The 68XXX Machines

Vol. 1, Issue 5, June 1991 Price: $2.75

The 68xxx Machines is published and copyright (C) 1991 by Chatham House Company, RD#1 Box 375, Wyoming, DE 19934. Ph. (302) 492-8511. The editor is James H. DeStefeno. One year USA subscription is $12.50. Canada and Mexico $14.25. All others (surface) $17.25. All major credit cards accepted. Our low prices reflect a 10% cash discount. Please add 10% to credit card orders. Any size display advertising is accepted. The half page rate is $10/issue. Write for other size/duration rates.

Readers are encouraged to contribute letters, articles, programming information and other material related to computers with the 68xx(x) processors; excepting Macs and Amigas. Please send material to the above address. Thank you for your support.

The Editor's Thoughts

By Jim DeStefeno

For those that have asked; yes, Ron Anderson's series, "Beginners Corner", was superseded in April by Ron's operating system over view, and in May by his two reviews. This month the series has been replaced by Peter Stark's letter of clarification and addition-to Ron's operating system overview article. However, don't get antsy. "Beginner's Corner" will be back next month.

This month marks the introduction of a series, "Tech Corner", by J. Scott Kasten. He is a professional programmer with a gift that allows him to share what he knows in an interesting manner. Welcome aboard Scott, we are pleased to have you, and are looking forward your future articles.

Again, there is no room for a "Letters to the Editor" column. However, lets slid a couple in right here:

HOW 'BOUT MORE

I'd like to see more for the CoCo-3.

This Issue:

Ernest Dotson, Jr
Charleston, WV

Keep in mind, all the 'C' action will work on the CoCo under OS-9. Also, we haven't had any 09BASIC as yet, but I'm sure it won't be long before we do.

I THOUGHT A MONITOR WAS...
I'd like to read about other 68XXX operating systems. What is REX/Monk anyway? What about Minix?

James Truesdale
Ferguson, MO

Our April issue had a good overview of REX, however Monk was lightly passed over. Monk is a monitor used with REX. Yes, monitor in computer terms can mean software too. Monk is an example of the software definition. As software, a monitor is a short M/L program. They are in ROM, automatically activated upon startup. Though a monitor may have several routines of its own, its principal duty is to automatically load and execute another program, like an operating system; in the case of Monk, REX-DOS. We have a review of the UNIX look a like, Minix coming soon.

COMM, HOW-TO?
One thing I'd like to see is information about the selection, installation and use of a modem in bulletin board accessing.

William Tobin
Jacksonville, FL

Call Gil Shaduck, the proprietor of Granite Computer Systems, one of our advertiser. He is as deep into modems as anyone. (Maybe there is a reader that would like to write an article addressing the question?)

MULTIPLE OPERATING SYSTEMS:
Write about multiple operating systems on one machine. The host should be UNIX or OS-9 and run MS-DOS as a guest. In fact, we should be able to run a different OS in each window.

Wood Elis
Arlington, VA

Well.... Wood, that's a tall request. IBM (OS-2) and most of the other leading companies have been trying to do the same thing. On our level, give Ed Gresick of the delmar company, one of our advertisers, a call. He and Peripheral Technology are working on the problem. They have a roundabout solution for today, and expect a direct solution in the future.

WHAT'A BOUT BATTERY POWER Editor:
I'd like articles on a battery powered ram disk, and a print spooler. I'd also like some new BBS phone numbers.

Bill Hughs
Valley Falls, KS

OK hardware hackers, can you write a generic battery powered memory article? I guess a generic spooler would be asking too much, but if you can't do either, send in a BBS number.

Advertiser's Index

More Random Number Uses

By Bob van der Poel

Last month we discussed a novel use for a random number generator (in justifying text). This month we'll be a bit more conventional and develop a general purpose card dealing function.

When creating a card dealing routine a programmer's first impulse could be to create a deck and then to shuffle it--just like a human dealer might. Unfortunately, this is fairly complex and time consuming. A much simpler method is to represent the cards in an array and deal them out at random. All that is needed to represent 52 cards is a 52 byte array with each element representing a unique card. In this case we will use:

```
char deck[53];
```

Note that we have allocated 53 slots for the deck. The reason for this is we don't want to bother with card '0'. This way the numbers of the cards can be 1 to 52. This can match slots 1 to 52. Slot 0 is unused. The cards in the deck are labeled 1 to 52: Card 1 is the Ace of Hearts, card 2 is the Ace of Diamonds . . . card 52 is the King of Clubs. To convert card numbers to conventional notation we can use:

```
suit=  (cardnumber-1)/13;
value= cardnumber-(13*suit);
```

To create the deck it is a simple matter to fill in the array with the card numbers:

```
for(c=1;c<53;c++) deck[c]=c;
```

All that is left is an efficient method to select a card and to keep track of what cards are left in the deck. The simplest method is to have a counter representing the number of cards left in the deck. This is initialized to 52 when the deck is created:

```
cardleft=52;
```

Now, to select a card we pick a random number between 1 and 'cardleft' and select that card:

```
n=(rand() % cardleft) + 1;
c=deck[n];
```

Here we have used a conventional rand() function which returns a random integer. Depending on the rand() or rnd() functions you have for your compiler you might have to fiddle with this.

Now, (the neat part) we move the last card in the deck to the slot we just selected. If we don't do this we'd have to set up flags to represent already picked cards, etc. This way we just shorten the array:

```
deck[n]=deck[cardleft--];
```

The following code contains the complete deal and shuffle routines. Shuffle() is automatically called whenever the deck is emptied, you should call it whenever you start a new hand. Note that it seeds the random number generator with the system time the first time it is called--you may need to change this for your compiler. Getcard() returns the card number and sets the variables passed to the card value and suit. To call it use something like:

```
int card, suit;
char *suitnames[]={
        "Clubs",
        "Diamonds",
        "Hearts",
        "Spades"
    };

....

getcard(&card, &suit);
printf("Card is the %s of %d\n",
        suitnames[suit],card);
```

Note we have passed pointers to 'suit' and 'card'. This is because it is not possible for a C function to return more than one value. The 'card' values returned will be in the range 1 to 13; the 'suit' values will be 0 to 3--you can assign "Clubs", etc. in any manner you wish to these values. Of course, the printf() in the above example needs some work---right now the card values are printed as "1", "2" . . . "13". Can you fix it up so that we get "Ace" . . . "King"?

```
/*---------------------------------
    This routine creates a deck of cards. The term
shuffle is actually a misnomer since the cards are
```

```
not held in a shuffled state. Deal() picks a random
card from the remaining cards in the deck. The deck
is an array of 52 numbers (1..52) representing the
different cards.
   */

   #include <os9.h>

   /* these vars are shared by shuffle() and
       getcard() */

   static char deck[53];     /* the deck */
   static direct int  cardleft=0; /*cards left
                                    in deck*/

   shuffle()
   {
      register int c;
      int t;

      static first=0;
      struct registers regs;
      char timebuf[6];

      if (!first){
         regs.rg_x=timebuf;  /* buffer for time
                                packet */
         os9(F_TIME,&regs);
         for(c=0;c<6;c++) t+=timebuf[c]; /* get
                                time number */
         srand(t);   /* seed rnd gen */
         first++;
      }

      for(c=1;c<53;c++)  deck[c]=c;
      cardleft=52;

   }

   /*--------------------------------------
   pull a card from the deck
   */

   getcard(card, suit)
   int *card; /* value (1..13) of card (king=13) */
   int *suit; /* suit of the card selected */
   {
      register int c;
      int n;

      if(!cardleft) shuffle(); /* just in case they
                                  are all gone */

      n=(rand()%cardleft)+1;     /* get random card
                                    number */
      c=deck[n];                 /* the value of the
                                    card */
      deck[n]=deck[cardleft--]; /* delete this card
                                   from the deck */
      *suit=(c-1)/13;
      *card=c-(13 * *suit);
      return c;                  /* return the card
                                    number */

   }
```

I have used the above code very successfully in a number of card game simulations. It could be expanded for multiple decks just by changing the array size.

Next month will be a bit more serious (hmmm, try telling some of my friends that cards are frivolous!). As always, if you have comments or suggestions on these columns drop me some mail at PO Box 355, Porthill, ID, 83853.

The Technical Corner

By J. Scott Kasten

Welcome to a new column! I hope to cover a number of topics of interest to the general programming public. Although we will talk a little bit about programming technique, the concentration will be on the technical information that one needs to program effectively.

The source code listings will be generated under OS9/68000 with C and assembly languages. The code will be designed such that readers with other systems can adapt it rather easily.

The first topic to be covered will be COMPUTER GRAPHICS, one of my favorite subjects. We will be spending several columns on this so let us lay some ground rules.

When one does graphics programming in C, a graphics library is normally used. Discussing graphics in a column like this is a problem because there are so many different graphics adapters and graphics libraries in use. Any source code listing given is pure gibberish to someone using a different system. In addition, some graphics are done in assembly for performance reasons. However, using assembly in C can be tricky if you have never seen examples. Fortunately, there is a simple solution. Let us start by building an ultra basic library of our own!

There are some basic routines any library must have, so we will start with those. We must be able to activate the desired graphics mode, clear the screen, set points, and return to a text mode. With these few routines, we can accomplish quite a bit.

These routines are quite simple to write using OS9 on a Peripheral Technology system. It is as simple as making a system call to the screen driver. The system calls are best accomplished with assembly; this is where the C-assembly hybrid comes in.

The design of Microware's C compiler facilitates the development of such a hybrid. The compiler translates the C source code into 68000 assembly. It is fed to the system assembler and the linker in turn. There are two ways to use assembly in your C programs. One is using the inline code option, the other is to use the assembler and link the code by hand. Either way it's done, there are certain rules to be followed.

The first rule concerns the CPU registers. On entry to a procedure, the arguments will be placed in registers D0 and D1, any other arguments will be placed on the stack. Data types less than 32 bits will be sign extended to the full 32 bit width of the register size, even when placed on the stack. The exception is the float data type. This will be converted to a double. If a double is the first argument passed, it will occupy both D0 and D1 when passed. Any other args will be put on the stack. On return from a procedure, the return value will occupy D0, and perhaps D1 if it's a type double. All registers not used in parameter passing must be preserved. There are some special registers that sort of belong to the programmer. These are registers D4-7, and A2-4. The registers in this group are used when a 'register' declaration is used in the C source code. Such a declaration causes the data element to be handled in one of the above registers as much as possible. Note: These registers must still be restored at the end of the routine if used!

The second rule, any procedure declaration done in assembly must be made a global symbol using the colon operator.

The third rule, libraries should have a header file to declare the procedures contained as external.

This is quite a mouthful to digest all at once. I never could

Classified Ads

- WANTED - SS-50 equipment. SWT CPU card, also Gimix PIO #28 (30 pin) FDC. Alen E. Gordon, MD / 160 NW 176 St / Miami, FL 33169 / (305) 653-8000.

- SALE - PT68K-2 complete system: 10MHz motherbd w/1M RAM, 20M hard drive, 720K floppy, amber monitor, SK*DOS, RBASIC, EDDL, all manuals, more! Used very little, $800 + shipping. Phil Krieg, 1122 1/2 Bayland, Houston TX 77009. 713/861-0367 eve/wkends.

Turn that old computer equipment into cash with a *68xxx* classified ad. Sale ads are $5 per 50 character line. Wanted ads are just $2.50 per 50 character line.

assimilate a large mish-mash of rules without working through them first. Nor do I expect you to either. Check them out to be sure you 'have them' and then lets start working with some real code. We will see how this all fits together. The first thing to do is to create the file 'GFX.C'. This is the library source code file.

Listing #1 for GFX.C

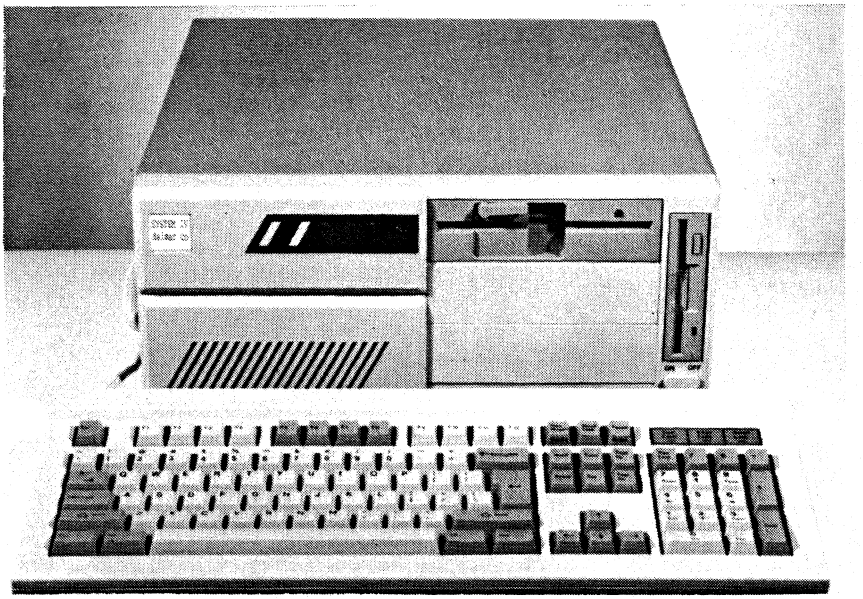
```
/*-----------------------------
  Routine to turn on graphics.
  Form: Graphics(charmode);
-----------------------------*/

Asm Graphics: * Save regs on stack.
        move.l  d0-d2,-(sp) * Copy mode (d0) to proper reg.
        move.b  #0,d2 * Set path to StdOut.
        moveq.l #1,d0 * Set operation code.
        move.w  #$80,d1 * Do sys call.
        OS9     I$SetStt * Recall regs.
        move.l  (sp)+,d0-d2
        rts Fendasm
```

Notice this routine is done entirely in assembly, yet it will be fed into the C compiler. As soon as the compiler sees the #asm statement, it starts sending the source code out to the assembler directly instead of translating it first. When the #endasm statement is encountered, the compiler starts processing the code as it did before. Also notice the colon after the procedure name; this is the directive that tells the linker this symbol will be used outside of this source code file. You must use it!

This routine has exactly one calling argument - the graphics mode number. I felt a one byte value would be sufficient to hold the value. The data type in C that consists of one byte is type CHAR. When calling this routine, you would use an argument of this same type. The calling argument is extended to 32 bits and placed in D0 before entering this routine.

The I$SetStt call to the driver needs 3 arguments. It needs a path number, an operation code, and a mode argument. These should be placed in regs d0-d2 respectively so the driver can find them. The path is assumed to be StdOut (1), and the operation code is $80 - the value for SS_SetMode. This is a function provided by the Peripheral Technology driver. The driver

SYSTEM IV



SYSTEM IV - The 68000 Computer already serving customers here and abroad!

- Expandable using readily available low-cost cards
- Flexible enough to tailor to your requirements
- Versatile enough to use OS-9/68000, OS-9/6809, MS-DOS and other Operating Systems
- Powerful - outperforms other machines in its' price class

THE BEST OF ALL WORLDS

1 - A Multi-User Multi-Tasking System
2 - Optional Plug-in Board for MS-DOS
3 - Optional Emulator/Interpreter for OS-9/68000 Software
4 - Will support Other Operating Systems

Just call, FAX, or drop us a line. We will respond promptly with complete pricing and specifications.

DELMAR CO - serving its' customers since 1975.

Terminal Systems . . . . from $999.00 Console Systems . . . . from $1,149.00

Assembled boards and kits available

delmar co

Middletown Shopping Center - PO Box 78 - Middletown, DE 19709

302-378-2555 FAX 302-278-2556

OS9/68000 SOFTWARE

M6809 - 6809 Emulator/Interpretor $ 99.95

M6809 emulates the user mode OS9 Level II operating system and interprets OS9-6809 binary object modules under OS9-68000.

QUICK ED - Screen Editor and Text Formatter $275.00

A high quality documentation tool and program editor ideally suited to laser printer users. Uses function and cursor keys on any terminal, configurable per user. Micro-justifies mixed proportional text. Automatic table of contents generation and user-definable macros and commands. Handles an unlimited number of fonts. Drives any printer. Ideal for multi-user systems. Available on a 30-day try before-you-buy basis.

FLEXELINT V4.00 - The C Source Code Checker $495.00

Flexelint finds quirks, idiosyncracies, glitches and bugs in C programs. 60 options control checking by symbol name or error number. Checks included intermodule inconsistencies, definition and usage of variables, structures, unions and arrays, indentation, case fall-through, type conversion, printf and scanf format string inconsistencies, and suspicious semi-colons. A must for all serious C programmers.

DISASM_OS9 - OS-9/68K Disassembler $250.00

This high-speed, three-pass 68000 disassembler can also handle the 68010 and 68020. It intelligently decodes module headers and produces symbol information that can be repeatedly edited and passed through the disassembler allowing iterative disassembly. The system libraries are read to supply symbols.

IMP - Intelligent Make Program $250.00

IMP does everything you wished Microware's Make would do, and a great deal more. It is well-behaved, consistent, and extremely flexible. It has a built-in C like preprocessor and has comprehensive debugging facilities. Rules can be user-defined, and make files for jobs other than assembly-language or C compilation are easily constructed.

WINDOWS - C Source Code Windowing Library $250.00

This C source code library package supports multiple overlapping windows displayed on one character-based terminal screen. It supports window headers and footers, and pop-up windows. Windows may be moved, panned, written to while off-screen, etc.

PROFILE - User State Program Profiler $270.00

Designed to profile user-state programs. Profile effectively samples a traced execution building statistical information as it goes. It reads symbol table modules to give a function-by-function account of the time spent during execution. The user may "zoom-in" on a function to find a smaller range of addresses where time is being spent.

PC9 - MS-DOS to OS-9 Windowing System $350.00

PC9 allows an MS-DOS computer to be used as a terminal to multiple processes on a remote OS-9 system linked by a single serial cable. Each OS-9 process is displayed through a resizable, moveable window on the PC screen. Terminal emulation facilities support uMACS and other screen editors and provide a programmable PC keyboard. Access to PC disk drives is also avilable through the OS-9 unified I/O system, giving disk capability to ROM based OS-9 systems.

delmar co

Middletown Shopping Center - PO Box 78 - Middletown, DE 19709

302-378-2555 FAX 302-378-2556

would also like the mode value in reg D2, not D0, so we must move it.

Please note the use of the movem instructions at the beginning and end of the procedure. We must preserve the registers we borrow, remember?!!

If you are using a different system, you will probably have to activate a graphics mode in a different way. Just use the code specified in the literature you got with the machine. BUT, please use the same procedure name as above! The whole point of this exercise is for everyone to have a similar set of graphics calls when we start to do some really serious stuff.

Next time, we'll try to finish this library and learn some more about the Microware C compiler. If you're having some trouble digesting all this and have the OSK C manual, look in section 3-1. We'll talk more about this next time.

(Questions and comments can be addressed to Mr. Kasten in care of The 68xxx Machines.)

Rush Caley, Live!

This was to be the month I outlined my perfect solution to the drug problem in America; but now, that will have to wait until next time. I'm finally to the boiling point concerning the Federal, State, and local Government monitoring our lives. Now don't get me

"Jello sheriffs watch everything you do."

wrong - I'm all for the CIA, FBI, AFT, State, and local police doing their jobs. But now we've got a situation where self-appointed "jello sheriffs" are trying to institutionalize their intrusive behavior.

For anyone unfamiliar with the term "jello sheriff, I'll digress a bit. These are the type of people you knew in grade school who

were appointed "monitor" when the teacher had to leave the room. They're the ones who put your name on the board with a check after it. Jello Sheriffs watch everything you do; and if they disapprove of your behavior, they want to report it to a higher authority. We have one at our office who keeps track of how long anyone takes on a smoke break and keeps our supervisor apprised of anyone "taking advantage".

One of the more virulent forms of the new police mentality that has gained national notoriety is this business of harassing pregnant women. Two bartenders in Seattle were ultimately fired because they refused to serve a drink to a pregnant woman and harassed her about pre-natal alcohol syndrome.

"...I'll smoke too many cigarettes, drink too much coffee, eat too much salt and absolutely will not wear my seat belt. So there!"

Another lady was forced out of a hot tub at her local spa. And the list goes on. This type of irrational "concern" makes me wonder about things. I find it so ironic. These same people will rail on and on about a woman's right to her own body and the right to kill an unborn in an abortion.. But noooo... we mustn't have a drink, or have a hot bath, or ride, or go bowling.

Also in our state, there is a bill proposed in the legislature to limit the number of pets people may have to a quantity of 2. Now what constitutes a pet? Will our publisher Destafeno be limited to 2 sheep on his ranch? What about cattle and horses? Who's going to tell 13 of my 15 cats they'll have to be put down because I'm over the limit? I also wonder when this will extend beyond the realm of pets.

But you can see what's happening. The jello sheriffs are having their way. These are the people that circulate petitions, and int-

roduce laws that infringe on individual liberties and drive the cost of goods and services to the ceiling. Warning labels are covering product containers to the extent that there's barely room for the brand name. The followers of the snoop mentality are attempting to create an employment opportunity infrastructure for jello sheriffs!

"Well, what do you do for a living, sir?"

"Oh, I count pets."

OR

"I'm on the preggers patrol."

C'mon! Don't we have enough concerns in our own lives to keep us busy? Well if we didn't before, we do now. Beware of people and laws that wish to protect you from yourself. I'm going to have my own week of personal protest. All next week, I'll smoke too many cigarettes, drink too much coffee, eat too much salt and butter, and absolutely, will not wear my seat belt. So there! RC

Murphy's Computer Dictionary

o o o

Hard drive: The 14th hole at the Torrey Pines golf course.

Lisp: The only programming language named after a speech impediment.

Obsolete: Any hardware or software more than three months old.

Recursion: See *Recursion*.

Sanskrit: Language in which most software documentation is written.

Reply to "Rex, SK*DOS and OS-9

By Peter Stark

(The following is the "mostly intact" letter written to "68xxx" in response to an article in the April '91 issue. I hope you will find it as I did; highly informative on several levels, as well as directions. ED)

Dear Jim,

I enjoyed reading Ron Anderson's comparative review of SK*DOS, OS-9/68K, and REX; but I have several comments to make; in order from shortest to longest:

1. SK*DOS for the 68000 is now priced at $70. But there is a special upgrade price of $50 to anyone who trades in any other 68K DOS.

2. SK*DOS is currently available only from Star-K Software Systems Corp., P.O. Box 209, Mt. Kisco NY 10549, and selected authorized licensees. It continues to operate properly on Peripheral Technology computers, but is no longer available from them. (VGA is still not supported, ED)

3. Ron forgot to mention many of the features which differentiate SK*DOS from some of the others. For example, SK*DOS can handle any combination of floppy drives at the same time, 40- or 80-track, high or normal density. It can double-step to read 40-track disks in an 80-track drive (it can even boot from this combination). It can have two disk controllers at the same time. Individual drives can be software write-protected, which can prevent accidental writes to your hard drive. You can select any disk interleave by adding an argument to the format command. And there is more. The point is that SK*DOS is designed to be flexible.

4. Although I have for some time felt SK*DOS could be substantially speeded up by reading and writing an entire track at a time, several knowledgeable users have insisted the only way to get MS-DOS-like

disk speed is to adopt the MS-DOS disk format. Now that Ron has pointed out whole-track operations do indeed speed things, I am making that change to SK*DOS. Incidentally, Ron's speed comparison between SK*DOS and other DOSes did not take into consideration that SK*DOS automatically verifies the disk after each write (unless you tell it not to), whereas other DOSes simply hope the disk was written OK, and do not check it. If verification had been turned off in SK*DOS, there would not have been nearly as much speed difference as he found.

5. Finally, Ron says SK*DOS is "based on FLEX" and is "a more grown up version of the FLEX style operating system than REX." I would like to make it clear that SK*DOS is VERY different from both FLEX and REX, both historically and internally.

First, a bit of history. SK*DOS came about almost by accident. In the early 1980's, I wrote a spelling checker called Magic Spell for the SWTP 6800 computer. With versions initially for the Percom DOS, and then for 6800 Flex. The latter was translated to run with 6809 Flex. A few months later, Don Williams (of the now discontinued) "68 Micro Journal" convinced me to buy a Radio Shack Color Computer and adapt Magic Spell to it.

As soon as I got the CoCo, I realized it had no documented entry points for assembly language programs to tie into. In other words, I would have to write my own disk management routines to use it. This was not easy, since the CoCo's disk format was totally different from any DOS I had ever used before.

To make the job as simple as possible, I decided to write the CoCo routines in such a way that Magic Spell (which was renamed Spell'n Fix when I sold the Magic Spell name to Peachtree Software -- but that's another story...) could tie into my disk routines with as few changes as possible. However, due to the many differences everything had to be written from scratch.

When I finished, I suddenly realized I had written a Disk Operating System! Internally it was very different from both Percom DOS and Flex DOS, but the entry

points matched those of 6809 Flex because they had to - I wanted the same version of Magic Spell to run under both systems without changes (except for shortening text strings to fit the CoCo's narrower video screen).

Since there was no DOS available for the CoCo at that time, I put my disk routines into a separate package, and marketed them under the name STAR-DOS. Since STAR-DOS was designed to run Magic Spell without changes, it could run other software that had been written for 6809 Flex too. People suddenly found they could run their old editors, assemblers, and other software on the CoCo. STAR-DOS eventually appeared inside other CoCo products as well. Other CoCo software vendors licensed it so they could make their Flex software run on the CoCo with minimal changes.

But I immediately ran into problems -- some software would work with Flex, but not with STAR-DOS. In some cases there was not much to be done, since the problem was caused by the different disk format. But in other cases I was very difficult to figure out what the problem was. Due the lack of debugging facilities for the CoCo I ported STAR-DOS back to my regular 6809 system.

The trouble with the CoCo disk format was it only supported single-sided disks, only 35 tracks, and was very inefficient. I didn't want to implement that on my "big" 6809 system; so I changed STAR-DOS to use the same disk format used by Flex.

And so, over the space of a year or so, STAR-DOS gradually grew to the point where, while its insides were totally different from Flex, it (a) used the same disk format, and (b) the interface to application programs (such as Magic Spell) matched Flex's to the degree that these programs could run with STAR-DOS (which was renamed to SK*DOS right about then.)

This was all very nice, but some of the application programs STILL didn't work with SK*DOS. Every few weeks, someone would send me a program with a comment like "it runs with Flex, why the h--l doesn't it run with SK*DOS??" Well, of course the problem was that SK*DOS didn't work like Flex.

It was different for two reasons - first, because it just sort of grew into place over several years. The more important second reason was - at this point, SK*DOS ran on exactly the same hardware as Flex, and was therefore a direct competitor for Flex, which was still marketed by TSC. I knew that if there was any slightest similarity with Flex, TSC's lawyers could have a ball!

I was careful to write STAR-DOS / SK*DOS under "clean-room" conditions, similar to how Phoenix wrote the first IBM-compatible BIOS for IBM clones. I made sure to stay away from anything and anyone who might have any source code or other confidential or unpublished information about Flex, and made sure never to look at any Flex code itself. Instead, SK*DOS was written from scratch, but in such a way that it would do the same things that magazine articles, other authors, or user's manuals claimed that Flex did. (I know that a number of people have carefully examined the innards of SK*DOS to look for similarities to Flex; I am quite sure that neither TSC nor anyone else ever found any.)

The real problem with application programs that didn't work was some of the people who wrote knew more about Flex than I did! They would use things in Flex which were not documented anywhere or generally known, or rely on things which Flex did for its own benefit, things which SK*DOS either did differently, or did not do at all. In some cases, they even seemed to take advantage of things that looked to me like Flex mistakes. The hard part was discovering what was going on without looking at Flex code, so I could modify SK*DOS to keep these application programs happy.

I eventually developed a workable technique. When someone sent me a program which did not work correctly under SK*DOS, I would run it simultaneously on two systems, one running Flex, the other running SK*DOS. I wrote a trace program which would send out the results of every step (except for steps inside the DOS, which were obviously different), and a third computer which compared these results, looking for differences. It might sometimes take hours before

this third computer would signal a difference, at which point I'd have to decide whether it was significant. If so, then I had to make a change to SK*DOS to give the errant program the same inputs as it was getting from Flex, and then run the whole thing again.

It took almost two years to get 6809 SK*DOS to the point where everything would run on it. In the process, I had to add some totally useless, and sometimes even wrong things to SK*DOS just to fool other programs so they would work the same way as they did with Flex.

The end of the story - SK*DOS's adaptation to the 68000 -- seems almost anticlimactic. Now that I was no longer limited to squeezing everything into 8K of memory, it became possible to extend and improve SK*DOS beyond its 6809 roots, adding things for which there was never room before.

When I look at it in retrospect, I see the development of 6809 SK*DOS, and ultimately 68000 SK*DOS, could have been done in two different ways. The easy way would have been to just wait until Flex was no longer supported. When the original manufacturer no longer cared, I could then simply copy it. This could have given us a Flex-like 68000 operating system with a minimum of fuss and effort.

But SK*DOS was developed the hard way, the honest way, knowing that at any time the manufacturer of Flex, or any one of his licensees, could pounce on me if there was even a hint of copying. Any operational similarities to Flex are there not because it was convenient to copy them, but because they were absolutely necessary to allow other people's software to work with it.

This is why I want to stress that SK*DOS is NOT "based on Flex." It also explains why SK*DOS source code is not available for distribution, and why it has not been placed in the public domain. SK*DOS is an original product, 100% legally protected with copyrights and a registered trademark, and a potentially valuable property, worth protecting.

PS

"The 68xxx Machines"
The Chatham House Company
RD#1 Box 375
Wyoming, DE 19934 USA

Address Correction Requested -

- First Class -