FLEX EXPLAINED

Paul Izod and Alan Stirling look at the industry standard FLEX OS with particular reference to the *E&CM* hi-res computer system.

It has often been pointed out in this series of articles that FLEX only comes to life when disc drives are added to a system. Last month we described the disc controller card for the *E&CM* computer system and this month we look at the inplementation and operation of the FLEX OS on the system.

Before going into the details of FLEX, it is necessary however to discuss the general features of single-user operating systems. Since these all require the use of disk drives, a brief explanation of the means of data storage on a disk is appropriate.

The Data Format

The data areas on all disks are split up into circular tracks on the disk, with each of these tracks sub-divided into sectors – normally between 10 and 32 per track. Each sector holds between 128 and 1024 bytes of information. The smaller the number of bytes per sector, the more sectors per track. A 5¼" disk holds about 2,560 data bytes per track in single density. A 40 track single-sided/single-density 5¼" disk has a total capacity of 102,400 bytes. The start of each track is marked by an index hole in the disk near it's hub.

There are two systems used to correctly identify the sectors around each track. The most popular, called 'Soft Sectored', uses a small header area recorded onto the disk, in front of each sector, which holds the track and sector number of the sector that follows (see **Fig 1**). This information is recorded at the time that the disk is formatted, since without it the disk drive head assembly is not able to check its position on the disk. The other method, called 'Hard Sectored', used extra holes around the hub of the disk to mark the start of each sector. Extra circuitry counts pulses generated by these holes and generates the sector numbers. Soft sectored disks operate more reliably, since each sector is uniquely coded, but this is at the expense of total data capacity, as the header information takes up valuable data information space on the disk.

The interface between the computer system, and the disk drive is undertaken by a disk controller card. The key component on this board is the disk controller chip, which controls the operation of the disk drives completely. Since this chip is itself as complex as a small microprocessor, it only requires a few support chips to fully control most types of disk drives. It is the task of the disk controller card to communicate with the disk drive, reading or writing sectors of data as required. Only complete sectors of information are transferred.

Due to the intelligence of the disk controller chip, it is only necessary to write short machine code routines (called "Drivers") to gain access to any individual sector. With the addition of some form of disk drive selection hardware, it becomes a simple matter to read sectors from one disk drive and write them to another, thus copying data between two disks. Without a detailed record of where data is stored on each disk however, this basic system is unable to ensure that the sectors that it is reading contain the required data and the sectors that it is writing to do not already contain valuable information.

What is needed is a system of indexing the data stored on a particular disk. Although this index or directory can be held in memory, the obvious place to keep it is on the disk itself, alongside the data. In all disk operating systems, the start of this directory is defined, so that the machine knows where to start searching for the name allocated to the data for which it is searching. Alongside the name entry in the directory is further information about the data. This includes the track and sector where the data starts and finishes, the number of sectors that the data occupies and the type of data itself (i.e. machine code programs, text files, basic programs etc.). Data in this format is normally known as a file.

It is for the control and manipulation of these files and their relevant directory entries that disk operating systems (DOS) have been developed. However since disks using the same DOS will normally be compatible, meaning that disks recorded on one systemn can be read by another – even if the two systems are produced by different manufacturers, most operating systems go further, providing a completely compatible software environment within each machine, so that programs written to run on one type of machine can be transferred on compatible disks to another machine, giving identical results when run.

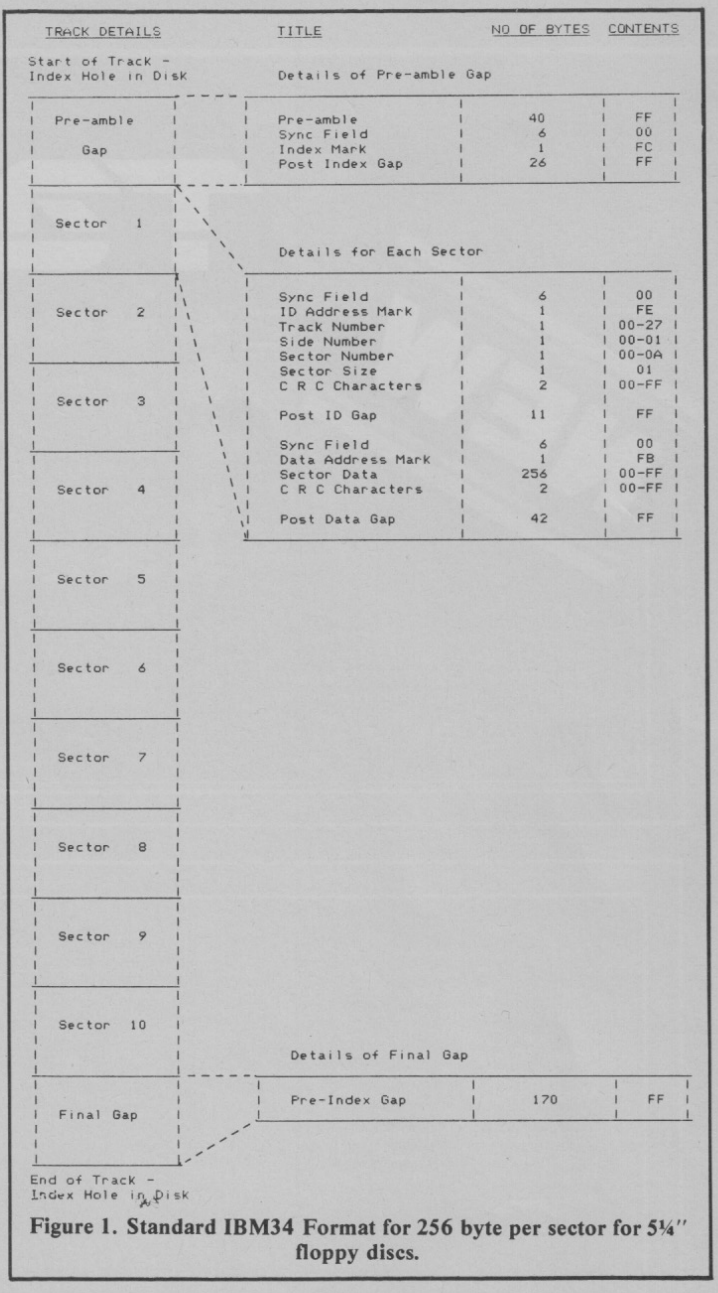
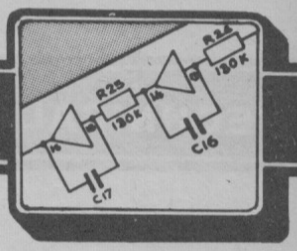


Figure 1. Standard IBM34 Format for 256 byte per sector for 5¼" floppy discs.

Software Compatibility

This is obtained using standard routines within the DOS. The address of these routines and their functions is provided to the programmer within the documentation of the DOS. The addresses and functions of



these routines do not change between different types of machines, providing the DOS is the same. Any differences between the machines hardware configurations, such as the address of the I/O port or memory mapped screen are taken into account when the DOS is implemented on that machine. During the implementation, machine code routines are used to link the DOS to the I/O routines of the machine. It is important that these routines are written such that they allow the DOS to perform in identical ways on differing machines. Programs written to link with the standard addresses defined in the DOS therefore should have the same results on any system. It is difficult however, to write sophisticated output routines without knowing the exact screen size and format. Thus the standardisation is limited to programs with simple I/O requirements– programs with complicated graphics requirements will only run on similar hardware, since this will be accessed directly by the program.

An Overview Of FLEX

FLEX was originally written in 1977 for systems using the Motorola MC6800 processor, by Technical Systems Consultants Inc, of Forest Hill, North Carolina, USA. In 1977 it was converted to run on the MC6809 and has become a standard for these machines. It is used on many makes of machine, including SWTPC, Positron, Gimix, Smoke Signal and Mororola, with implementations on Apple II, Tandy Colour Computer and Hitachi machines. With many users world-wide there is a wide range of system software with a number of good applications packages available. With the recent adaptation for the Dragon 32, this operating system will become even more popular.

FLEX Disk Formats

Although many other operating systems use sectors of 128 bytes, FLEX uses 256 bytes per sector. This gives 10 sectors per track on 5¼" single sided/single density (SS/SD) disks, or 15 sectors on 8" SS/SD. In FLEX the sectors are numbered from 01 (Hex) upwards and the first and outermost track is numbered 00 (Hex). **Table 1** gives details of the track, sector & capacities for the more popular disk formats.

The figures in this table reflect the total amount of data that can be stored in practice, since the theoretical maximum can never be reached since track zero on each disk is reserved for the directory and other information, sectors themselves only hold 252 bytes of data, and track zero is always recorded in single density, even on double density disks in order to maintain compatibility with single density systems.

FLEX File Format

Files can be of any length, and are made up from a collection of linked sectors. These are linked by the first two bytes of each sector, which hold the track and sector number of the next sector of the file. In this way a file can occupy any unused sectors on the disk, since successive sectors do not have to be physically consecutive. The last sector of a file has these two link bytes set to zero. On most files the following two bytes hold the consecutive sector number within the file, starting with sector 1 as the first sector. Since each sector has 256 bytes and four are used by the system, there remain 252 bytes available to the user. This slight loss in storage capacity is outweighed by the flexibility of file layout on the disk. Even the directory information on track 0 uses the same format.

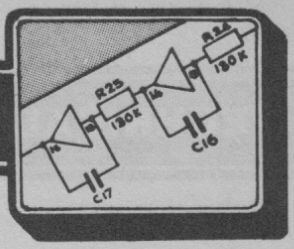
All empty or unusued sectors on the disk are linked as one large file, called the 'Free Chain'. This file also uses the track and sector links, but does not maintain a consecutive sector number.

When a file is to be written to the disk, FLEX removes the first sector from the free chain, and allocates it as the first sector of the new file. As the new file requires more sectors, FLEX re-allocates them from the free chain to the new file. They are already linked toghether, since they were linked in the free chain. At the end of the file, FLEX un-links the last sector of the file, (track and sector equal to zero), updates the directory entry for the file, and and changes the pointer to the start of the free chain. If a file is deleted, it is linked to the end of the free chain and it's entry is deleted from the directory. Although it has been deleted, with it's directory entry removed, the data will still be intact, until overwritten by another file. The disk is full when the free chain has no further free sectors.

TABLE 1 – Track, Sector and Byte Capacities of Different Sizes of FLEX Disks.

Disk Size	No. of Sides	Data Density	No. of Tracks	Largest Track Track	Largest Sector (Dec/Hex)	Total Disk Secotrs	Total Data Sectors	Total Data Bytes
5"	1	Single	40	39/27	10/0A	400	390	98280
5"	1	Double	40	39/27	18/12	712	702	176904
5"	2	Single	40	39/27	20/14	800	780	196560
5"	2	Double	40	39/27	36/24	1424	1404	353808
5"	1	Single	80	79/4F	10/0A	800	790	199080
5"	1	Double	80	79/4F	18/12	1432	1422	358344
5"	2	Single	80	79/4F	20/14	1600	1580	398160
5"	2	Double	80	79/4F	36/24	2864	2844	716688
8"	1	Single	77	76/4C	15/0F	1155	1140	287280
8"	1	Double	77	76/4C	26/1A	1991	1976	497952
8"	2	Single	77	76/4C	30/1E	2310	2280	574560
8"	2	Double	77	76/4C	52/34	3982	3952	995904

(Dec: Decimal, Hex: Hexadecimal)



Track Zero Information

This track has a special significance since it holds the disk directory and certain other important information.

The first sector holds the boot loader program. This program is machine specific, and is called into the machine by the monitor program, when FLEX is booted. Once loaded into the machine, it loads FLEX into memory, from wherever it is located on the disk. Finally, it passes control to FLEX, which begins operation. If the boot program is too large to fit into one sector, the second sector may also be used.

The third sector is called the 'System Information Record' (SIR). This holds information regarding the disk itself, and the chain of free sectors on the disk. See **Table 2.** This information is read by FLEX prior to accessing a disk file, so that FLEX can adapt to the individual disk's format.

TABLE 2 – List of Information held on System Information Record (SIR).

Volume Name of Disk	(8 Characters)
Volume Number of Disk	(1 – 65535)
Creation Date	(MM/DD/YY)
Start of Free Chain	(Track/Sector)
End of Free Chain	(Track/Sector)
Length of Free Chain	(No of Sectors)
Highest Sector Address on Disk	(Track/Sector)

Sector 4 is left blank, and is reserved for future expansion.

The directory file starts at sector 5, and continues through all the other sectors on track zero. Each directory entry takes 21 bytes, giving 12 entries per sector. If more directory entries are required, over and above those available on track zero, FLEX automatically allocates extra sectors as required from the free chain.

FLEX File Specifications

As with all operating systems, there is a structured syntax to be complied with, when entering the details of a disk file to be used. FLEX numbers the disk drives from '0' to '3', this number being used at the beginning of the file name. The file name itself must not be more than 8 characters. FLEX also allows files to use extension names of up to three letters, in order to describe the file type. The main extensions used are:

BIN	Binary Program Files
TXT	Text files
CMD	FLEX Command Files
BAS	Basic Source Code Files
SYS	FLEX System Files
BAK	Back-up Copy of Text File
BAC	Basic Compiled (Tokenised) Files
OUT	Printer-Spooling Output Files

The drive number normally precedes the file name, whereas the extension follows it. They are separated by a full stop, thus '1.LETTER.TXT', is a text file on drive '1', called 'LETTER'. Many commands presume default options in terms of drive number and extension, thus in most situations it is only necessary to enter the file name alone.

FLEX Command Structure

There are only two memory resident commands within FLEX, although others can be added if required. 'GET' is used to load a binary file into memory, where it will be loaded into the same memory locations from which it was saved. 'MON' is used to exit from FLEX, back to the system monitor. All other commands are loaded from disk. To initiate a command, simply enter the name of a command file. FLEX will look for a file with the same name, on the system disk, (normally Drive '0'), and with the extension of '.CMD', signifying a command file. Thus entering 'CAT' will loade the catalogue command from drive '0'. Its full file specification is '0.CAT.CMD'. Any of the default options can be changed merely by entering the option desired. For example to run a binary program on drive '1', called 'TEST', enter '1.TEST.BIN'.

The system also controls the default options on two drives, known as the 'System Drive' and the 'Working Drive'. The system drive is where FLEX looks for command files, and the working drive for text and other files. Thus if the system drive is set to '0' and the working drive to '1', the command 'LIST LETTER', will load the command file '0.LIST.CMD', which will in turn list on the screen the contents of the text file '1.LETTER.TXT'. These default drive numbers can be controlled using the 'ASN' (Assign) command. Using this same philosophy, many commands use a string of parameters entered after the command name in order to control the detailed operation of the command.

Within this simple framework, it is possible to control the operation of a sophisticated 6809 based computer, particularly if the range of commands is augmented, which is done by just adding the command file to the system disk.

E&CM

Next Month: More on Flex and the *E&CM* Hi-Res Computer.

FLEX EXPLAINED

Part 2

Paul Izod and Alan Stirling continue their look at implementing the FLEX OS on the *E&CM* hi-res computer system.

There are two groups of machine code routines that must be added to FLEX before it will run on the target machine. The disk driver routines, which control the operation of the disk drives, and the I/O or console driver routines, which enable FLEX to access a serial terminal, or keyboard and output screen. This second group of routines normally just link FLEX into similar routines already provided as part of your monitor program. Sample routines are provided with FLEX, together with specifications and guide-lines on their operation.

These two routines should be written and checked carefully. At this stage a small program is typed into the machine, which will simulate the operation of FLEX, using the drivers that you have just entered. Using this program, you are able to test the operation of your disk and I/O drivers, by reading and writing sectors to and from the disk, the contents being displayed on your screen or terminal. Since this program which is provided amongst the FLEX documentation, simulates exactly the way in which FLEX will use user routines, once it is operating as specified, the driver routines should be saved onto cassette, rady to be re-loaded once FLEX has been read into the machine for the first time.

This is done by entering another short program (listing also provided) which uses part of the 'Read Sector' routine that you have already tested. This program reads FLEX from the disk into memory. Once FLEX is loaded, reload the disk and I/O routines from cassette. These are designed to overwrite specific parts of FLEX, thus customising it for a particular machine. The acid test occurs when using the system monitor, a jump is made to the 'Cold Start' address of FLEX, to see if all has been in vain! If all is operating correctly, the machine will ask for the date to be entered and then output the FLEX prompt '+++' Success! Be careful not to hit reset and loose all the good work – however the more composed user at this point uses FLEX to save a complete copy of itself back onto the disk, from where it can now be booted when required.

Before FLEX can be booted however, the boot loader program normally stored in the first sector of the disk must be prepared. This program uses the 'Read Sector' routine already used within the disk drivers. Additional code is provided to allow FLEX to be booted from any part of the system disk. Once entered into the machine the command 'PUTLDR' is used to store the program onto the disk.

The final action is to inform the boot loader program of the actual starting track and sector number of the customised version of FLEX. There is a simple command (called 'LINK') that carries out this task. Once completed, FLEX can be booted when required from this disk.

There are a number of other programs and routines which must also be configured for your system, before the implementation is complete. The most important of these is called 'NEWDISK'. It is with this program that blank disks are formatted so that FLEX can use them. The source code for this program is provided as a text file on the FLEX disk and all that need be added is a 'Write Track' routine. This closely resembles the 'Write Sector' routine that will have already been written and tested. Once this routine has been added to the source provided, using the FLEX editor, the complete text file can be assembled using the FLEX assembler. This file is then added to your system disk, along with the other FLEX command files.

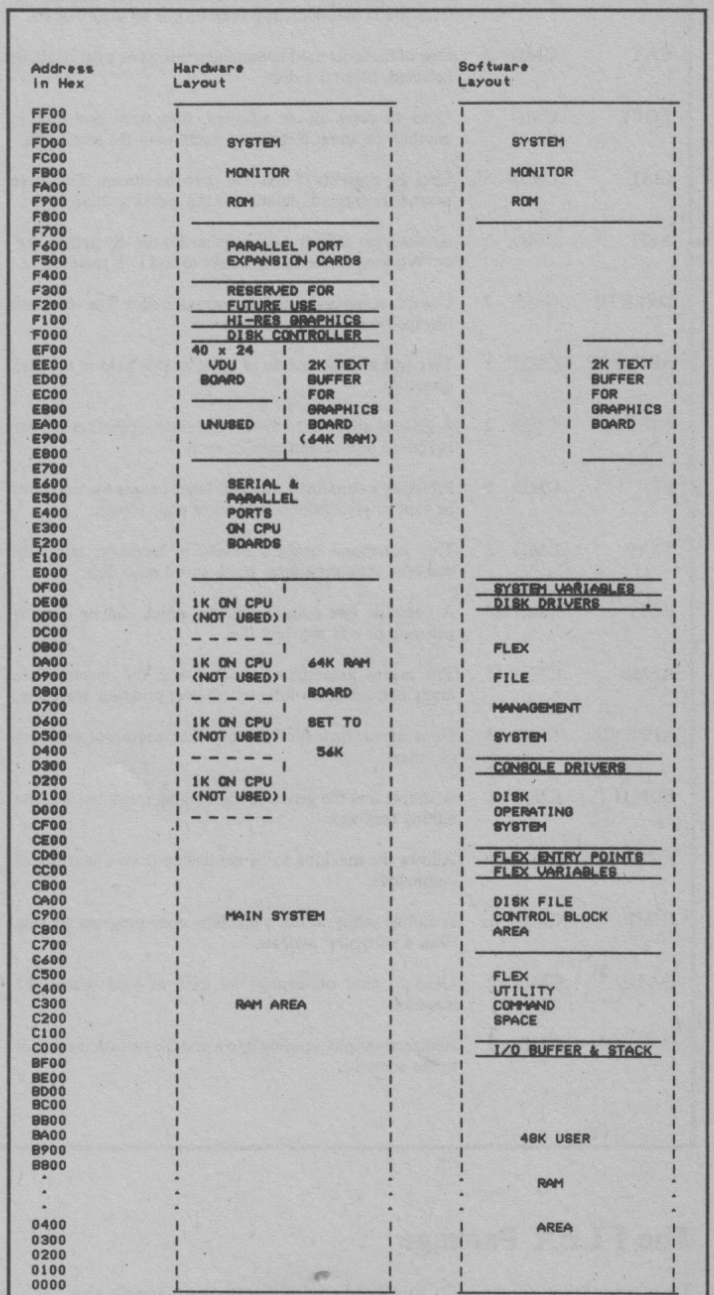


Figure 1. This shows how the *E&CM* hi-res computer is configured to run FLEX. The boards required are the 6809 CPU board, the 64K RAM board and the Disk Controller board. Some means of communicating with the system is also required.

The Contents Of A Flex System Disk

In order to provide more understanding of this operating system, below is listed all the files that are provided on the system disk, together with information as to their use:

Name	Type	Size	Description
FLEX	.COR	22	This is the main 'core' of FLEX, and it is the program to which the dirver routines already discussed, are added.
ERRORS	.SYS	9	A system file, holding a table of error code summaries. This file is automatically searched, if an error occurs.
CAT	.CMD	3	One of the most used commands, this gives a list of all, or selected, files on a disk.
COPY	.CMD	5	Used to copy all, or selected, files from one disk to another, or under a different name onto the same disk.
LIST	.CMD	3	Lists the contents of text files onto the screen. If no drive number is entered, defaults to the working drive.
ASN	.CMD	1	Amends the default drive number for the 'System Drive' or 'Working Drive'. Can be set to 'A11' if required.
DELETE	.CMD	2	Use this command to delete unwanted files. The complete file specification is required.
RENAME	.CMD	1	This will alter the name of a file as it is held in the disk directory.
TTYSET	.CMD	2	A general utility used to set the default operation of your keyboard and screen combination.
P	.CMD	1	Prefixing a command with this letter causes the output to be sent to your printer instead of your screen.
SAVE	.CMD	2	This command enables blocks of memory, normally machine code programs, to be saved onto disk.
EDIT	.CMD	28	A versatile line editor program, which can be used to generate or edit any text file.
ASMB	.CMD	48	The macro assembler provided with the system. The many options allow full control over program assembly.
APPEND	.CMD	3	Generates a single file from the concatenation of a number of others.
BUILD	.CMD	1	A simple text file generator command; quick but without editing facilities.
EXEC	.CMD	1	Allows the machine to be controlled from a text file of commands.
JUMP	.CMD	1	A simple utility to run a machine code program starting from a particular address.
DATE	.CMD	2	Used to alter or display the date as held within the machine.
O	.CMD	2	Redirects output, sending it to a text file on disk instead of to the screen.
LINK	.CMD	1	This program is used to set the boot loader program to the start address of FLEX.SYS, wherever it is on the disk.
VERSION	.CMD	1	Used to display the version number which can be encoded into any program.
PROT	.CMD	1	Enables particular files to be protected from deletion, modification or appearance in disk catalog.
VERIFY	.CMD	1	Controls the 'Read after Write' – Verify, operation of the disk drives.
PRINT	.CMD	2	Used to initiate the printer spooling system – if implemented.
QCHECK	.CMD	4	Controls the files in the print queue waiting for printer spooling.
I	.CMD	1	Commands a program to take input from a text file and not the Keyboard.
XOUT	.CMD	2	This command deletes all files which have been used for printer spooling.
SAVE	.LOW	2	A special version of the 'Save' command, used to save programs occupying the same memory area as 'SAVE.CMD'.
PUTLDR	.CMD	1	Used when first implementing FLEX, to store the boot loader program onto the first sector of the disk.
NEWDISK	.TXT	71	The source code of the 'NEWDISK' program, to be modified with a 'Write Track' routine for the target machine.
LOADER	.TXT	9	The source code of the 'Boot Loader' program, once modified with a 'Read Sector' routine.
PRINTSYS	.TXT	4	The source code of a sample parallel printer driver routine for a 'Centronics' type printer interface.

The following files are not to be found on the FLEX system disk initially, but will have to be provided in order to support a full system. The size of these files may vary from the figures gives, since this will depend upon the amount of extra code added to the original text files.

FLEX	.SYS	25	This is the final operating version of FLEX, combining FLEX.COR, with the new disk and I/O driver routines.
NEWDISK	.CMD	9	This is the assembled version of the 'NEWDISK.TXT' file, with the custom 'Write Track' routine included.
PRINT	.SYS	1	This is the assembled version of the 'PRINTSYS.TXT' file, once modified with the PIA address.
STARTUP	.TXT	1	This file is not mandatory, but if provided, it will operate as an 'EXEC' file on system startup.

This is the basic list of files that make up the FLEX operating system. There is a wide range of extra utilities and programming aids available, all of which can be stored on the system disk as command files and selected when desired.

The FLEX Package

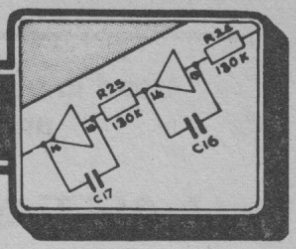
This operating system is available in a form that eases the task of configuring it to run on a non-standard 6809 system. Within the package there are two identical disks, either 5¼" or 8", depending on the size of the disk drives that you wish to use initially. The information provided is identical on either size of disk. Two disks are provided to give you a second chance should you have an accident with one of them! The documentation provided covers the following aspects, in a manual split into 5 parts:

Adaptation Guide – Full instructions and sample listings of programs and routines required to implement FLEX on a 6809 machine.

User's Manual – Full description and reference manual for the operation of the available system commands and utilities.

Text Editor – A complete description of the operation of the content-orientated editing system.

Mnemonic Assembler – Covering the operation and control of the extensive Macro-assembler.



Programmer's Manual – Providing the programmer with the information required to use the available system routines and functions.

FLEX And The Hi-Res Graphics Computer

Figure 1 shows how the HI-RES graphics system should be configured to run FLEX. The main boards required are the 6809 CPU board, 64K RAM board and the Disk Controller board. Some means of communicating with the system is also required. The options are either a serial terminal connected to the ACIA on the CPU board using the S-BUG monitor, the HI-RES GRAPHICS board with keyboard and HI-BUG monitor, or the 40 x 24 VDU board with keyboard and SM-BUG monitor. Due to the design of the disk controller board, users with S-BUG will be unable to boot the disk directly. A new version of S-BUG, called SP-BUG is available. This has the correct disk boot routines. The system will be at its optimum performance if the 4MHz crystal on the CPU board is changed to 8MHz, and the links set to select a 2 micro-second cycle time. In fact this modification converts the system to run at 1MHz. Remember that with the crystal frequency doubled, the baud rate generator on the CPU board will also run at twice the normal speed– the cassette interface may not be too happy about this . . .

As already explained, there are two sets of routines required to link FLEX to any particular 6809 system. In the case of the Hi-Res Computer, the simpler of the two are the Console Driver routines. Since all the recommended Monitor programs for this system use an identical table of indirect jump addresses to routines within the monitor, we can use the same machine code routines to interface either S-Bug, SM-Bug, or the HI-Bug graphics monitor. Unfortunately the jump table in FLEX does not allow the use of indirect addressing. Therefore use has been made of a work area within FLEX between the addresses D370 (hex) and D3E5 (hex), the top of the jump table. Within this area indirect jump instructions are located, which will interface correctly with any of the respective monitor routines.

The Console Drivers

A listing of the assembly code required by FLEX to interface to the system monitor programs (Console Drivers) is available from our offices – please enclose a large SAE. In order to implement these routines just enter the hex codes under the column 'Hex', sequentially into memory starting at the hex location shown under the column 'Addr'. Enter either 1, 2 or 4 bytes as shown.

Some of these routines do not require support initially, and these have been pointed at either RTI, RTS, or fixed memory, so as to disable the particular function. These are marked thus '★'. Once entered and checked carefully against the listing, store these routines on cassette.

The Disk Driver Routines

These routines are rather more complicated than those used in the console driver routines, since there are no useful routines provided within the system monitor program that can be linked into. Therefore these routines have to be complete in themselves; making them much longer than the console drivers.

A listing of the disk driver routines is also available from *E& CM's* offices and these have been tested on both 5¼" and 8" drives with FLEX. They only support single sided, single density operation, since this is the easiest way to get the system up and running. Once operating, those extra 'Bells and Whistles' can be added as required. As before, enter the hex codes as listed under the column 'Hex', starting at the hex address locations under the column 'addr'. Once entered check carefully against the listing, and save on cassette.

From this point it is just a matter of following the manual as outlined above, since the two driver routines deal with most of the differences between systems. If other routines are required, these can be developed from those already listed.

Conclusion

Much has been said about the advantages of a user friendly OS on any system. Being a staunch advocate of FLEX, over other, more popular 'Control Programs for Microprocessors', I believe that a FLEX system running on a 56K 6809 system is one of the most powerful single-user systems available today. As one gains experience with such a system, so the true value of this operating system will become apparent.

E&CM