Support Group Application Note
Number: 050
Issue: 1
Author:

Acorn

Master 512: Technical Information and Monitor Documentation

The information contained in this application note is designed to allow a programmer using the Master 512 to access the standard MOS calls of the Master 128 using the software interrupts provided. It details the calls, error trapping, etc which are necessary to write a program that is either to run under DOS+, or as a stand alone application in place of DOS+.

Applicable Hardware : BBC Master 512

Related Application Notes:

Support Group
Acorn Computers Limited
Acorn House
Vision Park
Histon
Cambridge CB4 4AE

OPERATING SYSTEM CALLS

Some of the operating system calls of the Master 128 may be accessed from the 80186 co-processor by using the 80186 software interrupts. There are 256 software interrupts supported and each one has a corresponding 4 byte vector in the first 1 kbyte of the 80186 memory.

Interrupts 040H to 04CH are reserved for the 13 MOS calls supported on the co-processor. All the operating system calls take parameters in the 80186 registers AL,BH and BL. corresponding to the 6502 registers A,X and Y. The operating system calls are explained below:

SUMMARY OF MASTER 128 MOS CALLS

Routine	Interrupt	Address	Function
OSFIND	040H	0100H	Open or close a file
OSGBPB	041H	0104H	Multiple Put/Get bytes
OSBPUT	042H	0108H	Write a single byte to a file
OSBGET	043H	010CH	Read a single byte from a file
OSARGS	044H	0110H	Load/save file parameters
OSFILE	045H	0114H	Load/save a complete file
OSRDCH	046H	0118H	Read character from input stream
OSASCI	047H	011CH	Write char to output strm (&D gives CR & LF)
OSNEWL	048H	0120H	Write CR & LF to screen
OSWRCH	049H	0124H	Write character to output stream
OSWORD	04AH	0128H	Various multi-byte calls
OSBYTE	04BH	012CH	Various single byte calls
OSCLI	04CH	0130H	Interpret command line

NOTE - The Master 128 MOS calls OSRDSC, OSWRSC, OSEVEN, GSINIT and GSREAD are not supported by the 80186 but OSWORD with AL = 0FAH provides the functions of OSRDSC and OSWRSC.

OSRDSC	OSWORD AL=0FAH	Read byte from screen or paged rom
OSWRSC	OSWORD AL=0FAH	Write byte to screen or paged rom

DESCRIPTION OF MASTER 128 MOS CALLS

OSFIND

Open a file for reading and writing.

On entry: AL specifies the operation.

AL = 0 File is to be closed
AL = 40H File to be opened for input only
AL = 80H File to be opened for output only
AL = C0H File to be opened for random access

DS:BX contain a pointer to the filename

Filename must be terminated by carriage return (&0D)

On exit: AL contains the file handle or 0 if file has been closed or could not be opened.

Flags: undefined

OSGBPB

Read/write a block of bytes from/to a specified open file.

On entry: AL specifies operation type from the following:

AL = 1	Put bytes to media using sequential pointer
AL = 2	Put bytes to media ignoring sequential pointer
AL = 3	Get bytes from media using sequential pointer
AL = 4	Get bytes from media ignoring sequential pointer
AL = 5	Get media title and boot option
AL = 6	Read currently selected directory and device
AL = 7	Read currently selected library and device
AL = 8	read filenames from currently selected directory

DS:BX point to a control block in the form:-

00	File handle
01	Pointer to data in either I/O processor or tube processor (low byte first)
02	
03	
04	
05	Number of bytes to be transferred (low byte first)
06	
07	
08	
09	Pointer value to be used for transfer (low byte first)
0A	
0B	
0C	

On exit: AL contains a return value where:

AL = 0	Operation attempted
AL = entry value	Call not supported in this filing system

Flags:	CF = reset	Transfer completed
	CF = set	End of file reached before transfer complete

OSBPUT

Write a single byte to a specified open file at the point in the file designated by the sequential pointer.

On entry: AL contains the byte to be written. BH contains the file handle (provided by OSFIND).

On exit: No exit values

Flags: undefined

OSBGET

Read a single byte from a specified open file at the point in the file designated by the sequential pointer.

On entry: BH contains the file handle (provided by OSFIND)

On exit: AL contains the byte read from the file

Flags: CF is set if an attempt is made to read past the end of the file

OSARGS

Load/save file parameters to/from the specified open file

On entry: AL contains the operation type (see below). AH contains the file handle (provided by OSFIND) or 0. BX points to a 4 byte attribute block.

If AH = 0,	AL = 0	Returns the current filing system in AL
	AL = 1	Returns the address of the rest of the command line in the base page control block
	AL = &FF	Update all files onto the media (make sure memory buffer saved)

If AH <= 1,	AL = 0	Read sequential pointer of file
	AL = 1	Write sequential pointer of file
	AL = 2	Read length of file

On exit: AL contains the filing system number when entered with AL = 0, AH = 0

AL = 0	No filing system
AL = 1	1200 baud cassette
AL = 2	300 baud cassette
AL = 3	ROM FS
AL = 4	DFS
AL = 5	ANFS/NFS
AL = 6	TFS
AL = 8	ADFS

Note 1: The control block always remains in the I/O processor memory not the 80186 processor memory.

Note 2: If AL=1 and AH=0 on entry, the address of the remainder of the last command line is returned in a four byte zero page block pointed to by BX. This address always points to the I/O processor and should therefore be read using OSWORD 5. The text making up the remainder of the last command line always terminates with a carriage return character (13H).

OSFILE

Read/Write a complete file or catalogue information

On entry: AL Contains the operation type:-

AL = 0 Save a block of memory as a file
AL = 1 Write the information in the parameter block to the catalogue for an existing file
AL = 2 Write the load address for an existing file
AL = 3 Write the execution address for an existing file
AL = 4 Write the attributes for an existing files
AL = 5 Read a files catalogue info with the file type returned in AL and the info returned in the parameter block
AL = 6 Delete the named file
AL = &FF Load the named file

DS:BX Point to a control block in the form:

00	Address of filename terminated by CR (&0D)
01	
02	Load address of file (low byte first)
03	
04	
05	
06	Execution address (low byte first)
07	
08	
09	
0A	Start address of data for save (low byte first)
0B	
0C	
0D	
0E	End address of data for save (low byte first)
0F	or File attributes (see below)
10	
11	

The file attributes are stored in four bytes the most significant 3 bytes are filing system specific. The LSB has the following meanings, where the relevant bit is set:

0	No read access to owner (i.e. filename /WR)
1	No write access to owner (i.e. filename LR/)
2	Not executable by owner (i.e. filename /)
3	Not deletable by owner (i.e. filename L)
4	No public read access
5	No public write access
6	Not executable with public access
7	Not deletable with public access

On exit: AL contains the file type:

AL = 0 File not found
AL = 1 File found
AL = 2 Directory found
AL = &FF Protected file

Flags: undefined

OSRDCH

Read a character from the currently selected input stream

On entry: No entry parameters

On exit: AL contains the character or an error code

Flags: CF = reset Valid character read
CF = set Error condition (value in AL)

OSWRCH

Write a character to the currently selected output stream

On entry: AL contains the character to be written

On exit: No exit parameters

Flags: undefined

OSASCI

Write a character to the currently selected output stream, but do a CR and LF if character is CR (&0D)

On entry: AL contains the character to be written

On exit: No exit parameter

Flags: Undefined

OSNEWL

Write CR and LF to currently selected output stream

On entry: No entry parameters

On exit: No exit parameters

Flags: undefined

OSWORD

Various functions using a control block

On entry: AL contains the OSWORD type (see list of OSWORD calls in the Master reference manual part 1)

DS:BX point to the control block which is call dependent (see Master reference manual part 1)

On exit: Parameters returned in control block are call dependent

Flags: undefined

OSBYTE

Various functions using registers

On entry: AL contains the OSBYTE type (see list of OSBYTE/*FX calls in the Master reference manual part 1). BL first OSBYTE parameter. BH second OSBYTE parameter (if needed).

On exit: BL contains the first return parameter BH contains the second return parameter

Flags: CF value is call dependent

OSCLI

Send a string to the command line interpreter which decodes and executes any recognised command

On entry: DS:BX point to the string

On exit: No exit parameters

Flags: undefined

For example setting up DS:BX to point to the string "cat", would produce a catalogue of the currently selected filing system directory.

Note: With this form of using OSCLI, you cannot pass variables with the string as you can from within BBC BASIC on the 65C12. Unrecognised commands will produce an error, which will be reported by the 80186 error routine, unless trapped.

Commands that can be passed to the command line, are the star (*) commands that are listed in response to a *help (OSCLI"help") command or *help application (OSCLI"help application"). i.e.

```
oscli       equ   04ch
cr          equ   13
            xor   al,al           ;clear al
            mov   bx,offset my_string
            int   oscli my_string:
```

```
ds      "help mos"
db      cr
```

Note that the string must be terminated with a Carriage return (13).

BLOCK DATA TRANSFER ON THE 80186

The 80186 ROM implements an additional OSWORD call with AL = 0FAH to allow efficient transfer of blocks of data between the 80186 processor and the host 65C12 processor. This OSWORD call is used by setting up the following control block which must be pointed to by DS:BX. The format of the control block is:

0	Number of parameters sent to I/O processor (0DH or 0EH)
1	Number of parameters read from I/O processor (01H)
2	LSB of I/O processor address
3	
4	
5	MSB of I/O processor address
6	LSB of 80186 offset address
7	MSB of 80186 offset address
8	LSB of 80186 segment address
9	MSB of 80186 segment address
A	LSB of length of transfer
B	MSB of length of transfer
C	Operation type (see below)
D	65C12 memory access control

The operation type specifies the type of transfer as follows:

0	Write to 65C12 at 24 us/byte
1	Read from 65C12 at 24 us/byte
2	Write to 65C12 at 26 us/pair of bytes
3	Read from 65C12 at 26 us/pair of bytes
6	Write to 65C12 at 10 us/byte using 256 byte blocks
7	Read from 65C12 at 10 us/byte using 256 byte blocks

The memory access control byte allows access to the paged ROMS, paged RAM and shadow RAM in the host machine and is laid out as follows:

7	6	5	4	3	2	1	0
x	sm	m/s	c	pr3	pr2	pr1	pr0

Where the bits have the following functions.

x	Unused
sm	If 3000H <= I/O address < 8000H sm=1 use screen memory regardless of state of *shadow - overrides bit 5
m/s	0 Use main screen memory if screen address specified 1 Use shadow screen memory if screen address specified
c	If 8000H <= I/O address < C000H If c=0 use specified ROM number, if c=1 use

currently selected ROM
pr3-pr0 Paged ROM number

The memory access byte is only used if the first byte of the control block is set to 0EH it is otherwise ignored. Use of the memory access byte allows paged ROM software to be copied and therefore should be restricted to system use. This however would prevent access to the shadow RAM which is not used by the system and cannot be legally accessed by other means.

A small example of the call is now given. This assumes that the control block has been setup correctly and is located in the first 64k segment. A contiguous 36 Kbyte area of memory is being used as a buffer for data written from 2000:1000 in the 80186. The host buffer starts at 3000H and extends to BFFFH. 3000H to 7FFFH is specified as shadow screen memory and 8000H to BFFFH is specified as paged RAM in bank 5.

```
osword          equ   04ah
transfer        equ   0fah
sub             ax,ax                      ;points ds:bx at control block
mov             ds,ax
mov             bx,offset transfer_block
mov             al,transfer                ;set up osword type
int             osword
transfer_block:
                db    0eh
                db    01h
                dw    3000h,0              ;base address in 65C12
                dw    1000h,2000h          ;base address in 80186
                dw    9000h                ;length = 36K
                db    6                    ;fast 256 byte blocks
                db    025h                 ;use shadow and paged RAM
```

ERROR HANDLING BY THE 80186 MONITOR

When an error is generated by the host processor the error number and string are passed across the Tube to the 80186 under interrupt. The error number and string are then placed in the error buffer of the 80186 and a pointer is initialized to point to the error number. The error string is terminated by a null byte (00H). The 80186 Tube code then jumps to the error handler, this prints out the error before returning control to the 80186 monitor.

The locations of the error handler vector and error pointer are given below:

0000:05F4	Error pointer offset
0000:05F6	Error pointer segment
0000:05F8	Error handler vector offset
0000:05FA	Error handler vector segment

Error handling by stand alone applications

The error handling provided by the 80186 monitor is not suitable for stand alone languages, that is languages which only use MOS functions and the host machines filing systems, not the DOS+ operating system; as control is returned to the monitor by the default error handler. When the language is started up it should initialise the error handler vector to point to its own error handler, which should then be able to deal with the error in an appropriate way and return control to a suitable point within the language.

An example is now given to illustrate a typical error handler. This assumes that the language is running at 0000:8000. The example is written using the Digital Research RASM86 assembler format.

```
        cseg    0
        org     08000h

        osnewl  equ     048h
        oswrch  equ     049h

        error_pointer_offset    equ     .05f4h
        error_pointer_segment   equ     .05f6h

        error_handler_offset    equ     .05f8h
        error_handler_segment   equ     .05fah

        ; initialise error handler to point to my error handler

                sub     ax,ax
                mov     ds,ax
                mov     ax,offset my_error_handler
                mov     error_handler_offset,ax
                mov     ax,seg my_error_handler
                mov     error_handler_segment,ax

        my_error_handler:
                lds     si,dword ptr error_pointer_offset
                int     osnewl                  ;new line
                inc     si                      ;skip error number
                cld                             ;set forward direction

        my_error_loop:
                lodsb                           ;get error string from buffer
                int     oswrch                  ;and write it out
                test    al,al                   ;end of string?
                jnz     my_error_loop           ;no - get next character
                jmp     my command_loop         ;yes - jump to command loop
```

80186 ERROR MESSAGES

Errors can also be generated by the 80186 using interrupt 04FH and following it with the error number and the error string, terminated with a null byte (00H). The error pointer will be initialised as for 65C12 errors and the same error handler will be used as given by the error handler vector.

An example is given below to illustrate the use of 80186 errors. In the following example a test is being made for the presence of a file before attempting to load it. The example assumes that the file name is in the current data segment.

```
; set up parameters

error                   equ   04fh    ; the error interrupt number
osfind                  equ   040h

open_for_input          equ   040h
not_found_error         equ   06dh
cr                      equ   13

cseg

look_for_file:
                        mov           al,open_for_input
                        mov           bx,offset my_file_name
                        int           osfind
                        or            al,al
                        jnz           load_the_file
                        int           error
                        db            not_found_error,'cannot find file',0

;note no return after writing out error

;file loaded here if present

load_the_file:

dseg

my_file_name:

                        db            '$.myfile',cr

;end
```

ESCAPE PROCESSING

When an escape condition is detected by the 65C12, the top bit of the escape flag at 0000:05F2H on the 80186 is set under interrupt. An escape condition should be tested for by checking this escape flag. If an escape condition exists the escape must be acknowledged using OSBYTE with AL=07EH and an optional 80186 error message can be generated. The escape flag should not be set or reset directly as the change will not be reflected on the host side of the Tube. OSBYTE calls with AL = 07CH or 07DH should be used to set or reset the escape condition.

THE 80186 MONITOR

After enabling the co-processor and either pressing ESCAPE before loading DOS+, or pressing BREAK from within DOS+ you should get the following displayed on the screen:

```
Acorn Tube 80186 512K
Acorn ADFS
BASIC
*
```

The star (*) prompt indicates that the 80186 monitor has been entered and is waiting for commands to send to the command line interpreter on the 80186 or the 65C12. In addition to the standard MOS and filing system commands, the 80186 recognises the following monitor commands:

Name	Function
*D	Memory dump in hex and ascii
*DOS	Boot DOS+ from hard disc or floppy
*F	Fill memory with a byte or word
*GO	Jump to a specified address
*MON	Re-enter the monitor
*S	Alter memory using hex or ascii
*SR	Search memory for a specified text string
*TFER	Transfer blocks of memory between 80186 and 65C12

Note that from within the monitor you cannot run 65C12 languages such as BASIC, but 65C12 utilities such as Advanced Disc Toolkit will work.

You cannot use the monitor from the 80186 under DOS+ by using the STAR utility, you will in fact find that the monitor is not resident and will not appear on the *HELP table from under DOS+.

The above commands are now explained in more detail. Where <offset> is used it refers to the hexadecimal offset address which can be entered as 1 to 4 digits - leading zeros (0) can be omitted i.e. 7A can be entered as 7A, 07A or 007A. If more than 4 hex digits are entered the most significant digits will be truncated i.e. 12345 will be treated as 2345. Where <segment> is used it refers to the 80186 segment address which can also be entered as above, but must be followed immediately by a colon (:) to indicate that it is a segment address. i.e. 23: .

In all relevant commands below if no segment address is specified then the most recently specified value will be used. If no previous value has been specified then the value should be 0. For all commands any leading spaces or asterisks or trailing spaces will be ignored. Items enclosed in <> brackets indicate parameters that the command uses, those that are also enclosed in () brackets indicate optional parameters that do not need to be specified. None of these commands are case sensitive, so both upper and lower case or a mixture of both may be used.

***D Memory dump**

Syntax: *D (<segment:>) (<start offset>) (<end offset>)

Function: This command produces a memory dump from the 80186 memory between the specified addresses in hex and ascii, showing the addresses in segment:offset form. Characters outside the ascii range 20H to 7EH are shown as a full stop on the ascii list with their corresponding hex value in the hex list. All of the parameters in this command are optional, If the segment address is omitted the last used segment value will be used. If the start and end offsets are omitted the last end address + 010H is used as the start address and the last end address + 080H is used as the end address, if just the end address is omitted then the start address + 080H is used.

For example: `*D 0000:8000 8050`

```
0000:8000 04 48 BA 6D 01 8B F0 05 50 00 3D 64 00 72 03 2D .H.m....P.=d.r.-
0000:8010 64 00 50 33 DB B9 0C 00 3B 97 E4 1A 76 05 83 C3 d.P3....;...v...
0000:8020 02 E2 F5 D1 EB 53 4B D1 E3 8B C6 8B CF 83 FB 02 .....SK.........
0000:8030 75 0F A9 03 00 75 0A D1 E8 D1 E8 40 3B C1 75 01 u....u.....@;.u.
0000:8040 42 2B 97 E4 1A 58 8A F0 59 58 C3 BB C2 1A EB 0F B+...X..YX......
0000:8050 80 3E 00 0F 00 74 05 B2 01 E8 7B EE BB B4 1A 8B .>...t....{1 2
```

***DOS Re boot DOS+**

Syntax: *DOS

Function: Allows DOS to be booted without CTRL+BREAK i.e. from stand alone languages or applications, or restart DOS after pressing BREAK to leave DOS. This command will try to boot DOS from a hard disc if one is present or from floppy.

***F Fill memory with a constant**

Syntax: *F (<segment:>) <start offset> <end offset> <fill byte|word>

Function: This command fills the 80186 memory within the specified range, with the specified constant. The constant used can be specified as a byte or word value. The end offset specified is the end address + 1 used by the fill command i.e.

*F 1000 1010 55

Will fill bytes 1000H to 100FH inclusive with the value 55H

*F 1000 1010 1234

Will fill bytes 1000h to 100FH inclusive with the word 1234H with the Lsb written first. i.e. 1000H will be 34, 1001H will be 12, 1002H will be 34, etc.

An end offset of 0 can be used to specify a fill operation to the last address in the specified segment.

*GO Jump to a specified address

Syntax: *GO (<segment:>) <offset>

Function: This command calls and transfers control to a piece of code that is resident at the specified address in the 80186. This command should be used with care, as calling an address which does not contain any executable code could cause the machine to hang or crash.

*MON Enter the 80186 monitor

Syntax: *MON

Function: Allows the monitor to be re-entered from stand alone languages or applications without pressing BREAK, or from within other routines.

*S Edit memory contents

Syntax: *S (<segment:>) <start offset>

Function: This utility allows the memory contents of the 80186 to be examined and altered if required. A line of 16 bytes of memory is displayed in hex and ascii formats initially with the cursor positioned under the least significant digit of the first byte specified.. The cursor movement and data entry is controlled using the following keys:

Cursor left	Move left - If at far left display previous 16 bytes
Cursor right	Move right - If at far right next 16 bytes
Cursor up	Display next 16 bytes
Cursor down	Display previous 16 bytes
Shift+cr left	Move cursor to far left of current field
Shift+cr right	Move cursor to far right of current field
Copy	Toggle between hex and ascii entry

The display consists of two 16 byte fields which are the hex display and the ascii display. The copy key is used to switch between the two.

For example: *S 0040

```
SEG :OFFS        HEX FIELD                                      ASCII FIELD
0000:0040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:0030 D6 1F 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:0010 00 00 00 0D 00 00 00 00 00 00 00 00 00 00 00 00 ................
0000:0020 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

```
0000:0030 D6 1F 00 F0 00 00 00 00 00 00 00 00 00 00 00 00 ................
```

While the cursor is in the hex field data is entered in hex digits, each digit being shifted in from the right as they are entered, i.e. to enter 23 you would move to the hex entry for the required address and type in the number 2 giving 02 then 3 which would cause the 2 to move left giving 23. If you now enter 4 then the 3 will move left replacing the 2 and the 4 will be entered giving 34.

If the cursor is in the ASCII field data is entered as ASCII bytes i.e. characters from the keyboard, including control characters (CTRL+char). To advance the cursor to the next byte, the cursor keys are used in the hex field, but it is done automatically in the ASCII field. When text is entered into the last field on the right, the cursor is advanced to the first field of the next 16 bytes.

This command is terminated by pressing ESCAPE.

***SR Search memory for a text string**

Syntax: *SR (<segment:>) <start offset> <end offset> <"string">

Function: Search memory for a specified text string reporting all occurrences in the segment:offset form. i.e.

```
*SR 0000 0200 "AAAA33"
0000:0040
0000:00C6
0000:011C
*
```

The address given is of the first byte of the matching string. The search string must be enclosed in double quotes (") and can be up to 72 characters in length (the maximum length of a command line is 80 characters). The end offset specified is the end address +1 of the search area, so to allow the search to continue right up to the end of a segment an end address of 0 can be specified i.e.

*SR 4000 0 "fred"

This will search from 4000H up to FFFFH inclusive. The condition for a string to be found is that it must be completely contained within the search area, i.e. if the string "fred" lives at 03FFDH then

*SR 0 4000 "fred"

Will not report it but if our string "fred" lives at 03FFCH then the above search will find it. Any 8 bit character string can be searched for using escape sequences to allow control codes and characters above 07FH to be specified. The | character is used to denote an escape sequence. i.e. |@ is ascii 0 and |G is ascii 7. |? is ascii 7F and characters over 80H as preceded by |!.

Any escape sequences that are not recognised are reduced to the argument alone, i.e. "|1" is reduced to "1". Any surplus |! operators are ignored, i.e. "|!|!|@" is reduced to "|!|@".

***TFER Transfer blocks of memory between 80186 and 65C12**

Syntax: *TFER <I/O address> (<segment:>) <offset> <length> <R/W>

Function: This utility allows fast block transfer of memory between the 80186 co- processor and the 65C12 host processor. The direction of transfer is specified by the final parameter which must be R (read) or W (write). W indicates a write to 65C12 memory from 80186 memory. R indicates a read from 65C12 memory to 80186 memory. The transfer is implemented using OSWORD 0FAH (described below) and is optimised to use fast transfer types 6 and 7 (10us/byte) where possible. If the transfer length is not a multiple of 256 (FFH) bytes any remaining bytes are transferred using types 0 and 1 (24us/byte).

REFERENCES

Detailed technical information on the 80186, including the extensions to the instruction set over the 8086 can be found in the 80186 or the 80C186 data sheet which is available from Intel.

A large amount of technical information on the Master 512 can be found in the Dab Hand Reference Guide to the Master 512, by Robin Burton.

Technical documents are available on the Tube interface and how it works. These are available from The Acorn Customer services dept.

Dabs Press, 76 Gardner Rd, Prestwich, Manchester
Glentop Publishers Ltd, Standfast House, Bath Place, Barnet, Herts, EN5 5XE
Digital Research, Oxford House, Oxford Street, Newbury, Berks, RG13 1JB
Intel UK Ltd, Pipers Way, Swindon, Wilts, SN3 1RJ

FOR REPAIRS

RCS, Headway House, Christy Estate, Ivy Rd, Aldershot, Hants, GU12 4TX. Tel: 0252 333575
Eltec Services, Campus Rd, Listerhills Science Park, Bradford, BD7 1HR. Tel: 0274 722512
Gosling Electronics, Hadleigh Rd, Ipswich, IP2 0ER. Tel: 0473 230075