 'NYMPHO', the flag that inhibits HUSKY's power off key. Try it!

Restore the situation with:

```
POKE 17724,0
```

and all is well.

The data can be a variable or an expression and must be in the range 0-255. Values greater than 255 will give an argument error.

POKE is primarily used for advanced control of HUSKY, and is fully discussed in Part 4, Advanced Programming, Section 1.

WARNING Use POKE with care, and only as specified.

POP

- Function** POP fetches (POP's) a value from the machine code subroutine linkage stack.
- Syntax** POP(N)
where N=number of parameters to be POPPED.
- Examples** A=POP(2)
POPs two values of the machine code linkage stack.
A=latter parameter.
- Remarks** Allows Basic to access 16 bit results from machine code subroutines.
Refer to Section 3.6.4 which discusses machine code calls.

PRINT

Function PRINT causes HUSKY to display on the LCD screen the value of the number, expression or function supplied as an argument.

Syntax PRINT variable 1, variable 2, variable 3,
 PRINT "literal string", variable,
 PRINT expression 1, expression 2,
 ? variable, "literal string", expression

Remarks PRINT may be represented by "?" for shorthand purposes. Typical examples:

```
PRINT 10
or
?10
```

displays the number 10 on the screen.

```
PRINT SIN(X)
```

displays the value of sine of X on the screen.

Several arguments can be strung together, separated by commas.

A comma following the last argument will suppress the carriage return following PRINT and will allow further PRINT statements in the program to output on the same line.

Example:

```
PRINT 10,X,COS(T),
```

displays 10. 0 .9999999999777

If X and T both equal to zero.

Literal strings can be displayed by using quotes as a string delimiter.

Example:

```
PRINT "THIS IS A STRING"
```

displays THIS IS A STRING on the LCD screen.

Also note that strings may be printed with embedded control characters (hold the control key and press an alphabetic character).

Example:

? "Control G" sounds the internal bleeper.

In practice, OPCHR or CHR\$(X) may be found more convenient for these functions since subsequent printer listings are unlikely to reveal control codes embedded in programs.

PRINT formats can be altered by the following statements:

PRINT % N %

sets the number of decimal places equal to N, a number in the range 0-9. Rounding occurs up or down as appropriate.

PRINT % Z N %

sets the number of trailing zeroes (or decimal places) equal to N, a number in the range 0-9.

PRINT % E %

sets all following PRINT formats to scientific notation.

PRINT % %

restores PRINT format to normal.

PUSH

- Function** **PUSH** pushes a variable or constant onto **HUSKY's** machine code subroutine linkage stack which is generally used with machine code **CALLS**.
- Syntax** **PUSH** variable 1, variable 2,variable n.
- Examples** **PUSH** A,B
- pushes the integer (16 bit binary) representation of the variables A and B onto the stack.
- Remarks** These variables may be accessed or modified by a machine code program. See section 4.9 on machine code programming.
- POP** is the reverse of **PUSH**.

READ

Function **READ** is used to input constants previously defined in a **DATA** statement.

Syntax **READ** STRING OF VARIABLES

Where string of variables is a string of variables each separated by a comma.

Examples **READ** X

sets X equal to the current **DATA** constant.

For example the program:

```
10 DIM D!(10):DIM A$(10,10)
20 DATA 3,4,5,2/3,"HELLO"
30 READ A,B,C,D!(0),A$(1)
40 PRINT A,B,C,D!(0),A$(1)
```

Prints:

```
3 4 5 .666666666666666 HELLO
```

with A,B,C equal to three sequential **DATA** constants, D!(0) a double precision constant and A\$(1) a string expression.

Remarks The constants are read sequentially until exhausted, when an error will result. The sequence may be restarted by using **RESTORE**.

The variable type in **READ** **must** be the same as the corresponding **DATA** items: if not, a **READ ERROR** will occur.

REM

- Function** **REM** (REMark) denotes that the following text is a comment statement only.
- Syntax** **REM** COMMENTS
- Examples** 10 **REM** THIS IS A COMMENT

 REM is also used for storage of **HELP** text. See 3.3.2.7
- Remarks** **REM** statements are ignored by the program.

 Each character or space in a comment line occupies one byte of memory.

 Unnumbered **REM** statements can be used in host computer storage files for additional comment lines: these will be ignored by **HUSKY** when the file is loaded.

RESTORE

- Function** **RESTORE** is used with **READ**. **RESTORE** initialises the current **DATA** pointer to the first item in the list.
- Syntax** **RESTORE** line number.
- Remarks** **RESTORE** may also be used with a line number argument. This will initialise further **READ** functions to a specified line.

RETURN

Function **RETURN** from subroutine

Syntax **RETURN**

Examples 10 GOSUB 100

 100 PRINT I
 200 RETURN

Remarks **RETURN** is the logical conclusion of a **GOSUB** statement. Placed at the end of the subroutine, **RETURN** causes execution to continue at the next statement following the original **GOSUB** statement.

RIGHT\$

Function `RIGHT$(String,n)` returns the last `n` characters of a given string.

Syntax `RIGHT$("STRING",N)`

where `N` is the number of rightmost characters to be returned.

Examples `RIGHT$("ABCDEF",3)`
returns DEF

Remarks Both string and `n` must be enclosed in parenthesis and can be expressions. If length of the string is less than `n` then the entire string is returned.

 If `A$ = "ABCDEF"`

 then:

`RIGHT$(A$,0) = null string`

`RIGHT$(A$,20) = ABCDEF`

RND

- Function** RND returns a 14-digit psuedo random number in the range 0-0.99999999999999.
- Syntax** RND(0)
where "0" is a dummy argument.
- Examples** A = RND(0)
sets A equal to a random number.
- Remarks** RND requires a dummy argument.

RUN

Function	RUN causes program execution to commence.
Syntax	RUN RUN line number RUN numerical expression
Examples	RUN 1000 causes execution to commence from line 1000.
Remarks	RUN is useful when program de-bugging as it initialises the system stacks; GOTO does not. The line number may also be a variable or an expression, for example: RUN A or RUN A*100 are equally valid.

The logo for the SGN function, consisting of the letters 'SGN' in a white, sans-serif font, centered within a solid black rectangular background.

Function SGN returns the value of the sign of the argument.

Syntax SGN(N)

where N is a numeric expression.

Examples A = SGN(-9.5)

sets A equal to -1

A = SGN(0)

sets A equal to 0

A = SGN(1.76E9)

sets A equal to 1

Remarks Result:

1 if Positive
0 if Zero
-1 if Negative

SIN

- Function** **SIN** returns Sine of the argument.
- Syntax** **SIN(N)**
where N is a numeric angle in radians.
- Examples** **J = SIN(2.56)**
sets J equal to the value of the sine of 2.56 radians.
- Remarks** The argument is expressed in radians.
To convert to degrees the $A = \text{SIN}\left(\frac{30}{100} \times 3.1415926\right) \text{ PI}$
 $A = \text{SIN } 30 \text{ deg.}$

SQR

Function **SQR (Expression)** returns the square root of the argument.

Syntax SQR(N)
 where N is a numeric expression.

Examples Q = SQR(16)
 sets Q equal to 4.

Remarks The argument must be positive.

SRCH

Function SRCH function enables an array to be searched for the occurrence of a specified element within that array.

Syntax SRCH(<array element>,<numerical expression>)

The array element specified is used as the reference to be searched for in the remaining array.

Examples

```
PRINT SRCH(A$)
A$=SRCH(A(5),3)
IF SRCH(A!(0),1)<>5 THEN OPCHR7
```

Consider for example the following arrays.

```
A$(0)="ABCDE"
A$(1)="AB"
A$(2)="ABCDE"
A$(3)="ABCDE"
A$(4)="XYZ"
A$(5)="ABCDE"
```

Remarks When found the element number is returned. If not found then zero is returned.

The function searches from the specified element to the end of the array. As an option, up to 255 occurrences can be checked for by specifying in the verb the number of repeat occurrences.

If this value is 256 or larger a syntax error will occur.

The result of SRCH is a numeric value which can be used in arithmetic expressions etc.

The search function can also be used on simple or double precision arrays.

The following are the results of search on the above array:

SRCH(A\$) gives 2 : as element (2) is the first occurrence of element (0).

SRCH(A\$,2) gives 3 : as element (3) is the second occurrence of element (0).

SRCH(A\$(2),1) gives 3 : as element (3) is the first occurrence of element (2).

SRCH(A\$(4)) gives 0 : as element (4) is not found elsewhere.

When search is applied to numeric arrays the comparison must be exact for equality.

For string arrays the rules apply as in the 'IF' statement, i.e. the comparison is equal if the reference string is equal to, or a subset of, the searched string.
Example:

```
PRINT SRCH(A$(1))
```

would return 2 as A\$(1) is a subset of A\$(2).

STEP

Function	STEP
Syntax	STEP N where N is the STEP increment.
Remarks	Can be positive or negative. STEP is a part of the FOR command.

STOP

Function	STOP terminates program execution and prints a message on the LCD screen.
Syntax	STOP
Examples	550 STOP causes: *BREAK IN LINE 550*
Remarks	Typing CONT causes execution to continue from the line following STOP

STR\$

Function **STR\$(Expression)** converts a numeric expression to a string.

Syntax **STR\$(N)**

where N is a numeric expression.

Examples

```
10 A = 123.6
20 B$ = STR$(A)

sets B$ = "123.6"

10 A = 7.941E12
20 B$ = STR$(A)

Sets B$ = "7.941E12"
```

Remarks This function permits the use of string operations on numbers.

Exponential notation numbers are represented literally in string form.

TAB

- Function** **TAB** allows neat columns of figures, etc., to be printed.
- Syntax** **TAB**(N)
 where N is the PRINT POSITION to begin PRINTING.
- Examples** 100 LPRINT **TAB** (10),X

 will print the value of X leaving 10 columns from the left hand border.
- Remarks** In an LPRINT statement **TAB** specifies how many columns to skip before starting to print.

 If the value of the numeric expression is less than the print position, then **TAB** is ignored. The value of expression should not exceed 255 or be negative.

 Due to leading space suppression in **HUSKY**'s display handler (necessary to optimise use of the 32 character lines), **TAB** can be confusing in PRINT statements. A character in the first column position will allow **TAB** to be used with the LCD display.

TAN

- Function** **TAN (expression)** returns the tangent of an angle.
- Syntax** TAN(N)
 where N is a numeric in radians.
- Examples** T = TAN(2.56)
 sets T equal to -.657453 (tangent of 2.56 radians)
- Remarks** The argument is expressed in radians.
 To get tangents of angles in degrees then convert:
 A=TAN (Angle x 3.14159269) PI
 180

VAL

Function **VAL(String)** performs the inverse of **STR\$**. It converts a string of digits into a number.

Syntax **VAL ("STRING")**
where string is a string expression.

Examples 10 A\$ = "123.6"
 20 B = VAL(A\$)

 sets B to 123.6

Exponential notation numbers are converted literally.
Example:

 10A\$ = "2.91E6"
 20 B = VAL(A\$)

 sets B = 2.91E6

VER

Function	VER reports the version number of the Basic interpreter implemented in this HUSKY .
Syntax	VER
Examples	VER HA29JUN82

The logo for the WINCHR function, consisting of the word "WINCHR" in a white, sans-serif font centered within a solid black rectangular box.

- Function** WINCHR inputs a single character from the optical wand.
- Syntax** WINCHR N
where N is a numeric variable.
- Remarks** This function acts exactly like INCHR, but with the optional optical wand. The value is returned from the optical bar code exactly like a keyboard input.
- If a multiple character bar code is scanned then WINCHR will return the first character and the rest of the code will be ignored.
- As with WINPUT if any key is pressed prior to the bar code input, then the input is taken from the keyboard instead.

WINPUT

- Function** **WINPUT** This function acts exactly like **INPUT**, but with the optional optical wand.
- Syntax** **WINPUT** prompt, variable. (See also **INPUT**)
- Examples** **WINPUT A\$**
 INPUTS WAND DATA INTO A\$
- Remarks** The string returned from the optical bar-code is read exactly like a keyboard input. If any key is pressed prior to bar-code input then the input will be taken from the keyboard instead.
- The bar code used is selected from the special functions menu, details of which may be found in section 6.4.1.
- The use of string variables is recommended to allow for any alpha content.

This page intentionally left blank

This page intentionally left blank

ERRORS AND WARNINGS

- 3.5
- 3.5.1 *Argument error
Invalid argument.
- 3.5.2 *Array error
Attempt to use an element of an undefined array.
- 3.5.3 *Break
Detection of a manual 'ESC' key operation (**HUSKY** not in auto-start) or execution of a 'STOP' instruction.
- 3.5.4 *Control Stack error
Incorrect RETURN or NEXT statement or GOSUB without RETURN or FOR without NEXT.
- 3.5.5 *Dimension error
Attempt to re-define an existing array variable. (DIM Statement)
- 3.5.6 *Direct Input error
Caused by typing NEXT X, for example, which is meaningless except as a numbered program line.
- 3.5.7 *Floating Point error
Value of constant or variable is irrational.
Example:
$$A = A/0$$
- 3.5.8 *HELP SOURCE ERROR
The reference HELP text line does not exist.
- 3.5.9 *Line Number error
Non-existent line number referred to.
- 3.5.10 *Line Overflow error
More than 96 characters in one line.
- 3.5.11 *Magnitude error
Attempt to access an array element that is outside of range defined in the corresponding DIM statement.
- 3.5.12 *Negative Square root
Negative numbers have no root!
- 3.5.13 *Read error
Invalid data present in a DATA statement. (Variable type and Data constant type are not the same).

- 3.5.14 *Storage Overflow error
Variable storage or DIM reservation has overflowed available memory space.
- 3.5.15 *Syntax error
Incorrect line.
- 3.5.16 *System Stack error
Attempt to POP a variable from the stack which has not previously been stored by PUSH.
- 3.4.17 *String complexity error
Attempt to evaluate a complex string expression which has exceeded the expression stack. To correct this error simply break the expression down into simpler parts.
- 3.5.18 *String pool overflow error
Insufficient space to manipulate the string expression.
- 3.5.19 *Type mismatch error
Invalid equate attempt.
Example:
A = A\$

3.6

POWER WARNING

When **HUSKY's** batteries become exhausted a power warning message will appear on the screen. Normal operation will continue, but keyboard entries will be punctuated by warning tones and repetition of the power warning message until the battery state is rectified either by replacement of primary cells or recharging of secondary cells, if installed.

Evenutally, if the warnings are ignored, the **HUSKY** will shut down and refuse to operate further.

Every **HUSKY** has to pass a stringent operating test in this low power regime. However, it is strongly recommended that operator training procedures and user programs are structured to avoid continued operation when warnings become persistent, especially when rechargeable cells are used.

The power warning sequence may be misleading if RS232 communications are used infrequently. When **HUSKY's** communications system is activated power drain increases significantly. This increase can cause the power warning stage to be missed altogether, and instead cause the **HUSKY** to shut down without warning.

While this is not hazardous, it can be very confusing for an operator who does not know what is wrong. Because of this it is recommended that user programs with infrequent communication requirements use a test routine. This activates the communications package, and then tests the power state before restoring the **HUSKY** to normal. This routine is used at the start of every data entry sequence.

A sample warning program of this kind is given below.

If batteries become exhausted whilst communications are in progress, then normal power warning messages will appear once the keyboard operation is resumed.

NOTE that power warning messages only occur when keyboard entries are in progress.

SAMPLE WARNING PROGRAM

```
20 OUT132,1
30 J=INP(2):OUT132,0
40 I=J-8*INT(J/8)
50 IF I>3 THEN 100
60 OPCHR1
70 PRINT"PLEASE CHANGE BY BATTERIES!"
80 OPCHR7,7,7,7,7,7,7,7
90 OUT131,0
100 RETURN
```

This program operates as follows:

- 1) The power status is available as bit 3 in **HUSKY** port 2. 1 = power OK, 0 = Low Power state.
- 2) Line 20 energises the RS-232 serial interface.
- 3) Line 30 reads the power state from port 2, and then closes down the interface to conserve power.
- 4) Line 40 extracts the power status bit.
- 5) Lines 50-90 Display a warning and ring **HUSKY'S** bell!

3.7

MACHINE CODE CALLS

HUSKY is designed to be extremely versatile and flexible in its use. For some applications it is not possible to perform every function using pure BASIC routines, usually for reasons of speed. It is possible to write machine code routines in these situations.

Also covered in this section is some general guidelines in the control of **HUSKY** by means of PEEK and POKE to specified memory addresses.

It should be emphasised that the use of features described here is not recommended for users unfamiliar with computing at machine code level.

3.7.1 The Machine Code

HUSKY uses the CMOS NSC-800 microprocessor, which provides an instruction set entirely compatible with the popular Z80 microprocessor. For that reason it is not proposed to go into detail about the facilities offered by machine code and users are referred to any of the standard books which cover the Z80.

It is not recommended, however, that the stack pointer register (SP) is either moved or used in a fashion incompatible with supporting interrupts. Standard use in the form of subroutine CALLs, PUSHes and POPs, however, is supported.

3.7.2 BASIC CALL

To activate a user or system program at a known address the function:

CALL (addr)

is used. This will pass control from BASIC to the address. A machine code RET will return the user to BASIC. The most common routines to be used will be the system calls for reading the clock, etc., which are fully documented in Part 4, Advanced Programming.

3.7.3 Passing Simple Parameters

It is possible to pass information both from BASIC to the 'called' program and return information back to BASIC.

To set up outgoing data the ARG function is used, format as shown. The variable will be equated to the value in brackets.

A = ARG(5)

will cause the NSC800 internal registers C and E to be set up as follows:

```
C = 5
E = 0
```

The pair are set up as a 16-bit integer, hence:

```
ARG (10*256+30)
```

causes:

```
C = 30 and:
E = 10 (decimal)
```

when CALL is executed.

These registers are used for compatibility with the "CP/M" style system call structure.

To return data to BASIC the line:

```
1000 X = CALL (addr)
```

causes the value of variable X to take on the value contained in register A.

3.7.4

Passing Multiple Variables

To pass more than one integer between a called program and BASIC the machine code linkage stack is used.

This is controlled in BASIC by PUSH and POP. In the routine:

```
1000 PUSH X,Y
1010 A = CALL (addr)
1020 Y = POP(1)
1030 X = POP(1)
```

The integer values of X and Y are first placed on the stack and made available to the called routine, and then removed by the POP instruction back into their own variables.

When CALL is executed the register pair HL is used as an address pointer to the bottom of the stack on which the variables are placed:

```
      X HIGH
      X LOW

      Y HIGH
HL, Y LOW
```

In other words HL is addressing the low byte of the most

recently PUSHED variable. To access others then HL is incremented. These values may either be used by the called routine or modified and used by BASIC. It is not necessary to preserve the contents of HL.

3.7.5

Controlling HUSKY

Part 4, Advanced Programming, Section 2 contains a list of available addresses of **HUSKY** parameters and system call addresses. The parameters may be written or read by means of PEEK and POKE operations. System calls may be used for functions such as the CLOCK or communications parameters.

An example of a program to read the clock is in the program examples.

3.8

PROGRAMMING TECHNIQUES

To the programmer, most of **HUSKY's** features are likely to seem familiar and quite comparable with many other, less portable, microcomputers.

To the operator, **HUSKY** is likely to be quite different from anything he's encountered before. This fundamental divergence in experience can present a major challenge to the skill of the programmer.

Because **HUSKY** operators tend to be newcomers to computer techniques, and worse, tend to capitalise on **HUSKY's** unique physical characteristics by using it far away from the comfort of the computer room, considerable demands are placed on the quality of the programming. These demands are met by making programs as error-free and 'bullet-proof' as possible, together with careful program structures. A **HUSKY** that interrupts a data entry sequence in the field with "Magnitude Error in Line....." is not likely to be appreciated by the user, or worse, by his customers.

Knowing that field operators would have difficulty in recovering from programming failures on the spot, **HUSKY's** designers have provided a number of facilities that help overcome these difficulties. But because **HUSKY's** programming has to have greater integrity than is ever required at the desk-top, there is no substitute for methodical discipline in programming.

The contents of this section explain some techniques that are successfully used to provide reliable and ergonomically friendly user programmes.

3.8.1

Data Capture Techniques

A typical **HUSKY** application program consists of 3 segments:

- A 'Data Capture' segment
- An 'Inspection' segment
- A 'Transmission' segment

All three segments are contained in a common program although treated as independent modules. All share a common database, the 'captured' information, generally stored in array structures.

The operator (as opposed to the programmer) is given a limited range of options within this framework and **never has access to the Basic interpreter**. On power-up, **HUSKY** will typically present a menu-style choice of options, often based on the segments or modules themselves.

Once a module is selected, **HUSKY** will lead the operator through a question and answer sequence until his objective is achieved. Consider the following data capture program:

```

10  REM TELEPHONE NUMBERS
20  REM THIS PROGRAM CAPTURES AND STORES NUMBERS
30  REM UP TO 6 DIGITS LONG.  THEY CAN THEN BE
40  REM RECALLED OR LISTED ON A PRINTER.

100 REM MODULE 0 WHICH FUNCTION?
110 PRINT "PLEASE CHOOSE A FUNCTION, TYPE '1' TO ENTER
    DATA; '2' TO INSPECT OR '3' TO TRANSMIT",
120 INPUT A
130 IF A=1 THEN 200
140 IF A=2 THEN 300
150 IF A=3 THEN 400

200 REM MODULE 1 DATA CAPTURE
210 DIM D(10)
220 FOR N=1 TO 10
230 INPUT D(N)
240 NEXT N
250 GOTO 100

300 REM MODULE 2 INSPECT DATA
310 PRINT "WHICH NUMBER DO WANT TO INSPECT?"
320 PRINT "PLEASE ENTER A NUMBER 1-10!",
330 INPUT N
340 PRINT D(N)
350 GOTO 100
400 REM TRANSMIT THE DATA
410 FOR N=1 TO 10
420 LPRINT D(N)
430 NEXT N
440 GOTO 100

```

This very simple program demonstrates the basic features of much more complex data capture routines, but has one dramatic failing: it is not "bullet proof".

If the operator does not stick precisely to the sequence laid down, the program will soon encounter a problem and resort to error messages that will not help a non-programmer. Because of this, the great majority of effort in programming Husky applications is devoted to preventing occurrences that might confuse the non-technical.

HUSKY's Basic interpreter contains many features designed to prevent the inexplicable happening in the field: but complex programs can sometimes outwit even their own authors!

A simple solution to this problem is simply to add three more lines:

```
50 ONERROR 500
500 PRINT "THAT CAN'T BE RIGHT - PLEASE TRY AGAIN"
510 GOTO 100
```

Now, all the error messages from the interpreter are intercepted and re-directed. The program simply tries again. This is fine, but the problem remains: the errors shouldn't be there in the first place!

3.8.2 Data Storage Arrays

3.8.2.1 Types of Arrays

Captured data is generally held in array structures created within HUSKY's very large memory. There are three types of array:

- Simple Variables
- Double Precision
- Strings

Simple Variables are used for storage of decimal values of up to 6 digit accuracy.

Double Precision arrays store numbers up to 14 digit accuracy, but use more memory.

NOTE: both these types store **much larger** numbers than indicated, but only by truncating the least significant digits. For example, the number 12345678 entered as a simple variable would be stored and reproduced as 12345600.

String Arrays store every character, whether numeric or otherwise, literally. They can have any number of characters (up to a maximum of 255 characters in each string). Facilities are provided for converting strings to numbers and vice versa.

3.8.2.2 Array Structures

Every array is denoted by a **name** and placed in memory in a sequential table. Multiple arrays are packed in memory by Basic and accessed via a **Symbol Table**.

Individual array elements are identified by the array name and the element number, together with a character to identify the array type. For instance: (**NOTE: arrays start from element 0**) A(10) is the eleventh element of simple variable array A.

A!(2) is the third element of double precision array A! (quote different and independent from array A in the example above).

A\$(0) is the first element in string array A\$. A\$(0) is identical with A\$, which can be used for shorthand-useful if there are multiple references to a single string in the program.

In each case, the number in parenthesis is the **element number**, and can be variable:

A\$(A)

or an expression:

A\$(A*5)

or another array element:

A\$(6)

Remember the element numbers start from 0!

3.8.2.3 Creating Arrays

HUSKY's designers have decided that all arrays used in application programs must be **explicitly declared in advance** to avoid overflow problems in the field, (much) later. Arrays are created, and memory space reserved, using DIM statements.

Arrays can only be created once: if a subsequent attempt is made to re-define an existing array, an error occurs. This is to warn the programmer that he may be accidentally duplicating an array name.

NOTE: String and double precision variables **must** have array space declared, even if there is only one element in use. For example, the statement:

```
A$="123ABC"
```

is only valid if the statement:

```
DIM A$(1,6)
```

has already been executed.

Otherwise, an error message:

```
*Array Error
```

will appear.

Also, if the statement DIM A\$(1,6) is repeated at **anytime** then an error message:

```
*Dimension Error
```

will appear.

This can be easily avoided by incorporating the DIM in a program sequence that is only executed once, and subsequently branched around. This is often called an **Initialisation Sequence**. A typical format is:

```
10 IF Z<>0 THEN 100
20 DIM A$(1,6)
30 Z=55
```

The variable Z is a flag, indicating that the program has been run before. (All variables are cleared to 0 by CLEAR or any change to program content, but not by power off or removing batteries).

A slightly more sophisticated version is:

```
10 IF Z<>0 THEN 100
20 DIM A$(1,6)
100 Z=Z+1
```

Where Z acts both as a flag and as a counter of the number of times the program has been run, a very useful statistic!

3.8.2.4 Array Sizes

Arrays are limited in size only by **HUSKY's** memory space. Remember that arrays compete with program for space, automatically allocated by **HUSKY's** operating system. Since the maximum number of single precision simple variables possible in a 144K **HUSKY** is approximately 28,000, array sizes will be less than this number.

Array size allocation is simplified by **self-sizing**. For this purpose, the Function FRE (Section 3.3.4.8) can be used as a variable. FRE provides the number of memory bytes remaining as follows:

After NEW : The total memory available
 After program entry : The memory available for data
 After array definition : The memory remaining

The number FRE can be used to automatically calculate the maximum number of array elements possible for any given array type and to sign on informing the operator of this information. Remember that a fixed space of 256 bytes is required for the **String Pool** if string arrays are to be used. An example for a simple variable array is:

```

10 IF Z<> THEN 100
20 N=2
30 Y=INT (((FRE(0)-7-N*7)/5)-1)
40 DIM A(Y)
100 PRINT "TOTAL NUMBER OF ELEMENTS AVAILABLE:",Y

```

Note that Y is taken as an integer (INT), that the symbol table entry (-7) is subtracted, that the element size is 5 bytes and that arrays start from zero, so the maximum number is one less (-1).

N*7 is an allowance for the number of variables used in the subsequent program, assuming 7 bytes per variable. Since both N and Y are variables, N must equal 2.

NOTE: problems can occur if more variables are used than space is provided for, since memory overflow is not checked when new variables are declared.

A string array can be sized as follows:

```

10 IF Z<>0 THEN 100
20 N=2
30 Y=INT (((FRE(0)-7-N*7-256)/15)-1)
40 DIM A$(Y,4)
100 PRINT "TOTAL NUMBER OF STRINGS AVAILABLE:",Y

```

3.8.2.5 Accessing Array Elements

Each element in a data array is accessible by its **array index**, expressed in parenthesis, e.g:

A#(235)

is the 236th element in A#

NOTE: array indexes **must** be within the range declared in the corresponding DIM statement. **HUSKY BASIC** warns the programmer of any attempt to access an element outside the declared range with:

*Magnitude Error

Be especially careful with incrementing indexes like FOR NEXT loops. For instance, the routine:

```
10 DIM A(10)
100 FOR N=1 TO 10
110 INPUT A(N)
120 NEXT N
```

is **not** OK, since there is no A(10). The program should read:

```
10 DIM A(10)
100 FOR N=0 TO 9
110 INPUT A(N)
120 NEXT N
```

Array indexes often have a direct relationship to external variables, like stock codes. It is always preferable to **directly access** an array in this fashion in the interests of speed, rather than working sequentially through every element.

If there are discontinuities (gaps) in the existing codes, or if the numbers are too large, a conversion table or algorithm may be needed. Since a literal table (one entry for every code) is likely to be very inefficient, it is generally worth putting considerable effort into deriving an efficient 'tree' structure for direct array access.

3.8.2.6 Array Searches

There is often a need to search an array for an element of known content. In Basic, this procedure can be painfully slow.

One solution to this dilemma is to use a specialised machine-code subroutine to search the array and return with the index of the target element.

A much more practical method is the Basic function SRCH (See section 3.4.67), which is not affected by Husky's paging structure.

3.8.3 **Data Input Techniques**

We are all familiar with the computer truism:

"Garbage in - garbage out"

Never was more true than in data capture applications where out-of-range entries can cause fatal errors that are just embarrassing in the office, but potentially disastrous in the field.

To help avoid the commonist forms of data entry error, **HUSKY** provides a unique facility **INPUT USING** (Section 3.3.2.31). Input using pre-filters incoming data against a predetermined mask and rejects entries that do not fit at the keyboard, warning the operator to try again.

Equally important to the user is optimum use of **HUSKY**'s screen to provide clear, unambiguous prompt messages and input fields.

The example program in section 3.8.5.6 inputs simple message strings and holds them in a string array. The program first prompts:

Please type message length (29 characters maximum):_

To which the user responds with a number of characters, for instance '12'.

HUSKY then responds with:

Total number of entries possible in this **HUSKY** is..... and then:

Please enter prompt message (up to 17 characters):_

To which the user responds with a prompt. This might be, for instance:

"Enter next name"

The program then requests entries in sequence, drawing an entry field on the screen as it does so:

ENTER NEXT NAME []

INPUT USING keeps all entries within the field.

3.8.4 USING HUSKY'S SCREEN

HUSKY's large, 32 x 4 character, LCD screen can be used to positive effect to help its operator. Because the screen is entirely flexible, these notes are provided as a source of ideas rather than as instructions.

This section refers to character-by-character use : more detailed information on direct dot addressing is available in Section 4.3

3.8.4.1 Word Justification

HUSKY's screen control software automatically maximises screen use and legibility by suppressing unwanted spaces and justifying whole words that occur at the end of lines by moving them, intact, to the next line.

These features are the key to using **HUSKY** as a terminal for remote mainframes, where display data is intended for 80 column or 132 column screens or printers. **HUSKY's** screen software allows these formats to be read 'on the fly', avoiding the need to re-configure the mainframe software.

When used with BASIC user programmes, word justification is sometimes confusing at first. Text will not 'wrap around' the end of lines as it would in a simple printer : neither will leading spaces serve to position a message at the centre of the screen.

3.8.4.2 The HUSKY Screen

HUSKY's screen is divided into 4 lines of 32 characters each. Any ASCII character can be written to any of the 128 character positions.

Character positions are denoted by X,Y co-ordinates as shown in Fig.3.1. Note that the Y co-ordinate counts down from the top.

NOTE: The right-hand bottom corner position (31,3) is normally occupied by the 'shift arrow'. Attempting to write to this position simply causes the entire screen to scroll upwards, since this is the last character on the screen.

In addition to the 7 x 5 dot matrix at each position, a separate cursor appears under a specified position. The cursor can only occupy one location at a time and is not available in the dot programming mode.

3.8.4.3 Cursor Addressing

The cursor position can be commanded from a BASIC programme, allowing subsequent characters to appear anywhere on the screen. Because HUSKY's screen software suppresses leading spaces and multiple line-feeds, this is the preferred way of building screen formats.

The format for cursor addressing is:

SI,X,Y

Where SI is a special cursor control character recognised by the screen software and X,Y are co-ordinate values derived from Fig.3. 1.

The standard BASIC format for cursor addressing is:

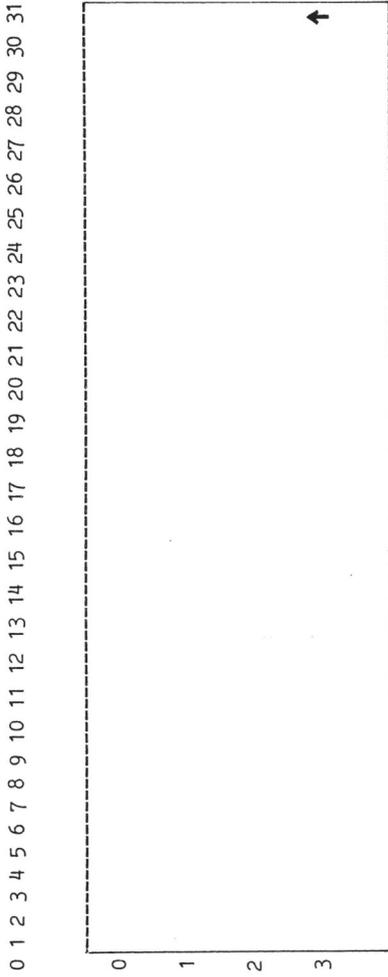
OPCHR 15,X,Y

Where OPCHR is the BASIC command (see Section 3.3.2.16), 15 is the decimal equivalent of SI and X,Y are the co-ordinates expressed in decimal.

NOTE: that X is taken modulo-32 and Y is modulo 4, but values greater than 256 will give an argument error.

This sequence cannot be used in direct mode because BASIC repositions the cursor following the command!

Fig.3.1



3.8.4.4 Lower Case

HUSKY's screen will support the full range of lower case characters, as defined in Appendix I.

The standard keyboard layout does not support lower case in the interests of simplicity, but can be easily programmed to replace existing characters with lower case equivalents if required. See Section 4.12.

The simplest method of producing screen formats with lower case characters is to edit the programme text on an external system, or to connect an external CRT terminal via the serial port (see Section 3.3.1.1).

Lower case characters can also be generated by appropriate OPCHR characters.

However, HUSKY's keyboard can be reprogrammed to generate upper/lower case characters, as is shown in sections 4.12.4, Keyboard 3. The use of lower case in pre-programmed prompt sequences is strongly recommended to improve readability and 'friendliness' of the system.

All **HUSKY** programmes generated at DVW feature a standard header page that appears at switch-on and remains on the screen for a fixed time or until a key is pressed. The header page supplies vital information about the programme and use of this format is strongly advocated to other users.

The appearance of the header page is:

Fig.3.2

```
-----  
! DVWu450 01 Jan 1984 09:30:00 !  
! --*==*==< DEMO PROGRAM >*==*-- !  
! 1000 hours use, 1000 Lines free !  
! DEMPROG.1(GNA 13/12/82) 67401 | !  
-----
```

This page was generated by the demonstration program presented in Section 3.8.5.

Notice how the header page uses **HUSKY**'s screen to the maximum effect. Notice also that the clock display is 'active', i.e. continuously updating. While this has the disadvantage of making the keyboard feel 'slow' (because the time needed to re-write the clock display is interleaved with keyboard scans), it has the great benefit of assuring the user that **HUSKY** is 'alive'. See the next section.

3.8.4.6 Dynamic Screens

A vital ergonomic factor in **HUSKY** program design is re-assurance. Many, perhaps most, **HUSKY** operators have never worked with any kind of computer before and are likely to have some suspicion about this example of 'advanced technology'.

The crucial point is that **HUSKY MUST COMMUNICATE WITH THE OPERATOR**. If it doesn't, he'll soon become frustrated at some situation he doesn't understand and resort to 'traditional' solutions - gently encouraging it with a series of sharp blows or worse!

DEMONSTRATION PROGRAMS

3.8.5

- 3.8.5.1 **"General Program"**
A very simple introduction to Husky programming.
- 3.8.5.2 **"Clock Reading Program"**
This program reads HUSKY's calendar clock.
- 3.8.5.3 **"Address Index Program"**
A simple data storage program.
- 3.8.5.4 **"Alarm Clock"**
A complete application demonstrating many useful Husky utilities.
- 3.8.5.5 **"Input Field Format"**
Demonstrates use of INPUT USING for screen formatted input data.

3.8.5.1 GENERAL PROGRAM

```
10 REM DUW/PPA * 09/09/83
20 RESTORE
30 IF Z<>0 THEN 100
40 CLEAR
50 DIM A(2)
60 Z=1
100 OPCHR1
110 INPUT"ENTER 1ST. NUMBER :-"A(1)
120 INPUT"ENTER 2ND. NUMBER :-"A(2)
130 OPCHR1
140 PRINT"1=ADD"
150 PRINT"2=SUBTRACT"
160 INPUT"SELECT 1 OR 2 :- ",I
170 IF I<1 THEN GOTO 160
180 IF I>2 THEN GOTO 160
190 IF I=1 THEN GOSUB 1000
200 IF I=2 THEN GOSUB 2000
210 INCHR,C
220 GOTO 100
1000 Y=A(1)+A(2)
1010 OPCHR1
1020 PRINTA(1)," + ",A(2)," = ",Y
1030 RETURN
2000 Y=A(1)-A(2)
2010 OPCHR1
2020 PRINTA(1)," - ",A(2)," = ",Y,
2030 RETURN
```

3.8.5.2 CLOCK READING PROGRAM

```
10000 A=CALL(223)
10010 A=PEEK(17795)
10020 B=PEEK(17796)
10030 C=PEEK(17797)
10040 D=PEEK(17798)
10050 E=PEEK(17799)
10060 F=PEEK(17800)
10070 G=PEEK(17801)
10080 H=PEEK(17802)
10090 I=PEEK(17803)
10100 J=PEEK(17804)
10110 REM THE TIME AND DATE IS NOW IN
10120 REM VARIABLES A TO J
10130 K=10*B+A
10140 L=10*D+C
10150 M=10*F+E
10160 N=10*H+G
10170 P=10*J+I
10180 PRINT"THE DATE IS :",M,"/",N,"/",P
10190 PRINT"THE TIME IS :",L,":",K,"HOUR
S."
10200 END
```

3.8.5.3 ADDRESS INDEX PROGRAM

```

10 ONEPP0F1010
20 IF <9<>999THENG0101000
30 OFCHR1:PFINT"*** ADDRESS STORE ***"
40 PRINT"A = NEW B = SEARCH C = PRINT"
50 INCHR"ENTER SELECTION: ",A
60 IFA=66THEN200
70 IFA=65THEN400
80 IFA=67THEN500
90 GOTO30
200 OPCHR1:INPUT"ENTER NAME >"X$
210 FORN=0TO2
220 IFX$=A$(N)THEN300
230 IFX$=B$(N)THEN300
240 IFX$=C$(N)THEN300
250 IFX$=D$(N)THEN300
260 NEXTN
270 OPCHR1
280 LOPCHR13,10,10:PRINT"NO OTHER RECORD
S FOUND":LOPCHR13,10,10
290 INCHR,I:GOTO30
300 LPRINTA$(N):LPRINTB$(N):LPRINTC$(N):
LPRINTD$(N):LOPCHR13,10,10
310 INCHR,I:OPCHR1
320 GOTO30
400 OPCHR1:INPUT"ENTER NAME >"A$(Z)
410 INPUT"ENTER 1ST LINE OF ADDRESS >"B$(
Z)
420 INPUT"ENTER 2ND LINE OF ADDRESS >"C$(
Z)
430 INPUT"ENTER LAST LINE OF ADDRESS >"D$(
Z)
440 Z=Z+1
450 GOTO30
500 OPCHR1:FORN=0TO2
510 LPRINTA$(N):LPRINTB$(N):LPRINTC$(N):
LPRINTD$(N):LOPCHR13,10,10
520 NEXTN
530 GOTO270
1000 DIMA$(50,32),B$(50,32),C$(50,32),D$(
50,32),X$(1,32)
1010 ONEPP0F1010
1020 X=000
1030 GOTO30

```

3.8.5.4 ALARM CLOCK

```
0 IF IP>0 THEN GOSUB 200: IU=IT: ON POWER GOTO 21
0
1 IF IP=1 THEN 250
2 GOSUB 99: PRINT "DUUU000": PRINT "-*==*==*=<
    title      >==*==*=-"
3 PRINT IB, : OPCHR 15, 4, 2: PRINT " hours use,
", : OPCHR 15, 15, 2: PRINT ID-D, "lines free"
4 PRINT "filename.0(GNA 23/01/83)",
5 POKE 17832, 170: REM * ONERROR 50
10 IF IP=0 THEN IY=PEEK(17825): IZ=PEEK(1782
6)
11 POKE 18497, 17, 00, 78, 33, 0, 0, 6, 0, 253, 33
, 0, 72, 26, 79, 9, 19, 123, 253, 190, 0, 194, 72
12 POKE 18519, 72, 122, 253, 190, 1, 194, 72, 72
, 253, 33, 2, 72, 253, 117, 0, 253, 116, 1, 201
14 POKE 18432, IY, IZ: IX=CALL(18497): IX=PE
EK(18434)+256*PEEK(18435)
15 IF IP=0 THEN PRINT " CHECKSUM=", IX: IW=IX:
GOTO 100
17 IF IP=1 THEN OPCHR 15, 31, 0, 65, 7
18 OPCHR 15, 24, 3: PRINT IX, : OPCHR 15, 0, 3: I=I
(4)*40+I(3): OUT 132, I
20 IF I>50 THEN PRINT "* * * MERRY CHRISTM
AS *",
21 IF I<50 THEN PRINT "* * * * HAPPY NEW YEA
R *",
25 J=INP(2): OUT 132, 0: IF J-8*INT(J/8)<4 THEN
NOPCHR 7, 7, 7: PRINT "*WARNING, LOW BATTERIES
!*",
28 IF IX=IW THEN 40
29 PRINT IW, "SELF TEST",
30 PRINT " FAILURE!", : OPCHR 7, 7, 7, 7, 7
40 GOSUB 90: I=0
42 IA=CALL(226): I=I+1: IF I>42 THEN 900
44 INKEY IA=0 THEN 42
45 GOTO 80
50 ONERROR 50: GOTO 30
```

```

90 I=255:GOTO92
91 I=0
92 POKE17665,I:I=CALL(220):RETURN
99 PRINT">":OPCHR1:GOSUB91:PRINT*2*:ONBR
EAKGOTO900:HELP450:RETURN
100 INPUT"HUSKY S/N=?",IS:POKE17724,1:PR
INT" WAIT!",
110 DIMI(9),IO(32),IO*(2,20),IM*(0,100),
I*(2,20):I*(1)="":I*(2)="PLEASE"
111 IO*(0)="Futuredata,LSJ etc.":IO(0)=0
:IO(1)=6:IO(2)=1:IO(3)=1:IO(7)=1:IO(9)=1
190 ID=INT((FRE(0)-1000)/5):IFID>16000TH
ENID=16000
191 DIMD(ID)
199 IP=42:POKE17724,0:GOTO0
200 I=CALL(223):FORI=1TO5:I(I)=10*PEEK(1
7794+I+I)+PEEK(17793+I+I):NEXTI
201 IT=I(2)+I(1)/60:RETURN
210 GOSUB200:IFIT>IUTHENIB=IB+IT-IU
211 IFIP=1THENOUT79,12
212 OUT131,0
215 GOSUB91
220 GOSUB200:PRINT"0=CANCEL":INCHR"1=SET
",I:OPCHR13:IP=1:IFI<>49THENIP=42:GOTO90
0
221 FORI=1TO4:I(I+5)=I(I):NEXTI:GOSUB90:
INPUT"MESSAGE: ",IM*:OPCHR1,15,0,1
222 GOSUB91:I=CALL(226):INPUT"MINS(0-59)
":I(6):INPUT" HOUR:(1-24)":I(7)
223 INPUT"DATE(1-31)":I(8):INPUT" MONT
H(1-12)":I(9):GOTO210
250 IFI(3)=I(8)THENIFI(4)=I(9)THENIFI(2)
=I(7)THENIFI(1)=I(6)-1THEN280
260 IFPEEK(17723)=2THENOUT79,12:OUT131,0
269 GOTO2
280 OPCHR1,7,7,7,7:PRINTIM$:INCHR,I:IP=4
2:GOTO2

```

```

400 PRINT"Husky ser#",IS,
401 OPCHR15,16,1:PRINT"Prog.bytes",PEEK(
17825)+256*PEEK(17826)-IY-256*I2
402 PRINT"Capacity ",ID,-OPCHR15,16,2:PR
INT"Free bytes",FRE(0)
404 PRINT"Lines used",D
406 PRINT"Alarm at:",I(7),I(6),"on",I(8)
,I(9),I(5),
409 GOTO800
450 REM===== DON'T PANIC! =====
452 REM
460 REMUse up/down arrows to view HELP
461 REMPress Enter to resume work...
462 REM   more help.....
463 REM           etc
500 LOPCHR13:GOSUB99:PRINT"0=Program" PR
INT"1=",IO$(0):PRINT"2=",IO$(1):PRINT"3=
",IO$(2),
510 INCHR,I:I=I-49:IFI<0THEN550
512 IFI>2THEN500
520 GOSUB530:GOSUB99:RETURN
530 FORIC=0TO10:POKE17808+IC,IO(IC+11*I)
:NEXTIC:IC=CALL(229):RETURN
550 GOSUB99:INCHR"Code (1to3)? ",I:I=I-4
9:OPCHR13:INPUT"Description? ",IO$(1)
560 GOSUB530:OPCHR1:IC=CALL(241)
570 FORIC=0TO10:IO(IC+11*I)=PEEK(17808+I
C):NEXTIC:GOTO500
600 GOSUB500:GOSUB690:LPRINTI$(1)
610 LPRINTD:FORIC=0TO10:LPRINTD(IC):NEXT
IC
620 LPRINTI$(1):OUT132,0:OPCHR7,2,7:IO-1
:GOTO2
650 GOSUB500:GOSUB695:IFI$(2)<>I$THEN900
660 INPUTI$:OPCHR7:IFI$(1)<>I$THEN660
670 INPUTD:FORIC=0TO10:INPUTD(IC):NEXTIC
680 INPUTI$:OUT132,0:IFI$(1)<>I$THEN30

```

```

689 OPCHR2,7:GOTO900
690 INCHR"Press Enter when ready to send
:",I:RETURN
695 IFIR=0THENOPCHR2,7,7:PRINT"DATA NOT
PRINTED!"
696 IFIQ=0THENOPCHR2,7:PRINT"DATA NOT ST
ORED!"
698 OPCHR13,7:PRINT"WARNING! Data loss i
mminent!"
699 GOSUB90:INPUT"Enter password:",I$:RE
TURN
700 INCHR"0=HEX>DEC 1=DEC>HEX" I:IFI=49I
HEN725
705 GOSUB91:PRINT"ENTER 4-DIGIT HEX:",
710 GOSUB720:IH=I:GOSUB720:IJ=I:GOSUB720
:IK=I:GOSUB720:IL=I
715 PRINT4096*IH+256*IJ+16*IK+IL:GOTO725
720 INCHR,I:I=I-48:IFI>9THENI=I-7
721 RETURN
725 OPCHR13:INPUT"DECIMAL(0-65535)? ",I:
IFI>65535THEN725
730 IJ=4096:PRINT" = ",.IFI<0THEN725
735 FORIC=1TO4:IA=INT(I/IJ):GOSUB745:I=I
-IJ*IA:IJ=IJ/16:NEXTIC:GOTO725
745 IK=IA+48:IFIK>57THENIK=IK+7
747 OPCHR1K:RETURN
750 WINPUTI$:GOTO750
760 IH=INP(193):IL=INP(192):PRINT"H=",IH
,"L=",IL,"HL=",IH*256+IL:GOTO760
800 GOSUB90:INCHR,IA
801 GOSUB99:GOTO800+IA
864 GOTO900
865 GOTO220
866 IB=0:GOTO2
867 I=CALL(235):GOTO900
868 IR=0:IQ=0:GOTO3000
870 GOTO400

```

```

873 GOTO1000
877 GOTO900
880 GOTO2000
882 GOTO650
883 GOTO600
884 GOTO760
885 GOTO950
887 GOTO750
888 GOTO700
900 GOSUB99:ONBREAKGOTO2
910 PRINT"U=UTILITIES
"
911 PRINT"I=INITIATE
"
912 PRINT"P=PRINTOUT
"
913 PRINT"D=DATA IN
",:GOTO800
927 GOTO900
950 PRINT"A=Alarm M=Main Menu W=Wand tes
t"
951 PRINT"B=Batt' R=Receive X=hex/dec"
952 PRINT"C=Clock S=Send data Y=
"
953 PRINT"F=Facts T=Test port Z=
",:GOTO800
1000 GOSUB695:IFI$(2)<>I#THEN900
1010 FORI=0TOID:PRINT">",:D(I)-0:NEXTI:D
=0:GOTO900
2000 GOSUB500
2100 LPRINT"DATE:",I(3),I(4),I(5):LPRINT
"TIME:",I(2),I(1):LPRINT"HUSKY#",IS
2200 FORI=0TOID
2210 LPRINTI,D(I)
2399 NEXTI
2499 IR=1:OUT132,0:GOTO900
3000 PRINT"
"

```

```
3011 PRINT"  
"  
3012 PRINT"  
"  
3013 PRINT"Line#",D,  
3019 INPUT,D(D):D=D+1  
3099 GOTO3000
```

3.8.5.5 INPUT FIELD FORMAT

```

100 REM INPUT FIELD FORMAT DEMO
110 IF Z<>0 THEN 300
120 INPUT USING ("99",1,2)"PLEASE TYPE MESSAGE LENGTH (29
    CHARACTERS MAXIMUM):",L
130 IF L=0 THEN PRINT "SORRY, CANNOT BE ZERO!":GOTO 120
140 IF L>29 THEN PRINT "LINE LENGTH EXCEEDED:PLEASE RE-
    ENTER":GOTO 120
150 S=(FRE(0)-1000)/(L+1)
160 PRINT "TOTAL NUMBER OF ENTRIES POSSIBLE IN THIS HUSKY IS",
170 S=INT(S)
180 PRINT S
190 DIM P$(1,32)
200 DIM D$(S,L)
205 IF 32-L-3=0 THEN 250
210 PRINT "PLEASE ENTER PROMPT MESSAGE (UP TO",
220 PRINT 32-L-3
230 PRINT "CHARACTERS):",
240 INPUT USING ("*",32-L-3),P$
250 Z=55
260 Y=1

300 REM SCREEN ADDRESSED INPUT FIELD
310 FOR Y=Y TO S
320 OPCHR 1
330 OPCHR 15,10,1
340 PRINT "ENTRY NUMBER",
350 PRINT Y
360 OPCHR 15,0,2
370 PRINT P$,"["
380 FOR N=1 TO L:PRINT " ",:NEXT
390 PRINT "]"
400 FOR N=1 TO L+1:OPCHR 8:NEXT
410 INPUT USING ("*",1,L),D$(Y-1)
420 NEXT Y

```

NOTES:

Line 120 The input using mask is "99", numeric values only.
Line 130 A length of zero is not permitted.
Line 150 A provision of 1000 bytes is allowed for string pool
and program variables.
Line 205 If 32-L-3 equals 0, a 'syntax error' would occur in
line 240.
Line 260 Y (Entry Number) starts at 1, not zero.
Line 320 OPCH1 clears the screen.
Line 330 Screen addressing.

OFF-LINE PROGRAM STORAGE

3.9

Most **HUSKY** applications require storage of users BASIC source programs in an easily accessible bulk-memory system for support, updating, exchange and maintenance.

One very popular method of storing **HUSKY** programs is in users' multi-access mainframe database systems, although minis, micros and other **HUSKYS** are also used extensively. **HUSKY**'s ability to communicate freely with these other systems is vital to supporting its programming and application.

This section deals with both manual and programmed source code loading and unloading using BASIC's complimentary **LLIST** and **LLOAD** commands. Other methods of transferring programs, e.g. 'CLONING', are dealt with elsewhere.

3.9.1

LLOAD

LLOAD is the command for loading BASIC source programs into **HUSKY**. It may be used to load entire programs or to modify existing ones by appending or overwriting lines.

LLOAD can be used with any of the communications protocols detailed in part 5, 'COMMUNICATIONS', with the exception of Intel Hex, which is reserved for object code only. Certain speed/protocol limitations are detailed below.

3.9.1.1

Manual Control

To enter program lines, simply type '**LLOAD**' followed by carriage return. **HUSKY** will then load to memory source text presented on the serial port and echo the input on the screen. Note that syntax is checked on a line by-line basis and that if an error is detected, the **LLOAD** will terminate and a 'Syntax Error' will be displayed on the screen.

If the program provided overflows the memory space available, a 'Storage Overflow Error' will appear. An acceptable shorthand for **LLOAD** is 'Control Enter', (see Appendix III), which has the same effect.

LLOAD mode can be terminated by pressing '**ESC**' on the **HUSKY** keyboard or 'power off'. (Data will not be lost).

Alternatively, an '**ESC**' character sent over the interface will return control to the keyboard.

3.9.1.2

Programmed Control

Logically, it is difficult for a program to append to or modify itself, since the sequence controlling the modification may be modified as well! However, **HUSKY** programs can contain sequences that will command a remote processor to send program text and

then accept that text as BASIC source.

Current versions of **HUSKY** BASIC do **not** support LLOAD as a program statement, e.g. 100 LLOAD. This situation will be rectified in future releases. However, the 'Logical Keyboard' Flag (See Section 4.1.4) has the same effect as LLOAD and can be commanded in a program as a 'POKE' instruction. By setting the flag to the serial input port and then returning to interpreter mode, **HUSKY** is configured to accept new program lines. These lines may overwrite the lines originally used to enter the LLOAD mode.

Loading may be terminated by sending an unnumbered 'RUN' statement which will cause execution to re-commence from the start of the new or updated BASIC program, making the load sequence virtually transparent to the operator.

A possible program format might be:

```
100 REM Program reload routine
110 INPUT "Please enter new program name", X3$
120 LPRINT "Hi there, mainframe. Please submit file name",
130 LPRINT X3$
140 POKE 17664,1 : REM This sets the logical keyboard as the
    serial input port
```

```
150 END : REM Returns control to the interpreter
```

```
.
.
.The new program lines now load
.
.
.
```

.The last line transmitted is unnumbered RUN, which executes

```
10 POKE 17664,0 : REM Restore the logical keyboard
20 PRINT "New program loaded OK : Continue?"
```

HUSKY's input buffer ensures that the first line of program is captured, no matter how quickly the distant computer responds.

3.9.1.3 **IMPORTANT NOTE**

In present versions of the **HUSKY** operating system, any alteration to program content, including deletion of lines, **automatically** clears all variables and arrays. This is essential to **HUSKY's** operation, but means that data **cannot** be carried across reload boundaries.

3.9.2

LLIST

LLIST is the principal method of copying and recording **HUSKY** programs. Other methods, e.g. 'Cloning', are not described here.

Invoking LLIST causes program lines to list sequentially from the lowest line number or from a line number specified as an argument in the LLIST. Examples are :

```
LLIST Lists an entire program
LLIST 170 Lists from line 170
LLIST 2133 Lists from 2133 or, if 2133 does not exist, from the
next highest line.
```

LLIST terminates when all line numbers are listed, when 'ESC' is pressed or when **HUSKY** is powered down. Data will not be lost if **HUSKY** powers down during a listing.

LLIST cannot be incorporated in a program, and is only available for manual use.

Protocol and format selections made in **HUSKY**'s communication package are observed transparently by LLIST. Carriage Return terminator characters are provided at the end of each program line, line feed and null characters are optional.

IMPORTANT NOTE

Remember that **HUSKY**'s communication system needs to power up to its RS-232 state for transmission to occur. This power-up will cause an off-to-low transition that many systems will read as a single, spurious, character. This event only occurs once and can be avoided by powering up the interface prior to activating a receiving device. Methods of pre-powering the interface include :

```
LPRINT " " : REM the interface powers up automatically
OUT 132,1 : REM Direct control of the communications inverter
```

Remember that once powered up, the RS-232 output remains active until **HUSKY** is powered down or the inverter is commanded 'off' by :

```
OUT 132,0
```

HUSKY power consumption increases substantially during RS-232 transmission with corresponding reduction in battery life.

It should be noted, however, that if the RS232 line is powered off further serial transmission will be lost unless the interface is powered up again, by:

```
OUT 132,1
```

NOTE: The interface will not power-up automatically with a simple LPRINT statement with no argument.

3.9.3 **Speed/Protocol Limitations**

It is always advisable that some form of protocol handling is established between **HUSKY** and associated computers.

Some multi-access mainframes and some popular microcomputers are unable to support any form of protocol, however.

In these cases, **HUSKY** will generally support direct program transfer at speeds of 300 baud or less. At higher speeds, **HUSKY's** input buffer may overflow leading to loss or truncation of lines. At 300 baud, the buffer smooths the flow of incoming program through **HUSKY BASIC's** syntax checking routines.

Problems with truncation will invariably lead to the load operation being aborted and a 'Syntax Error' message being displayed, since the Interpreter will not accept partial lines.

The capability of supporting simple 300 baud communication is invaluable in many situations.

3.9.4 **Communications with Databases**

A very useful **HUSKY** feature is the ability to communicate directly with mainframe databases using 'VDU mode'.

This interactive mode can be used manually to establish logon procedures, passwords, etc., before attempting to transfer or copy files or inspect data.

Sequences proven out manually can, in most cases be implemented in **HUSKY Basic** as automatic features of the user's program once protocols, etc, have been finalised. Such sequences can present inputs to the mainframe and inspect the reply for keywords like 'READY' or simple cursor prompts.

NOTE Many mainframes are unpredictable in the response they provide to login sequences, with variable 'welcome' or 'news' messages. Make sure your automatic sequence is robust enough to handle these eventualities!

Some general points about database communications should be noted:

3.9.4.1 **Rate**

Communication occurs generally at 300 baud, although other configurations are possible. At this speed, **HUSKY'S** screen handler will give a 'line-by-line' presentation that can be read 'on the fly', even if long pages are involved. Remember that the screen character generator will display all ASCII characters, even if they are not shown on the keyboard.

Other rates encountered on dial-up systems are 110 (very rare) and 1200, but only with sophisticated modems.

3.9.4.2 Parity

Dial-up systems can expect any of even, odd or no parity selections. Transmit and receive parities are generally the same. Always use receive parity if it is available - the occasional appearance of the parity error symbol is a useful indication of bad lines. Remember that telephone lines can be bad in **one direction only**.

3.9.4.3 Full/Half Duplex

Database systems vary widely in the use of full or half duplex operation.

Full duplex occurs when the host system 'echos' every received character back to the HUSKY. In this mode, HUSKY's transmission echo should be selected 'off' so that characters typed on the keyboard only appear on the screen if they have completed the whole circuit of HUSKY - Mainframe - HUSKY. BASIC routines can check full duplex replies as an absolutely secure communication protocol, provided the mainframes' own messages don't confuse the issue.

If the echo switch is left 'on', double characters will appear on the screen. This in no way affects communication, but makes outgoing messages hard to read!

Half duplex occurs when the host does not echo characters back to HUSKY. In this situation, the echo switch should be 'on' to allow outgoing messages to be read by the operator. This method does not guarantee that data sent to the host is being correctly interpreted.

A third mode, **Simplex**, occurs when data is sent in only one direction at one time and is otherwise similar to Half duplex. HUSKY is not concerned which mode is in use.

3.9.4.4 Protocol

Unfortunately, very few dial-up databases support any kind of protocol, so that generally HUSKY'S 'none' option should be used. Note that 300 baud is the fastest recommended speed for no protocol program loading.

However, some systems do support 'XON/XOFF' although implementations vary between computers - seek advice from DVW if difficulty is encountered.

3.9.4.5 Other Parameters

The setting of 'NULL' and 'LF' are generally not material in mainframe communication, so that 'O' is recommended for NULL count. See below for LF.

3.9.4.6 Terminator

HUSKY'S 'ENTER' key generates CR (Carriage Return), but some mainframes expect other terminators. Examples are Control C and LF (Line Feed), or 'New Line'. Consult your computer manual for

details.

3.9.4.7 Delete

HUSKY'S DEL (Delete) key generates 'rubout'. Few mainframes recognise this, or in many cases allow any deletion at all! 'Back Arrow' (←) may work in some cases.

3.9.4.8 Acoustic Couplers

Good quality communication can often be achieved over surprising distances with acoustic couplers at 300 baud, but this method is not recommended for routine daily use.

3.9.5 Extra Documentation Lines

It is worth knowing that source files kept on computer systems may have further documentation within them by using unnumbered 'REM' lines, e.g.

```
100 PRINT "HELLO"
```

REM This line of text will be ignored and will only load line 100 into the **HUSKY**. This prevents loading unnecessary text. Naturally, these lines can only be created on the host or data-base computer.

3.9.6 IBM and Similar Editors

Some mainframe editors are confused by multiple carriage returns generated at the start of an "LLIST" sequence. An example is to suppress 'LF' when listing programmes to VM370 IBM running under CMS operating system. (Husky generates a CR/LF sequence to provide initial formatting in printed listings).

At the very start of program to be dumped put a single line program as follows:

```
10 LOPCHR73:FOKE19765,1:A=CALL(5249)=END
```

Output I prefix for editor	Suppress LF's	Call to LIST program
-------------------------------	------------------	----------------------

User types 'RUN' to dump the program.

Format:

```
I 20PRINT"HELLO" CR 30PRINT".....
```

The above will only work with HC0103/HC0104 firmware revisions as it relies on knowing the location of 'LIST' in the operating system. Contact the factory for assistance if your revision does not match up with the above.

ASCII CHARACTER SET

LSD \ MSD	0 000		1 001		2 010		3 011		4 100		5 101		6 110		7 111	
	Char	Dec														
0	0000	NUL	0	16	SP	32	0	48	@	64	P	80	\	96	P	112
1	0001	SOH	1	17	!	33	1	49	A	65	Q	81	a	97	q	113
2	0010	STX	2	18	"	34	2	50	B	66	R	82	b	98	r	114
3	0011	ETX	3	19	#	35	3	51	C	67	S	83	c	99	s	115
4	0100	EOT	4	20	\$	36	4	52	D	68	T	84	d	100	t	116
5	0101	ENG	5	21	%	37	5	53	E	69	U	85	e	101	u	117
6	0110	ACK	6	22	&	38	6	54	F	70	V	86	f	102	v	118
7	0111	BEL	7	23	'	39	7	55	G	71	W	87	g	103	w	119
8	1000	BS	8	24	(40	8	56	H	72	X	88	h	104	x	120
9	1001	HT	9	25)	41	9	57	I	73	Y	89	i	105	y	121
A	1010	LF	10	26	*	42	:	58	J	74	Z	90	j	106	z	122
B	1011	VT	11	27	+	43	;	59	K	75	[91	k	107	{	123
C	1100	FF	12	28	,	44	<	60	L	76	\	92	l	108		124
D	1101	CR	13	29	-	45	=	61	M	77]	93	m	109	}	125
E	1110	SO	14	30	.	46	>	62	N	78	^	94	n	110	~	126
F	1111	SI	15	31	/	47	?	63	O	79	←	95	o	111	DEL	127

HUSKY IMPLEMENTATION OF CONTROL CODES

The HUSKY keyboard implements alphabetic ASCII control characters which are obtained with control and an alphabetic key.

The LCD display control software will respond to some of the control codes to provide specific functions, any other control characters received are ignored. The characters may be sent to the display either from user programs or via the serial port.

- a) **Clear Screen** Decimal value : **01 ASCII : SOH**
This code clears the HUSKY screen, apart from the shift arrow, and places the cursor at the top left character position.
- b) **Bell** Decimal value : **07 ASCII : BEL**
This character causes the HUSKY's internal sounder to bleep, for use as an audible warning.
- c) **Cursor Back** Decimal value : **08 ASCII : BS**
Moves the cursor one space to the left. It does not affect any characters on the screen. If the cursor is already at the top left of the screen then there is no effect.
- d) **Cursor UP** Decimal value : **11 ASCII : VT**
The cursor is moved vertically up one line, if the cursor is already at the top of the screen then there is no effect.
- e) **Forward cursor** Decimal value : **12 ASCII : FF**
This character moves the cursor one position forward unless the cursor is at the bottom right of the screen in which case the screen is scrolled.
- f) **Carriage Return** Decimal value : **13 ASCII : CR**
The command performs the usual carriage return, line feed. The effect varies dependent on the position of the cursor on the screen. If there are no characters on the line then the command is ignored, to maximise the information content on the screen. If the cursor is at the bottom line then the screen is 'scrolled up' to provide a blank bottom line.
- g) **Down cursor** Decimal value : **14 ASCII : SO**
Similar to carriage return except that the cursor moves vertically down and is not necessarily placed on the left of the screen. Also, the function is always implemented even if there are no characters on the line. The screen is scrolled up if the cursor is already on the bottom line.

- h) **Cursor addressing** Decimal value : 15 ASCII : SI
This is the cursor addressing character. It signifies the start of a three character sequence consisting of first itself, followed by the X coordinate and then the Y coordinate. The values used for X and Y can be any in the ASCII range 0-7FH. However, X is taken modulo 32 to define the character on the line and Y is taken modulo 4 to define which line. See section 3.10.3.
- i) **Delete** Decimal value : 127 ASCII : DEL
Acts just as cursor back but erases the character on which the cursor lands.
- The keyboard implements all standard control keys. There are further functions used to control the HUSKY directly.
- a) **Control Enter**
Holding the control key down and pressing Enter causes the HUSKY to receive characters from its serial interface rather than the keyboard. This function is identical to BASIC's LLOAD. Its function is, for example, loading BASIC source from an external serial source such as a disk based microcomputer. It should be noted that handshaking needs to be used if BASIC programs are being loaded to prevent characters from being lost at speeds greater than 300 baud (Typical).
- Control is returned to the keyboard upon pressing ESC.
- b) **Control Delete**
This function suppresses the sounder during keyboard operations. Pressing Control Delete once suppresses the sounder, pressing it again re-activates the sounder.
- There are some further keys which return control codes not currently implemented on the HUSKY, or which have special functions:
- a) **ESC** Decimal value: 27 ASCII:ESC
Used to break into a running BASIC program. Requires entry of the escape code if HUSKY is in auto-start mode, see Section 3.1.4
- b) **CNVL HELP** Decimal value: 28 ASCII:FS
Used to escape from VDU mode.

- c) **INSERT** Decimal value: **30 ASCII:RS**
Not currently used
- d) **BREAK** Decimal value: **31 ASCII:VS**
Used with ON BREAK command, see Section 3.3.4.16 for details.

ADVANCED PROGRAMMING

HUSKY ADVANCED PROGRAMMING

- 4.1 MEMORY MAP
- 4.2 SYSTEM FUNCTION CALLS
- 4.3 THE HUSKY SCREEN
- 4.4 HELP
- 4.5 THE PARALLEL PORT
- 4.6 SINGLE BIT INPUT PORT
- 4.7 THE ON/OFF KEY
- 4.8 CLONING HUSKY
- 4.9 MACHINE CODE PROGRAMMING
- 4.10 PORT ALLOCATIONS
- 4.11 PANIC
- 4.12 PROGRAMMABLE KEYBOARD

APPENDIX I NSC800 MACHINE CODE

APPENDIX II BASE CONVERSION - HEX/DECIMAL

4.1

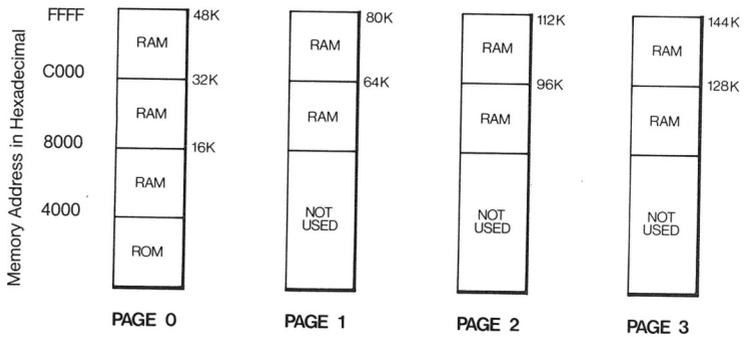
MEMORY MAPS

4.1.1

General

HUSKY memory can be expanded in regular increments up to a maximum storage size of 144K bytes. The memory is physically organised into four pages as shown in the diagram.

Fig.4.1



NOTE: Not all the memory options shown here are available as standard products. Please consult a current price list for details.

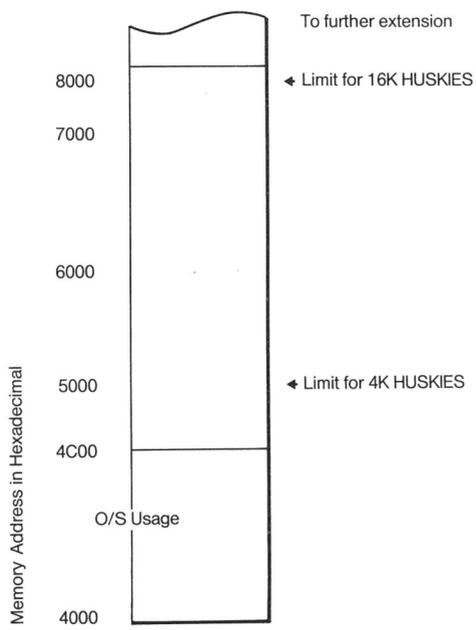
In order to give users the full benefits of this large memory potential without any of the complexities involved in paging memory, the operation involved in paging memory within the **HUSKY** is entirely transparent, being handled by the resident operating system.

4.1.2

RAM Memory Map

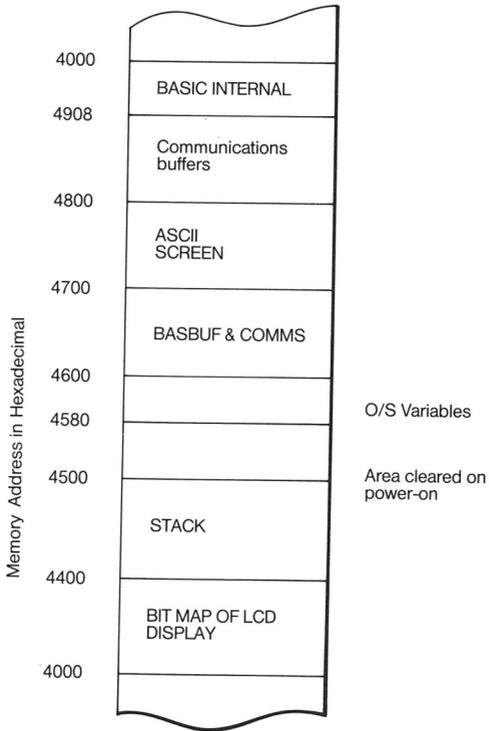
In considering the RAM allocation we shall only be dealing with Page 0 RAM, which contains all the **HUSKY** operating parameters. Extension RAM merely extends the space for use by BASIC.

Fig.4.2



Expanding the memory used by the O/S we have:

Fig.4.3



As can be seen there is fixed usage of RAM of C00 (Hex) bytes (3K) leaving, in a 144K machine (actually 147,456 Bytes), an actual 144,000 bytes for program and storage.

4.1.3 HUSKY Internal Storage Allocations

- i) Bit map of LCD display. The LCD display extracts the dot information directly from this RAM area. Do not attempt any operation to this area of RAM.
- ii) The stack is between 4400H and 4500H. Under no circumstances should this area be altered.
- iii) The RAM from 4500 and 4580 is cleared on switch- on. Some of the variables in this area are detailed later.
- iv) From 4580 to 45FF are variables which are not cleared on power on. Some variables are detailed later.
- v) From 4600H to 4700H are internal variables and should not be altered.
- vi) ASCII screen:
This area of 128 bytes from 4700H to 4780H contains the ASCII mirror of the display. It is possible to read from the screen by addressing this area. If characters are written to this area, then they will be displayed if the screen refresh function is executed (or the characters are written normally to the screen causing it to refresh).
- vii) Communications Buffer:
This area from 4800H to 4908 is used by the communications package as buffers.
- viii) The area from 4908H to 4C00H is used internally by the Basic interpreter, and should not be used.
- ix) 4C00H upwards is program store for Basic or user loaded programs.

4.1.4 Useful Memory Locations

These following locations may be useful (to 'PEEK' or 'POKE' to) in Basic to provide the desired response.

MEMORY LOCATIONS

ADDRESS		NAME	FUNCTION
DECIMAL	HEXADECIMAL		
18176 -18303	4700H -477FH	SCRNBUF	The ASCII screen buffer organised as: 4700H=Top left corner 477FH=Bottom right corner Note 1
17664	4500H	IPFLAG	Logical Keyboard flag: 0H=Husky keyboard 1H=Serial I/P port
17665	4501H	FSHIFT	The keyboard shift key: 0H=Number or upper shift 0FFH=Letter or lower shift Note 1
17666	4502H	CURADDR	The cursor address register contains a number from 0H to 7FH designating the current position. Note 1
17667	4503/ 4504	VECTOR	The HELP vector address (see Note 3) See section 4 on use of HELP.
17704	4528	QUIET	Setting this byte non-zero turns off "automatic" bleeping of the keyboard.
17723	453BH	APHRO	This byte holds the method by which power-up took place. 0 = Manual, thro' keyboard 1 = Serial port 2 = Automatic via clock
17724	453CH	NYMPHO	If the standard power down routine is used then setting this byte non-zero will inhibit operation of the power off key. Note: This byte is cleared on power-up.
17727	453F	FOREVER	Setting non-zero prevents auto-matic switch-off after time out.
17756	455C	PAGE	This byte holds the page of either program or data being accessed.
17792	4580	TENTHSEC	Time Memory 1/10 sec
17793	4581	UNITSEC	" " unit secs

ADDRESS		NAME	FUNCTION
DECIMAL	HEXADECIMAL		
17794	4582	TENSEC	" " tens secs
17795	4583	UNITMIN	" " unit mins
17870	45CE	PDEF	The page of definition storage is held in this byte.
17871	45CF	PLIM	Start of array definition page.
17874- source.	45D2-	ENDBAS	Address of the very last byte of BASIC
17875	45D3		
17876- 17877	45D4- 45D5	STRBAS	Address of start of BASIC source.
17879- 17880	45D7- 45D8	BASE	Start of memory used either for BASIC or user machine code. If there is no machine code then BASE=STRBAS.
17796	4584	TENMIN	Time Memory tens mins
17797	4585	UNITHRS	" " unit hours
17798	4586	TENHRS	" " tens hours
17799	4587	UNITDAY	" " unit days
17800	4588	TENDAY	" " ten days
17801	4589	UNITMTH	" " unit month
17802	458A	TENMTH	" " tens month
17803	458B	UNITYR	" " unit years
17804	458C	TENYR	" " tens years
17808	4590	TXSPEED	Set serial O/P baud rate: 0=50 baud 3=150 baud 6=1200 baud 1=75 baud 4=300 baud 2=110 baud 5=600 baud See Note 2
17809	4591	RXSPEED	Set serial I/P baud rate (same as TX)
17810	4592	CTSAF	The 'CTS' enabled flag: 0 = no 1= yes
17811	4593	RTSAF	The 'RTS' enabled flag: 0 = no 1= yes
17812	4594	TXPTY	Transmit parity flag: 0 = none 1 = odd 2 = even

ADDRESS		NAME	FUNCTION
DECIMAL	HEXADECIMAL		
17813	4595	RXPTY	Receive parity flag: 0 = none 1 = odd 2 = even
17814	4596	TXPROT	Transmission Protocol: 0 none 1 = XON/XOFF 2 = ETX/ACK 3 = ACK/NAK 4 = SYSTIME
17815	4597	LEAF	Line feed active or not: 0 = no 1 = yes
17816	4598	NULAF	Number of nulls following CR/LF 0 = none 3 = ten 1 = two 4 = twenty 2 = five
17817	4599	ECHOF	The serial output echo flag: 0 = no echo (FULL DUPLEX) 1 = output echoed onto HUSKY screen
17818	459A	RXPROT	Receive protocol conrols as TXPROT
17821/ 17822	459D 459E	CTSVECT	CTS interrupt vector. These locations contain an address for the CTS input handler routine.
17823 17824	459F/ 45A0	DEFLIM	Address of the lower limit of Basic simple variable and Array definitions. Begining of free memory after Basic source and arrays (note 3).
17825/ 17826	45A1/ 45A2	ALIM	Address of the upper limit of Basic program and Arrays. End of free memory before Symbol Table (Note 3).
17827- 17831	45A3- 45A7	ESCODE	Escape code. Used when escaping from fully running code. Used to interrupt execution of user programs when Basic auto-start is in use. ASCII characters are stored in Decimal form, most significant digit at lowest memory location.

ADDRESS		NAME	FUNCTION
DECIMAL	HEXADECIMAL		
17832	45A8	STARTF	Flag to show immediate running of user programs. AAH (170 Decimal) = immediate run.
17725- 17726	453DH- 453EH	OFFVECT	Power off vector contains the address of the power off routine, may be altered for user power down.
17842	45B2H	SERIG	This contains the ASCII code of a character to ignore when receiving serial data. If not required set to 80H (128 Decimal).

NOTE 1: If these locations are changed then to display on screen system call 37 (Update Husky Screen) should be executed.

NOTE 2: If these locations are changed then call Function 40 (Revise Serial I/O) to actually implement.

NOTE 3: Double-byte locations are stored in 8080 convention, with the first (lowest) byte containing the least significant data.

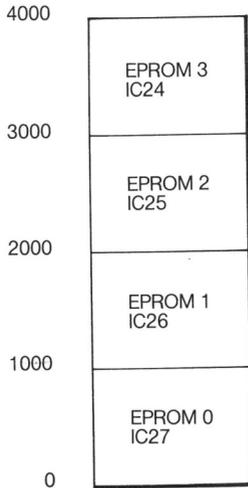
4.1.5 **The ROM Memory Map**

Memory used by the firmware is divided into 4 EPROMs each occupying 1000H of memory.

EPROM's are any of 2732, 2764 or 27128 varieties.

The locations of interest are mainly at the bottom of memory:

FIG.4.4



CP/M INTERFACE

4.2

4.2.1

Purpose

The calls to the 'system' are designed for users who wish to write their own assembler, or compiler software for the HUSKY.

4.2.2

General

The calls are made through location 5H. The C register contains a number from 0H to 29H which will identify the required function. Other registers may be required to pass information to the called program. Values are also retained in the indicated registers. Registers not mentioned are irrelevant on entry, but are not necessarily preserved during the call. Also provided is the actual 'CALL' able address (the contents of REG C are then irrelevant).

Example:

To print the character '1' on the screen could be implemented as follows in assembly code:

```

      .
      .
      .
LD     E,'1'      ; Required character in E
LD     C,2        ; Required function
CALL  5H          ; The call

```

In BASIC, this call would be implemented as follows:

```

100 A = ARG(49 x 256 + 2)
110 A = CALL(5)

```

NOTE: The character '1' is equivalent to 49 decimal. See Basic Programming Appendix I for further details.

A call not requiring an argument could be written as follows:

```

100 A = CALL(226)

```

Eliminating the need for the ARG statement. 226 is the actual decimal address of the routine.

IMPORTANT NOTE

There is **no** relationship between register names (A,C,E etc.) in this section and Basic variable names.

4.2.3 SYSTEM CALLS

- 4.2.3.1 (0) FUNCTION : System Reset 'CALL' address
 'CALL' ADDRESS : 109 Decimal
 REG C : 0
 Exit : No direct exit
 Purpose : System restart - re-enters the HUSKY Operating System.
- 4.2.3.2 (1) FUNCTION : Fetch and Echo key
 'CALL' ADDRESS : 112 Decimal
 REG C : 1
 Entry : None
 Exit : ASCII value in A, Bit 7 = 0
 Purpose : Fetch a key, entry from the logical keyboard (HUSKY keyboard or serial input) and echo onto the HUSKY screen
- 4.2.3.3 (2) FUNCTION : Console output
 'CALL' ADDRESS : 115 Decimal
 REG C : 2
 Entry : ASCII value in E, bit 7 must be 0
 Exit : None
 Purpose : To display a character on the HUSKY screen.
- 4.2.3.4 (3) FUNCTION : Reader input
 'CALL' ADDRESS : 118 Decimal
 REG C : 3
 Entry : ASCII value in A, bit 7 = 0
 Purpose : To receive a character from the serial I/P port. The character is received with the speed set up by Port Initialisation.
 If parity is requested then it is checked. If parity fails then a special OFFH character is returned.
NOTE: OFFH displayed on the screen shows as a special error character
 ...

- 4.2.3.5 (4) FUNCTION : Punch output
 'CALL' ADDRESS : 121 Decimal
 REG C : 4
 Identical to Function (5)
- 4.2.3.6 (5) FUNCTION : List output
 'CALL' ADDRESS : 124 Decimal
 REG C : 5
 Entry : ASCII value in E, 7 must be 0
 Exit : None
 Purpose : To transmit the ASCII character to the serial output port. The parameters set up during Port Initialise determines:
 speed
 parity
 line feed
 suppression or not Nulls at the end of lines.
 There is also the 'echo' flag which will display the character on the HUSKY screen as well.
- 4.2.3.7 (6) FUNCTION : Console I/O
 'CALL' ADDRESS : 127 Decimal
 REG C : 6
 Entry : Reg E = OFFH input or
 Reg E = char output
 : Reg A = char or status
 Purpose : This function is to force communication with the HUSKY screen and keyboard.
 If Reg E = OFFH then the keyboard is tested and returns with A = ASCII character when key is pressed.
 If E = ASCII character (bit 7-0) then the character is displayed on the screen as for function 2.
- 4.2.3.8 (7) FUNCTION : Get I/O byte
 'CALL' ADDRESS : 130 Decimal
 REG C : 7
 Entry : None
 Exit : Reg A = I/O byte value
 Purpose : To see the current I/O set up of HUSKY.
 See Section on I/O handling for details

- 4.2.3.9 (8) FUNCTION : Set I/O
 'CALL' ADDRESS : 133 Decimal
 REG C : 8
 Entry : E = I/O byte value
 Exit : None
 Purpose : For setting the I/O byte as above.
- 4.2.3.10 (9) FUNCTION : Print string
 'CALL' ADDRESS : 136 Decimal
 REG C : 9
 Entry : DE register points to string address
 Exit : None
 Purpose : Print a string indicated by DE to the logical console terminated with a '\$' character.
- 4.2.3.11 (10) FUNCTION : Read console buffer
 'CALL' ADDRESS : 139 Decimal
 Function not yet implemented
- 4.2.3.12 (11) FUNCTION : Get console status
 'CALL' ADDRESS : 142 Decimal
 REG C : 11 D
 Entry : None
 Exit : A = console status
 Purpose : To read the status of the logical input device (keyboard or serial input).
 A = 0H for no character ready else A Non-zero.
- 4.2.3.13 (12) FUNCTION : Return version No.
 'CALL' ADDRESS : 145 Decimal
 REG C : 12 D
 Entry : None
 Exit : HL with version No.
 Purpose : To simulate CPM returns a value of 20H in HL.

- 4.2.3.14 FUNCTION (13) to (36) are disk system calls and are not implemented. Attempts to use them will cause the **HUSKY** to re-start. REG C values from 0DH to 24H.

The following functions are special to the Husky and perform tasks useful in the Husky environment.

- 4.2.3.15 (37) FUNCTION : Update **HUSKY** screen
 'CALL' ADDRESS : 220 Decimal
 REG C : 37D
 Entry : None
 Exit : None
 Purpose : This routine will re-write the **HUSKY** screen. If direct writing to the ASCII string buffer has been used, then it will be displayed.
- 4.2.3.16 (38) FUNCTION : Read time and date
 'CALL' ADDRESS : 223 Decimal
 REG C : 38D
 Entry : None
 Exit : The 'clock' memory is updated
 Purpose : This function causes the clock to be read and placed in the memory specified by **HUSKY** Technical Notes memory map. Basic can use Peek or Poke to operate on them.
- 4.2.3.17 (39) FUNCTION : Display time and date
 'CALL' ADDRESS : 226 Decimal
 REG C : 39D
 Entry : None
 Exit : None
 Purpose : As with function (36) this updates the time memory registers, it also places the current time and date on the top line of the display. This function will overwrite the previous contents of the top line.

4.2.3.18 (40) FUNCTION : Revise Serial I/O
 'CALL' ADDRESS : 229 Decimal
 REG C : 40D
 Entry : None
 Exit : None
 Purpose : This function permits the user to modify the serial I/O parameters (specified in the memory map) and then to actually set up the port by calling this function.

This call will also clear the contents of the received buffer. Care should be taken that data is not being received during this call, or data will be lost.

See Section 4.1.4 for details of the parameter locations.

4.2.3.19 (41) FUNCTION : Start clock
 'CALL' ADDRESS : 232 Decimal
 REG C : 41D
 Entry : Correct time in the clock registers
 Exit : A= 0 or 255D
 Purpose : This function starts the clock with the time and date in the clock registers. The seconds are reset to zero at the time the function is called. No checking is made.

4.2.3.20 (42) FUNCTION : Set clock
 'CALL' ADDRESS : 235 Decimal
 REG C : 42D
 Entry : None
 Exit : None
 Purpose : This call allows setting of the clock exactly as the standard HUSKY Initialise Clock routine. It performs all checking, etc., necessary.

See 'HUSKY Operation' Section 2.5 for details of the use of this program.

- 4.2.3.21 (43) FUNCTION : Read the single input bit
 'CALL' ADDRESS : 238 Decimal
 REG C : 43D
 Entry : None
 Exit : Reg A = 0 for I/P = logic 0
 : Reg A = 1 for I/P = logic 1
 Purpose : This call will read the single input
 : bit. All the revectoring etc is
 : handled.
 : Note: Communications should not be
 : used when this call is executed.
- 4.2.3.22 (44) FUNCTION : Initialise Communications
 'CALL' ADDRESS : 241 Decimal.
 REG C : 44D
 Entry : None
 Exit : None
 Purpose : This call allows configuration of the
 : communications exactly as the
 : standard HUSKY Initialise
 : Communications routine.
 : This will free users of re-writing
 : the routine in Basic.
- 4.2.3.23 (45) FUNCTION : Get serial input status
 'CALL' ADDRESS : 244 Decimal.
 REG C : 45D
 Entry : None
 Exit : A = port status
 Purpose : To test for pending serial input
 : characters. A=0H for no characters
 : read else A is non-zero.
- 4.2.3.24 (46) FUNCTION : VDU Mode 'Call' Address 247 Decimal.
 REG C : 46D
 Entry : None
 Exit : None
 Purpose : None
 : When called from Basic, VDU mode
 : allows HUSKY to operate transparently
 : as a remote terminal. No Basic
 : variables or parameters are
 : affected. Exit from VDU mode is by
 : manual operation of control Help.
 : If VDU mode was 'called' from a
 : Basic program, then control will be
 : returned to Basic after control
 : Help.

SCREEN

4.3

As may have become apparent when using **HUSKY**, the screen control software does not treat the screen in a purely simplistic fashion. It aims to achieve maximum use of the available space and maximum intelligibility of the displayed information.

4.3.1 **Shift Indicator**

This character (| |) is displayed in the bottom right hand corner of the screen to indicate the current keyboard shift level. It is not stored in the screen ASCII buffer but is generated directly by the screen control software. Any character stored in the buffer will come into view if the screen is scrolled.

4.3.2 **Leading Spaces**

Under normal circumstances the screen software will suppress leading spaces on a line with a view to preventing the waste of space. This feature can be overcome by using the right cursor shift character (OCH) instead.

4.3.3 **Blank Lines**

The **HUSKY** will not permit two carriage returns to leave a blank line. This makes use as a VDU on standard computers very much easier. If a blank line is required then the down cursor (OEH) character should be used.

4.3.4 **Word Justification**

If a word is going to overlap the end of a line, then the screen software will move the entire word down onto the next line. This makes reading of the display considerably easier.

4.3.5 **Cursor Addressing**

This feature is fully described in Appendix II of the Basic Manual.

4.3.6 **Direct Screen Use**

The ASCII buffer is contained in memory at locations 4700H-477FH. It is quite feasible to write ASCII characters (in the range 20H to 7FH) directly into the buffer (using POKE in Basic) and then updating the screen using function call 37. Similarly characters may be read directly from the screen using PEEK. However, care should be taken that the screen controller has not changed the position of the character.

4.3.7 **The Screen Dot Buffer**

The LCD display reads the dot information directly from RAM in the range 4000H to 43FFH. For special displays not using the standard character set it is possible to write directly into this buffer. If the normal display software is used then the dot buffer will be changed.

The format of the Dot buffer is:

- i) Each character occupies 8 bytes, although only 7 bytes are used.
- ii) The buffer is in order starting with the top left character at 4000H.
- iii) The dots are used in bit positions D0 to D4 with D0 on the left and D4 on the right.

HELP

4.4

The HELP facility provides a means of breaking into normal programs to display some text for operator assistance.

After the HELP text has been read control is returned to the main program, as if nothing had happened, by pressing 'ENTER'.

4.4.1

The Help Vector

The text displayed by Help can be linked to the operation currently being performed. The start position of the text can be set under program control. This is achieved by the pair of ram locations 'VECTOR' (See Section 4.1.4), Decimal Addresses 17667-17668, indicating the memory address of the start of text. This is controlled by the HELP verb in BASIC.

VECTOR can also be set by POKE.

4.4.2

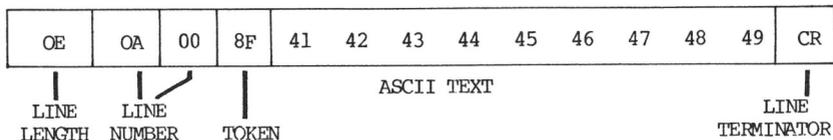
Text Storage

As can be seen in the BASIC section, the text is stored by means of REM statements, which allows storing of ASCII text within programs. The structure of BASIC program lines also has to be contended with. The line:

```
10 REM ABCDEFGHI
```

is stored as:

FIG.4.5



The address in the vector points to the start of the line number. The Help program will skip forward over the first four bytes, checking the REM verb and then displaying the text.

NOTE: The REM verb is stored as 143 Decimal.

4.4.3 **Help Text Display**

The software which controls Help will scroll forward through the lines of BASIC text, or pseudo Basic text, until the verb found is not a REM. Scrolling backwards is also allowed until a line is found without a REM. In each case, further scrolling is not allowed.

4.4.4 **Storing the Current Display**

The contents and cursor position of the screen is stored in a backup ASCII string store. This store is located immediately above the normal screen store.

4.4.5 **HUSKY Action during HELP**

When a program is interrupted for Help display all normal action is suspended. Serial data reception will continue up to the capacity of the receive buffer. If handshaking is enabled then it will continue in a transparent fashion.

Programs will continue after 'Enter' is pressed.

Help will be entered during any scan of the keyboard, whether for status or waiting for an actual key depression.

4.5

PARALLEL PORT

4.5.1

Purpose

The parallel port has been designed for high speed data transfer to the Micropolis disk controller board. However, it does provide a flexible parallel interface for external user systems. This section describes the available signals and their relative timing.

4.5.2

Availability

The port is provided as an option in any **HUSKY** with more than 32K RAM. It is not available in 4K and 16K **HUSKIES**.

The parallel port cannot be user installed and must be ordered from Husky Computers for installation and testing. The port can be retro-fitted to existing **HUSKIES**.

4.5.3

The Interface

The signals are essentially an extension of the microprocessor bus.

The BUS lines (**BUS0-BUS7**) are bidirectional data bits.

To provide timing the following signals are installed:

SLT (Select) **RSTR** (Read Strobe) and
WSTR (Write Strobe)

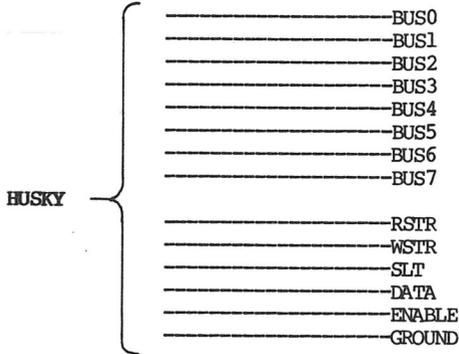
Two additional signals are implemented:

DATA which may be used as either additional single output bit or for further addressing, and **ENABLE** which gives out a reset pulse on **HUSKY** switch on.

It should be noted that the output signals are not latched and that **SLT** must be implemented because the **RSTR** and **WSTR** signals are used for the internal **HUSKY** operation and are continually being activated.

4.5.4 **Signals**

FIG. 4.6



SIGNALS:

BUS0 - BUS7	Bidirectional data bits
DATA	Single output bit
SLT	Active low port active strobe
RSTR	Active low port read strobe
WSTR	Active low port write strobe
ENABLE	Switch-on reset pulse
GROUND	Electrical ground

4.5.5 **Signal Levels**

All signals are CMOS compatible with a 5V supply rail.
Outputs

(NOTE: internal Vcc assumed 5.00V)

FIG. 4.7

BUS0-BUS7

Output high voltage / IO < 1uA	4.95V
Output low voltage / IO < 1uA	0.05V
Output high (source) current VO=4.6V	-2.0mA
Output low (sink) current VO=0.4V	+2.0mA

FIG.4.8

Enable, RSTR, WSTR,SLT

Output high voltage /IO/<1uA	4.95V
Output low voltage /IO/<1uA	0.05V
Output high (source) current VO=4.6V	-0.9mA
Output low (sink) current VO=0.4V	4.0mA

Inputs

FIG.4.9

BUS0-BUS7

Input high voltage	3.50V min
Input low voltage	1.50V max

All figures worst case at 25°C.

4.5.6 Pinout of the Parallel Port

The port is connected via the unused pins on the 25 way RS232 connector. The RS232 Serial data remains connected as detailed in Section 5.

NOTE: Care should be taken in parallel port-equipped HUSKIES to use only suitably-wired cables.

FIG.4.10

SIGNAL	PIN NO
BUS0	14
BUS1	15
BUS2	16
BUS3	18
BUS4	21
BUS5	23
BUS6	24
BUS7	25
RSTR	11
WSTR	13
SLT	22
DATA	19
ENABLE	17
GND	1

4.5.7

Timing

FIG.4.11 Writing

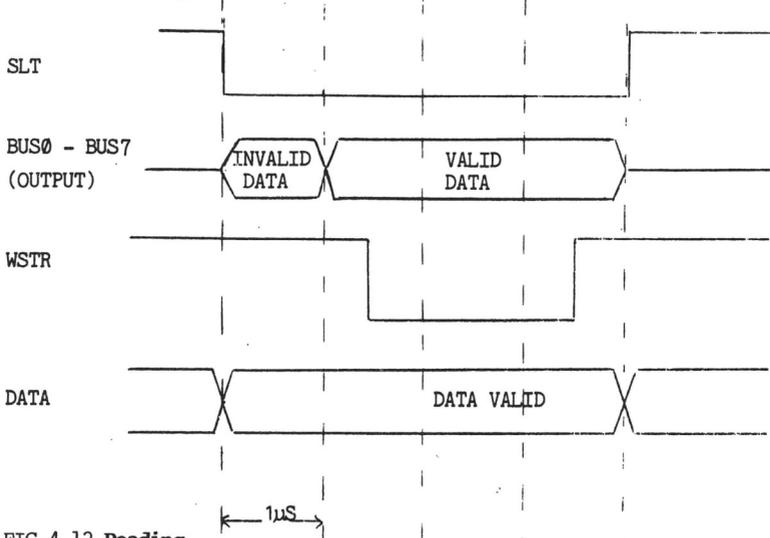
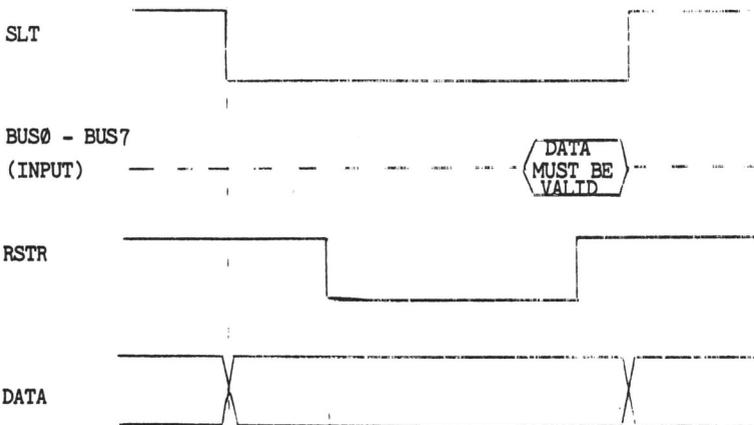


FIG.4.12 Reading



4.5.8 Use in Software

The port is configured at port addresses C0H and ClH (hexadecimal values), 192 and 193 (Decimal Values).

Addressing C0H has the data line low during reading or writing and addressing ClH has the data line high during addressing.

The assembler code access is gained through:

```

OUT (C0H), Writes data
or  OUT (ClH),

IN (A),C0H Reads data
IN (A),ClH

```

The Basic INP or OUT will also access the port, e.g.

```

10 OUT 192,5   Writes 5 to the lower port.
20 A = INP(192) Inputs data into variable A.

```

4.5.9 Using the Interface

A few suggested circuits are shown below.

It is important that the power supplies used are $5V \pm 5\%$. Also the length of the wires used should not exceed 60 cm.

4.5.9.1 Suggested circuit (1)

8 bit parallel Input

In this configuration no external circuitry is required. The desired data may be connected to the BUS0-BUS7 lines. However, the port should not be written to, or a bus conflict will occur.

FIG.4.13

Suggested circuit (2)

8 bit parallel Output

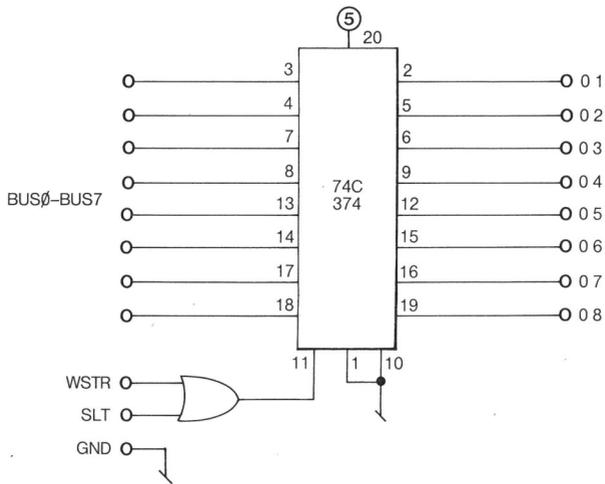
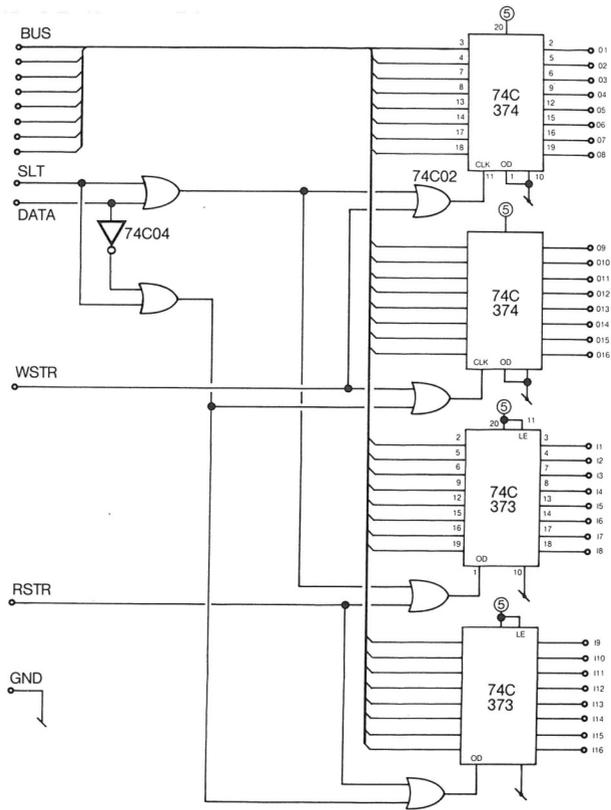


FIG.4.14 16-BIT OUTPUT/16-BIT INPUT CIRCUIT



SINGLE BIT INPUT PORT

4.6

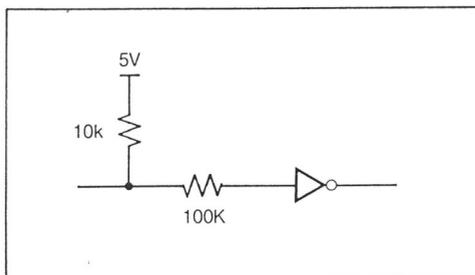
The single input bit has been implemented for use with devices such as bar-code wands, voltage free switch contacts, etc. It is provided via the 4 pin Lemo socket pin 2. Note should be made that it is shared with the CTS signal on the V24 interface (pin 5). It should not, therefore, be used at the same time as the communications when hardware handshaking is in use.

4.6.1

Circuit Implementation

The input is configured as a protected, pulled up CMOS compatible line:

FIG.4.15



It has a logic 1 threshold of $> 3.5V$ and a logic 0 $< 2.5V$. It is protected to $\pm 25V$.

The pull-up resistor enables the use of a switch to ground to provide the input.

No debounce circuitry is provided. If it is required then software must be written for this purpose.

4.6.2

Software Configuration

The input is connected directly to the NSC800 interrupt RSTC.

The normal vector for this is at memory location 02CH, located in EPROM 0. Memory locations 4590H and 4591H have been provided as the vector address. The contents of these locations are the address (lo-hi storage) of the target routine. This is available for Basic to POKE a new address for the user input routine.

The NSC800 will be interrupted with the input at logic 1 or quiescent. The interrupt is also controlled by the NSC800 mask located at port location 0BBH and the interrupt enable and disable instructions. To enable the interrupt firstly the mask must be opened by writing 02H to port 0BBH, and then executing the EI instruction. The bit may be tested by opening the overall mask interrupt and seeing if the interrupt occurs or not.

The interrupt may be used to interrupt Basic.

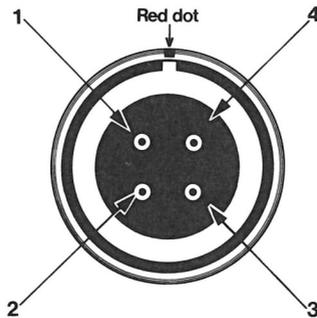
If the interrupt is used and communications are subsequently required, the system call 'revise serial I/O' (Function 40) should be executed.

Alternatively, system call 45 can be used, when the communications will be kept intact. The disadvantage of using this call is that it is relatively slow.

4.6.3 Pin-out of the Lemo Connector

FIG.4.16

PIN NO	SIGNAL
1	: GROUND
2	: CTS or input
3	: Charger/reset
4	: 5V output (Wand option only)



Connector as seen on HUSKY

4.7

ON/OFF KEY

Unlike most equipment, the **HUSKY ON/OFF** switch is not actually in control of power removal. The key is seen by the software scanning the keyboard and the actual decision to turn off is up to the software itself.

If a machine code program places the **HUSKY** into an endless loop, it will be impossible to turn off the machine. This may be overcome by either removing the main battery, or by shorting the Lemo socket pin 3 to the case, which causes a processor reset.

If the program needs to turn off the **HUSKY** then this may be achieved by outputting a 0 to port 131 with:

```
1000 OUT 131,0
```

In some applications, accidental operation of the power key during a critical keyboard operation (data entry, for instance) can be very annoying. The flag location NYMPHO (17724 Decimal) will inhibit power off if set non-zero.

See Section 4.1.4, also Section 3.3.4.21 for details of the 'On

4.8

CLONING HUSKY

The purpose of cloning is to copy the actual memory contents of a **HUSKY** directly into either another **HUSKY** or a receiving device. Conversely, memory may be written into directly by means of cloning allowing the loading of machine code programs directly into the **HUSKY**.

Cloning uses the well known 'Intel Hex' data format and is compatible with most CP/M computers, whose '.HEX' files may be transmitted by means of the PIP program.

4.8.1

Selecting Cloning

To enable cloning in either direction, the main menu selection 'Cloning Husky' should be selected by pressing Enter. This gives a sub-menu of:

- 3 This Husky receiving
- 2 This Husky transmitting
- 1 Transmitting Basic source

Selection of loading gives a warning on the screen that the current memory contents of the **HUSKY** will be destroyed by overwriting with new information. Any other key than 'Y' will abort the selection.

4.8.2

Principle of Cloning

Within the free RAM space the **HUSKY** will generally have all of its Basic program and data. If this is directly copied into another **HUSKY** then the second **HUSKY** will assume all the features and knowledge of the originating **HUSKY**. Of course, the two **HUSKIES** should have the same memory size.

There is, however, a further benefit in that loading the **HUSKY** memory with machine code programs can be achieved with any machine which conforms to the data formats specified below.

4.8.3 Data Format

The **HUSKY** uses the popular Intel Hex format for the data transfer. This format is used by CPM, Intel etc., on their microcomputers. The data is split into blocks with header and trailer information and the data in between. Each data block contains the memory locations into which the data is to be placed.

A single block of data appears as:

```
:BAA0DD.....DDC
```

where

```
B = number of data bytes
A = Address
0 = signifies data record
C = checksum
```

Each byte is transmitted as two hexadecimal characters (0-9 and A-F).

The record starts with a colon, which is detected by the receiving system. This is followed by a count of the data bytes, the load address, a delimiter, the data itself and a checksum on the end. The record is terminated by a carriage return.

The transmission is ended with a null record:

```
:0
```

The checksum is calculated as the modulo 256 addition of all the preceding bytes of information.

The **HUSKY** always transmits records of 16 bytes.

4.8.4 Cloning Data In

When loading the **HUSKY** the address of the information is checked to ensure that it has written the useable address space of the physical memory. Also the checksum is checked. If either of these checks fails, then an error message is displayed:

```
* * Memory overflow * *
```

or

```
* * * Loading error * * *
```

as appropriate.

Reception of the end of record gives:

* * Loading completed * *

The loading may be aborted at any time by pressing ESC.

The length of input data records is optional and is specified by the record up to 255 bytes.

4.8.5

Cloning Data Out

The length of the record is fixed at sixteen data bytes. To start the transmission the **HUSKY** awaits the Enter key to ensure that all connections have been made. The data may be Echoed onto the screen to verify active transmission (see section 5.3.10).

The transmission may be aborted by pressing ESC.

4.8.6

Loading Machine Code Programs

The clone-in program will load any object code into memory. This may then be executed using the **CALL** verb in **BASIC**.

Care should be taken not to overwrite any part of a **BASIC** program or **BASIC**'s data.

4.9

MACHINE CODE PROGRAMMING

The ability to write machine code routines has been mentioned throughout this manual. The use of machine code is desirable from the point of view of increased speed for certain operations, ability to do bit level operations, data compaction for particular applications, etc. Also it may be desired to implement compiled programs within the **HUSKY**, which can be loaded by means of cloning.

4.9.1

Linkage to HUSKY BASIC

BASIC is provided on all standard **HUSKIES**. Machine code routines can be started from Basic, primarily by the CALL verb. The argument passing is detailed in the BASIC Programming Section 3.6.

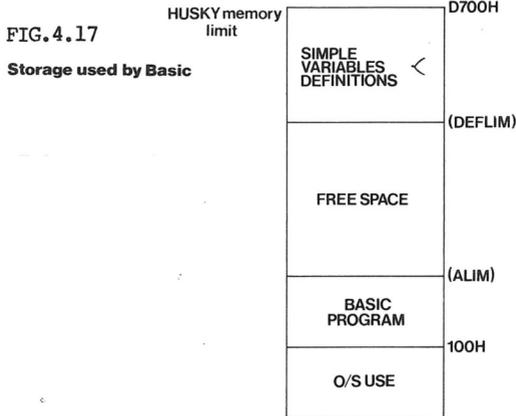
4.9.2

Available Memory

The amount of memory obviously depends upon the **HUSKY**. The system memory has been defined in Section 4.1. For the most part, this should be avoided by the user, apart from locations defined for particular functions.

In some cases, it may be convenient to use the communications buffers for small routines set up by BASIC. This space, located at 4800H (18,432 Decimal) to 4900H (18,688 Decimal) is only used by the system during serial I/O. (If the user is **totally** confident that no serial input will occur). Recall that serial input is transparent to current programs and will write into the input buffer if any data is received. Many field applications will not be using RS232 and can therefore use this space.

Storage used by Basic in **HUSKYs** up to 48K is allocated as follows:



As can be seen program and arrays work up through memory, simple variables and array definitions work down. The gap in the middle is available and unused, the amount being displayed using the FRE verb. The limits are contained in memory locations ALIM and DEFLIM. The address contained at these locations indicates the position of the free space.

It should be remembered that these locations are allocated dynamically and are entirely dependent upon the program and variable storage. Also, if CLEAR or a new program line is inserted then the data storage is removed, so DEFLIM will point to the top of memory and ALIM to the top of the program.

Hence, the program should be RUN before using these values. Clearly, if BASIC is only used to start a large machine code program then there will be no variables and only a small amount of BASIC source.

4.9.3. **System Calls**

The system calls are all defined in Section 4.2.3. They should be used by CALLing location 5 with the relevant value in C for CPM compatibility. NO guarantee is made of the contents of undefined registers on exit and in general they will be destroyed.

4.9.4 **The Stack**

The machine code stack pointer is initialised at switch on. It is not recommended that it should be moved as the system can use a considerable amount of stack space. Under no circumstances should routines be written which preclude the use of interrupts; the V24 I/O uses them liberally.

If a large amount of stack is used which causes the stack to be filled up, then the display will start to contain the overflow. If strange dots and characters start to appear then too much stack is being used!

4.9.5 **ON/OFF**

As detailed in section 8, the ON/OFF button is software controlled. The keyboard should be scanned periodically in any user written software if manual power off is required.

4.9.6 **The NSC800**

It is not proposed to go into software techniques, meaning of object code, etc., here as the NSC800 is totally software compatible to the Z80. A list of the machine code instructions is presented in Appendix I. Users are referred to any tutorial book on the Z80 for detailed programming information.

4.9.7 **Execution Time**

Of interest in machine code loops, etc., is the execution time of the program. The number of cycles used by each instruction is detailed in Appendix I. The cycle time in the HUSKY Is 1 μ S (an enhanced HUSKY with 400nS cycle time is available to special order).

However, the display uses a significant amount of processor time which will slow execution by approximately 17%, depending upon the program being executed.

If serial data is received, then the routines will be considerably slower due to interrupts. However, if no incoming data is expected then the disable interrupts instruction, DI can be used to prevent them. Remember to use an EI (enable interrupt) instruction to re-enable RS-232.

4.9.8 **Paged Systems**

HUSKY's with memory greater than 48K use a transparent paging system under control of Basic. This paging system is not available for direct user control at present and is not compatible with other microcomputers.

User programs can occupy most of page 0 ram, up to 48K Bytes.

Further information is outside the scope of this manual.

Contact Husky Computers if assistance is required.

4.10

PORT ALLOCATIONS

The **HUSKY** uses the NSC800 port map for all its I/O functions, including the keyboard, RS232, etc. The following gives a list of the most useful ports which can be controlled from BASIC using OUT or INP verbs or from machine code.

FIG.4.18

Address		Name	Description
Decimal	Hex		
32	20H	CURSOR	Output with a number 0-7F for the cursor position on the display. The screen software controls this port.
129	81H	V24OUT	Directly outputs to the V24 data line signal on BIT0. The output is voltage inverted i.e. 0=+Ve, 1=-Ve
130	82H	CURINH	Inhibit cursor. BIT0 = 1 : Cursor off BIT0 = 0 : Cursor on
131	83H	POWHL D	Power control. BIT0 = 1 : Husky on BIT0 = 0 : Husky off
132	84H	INVCON	V24 inverter control. BIT0 = 1 : Inverter on i.e. V24 output active BIT0 = 0 : Inverter off i.e. V24 inactive
133	85H	PAGA 16	Memory paging address 16
134	86H	PAGA 17	Memory paging address 17
135	87H	PAGA 18	Memory paging address 18

Address		Name	Description
Decimal	Hex		
187	BBH	ICRREG	NSC800 internal interrupt mask register. See NSC800 handbook.
192	COH	PPORT0	Parallel port 0. Eight bit parallel read/write port. See section 4.6.
193	CLH	PPORT1	Parallel port 1.

4.11

PANIC

4.11.1. INTRODUCTION

HUSKY is designed, built and quality tested for robustness in every sense including robustness of program execution.

Husky's microprocessor system has wide design tolerances (much more than conventional computers) to guarantee absolutely reliable execution of millions upon millions of machine-code instructions every hour. Its physical construction protects it against mechanical disturbance, while the conventional computer's Achilles heel - external electrical interference - is virtually eliminated.

Because of these factors, **HUSKY** is most unlikely ever to mis-execute a user's program or behave in a way that is not thoroughly predictable, given sufficient insight.

4.11.2 CRASHES

However, there are some specific situations that can cause **HUSKY** to mis-execute its internal programming, or "Crash".

The commonest causes are:

4.11.2.1 Illegal System Calls

System calls from Basic are not 'trapped' (this would restrict user programming) and can cause mis-execution if not valid. Ensure that system calls are always precisely in the format indicated in section 4.2 of this manual.

4.11.2.2 Invalid user machine code

User assembly-level programs or subroutines can easily cause mis-execution by containing, for instance, invalid jump instructions.

4.11.2.3 Physical degradation

Ingress of water, corrosion of internal parts, damage to components through excessive shock or persistent high level vibration can reduce electronic tolerances.

4.11.2.4 Operation outside specified temperature range

HUSKY is specified for operation in the range 0-55°C. Operation above 55°C reduces tolerances, while below 0°C battery capacity becomes severely restricted. Sub-zero temperatures also slow LCD screen response time substantially. **HUSKY** is unlikely to mis-execute program simply because of low temperatures, however, in the range 0°C to -20°C.

4.11.3 SYMPTOMS

"Crashes" are identified by these symptoms:

4.11.3.1 Keyboard Lock-out

HUSKY's keyboard is entirely "soft" to allow user re-definition of key functions, including the power key.

If HUSKY's internal program is caused to 'Crash', power control may be lost. **The HUSKY will not switch off.**

4.11.3.2 Clock wipe-out

Following a mis-execution episode, HUSKY's calendar clock registers may be corrupted. The clock's display will probably reset, while the display program (after power-up) may chatter instead of returning regular 'ticks'.

4.11.3.3 Program Corruption

The most serious consequence of mis-execution is when the micro-processor runs 'wild', writing data randomly to all parts of memory and, occasionally, corrupting user programs. This failure is generally catastrophic, and is unlikely to go undetected. The most likely consequence is that any attempt to 'RUN' the corrupted program will result in keyboard lock-out (see above). Attempts to 'LIST' corrupted programs may also result in lock-out.

4.11.3.4 I/O Corruption

I/O selections may be affected by mis-executions in a random fashion. In some cases spurious options may appear in the parameter selection screens after a 'Crash'. In this event, hold down the option selection (| | keys) until recognisable options appear.

4.11.4 RECOVERY

4.11.4.1 Program Corruption

If a mis-execution has occurred, any user program stored in RAM memory must be viewed with suspicion. The safest solution, assuming that the use of keyboard is still available, is to type 'NEW', followed by re-loading of the program.

Remember that even 'LIST' may result in keyboard lockout, although holding the power key down will generally restore control eventually.

Most types of corruption (corrupted lines, spurious line numbers) are unrecoverable in Basic and can only be eliminated by 'NEW'.

4.11.4.2 Keyboard Lock-out

If keyboard lock-out (failure to power down) occurs, there are two possible solutions:

- 1) Apply 'RESET' by shorting LEMO connector pins 1 and 3.
- 2) Remove the main battery plug, disconnecting HUSKY's power supply.

In both cases, HUSKY should power-up again normally. If desired, try running the user program, but be ready to clear any lock-out that occurs using the above methods. If lock-out persists, type 'NEW' after restoring operation.

4.11.4.3 System Lock-out

In very rare cases, HUSKY may fail to restart correctly after a mis-execution episode. This is possible in a program that uses auto-start (section 3.1.4 in Part 3, Basic Programming) and contains an invalid system call.

Any program corruption resulting could leave the auto-start flag set, but cause a lock-out when power-up is attempted.

The solution is to apply the 'ESC' sequence immediately after power-up, in order to reset the auto-start flag.

KEYBOARD

4.12

4.12.1

Introduction

ALL **HUSKY** computers are supplied with a standard general-purpose keyboard (Fig.4.19), unless specifically ordered otherwise. Users are encouraged to design customised keyboard layouts tailored to their application, especially if dedicated function keys are likely to be an advantage.

Since **HUSKY's** keyboard graphics panel is a peel-off self adhesive overlay, replacement of the keyboard graphics is an easy matter. Simply prise up the corner of the overlay panel and peel off, but be careful not to crease the panel if you intend to use it again. Replacement panels to user's designs are easily printed by Husky Computers. Please see the 'custom keyboard order form' at the rear of this section.

Keyboard overlays are reverse - screen printed on polycarbonate material 0.005" (0.127 mm) thick and are proof against wear and most forms of accidental damage.

Some samples of specialised keyboard layouts are shown in Figs. 4.20 - 4.23

NOTE: **Husky's** standard keyboard layout was changed at the end of 1982 to improve ergonomic factors. The layout for current production **Husky's** is shown in Fig. 4.19. Units made before this date may have the layout shown in Fig. 4.20 - the 'numeric keypad' layout.

FIG.4.19

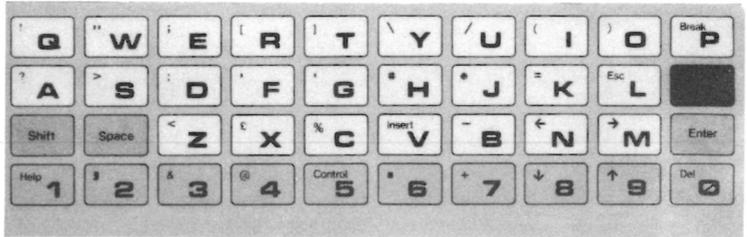


FIG.4.20

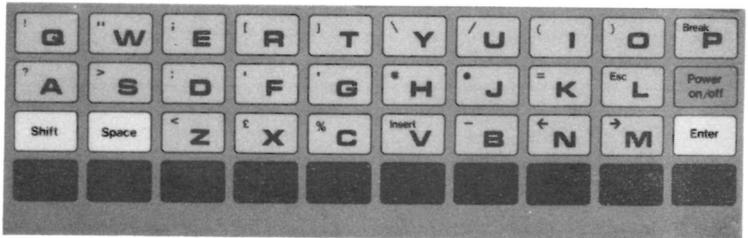


FIG. 4.21

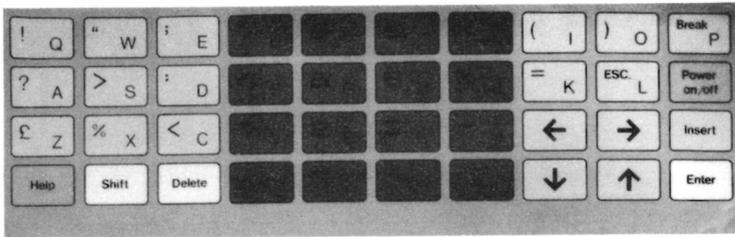


FIG. 4.22

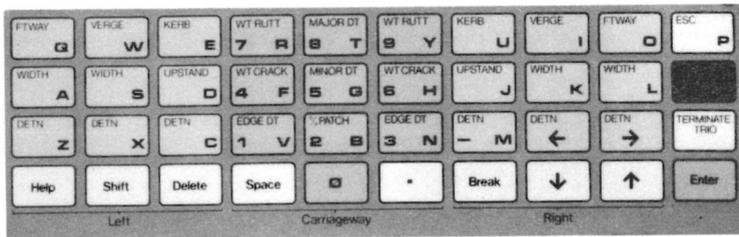
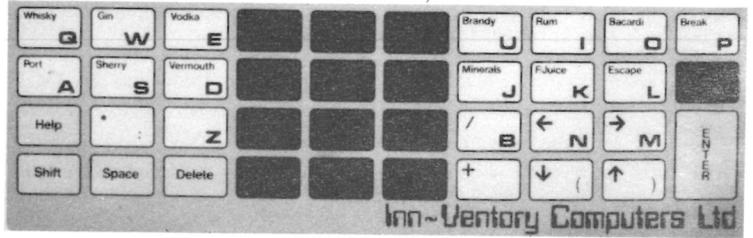


FIG.4.23



4.12.2 Electrical re-definition

Keyboard keys are each equated to the characters they represent by unique numerical codes, as listed in Part 3, Appendix I. These values are defined according to the American Standard Code for Information Interchange (ASCII).

The value returned to BASIC in, for instance, an INCHR statement is the numerical equivalent of the key pressed.

The ASCII definitions of each key for the keyboard are contained in eighty consecutive RAM locations named KEYBUF (4600H-decimal 17920). The upper shift keys are defined in the first forty locations, followed by the forty lower shift keys. These memory locations form a 'map' of the physical keyboard, as shown in Fig. 4.25.

On power up, the HUSKY automatically defines these locations to a standard configuration. However, the programmer has the option to be able to define special keyboard arrangements.

The schematic of the keyboard shows the offset value in KEYBUFF for each key.

As an example, consider the key fourth from the left on the top row. If it was required to program this key to be \$ in upper case and S in lower case, two Poke operations would be required.

```
POKE 17926,36  
POKE 17966,83
```

It is important to note that on powering up the HUSKY, the keyboard will revert to the standard version. The programmer should define the special keyboard requirements at an early stage in the program and ensure that the re-definition occurs each time the program is run.

NOTE: The power on/off key is fixed and cannot be moved.

FIG. 4.25

! Q	" W	: E	[R	\ Y	/ U	(I) O	Break P
? A	> S	: D	' F	* H	* J	= K	Esc L	Power on/off
Shift	Space	< Z	£ X	Insert V	- B	← N	→ M	Enter
Help 1	' 2	& 3	@ 4	' 6	+ 7	↓ 8	↑ 9	Del Ø

Upper Case	17929	17928	17927	17926	17925	17924	17923	17922	17921	17920
Lower Case	17969	17968	17967	17966	17965	17964	17963	17962	17961	17960
Upper Case	17939	17938	17937	17936	17935	17934	17933	17932	17931	
Lower Case	17979	17978	17977	17976	17975	17974	17973	17972	17971	
Upper Case	17949	17948	17947	17946	17945	17944	17943	17942	17941	17940
Lower Case	17989	17988	17987	17986	17985	17984	17983	17982	17981	17980
Upper Case	17959	17958	17957	17956	17955	17954	17953	17952	17951	17950
Lower Case	17999	17998	17997	17996	17995	17994	17993	17992	17991	17990

Memory locations (in decimal) which define key functions.

POWER KEY (NOT PROGRAMMABLE)

4.12.3. Special Codes

Special codes are assigned to keys whose functions are:

- (1) Control key
- (2) Shift key (latched)
- (3) Help key
- (4) Momentary shift key

In order for the operating system to detect a change of these keys, the SPELFLG (decimal 18000) location must be cleared. Changes modifying the special codes should only be done under software control, not by direct POKE statements.

NOTE: 'Enter' is not a special code: it is simply CR (Decimal 13) !

LATCHED SHIFT CODE 129 (81H)

This key should be defined in both shift halves in order to function correctly! This function gives a 'toggle' action, changing shift with each operation.

MOMENTARY SHIFT CODE 130 (82H)

As HUSKY powers up in the lower shift, it is important for the momentary shift code be defined in the lower shift half of KEYBUF. This function gives a 'conventional' shift requiring simultaneous depression of shift and the desired key.

CONTROL CODE 132 (84H)

On detection of this key the control code for the key pressed is derived from the code located in the upper shift half of KEYBUF, if this key has a valid 'control' equivalent.

CONTROL CODE 133 (85H)

On detection of this key the control code for the key pressed is derived from the code located in the lower shift half of KEYBUF, if this key has a valid 'control' equivalent.

HELP CODE 134 (86H)

The detection of this key causes the firmware to enter the HELP text mode. Help mode is exited by typing 'Enter'.

4.12.4 **Example Keyboards**

Following are some examples of special keyboards set up by the programs shown. Note that these program lines need to be executed every time the HUSKY is powered up.

KEYBOARD 1

```

REM TEST KEYBOARD ROUTINE
REM GENERATES 'NUMERIC KEYPAD' KEYBOARD (Fig.4.20)
10 POKE 17920,31,41,40,47,57,56,55,59,34,33
20 POKE 17930,0,27,61,42,54,53,52,58,62,63
30 POKE 17940,30,12,8,45,51,50,49,60,37,36
40 POKE 17950,13,11,14,43,46,48,32,127,129,134
50 POKE 17960,80,79,73,85,89,84,82,69,87,81
60 POKE 17970,0,76,75,74,72,71,70,68,83,65
70 POKE 17980,30,12,8,77,78,66,86,67,88,90
80 POKE 17990,13,11,14,133,46,44,32,127,129,134
90 POKE 18000,0
100 END

```

KEYBOARD 2

```

REM TEST KEYBOARD ROUTINE
REM GENERATES 'NUMERIC KEYPAD' KEYBOARD WITH MOMENTARY SHIFT
10 POKE 17920,31,41,40,47,57,56,55,59,34,33
20 POKE 17930,0,27,61,42,54,53,52,58,62,63
30 POKE 17940,30,12,8,45,51,50,49,60,37,36
40 POKE 17950,13,11,14,43,46,48,32,127,130,134
50 POKE 17960,80,79,73,85,89,84,82,69,87,81
60 POKE 17970,0,76,75,74,72,71,70,68,83,65
70 POKE 17980,30,12,8,77,78,66,86,67,88,90
90 POKE 17990,13,11,14,133,46,44,32,127,130,134
100 END

```

KEYBOARD 3

```

20000 REM UPPER/LOWER CASE KEYBOARD SUBROUTINE
20010 REM CALL 21000 FOR UPPER/LOWER CASE
21000 POKE 17960,112,111,105,117,121,116,114,101,119,113
21010 POKE 17971,108,107,106,104,103,102,100,115,97
21020 POKE 17981,109,110,98,118,99,120,122
21050 POKE 17920,80,79,73,85,89,84,82,69,87,81
21060 POKE 17931,76,75,74, 72,71,70,68,83,65
21070 POKE 17941,77,78,66,86,67,88,90
21090 RETURN

```

CUSTOMER KEYBOARD ORDER FORM

Customer name:

Date:

Address:

Order No.

REQUIREMENT

Number of machines:

Colour of machine:

Is keyboard layout as standard?

Yes No

Number of different colours on keyboard:

Please specify on layout – see reverse.

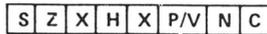
NSC800 MACHINE CODE

INSTRUCTION	C	Z	P/V	S	N	H	COMMENTS
ADD A, s; ADC A, s	:	:	V	:	0	:	8-bit add or add with carry
SUB s; SBC A, s, CP s, NEG	:	:	V	:	1	:	8-bit subtract, subtract with carry, compare and negate accumulator
AND s	0	:	P	:	0	1	Logical operations
OR s, XOR s	0	:	P	:	0	0	And sets different flags
INC s	•	:	V	:	0	:	8-bit increment
DEC m	•	:	V	:	1	:	8-bit decrement
ADD DD, ss	:	•	•	•	0	X	16-bit add
ADC HL, ss	:	•	•	•	0	X	16-bit add with carry
SBC HL, ss	:	•	•	•	1	X	16-bit subtract with carry
RLA; R'CA, RRA, RRCA	:	•	•	•	0	0	Rotate accumulator
RL m; PLC m; RR m; R5C m	:	•	•	•	0	0	Rotate and shift location s
SLA m; SRA m; SRL m	•	•	•	•	0	0	Rotate digit left and right
RLD, RRD	•	•	P	:	0	0	Decimal adjust: accumulator
DAA	•	•	P	:	•	:	Complement accumulator
CPL	•	•	•	:	1	1	Set carry
SCF	1	•	•	•	0	0	Complement carry
CCF	:	•	•	•	0	X	Input register indirect
IN r, (C)	•	:	P	:	0	0	Block input and output
INI; IND; OUT; OUTD	•	:	X	X	1	X	Z = 0 if B ≠ 0 otherwise Z = 1
INIR; INDR; OTIR; OTDR	•	1	X	X	1	X	Block transfer instructions
LDI, LDD	•	X	:	X	0	0	P/V = 1 if BC ≠ 0, otherwise P/V = 0
LDIR, LDDR	•	X	0	X	0	0	Block search instructions
CPI, CPIR, CPD, CPDR	•	:	:	X	1	X	Z = 1 if A = (HL); otherwise Z = 0 P/V = 1 if BC ≠ 0, otherwise P/V = 0
LD A, I; LD A, R	•	:	IFF	:	0	0	The content of the interrupt enable flip-flop (IFF) is copied into the P/V flag
BIT b, s	•	:	X	X	0	1	The complement of bit b of location is copied into the Z flag
NEG	:	:	V	:	1	:	Negate accumulator

The following notation is used in this table:

SYMBOL	OPERATION
C	Carry/link flag. C=1 if the operation produced a carry from the MSB of the operand or result.
Z	Zero flag. Z=1 if the result of the operation is zero.
S	Sign flag. S=1 if the MSB of the result is one.
P/V	Parity or overflow flag. Parity (P) and overflow (V) share the same flag. Logical operations affect this flag with the parity of the result while arithmetic operations affect this flag with the overflow of the result. If P/V holds parity, P/V=1 if the result of the operation is even, P/V=0 if result is odd. If P/V holds overflow, P/V=1 if the result of the operation produced an overflow.
H	Half-carry flag. H=1 if the add or subtract operation produced a carry into or borrow from bit 4 of the accumulator.
N	Add/Subtract flag. N=1 if the previous operation was a subtract.
	H and N flags are used in conjunction with the decimal adjust instruction (DAA) to properly correct the result into packed BCD format following addition or subtraction using operands with packed BCD format.
:	The flag is affected according to the result of the operation.
•	The flag is unchanged by the operation.
0	The flag is reset by the operation.
1	The flag is set by the operation.
X	The flag is a "don't care."
V	P/V flag affected according to the overflow result of the operation
P	P/V flag affected according to the parity result of the operation.
r	Any one of the CPU registers A, B, C, D, E, H, L.
s	Any 8-bit location for all the addressing modes allowed for the particular instruction.
ss	Any 16-bit location for all the addressing modes allowed for that instruction.
ii	Any one of the two index registers IX or IY.
R	Refresh counter.
n	8-bit value in range <0, 255>.
nn	16-bit value in range <0, 65535>.
m	Any 8-bit location for all the addressing modes allowed for the particular instruction.

SUMMARY OF FLAG OPERATION



Sequence of flags in F register

APPENDIX I

MNEMONIC	SYMBOLIC OPERATION	FLAGS					OP-CODE 76 543 210	NO OF T CYCLES	COMMENTS	
		C	P/V	S	N	H			r, r'	Reg
LD r, r'	r - r'	*	*	*	*	*	01 r r'	4	r, r'	Reg
LD r, n	r - n	*	*	*	*	*	00 r 110	7	000	B
LD r, (HL)	r - (HL)	*	*	*	*	*	01 r 110	7	010	C
LD r, (IX+d)	r - (IX+d)	*	*	*	*	*	11 011 101	19	011	D
							01 r 110		100	H
							- d -		101	L
LD r, (IY+d)	r - (IY+d)	*	*	*	*	*	11 111 101	19	111	A
							01 r 110			
							- d -			
LD (HL), r	(HL) - r	*	*	*	*	*	01 110 r	7		
LD (IX+d), r	(IX+d) - r	*	*	*	*	*	11 011 101	19		
							01 110 r			
							- d -			
LD (IY+d), r	(IY+d) - r	*	*	*	*	*	11 111 101	19		
							01 110 r			
							- d -			
LD (HL), n	(HL) - n	*	*	*	*	*	00 110 110	10		
							- n -			
LD (IX+d), n	(IX+d) - n	*	*	*	*	*	11 011 101	19		
							00 110 110			
							- d -			
							- n -			
LD (IY+d), n	(IY+d) - n	*	*	*	*	*	11 111 101	19		
							00 110 110			
							- d -			
							- n -			
LD A, (BC)	A - (BC)	*	*	*	*	*	00 001 010	7		
LD A, (DE)	A - (DE)	*	*	*	*	*	00 011 010	7		
LD A, (nn)	A - (nn)	*	*	*	*	*	00 111 010	13		
							- n -			
							- n -			
LD (BC), A	(BC) - A	*	*	*	*	*	00 000 010	7		
LD (DE), A	(DE) - A	*	*	*	*	*	00 010 010	7		
LD (nn), A	(nn) - A	*	*	*	*	*	00 110 010	13		
							- n -			
							- n -			
LD A, I	A - I	*	*	IFF	:	0 0	11 101 101	9		
							01 010 111			
LD A, R	A - R	*	*	IFF	:	0 0	11 101 101	9		
							01 011 111			
LD I, A	I - A	*	*	*	*	*	11 101 101	9		
							01 000 111			
LD R, A	R - A	*	*	*	*	*	11 101 101	9		
							01 001 111			
LD dd, nn	dd - nn	*	*	*	*	*	00 dd0 001	10	dd	Parr
							- n -		00	BC
							- n -		01	DE
LD IX, nn	IX - nn	*	*	*	*	*	11 011 101	14	10	HL
							00 100 001		11	SP
							- n -			
							- n -			
LD IY, nn	IY - nn	*	*	*	*	*	11 111 101	14		
							00 100 001			
							- n -			
							- n -			
LD HL, (nn)	H - (nn+1) L - (nn)	*	*	*	*	*	00 101 010	16		
							- n -			
							- n -			
LD dd, (nn)	dd _H - (nn+1) dd _L - (nn)	*	*	*	*	*	11 101 101	20		
							01 dd1 011			
							- n -			
							- n -			
LD IX, (nn)	IX _H - (nn+1) IX _L - (nn)	*	*	*	*	*	11 011 101	20		
							00 101 010			
							- n -			
							- n -			
LD IY, (nn)	IY _H - (nn+1) IY _L - (nn)	*	*	*	*	*	11 111 101	20		
							00 101 010			
							- n -			
							- n -			
LD (nn), HL	(nn+1) - H (nn) - L	*	*	*	*	*	00 100 010	16		
							- n -			
							- n -			

MNEMONIC	SYMBOLIC OPERATION	FLAGS						OP-CODE 76 543 210	NO OF T CYCLES	COMMENTS
		C	Z	P/V	S	N	H			
LD (nn), dC	(nn+1) - d _H 'nn) - d _L	*	*	*	*	*	*	11 101 101 01 d60 011 - n - - n -	20	
LD (nn), IX	(nn+1) - IX _H (nn) - IX _L	*	*	*	*	*	*	11 011 101 00 100 010 - n - - n -	20	
LD (nn), IY	(nn+1) - IY _H (nn) - IY _L	*	*	*	*	*	*	11 111 101 00 100 010 - n - - n -	20	
LD SP, HL	SP - HL	*	*	*	*	*	*	11 111 001	6	
LD SP, IX	SP - IX	*	*	*	*	*	*	11 011 101 11 111 001	10	
LD SP, IY	SP - IY	*	*	*	*	*	*	11 111 101 11 111 001	10	
PUSH qq	(SP-2) - qq _L (SP-1) - qq _H	*	*	*	*	*	*	11 qq0 101	11	qq Par 00 BC
PUSH IX	(SP-2) - IX _L (SP-1) - IX _H	*	*	*	*	*	*	11 011 101 11 111 101	15	10 HL 11 AF
PUSH IY	(SP-2) - IY _L (SP-1) - IY _H	*	*	*	*	*	*	11 100 101 11 100 101	15	
POP qq	qq _L - (SP+1) qq _H - (SP)	*	*	*	*	*	*	11 qq0 001	10	
POP IX	IX _H - (SP+1) IX _L - (SP)	*	*	*	*	*	*	11 011 101 11 100 001	14	
POP IY	IY _H - (SP+1) IY _L - (SP)	*	*	*	*	*	*	11 111 101 11 100 001	14	
EX DE, HL	DE - HL	*	*	*	*	*	*	11 101 011	4	
EX AF, AF'	AF - AF'	*	*	*	*	*	*	00 001 000	4	
EXX	BC - BC DE - DE HL - HL H - (SP+1) L - (SP)	*	*	*	*	*	*	11 011 001	4	Register bank and auxiliary register bank exchange
EX (SP), HL	H - (SP+1) L - (SP)	*	*	*	*	*	*	11 100 011	19	
EX (SP), IX	IX _H - (SP+1) IX _L - (SP)	*	*	*	*	*	*	11 011 101 11 100 011	23	
EX (SP), IY	IY _H - (SP+1) IY _L - (SP)	*	*	*	*	*	*	11 111 101 11 100 011	23	
LDI	(DE) - (HL) DE - DE+1 HL - HL+1 BC - BC-1	*	*	1	*	0	0	11 101 101 10 100 000	16	Load (HL) into (DE), increment the pointers and decrement the byte counter (BC)
LDIR	(DE) - (HL) DE - DE+1 HL - HL+1 BC - BC-1 Repeat until BC = 0	*	*	0	*	0	0	11 101 101 10 110 000	21 16	If BC ≠ 0 If BC = 0
LDD	(DE) - (HL) DE - DE-1 HL - HL-1 BC - BC-1	*	*	1	*	0	0	11 101 101 10 101 000	16	
LDDR	(DE) - (HL) DE - DE-1 HL - HL-1 BC - BC-1 Repeat until BC = 0	*	*	0	*	0	0	11 101 101 10 111 000	21 16	If BC ≠ 0 If BC = 0
CPI	A - (HL) HL - HL+1 BC - BC-1	*	1	1	1	1	1	11 101 101 10 100 001	16	
CPIR	A - (HL) HL - HL+1 BC - BC-1 Repeat until A = (HL) or BC = 0	*	1	1	1	1	1	11 101 101 10 110 001	21 16	If BC ≠ 0 or A ≠ (HL) If BC = 0 or A = (HL)
CPD	A - (HL) HL - HL-1 BC - BC-1	*	1	1	1	1	1	11 101 101 10 101 001	16	

MNEMONIC	SYMBOLIC OPERATION	FLAGS					OP-CODE 76 543 210	NO OF T CYCLES	COMMENTS
		C	Z	P/V	S	N			
CPDR	A - (HL)	*	1	1	1	1	11 101 101	21	If BC = 0 and A ≠ (HL) If BC = 0 or A = (HL)
	HL - HL-1 BC - BC-1 Repeat until A = (HL) or BC = 0						10 111 001	16	
ADD A, r	A + A + r	:	:	V	:	:	10 000 r	4	r Reg
ADD A, n	A + A + n	:	:	V	:	:	11 000 110	7	000 B 001 C 010 D 011 E 100 H 101 L 111 A
ADD A, (HL)	A + A + (HL)	:	:	V	:	:	10 000 110	7	
ADD A, (IX+d)	A + A + (IX+d)	:	:	V	:	:	11 011 101 10 000 110 d	19	
ADD A, (IY+d)	A + A + (IY+d)	:	:	V	:	:	11 111 101 10 000 110 d	19	
ADC A, s	A + A + s + CV	:	:	V	:	:	001		s is any of r, n, (HL), (IX+d), (IY+d) as shown for ADD instruction
SUB s	A - A - s	:	:	V	:	:	010		
SBC A, s	A - A - s - CV	:	:	V	:	:	011		
AND s	A & A & s	0	:	P	:	:	110		
OR s	A A s	0	:	P	:	:	110		
XOR s	A ^ A ^ s	0	:	P	:	:	101		The indicated bits replace the 000 in the ADD set above
CP s	A - s	:	:	V	:	:	111		
INC r	r + r + 1	*	:	V	:	:	00 r 100	4	
INC (HL)	(HL) - (HL) + 1	*	:	V	:	:	00 110 100	11	
INC (IX+d)	(IX+d) - (IX+d) + 1	*	:	V	:	:	11 011 101 00 110 100 d	23	
INC (IY+d)	(IY+d) - (IY+d) + 1	*	:	V	:	:	11 111 101 00 110 100 d	23	
DEC m	m - m - 1	*	:	V	:	:	101		m is any of r, (HL), m (IX+d), (IY+d) as shown for INC Same format and states as INC Replace 100 with 101 in OP code
ADD HL, ss	HL - HL + ss	*	*	*	*	0 X	00 ss 1 001	11	ss Reg 00 BC 01 DE 10 HL 11 SP
ADC HL, ss	HL - HL + ss + CV	:	:	V	:	0 X	11 101 101 01 ss 1 010	15	
SBC HL, ss	HL - HL - ss - CV	:	:	V	:	1 X	11 101 101 01 ss 0 010	15	
ADD IX, pp	IX - IX + pp	*	*	*	*	0 X	11 011 101 00 pp 1 001	15	pp Reg 00 BC 01 DE 10 IX 11 SP
ADD IY, rr	IY - IY + rr	*	*	*	*	0 X	11 111 101 00 rr 1 001	15	rr Reg 00 BC 01 DE 10 IY 11 SP
INC ss	ss + ss + 1	*	*	*	*	*	00 ss 0 011	6	
INC IX	IX - IX + 1	*	*	*	*	*	11 011 101 00 100 011	10	
INC IY	IY - IY + 1	*	*	*	*	*	11 111 101 00 100 011	10	
DEC ss	ss - ss - 1	*	*	*	*	*	00 ss 1 011	6	
DEC IX	IX - IX - 1	*	*	*	*	*	11 011 101 00 101 011	10	
DEC IY	IY - IY - 1	*	*	*	*	*	11 111 101 00 101 011	10	
RLCA		:	*	*	*	0 0	00 000 111	4	Rotate left circular accumulator
RLA		:	*	*	*	0 0	00 010 111	4	Rotate left accumulator
RRCA		:	*	*	*	0 0	00 001 111	4	Rotate right circular accumulator
RRA		:	*	*	*	0 0	00 011 111	4	Rotate right accumulator

MNEMONIC	SYMBOLIC OPERATIONS	FLAGS						OP-CODE 76 543 210	NO OF T CYCLES	COMMENTS
		C	Z	PV	S	H	N			
RLC r		1	1	P	1	0	0	11 001 011 00 [000] r	8	Rotate left circular register r
RLC (HL)		1	1	P	1	0	0	11 001 011 00 [000] 110	15	r Reg 000 B 001 C 010 D 011 E 100 H 101 L 111 A
RLC (IX+d)		1	1	P	1	0	0	11 011 101 11 001 011 - d - 00 [000] 110	23	
RLC (IY+d)		1	1	P	1	0	0	11 111 101 11 001 011 - d - 00 [000] 110	23	
RL m		1	1	P	1	0	0	00 [000] 110 010		Instruction format and states are as shown for RLCs. To form new OP-code replace 000 of RLCs with shown code
RRC m		1	1	P	1	0	0	[001]		
RR m		1	1	P	1	0	0	[011]		
SLA m		1	1	P	1	0	0	[100]		
SRA m		1	1	P	1	0	0	[101]		
SRL m		1	1	P	1	0	0	[111]		
RLD		*	1	P	1	0	0	11 101 101 01 101 111	18	Rotate digit left and right between the accumulator and location (HL). The content of the upper half of the accumulator is unaffected.
RRD		*	1	P	1	0	0	11 101 101 01 100 111	18	
DAA	Converts acc. content into packed BCD following add or subtract with packed BCD operands	1	1	P	1	*	1	00 100 111	4	Decimal adjust accumulator.
CPL	A ← A̅	*	*	*	*	*	1	00 101 111	4	Complement accumulator (one's complement)
NEG	A ← o-A	1	1	V	1	1	1	11 101 101 01 000 100	8	Negate acc. (two's complement)
CCF	CY ← CY̅	1	*	*	*	*	X	00 111 111	4	Complement carry flag
SCF	CY ← 1	1	*	*	*	*	0	00 110 111	4	Set carry flag
NOP	No operation	*	*	*	*	*	*	00 000 000	4	
HALT	CPU halted	*	*	*	*	*	*	01 110 110	4	
DI	IEF ← 0	*	*	*	*	*	*	11 110 011	4	
EI	IEF ← 1	*	*	*	*	*	*	11 111 011	4	
IM0	Set interrupt mode 0	*	*	*	*	*	*	11 101 101 01 000 110	8	
IM1	Set interrupt mode 1	*	*	*	*	*	*	11 101 101 01 010 110	8	
IM2	Set interrupt mode 2	*	*	*	*	*	*	11 101 101 01 011 110	8	
BIT b, r	Z ← Z _b	*	1	X	X	0	1	11 001 011 01 b r	8	r Reg 000 B 001 C 010 D 011 E 100 H 101 L 111 A
BIT b, (HL)	Z ← (HL) _b	*	1	X	X	0	1	11 001 011 01 b 110	12	
BIT b, (IX+d)	Z ← (IX+d) _b	*	1	X	X	0	1	11 011 101 - d - 11 001 011	20	
BIT b, (IY+d)	Z ← (IY+d) _b	*	1	X	X	0	1	11 111 101 - d - 11 001 011 01 b 110	20	b Bit Tested 000 0 001 1 010 2 011 3 100 4 101 5 110 6 111 7

MNEMONIC	SYMBOLIC OPERATION	FLAGS					OP-CODE 76 543 210	NO. OF T CYCLES	COMMENTS	
		C	Z	P/V	S	N				H
SET b, r	b - 1	•	•	•	•	•	•	11 001 011	8	
SET b, (HL)	(HL) _b - 1	•	•	•	•	•	•	11 b 0 r 11 001 011	15	
SET b, (IX+d)	(IX+d) _b - 1	•	•	•	•	•	•	11 b 110 11 011 101	23	
SET b, (IY-d)	(IY-d) _b - 1	•	•	•	•	•	•	11 001 011 - d - 11 b 110 11 111 101	23	
RES b, m	b ₀ = 0 m = r, (HL), (IX+d), (IY+d)	•	•	•	•	•	•	11 b 110 10		To form new OP-code replace 11 of SET b, i with 10. Flags and time states for SET instruction
JP nn	PC - nn	•	•	•	•	•	•	11 000 011 - n - - n -	10	
JP cc, nn	If condition is true PC - nn, otherwise continue	•	•	•	•	•	•	11 cc 010 - n - - n -	10	cc Condition 000 NZ non zero 001 Z zero 010 NC non carry 011 C carry 100 PO parity odd 101 PE parity even 110 P sign positive 111 M sign negative
JR e	PC - PC + e	•	•	•	•	•	•	00 011 000 - e-2 -	12	
JR C, e	If C = 0, continue If C = 1, PC - PC + e	•	•	•	•	•	•	00 111 000 - e-2 -	7	If condition not met
JR NC, e	If C = 1, continue If C = 0, PC - PC + e	•	•	•	•	•	•	00 110 000 - e-2 -	7	If condition not met
JR Z, e	If Z = 0 continue If Z = 1, PC - PC + e	•	•	•	•	•	•	00 101 000 - e-2 -	7	If condition not met
JR NZ, e	If Z = 1, continue If Z = 0, PC - PC + e	•	•	•	•	•	•	00 100 000 - e-2 -	7	If condition not met
JP (HL)	PC - HL	•	•	•	•	•	•	11 101 001	4	
JP (IX)	PC - IX	•	•	•	•	•	•	11 011 101	8	
JP (IY)	PC - IY	•	•	•	•	•	•	11 101 001 11 111 101 11 101 001	8	
DJNZ e	B - B - 1 If B = 0, continue If B ≠ 0, PC - PC + e	•	•	•	•	•	•	00 010 000 - e-2 -	8	If B = 0
CALL nn	(SP-1) = PC _H (SP-2) = PC _L PC - nn	•	•	•	•	•	•	11 001 101 - n - - n -	17	
CALL cc, nn	If condition cc is false continue, otherwise same as CALL nn	•	•	•	•	•	•	11 cc 100 - n - - n -	10	If cc is false
RET	PC _L = (SP) PC _H = (SP+1)	•	•	•	•	•	•	11 001 001	10	
RET cc	If condition cc is false continue, otherwise same as RET	•	•	•	•	•	•	11 cc 000	5	If cc is false
RETI	Return from interrupt	•	•	•	•	•	•	11 101 101	14	
RETN	Return from non maskable interrupt	•	•	•	•	•	•	01 001 101 11 101 101 01 000 101	14	

MNEEMONIC	SYMBOLIC OPERATION	FLAGS					OP-CODE			NO OF T CYCLES	COMMENTS
		C	Z	P/V	S	N	76	543	210		
RST p	[SP-1] - PC _H [SP-2] - PC _L PC _H ← 0 PC _L ← p	*	*	*	*	*	11	r	111	11	
IN A, (n)	A ← (n)	*	*	*	*	*	11	011	011	11	n to A ₀ A ₇ Acc to A ₈ A ₁₅
IN r, (C)	r ← (C) If r = 110 only the flags will be affected	*	*	P	:	0	11	101	101	12	C to A ₀ A ₇ B to A ₈ A ₁₅
INI	(HL) ← (C) B ← B-1 HL ← HL + 1	*	:	X	X	1	X	10	101	101	C to A ₀ A ₇ B to A ₈ A ₁₅
INIR	(HL) ← (C) B ← B-1 HL ← HL + 1 Repeat until B = 0	*	1	X	X	1	X	10	110	010	C to A ₀ A ₇ B to A ₈ A ₁₅
IND	(HL) ← (C) B ← B-1 HL ← HL-1	*	:	X	X	1	X	10	101	101	C to A ₀ A ₇ B to A ₈ A ₁₅
INDR	(HL) ← (C) B ← B-1 HL ← HL-1 Repeat until B = 0	*	1	X	X	1	X	10	111	010	C to A ₀ A ₇ B to A ₈ A ₁₅
OUT (n), A	(n) ← A	*	*	*	*	*	11	010	011	11	n to A ₀ A ₇ Acc to A ₈ A ₁₅
OUT (C), r	(C) ← r	*	*	*	*	*	11	101	101	12	C to A ₀ A ₇ B to A ₈ A ₁₅
OUTI	(C) ← (HL) B ← B-1 HL ← HL + 1	*	:	X	X	1	X	10	100	011	C to A ₀ A ₇ B to A ₈ A ₁₅
OTIR	(C) ← (HL) B ← B-1 HL ← HL + 1 Repeat until B = 0	*	1	X	X	1	X	10	110	011	C to A ₀ A ₇ B to A ₈ A ₁₅
OUTD	(C) ← (HL) B ← B-1 HL ← HL-1	*	:	X	X	1	X	10	101	011	C to A ₀ A ₇ B to A ₈ A ₁₅
OTDR	(C) ← (HL) B ← B-1 HL ← HL-1 Repeat until B = 0	*	1	X	X	1	X	10	111	011	C to A ₀ A ₇ B to A ₈ A ₁₅

Notes: r, r' means any of the registers A, B, C, D, E, H, L
 ss is any of the register pairs BC, DE, HL, SP
 rr is any of the register pairs BC, DE, IX, SP
 (1) P/V flag is 0 if the result of BC ← 0, otherwise P/V = 1
 (2) Z flag is 1 if A = (HL), otherwise Z = 0
 (3) If the result of B ← 0, the Z flag is set, otherwise it is reset
 e represents the extension in the relative addressing mode
 e is a signed two's complement number in the range: -126, 129
 e-2 in the op-code provides an effective address of pcre as PC is incremented by 2 prior to the addition of e
 The notation s₆ indicates bit 6 (0 to 7) of location s

Flag Notation: * = flag not affected, 0 = flag reset, 1 = flag set, X = flag is unknown
 : = flag is affected according to the result of the operation

Z80—CPU INSTRUCTION SET

OBJ CODE	SOURCE STATEMENT	OPERATION	
8E DD8E05 FD8E05 8F 88 89 8A 8B 8C 8D CE20	ADC ADC ADC ADC ADC ADC ADC ADC ADC ADC	A,(HL) A,(IX+d) A,(IY+d) A,A A,B A,C A,D A,E A,H A,L A,n	Add with Carry Oper and to Acc
ED4A ED5A ED6A ED7A	ADC ADC ADC ADC	HL,BC HL,DE HL,HL HL,SP	Add with Carry Reg Pair to HL
86 DD8605 FD8605 87 80 81 82 83 84 85 C620	ADD ADD ADD ADD ADD ADD ADD ADD ADD ADD	A,(HL) A,(IX+d) A,(IY+d) A,A A,B A,C A,D A,E A,H A,L A,n	Add Operand to Acc
09 19 29 39	ADD ADD ADD ADD	HL,BC HL,DE HL,HL HL,SP	Add Reg. Pair to HL
DD09 DD19 DD29 DD39	ADD ADD ADD ADD	IX,BC IX,DE IX,IX IX,SP	Add Reg. Pair to IX
FD09 FD19 FD29 FD39	ADD ADD ADD ADD	IY,BC IY,DE IY,IY IY,SP	Add Reg. Pair to IY
A6 DDA605 FDA605 A7 A0 A1 A2 A3 A4 A5 E620	AND AND AND AND AND AND AND AND AND AND	(HL) (IX+d) (IY+d) A B C D E H L n	Logical 'AND' of Operand and Acc
CB46 DDCB0546 FDCB0546 CB47 CB40 CB41 CB42 CB43 CB44	BIT BIT BIT BIT BIT BIT BIT BIT	0,(HL) 0,(IX+d) 0,(IY+d) 0,A 0,B 0,C 0,D 0,E 0,H	Test Bit b of Location or Reg

OBJ CODE	SOURCE STATEMENT	OPERATION
CB45	BIT 0,L	Test Bit b of Location or Reg.
CB4E	BIT 1,(HL)	
DDCB054E	BIT 1,(IX+d)	
FDCB054E	BIT 1,(IY+d)	
CB4F	BIT 1,A	
CB48	BIT 1,B	
CB49	BIT 1,C	
CB4A	BIT 1,D	
CB4B	BIT 1,E	
CB4C	BIT 1,H	
CB4D	BIT 1,L	
CB56	BIT 2,(HL)	
DDCB0556	BIT 2,(IX+d)	
FDCB0556	BIT 2,(IY+d)	
CB57	BIT 2,A	
CB50	BIT 2,B	
CB51	BIT 2,C	
CB52	BIT 2,D	
CB53	BIT 2,E	
CB54	BIT 2,H	
CB55	BIT 2,L	
CB5E	BIT 3,(HL)	
DDCB055E	BIT 3,(IX+d)	
FDCB055E	BIT 3,(IY+d)	
CB5F	BIT 3,A	
CB58	BIT 3,B	
CB59	BIT 3,C	
CB5A	BIT 3,D	
CB5B	BIT 3,E	
CB5C	BIT 3,H	
CB5D	BIT 3,L	
CB66	BIT 4,(HL)	
DDCB0566	BIT 4,(IX+d)	
FDCB0566	BIT 4,(IY+d)	
CB67	BIT 4,A	
CB60	BIT 4,B	
CB61	BIT 4,C	
CB62	BIT 4,D	
CB63	BIT 4,E	
CB64	BIT 4,H	
CB65	BIT 4,L	
CB6E	BIT 5,(HL)	
DDCB056E	BIT 5,(IX+d)	
FDCB056E	BIT 5,(IY+d)	
CB6F	BIT 5,A	
CB68	BIT 5,B	
CB69	BIT 5,C	
CB6A	BIT 5,D	
CB6B	BIT 5,E	
CB6C	BIT 5,H	
CB6D	BIT 5,L	
CB76	BIT 6,(HL)	
DDCB0576	BIT 6,(IX+d)	
FDCB0576	BIT 6,(IY+d)	
CB77	BIT 6,A	
CB70	BIT 6,B	
CB71	BIT 6,C	
CB72	BIT 6,D	
CB73	BIT 6,E	
CB74	BIT 6,H	
CB75	BIT 6,L	
CB7E	BIT 7,(HL)	
DDCB057E	BIT 7,(IX+d)	

OBJ CODE	SOURCE STATEMENT	OPERATION
FDC8057E	BIT 7,(IY+d)	Test Bit b Location or Reg.
CB7F	BIT 7,A	
CB78	BIT 7,B	
CB79	BIT 7,C	
CB7A	BIT 7,D	
CB7B	BIT 7,E	
CB7C	BIT 7,H	
CB7D	BIT 7,L	
DC8405	CALL C,nn	Call Subroutine at Location nn if Condition True
FC8405	CALL M,nn	
D48405	CALL NC,nn	
C48405	CALL NZ,nn	
F48405	CALL P,nn	
EC8405	CALL PE,nn	
E48405	CALL PO,nn	
CC8405	CALL Z,nn	
CD8405	CALL nn	Unconditional Call to Subroutine at nn
3F	CCF	Complement Carry Flag
BE	CP (HL)	Compare Operand with Acc
DDBE05	CP (IX+d)	
FD8E05	CP (IY+d)	
BF	CP A	
B8	CP B	
B9	CP C	
BA	CP D	
BB	CP E	
BC	CP H	
BD	CP L	
FE20	CP n	
EDA9	CPD	Compare Location (HL) and Acc Decrement HL and BC
EDB9	CPDR	Compare Location (HL) and Acc Decrement HL and BC, Repeat until BC = 0
EDA1	CPI	Compare Location (HL) and Acc, Increment HL and Decrement BC
EDB1	CPIR	Compare Location (HL) and Acc, Increment HL, Decrement BC, Repeat until BC = 0
2F	CPL	Complement Acc (1's Comp)
27	DAA	Decimal Adjust Acc.
35	DEC (HL)	Decrement Operand
DD3505	DEC (IX+d)	
FD3505	DEC (IY+d)	
3D	DEC A	
05	DEC B	
08	DEC BC	
0D	DEC C	
15	DEC D	
18	DEC DE	

OBJ CODE	SOURCE STATEMENT		OPERATION
1D	DEC	E	Decrement Operand
25	DEC	H	
2B	DEC	HL	
DD2B	DEC	IX	
FD2B	DEC	IY	
2D	DEC	L	
3B	DEC	SP	
F3	DI		Disable Interrupts
102E	DJNZ	e	Decrement B and Jump Relative if B \neq 0
FB	EI		Enable Interrupts
E3	EX	(SP),HL	Exchange Location and (SP)
DDE3	EX	(SP),IX	
FDE3	EX	(SP),IY	
08	EX	AF,AF'	Exchange the Con- tents of AF and AF'
EB	EX	DE,HL	Exchange the Con- tents of DE and HL
D9	EXX		Exchange the Con- tents of BC,DE,HL with Contents of BC',DE',HL' Respec- tively
76	HALT		HALT (Wait for Inter- rupt or Reset)
ED46	IM	0	Set Interrupt Mode
ED56	IM	1	
ED5E	IM	2	
ED78	IN	A,(C)	Load Reg. with Input from Device (C)
ED40	IN	B,(C)	
ED48	IN	C,(C)	
ED50	IN	D,(C)	
ED58	IN	E,(C)	
ED60	IN	H,(C)	
ED68	IN	L,(C)	
34	INC	(HL)	Increment Operand
DD3405	INC	(IX+d)	
FD3405	INC	(IY+d)	
3C	INC	A	
04	INC	B	
03	INC	BC	
0C	INC	C	
14	INC	D	
13	INC	DE	
1C	INC	E	
24	INC	H	
23	INC	HL	
DD23	INC	IX	
FD23	INC	IY	
2C	INC	L	
33	INC	SP	
DB20	IN	A,(n)	Load Acc. with Input from Device n
EDAA	IND		Load Location (HL) with Input from Port (C), Decrement HL and B

OBJ CODE	SOURCE STATEMENT	OPRRTION
EDBA	INDR	Load Location (HL) with Input from Port (C), Decrement HL and Decrement B, Repeat until B = 0
EDA2	INI	Load Location (HL) with Input from Port (C), Increment HL and Decrement B
EDB2	INIR	Load Location (HL) with Input from Port (C), Increment HL and Decrement B, Repeat until B = 0
E9	JP (HL)	Unconditional Jump to Location
DDE9	JP (IX)	
C38405	JP nn	
FDE9	JP (IY)	
DA8405	JP C,nn	Jump to Location if Condition True
FA8405	JP M,nn	
D28405	JP NC,nn	
C28405	JP NZ,nn	
F28405	JP P,nn	
EAB405	JP PE,nn	
E28405	JP PO,nn	
CAB405	JP Z,nn	
382E	JR C,e	Jump Relative to PC+e if Condition True
302E	JR NC,e	
202E	JR NZ,e	
282E	JR Z,e	
182E	JR e	Unconditional Jump Relative to PC+e
02	LD (8C),A	Load Source to Destination
12	LD (DE),A	
77	LD (HL),A	
70	LD (HL),B	
71	LD (HL),C	
72	LD (HL),D	
73	LD (HL),E	
74	LD (HL),H	
75	LD (HL),L	
3620	LD (HL),n	
DD7705	LD (IX+d),A	
DD7005	LD (IX+d),B	
DD7105	LD (IX+d),C	
DD7205	LD (IX+d),D	
DD7305	LD (IX+d),E	
DD7405	LD (IX+d),H	
DD7505	LD (IX+d),L	
DD360520	LD (IX+d),n	
FD7705	LD (IY+d),A	
FD7005	LD (IY+d),B	
FD7105	LD (IY+d),C	
FD7205	LD (IY+d),D	
FD7305	LD (IY+d),E	
FD7405	LD (IY+d),H	
FD7505	LD (IY+d),L	
FD360520	LD (IY+d),n	
328405	LD (nn),A	
ED438405	LD (nn),BC	

OBJ CODE	SOURCE STATEMENT	OPERATION
ED538405	LD (nn),DE	Load Source to Des-
228405	LD (nn),HL	tination
DD228405	LD (nn),IX	
FD228405	LD (nn),IY	
ED738405	LD (nn),SP	
0A	LD A,(BC)	
1A	LD A,(DE)	
7E	LD A,(HL)	
DD7E05	LD A,(IX+d)	
FD7E05	LD A,(IY+d)	
3A8405	LD A,(nn)	
7F	LD A,A	
78	LD A,B	
79	LD A,C	
7A	LD A,D	
7B	LD A,E	
7C	LD A,H	
ED57	LD A,I	
7D	LD A,L	
3E20	LD A,n	
ED5F	LD A,R	
46	LD B,(HL)	
DD4605	LD B,(IX+d)	
FD4605	LD B,(IY+d)	
47	LD B,A	
40	LD B,B	
41	LD B,C	
42	LD B,D	
43	LD B,E	
44	LD B,H	
45	LD B,L	
0620	LD B,n	
ED4B8405	LD BC,(nn)	
018405	LD BC,nn	
4E	LD C,(HL)	
DD4E05	LD C,(IX+d)	
FD4E05	LD C,(IY+d)	
4F	LD C,A	
48	LD C,B	
49	LD C,C	
4A	LD C,D	
4B	LD C,E	
4C	LD C,H	
4D	LD C,L	
0E20	LD C,n	
56	LD D,(HL)	
DD5605	LD D,(IX+d)	
FD5605	LD D,(IY+d)	
57	LD D,A	
50	LD D,B	
51	LD D,C	
52	LD D,D	
53	LD D,E	
54	LD D,H	
55	LD D,L	
1620	LD D,n	
ED5B8405	LD DE,(nn)	
118405	LD DE,nn	
5E	LD E,(HL)	
DD5E05	LD E,(IX+d)	
FD5E05	LD E,(IY+d)	
5F	LD E,A	
58	LD E,B	
59	LD E,C	

OBJ CODE	SOURCE STATEMENT	OPERATION
5A	LD E,D	Load Source to Destination
5B	LD E,E	
5C	LD E,H	
5D	LD E,L	
1E20	LD E,n	
66	LD H,(HL)	
DD6605	LD H,(IX+d)	
FD6605	LD H,(IY+d)	
67	LD H,A	
60	LD H,B	
61	LD H,C	
62	LD H,D	
63	LD H,E	
64	LD H,H	
65	LD H,L	
2620	LD H,n	
2A8405	LD HL,(nn)	
218405	LD HL,nn	
ED47	LD I,A	
DD2A8405	LD IX,(nn)	
DD218405	LD IX,nn	
FD2A8405	LD IY,(nn)	
FD218405	LD IY,nn	
6E	LD L,(HL)	
DD6E05	LD L,(IX+d)	
FD6E05	LD L,(IY+d)	
6F	LD L,A	
68	LD L,B	
69	LD L,C	
6A	LD L,D	
6B	LD L,E	
6C	LD L,H	
6D	LD L,L	
2E20	LD L,n	
ED4F	LD R,A	
ED7B8405	LD SP,(nn)	
F9	LD SP,HL	
DDF9	LD SP,IX	
DDF9	LD SP,IY	
318405	LD SP,nn	
EDA8	LDD	Load Location (DE) with Location (HL), Decrement DE,HL and BC
EDB8	LDDR	Load Location (DE) with Location (HL), Repeat until BC = 0
EDA0	LDI	Load Location (DE) with Location (HL), Increment DE,HL, Decrement BC
EDB0	LDIR	Load Location (DE) with Location (HL), Increment DE,HL, Decrement BC and Repeat until BC = 0
ED44	NEG	Negate Acc (2's Complement)
00	NOP	No Operation
B6	OR (HL)	Logical "OR" of Operand and Acc.
DDBB605	OR (IX+d)	

OBJ CODE	SOURCE STATEMENT		OPERATION
FDB605	OR	(IY+d)	Logical "OR" of Operand and Acc.
B7	OR	A	
B0	OR	B	
B1	OR	C	
B2	OR	D	
B3	OR	E	
B4	OR	H	
B5	OR	L	
F620	OR	n	
ED8B	OTDR		Load Output Port (C) with Location (HL) Decrement HL and B, Repeat until B = 0
EDB3	OTIR		Load Output Port (C) with Location (HL), Increment HL, Decre- ment B, Repeat until B = 0
ED79	OUT	(C),A	Load Output Port (C)
ED41	OUT	(C),B	with Reg.
ED49	OUT	(C),C	
ED51	OUT	(C),D	
ED59	OUT	(C),E	
ED61	OUT	(C),H	
ED69	OUT	(C),L	
D320	OUT	(n),A	Load Output Port (n) with Acc.
EDAB	OUTD		Load Output Port (C) with Location (HL), Decrement HL and B
EDA3	OUTI		Load Output Port (C) with Location (HL), Increment HL and Decrement B
F1	POP	AF	Load Destination
C1	POP	BC	with Top of Stack
D1	POP	DE	
E1	POP	HL	
DDE1	POP	IX	
FDE1	POP	IY	
F5	PUSH	AF	Load Source to Stack
C5	PUSH	BC	
D5	PUSH	DE	
E5	PUSH	HL	
DDE5	PUSH	IX	
FDE5	PUSH	IY	
CB86	RES	0,(HL)	Reset Bit b of Operand
DDCB0586	RES	0,(IX+d)	
FDCB0586	RES	0,(IY+d)	
CB87	RES	0,A	
CB80	RES	0,B	
CB81	RES	0,C	
CB82	RES	0,D	
CB83	RES	0,E	
C684	RES	0,H	
CB85	RES	0,L	
CB8E	RES	1,(HL)	
DDCB058E	RES	1,(IX+d)	
FDCB058E	RES	1,(IY+d)	
CB8F	RES	1,A	

OBJ CODE	SOURCE STATEMENT	OPERATION
CB88	RES 1,B	Reset Bit b of Operand
CB89	RES 1,C	
CB8A	RES 1,D	
CB8B	RES 1,E	
CB8C	RES 1,H	
CB8D	RES 1,L	
CB96	RES 2,(HL)	
DDCB0596	RES 2,(IX+d)	
FDCB0596	RES 2,(IY+d)	
CB97	RES 2,A	
CB90	RES 2,B	
CB91	RES 2,C	
CB92	RES 2,D	
CB93	RES 2,E	
CB94	RES 2,H	
CB95	RES 2,L	
CB9E	RES 3,(HL)	
DDCB059E	RES 3,(IX+d)	
FDCB059E	RES 3,(IY+d)	
CB9F	RES 3,A	
CB98	RES 3,B	
CB99	RES 3,C	
CB9A	RES 3,D	
CB9B	RES 3,E	
CB9C	RES 3,H	
CB9D	RES 3,L	
CBA6	RES 4,(HL)	
DDCB05A6	RES 4,(IX+d)	
FDCB05A6	RES 4,(IY+d)	
CBA7	RES 4,A	
CBA0	RES 4,B	
CBA1	RES 4,C	
CBA2	RES 4,D	
DBA3	RES 4,E	
CBA4	RES 4,H	
CBA5	RES 4,L	
CBAE	RES 5,(HL)	
DDCB05AE	RES 5,(IX+d)	
FDCB05AE	RES 5,(IY+d)	
CBAF	RES 5,A	
CBA8	RES 5,B	
CBA9	RES 5,C	
CBAA	RES 5,D	
CBAB	RES 5,E	
CBAC	RES 5,H	
CBAD	RES 5,L	
CBB6	RES 6,(HL)	
DDCB05B6	RES 6,(IX+d)	
FDCB05B6	RES 6,(IY+d)	
CBB7	RES 6,A	
CBB0	RES 6,B	
CBB1	RES 6,C	
CBB2	RES 6,D	
CBB3	RES 6,E	
CBB4	RES 6,H	
CBB5	RES 6,L	
CBBE	RES 7,(HL)	
DDCB05BE	RES 7,(IX+d)	
FDCB05BE	RES 7,(IY+d)	
CBBF	RES 7,A	
CBB8	RES 7,B	
CBB9	RES 7,C	
CBBA	RES 7,D	

OBJ CODE	SOURCE STATEMENT	OPERATION
CB8B CB8C CB8D	RES 7,E RES 7,H RES 7,L	Reset Bit b of Operand
C9	RET	Return from Subroutine
D8 F8 D0 C0 F0 E8 E0 C8	RET C RET M RET NC RET NZ RET P RET PE RET P0 RET Z	Return from Subroutine if Condi- tion True
ED4D	RETI	Return from Interrupt
ED45	RETN	Return from Non- Maskable Interrupt
CB16 DDCB0516 FDCB0516 CB17 CB10 CB11 CB12 CB13 CB14 CB15	RL (HL) RL (IX+d) RL (IY+d) RL A RL B RL C RL D RL E RL H RL L	Rotate Left Through Carry
17	RLA	Rotate Left Acc. Through Carry
CB06 DDCB0506 FDCB0506 CB07 CB00 CB01 CB02 CB03 CB04 CB05	RLC (HL) RLC (IX+d) RLC (IY+d) RLC A RLC B RLC C RLC D RLC E RLC H RLC L	Rotate Left Circular
07	RLCA	Rotate Left Circular Acc.
ED6F	RLD	Rotate Digit Left and Right between Acc. and and Location (HL)
CB1E DDCB051E FDCB051E CB1F CB18 CB19 CB1A CB1B CB1C CB1D	RR (HL) RR (IX+d) RR (IY+d) RR A RR B RR C RR D RR E RR H RR L	Rotate Right Through Carry
1F	RRA	Rotate Right Acc Through Carry
CB0E DDCB050E FDCB050E CB0F	RRC (HL) RRC (IX+d) RRC (IY+d) RRC A	Rotate Right Circular

OBJ CODE	SOURCE STATEMENT	OPERATION	
CB08	RRC B	Rotate Right Circular	
CB09	RRC C		
CB0A	RRC D		
CB0B	RRC E		
CB0C	RRC H		
CB0D	RRC L		
OF	RRCA	Rotate Right Circular Acc.	
ED67	RRD	Rotate Digit: Right and Left Between Acc. and Location (HL)	
C7	RST 00H	Restart to Location	
CF	RST 03H		
D7	RST 10H		
DF	RST 18H		
E7	RST 20H		
EF	RST 28H		
F7	RST 30H		
FF	RST 38H		
9E	SBC A,(HL)		Subtract Operand from Acc. with Carry
DD9E05	SBC A,(IX+d)		
FD9E05	SBC A,(IY+d)		
9F	SBC A,A		
98	SBC A,B		
99	SBC A,C		
9A	SBC A,D		
9B	SBC A,E		
9C	SBC A,H		
9D	SBC A,L		
DE20	SBC A,n		
ED42	SBC HL,BC		
ED52	SBC HL,DE		
ED62	SBC HL,HL		
ED72	SBC HL,SP		
37	SCF	Set Carry Flag (C - 1)	
CBC6	SET 0,(HL)	Set Bit b of Location	
DDCB05C6	SET 0,(IX+d)		
FDCB05C6	SET 0,(IY+d)		
CBC7	SET 0,A		
CBC0	SET 0,B		
CBC1	SET 0,C		
CBC2	SET 0,D		
CBC3	SET 0,E		
CBC4	SET 0,H		
CBC5	SET 0,L		
CBCE	SET 1,(HL)		
DDCB05CE	SET 1,(IX+d)		
FDCB05CE	SET 1,(IY+d)		
CBCF	SET 1,A		
CBC8	SET 1,B		
CBC9	SET 1,C		
CBCA	SET 1,D		
CBCB	SET 1,E		
CBCC	SET 1,H		
CBCD	SET 1,L		
CBD6	SET 2,(HL)		
DDCB05D6	SET 2,(IX+d)		
FDCB05D6	SET 2,(IY+d)		
CBD7	SET 2,A		
CBD0	SET 2,B		
CBD1	SET 2,C		
CBD2	SET 2,D		

OBJ CODE	SOURCE STATEMENT	OPERATION
CB03	SET 2,E	Set Bit b of Location
CB04	SET 2,H	
CB05	SET 2,L	
CB08	SET 3,B	
CBDE	SET 3,(HL)	
DDCB05DE	SET 3,(IX+d)	
FDCB05DE	SET 3,(IY+d)	
CBDF	SET 3,A	
CB09	SET 3,C	
CBDA	SET 3,D	
CBDB	SET 3,E	
CBDC	SET 3,H	
CBDD	SET 3,L	
CB6E	SET 4,(HL)	
DDCB05E6	SET 4,(IX+d)	
FDCB05E6	SET 4,(IY+d)	
CBE7	SET 4,A	
CBE0	SET 4,B	
CBE1	SET 4,C	
CBE2	SET 4,D	
CBE3	SET 4,E	
CBE4	SET 4,H	
CBE5	SET 4,L	
CBEE	SET 5,(HL)	
DDCB05EE	SET 5,(IX+d)	
FDCB05EE	SET 5,(IY+d)	
CBEF	SET 5,A	
CBE8	SET 5,B	
CBE9	SET 5,C	
CBEA	SET 5,D	
CBEB	SET 5,E	
CBEC	SET 5,H	
CBED	SET 5,L	
CBF6	SET 6,(HL)	
DDCB05F6	SET 6,(IX+d)	
FDCB05F6	SET 6,(IY+d)	
CBF7	SET 6,A	
CBF0	SET 6,B	
CBF1	SET 6,C	
CBF2	SET 6,D	
CBF3	SET 6,E	
CBF4	SET 6,H	
CBF5	SET 6,L	
CBFE	SET 7,(HL)	
DDCB05FE	SET 7,(IX+d)	
FDCB05FE	SET 7,(IY+d)	
CBFF	SET 7,A	
CBF8	SET 7,B	
CBF9	SET 7,C	
CBFA	SET 7,D	
CBFB	SET 7,E	
CBFC	SET 7,H	
CBFD	SET 7,L	
CB26	SLA (HL)	Shift Operand Left
DDCB0526	SLA (IX+d)	Arithmetic
FDCB0526	SLA (IY+d)	
CB27	SLA A	
CB20	SLA B	
CB21	SLA C	
CB22	SLA D	
CB23	SLA E	
CB24	SLA H	
CB25	SLA L	

OBJ CODE	SOURCE STATEMENT	OPERATION
CB2E	SRA (HL)	Shift Operand Right
DDCB052E	SRA (IX+d)	Arithmetic
FDCB052E	SRA (IY+d)	
CB2F	SRA A	
CB28	SRA B	
CB29	SRA C	
CB2A	SRA D	
CB2B	SRA E	
CB2C	SRA H	
CB2D	SRA L	
CB3E	SRL (HL)	Shift Operand Right
DDCB053E	SRL (IX+d)	Logical
FDCB053E	SRL (IY+d)	
CB3F	SRL A	
CB38	SRL B	
CB39	SRL C	
CB3A	SRL D	
CB3B	SRL E	
CB3C	SRL H	
CB3D	SRL L	
96	SUB (HL)	Subtract Operand
DD9605	SUB (IX+d)	from Acc.
FD9605	SUB (IY+d)	
97	SUB A	
90	SUB B	
91	SUB C	
92	SUB D	
93	SUB E	
94	SUB H	
95	SUB L	
D620	SUB n	
AE	XOR (HL)	Exclusive "OR"
DDAE05	XOR (IX+d)	Operand and Acc.
FDAE05	XOR (IY+d)	
AF	XOR A	
A8	XOR B	
A9	XOR C	
AA	XOR D	
AB	XOR E	
AC	XOR H	
AD	XOR L	
EE20	XOR n	

Example Values

nn EQU 584H
d EQU 5
n EQU 20H
e 30H

HEX TO DECIMAL CONVERSION

The following is a conversion from Hex to Decimal for all one byte values

Dec	Hex																
0	00	32	20	64	40	96	60	128	80	160	A0	192	C0	224	E0		
1	01	33	21	65	41	97	61	129	81	161	A1	193	C1	225	E1		
2	02	34	22	66	42	98	62	130	82	162	A2	194	C2	226	E2		
3	03	35	23	67	43	99	63	131	83	163	A3	195	C3	227	E3		
4	04	36	24	68	44	100	64	132	84	164	A4	196	C4	228	E4		
5	05	37	25	69	45	101	65	133	85	165	A5	197	C5	229	E5		
6	06	38	26	70	46	102	66	134	86	166	A6	198	C6	230	E6		
7	07	39	27	71	47	103	67	135	87	167	A7	199	C7	231	E7		
8	08	40	28	72	48	104	68	136	88	168	A8	200	C8	232	E8		
9	09	41	29	73	49	105	69	137	89	169	A9	201	C9	233	E9		
10	0A	42	2A	74	4A	106	6A	138	8A	170	AA	202	CA	234	EA		
11	0B	43	2B	75	4B	107	6B	139	8B	171	AB	203	CB	235	EB		
12	0C	44	2C	76	4C	108	6C	140	8C	172	AC	204	CC	236	EC		
13	0D	45	2D	77	4D	109	6D	141	8D	173	AD	205	CD	237	ED		
14	0E	46	2E	78	4E	110	6E	142	8E	174	AE	206	CE	238	EE		
15	0F	47	2F	79	4F	111	6F	143	8F	175	AF	207	CF	239	EF		
16	10	48	30	80	50	112	70	144	90	176	B0	208	D0	240	F0		
17	11	49	31	81	51	113	71	145	91	177	B1	209	D1	241	F1		
18	12	50	32	82	52	114	72	146	92	178	B2	210	D2	242	F2		
19	13	51	33	83	53	115	73	147	93	179	B3	211	D3	243	F3		
20	14	52	34	84	54	116	74	148	94	180	B4	212	D4	244	F4		
21	15	53	35	85	55	117	75	149	95	181	B5	213	D5	245	F5		
22	16	54	36	86	56	118	76	150	96	182	B6	214	D6	246	F6		
23	17	55	37	87	57	119	77	151	97	183	B7	215	D7	247	F7		
24	18	56	38	88	58	120	78	152	98	184	B8	216	D8	248	F8		
25	19	57	39	89	59	121	79	153	99	185	B9	217	D9	249	F9		
26	1A	58	3A	90	5A	122	7A	154	9A	186	BA	218	DA	250	FA		
27	1B	59	3B	91	5B	123	7B	155	9B	187	BB	219	DB	251	FB		
28	1C	60	3C	92	5C	124	7C	156	9C	188	BC	220	DC	252	FC		
29	1D	61	3D	93	5D	125	7D	157	9D	189	BD	221	DD	253	FD		
30	1E	62	3E	94	5E	126	7E	158	9E	190	BE	222	DE	254	FE		
31	1F	63	3F	95	5F	127	7F	159	9F	191	BF	223	DF	255	FF		

6		5		4		3		2		1	
HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC		HEX = DEC	
0	0	0	0	0	0	0	0	0	0	0	0
1	1,048,576	1	65,536	1	4,096	1	256	1	16	1	1
2	2,097,152	2	131,072	2	8,192	2	512	2	32	2	2
3	3,145,728	3	196,608	3	12,288	3	768	3	48	3	3
4	4,194,304	4	262,144	4	16,384	4	1,024	4	64	4	4
5	5,242,880	5	327,680	5	20,480	5	1,280	5	80	5	5
6	6,291,456	6	393,216	6	24,576	6	1,536	6	96	6	6
7	7,340,032	7	458,752	7	28,672	7	1,792	7	112	7	7
8	8,388,608	8	524,288	8	32,768	8	2,048	8	128	8	8
9	9,437,184	9	589,824	9	36,864	9	2,304	9	144	9	9
A	10,485,760	A	655,360	A	40,960	A	2,560	A	160	A	10
B	11,534,336	B	720,896	B	45,056	B	2,816	B	176	B	11
C	12,582,912	C	786,432	C	49,152	C	3,072	C	192	C	12
D	13,631,488	D	851,968	D	53,248	D	3,328	D	208	D	13
E	14,680,064	E	917,504	E	57,344	E	3,584	E	224	E	14
F	15,728,640	F	983,040	F	61,440	F	3,840	F	240	F	15
0123		4567		0123		4567		0123		4567	
		BYTE				BYTE				BYTE	

APPENDIX II BASE CONVERSION

The following is a conversion from Hex to Decimal for all one byte values

Dec	Hex														
0	00	32	20	64	40	96	60	128	80	160	A0	192	C0	224	E0
1	01	33	21	65	41	97	61	129	81	161	A1	193	C1	225	E1
2	02	34	22	66	42	98	62	130	82	162	A2	194	C2	226	E2
3	03	35	23	67	43	99	63	131	83	163	A3	195	C3	227	E3
4	04	36	24	68	44	100	64	132	84	164	A4	196	C4	228	E4
5	05	37	25	69	45	101	65	133	85	165	A5	197	C5	229	E5
6	06	38	26	70	46	102	66	134	86	166	A6	198	C6	230	E6
7	07	39	27	71	47	103	67	135	87	167	A7	199	C7	231	E7
8	08	40	28	72	48	104	68	136	88	168	A8	200	C8	232	E8
9	09	41	29	73	49	105	69	137	89	169	A9	201	C9	233	E9
10	0A	42	2A	74	4A	106	6A	138	8A	170	AA	202	CA	234	EA
11	0B	43	2B	75	4B	107	6B	139	8B	171	AB	203	CB	235	EB
12	0C	44	2C	76	4C	108	6C	140	8C	172	AC	204	CC	236	EC
13	0D	45	2D	77	4D	109	6D	141	8D	173	AD	205	CD	237	ED
14	0E	46	2E	78	4E	110	6E	142	8E	174	AE	206	CE	238	EE
15	0F	47	2F	79	4F	111	6F	143	8F	175	AF	207	CF	239	EF
16	10	48	30	80	50	112	70	144	90	176	B0	208	D0	240	F0
17	11	49	31	81	51	113	71	145	91	177	B1	209	D1	241	F1
18	12	50	32	82	52	114	72	146	92	178	B2	210	D2	242	F2
19	13	51	33	83	53	115	73	147	93	179	B3	211	D3	243	F3
20	14	52	34	84	54	116	74	148	94	180	B4	212	D4	244	F4
21	15	53	35	85	55	117	75	149	95	181	B5	213	D5	245	F5
22	16	54	36	86	56	118	76	150	96	182	B6	214	D6	246	F6
23	17	55	37	87	57	119	77	151	97	183	B7	215	D7	247	F7
24	18	56	38	88	58	120	78	152	98	184	B8	216	D8	248	F8
25	19	57	39	89	59	121	79	153	99	185	B9	217	D9	249	F9
26	1A	58	3A	90	5A	122	7A	154	9A	186	BA	218	DA	250	FA
27	1B	59	3B	91	5B	123	7B	155	9B	187	BB	219	DB	251	FB
28	1C	60	3C	92	5C	124	7C	156	9C	188	BC	220	DC	252	FC
29	1D	61	3D	93	5D	125	7D	157	9D	189	BD	221	DD	253	FD
30	1E	62	3E	94	5E	126	7E	158	9E	190	BE	222	DE	254	FE
31	1F	63	3F	95	5F	127	7F	159	9F	191	BF	223	DF	255	FF

COMMUNICATIONS

PART 5

HUSKY COMMUNICATIONS

- 5.1 INTRODUCTION
- 5.2 APPLICATIONS
- 5.3 COMMUNICATION PORT SOFTWARE
- 5.4 ASYNCHRONOUS CHARACTER HANDLING
- 5.5 ASYNCHRONOUS PROTOCOLS
- 5.6 SYNCHRONOUS PROTOCOLS
- 5.7 HARDWARE CHARACTERISTICS
- 5.8 TERMINAL EMULATION
- 5.9 ERROR MESSAGES

5

INTRODUCTION

5.1

Introduction

A principle feature of **HUSKY's** architecture is its powerful and flexible communications facility.

HUSKY can communicate freely with most other types of computer, from micros to mainframes, using a variety of easily selected communication 'Protocols'. These protocols, of both synchronous and asynchronous varieties, are chosen for compatibility with other systems and are not special to **HUSKY**.

To establish communications, **HUSKY** uses a subset of the universally-accepted RS232/V24 communications interface. This interface, used in virtually every computer system, terminal, printer and modem, allows **HUSKY** to 'plug in' to other systems without modification.

HUSKY is equipped to act as a 'Data Terminal Equipment' (DTE), and together with appropriate protocol selections can resemble (partially emulate) many popular computer terminals. In this mode, it can be used as a portable computer terminal, accessing and communicating with other systems.

Alternatively, **HUSKY** can act as a self-contained computer, directly supporting peripherals like printers without additional equipment. Remember that in this mode, **HUSKY** may need a 'crossed cable' or 'modem simulator' to communicate with other terminal-equipped devices.

HUSKIES can communicate with each other, either side-by-side on the bench or over thousands of miles by telecommunication channels.

APPLICATIONS

HUSKY's communications can be used for transferring both programs and data to other systems.

Programs can be loaded into **HUSKY** daily, or as required, from a central database. Secure protocols and program-based error checks can guarantee accurate copies.

New, revised or corrected programs can be returned to the database for storage. Database files and command functions can be accessed automatically, or under manual control using **HUSKY's** terminal emulation.

Data can be collected and down-loaded to mainframe, minicomputer or microprocessor storage whenever required. Data transmissions can be formatted exactly to resemble existing data terminal configurations, allowing **HUSKY** to slot directly into existing software support structures.

Database information (for instance, tables of names and addresses) can be sent to **HUSKY** for presentation to operatives in the field. Collected data can be linked to previously loaded database information for instant verification and validation.

COMMUNICATION PORT SOFTWARE

5.3

HUSKY's communications format is **software controlled**.

If **HUSKY** fails to communicate, always check the port initialisation for the correct format.

5.3.1 **Initialising the Port**

To initialise, or alter, **HUSKY's** communications procedure, follow these steps:

- 5.3.2 From **Main Menu** or application program (see Section 4.2.3.18) select 'Initialise communications'.

You are now in the communications program. All entries are menu-type choices, selected by the 'scroll' keys and confirmed by 'enter'.

Values previously selected are displayed as the initial menu option presented. For instance, if '1200 baud' had previously been selected, then this option will appear after the heading 'Rate'.

- 5.3.3 **HUSKY** will prompt first for 'transmission parameters', followed by 'Receiving Parameters'.

Specific parameters are modified by positioning the cursor with the ___ ___ keys, and selecting the desired option with the | | keys.

The transmission parameter screen looks like this:

FIG.5.1

Transmission Parameters				
Rate-1800	Prtcl-none	Pty-none		
CTS-n	DTR-n	LF-n	Echo-y	Null-0
press ENTER if acceptable				

The receiving parameter screen looks like this:

FIG 5.2

Receiving Parameters				
Rate-1200	Prtcl-none	Pty-none		
RTS-off	DSR-n	DCD-n	Serig-off	
press ENTER if acceptable				

To re-select the previous entries, just press 'ENTER'.

5.3.4

Baud Rate

Transmission and reception baud rates may be independently set. It should be remembered that if any software handshaking is used then both transmission and reception should be at the same speed, as most systems operate in this way. (If the remote system has different transmission and reception rates then HUSKY should be set up appropriately).

The available baud rates are:

50, 75, 110, 150, 300, 600, 1200

5.3.5

Select Protocol

Protocols available (not necessarily on every machine) include the following:

NONE

XON/OFF

ACK/NAK

ETX/ACK

5.3.6

Select Parity (PTY)

On transmission parity is implemented as follows:

- | | |
|---------|---|
| a) None | -bit 8 is reset i.e. logical 0 |
| b) Odd | -bit 8 is set or reset to cause an odd number of logical 1's in the data bits. |
| c) Even | -bit 8 is set or reset to cause an even number of logical 1's in the data bits. |

On reception the implementation is:

- | | |
|----------------|--|
| a) None | -parity bit ignored. |
| b) Odd or Even | -the data is checked for an odd or even number of logical 1's. If the check fails then the character is received as OFFH which is displayed as |

Whichever parity selection is made the data always has bit 7 = 0 to the calling program.

5.3.7 **Select Handshaking**

This function determines the function of the CTS and RTS signals in the RS-232 interface during communications.

- a) In Receive Mode: CTS active (y).
If selected, **HUSKY** will assert CTS when it is ready to receive data. (CTS=1 when ready for data, CTS=0 when not). If not selected (n), is always '1'.
- b) In Transmit Mode: RTS active.
If selected, **HUSKY** will only transmit when the RTS signal from the peripheral is active. (RTS=1 when OK to transmit, RTS=0 when not). If not selected, **HUSKY** will ignore the RTS line.

NOTE: See Section 5.7.2 'Interface connections'.

5.3.8 **Line Feed Enable (Transmit mode only)**

To make transmission to devices such as printers and VDU's it is convenient to have control of whether or not Line Feeds (10 Decimal or Control J) are sent following carriage return (CR) (13 Decimal or Control M).

If a CR (13 Decimal or Control M) is sent, and if the LF option is selected, an LF is sent out following the CR. If LF is not selected only the CR is sent.

NOTE: in firmware revisions issued before January 1983, LF characters are **always** suppressed if LF is not selected, even if sent in the form:

LOPCHR 10.

It is necessary to POKE 17815 (LEAF), 1=LF active, 0=not. See section 4.1.4.

This problem is eliminated in current revisions.

5.3.9 **Null Select (Transmit mode only)**

To help devices which do not support handshaking, e.g. some printers, the NULL option allows a delay at the end of lines by transmitting a number of NULLs. The NULLs are sent after the CR of CR,LF. **HUSKY** permits 0,2,5,10 or 20 NULLs to be sent at the end of lines.

5.3.10 **Transmission Echo**

By selecting this option any characters which are being transmitted will also be displayed on the **HUSKY** screen. This is a very useful method of checking that transmission is actually taking place. The function is equivalent to full/half duplex selection.

5.3.11 Select Serial Ignore Character (Receive Mode only)

Some devices which communicate with **HUSKY** may transmit characters which the **HUSKY** does not require. This option **SERIG** is selected to inform the **HUSKY** to ignore such characters. The decimal value of the character to be ignored is selected by using the **|** and **|** keys to increase or decrease the number displayed next to **SERIG** and then pressing Enter. See Part 3, Fig. 3.8.2 for ASCII-decimal equivalent.

If the **HUSKY** is required to ignore the character 'DEL' (delete) then 127 should be selected.

NOTE: It is important that 'off' should be selected in this option if not required.

ASYNCHRONOUS CHARACTER HANDLING

5.4

5.4.1

Introduction

HUSKY's asynchronous serial data communications are handled by a communications software package totally separate to the user's program or the Basic interpreter.

The user needs no detailed knowledge of data communication to be able to utilise **HUSKY's** powerful facilities: The required configuration is selected from 'menus' and operation of the package is virtually invisible.

5.4.2

Buffers

Characters are transferred to and from the serial interface via invisible 'buffers' that effectively disconnect the application program from the outside world. These buffers ensure a smooth flow of data and correct reaction to protocol responses even when **HUSKY** is busy doing something quite different!

Because the communication package is interrupt driven, the execution of user programs is not affected by communication. However, the user program **may slow down** if the serial line is busy. **HUSKY's** communication buffers are available whenever the machine is powered up, even if the application program is **not requesting communication**. Incoming messages will be received and held in the buffer, while any outgoing data remaining will continue to be sent down the line. Any protocols selected will continue to be observed.

5.4.3

Communicating with Basic

HUSKY's Basic interpreter transmits and receives characters via the serial interface either singly or in blocks.

Individual characters can be transmitted by statements like LOPCHR and received by LINCHR. Whole words, variables, strings or lines can be handled by LPRINT and LINPUT. See Part 3, '**HUSKY's** Basic Programming', for details.

Irrespective of the content of the user program, all data rates, protocols and other selections previously made will be observed invisibly by **HUSKY's** communication software package.

5.4.3.1

When **transmitting** characters, **HUSKY** Basic will pause while each character or block is sent. If the transmission echo is selected, each character will appear on the screen as it is transmitted. **HUSKY** Basic will not move forward to the next program line until the character or block is complete. If an error checking protocol is selected, **HUSKY** will wait until confirmation of reception ('ACK') is received. If a 'communication failure' is detected, execution of the user program will be suspended until manually overridden.

- 5.4.3.2 When **receiving** characters, LINCHR or LINPUT statements will fetch any characters or blocks currently held in the reception buffer.

NOTE: This data may have been received **prior** to execution of the LINCHR or LINPUT program line.

If data is already in the buffer, Basic will return immediately with the message. If no data is present, Basic will wait for incoming data, and will return when characters or blocks are available.

Reception of data is completely transparent to the user Basic program being run by the **HUSKY**. Data may be sent at any time to the **HUSKY**, communications protocols being implemented as selected.

Use is made of a 132 character input buffer. Received characters are placed into the buffer in a 'barrel' fashion. If more characters are received than this, without any being read by a calling program then the earliest characters are overwritten. This is true even if RTS or XON/XOFF handshaking is used and is ignored by the sending device.

The effect of this buffering is to permit reception of characters to carry on steadily through calling programs which process the characters intermittently. This can give a greater overall throughput.

The buffer is needed by definition for the XON/XOFF and ETX/ACK protocols.

ASYNCHRONOUS PROTOCOLS

5.5

NOTE: In this section, 'host' means the device communicating with **HUSKY**, whether it be computer, printer, modem or otherwise.

These protocols may be used by Basic application programs, user assembly code or directly in 'VDU simulation' mode.

5.5.1.1 **None**

RECEPTION

As implied, this selection does nothing to control the incoming data. It behaves as a simple teletype. Characters are placed into the reception buffer and a calling program may read them out asynchronously.

5.5.1.2 **None**

TRANSMISSION

Each character is transmitted as the calling program requests it (except when waiting for the previous character to be sent). There is no buffering of the output data.

5.5.2.1 **XON/XOFF (DC1/DC3)**

RECEPTION

If an incoming character causes the receive buffer to have 90 or more unread characters then an XOFF character (DC3, 19 Decimal or Control S) is transmitted, which should prevent the host sending further characters to **HUSKY**. When the buffer is reduced to 10 pending characters, an XON (17 Decimal or Control Q, DC1) is transmitted to re-enable the host to **HUSKY** transmission.

Subsequent XONs or XOFFs are never transmitted without an intervening XOFF or XON respectively.

5.5.2.2 **XON/XOFF (DC1/DC3)**

TRANSMISSION

The input data line is also examined for XON or XOFF characters. If an XOFF or stream of XOFFs are received then transmission is stopped as soon as the character being transmitted is finished. The protocol requires an XON to restart transmission. This protocol procedure strips 'XON' or 'XOFF' Control characters, thus passing only valid data to Basic.

5.5.2.3 SLAVE, HUSKY TO HOST TRANSMISSION

TRANSMISSION

It should be noted that following the host transmitting an XOFF, it is possible, depending on data link speed, to have a further two characters sent out by **HUSKY**. This is due to the time taken to receive the XOFF and the possibility of just missing the start of the next character.

At the start of transmission a received XON is assumed, to avoid lock-up.

5.5.3.1 ETX/ACK

RECEPTION

In this protocol the **HUSKY** will send an ACK character (06 Decimal or Control F) after reception of an ETX character terminating a line that has been ready by BASIC. The ETX is stripped from incoming data and not sent to Basic. This protocol causes the host device to wait at the end of each block of data, which should not exceed 130 characters, for an acknowledgement that the block has been read. The ETX/ACK acknowledgement is only transmitted when the block has been fetched from the buffer by BASIC.

5.5.3.2 ETX/ACK

TRANSMISSION

This protocol will send out an ETX (03 Decimal or Control C) character after any transmitted carriage return. Transmission is then halted until an ACK (06 Decimal or Control F) is received from the host. This enables individual data blocks to be sent and an acknowledgement awaited. If transmission is to another **HUSKY** the blocks should not exceed 130 characters or the input buffer may be exceeded. If an ACK is not received within three seconds then communication failure is assumed and an error message is put up on the screen.

The ACKs are not sent to BASIC.

5.5.4.1 ACK/NAK

RECEPTION

In this mode the **HUSKY** will receive a complete block of data, terminated with a carriage return. It will then check for parity errors. If one exists then a NAK (21 Decimal or Control U) is sent out to the host and the data block ignored.

If the block is error-free then the received block is sent character by character to BASIC. When the carriage return is sent to BASIC an ACK (06 Decimal or Control F) is sent out to the host to indicate successful data block reception.

If HUSKY is waiting to send a character and an XOFF has held up transmission, then if an XON is not received within 30 seconds, a communication failure is assumed and an error message is put on the screen.

5.5.4.2 ACK/NAK

TRANSMISSION

Data blocks are first placed into a transmission buffer. Upon receipt of a carriage return from BASIC, the entire buffer is transmitted. HUSKY then awaits an acknowledgement. If an ACK is received then the next data block is assembled into the buffer and sent. If a NAK (21 Decimal or Control U) is received then a failure is indicated and the entire buffer is sent again. If nothing is received after three seconds a NAK is assumed and the buffer sent again. The NAKs may be sent a maximum of three times. If an ACK is still not received then the communications failure message is put on the screen.

5.5.5 System**5.5.5.1 HUSKY to System communications Protocol Specification**

This protocol is provided to facilitate communication between **HUSKY** and System model S500 minicomputers. The protocol embodies a checksum in addition to parity checks.

Data is sent in the form of messages which are subdivided into one or more fixed length blocks. Each block is individually checked and acknowledged before any further blocks are sent.

DATA BLOCKS

Each block consists of:-

- 1 start character
- 64 data characters
- 3 checksum
- 1 stop character

The purpose of each character group is as follows:-

Start Character

The first block of a message uses SOH (01 Decimal). Further blocks use STX (02H) as the start character.

Stop Character

The last block of a message uses EOT (04 Decimal), previous blocks use ETX (03H) as the stop character.

Data Character

The data characters may consist of any 7 bit pattern with the exception of the handshake characters : ACK (06 Decimal), NAK (15H) and WACK (13H).

Checksum

The checksum is formed as the modulo 256 addition of all data characters, with parity reset. The number formed is transmitted in ASCII as three decimal digits. Leading zeros are not suppressed. The most significant digit is sent first.

Insufficient Data Characters

If there are less than 64 characters for the data block, then the remainder of the block is filled with NULLS (00 Decimal). The characters are ignored on reception.

Handshaking

Each block of data must be acknowledged by the receiving device. After a successful reception and if the receiver is ready to receive another block, then it returns an ACK (06 Decimal). If the transfer is deemed unsuccessful for any of the following reasons:-

- i) parity fault on any character
- ii) Incorrect start or end characters
- iii) wrong number of characters
- iv) checksum error
- v) timeout (waiting for characters for 4 secs)

then a failure is returned as a NAK (21 Decimal).

If the receiver accepts the block, but is not ready for further data, then a wait acknowledge WACK (19 Decimal) character delaying the transmitter's time out is transmitted. Further delays can be accumulated by transmitting WACK's at the rate of one every 2 secs. The message is finally accepted with an ACK.

5.5.6

Communications Failure

If a failure occurs on transmission and a timeout activates, then HUSKY will display 'Communications Failure' on the top line of the display. This message may be overridden by pressing 'X' on the keyboard. This will force transmission for ETX/ACK or XON/XOFF; in the case of ACK/NAK then the buffer will be sent another three times.

5.6

SYNCHRONOUS PROTOCOLS

To provide the fast high integrity communication required by most mainframe computers, Husky has a synchronous data communication capability.

Communication takes place using synchronous transmission of data blocks. These blocks are checked using check characters and acknowledged as appropriate using particular protocols.

Currently available as a factory option is the IBM2780* communication protocol carried over the bisync synchronous communication carrier. Other protocols are in development.

5.6.1

IBM 2780 ON THE HUSKY**BACKGROUND**

The IBM 2780 is a very popular data transmission terminal used for remote job entry. The transmission protocol developed for this device has been widely implemented on other mainframes to the extent of becoming a de facto industry standard. Husky's implementation mimics 2780 protocol to provide well proven, fast and extremely reliable data transfers.

IBM 2780 is used for batch entry by use of "job cards". The Husky is easily programmed to simulate the use of job cards providing all the processing facilities normally available on a remote terminal.

HUSKY is the first portable to offer a comprehensive implementation of IBM 2780 in true synchronous mode.

ADVANTAGES

This well proven and sophisticated protocol gives Husky reliable, high speed communications over relatively poor data links such as the public switched telephone network at speeds as high as 2400 baud.

Using a widely used standard provides access to a wide range of mainframe computer systems, directly and without need for supporting equipment. Data files can be exchanged between Husky and mainframe with great flexibility, convenience and dependability.

***NOTE: IBM and 2780 are registered trademarks of IBM Corp.** Husky is the first portable to offer a comprehensive implementation of IBM 2780 in true synchronous mode.

TECHNICAL IMPLEMENTATION

- :Synchronous transmission using BiSync
- :Character set - EBCDIC
- :Full CRC generation, error handling and flow control
- :Modem handshake lines available
- :Internal or external clocking options
- :May be used over the public switched telephone network or private lines
- :Records terminated using the EM character, providing optimum throughput
- :The "wait acknowledge" is fully implemented
- :Protocol use is completely transparent to user programs
- :Data Rates - up to 2400 Hz with external clocking
 - up to 1200 Hz using internal clocking

The internal self-synchronising feature enables Husky to Husky communications over standard 300 baud asynchronous modems.

5.6.2

Synchronous communication

Synchronous communication transfers data as a continuous stream of data characters, without the need for start and stop bits as used in asynchronous communication. This maximises the data transfer rate for a given channel bandwidth. For the receiver to be able to use the data two levels of synchronisation need to take place:

- i) **Bit synchronisation**
This allows the receiver to save the state of each bit correctly, preferably sampled at the centre of each bit.
- ii) **Character Synchronisation**
Each block of eight bits which represent a character needs to be identified from within the data stream.

5.6.2.1

Bit Synchronisation

Two techniques are used by HUSKY to achieve bit synchronisation. **External clocking** allows both transmit and receive to be synchronised using two independent clocking signals. These signals are generated by either a modem or modem eliminator. The selection 'CLK' on the speed selection of transmit or receive menus selects this mode.

Two pins are provided for the clock signals. Pin 15 provides the transmit clock and pin 17 the receive clock.

On transmit, a rising edge (-12V to +12V) causes a bit to change on the transmit output (pin 2).

On receive, a falling edge (+12V to -12V) causes the state of the receive input (Pin 3) to be sampled.

It is **important** that maximum clock speed on Husky should not exceed 2400Hz (giving a data rate of 2400 baud). The minimum speed is approx 100Hz.

Exceeding this clock rate may cause unpredictable results.

Alternatively, the Husky can **self clock**. This is achieved by selecting from the menu the desired transmit and receive baud rates in the range 50-1200 baud, as for asynchronous. On transmit, the data is sent out by using internal timing, with no reference to any external clock source.

To receive incoming data the Husky times from rising edges (-12V to +12V) on the receive data line to give the internal clocking.

Self clocking is useful for HUSKY - HUSKY communications. It may be used over asynchronous modems to provide both increased throughput and data integrity.

5.6.2.2 Character Synchronisation

To be able to extract eight bit data characters, the HUSKY has to synchronise itself to the incoming characters correctly. This is done by detecting special synchronising characters called SYN characters (32H). The HUSKY considers synchronisation to have been achieved after receiving two contiguous SYN characters.

After receiving the two synchronising characters each group of eight received bits are then considered to be a character.

Synchronisation is considered to have been lost if a line turnaround character is received (OFFH). This is equivalent to an open line.

Prior to sending the synchronising characters, the HUSKY transmits an OAAH pad character which is to assist in bit synchronisation.

Every data block whether it consists of one or 500 characters, must start with a pad character, and two SYN characters and terminate with line turnaround.

5.6.3 Binary Synchronous Communications (BSC)

The information carrier for 2780 is the binary synchronous communications (BSC) procedure. The character set used is EBCDIC (Extended Binary Coded Decimal Interchange Code). The following is a description of the subset of BSC used for HUSKY 2780 emulation.

NOTE: HUSKY generally operates using the USASCII (United States of America Standard Code for Information Interchange). Characters either transmitted or received by BASIC will be in ASCII, so CHR\$, LOPCHR etc., instructions all generate ASCII characters. To use EBCDIC the 2780 driver software utilizes an ASCII to EBCDIC and EBCDIC to ASCII code converter. The conversions used are in table 5.6.1 at the end of this section.

The code used on the transmission channel is transparent to BASIC programs, and need not concern the programmer.

5.6.3.1 Text Blocking

BSC defines general structural procedures for the blocking of information. Several control characters are used for the control of message blocks.

A message consists of one or more blocks of text data. The start of text character (STX) is used immediately preceding each block of data. Each data block but the last is terminated by an end of transmission block (ETB) character or an intermediate text block (ITB) character. The last data block ends with an end of text (ETX) character.

5.6.3.2 Error Checking

Each block of data is error checked by the receiver by a cyclic redundancy check (CRC). After each transmission the receiver normally replies with ACK0 or ACK1 - data accepted continue sending; or with NAK - data not accepted (e.g. due to a data transmission error), retransmit the previous block.

5.6.3.3 CRC-16

Husky uses the accepted error checking method of EBCDIC 2780. The cyclic redundancy check is a division at both the transmitting and receiving stations using the numeric value of the message as a dividend, divided by a constant. The quotient is discarded and the remainder is used as the check character. This is transmitted immediately following an ITB, ETB or ETX. The receiver checks this value and finds no error if they are equal.

CRC-16 uses the polynomial $(x^{16}+x^{15}+x^2+1)$ as the divisor constant. The 16 bit remainder is sent as two eight bit characters. This is also sometimes referred to as the BCC (Block Check Character).

- 5.6.3.4 **Line Bid**
For either the HUSKY or base computer to seize the line for transmitting a message, the line must first be seized. This requires sending an enquiry (ENQ) character. A positive acknowledgement permits the start of a message.
- 5.6.4 **Link Control Characters**
The data link is controlled by the use of the following control characters:
- 5.6.4.1 **SYN (32H) - Synchronous Idle**
This character is used to establish synchronisation. Two contiguous SYN's are required for synchronisation. They may also be used as pad characters, although never used as such by HUSKY. Syncs will be ignored after synchronisation has been established except as part of CRC.
- 5.6.4.2 **STX (02H) - Start of Text**
This character is used as the start of text preceeding all text blocks. It is not part of the CRC accumulation.
- 5.6.4.3 **ETB (26H) - End of Transmission Block**
The ETB character indicates the end of a block of data characters. A block check character is sent immediately following an ETB. A reply is required after an ETB (ACKO, NAK etc).
- 5.6.4.4 **ITB (1FH) - End of Intermediate Transmission Block**
ITB is also called IUS or US (Unit Separator). This enhances the throughput due to not having to go through a line turn-around. The CRC accumulation is reset after an ITB. It is not necessary to send an STX after the ITB.
- If the error check fails then a NAK is sent at the end of the transmitted message i.e. after the ETB or ETX. The whole of the message is retransmitted in the event of a NAK being the reply to a complete message block.
- 5.6.4.5 **ETX (03H) - End of Text**
ETX is transmitted after a block of characters started by an STX. The CRC is sent immediately after an ETX. A reply is required to indicate the receiver status.
- 5.6.4.6 **EOT (37H) - End of Transmission**
This character terminates a message transmission containing a number of blocks. It is also used to indicate a system malfunction. See section 5.6.8.
- 5.6.4.7 **ENQ (2DH) - Enquiry**
ENQ is used as the line bid to start a transmission sequence. ENQ is also used to obtain a repeat transmission of a response, in the case of a garbled response.

- 5.6.4.8 **ACK0 (10H,70H) - Affirmative Acknowledgement**
This is one of two positive acknowledgements to error checked message buffers. This informs the transmitting station that the message was received satisfactorily. This acknowledgement is also used to confirm a request for the line. ACK0 alternates with the ACK1 positive acknowledge.

Alternating Affirmative Acknowledgements

Alternating ACK0 and ACK1 provides sequential checking of replies. This ensures that the blocks are replied to correctly. ACK0 is used as the response to a line bid.

It should be noted that ACK0 and ACK1 consist of two 8 bit characters.

- 5.6.4.9 **ACK1 (10H,61H) - Positive Acknowledge**
This is the alternate acknowledge. It is used with ACK0 to respond correctly to incoming messages.

- 5.6.4.10 **WACK (10H,1CH) - Wait Positive Acknowledge**
A receiving station that has correctly received a message but is not yet in a condition to use the data, can send a WACK. It is a positive acknowledge. The transmitting station should then respond with an ENQ.

WACK is also a two character sequence. WACKs may be sent indefinitely at 3 second intervals to stop further transmission.

- 5.6.4.11 **NAK (15H) - Negative Acknowledge**
This character informs the transmitting station that a message was received in error. It causes retransmission of the erroneous message.

- 5.6.4.12 **DLE (10) - Data Link Escape**
Data link escape is used to provide various line control characters. e.g. WACK,ACK0,ACK1, and RVI.

- 5.6.4.13 **RVI (10H,7CH) - Reverse Interrupt**
Reverse interrupt is used as a positive acknowledgement (in place of ACK0 or ACK1). However, it requests termination of the current message, due to a high priority message needing to be sent from the receiving station.

Husky will never generate an RVI but will respond to one.

- 5.6.4.14 **DLE EOT (10H,04H) - Disconnect Sequence**
This sequence informs the receiver that the transmitter is shutting down.

5.6.5 **Message Transmission**

The following illustrate various message sequences and error handling.

NOTE:TX=Transmitting station. RX=Receiving station.

5.6.5.1 **Normal Message**

		ODD		EVEN			ODD		
TX	E N Q	S T (TEXT) X	E T B	S T (TEXT) X	E T B	S T (TEXT) X	E T B	E T O T	
RX		A C K 0	A C K 1		A C K 0		A C K 1		

5.6.5.2 **Unanswered Line Bid (Error 01)**

TX	E N Q	(3 sec bid timeout)	E N Q	(3 sec bid timeout)	E N Q	Number of retries -----	E O T
----	-------------	------------------------	-------------	------------------------	-------------	-------------------------------	-------------

NOTE: The HUSKY will retry for a total of 10 ENQs. Many mainframe systems never give up.

5.6.5.3 **Accepted Retransmissions**

		ODD		ODD		EVEN			
TX	E N Q	S T (TEXT-A) X	E T B	S T (TEXT-A) X	E T B	S T (TEXT-B) X	E T B	E O T	
RX		A C K 0	N A K		A C K 1		A C K 0		

5.6.5.4 Retransmission rejected (Error 04)

		ODD			ODD			ODD		
TX	E N Q	S T X	(TEXT-A)	E T B	S T X	(TEXT-A)	E T B	S T X	(TEXT-A)	E T B
RX		A C K O		N A K		N A K		N A K		N A K

NOTE: Number of retrys can vary. HUSKY will retry 10 times, but some mainframe systems will retry indefinitely.

5.6.5.5 Transmission Delay (receiver initiated)

		ODD			EVEN				ODD	
TX	E N Q	S T X	(TEXT)	E T B	S T X	(TEXT)	E T B	E N Q	E N Q	S T X etc.
RX	A C K O		A C K I		(2 sec A int)	W C K	(2 sec A int)	W C K		A C K I

NOTE: HUSKY does not count WACK-ENQ sequences. The receiver buffer may be cleared.

5.6.5.6 Transmission delay (transmitter initiated)

TX	E N Q	S T (TEXT) X	E T B	2 sec interval	T D	2 sec interval	T D	S T (TEXT) X	E T X	E O T
RX	A C K O		A C K I		N A K		N A K		A C K I	

NOTE: HUSKY does not count TTD/NAK sequences or generate TTDs.

5.6.5.7 STX lost and data ignored.

		ODD		EVEN				EVEN		
TX	E N Q	S T (TEXT A) X	E T B	(TEXT B)	E T B	(3 sec timeout)	E N Q	S T (TEXT B) X	E T B	E O T
RX	A C K O		A C K I		No response		A C K I		A C K O	

5.6.5.8 Incorrect positive acknowledgement (error 03)

TX	E N Q	S T (TEXT A) X	E T B	(TEXT B)	E T B	E N Q	E N Q	E N Q	E O T
RX	A C K O		A C K I			A C K O	A C K O	A C K O	

NOTE: HUSKY will send 10 ENQs before giving up.

5.6.5.9 Data link abortion on no response (Error 02)

	E	S	E	E	E	E
TX	N	T (TEXT)	T (3 sec response	N	(3 sec	N
	Q	X	B timeout)	Q	timeout)	Q
						T
RX		A	No		No	
		C	response		response	
		K				
		0				

NOTE: HUSKY sends out 10 ENQs

5.6.5.10 Data link stalemate

	E	S	E	
TX	N	T (TEXT)	T	No continuation
	Q	X	B	
RX		A	A	
		C	C	
		K	K	
		0	1	

Note: Failure to receive further messages could be timed out by the BASIC applications program.

5.6.6 HUSKY/2780 Specific Implementation features

The foregoing describes in general operations on the communications line for 2780 protocol. The HUSKY has particular features and handling which may be important to users.

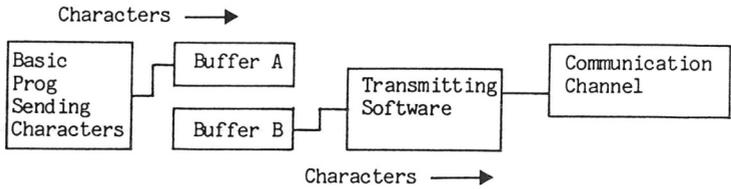
5.6.6.1 Buffer operation

To implement this synchronous protocol it is necessary to use buffering on transmit and receive so that error checking and retransmission can be performed.

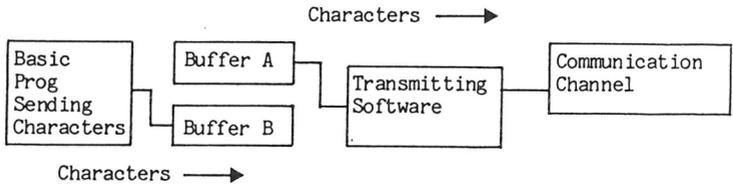
To enhance throughput there are two buffers each for transmit and receive, the operation of which is described below.

5.6.6.1.1 Transmit buffer

On transmit while one buffer is being filled from the user program the other is being sent. When the new buffer is full and the old buffer has been sent then the two buffers effectively swap over and the cycle continues:



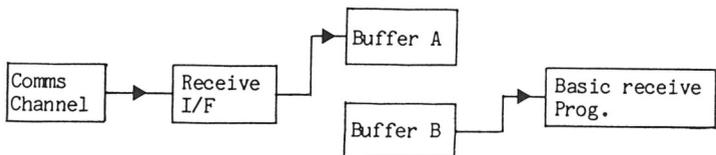
Buffer A is filling whilst buffer B transmits.



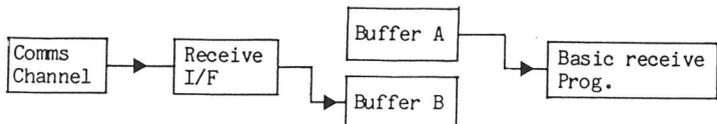
Buffer B is filling whilst Buffer A transmits.

The maximum buffer length that can be received by a 2780 protocol receiver is 512 characters. However, some systems are only configured for 256 characters. For this reason Husky will only transmit 200 characters for ease of interfacing.

5.6.6.1.2 Receive Buffer



Filling buffer A, emptying buffer B.



Filling buffer B, emptying buffer A.

As with the transmit buffer arrangement, one buffer is being filled from the communications line while the other is being emptied by the user program.

As there are 2780 protocol communication with up to 512 character buffers then Husky has receive buffers of 512 bytes.

5.6.6.2 Data Format

The 2780 batch entry protocol is designed around card formats. In many cases it is expected that the cards would transmit 80 characters irrespective of whether or not all 80 columns are used. Each card is considered to be a block of data and is therefore separated from further cards by an end of block character (IUS, ETB or ETX) and the following CRC check. If it is not necessary for a card to have 80 characters, premature termination can be caused using the EM (end of media) character, which is then followed by a block termination.

A message block may not split a valid card, so blocks are variable in size dependent upon the length of the final card. If fewer than 80 spaces exist in a message buffer at the end of a card, then the message is sent.

5.6.6.3 Husky Card Format

Virtually all Husky data is delimited by CR (carriage return). Blocks of data loaded by LINPUT statements are terminated by CR, as are lines of printout. The most straightforward method of block creation is to treat data between CR's as cards of data.

5.6.6.3.1 Transmitting Cards

On transmit if a CR is detected in the data then the communication software will treat it as a block end. The action taken depends upon the amount of room left in the current buffer being prepared for transmission. If fewer than 80 characters remain then the following is prepared:

EM ETB CRC

this is inserted in the data stream in place of the CR. The EM will inform the receiving station that a card has finished. The ETB and CRC terminate the message, which is then sent when previous buffers have been acknowledged.

If there are greater than 80 characters remaining then the following:

EM IUS CRC

is inserted into the current buffer. Further data can then be loaded into the buffer prior to transmission.

If sufficient characters are sent without a CR to fill the buffer, it is sent with a terminating ETB CRC but no EM character.

5.6.6.3.2 Sending the last Card

To maintain compatibility with existing transmission programs it is essential they should operate without modification. This leaves a problem of how to send the final message buffer if it has not been filled up. If greater than 5 seconds elapses between characters sent from the BASIC applications program then the end of a complete transmission sequence is assured and the current block terminated by ETX CRC. After this has been acknowledged the EOT (end of transmission) is sent to close down the line.

Alternatively, if an ETX is transmitted then this will be interpreted by the 2780 software as the end of transmission, and the buffer terminated and sent.

5.6.6.3.3 Receiving Cards

The computer generating data for transmission to the Husky will also block data into messages. It need not send EM (end of media) characters. If any are received by Husky then they are ignored, since Husky uses variable length records.

To provide BASIC programs with CR delimiters the end of block sequences:

IUS CRC
ETB CRC
ETX CRC

are all converted to CR (carriage return) when returned to the Basic applications program.

This does mean that Huskies will communicate properly to each other when connected back to back.

2780 receiving stations use ESC sequences for special functions like skipping line. These sequences are ESC n where n can be a number of printable characters. To avoid confusion when these are received the Husky will ignore received ESC's and their associated following character.

5.6.6.4 Character Restrictions

Husky has not implemented any of the transparency features of BSC. It is, therefore, not possible to send any of the following characters:

```
STX - 02H (ASCII)
ETX - 03H ( " )
ETB - 17H ( " )
EDT - 04H ( " )
SYN - 16H ( " )
NAK - 15H ( " )
DLE - 10H ( " )

US - 1FH ( " )
ESC - 1BH ( " )
```

If any of these characters are sent then they may either be lost or cause disruption to the transmission channel.

Maximum block length of 80 characters is specified. If more characters before a carriage return are sent then it is possible that they will cross a buffer boundary, causing a spurious end of block. This would give a receiving Husky a spurious CR.

5.6.7 Use in VDU Simulation

Although 2780 work stations are not generally interactive devices it is possible to use the Husky in this mode when, for example, testing the system with LOG ON cards, etc.

When in VDU simulation any received data is displayed on the screen.

To transmit, a message character may be typed in directly. Provided no mistakes are made, and there is less than 5 secs between each character, the message is sent on the transmission channel.

5.6.8 Communications Errors

There are a number of different line errors which may occur when using this protocol. These are detected and then displayed as communications errors. An error number is also displayed. The meaning of these error numbers is shown in section 5.6.5 (Message transmission). If use of the ON COMMS verb is made (see section 3.3.2.32) then the error number will be in the COMERR flag.

There is one exception to this rule, the unanswered line bid as shown in (5.6.5.2). This is the most usual error and will generally be due to a connection fault, as no characters have been received. The option is then given to: Retry transmission ?(Y/N) if 'Y' is pressed then the Husky will send out up to 10

more ENQs (enquiries) and the offer of retry made again. If 'N' is pressed then the error 01 is returned, and if programmed through ON COMMS, returns control back to BASIC.

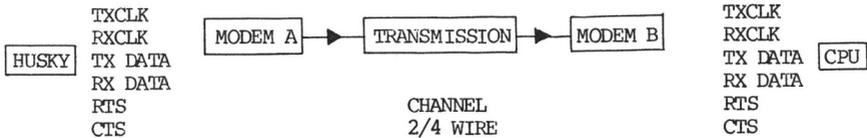
ERROR MESSAGES:

- 5.6.8.1 **Error 01: Unanswered line bid**
This is due to not receiving any reply in a request for the line, usually caused by incorrect connection of the communications channel.
- 5.6.8.2 **Error 02: No response from receiver**
After establishing the line, a buffer has been sent to which no response has been given, despite 10 ENQ's having been sent.
- 5.6.8.3 **Error 03: Response not matched by odd-even block count**
The wrong positive acknowledgement has been given to a buffer, despite confirmation requests.
- 5.6.8.4 **Error 04: Retransmission rejected**
Repeated transmission of a buffer gets only the NAK response, has tried 10 times.
- 5.6.8.5 **Error 05: No ENQ after WACK**
Having sent out a WACK no acknowledging ENQ has been received for 5 secs.
- 5.6.9 **Hardware Handshaking**
This protocol is generally used with modems or modem simulators. The use of RTS (Request to send) and CTS (Clear to send) is therefore specified by the modem, particularly if half duplex or two wire operation is desired (as on the public switched telephone network). Both RTS and CTS may be deactivated as usual. If they are used, then the following takes place:
- i) If the Husky is not transmitting then RTS is inactive, allowing the remote modem to transmit.
 - ii) When starting a transmission buffer the RTS is activated. Before transmission starts CTS is awaited from the modem. When CTS becomes active Husky starts transmission.
 - iii) During transmission RTS is kept active, but CTS is not checked as if it became inactive the transmission would be broken and synchronism dropped.
 - iv) At the end of a transmission RTS is de-activated and the Husky awaits reception.

There are no timeouts on the handshakes.

5.6.10 **Hardware/configuration**

The usual data link configuration for a Husky to mainframe is:



The transmission channel could be the switched public network, 4-wire private lines etc.

The two modems and the channel could be a modem eliminator for nearby operation.

As can be seen the modems generally create all the clocking in this simple arrangement. The Husky will not generate any actual clocking of its own, except when self synchronising to the data.

The handshake signals RTS and CTS ensure proper data flow over a half duplex line (allowing the modems to settle etc). Then may not be necessary over 4-wire links.

Other signals may be sourced by the modems (e.g. DTR etc.) but these are not currently used by the Husky.

5.6.11 **Husky to Husky connection**

To operate this protocol 'back-to-back' between two Huskies, it is only necessary to use a crossed 3-wire lead (TX,RX and ground). If the handshake lines are connected, then CTS must be deselected on the menu as the handshaking is only configured for modem operation.

The Huskies need to be set for self clocking at the designed speed, typically 1200 baud or 300 baud over slow speed async modems.

TABLE 5.6.1

ASCII TO EBCDIC CONVERSION

When using EBCDIC communications HUNTER provides ASCII to EBCDIC conversion as follows:

(all values are in Hex)

ASCII	CHAR	EBCDIC	ASCII	CHAR	EBCDIC
00	NULL	00	20	SPACE	40
01	SOH	01	21	!	5A
02	STX	02	22	"	7F
03	ETX	03	23	HEX	7B
04	EOT	37	24	\$	5B
05	ENQ	2D	25	%	6C
06	ACK	2E	26	&	50
07	BEL	2F	27	'	7D
08	BS	16	28	(4D
09	HT	05	29)	5D
0A	LF	25	2A	*	5C
0B	VT	0B	2B	+	4E
0C	FF	0C	2C	,	6B
0D	CR	0D	2D	-	60
0E	SO	0E	2E	.	4B
0F	SI	0F	2F	/	61
10	DLE	10	30	0	F0
11	DC1	11	31	1	F1
12	DC2	12	32	2	F2
13	DC3	13	33	3	F3
14	DC4	14	34	4	F4
15	NAK	3D	35	5	F5
16	SYN	32	36	6	F6
17	ETB	26	37	7	F7
18	CAN	18	38	8	F8
19	EM	19	39	9	F9
1A	SUB	3F	3A	:	7A
1B	ESC	27	3B	;	5E
1C	FS	22	3C	<	4C
1D	GS	1D	3D	=	7E
1E	RS	1E	3E	>	6E
1F	US	1F	3F	?	6F

ASCII	CHAR	EBCDIC	ASCII	CHAR	EBCDIC
40	@	7C	60	\	79
41	A	C1	61	a	81
42	B	C2	62	b	82
43	C	C3	63	c	83
44	D	C4	64	d	84
45	E	C5	65	e	85
46	F	C6	66	f	86
47	G	C7	67	g	87
48	H	C8	68	h	88
49	I	C9	69	i	89
4A	J	D1	6A	j	91
4B	K	D2	6B	k	92
4C	L	D3	6C	l	93
4D	M	D4	6D	m	94
4E	N	D5	6E	n	95
4F	O	D6	6F	o	96
50	P	D7	70	p	97
51	Q	D8	71	q	98
52	R	D9	72	r	99
53	S	E2	73	s	A2
54	T	E3	74	t	A3
55	U	E4	75	u	A4
56	V	E5	76	v	A5
57	W	E6	77	w	A6
58	X	E7	78	x	A7
59	Y	E8	79	y	A8
5A	Z	E9	7A	z	A9
5B	[00	7B	{	C0
5C		E0	7C		6A
5D]	00	7D	}	D0
5E	^	5F	7E		A1
5F	_	6D	7F	DEL	07

5.7

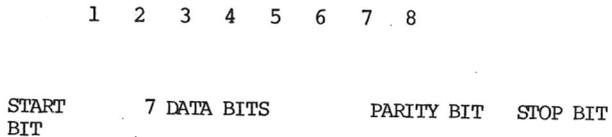
HARDWARE CHARACTERISTICS

5.7.1

Data Format

The format is of 8 bits of data, seven of which are used to define the ASCII character, the eighth bit being parity, if in use. There is one start bit and one or two stop bits:

FIG.5.3



The number of stop bits is irrelevant on reception. On transmission there are two stop bits for 110 bauds and one stop bit otherwise.

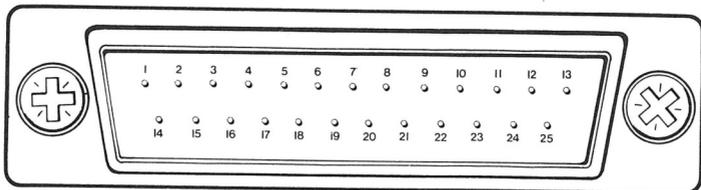
5.7.2

Interface Connections

EIA Interface

HUSKY operates on full duplex, asynchronous communication lines via a 25-pin connector which is compatible with the requirements of EIA specification RS-232-C. Table 1 summarises the EIA connector signals; the following paragraphs explain each signal as used in HUSKY.

FIG.5.4



MALE (HUSKY)

TABLE 1

EIA RS-232-C
Connector Signals

HUSKY is configured as a data terminal equipment (DTE)

Pin Number	Description	Pin Number	Description
1	Protective Ground	14	(not used)
2	Transmitted data	15	(not used)
3	Received data	16	(not used)
4	Request to send	17	(not used)
5	Clear to send	18	(not used)
6	(not used)	19	(not used)
7	Signal ground (common return)	20	(not used)
8	(not used)	21	(not used)
9	(not used)	22	(not used)
10	(not used)	23	(not used)
11	(not used)	24	(not used)
12	(not used)	25	(not used)
13	(not used)		

Protective Ground - Pin 1

This conductor is electrically bonded to the HUSKY chassis. This signal is identical to pin 7 (signal ground).

Transmitted Data (from HUSKY) - Pin 2

HUSKY transmits serially enclosed characters and break signals on this circuit, which is held in the mark state when neither characters nor break signals are being transmitted.

Received Data (to HUSKY) - Pin 3

HUSKY receives serially encoded characters generated by the user's equipment on this circuit.

Request to Send (from HUSKY) - Pin 4

Clear to Send (to HUSKY) - Pin 5

Signal Ground - Pin 7

This conductor establishes the common ground reference potential for all voltages on the interface. It is permanently connected to the HUSKY chassis.

5.7.3 Electrical Characteristics

HUSKY Output Voltages - On signals designated "from **HUSKY**", -25.0V to +1.5V or an open circuit is interpreted as a mark or unasserted state, and +25.0V to +3.5V is interpreted as a space or asserted state. Voltages greater in magnitude than ±25V are not allowed. These levels are compatible with EIA STD RS-232-C and CCITT Recommendation V24/V28.

5.7.4 Hardware Handshaking

There are two handshake lines provided. CTS is used to control transmission, and RTS is used to request transmission of data.

5.7.4.1 CTS

Clear to send is an incoming control signal used to enable or disable **HUSKY** transmission.

CTS, if enabled, is tested at the beginning of each transmitted character. If it is at logical 0 (+12V or open circuit) then the character is sent. If at logical 1 (-12V) then the character is held until CTS is enabled. A transmitted character, once started, is always completed, unless the **HUSKY** is switched off.

5.7.4.2 RTS

Request to Send is an outbound control signal from **HUSKY** to the connected Data Communications Equipment requesting permission to transmit data.

When a calling program requires to initiate the transmit function, RTS is set high to a logical 0(+Ve). CTS is then expected to be returned high at logical 0(+Ve) to **HUSKY** from the DCE interface as a delayed function of RTS, thus permitting transmission to proceed.

Immediately the stop bit of the last character from the program buffer is transmitted, RTS is set low to logical 1(-Ve), initiating a return from the DCE of CTS at a low logical 1(-Ve) state.

This is interrogated by the communication driver program and further transmission disabled.

Subsequent data transmission is enabled, i.e. the next block of data, by repeating the RTS/CTS sequence.

5.7.5 Interface Power Control

Under normal circumstances the serial output interface is powered down to conserve battery life. The first character which requires transmission causes the interface to be powered up. A pause of approximately 2 seconds is allowed for the interface to settle down and then the character is sent. Further characters are not affected in this way. Once powered

the interface is latched on until the **HUSKY** is switched off or power removed using an OUT instruction in either BASIC or machine code.

When the interface is powered up it may cause spurious characters to appear to be sent out, causing problems with the receiving system. These may be overcome by powering the interface, prior to requiring the data, by sending any character as soon as the **HUSKY** is switched on. A BASIC LPRINT command will achieve this.

Alternatively, the interface may be controlled as follows:

```
OUT 132,1
```

will turn the interface on and

```
OUT 132,0
```

will turn the interface off. If the interface is turned off after characters have been sent, then further characters will not cause the interface to be powered up and they will be lost.

TERMINAL EMULATION

5.8

5.8.1

Application

In Terminal Emulation mode, **HUSKY** can be used as:

- a) A remote terminal for dial-up time-share computer services.
- b) As a peripheral to other computers.
- c) As a portable 'Telex', able to communicate with other **HUSKIES** or data terminals.

5.8.2

Configuration

HUSKY's RS-232 serial interface is configured to represent a Data Terminal Equipment (DTE) for compatibility with Modems and other items. (See Appendix III for details).

5.8.3

Operation

First, check that the communication port is correctly initialised (see Section 5.3).

5.8.4

Selection

From 'Main Menu' select 'TERMINAL EMULATION' mode. Press 'ENTER'. You are now in **HUSKY**'s remote computer terminal simulation mode.

HUSKY is now ready to communicate.

Keyboard entry causes characters to be transmitted, including control characters (see Appendix I).

Characters sent to the **HUSKY** will appear on the screen if printable. Upper and lower case alphabets are available. (The standard **HUSKY** only transmits upper case).

To return to 'Main Menu', type control HELP. **HUSKY** will leave terminal emulation.

5.8.5

Protocols

Any protocols selected will be observed by **HUSKY**'s communication software package automatically. Remember that block protocols like ACK/NAK will not transmit until 'ENTER' (CR) IS PRESSED.

5.8.6

Access from Basic

Terminal Emulation may also be entered from Basic with the command 'CRT'. See Part 3, **HUSKY** BASIC Programming, Section 3.3.1.1, for details.

ACCESSORIES

PART 6

TECHNICAL DATA & ACCESSORIES

- 6.1 SPECIFICATION
- 6.2 REPLACING **HUSKY** FIRMWARE
- 6.3 CASE SEALING
- 6.4 BARCODE SCANNING
- 6.5 BATTERY CHARGER
- 6.6 A/D CONVERTER
- 6.7 CASSETTE TAPE RECORDER

SPECIFICATION

6.1

6.1.1

PHYSICAL

Construction	:Lightweight aluminium alloy (LM6) casting
Size	:9.5" x 8" x 1.75" (24.1 x 20.3 x 4.4cm)
Weight (including batteries)	:2Kg.
Sealing	:Waterproof against accidental immersion.
Colour	:White, Military green
Straps	:Clip-on shoulder strap

6.1.2

FACIA

Screen	:4 lines of 32 characters :5x7 dot matrix alpha-numeric, independent cursor line. :Daylight visible liquid crystal display.
Keyboard	:40 Keys arranged as 4 rows of 10 QWERTY type format. :Tactile 'feel' with audio feedback and auto repeat. :Fully waterproof

6.1.3

PROGRAMMING

Language	:Built-in Basic interpreter (the best known programming language) for user programming via HUSKY keyboard and display.
Size	:32K machine typically allows storage of approximately 2000 floating-point variables and 800 lines of program.
Storage	:Program stored indefinitely in non-volatile memory, or easily exchanged with other HUSKIES or computers. :Basic interpreter stored in firmware - doesn't use memory space.
Flexibility	: HUSKY programming suits every need; programs can be 'locked away' to avoid unauthorized tampering, or accessible for easy field modification at users' discretion.

6.1.4 COMMUNICATIONS

Type	:RS-232/V24 serial port on standard 25-pin 'D' type connector. True RS-232 levels.
Configuration	:Entirely software controlled. All parameters can be pre-set by the user or commanded by applications programs. :Full or half duplex operation with local 'ECHO' mode. :Transmit and Receive parameters set independently.
Baud Rate	:50, 75, 110, 150, 300, 600, 1200
Protocols	:Standard 'Invisible' protocols for flow control and security and transparent to user programs. (Some protocols are optional - consult factory for details). :Formats provided include: a) None - Simple 'TTY' communication b) XON/XOFF - used by most mini and mainframe computer systems. c) ETX/ACK - used by many popular printers. d) ACK/NAK - for secure telephone communication. e) System - asynchronous block-oriented protocol with BCC checks. f) Level 3 - asynchronous block-oriented protocol with identity and line turn-around features. g) 2780- Fully synchronous implementation. h) 'PIP' - as implemented in CP/M - allows hex dump/load of memory contents.

Error Checking :Odd, even or no character parity generated (transmit) or checked (receive). BCC and CRCC checks as required by protocols

Handshaking :Hardware RS232/V24 handshaking lines RTS and CTS are user selected.

Modem Control :Full modem control signals available to option, including synchronous clocking.

6.1.5

MEMORY

Type :CMOS low power semiconductor RAM.

Retention :Triple battery supported non-volatile design.

Capacity :Available in size options of:
32K, 48K, 64K, 96K and 144K bytes.

Firmware :Every **HUSKY** has a 32K firmware space used for the Basic interpreter and housekeeping features.

Upgrading :Memory sizes can be upgraded on a return-to-factory basis.

Real time Clock :Microprocessor accessible time-of-day and calendar clock. Totally independent of microprocessor.

Batteries :4 Main batteries 'C' cells, Ni-Cd rechargeable for daily use, Mallory MML400 for long shelf life and reliability. Life 14 hours (rechargeable) 45 hrs (Alkaline) approximately.

:1 Secondary Mallory PX23 mercury battery.

:Emergency - factory fitted lithium cell guards against data loss in extreme circumstances.

6.1.6

OPTIONS

I/O Port :This port is suitable for communication with electronic instruments, measuring devices, etc.

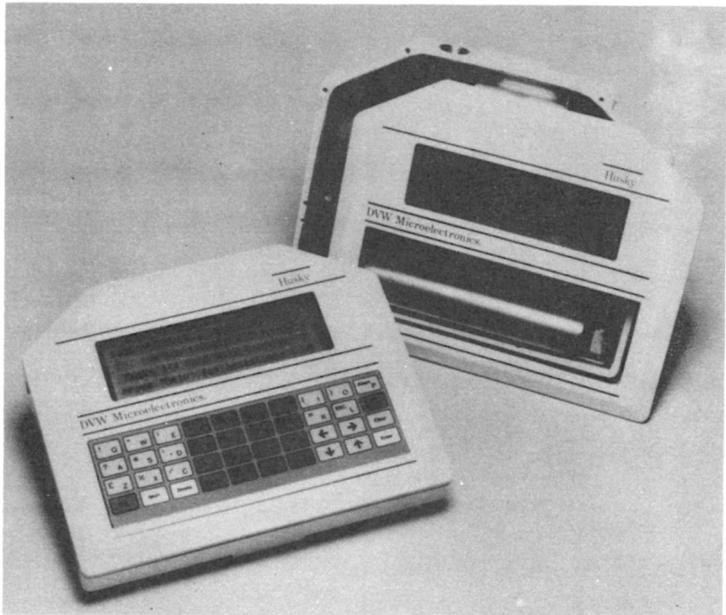
- :Available to special order on 32K and larger **HUSKIES**.
- Appearance :Special paint schemes, logos and graphics can be supplied to special order.
- Application Software :DVW will supply complete application packages at moderate cost to agreed customer specifications, or undertake training of customer personnel.

6.1.7 ACCESSORIES

- Printers :**HUSKY** interfaces, via its flexible serial port, with most types of printer. However, DVW can supply particular printers on request, including:
- :Tandy Line Printer VII. Very low cost, compact.
 - :Centronics 152/4. 132 column, 150 char/sec., 15" fan-fold paper.
- Carrying Case :High-quality leather protective case with folding cover flap. Supplied with shoulder strap and unique 'hand-free' harness.
- Acoustic Coupler :The use of a permanently installed modem is recommended. For situations where this is not practical, DVW can supply the Sendata 'Series 700' originate-answer acoustic coupler for reliable telephone communication.
- Battery Charger :Battery charger provides recharging of the installed Ni-Cd batteries whilst still inside **HUSKY**. This allows charging even when the **HUSKY** is in use.
- Communication Leads :DVW can offer serial communication leads; details available on request.
- A/D Converter :Installed internally within the **HUSKY**, the A/D converter provides 12 bit conversion and an 8-channel analogue multiplexer. This option is not available with parallel port-equipped **HUSKIES**.

HUSKY is housed in a robust cast aluminium case consisting of separate 'lid' (containing the keyboard and protective screen) and 'base' (containing batteries, electronics and connectors) units. See Fig.6.1.

Fig.6.1



6.2

REPLACING FIRMWARE

DVW pursue a policy of continuous improvement in **HUSKY's** performance.

Users may wish to update their **HUSKY** operating system to a later version, as enhancements become available. These instructions are to facilitate the quick exchange of the program ROM's by the user. It should be stressed that care should be taken in performing this operation as it is easy to damage the **HUSKY** internally, despite its tough exterior!

If any doubt is felt then the unit should be returned to Husky Computers or the local agent.

6.2.2 Removing the Lid

The **HUSKY** should be switched off before commencing any work, but it is unnecessary to remove the batteries. When the lid is off care should be taken that no foreign bodies, particularly metal objects, fall into the case.

The lid is removed by unscrewing the eight 3mm Allen headed bolts around the periphery of the base (see fig 6.2.) with the unit face down on a soft, swarf-free, surface. By holding the lid to the base the **HUSKY** may be turned back over to be the right way up. The lid may then be swung forward to reveal the inside of the **HUSKY** (see fig. 6.3). There is a flat connecting strip linking the keyboard to the main electronics board. This lead does not need to be removed. The lid may be placed face down in front of the base.

6.2.3 Releasing the RAM Board

This operation is only applicable to 32K and larger **HUSKIES**. If the optional parallel port is fitted then remove the 16 pin d.i.l. plug on the top right limb by gently prising with a small screwdriver. The RAM board itself is fixed by two cross-head screws at the front and by two 3mm nuts to the display mounting. These should be removed. (See Fig.6.4)

The RAM board will now swing to the left into a near vertical position. **DO NOT FORCE** (see fig. 6.5).

The firmware EPROM's can now be seen at the front to the left on the main board.

16K **HUSKIES** do not have a RAM board fitted. The EPROM's are visible as soon as the lid is removed.

6.2.4 Replacing the EPROMs

There are four EPROMs numbered 0,1,2 and 3. These are placed from right to left in sockets IC27, IC26, IC25 and IC24 respectively.

Fig.6.2

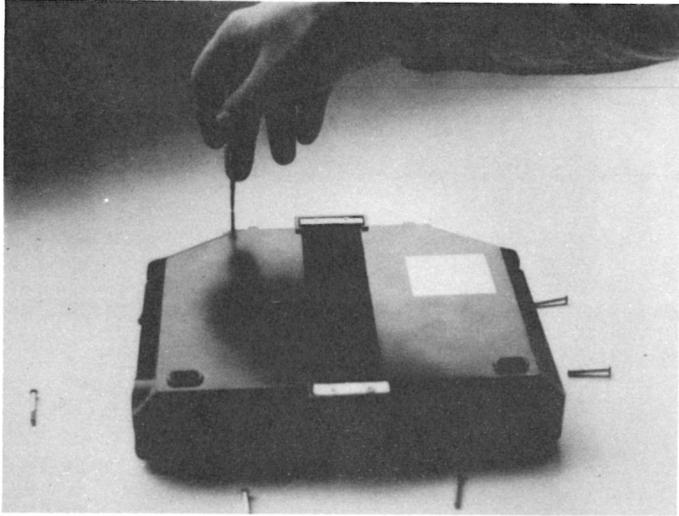


Fig.6.3

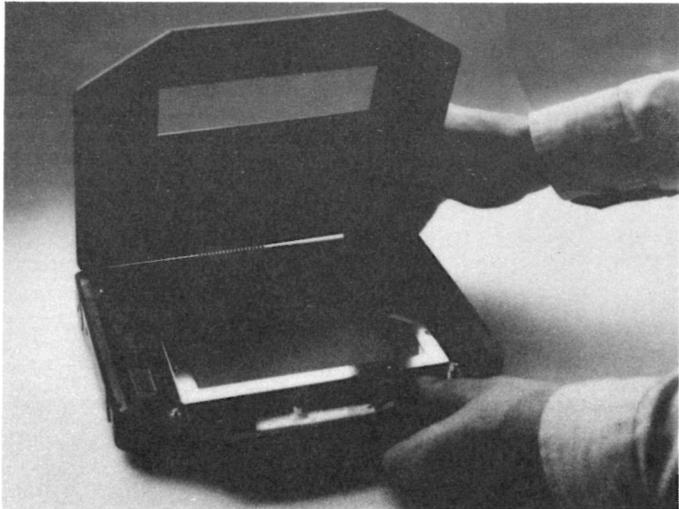


Fig.6.4

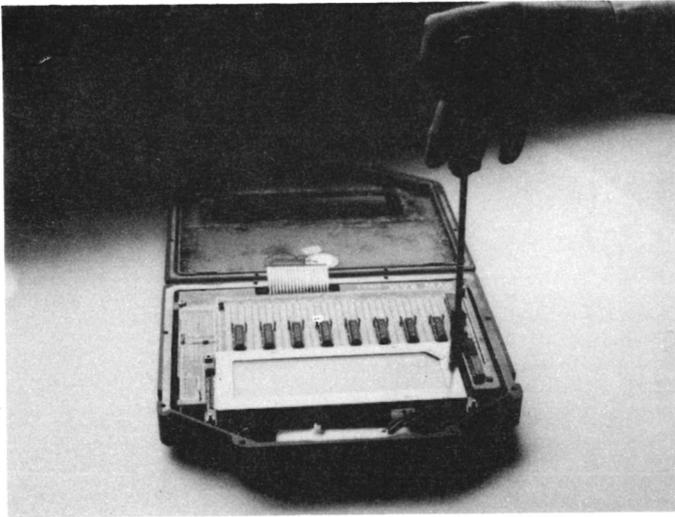
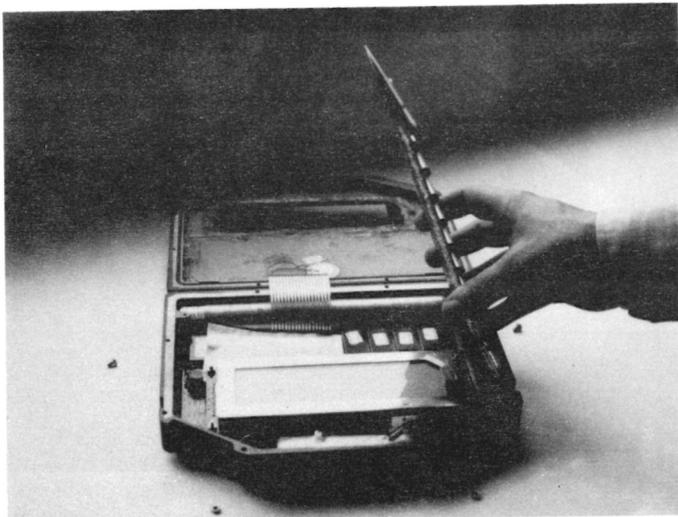


Fig.6.5



These may be taken out using either a chip removing tool or the angle of an Allen key. Do not use excessive force in removing them but gently prise them from the sockets.

The new set of EPROMs may be fitted in the same order. The parts are orientated with pin 1 at the top left. This is denoted with either a mark or a semicircle removed from one end of the chip. If the 8K byte, 2764 EPROM are used, then there are only 24 pins to fit in the 28 pin sockets provided. They are mounted as close to the battery tube as possible with four spare holes above pin 1.

Extreme care should be taken inserting the EPROMs as it is very easy to fold a pin over, or not to insert it into the socket.

Before starting reassembly check that all the pins are in sockets and that the EPROMs are pushed fully home.

6.2.5 **Replacing the RAM Board**

The board is now swung back into position and the 2 screws and 2 nuts replaced. If the parallel port is fitted, replace the 16 pin connector. Support the back of the board when pressing the parallel port connector home.

6.2.6 **Replacing the Lid**

Ensure that there is no dust or dirt on the inside of the window or on the LCD display. The display is easily scratched so care should be exercised.

The lid can now be swung into position. The flexible strip will slide back under the fitted RAM board. Push the lid assembly gently toward the top of the unit to achieve this.

The lip around the inside of the lid will fit inside the box. The unit can now be inverted and the screws replaced. Tighten down evenly until there is no gap around the sealing joint. The HUSKY is now ready for use.

6.2.7 **Changing the Desiccant**

Approximately every twelve months the desiccant should be changed or replenished. Access is as for the EPROMs, the silica gel sachet being placed just over them.

Note that the humidity indicator strip in the bottom right corner of the display reverts to a deep blue colour soon after re-assembly. If the indicator does not take on a blue colour, suspect the desiccant.

Used desiccant bags can be recharged by heating to 80-90 degrees centigrade in an oven for 1-2 hours. Alternatively, a microwave cooker will do the job in a few minutes!

CASE SEALING

6.3

6.3.1

Standard HUSKY

All **HUSKIES** are built with integral 'O' ring seals to provide a high degree of protection against atmospheric moisture, corrosive gases and accidental immersion.

6.3.2

The Seals

HUSKY has 3 principal seals

- : The lid, or cover seal
- : The main battery seal
- : The secondary battery seal

Remember that the battery compartments are vented to the interior of **HUSKY**.

NEVER OPEN THE BATTERY COMPARTMENTS IN THE RAIN!

6.3.3

The Lid Seal

This is a large 'O' ring seal running in a cast groove in **HUSKY's** lid casting. The seal seats on the machined face of **HUSKY** base.

Always ensure that the seal is firmly seated in its groove before assembling lid to base.

6.3.4

The Battery Seal

These are smaller 'O' rings running in machined grooves in the battery plugs and seating in machined faces in the **HUSKY** casting. These seals require a smear of silicone grease to assist in seating.

6.3.5

Seal Specifications

The seals used in **HUSKY** are specified as follows:

Seal	Dimensions	Standard Part No.	HUSKY Part No.
Lid	3mmx740mm	206-534-4470	279-330-3
Main battery	2mmx80mm	206-026-4470	279-330-2
Standby battery	1.5mmx30mm	206-016-4470	279-330-1.5

Replacement seals can be obtained from Husky Computers.

6.3.6

PRESSURE RELIEF

Under certain circumstances (change of altitude, etc) the interior of **HUSKY** may have a substantial pressure difference compared with the environment. This can lead to two consequences in the standard **HUSKY**:

- 1) Internal overpressure: a visible 'ballooning' of the keyboard, making the keys stiff and difficult to operate.
- 2) External overpressure: a tendency to ingest any rainwater adjacent to the seals, and in extreme cases, the possibility of activating the keys.

In either case, any pressure differential can be released by slightly loosening the standby battery cap.

If you are taking **HUSKY** on an airline trip, this is a sensible precaution to take.

6.3.7

Desiccant

HUSKY's interior is kept dry by an internal store of silica gel desiccant, kept in a muslin bag. The dryness of **HUSKY**'s interior is essential to its correct operation, and is indicated by a humidity indicator in the bottom left-hand corner of the display window.

The colour of the indicator should be checked at intervals.

THE HUMIDITY INDICATOR MUST ALWAYS BE BLUE

If it takes on a pinkish colour, the desiccant must be changed or restored promptly to avoid corrosion of internal parts. If the colour changes immediately after an accident involving water or high humidity, a fault has occurred in the sealing, and there could be water inside the unit.

If you suspect that water has entered the interior, you **MUST** take the following action:

- 1 Remove the main battery
- 2 Remove the standby battery
- 3 As soon as practicable, remove the lid
- 4 Disconnect the lithium cell at the top left-hand corner of the main printed circuit board. (The cell is retained by a phosphor bronze clip). Prise this up, allowing the cell to drop out with the **HUSKY** inverted.

- 5 Dry out the **HUSKY**, but not above 70 degrees C. Don't replace batteries or battery plugs until the unit is completely dry. Use of a conventional hair dryer is recommended.

The desiccant bag can either be replaced or re-charged by heating to 80-90 degrees C in a dry environment for 1-2 hours. See Section 6.2.7.

6.4

BAR CODES AND LIGHT PENS

6.4.1 The standard **HUSKY** is capable of reading 2 types of barcodes:

- a) Code 39
- b) EAN 8/13

Other codes are available to special order.

The "Special Facilities" menu is used to select which barcode format the **HUSKY** will read.

The options are displayed by using the | and | keys, with the displayed option being selected by pressing 'Enter'.

Example:

If the display reads:

Barcode - EAN 8/13

and 'Enter' is pressed, the **HUSKY** will read only EAN 8/13 barcodes.

This "Special Facilities" menu is accessible directly from BASIC by implementing a system call. The sequence:

A=ARG(57)
A=CALL(5)

will place the **HUSKY** in this menu.

6.4.2. **Wand Scanning Techniques**

There are a few simple rules to follow when using a hand held wand:

Check that the tip of the wand is free from dirt. Prolonged use of the wand may lead to a build up of dust **inside** the tip, covering the lens. To check for this the tip must be unscrewed, and if dust is found, then a gentle wipe with a soft cloth will remove it. The performance of the wand will not be immediately affected by a build up of dust, but rather a gradual decrease in the reliability of the wand operation. A weekly check should be all that is required for normal use.

The tips of certain wands are made of plastic and as such, pressing the wand firmly onto the bar code will result in tip wear. This will affect the wand performance since the focal length of the lens will not coincident with the distance between the barcode and the lens, i.e. the bar code will appear to be 'out of focus'. It is recommended that the wand be held gently in the hand and moved lightly across the bar code for best results.

The scan should be carried out at a constant speed with the wand in contact with the barcode throughout the scanning period. The scan should be at right angles to the bars of the bar code. However, the HUSKY tolerates a considerable variation in the scan technique. The scan can be slow, fast, traversed across the bar code in an arc rather than a perfect straight line, or even scanned in a 'wavy' line, provided that the wand remains within the bar code region.

The HUSKY will allow both Code 39 and EAN 8/13 bar code types to be scanned either left to right or right to left.

It is not necessary to place the wand precisely at the beginning of the bar code before a scan since the HUSKY will work out which information transmitted from the wand actually corresponds to the bar code itself. If the wand is resting on a bar code, it is possible to move the wand to either end of the bar code and then scan in the opposite direction. The HUSKY will then work out the beginning and end of the bar code.

HUSKY will allow a maximum of 16 characters to be represented by a Code 39 bar code. This should be borne in mind if the reader intends to produce his own Code 39 bar codes. If larger codes are required, contact the factory for details.

6.4.3 European Article Number, EAN 8/13

There are 2 EAN barcode formats:

- a) EAN 8 Short Version
- b) EAN 13 Full Length Version

Both a) and b) are used in marking retail articles of sale in shops, hypermarkets, warehouses, etc.

EAN 13 is the general name used to describe a series of barcode formats of which ANA (United Kingdom) is one particular version. See Fig.6.6 for precise details.

The general form of EAN 13 is 13 all-numeric digits, comprising:

First 2 digits

Prefix denoting the National Numbering Authority administering the remainder of the number.

Next 10 digits

National Article Number, the structure of which is determined by the National Numbering Authority.

Last digit

Check digit, calculated by modulo-10 arithmetic, i.e:

Prefix	National Article No.	Check
P1 P2	xxxxxxxxxxxxxxxxxxxx	C

e.g:

5000183962862

In the above example the prefix is 50, i.e. the numbering authority is ANA, the United Kingdom's authority. The National Article Number is 0018396286 and the check digit is 2.

The EAN 8 system is an entirely independent series of numbers of 8 Digit length. The general form of which is:

First 2 digits

Prefix, as in EAN 13

Next 5 digits

National Short Article Number

Last digit

Check digit, as in EAN 13

Prefix	National Article No.	Check
P1 P2	xxxxxxxxxxxxxxxxxxxx	C

e.g:

50159109

where, the prefix is 50 and so the National Numbering Authority is ANA. The Short Article Number is 15910, and the Check Digit 9.

The general format of EAN 13 is shown in Fig.6.7. As can be seen, there are 3 types of guard patterns; 2 normal guard patterns and a centre guard pattern. Only 12 of the 13 digits are represented by barcodes and the 13th digit must be calculated by considering the mixture of characters which represent the first 6 digits. The first 6 digits are represented by characters chosen from either Set A or Set B, whilst the last 6 digits are represented by characters from Set C only. See Fig.6.8. The allowed combinations of Set A and Set B characters are shown in Fig.6.9.

The general format of EAN 8 is shown in Fig.6.10. Note in this case that the first 4 digits are all chosen from Set A and the last 4 digits from Set C. Again, the guard bars are present.

Due to the fixed parity (all from Set C) of the last 6 digits in EAN-13 and the last 4 digits in EAN-8, both types of barcodes may be scanned in either direction, i.e. the barcodes are bi-directional.

FIG. 6.6 ASSIGNMENT OF PREFIX DIGITS BY EAN

Prefix Values

00-09	(Reserved for UPC)
20-29	In-Store Numbers
30-37	Gencod (France)
40-43	CGS (Germany)
49	Distribution Code Centre (Japan)
50	ANA (United Kingdom)
54	ICODIF (Belgium)
57	Dansk Varekode Administration (Denmark)
61-62	(Reserved DCI)
64	The Central Chamber of Commerce (Finland)
65-69	(Reserved for DCI)
70	(Norway)
73	Swedish EAN Committee (Sweden)
76	Schweizerische Artikelkode Vereinigung (Switzerland)
77	APNA Australia
80-83	(Italy)
84	AECOC (Spain)
87	UAC (Netherlands)
90-91	BAN - Austria
978	ISBN
979	Reserved for ISBN
98-99	Coupon Numbers

FIG. 6.7

12 CHARACTER BAR CODE

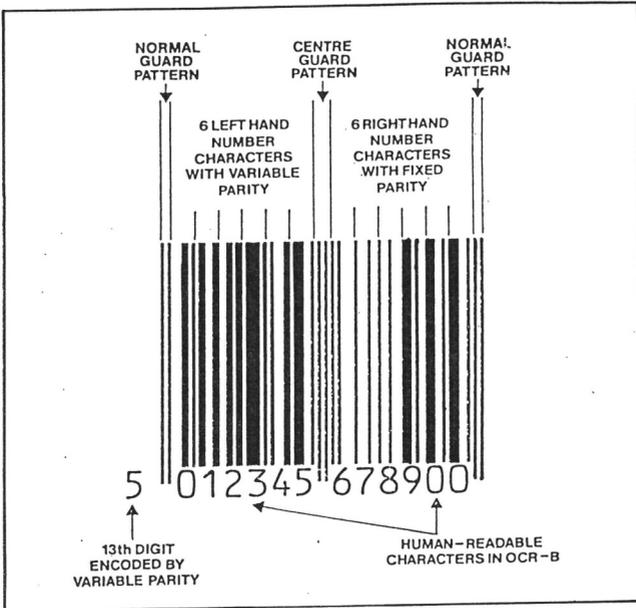


FIG. 6.8 CODING OF NUMBER CHARACTERS

VALUE OF CHARACTER	NUMBER SET A (odd)	NUMBER SET B (even)	NUMBER SET C (even)
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			

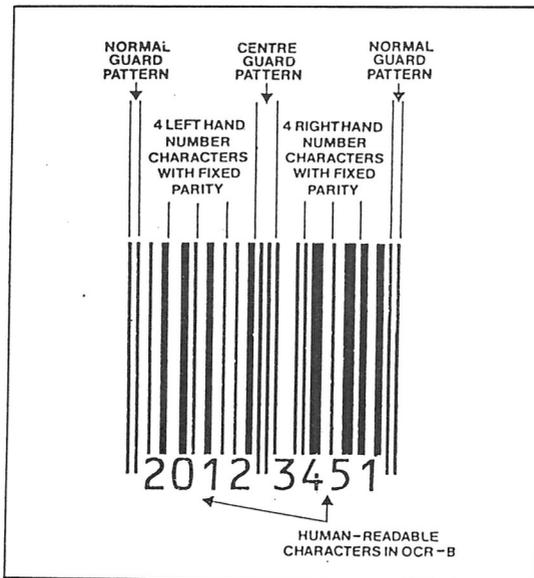
FIG. 6.9

COMBINATIONS OF SET A. AND SET B. CHARACTERS

Value of 13th Digit	Number Sets used for Coding left half of symbol					
	1	2	3	4	5	6
0	A	A	A	A	A	A
1	A	A	B	A	B	B
2	A	A	B	B	A	A
3	A	A	B	B	B	A
4	A	B	A	A	B	B
5	A	B	B	A	A	B
6	A	B	B	B	A	A
7	A	B	A	B	B	A
8	A	B	A	B	B	A
9	A	B	B	A	B	A

FIG. 6.10

8 CHARACTER BAR CODE



6.4.4 CODE 39

Code 39 is an alphanumeric bar code consisting of 43 data characters (0-9, A-Z, 6 symbols and space) and a unique start/-stop character "**". These characters are represented by light and dark bands, as shown in Fig.6.11.

Code 39 is so named due to the structure of each character being represented by 9 elements (5 bars and 4 spaces) 3 of which are wide and the remaining 6 are narrow. A wide bar or space is assigned a value 1, and a narrow bar or space is assigned a value 0. Gaps between characters have no value.

The number of characters in a code is limited only by the capabilities of the reader equipment or by human factors if a hand held wand is used, although **HUSKY** places a limit of 16 characters for normal use.

Each code starts and ends with an asterisk, "**", thus the code may be scanned in either direction.

The width of a unit bar may vary considerably allowing various printing methods to be used to produce the characters, e.g: offset, letter press, dot matrix printers.

The recommended standard density is 9.4 characters per inch, but a density of 1.4 characters per inch may be used for corrugated containers.

A complete Code 39 bar code consists of a leading white space (referred to as a quiet zone), a start character, data characters, a stop character and a trailing quiet zone.

A check digit may be produced if required and is modulus 43. The check digit is the last data character and calculated in the following manner:

Suppose the data characters are:

12345ABCDE/

then the sum of the data characters are:

$1+2+3+4+5+10+11+12+13+14+40=115$

Now, $115/43=2$ remainder 29.

The check digit is the character corresponding to the value of the remainder, which in this example is 29, i.e. "T".

Therefore, the complete data character sequence is:

12345ABCDE/T

The numeric values assigned to each Code 39 character is shown in Fig.6.12.

Fig.6.11 CODE 39 CODE CONFIGURATION

TABLE 1 CODE 39 Code Configuration

CHAR.	PATTERN	BARS	SPACES	CHAR.	PATTERN	BARS	SPACES
1		10001	0100	M		11000	0001
2		01001	0100	N		00101	0001
3		11000	0100	O		10100	0001
4		00101	0100	P		01100	0001
5		10100	0100	Q		00011	0001
6		01100	0100	R		10010	0001
7		00011	0100	S		01010	0001
8		10010	0100	T		00110	0001
9		01010	0100	U		10001	1000
0		00110	0100	V		01001	1000
A		10001	0010	W		11000	1000
B		01001	0010	X		00101	1000
C		11000	0010	Y		10100	1000
D		00101	0010	Z		01100	1000
E		10100	0010	.		00011	1000
F		01100	0010	SPACE		10010	1000
G		00011	0010	*		01010	1000
H		10010	0010	\$		00110	1000
I		01010	0010	/		00000	1110
J		00110	0010	+		00000	1101
K		10001	0001	%		00000	1011
L		01001	0001			00000	0111

The * symbol denotes a unique start/stop character which must be the first and last character of every bar code symbol. Note that the start/stop character is distinct from the "asterisk" defined in Table 6.

Fig.6.12 CODE 39 CHARACTER VALUES

0	0	F	15	U	30
1	1	G	16	V	31
2	2	H	17	W	32
3	3	I	18	X	33
4	4	J	19	Y	34
5	5	K	20	Z	35
6	6	L	21	-	36
7	7	M	22	.	37
8	8	N	23	Space	38
9	9	O	24	\$	39
A	10	P	25	/	40
B	11	Q	26	+	41
C	12	R	27	%	42
D	13	S	28		
E	14	T	29		

BATTERY CHARGER

THESE INSTRUCTIONS RELATE TO PERSONNEL SAFETY AS WELL AS RELIABLE OPERATION OF HUSKY AND BATTERY CHARGER. IT IS IMPORTANT THAT THEY ARE READ AND UNDERSTOOD.

6.5.1 DESCRIPTION

HUSKY's rechargable battery system consists of four rechargable cells and a HUSKY battery charger.

The cells are C size Nickel Cadmium (NiCad) of nominally 1.2 volts, 2.2 Ah and are capable of being charged at 65mA continually.

The special HUSKY battery charger is the only unit which may be used to recharge the NiCad batteries when they are installed in HUSKY. The use of an ordinary AC adapter/battery eliminator may damage HUSKY or the batteries.

6.5.2 WARNING

When using the charger be absolutely certain that HUSKY has four NiCad cells correctly aligned and no alternative type is present. Do not even mix NiCads of differing age or state of charge. A risk of chemical leakage, gassing or explosion exists if anything other than a matched set of recommended NiCad cells are charged. Do not attempt charging at temperatures of 50°C or less.

The charger contains lethal voltages, under no circumstances must it be opened, in any way tampered with, or used for any other purpose. It must of course be kept dry. There are no user serviceable items inside the case.

If the charger is suspected of being faulty then the fuse inside the mains plug may be changed for a similar fuse not exceeding 2 amps rating. The charger unit is fully protected against continuous short circuits but if unlikely fault conditions arise which cause overheating to the charger, it will self destruct, quietly and safely. The charger is double insulated, making no earth connection.

6.5.3 OPERATION

To use the charger unit the following procedure is recommended :

- 1) Switch off HUSKY.
- 2) Plug the charger into HUSKY's LEMO connector.
- 3) Plug the charger into the mains.
- 4) Finally switch mains on.

See section 4.6 LEMO CONNECTOR, for details of connecting and removing the LEMO plug.

HUNTER operation is possible during use of the charger but the current drain will counteract the charging current and consequently prolong the time to fully charge the batteries. Removal of the cells under this condition is forbidden.

As with all electrical apparatus the charger should be disconnected from the mains when not in use. Only 240 VAC (220 to 250V) mains, 50 or 60 Hz is suitable for standard units. Chargers for alternative mains supplies, e.g. 110V, are available.

Exhausted batteries will typically be fully charged in 12 hours, overnight charging is a popular practice.

6.5.4 NICAD REPLACEMENT

It must not be forgotten that rechargable batteries do not have an unlimited life and will ultimately need replacing as do car batteries. If the batteries become exhausted after little use then the set may need replacing. Their life expectancy can be several years but this is reduced by repeated or prolonged total discharge and by excessively high temperatures.

NOTE : Several discharge/charge cycles are required before NiCad cells reach their peak capacity. This type of battery will also self discharge, especially at elevated temperatures, which can result in a fully charged set of batteries becoming flat in a few weeks. An occasional charging session, say, every fortnight is recommended if HUSKY is being stored.

6.5.5 MAINS OPERATION

For critical or remote operations, permanent charging may be called for. This acceptable provided ventilation around the charger and HUSKY is good, but ageing of the cells may be accelerated. Annual replacement would be a prudent action in this instance.

6.6 INTERNAL ANALOGUE TO DIGITAL CONVERTER

6.6.1 OPTIONS

Specify 50Hz or 60Hz when ordering to obtain suitable rejection of line frequency pick-up.

6.6.2 CONNECTOR

Husky Socket: Cannon KPT02E-12-10P
 Matching Plug: Cannon KPT06F-12-10S
 10 pin Locking Bayonet - circular type

PIN CONNECTIONS

	A Ch 1	F Ch 2
(8 channel)	B Ch 0	G Ch 6
(common ground)	C	H Ch 4
	D Common	J Ch 5
	E Ch 7	K Ch 3

(See Section 6.6.7 for diagram)

NOTE: Channels 0,1,2,3 incorporate filters of 50 ms time constant. Allow up to 1 second for accurate readings.

6.6.3 INPUT CHARACTERISTICS

Input impedance, all channels: 10 M

Voltage range: + 4V

Resolution: 1mV

Calibration: The display reads in millivolts, in the range -4096 to +4095.

Overrange indication: Outside +4095mV

Integration period:

Version 50Hz: 20.00ms (crystal controlled)

Version 60Hz: 16.67ms (crystal controlled)

Conversion rate using ADIN: 3 readings per second (approximately)

Mains (line) rejection: 60 dB on channels 0,1,2,3. DC readings are unaffected by 10V rms superimposed 50/60Hz line pick up. Channels 4,5,6,7 provide rejection of line frequency only. Note that the Husky is available in 2 versions. The line rejection is only true if the appropriate version is selected to match the local line frequency up to about 1V p-p.

Accuracy: +10mV
 Channel Selection: Program Control
 Number of Channels: 8 numbered 0 to 7

6.6.4 **SAMPLE PROGRAM**

```

10  OPCHR1
20  ?"ANALOG READINGS : (MILIVOLTS)"
30  FOR CH = 0 TO 7
40  OPCHR 15,CH*8,1+CH/4
50  ?CH,:OPCHR 8,61
60  ?ADIN (CH),
70  NEXT
80  INCHR"PRESS ANY KEY TO REPEAT....",K
90  GOTO 10

```

NOTES

CH is the Channel number 0 to 7.

Line 10 clears the screen.

Line 30 repeats the procedure for all 8 channels.

Line 40 tabulates the eight readings.

Line 50 prints the channel number =.

Line 60 issues a measurement and prints the channel value.

Line 80 allows the values to be read until a key is pressed.

Holding a key down will cause continual scanning. If any readings is oversize, 9999 will be seen.

6.6.5 APPLICATION NOTES

1. The ADC is continually free running.
2. Source impedances of 1K or less are recommended.
3. Disconnected channels will give strange readings. If necessary, connect to ground directly or through a 1K resistor.
4. The input common pin is connected to the HUSKY case.
5. Avoid excessive input voltages. Keep within the range -10V to +10V DC to avoid damage. Up to 50V rms can be tolerated at frequencies of 50Hz and higher.
6. In addition to (5), transient protection is included against spikes of some hundreds of volts on all channels.
7. Where significant mains interference is present (e.g. 1 volt rms) or for general metering, use channels 0 to 3 which incorporate extra filtering. If the voltage suddenly changes or after first connecting, allow about 1 second for the reading to stabilise in the same way as conventional digital multi-meters. No settling time is required after changing channel. If this delay is unacceptable use channels 4 to 7 which give immediate accurate readings where no interference is present.
8. Note that the 50Hz version offers no rejection at all of 60Hz on channels 3 to 7, and similarly a 60Hz version offers no rejection of 50Hz on channels 3 to 7.
9. There are no provisions for electronic adjustment for calibration purposes. However, programmed calibration in software may be used to achieve short term accuracy approaching 1mV. For a steady clean D.C. signal short term repeatability is excellent.

6.7 CASSETTE INTERFACE HUSKY TO CRISTIE

6.7.1 There are two altenative methods of interfacing these two units.

- (a) HARDWARE HANDSHAKING
- (b) SOFTWARE PROTOCOL (X ON/X OFF)

(a) allows files to exceed the capacity of one side of one cassette, since both units standby while the cassette is turned round or exchanged. There is no limit at all imposed on the file size.

(b) permits the use of character, line or block transfers under HUSKY control. Transmission is started from software. Note, that when a cassette is turned over during playback the transmission will only recommence if the HUSKY sends a DC1 character (ASCII decimal 17), limiting file sizes in many applications.

IMPLEMENTATION	A	B	
Baud rate	1200	1200	(both send & receive)
Protocol	none	XON/XOFF	(both send & receive)
Echo	Y	Y	
Parity	even	even	
Nulls	0	0	
CTS	Y	Y	
RTS	Y (important)	Y	

6.7.2 CRISTIE SETTINGS

The Cristie unit has 16 internal switches which must be set correctly. They are arranged in two blocks of 8, blocks 1 and 2. The switches of each block are numbered 1 to 8 and are either ON or OFF.

When received from the supplier the settings will be WRONG.

IMPLEMENTATION	A (Hardware)	B (Software)
BLOCK 1, SWITCH (Upper, front)	8 ON 7 OFF 6 ON 5 ON 4 ON 3 OFF 2 OFF 1 OFF	ON (Top) OFF ON ON ON OFF OFF OFF (Btm)
BLOCK 2, SWITCH (Lower, rear)	8 OFF 7 OFF 6 ON 5 OFF 4 OFF 3 OFF 2 OFF 1 OFF	OFF (Top) OFF ON OFF ON * OFF OFF OFF (Btm)

6.7.3 CABLE CONNECTIONS

HUSKY	CRISTIE A	CRISTIE B
25 way female plug	25 way male plug	
2 (DOUT) to	2 (DIN)	2 (DIN)
3 (DIN) from	3 (DOUT)	3 (DOUT)
4 (RTS) to	20 (DTR)	4 (RTS)
5 (CTS) from	5 (CTS)	5 (CTS)
	-- 6 (DSR)	-- 6 (DSR)
	link-- 4 (RTS)	link--20 (DTR)
HUSKY Part number	S5 FM4	S5 FM1

6.7.4 OPERATION SEQUENCE

TO RECORD

1. Press "Record" button.
2. Insert tape.
3. Send file from HUSKY
4. If necessary change cassette when full. Press "RECORD" before insertion.
5. Press "END".

PLAYBACK

- | A | B |
|---|--|
| 1. Insert cassette
HUSKY will load
file as soon as
LINPUT, LLOAD etc
activates RTS. | 1. Insert Cassette |
| 2. If the cassette
ends, turn over | 2. In BASIC send LOPCHR 17
OR LPRINT Control Q
OR in VDU mode: Control Q |
| | 3. If the cassette is turned
over, repeat (2). |

6.7.5 OPERATING NOTES

1. If the HUSKY automatically runs your program at Power on, you must stop it. With the HUSKY OFF, follow this procedure:
 - (a) Press Power ON.
 - (b) Press Esc. - This must be done very quickly after (a). If the program continues, turn off and try again.
 - (c) If the display freezes, continue by slowly keying-in the code-number 56580.

If all is well the BASIC INTERPRETER will display "READY". If this is not the case try again. (This procedure is deliberately difficult to avoid accidental operation).

2. With "READY" displayed, type NEW, then press the ENTER key. This action will totally erase the HUSKY's memory and therefore confirmation is sought. Press Y.

NOTE If at any time the HUSKY is switched off, the BASIC INTERPRETER can be re-entered by pressing POWER ON, ENTER then ENTER again. "READY" will be displayed.

3. With "READY" displayed, type LLOAD, then press the ENTER key. The HUSKY is now waiting for a program to be sent to it.

4. With the Cristie unit plugged into the mains, switched on and connected to the HUSKY with the correct cable, insert the cassette which holds the program, and close the cassette door. (Identical to loading data).

5. When the program has finished loading, turn off the HUSKY, then re-enter the BASIC INTERPRETER with the sequence ON, ENTER, ENTER. Type the command RUN. Check that this is correctly displayed on the HUSKY screen before pressing ENTER.

6. All programs will differ when RUN for the first time. It is most important to NOT switch off the HUSKY whilst it is dimensioning its arrays. If in doubt, leave it for a minute until it bleeps, and a familiar display is seen.

7. Many programs feature SELFTEST in which case a CHECKSUM number is displayed. It is worth recording this number since the same program will always display the same number if correctly loaded.

8. If loading fails, check the communications parameters. To gain access from the OFF state, press ON, ENTER, 1,1,1, displays "3 Initialize communications"), then ENTER.

