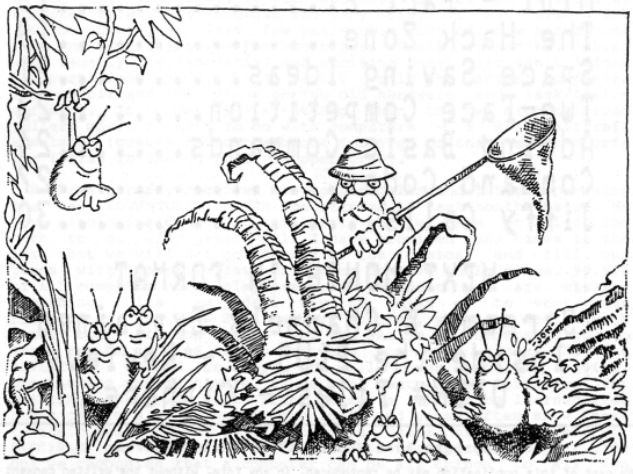
Vol 2 - No 3. October 1988.

FORMAT

THE MONTHLY MAGAZINE FOR DISCiPLE & PLUS D USERS



WE GO BUG HUNTING ON THE SPECTRUM

Vol 2 No 3. # CONTENTS Oct 1988.

NEXT MONTH IN FORMAT

Streams & Channels Explained
Inside the PLUS D - Reviews
OPEN# Files - Ciphers

FORMAT is published by INDUG. 34 Bourton Road, Gloucester, GL4 0LE, England. Telephone 0452-412572. **DISCiPLE** and **PLUS D** are trade marks of MILES GORDON TECHNOLOGY, Lakeside Technology Park, Phoenix Way, Swansea, South Wales, SA7 9EH. Telephone 0792-791100. The DISCiPLE is marketed by Rockfort Products, 81 Church Road, Hendon, London, NW4 4DP.

Well the postal strike is settled but it will take some time to get over the after effects. There still appears to be a large backlog in some areas and it looks like it will be at least the end of October before things are back to normal (or what they call normal for our appalling postal system).

I went to the PC Show at Earls Court in London (used to be called the PCW Show) in mid September. It was (as is usual at computer shows) Hot, Crowded and Very Noisy. It was also VERY BORING. Over the last few years this sort of show has become more and more expensive for exhibitors. Small companies, which are usually the innovative and exciting ones, just can't afford the sort of outlay required. Result - only the largest companies can exhibit and that means boring old hardware and lack-lustre software. It comes to something when the two biggest crowd pullers where nothing to do with computers - a REAL aircraft simulator (queues round the block for a go on that) and a sports car (prize in the Computer Shopper draw).

We have already received many phone calls and letters praising the new ADVENTURE CORNER we started in last months issue. We have also had a couple of complaints that we are printing too much to do with games. Well I have promised many times in the past that we will not go the way of the 'comics' and fill our pages with games reviews but games do have their place. 99.99% of ALL computer users play games, if you dont you are missing something, its an ideal way to relax or to vent your hostilities. The HACK ZONE is very popular, yes it deals with games most of the time, but the skills it encourages you to learn can be applied to any software not just games. As the ADVENTURE CORNER develops it will cover problem solving, programming and much more. If we are sent games programs for printing then we will print them, a lot can be learned from other people's programming techniques and games often show these to their best. FORMAT is for serious users of the Spectrum, DISCiPLE or PLUS D, we dont treat our readers as ten year olds but we recognise that a computer can be all things to all men and we must cover as broad a spectrum (no pun) as possible.

Next month will see a change to the YOUR LETTERS page. We will have a STAR LETTER spot with a prize of a £5 FORMAT Voucher to the writer of each months Star Letter. The voucher can be used as part payment of your next years subscription or to buy anything we sell through FORMAT. So keep those letters coming.

See you next month. **Bob Brenchley. Editor.**

NEWS ON 4

DONT BUY BLACK +2

There were very long faces on the TELECOM SOFT stand at the recent PC Show at Earls Court. They had just taken delivery of several Amstrad+2s to demo their new releases, only they couldn't.

Yes readers, Amstrad have finally done the dirty and releases the +2.5 - a +3 circuit board in a +2 case. It wont run a lot of 128k software, it wont work with the DISCiPLE / PLUS D, it wont even work with some simple interfaces. Be warned, the only way to tell the difference is the change of colour to a black casing. Remember we warned you about this in February.

SINCLAIR

If Sir Clive Sinclair was dead he would be turning in his grave after the Earls Court show. The man who gave his name to a range of computers which, love them or hate them, were always innovative, now has to stand by and watch ten year old technology appear under his logo.

I refer to the Sinclair Professional PC200, a cut-down, plasticy and bulky IBM clone launched at the show by Amstrad. When the QL was launched Uncle Clive was criticized for not giving it IBM compatibility. His answer was simple "I wanted to do something new, not rehash other peoples old ideas".

Amstrad's new machine (I wont call it a Sinclair) has 512k of memory, the old CGA graphics, a built in TV modulator for 40 column work, a keyboard that reminded me of the Commodore 64, and a single 3.5 inch disc drive. At £399 (with Mono monitor) its not really that cheap and at 18" x 13" x 4" it looks and feels rather bulky.

DODGY DISCS

MEMCON, a major US manufacturer of disc duplication and certification equipment, has just completed a survey of 25 brands of 3.5 inch disc. Its findings make interesting reading. The company bought quantities of each brand in ten different American cities. It then subjected them to tests laid down by the American National Standards Institute (ANSI).

Only four brands:- Sony; TDK; IBM and C Itoh passed with flying colours. Most others failed one or more tests even though their manufacturers claim 100% certification. Memcom president Jerry Korth said "It is a sad report, with 3.5 inch discs becoming very popular, quality simply hasn't kept up with quantity". He went on to say "There are strong reasons to doubt that discs are tested to ANSI standards despite what producers would have you believe".

The survey also found that there was no link between price and quality, in fact Sony, TDK and C Itoh were all close to the average price.

YOUR LETTERS

Dear Editor,

I would like to pass comment on the excellent article by Clyde Bish (Format V2 #1). While I agree with everything Clyde says in his article I do feel that it is incomplete without stating that all the space saving methods described will reduce the speed of any program using them. The reason for this is that when a number is entered in a basic line the editor places the floating point form of the number after the ASCII data. Then when the program is run the number is immediately available for use by the computer. If you use any of the space saving techniques then the number is not available and has to be calculated or fetched. The following program demonstrates this:

```
10 FOR N=1 TO 10000: NEXT N: STOP
20 FOR N=1 TO 10000: LET X=0.123: NEXT N: STOP
30 FOR N=1 TO 10000: LET X=VAL"0.123": NEXT N
40 LET Y=0.123: FOR N=1 TO 10000: LET X=Y: NEXT N
```

The timings I measured on a +2 in 48K mode were:

LINE #	TOTAL TIME	ASSIGN TIME	EFFICIENCY
10	41 secs	0 secs	----
20	61 secs	20 secs	100%
30	237 secs	196 secs	10.2%
40+	70 secs	29 secs	68.9%
40*	117 secs	76 secs	26.3%

\+ no other vars declared * 52 other vars declared

Although this in no way detracts from the article it could be misconstrued by the novice that space saving is more important than program speed. I feel the article should have pointed out that the techniques described should only be used when space is short and then used with discretion, as there is a speed penalty for each compaction. I look forward to more well presented articles such as Clyde's in future issues.

Yours, Paul Conway.

Dear Editor,

It may interest PLUS D users to know that you can save space on disc when saving old 16k games by doing the following - POKE @299,64: POKE @2330,64. This alters the 48k Snapshot to 16k (33 sectors). To return to normal - POKE @299,192: POKE @2330,192.

Yours Sincerely, Ian Spencer.

Letters printed may sometimes be edited for length or clarity.

Your letters are needed for this page. You can write on any subject you think may be of interest to other readers. Keep them short and to the point. Go on, get writing..

WORDMASTER REVIEW

By: John Wase.

Although more than a mere word processor (having been designed to extend to a complete Desk Top Publishing outfit), Wordmaster is sold on its own, can be used on its own and is eminently usable on its own. As a mail order purchase, it arrives either as a cassette with instructions to transfer to a disc, or on a disc (at a reasonable extra charge) direct from PCG (note the new name for CARDEX), who have sensibly arranged their code with returns to Basic for Loads and Saves. This means that any disc system (or a microdrive, for that matter) is easily catered for with the simple instructions provided. I transferred all the files (including those for DTP) to a PLUS D disc, and for good measure, to a Discovery disc as well. Once Wordmaster itself is transferred (good, straightforward instructions), any other files are shifted through the program menu by loading each file from tape and then selecting a save to disc, so you certainly don't need a header reader or anything complicated; the program does it all for you. Beware, though: transferring the complete DTP suite (including coffee and mistakes) takes around an hour and a half - it's probably well worth buying their disc version.

So, it's all there on the disc. What you've got is a stand alone wordprocessor which runs on the Spectrum in 48K mode, yet still has room for 15 or so pages of double-spaced A4 text. Load it. A cyan screen appears with a prompts section entitled "File handling" in the bottom three lines. Before you can type in anything, you must press "C" to create a file - in other words to give what you are going to type a name! Beware, though, it's all too easy to bungle the initial keystrokes and call it nothing! The program will accept this, but try recalling it after saving several other files to memory..... As soon as you have named your file, you are into the "Options" menu: the cyan screen changes to white, and prompts include saving the file to disc or tape, renaming the file, P for the print menu, alter column width, word, character and free-space count, Q to quit back to "File handling" (which leaves your file intact, but in memory, enabling you to work on several documents at once) and "Enter" to enter the file, get the cursor and start typing. But I'm going too fast; there are still several other commands in the "File Handling" menu that I've not yet mentioned. For instance, you can load a file from tape or disc (D toggles drives 1, 2 or tape, K overwrites screen with a disc catalogue. Q with a "catalogue" of files in memory: you can erase files from disc, delete from memory, join files in memory or save the complete contents of memory to disc as one file).

Back to the file we've created or loaded: "Enter" and we're away. There are surprisingly few prompts in this menu: you've got minimal hints and then you have to guess or refer to the

handbook. Extended mode toggles insert/overwrite modes (it gets confused, however, when you use it in the text, for instance to insert square brackets which invariably overwrite the text into which they're being inserted). Fast scroll (8 lines at a time) is toggled by symbol shift/q, and there are facilities for deleting/undeleting words (useful for swapping odd bits round). There are also block move/delete/save/copy facilities and search/replace (quite good; it can be set for case or ignores it, and will ignore punctuation, etc.). Underline shows on screen with symbol shift/i, all the other special control codes have to be put in either as embedded control codes (using the graphics key) or in a "Command line". These are great: true video inserts a new,yellow line beginning with a "greater than" sign followed by the cursor, and you can then write real English commands like "margin 8", "fill" (this right-justifies the printed page by padding each line with spaces) or "L" (for LPRINT) followed by the appropriate decimal codes. The command line is not, of course, printed; it just sends special instructions. The embedded codes are also great, allowing you easily to print H_2SO_4 without loss of justification: a current limitation is that they will accept only three numbers (though they tell me they're working on this).

Although cursor movements are otherwise pretty limited, there is the facility to jump to any page. Headers, footers, set page length, page numbering and form-feeds are available, and lots of special effects are enabled by inserting command lines in a header. The tab command works only with the printer and is likely to ruin justification: because of this and because of the way the thing is not WYSIWYG, tabular work is difficult with tables of up to 64 characters across and almost impossible with longer lines. although the printer can quite happily be instructed to print lines as long as 255 characters, and ordinary text will equally happily be printed thus.

There are one or two special features. For instance, the options menu contains an ASCII command that will strip out all non-standard codes from textfiles, so that any code file can be read in. Beware, though: I tested it with Tasword+2 and The Last Word and found that they must be unjustified first (i.e. ragged right edge), or all sorts of odd spaces appeared in the text and were very difficult to remove subsequently. Pictures are also accepted as a SCREEN$: you can then cut out a section, flow the text round it and save it along with the text: compression techniques ensure minimal amounts of code and prevent your filling the disc with screendump information. Finally, a helpline is available from PCG; invaluable support when you get stuck!

I found only one bug, and this only in the DISCiPLE/PLUS D version (aren't you lucky, Discovery users). If you "break" during printing, you get dropped into Basic and lose your text file (yes! I lost all this). So save it first. If you didn't (like me), you can recover with RANDOMIZE USR 59371: name the file anything you like at the prompt and back will come your textfile with the new name (bless you, helpline).

The instruction book is very small, and I found it a little

difficult to follow, particularly at the start, although once you get to know the program, it's logical as a reference. I also did not like some of Wordmaster's 64-column characters and found the screen scrolling rather slow and the cursor keys over sensitive for my personal taste. These, however, are minor gripes. Wordmaster is a simple, straightforward program without data or mail-merge facilities, macros or conditional printing: nor is there at present any provision for spell-checking or thesaurus. Many users will not really want these facilities, and apart from the one or two odd quirks in tabular work and long embedded control codes, they will find that Wordmaster is particularly easy to use, enables a printer to be driven very easily, and, most of all, offers excellent value. I will bring you a review of the full desktop publishing package in the near future.

Wordmaster is £11.90, PLUS D/DISCiPLE 3.5" disc £1.50 extra, from PCG, 3 Barton Street, Barrow in Furness, Cumbria, LA14 2EP.

- + - + - + -

THE SAM REPORT

By: Bob Brenchley.

Each month from now on I will try to bring you the latest news (and some of the inner secrets) on MGT's new computer SAM.

WHAT IS SAM?

Well, some of the finer details are still only in the mind of its creator Bruce Gordon. SAM is the culmination of many years work which, along the way, also produced the Gordon Microframe; the DISCiPLE and the PLUS D. Bruce has, without doubt, produced some of the best add-ons the Spectrum ever saw, but like many people in the industry his long held dream has been to build his own computer. SAM is the result of this dream, and what a dream machine its going to be.

I first heard of SAM in late 1987, at that time it was going to be a 64k or 128k machine with two screen modes, 16k of ROM and a few extras. A year later it boasts 256k of RAM (expandable to 512k), 32k of ROM, Z80B processor (the faster brother of the one in your Speccy, 4 screen modes, a palette of 64 colours, a super sound system, MIDI interface (in and out), DADI interface, RGB/Composite and TV output. All for less than £150. All 256k of on-board memory will be usable in a sensible way - none of the hideous page layouts of the +2 and +3.

Next month I hope to bring you details of the screen layouts. In the mean time I recommend you get on MGT's SAM mailing list so you get to the chance of being one of the first owners of this new wonder machine. Write to them at the address on page 2.

To Be Continued.....

BUG HUNTING

For Spectrum Programmers

By: Ken Elston.

Bugs and Errors, the bane of a programmers life. They say "There is ALWAYS ONE MORE BUG in a program" and I've never found a truer saying. So come, read on, I will try to show you how to track down those elusive bugs. This article is written with 48k programmers in mind, but most applies to 128k owners as well.

The first error encountered by most Spectrum users is the flashing question mark which indicates a syntax error in the line you are trying to enter. Experience, and the manual, soon show that it is caused usually by typing in a keyword letter-by-letter, or by bad punctuation, for example omitting a semicolon or left out a set of quotes ("). The marker is set to the end of the part of the line the Spectrum has understood. So normally (but not always) the error is after the flashing question mark. Look along the line, is there a mismatch of brackets or quotes? Try moving the edit cursor along the line, does the cursor jump over each keyword? If not then you have typed out the word instead of using the single key entry system.

The most frequently occurring error code is - 2 Variable not found. When you create a variable - lets say by **LET A=10** - then the Spectrum will know what you are taking about whenever it finds the variable 'A' in your program. Error code 2 results when the computer reaches a variable in the program that has, so far, not been given a value. After an error report are two numbers separated by a colon (ie 120:2) these give the line and statement number which produced the error. Check the line, lets say it reads **120 LET A=10:LET B=A*T1** well the error is being reported in statement 2 and as 'A' has been set in the first statement it must be the variable 'T1' that can't be found. Look back through the listing, if you can not find a statement giving 'T1' its initial value then add a line setting it a value you think is right (you can always come back later and alter it if you need to). Also check that you haven't mistyped something, the number 1 and the letter I (or a zero and the letter O) are very easy to confuse.

Remember that the mistake may not be at a lower line number, with GOTOs and GOSUBs in a program it could be almost anywhere so follow the program through until you find it.

Although the majority of error codes are explained adequately in the manual, the report "B-Integer out of range" can be confusing. An integer is a whole number - 1 is an integer, 1.5 and 3.75647 are not. The error occurs most frequently when you try to print something beyond the limits of the screen. PRINT AT 0,31;"a" is acceptable and will print a letter "a" at the top right of the screen. PRINT AT 0,32;"a" would not be

possible as its off the edge of the screen. The integer 32 would be out of range, resulting in error code "B". The error is more difficult to detect if variables have been used as co-ordinates and your character, or series of characters, is being printed in varying positions. If the instruction PRINT AT 0,x;"a" produces report code B, make sure that the value of x never increases beyond 31. Integer out of range can also be found when you are POKEing a number into memory. The biggest number which can be POKEd is 255. In this case the error code will occur in the statement containing the POKE statement. In many cases though, the error will have occurred in a DATA statement, somewhere else in the program, which is used to hold the values you are trying to POKE into memory. Have you left out a comma? has 201,16 become the large number 20116? Negative numbers are also 'out of range' so did you put a minus sign in by mistake?

Another very frequent error code is "E - Out of Data". That will occur in a line containing a READ statement. The error though, will have occurred in one of the program DATA lines, which may be nowhere near the READ line. A READ command sends the computer to a DATA line to collect the next piece of DATA contained there. This is often done within a FOR-NEXT loop, especially when graphics are being set up. the line - FOR I=1 TO 8: READ N: NEXT I - will send the computer to the DATA lines eight times, for eight separate pieces of information. If there are only seven pieces of DATA there it will return to the READ line and product the OUT OF DATA error. When there are several DATA lines they will all have to be checked, because the piece of DATA you have omitted was not necessarily the last.

In some cases the computer will follow the program correctly, without producing an error code, but from the programmers point of view the program contains a BUG - it ain't doing what you want it to do. In that case RUN the program and BREAK into it at the point where it appears to go wrong. That will produce report code '9 STOP statement' and the line/statement on which you have STOPed the program. This method makes it easy to locate the rough area of the program which may contain the problem. Now PRINT some of the variables, look for values outside the range you think they should be, print the value of the control variable in a FOR-NEXT loop so you can see how far it got before you pressed BREAK. If your program crashes before you can stop it then add STOP statements into the basic just before the point at which you think it goes wrong. Keep looking, keep PRINTing.

Programs which you copy from magazines, books or from friends can be difficult to de-bug because they may contain programming techniques which you have not yet learned, or simply because it is often difficult to follow another programmer's logic. The flashing '?' indicating a syntax error may appear frequently. In that case check carefully what you have copied. You may not have recognised words such as TO or THEN as keywords. Make sure that when the "is not equal" sign '<>' appears in a listing you always enter it as one character and not as "is less than" '<' followed by "is greater than" '>'. Most problems are caused by mistyping so double check each line, or better still get a friend to go through it (he may find the error you missed).

Sometimes a program listing in a book or magazine will contain what seems to be a very obvious error. If it contains key words or symbols which are not on your computer, check that it is intended for your machine, you can not type a BBC program into the Spectrum no matter how good the program is. If a program contains the command GOTO or GOSUB to a non-existent line number then the computer will simply go to the next numbered line after that one. That is a sign that a program has been developed and improved and is rarely an error. When you have errors in a program, first check the report codes listed in Appendix B of the manual. It may then be necessary to read the appropriate section of the manual.

Always type in a program exactly as it is listed, dont make changes (however good you think they may be) until after you have tested the program as the author intended. Once the program is working you can make any alterations you like and when things go wrong you can confine your search to the changes you made.

Again - remember that the error is not necessarily on the line which produces the report code; that is simply the line where the computer meets the problem for the first time. Always check carefully every line connected with the line containing the error code and the mistake should be easy to locate.

A few other points to keep in mind when writing programs:-

1) Always make regular saves while entering your program. Dont RUN the program until it has been saved just in case you get that fatal crash.

2) When saving your 'improved' version of a program DONT overwrite your old version - you may want to go back to it at a later date.

3) Put lots of REM statements in. Six months later you wont understand your own coding unless you do.

4) Split you programs up into small sections and test each on their own. Much easier than testing the whole program.

Finally, hardware problems may also cause errors in programs. Any alteration to the power supply can cause a program to CRASH. In that case the screen display may change dramatically the keyboard will have no effect. The only solution is to unplug your computer or reset it and begin again, making sure that your power supply and interfaces are connected firmly. Clean up the edge connector with a hard (ink) rubber, dont use a chemical cleaner as you may damage the circuit board.

I hope the above will help you the next time you find the odd bug in a program. I will be back in the near future with another article to help you improve your basic programming skills. If there are any problems I can help with then please drop me a line c/o FORMAT. I can not give a personal reply but I will try to help you out in a future issue. Thank you for reading.

Back soon.....

By:- Paul Rigby.

Having launched a new column devoted to adventures, Format is not about to assume that every reader out there is an expert on the subject. Therefore, this month, I will be concentrating on what adventures actually are and how you play them. From the basic principles to a few examples of the many features, good and bad, that can be found within adventures of the past and the present.

All adventures have basic similarities. For example, you, the player, are thrust into a fictional world or situation and are given a task or quest to complete, upon which the game ends. In the meantime a number of puzzles are presented to the player which are designed to hinder, educate and hopefully entertain the player. Interweaved throughout all of this is the plot. Part of the skill of the adventure author in this situation is to maintain the integrity of the storyline while still presenting thoughtful and logical puzzles of a varied nature.

The game world consists of a number of "locations". These locations can be rooms, parts of a landscape such as fields and mountain areas, a mixture of the two, and so on. They can be situated on earth in the present day, on an alien spaceship in the future, in a haunted house during the Victorian era or wherever the adventure author decides to base the story. Movement is possible between locations by moving in a certain direction to the desired location.

To illustrate this structure and the method of movement within an adventure consider the timeless story of a Knight in shining armour who is about to attempt to rescue his beloved Princess. Unfortunately, the Princess is imprisoned in, for the sake of simplicity, a small garden shed which lies amongst a cabbage patch. Between them lies a large river. Here we have four locations. The Knight, which is the character that you are controlling, sits on his horse amongst some bare grassland. Thus our first location is the "bare grassland". The "river" is the second location, the "cabbage patch" third and the "garden shed" is the fourth, and final, location. Each location will, when entered, give the player a description of the terrain, any objects to be found within the location, the directions that the player can move to and any additional information which will probably be superfluous to completing the game but will build the much needed atmosphere.

Movement, in adventures, follow compass directions with the imaginary compass laying flat on the ground. In the above example the river lies directly in front of the Knight so to move to the river location the player types "Go North" or

"North" or just simply "N". If the river lies to the left of the Knight the player would type "Go West" or "West" or just "W". There are a number of permutations of commands when moving in different directions. The latter are just three examples.

However, the universal method of movement is to type in the first letter of the direction you wish to go. It is, obviously, a much quicker method too. While some adventures use the four cardinal points for directional movement many have the additional options of moving South-West (SW), North-East (NE) and so on. In addition many adventures attempt to introduce a three dimensional aspect to the adventure by allowing the player to move Up (U) and Down (D). However, just because an adventure says that you can only move in the four cardinal directions try the other directions just in case! Similarly, a frequent ploy, which mainly occurs in some of the older adventures, is to provide a brief locational description and then only give, for example, two of the possible three directions that are available for movement! The reasons for using this rather unseemly tactic are to make the adventure unnecessarily difficult, which smacks of a lack of imagination on the author's part, or because the adventure was badly designed and playtested, or, more rarely, to reflect problems presented by the game environment.

When moving from one location to another, whatever the adventure, you will have to face the consequences. For example, our Knight has just been instructed to go north, to which lies the river. On his arrival he and his horse may be swept away by the current to die a watery death or his horse may be an excellent swimmer or a sea-monster, or rather, a river-monster may arise to gobble horse and rider in one mighty gulp. A good adventure will give a subtle hint that all is not well within the river if danger lies within. Many poor adventures liberally sprinkle "sudden-deaths" in numerous areas of the adventure. This leads to frustration on behalf of the player who has to save the game position before entering a new location for fear of, yet again, being done away with. There is always the possibility of not being able to enter the river at all until an alternative means of crossing is found. However, let us assume that our horse is treading water in the centre of the river. To proceed to the cabbage patch, which again lies straight ahead of us, the player types "N", or similar. If for some reason that is not possible then sit and think for a second because here we see another aspect of adventuring. The appreciation of the game environment. As we are sitting on a horse which is treading water in the centre of a river the player may try "Swim north" which is specific to the situation in hand.

Upon the arrival at the cabbage patch, the player is given a description of the location. In the location can be seen a golden cabbage growing quite happily amongst the rest of the crop. Obviously such an object will be important and extremely useful. So you immediately attempt to GET the prize cabbage. Here we see another side to adventuring - the infamous parser and its associated database of vocabulary.

Firstly, the game might deny any knowledge of the cabbage's existence by replying; " I cannot see a cabbage." The player

must try again from a different angle before giving up on the cabbage and deciding that the it will play no part in the game. For example, the game might give a favourable reply if you attempt to GET the GOLDEN cabbage. Again this fault will generally occur in older games but be aware that it can occur at any time. The reason for the initial negative response could be the result of an adventure author's very small vocabulary database. The program only responds to the "golden cabbage" but not "cabbage". If two or more cabbages are used in the game as objects then the program should ask which cabbage you wish to GET.

Secondly, the word GET, one of the most used words in adventure games, may not be accepted. Other similies must be tried such as TAKE. Other situations may occur where the word you type in is not accepted so the player must try other, similar words. Another example may be the lack of response to the popular EXAMINE command. The player should try alternatives when objects are present within a location such as LOOK or SEARCH. Many adventures will, after an object has been examined, give extra information if you then search it. For example, a bed may be EXAMINEd without much valuable information being obtained. SEARCHing the bed may result in the blankets being removed to reveal a further object. In addition if you LOOK under the bed more information may be revealed. Of course the adventure may be so basic that none of the latter work. Unfortunately, a great deal of atmosphere is lost when this facility is neglected to the degredation of the entire game.

I will continue with the parser and the vocabulary next month in Part 2 of this feature. In addition to which I will be catching Red Herrings, solving Puzzles and rescuing that Princess.

MIDI

BEGINNING MIDI PART 2.

A practical look at setting up a MIDI system

By: Ray Elder.

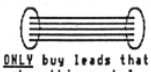
CONNECTIONS.

The whole system depends on the MIDI sockets that connect the computer interface and the instruments. These are five pin DIN sockets and need suitable leads fitted with five pin DIN plugs. The common Hi-Fi leads can be used but in a multi-instrument set up then they can cause problems, it is better to spend the extra cash and buy leads designed for the purpose. These are higher quality and will eliminate any chance of connection problems.

However a simple single or 2/3 instrument set up will probably work without problems using the cheaper leads. Should you purchase this type then be careful as they can be wired in a mirror image - do not buy them if they have the diagram shown in fig 1, make certain that they are the straight line wired type which usually display the diagram shown in fig 2.

DO NOT buy leads that show this symbol.

FIG 1.



ONLY buy leads that show this symbol.

FIG 2.

There are at least two sockets on a midi instrument and the better ones have three. Having said this, Yamaha only fit two but provide an internal system that 'echoes back' the signal thereby making the midi out double as a midi through. These sockets are:

MIDI IN. - This is the socket that accepts midi data from another instrument or computer, it is usually connected to the computer or instrument's midi out or midi through socket.

MIDI OUT. - This socket sends the midi data that is generated by the instrument or computer to which it is fitted. It does not (except in the case of Yamaha instruments) pass on signals from other instruments in the system.

MIDI THROUGH (or THRU) - This socket passes on all the data that comes into the midi IN socket to the the next instrument in the system.

I have included two diagrams (figs 3 & 4) to try and help clarify this. Fig 3 is an example of the most basic of

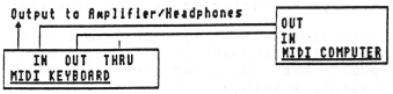
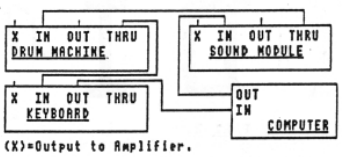


FIG 3. MINIMUM MIDI SYSTEM.



(X)=Output to Amplifier.

FIG 4. A MORE VERSATILE SYSTEM.

systems - a computer linked to a single instrument - and fig 4 is an example of a more sophisticated system - one very similar to mine - and is still a simple set up in comparison with those found in many recording studios!

The problem with all midi systems is that there is only one IN socket on the recieving device, and when this is a computer it can be rather limiting. I always use the keyboard when recording sequences, but find it best to use the drum machine to control the tempo on playback. I also like to make dumps of system exclusive data (this is drum patterns, synthesizer programs specific to each instrument) for tape storage via the computer.

You cannot simply wire two or more midi leads together, the receiver becomes terribly confused, and so it often means a lot of lead swapping with of course, the usual wear and tear on leads, plugs and sockets. The solution is to buy one of the many switch boxes on the market, I include one supplier's name & address at the end. If you have more than one instrument then I strongly advise buying one.

TERMINOLOGY.

In this chapter I also include a glossary of the terms that are related to the music and technology we are using, this is to save having to explain each one every time I use it.

During the series I will be making references to using the MICON system and software because it is the one I use almost exclusively. This is due to personal working preference and in no way decries the quality of the other midi interfaces and software. Although a few things are specific to this system, most of the techniques and suggestions are easily used on any of the others on the market. Certainly one advantage of the XRI system and software is that it will work with microdrives and hence the disc system - the other units do not.

GLOSSARY

SEQUENCE - A series of notes or MIDI events that is stored on the computer and sent back to the instrument(s) on replay.

REAL TIME - A method of recording MIDI data as it happens, ie. playing a tune on a keyboard and recording it as you would on a tape recorder. This is a fast way of working but errors have to be corrected by replaying the whole bit of music.

STEP TIME - Entering MIDI data item by item, ie. typing each note pitch and length one at a time. A slower method of entering music but more accurate and editing of each item is usually possible.

EDITOR - This is usually used to refer to a program which allows you to use the computer to amend or create the sounds patches in a synthesiser. They have to be designed specifically for the instrument you own and make sound synthesis much easier by displaying many parameters, often in graphical form, at once.

PATCH - The combination of waveforms, oscillators, envelopes, etc. settings that make up the sound of a specific voice. Often referred to as the 'voice' itself especially with keyboards that do not provide synthesis programming.

VOICE - Each individual note that can be played is a voice. Often confused with a 'patch'. For instance a CZ101 may be used with four voices simultaneously but limited to one patch, or alternatively, four single voices each set to a different patch.

INSTRUMENT - An individual musical unit. Yamaha refer to their FB01 as having a maximum of eight instruments, in this mode each has one voice and each can be set to a different patch.

CHANNEL - There are sixteen MIDI channels 1 - 16 (on some instruments they may be numbered 0 - 15). When an instrument is set to a channel in Poly mode it will only play the data on that channel and ignore anything on any other channel. All data including the data on the channel the instrument is set to is passed on out of the THRU socket.

TRACK - Similar to a tape track, this refers to a sequence of data recorded on the computer or sequencer and can usually be set to a specific MIDI channel. The XRI real time sequencer can record up to eight tracks and each track can have as many notes as you wish (eg. chords). Their Step Time sequencer also has eight tracks, but each track can only play one note at a time.

QUANTISE - A method of auto correcting a piece of music. If you play slightly out of time then notes can be moved to begin exactly on the beat. Different levels of quantisation are usually provided to allow for time variations. Too much quantisation tends to produce a rather mechanical effect.

SYNC - Short for syncronisation, this keeps all the instruments playing in time, there is usually two main options:

1. Internal sync - the instrument produces its own MIDI time pulses for other instruments to work with,
2. External sync - the instrument takes its time signals from another instrument.

These are usually needed when using a drum unit. I tend to record using the computers internal sync, switch leads and then play back using the drum machines time data. The older drum machines used a different type of sync data but the XRI Micon unit has a socket to allow these to work.

MONO - Has two meanings: When referring to an instrument it means it can play one note, and when used to refer to MIDI means the instrument can split its voices between several channels.

POLY - Also means two things: When referring to an instrument means it can play several notes at once, when referring to MIDI means that you can select which channel the instrument will receive data on.

OMNI - In this mode the instrument will receive and play data on all the MIDI channels. Some cheaper instruments will only operate in this mode and they are of very limited use.

EVENTS - All MIDI data is referred to as an event, each event is one single command such as note on, note pitch, note off.

GATE - With the XRI system the gate represents the percentage of time the note is played to its full value, this is useful in playing staccato and legato notes.

Finally this month I would like to request for any other midi users to send in suggestions and hints & tips to share with us, anyone who has any queries or problems please write to me and I'll do my best to help, and anyone who has any music and would like to submit it I would be pleased to listen and perhaps we could make a compilation tape for distribution through our own PD supply. Meanwhile I'll remind you that my own tape is available for those interested, for £3.95. This is an audio tape and you do not need a MIDI system to listen to it.

Please send all contributions directly to me at 1 Periton Court, Parkhouse Rd. Minehead, Somerset, TA24 8AE.

HACK-ZONE

By: Hugh J. McLenaghan.

I'm sorry that this article on the conversion of Karnov was delayed, but we decided that the way I had planned was going to be too long. I will describe here how the conversion was done and the problems encountered. If you wish the conversion, then send a S.A.E. and a disc and how you wish it saved, i.e. 40/80 tracks SD/DD and SS/DS. The programs and the instructions on the conversion will be sent A.S.A.P.

Now for the method:-

After MERGEing the BASIC loader the following lines were of use:-.

```
10 BORDER 0: INK 0: PAPER 0: CLEAR 32767
20 LOAD "" CODE 65088
30 RANDOMIZE USR 65088
51 SAVE "load" CODE 32768,448
```

LINE 10 gives us the CLEAR address, line 20 is the load address, line 30 gives us the CALL address and line 51 tells us the length

After loading and disassembling the code I noticed that the first 13 commands were doing something special:-

```
65088        LD HL,65107 ;Start address
65091        LD BC,384   ;Number of bytes
65094        DI          ;Disable interrupts
65095        XOR A       ;Let a=0
65096        LD R,A      ;Let refresh register=0
65098 LOOP:  LD A,R      ;Let a=refresh
65100        XOR (HL)    ;Xor a with PEEK HL
65101        LD (HL),A   ;POKE HL,A
65102        INC HL      ;Let HL=HL+1
65103        DEC BC      ;Let BC=BC-1
65104        LD A,B      ;IF BC>0
65105        OR C        ;THEN
65106        JR NZ,LOOP  ;GOTO LOOP
```

This is a decoding routine, the bytes from 65107 to 65490 are XORed one at a time with the refresh register. This method is a simple one, but it is used in many games. After I decoded the loader I then disassembled it, from this the load address (32512) and the length (17152) of the main code were found.

The next thing I did was to look for a level loading routine in the main block of code, after quite a long search I found the routine at 33534. At a first glance it looked very complex, but

after examining it closely I found it very easy to understand.

What it does first is it checks a byte in R.A.M. which holds the level number that the computer wants to load, it then compares this number against another byte in R.A.M. which holds the level loaded in the computer if the numbers are the same then the computer returns to the main code, otherwise it continues. The computer then loads the header of the level from tape and compares this against the level that it wants if it is not the right one then it repeats the above.

The screen is then loaded in a compressed form and expanded. Next the load address is set to 30462 and the length to 2306 to loads the next block of code. This block of code has not yet finished loading, this is because of its clever loading system. The start is now set to 49152 and the length set to the contents of 30462 and the loading is continued. Therefore the computer has loaded two blocks of code while you think that its just one block. Here is a shorter explanation of whats going on:-

```
-> If level wanted=level loaded then return
-> Load header of next level
-> If not level wanted then repeat above
-> Load compressed screen then expand it
-> Set load address to 30462
-> Set length to 2306
-> Start loading
-> Set load address to 49152
-> Set length to contents of 30462
-> Finish loading
-> If level loaded<>10 then return
-> Load final screen
-> Return
```

All of the above happens without you seeing it.

After I discovered the above the rest of the conversion was quite easy to do. The hardest bit is finding places to put the new loading routines, this is because the DISCiPLE loading commands are larger in length than the normal loading routines.

When you get the conversion disc you need to format a blank disc, then save a system file onto it. You then boot up the conversion disc by typing LOAD d*"KarnConv" After doing this you insert your blank disc and press ENTER. Then insert the conversion disc and press ENTER, then the program will tell you what to do. I hope that you enjoy your converted Karnov!

If you wish a better explanation then send me a large S.A.E. and I will send you notes and more information.

I would like to hear from anyone who has ideas for the HACK-ZONE as all the ideas so far have been my own. What I would like is some ideas on either programs you would like to see converted or games you would like POKE's for. I would also like to see your conversions, pokes etc, which I will try to print in future HACK-ZONES.

Thats all for now so see you next month.

SPACE SAVING IDEAS

By: Clyde Bish.

This article is a pot-pourri of ideas to save that odd byte.

Messages appearing in the edit area, lines 22 & 23, (PRINTed using #0) can be cleared by using simply INPUT; rather than the more "expensive" PRINT #0;"(32 spaces)". Text on the lower part of the main screen, for example, below a map or illustration. can be cleared by using **INPUT AT x,0;** where x = number of lines to be cleared +1. The current PRINT position must be above the area to be cleared or the screen display will scroll up. So to clear the bottom five lines of screen (assuming the PRINT position was below row 17) use:-

```
PRINT AT 0,0;: INPUT AT 6,0;
```

which is 33 bytes shorter than the more usual:-

```
FOR f=17 TO 21: PRINT AT f,0;"(32 spaces)": NEXT f
```

If you need to clear rows in the central area of the screen you will have to resort to this loop method, but with a difference: There is no need to use 32 blank spaces. Two commas will do the job! So to clear rows 10 to 15 use:-

```
FOR f=10 TO 15: PRINT AT f,0,,: NEXT f   (saving of 30 bytes)
```

Windows (i.e. not whole rows) are more difficult as you can't use the double comma dodge, (although a single comma will clear the left half of screen only!) If its a routine to be used often then make it a subroutine. So if you want to clear the centre 10 columns of the bottom five lines use the subroutine:-

```
9000 FOR f=17 TO 21: PRINT AT f,11;"(10 spaces)": NEXT f: RETURN
```

and call it whenever needed. In a similar way you can use a method very similar to the PROCEDURE statement available on some computers, to clear a window of any size anywhere on screen. Your subroutine would be:-

```
9000 FOR f=r1 TO r2: FOR n=c1 TO c2: PRINT AT f,n;"";: NEXT n: N
EXT f: RETURN
```

Before you call it you must set the variables to the row/column of the corners of the window. So to clear a window from rows 5 to 10, and columns 10 to 20 use:-

```
10 LET r1=5: LET r2=10: LET c1=10: LET c2=20: GOSUB 9000
```

Two final points about subroutines. Firstly don't assume that

using a subroutine will always save you bytes. There's no point in having a subroutine you only call once! Also, a small subroutine and its call may take up more bytes than just including the routine each time you need it. Secondly, never jump out of a subroutine. Always leave by way of a RETURN. Calling a subroutine puts 18 bytes on the memory stack. When you RETURN these come off, but if you jump out they stay and the stack grows until memory is used up. You can, of course, call one subroutine from another, or even have a subroutine call itself, provided you eventually RETURN back to the main program.

Another byte-user is the control variable used in a FOR ... NEXT loop, i.e. the 'f' or 'n' in the above examples. Setting up takes 18 bytes which you wont get back when the loop is finished so its a good idea to reserve certain variables for this purpose only. (I personally use f or n as they are on the FOR and NEXT keys and so less finger movements are needed when typing).

The little understood, and therefore little used, DEF FN command can also save a lot of bytes if used sensibly. Look in your manual for the various forms of syntax but here's a simple use. Say you have an adventure where some outcome is determined by the throw of a dice, Dungeons and Dragons style. Normally you would need to include a line such as **IF INT (RND*6+1).....** Subroutines won't help here as you'll probably want to compare it with a different variable each time, but DEF FN can be used. First define your function with:-

```
DEF FN r()=INT(RND *6+1)
```

Then whenever you need the dice throw use:-

```
IF FN r()= .....
```

If you want the total of two dice use

```
DEF FN r()=INT(RND*6+1)+INT(RND*6+1)
```

Because of the way numbers are held, numeric DATA always takes up a lot of space. If variables are set for numbers (e.g. LET o=0) then a great deal of space can be saved. This can be put to good effect using the BEEP command. A succession of BEEPs to play a tune would obviously be byte-wasting, so use a loop, set to the number of notes, READ the length and pitch of each note from DATA, and BEEP using the variables read:-

```
10 FOR f=1 TO 10: READ l,p: BEEP l,p: NEXT f
20 DATA ...
```

If you declare variables with capital letters the same as the name of the note - A, B, C etc (use TC for top C, Bb for B flat, C# for C sharp), and others for the note length, declaring the shortest first, then the others in terms of this one (e.g. q=.5 (for quaver), k=q+q (for crochet - You've already used C!), dk=q+q+q (for dotted crochet) etc). you can enter the BEEP DATA directly from the stave music simply by entering the variables for the note length and pitch.

More space saving next month.

TWO-FACE COMPETITION

When I visited MGT at their Swansea H.Q. a few weeks ago I was shown a collection of publicity photos they had commissioned. Well one of them caught my eye and gave me the idea for a fun competition for FORMAT readers.

Here is the photo of Alan Miles (holding a Two-Face) and Bruce Gordon. It reminded me of a famous television comedy duo and immediately begs the question - What Are They Saying?

The competition is simple, write on a plain peice of paper what you think Alan and Bruce are saying to each other. Then write your name, address and membership number on the back of the paper and sent it to:- FORMAT Two-Face Compo, 34 Bourton Road, Gloucester, GL4 0LE. To arrive no later than first post on 16th December 1988. You can make as many entries as you like but each must be on a separate piece of paper.

Alan and Bruce will then pick what they believe to be the two funniest. The two winners will each receive a Two-Face of their very own. We will also print some of the funnier entries in a later issue so we can all have a good laugh.

ADDING COMMANDS TO BASIC

By: Nev Young.

This new series of articles will attempt to show how the experienced machine code programmer can extend the BASIC language by adding new COMMANDs and functions. It will also attempt to highlight some of the pitfalls that I have fallen into whilst doing this. This article refers to GDOS 3d on the DISCiPLE or G+DOS 1a/2a on the PLUS D. If you have earlier versions you should run the upgrade programs published in past issues of FORMAT.

Although this is aimed at the more experienced programmer do not be afraid of reading it. You just may learn something. You may find it useful to have your Spectrum handbook open at chapter 25 'The System Variables'.

As you know, I hope, when you enter a BASIC COMMAND it is first checked for correct syntax and then either stored in the programme area or executed depending on whether there was a line number or not. The spectrum shows what state it is in by a flag (bit 7 of FLAGS) which is zero for syntax checking and one at run time. There are also several system variables used by the interpreter and these are CH ADD, X-PTR, ERR NR and ERR SP.

During syntax checking CH ADD contains the memory address of the character being interpreted. Two restart calls are used to manipulate this RST 24 (18H) which loads the current character into the A register and RST 32 (20H) which loads the next character into the A register. In this way the interpreter steps along the BASIC line.

Before starting to look at a line of BASIC the interpreter will load an error address onto the machine stack, that is the address of the routine to goto if an error has occurred. The current value of the stack pointer is stored in ERR SP so that the machine stack can be cleared to a known state on error.

If an error does occur then an RST 8 instruction is executed followed by a byte that contains the error number. Under normal conditions the value of CH ADD is copied to X-PTR, and the error number is copied to ERR NR. Then the calculator stack is cleared and the machine stack is set to the value stored in ERR SP. If a line is being entered this will cause the flashing '?' to appear at X-PTR. If the error is found during run time then a message will be printed. Therefore ERR SP normally (but not always) points to a machine stack entry containing 4223 (107FH) during line entry and 4867 (1303H) during execution.

When syntax checking the Spectrum verifies that a sufficient number of parameters of the correct type follow a COMMAND word and that they are followed by the end of the statement. However,

during execution the check for the end of the statement only happens AFTER the COMMAND has been executed. You can verify this by typing PAPER 6: CLS #. You will see that the normal CLS is executed before the DISCiPLE takes over to do a CLS #. This can cause some problems.

When you have a DISCiPLE connected, and BOOTed, something different happens. The interpreter will still do a RST 8 when a error is detected but the DISCiPLE hardware will detect this and switch out the normal machine ROM and replace it with the DISCiPLE's memory. The DISCiPLE will then copy CH ADD to X-PTR. The machine code program in the DISCiPLE then performs a number of tests to find out why a RST 8 was done as it is NOT always due to an error, there are two other cases. We will assume it is an error, in this case the DISCiPLE will search backwards from CH ADD until it finds a COMMAND word. (This is not the most reliable method of finding the start of the statement). It will then compare the COMMAND with its own internal list and if a match is found it will syntax or execute it as required.

If no match is found then the DISCiPLE will return control to the main spectrum ROM, at address 88 (58H), to report the error that was originally found. Just prior to this return, however, the DISCiPLE will test its own system variable ONERR (at address @14) and if this is NOT zero then the address stored there will be called. Sadly if the DISCiPLE finds the COMMAND is one of its own and there is a syntax error, it does not allow us to use ONERR and it just reports an error. We cannot therefore use any of the COMMANDs that are used by the DISCiPLE for extensions.

It is by placing the address of our own routines at ONERR that we can add to BASIC. When the call to ONERR is made the following conditions exist:-

CH ADD points to the COMMAND word.
X-PTR points to where in the statement the original error happened.
ERR NR contains the original error number.
The A register contains the COMMAND word.
The machine stack is conditioned, by use of the DISCiPLE variable ERRSP, so that a return via the DISCiPLE will continue without error.
The DISCiPLE is conditioned to report any new error immediately via the main spectrum ROM. (DOS 3d only)
The main spectrum ROM is paged IN. (DOS 3d)*
The Y index register points to ERR NR.
The Workspace has been cleared. *
The Calculator stack has been cleared. *

* These will cause problems for us later.

From this information we can build our new COMMANDs.

There is what I consider to be a major flaw in this design, and that is that when the spectrum is reset you will lose the new code but ONERR will still point to it. Thus any error you get will cause a crash unless you reload the new code or zero out ONERR.

While the DISCiPLE is paged in there are a number of useful routines that we can use to build our new routines. Many of these are restart codes (eg RST 16).

RST 16 (10H) = CMR (Call Main Rom). This is used to call any machine code subroutine with the disciple paged out and then return to the instruction after the call with the disciple paged in. It is used by following the RST 16 with a DEFW that contains the address of the main ROM code.

RST 40 (28H) = GTNC (Get The Next Char). This does the same as a RST 32 in the main ROM.

CALL 44 (2CH) = RS18. This does the same as a RST 24 in the main ROM.

RST 48 (30H) = CSFO (Check For Syntax Only). Returns with the zero flag set if in syntax mode.

JP 70 (46H) = RTBC (Return To BC)

JP 79 (4FH) = RTHL (Return To HL) These will both page in the main ROM and jump to the address stored in BC or HL as their name implies.

CALL 1033 (409H) = CEOS (Check for End Of Statement). A useful routine that tests if the character pointed to by CHADD is a ':' or CHR$ 13. If not it reports an error but if so, it will test for syntax only and returns to the correct place. (ie to the main ROM or to the instruction after the call).
NOTE: This is at a different address, 1134, on the PLUS D.

There are many more.

Well thats most of the theory covered. As a pratical example I have written a small routine that gives a new COMMAND - "OUT * n,m" - which will delete lines n through m from a program. Next month I will give you the assembly listing with notes on how the routine works.

More Next Month...

DOS COMMAND CODES

By:Bob Brenchley.

When Uncle Clive produced the Interface 1 and Microdrives he introduced the concept of 'HOOK CODES'. These were a way of controlling the interface from machine code. They were fairly simple in concept (although not always very easy to use) and took the form:-

```
RST 8
DEFB n
```

Now some of you will already know that RST 8 is the call to the Spectrum's error handler. If the DEFB has a number in the range 0 to 26 it is taken as an error code (see appendix b in the Spectrum manual). Error code 27 to 50 where used with Interface one to tell it which of its routines you wanted to use. What happened was the Interface 1 ROM was paged in when the RST 8 instruction was executed, if the error number was one of its 'HOOK CODES' then it carried out the required operation before handing back to the main ROM.

When the DISCiPLE was designed Bruce Gordon decided to use the same methord for calling his Shadow ROM, indeed most of the Interface 1 Hook Codes are supported by GDOS so that machine code software written for Microdrives can still be used on disc. However, to exploit the full power of the disc & printer system a new set of codes were introduced and, to differentiate between them and Hook Codes, the name COMMAND CODES was coined.

DISCiPLE & PLUS D Command Codes lie in the range 51 to 71, or 33-47 hex for those of you who insist on working in that outmoded number system. I assume that you will be using the latest version of the operating system - DISCiPLE 3d or PLUS D 1a/2a - if not then some codes will not work.

The command codes recognised by the DISCiPLE & PLUS D are as follows:-

COMMAND NAME	CODE	HEX	COMMAND NAME	CODE	HEX
HXFER	51	33h	WSAD	62	3Eh
OFSM	52	34h	RSAD	63	3Fh
HOFLE	53	35h	REST	64	40h
SBYT	54	36h	HERAZ	65	41h
HSVBK	55	37h	COPS2	66	42h
CFSM	56	38h	PCAT	67	43h
PNTP	57	39h	HRSAD	68	44h
COPS	58	3Ah	HWSAD	69	45h
HGFLE	59	3Bh	OTFOC	70	46h
LBYT	60	3Ch	PATCH	71	47h
HLDBK	61	3Dh			

Now lets deal with each Command Code (CC) in turn and try to explain what each does. Dont worry about some of the labels used in the explanations, all will become clear as this series develops.

HXFER Code-51 33
This command transfers the file description and header to the Disc File Channel Area (DFCA). The IX register must point to the first byte of the 24 byte user's file information area (UFIA).

OFSM Code-52 34
Opens a File Sector Map with the information contained in the DFCA. A return is made with the disc buffer pointer (RPT) set to the start of the disc buffer RAM (DRAM).

HOFLE Code-53 35
Open a file. IX must point to the user's file information area (UFIA). This code combines the functions of HXFER and OFSM, also setting the 9 bytes of header information from HD00 to the beginning of the file. This header on the file allows the normal LOAD D1 "Filename" syntax to call the file.

SBYT Code-54 36
Save the byte in register "A" to DRAM at the location pointed to by RPT. RPT is incremented, if the disc buffer is full the sector is saved to disc and RPT is reset.

HSVBK Code-55 37
Save a block of data to the disc. The DE register points to the start of the data and the BC register holds the byte count.

CFSM Code-56 38
Close File Sector Map. This routine empties the DRAM and copies the header area onto the directory. Files that are not closed will not be in the directory and therefore dont exist.

PNTP Code-57 39
Output the 8 bit code in register 'A' to the printer port, after checking busy status of the printer. Note: the Break key is scanned but if pressed you are returned to Basic.

COPS Code-58 3A
Copy the screen RAM to the printer in high resolution screen graphics as set up under the graphics control variables. This is equivalent to Snapshot 1 or SAVE SCREEN$(1).

HGFLE Code-59 3B
Get a file from disc. IX register must point to the start of the UFIA. The return is made with the first sector of the file loaded into DRAM and RPT pointing to the first byte (usually the start of the nine byte header information).

LBYT Code-60 3C
Load the byte pointed to by RPT from DRAM, and return with it in the "A" register, RPT is updated. If the buffer is empty, another sector is read from the disc. Be careful - you must make sure there is another sector to read from the file or a system crash can result.

HLDBK Code-61 3D
Load a Block of data from disc to the memory pointed to by the register DE with the block count in register BC.

WSAD Code-62 3E
Write Sector at DE. Writes the contents of the DRAM at the track pointed to by D and the sector pointed to by E, and sets the pointer to the start of DRAM.

RSAD Code-63 3F
Read sector at DE. Reads the sector at track D, sector E into the DRAM, and sets the pointer RPT to the start of DRAM.

REST Code-64 40
Reset drive to Track 0. The drive number is taken from DFCA.

HERAZ Code-65 41
Erase the file on disc using the information contained in UFIA - so make sure that IX points to this.

COPS2 Code-66 42
Copy screen RAM to printer as A4 Gray Scale dump. This is the machine code equivalent of Snapshot 2 or SAVE SCREEN$(2).

PCAT Code-67 43
Print Catalogue to stream. Use HXFER to move the UFIA into DRAM.

DSTR1 = Drive No
SSTR1 = Stream No (if disc file it must be on same disc)
HD00 = 2 for short (CAT!) 4 for long INSTR2= "* " for full cat or more selective filename
Note: No CLS even with full CAT.
Note: Due to a ROM bug the DISCiPLE can't CAT into a disc file.

HRSAD Code-68 44
Read Sector to address - full 512 bytes the same as the Basic LOAD @ command.

A = Drive
DE = Track/Sector
IX = Address to load

HWSAD Code-69 45
As HRSAD but write sector.

OTFOC Code-70 46
Open type file OPEN and CLOSE if A=0 then open file using details transferred by HXFER. If A<>0 then close stream A.

PATCH Code-71 47
Returns with shadow system paged in. On return to your program HL is set to zero if you are on a PLUS D or to 1 if running on a DISCiPLE.

In the next article I will explain how the UFIA works and show you how to use a few of the Command Codes.

JIFFY CALLS

By: Paul King.

Some time ago, while writing a large basic program, I came up against one short fall in the 48k editor. One of my program lines was almost half a screen long and I needed to add an instruction on the end. Five minutes later and the new instruction was on the end of the line and I pressed enter... just as I realised I had made a mistake in one of the numbers, so back to the l o n g delay getting the cursor to the end of the line again.

Now the DISCiPLE came to my rescue. Every 50th of a second the Spectrum is interrupted in its normal processing so the keyboard can be scanned. During this keyboard scan the DISCiPLE pages in to do odd housekeeping functions and to look for network activity. It also checks the location relating to POKE @16 & 17, if these are non-zero the DOS uses the contents of these two locations as a CALL address for a machine code routine. Its known as a JIFFY call, because the next one comes along in just a jiffy.

The machine code routine give below adds two new functions to the 48k editor. Press Cursor-Up (cap-shift 7) and the cursor is moved to the start of the line in the edit area. Press Cursor-Down (cap-shift 6) and 'in a blink of an eye' your cursor is at the end of the program line, no matter how long it is.

The code is fully relocatable. Just load the code somewhere safe the enter the following direct command:-

POKE @16,address_of_code

Thats all there is to it, except for a word of warning. Once the POKE @ is done, the jiffy call will continue to be made even after a system reset and, as the code wont then be there, you will get a crash.

```
0010   ; 48K LINE EDITOR EXTENTIONS.
0020   ; V1.2  MARCH 1988.
0030   ; for DISCiPLE or PLUS D.
0040   ;
0050   ; (C)1988 INDUG.
0060   ;
0070   LASTK  EQU  23560
0080   KCUR   EQU  23643
0090   WORKSP EQU  23649
0100   ELINE  EQU  23641
0110   FLAGS  EQU  23611
0120   TVFLAG EQU  23612
0130   ;
```



```
0140   ;
0150           ORG  50000
0160   ;
0170   START   PUSH AF
0180           LD   A,(FLAGS)     ; Check if new
0190           BIT  5,A           ; key pressed an
0200           JR   NZ,EXIT       ; return if not
0210           PUSH HL
0220           PUSH BC
0230           PUSH DE
0240           LD   HL,(WORKSP)
0250           DEC  HL
0260           DEC  HL            ; HL = end of line
0270           LD   DE,(ELINE)    ; DE = Start of line
0280           LD   A,(LASTK)     ; A = New key
0290           CP   10            ; Test if Cursor-Down
0300           JR   Z,ATEND
0310           CP   11            ; Test if Cursor-Up
0320           JR   Z,ATTOP
0330   ;
0340   FINISH  POP  DE
0350           POP  BC
0360           POP  HL
0370   ;
0380   EXIT    POP  AF
0390           RET                ; Return to DOS
0400   ;
0410   ;
0420   ATTOP   EX   DE,HL         ; Swop DE/HL if rec.
0430   ;
0440   ATEND   LD   (KCUR),HL     ; Store new cursor poss
0450   LAST    LD   HL,FLAGS
0460           RES  5,(HL)        ; Set to say key used
0470           JR   FINISH
0480
0490   LEN     EQU  $-START
```

To explain, if a new key is pressed (bit 5 of FLAGS) then HL is made to point to the end of the line and DE to the start. Now a test is made for both of the keys we use, if neither is pressed then everything is restored and a return is made. If cursor-up was pressed then DE is exchanged into HL so it points to the start of the edit line. Cursor down leaves HL pointing to the end of the edit area. All that now needs doing is to store HL on the system variable that holds the cursor position and then flag the key press as used.

For those of you without an assembler - GET ONE SOON. No seriously, here are the pokes, remember you can change the address of the code if you want as its completely relocatable.

```
10 FOR I=50000 TO 50046: READ N: POKE I,N: NEXT I
20 DATA 245,58,59,92,203,111,32,26,229,197
25 DATA 213,42,97,92,43,43,237,91,89,92
30 DATA 58,8,92,254,10,40,10,254,11,40
35 DATA 5,209,193,225,241,201,235,34,91,92
40 DATA 33,59,92,203,174,24,240
```