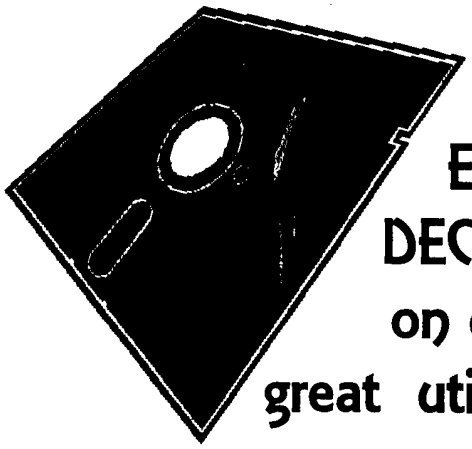
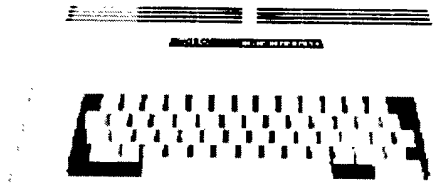
May/June1998 Vol. 5 Number 6 $4.00US (North America)

the world of 68' micros

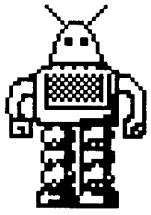
Support for Motorola based computer systems and microcontrollers, and the OS-9 operating system

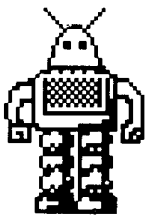


Exploring how DECB stores data on disk -- and a great utility to edit it!



CONTENTS

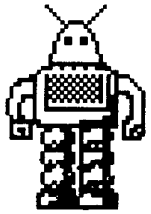


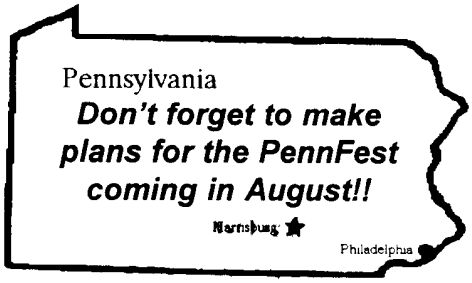


Play

Robot Zap!

Just type in this DECB program, and if you have a Speech/Sound pak listen in!





POSTMASTER:
If undeliverable return to:
FARNA Systems PB
Box 321
WR, GA 31099

If your address is incorrect, send me a postcard!

The Editor's Page

WANTED: EDITOR FOR "68' micros" MAGAZINE

Must have good general knowledge of the CoCo and all hardware, interest in OS-9 and OS-9/68K machines. Must have at least a 486 clone with 8MB RAM and 40MB free hard drive space for necessary software. Should have some experience with DTP software (software will be provided). Must have reliable Internet connection or ability to get one. Token payment will be made to editor -- negotiable. Editor will be required to gather information and layout pages on provided template. Publisher will maintain mailing database, advertising, shipping, and production. Write or call Frank Swygert c/o FARNA Systems (mailing and e-mail address below), 912-328-7859 5-10pm EST weekdays, 9am-10pm EST weekends. All applicants considered.

So a few people panicked. Don't worry, "the world of 68' micros" will keep coming even if someone doesn't apply for the position advertised above. Yes, I'm still looking. I'm considering dropping the price to $20 per year and the frequency to quarterly (those already with subscriptions will get an extra issue -- no one will lose anything!). That will keep the magazine going a couple more years and give me ample time to actually do all the necessary work. Will this cause a lot of you to drop your subscriptions? I hope not. You are in no danger of losing any money -- you never have been. I keep all magazine money in a separate account so that if I did have to suddenly quit publishing I could refund any remaining funds. As long as the magazine IS being published, no refunds will be made. And I still intend to publish through volume seven (until July 2000) as long as there are enough subscribers to make it worth while (at least 75-80 -- still have about twice that number). Let me know what you think!

You know, I got to thinking. Maybe no one wants to edit the magazine while I am still over the operation. Does anyone feel more like taking it over altogether? I would be willing to discuss that as a possibility. If interested in taking over publication and production of 268'm, just get in touch with me and we'll explore some options.

Well, with spring here and more people getting outside and away from their computers, there haven't been many letters. So no letters column this time around. Don't forget me! A letter or two for next issue would be nice. Ask a question, let me know what you are still doing with your CoCo or 68K OS-9 machine, etc.

I know this issue is late -- later than I'd like. My military unit has had some deployments, meaning we have been short handed, and I've had to work some overtime. On top of that, we're getting ready for a major "test" of the unit's capabilities early next year. That means week long "war games" with lots of overtime. Getting ready for one of those now, and another is scheduled for August and November. I should be able to work around all this and keep issues from being so late, but they will occasionally be late again. Sorry for the inconvenience, but this is a part time, one man operation!

Until next issue, keep your CoCo's alive and kicking!

A Quick Letter for ColorZap93

I just went through an issue with Jeff and someone I sent Colorzap-93 to. The program will only work on the Coco3 emulator if the "horizontal interrupt mode" is engaged. This can be done with two options in the latest version of the emulator.

You might want to add an editorial comment about this. I have since made a minor code change as a work-around. Look at the start of the source code where I time the horizontal interrupt to determine the clock speed. That can be changed to count the vertical interrupt if a 16 bit register is used to do the counting.

Robert Gault

Thanks for the tip Robert! Now people with the emulator can make the change or run in the proper mode.

I'll add that anyone wishing the program on disk can send $5 to cover copying, shipping and handling. But if you want to learn assembly, enter the program into your assembler and study what each segment of the well commented listing does.

the world of 68' micros

Publisher:
FARNA Systems PB
P.O. Box 321
Warner Robins, GA 31099-0321

Editor:
Francis (Frank) G. Swygert

Subscriptions:
US/Mexico: $24 per year
Canada: $30 per year
Overseas: $50 per year (airmail)
Back and single issues are cover price. Overseas add $3.00 one issue, $5.00 two or more for airmail delivery.

The publisher is available via e-mail
dsrtfox@delphi.com

Advertising Rates:
Contact publisher. We have scales to suit every type of business. Special rates for entrepreneurs and "cottage" businesses.

Contributions:
All contributions welcome. Submission constitutes warranty on part of the author that the work is original unless otherwise specified. Publisher reserves the right to edit or reject material without explanation. Editing will be limited to corrections and fitting available space. Authors retain copyright. Submission gives publisher first publication rights and right to reprint in any form with credit given author.

General Information:
Current publication frequency is bimonthly. Frequency and prices subject to change without notice. All opinions expressed herein are those of the individual authors, not necessarily of the publisher. No warranty as to the suitability or operation of any software or hardware modifications is given nor implied under any circumstances. Use of any information in this publication is entirely at the discretion and responsibility of the reader.

Changing Directories

A standalone "cd" utility

Mark Heilpern

Here is a program I wrote in "c" quite a while back to implement a stand-alone 'cd' utility. It gets around the memory protection problem by calling _os_permit() to get access. For this to work the program must execute as super-user (group 0). If you will not always run this as super-user you must modify the code somewhat (make the module owned by group 0 and toss in a _os_setuid() to change yourself to group 0 early in the program). If you have no MMU or are not running the SSM extension there is no memory protection.

```
/*** This is the first of 2 files ***/
#include <types.h>
#include <stdio.h>
#include <process.h>
#include <errno.h>
#include <cglob.h>
#include <modes.h>
error_code find_proc_desc(process_id, procid **);
main(u_int32 argc, char **argv)
{
  char *pathname = argv[1];
  u_int32 mode = S_IREAD;
/* change data directory */
  procid *me, *dad;
 /* if no directory was specified, check for default */
  if (argc==1) pathname = (char*)getenv("HOME");
  if (pathname==NULL) exit(E_BPNAM);
/* check for execution directory change request */
  if (argc>2)
  {
    if (!strcmp(argv[1],"-x")) pathname = argv[2];
     mode = S_IEXEC;
/* change execution directory */
  }
/* do the directory change */
  errno = _os_chdir(pathname,mode);
  if (errno) exit(errno);
 /* find my process descriptor */
  errno = find_proc_desc(_procid, &me);
  if (errno) exit(errno);
/*find my parent's process descriptor */
  errno = find_proc_desc(me->_pid, &dad);
  if (errno) exit(errno);
/*copy over the directory information */
  errno = _os_cpymem(_procid,&me->_dio,&dad->_dio,DEFIOSIZE);
  if (errno) exit(errno);
/* and exit */
  exit(0);
}

/*** the next file is for the find_proc_desc() function ***/
#include <process.h>
#include <sysglob.h>
 /* to get system globals */
#include <stddef.h>
/* for 'offsetof() macro */
#include <modes.h>
 /* for permit access modes */
#include <errno.h>
extern process_id _procid;
/* my id */
/*
** Usage:
**    process_id proc_id;
**    procid *proc_desc;
**      errno = find_proc_desc (proc_id,&proc_desc);
*/
error_code find_proc_desc(process_id proc_id,
  procid **proc_desc)
{
  u_int32 *ptab;
  u_int32 size;

  /* first, find the system's process */
  /* database table */
    (void)_os_getsys((offsetof (sysglobs,d_prcdbt)),sizeof(u_int32*),
(glob_buff*)&ptab);

  /* get access to this memory region */
  /* number of bytes we need access to */
  /* (size) is the process id of interest, */
  /* times size of each pointer entry (4) */
  /* plus the size of one entry (4) */
  size = (proc_id+1)*4;
  errno = _os_permit(ptab,size,S_IREAD,_procid);
  if (errno) return(errno);
 /* got the table. lets index into it */
  *proc_desc = (procid*)ptab[proc_id];
 /* finally, get access to that memory */
  (void)_os_permit(*proc_desc,sizeof(procid),
     S_IREAD|S_IWRITE,_procid);
 /* note, don't need error checking on */
 /*the last _os_permit(),since the call */
 /*should only fail if not running */
 /*as super user. since the first permit */
 /*call worked, we must be a SU */

  return(0);
}
```

Questions? I can be reached via e-mail at: heilpern@microware.com. If you don't have e-mail access, feel free to write the editor in reference to this article

Color Computer 3 Consumer Information

Frank Swygert & "Others"

Editor: *Rather than edit the following, I printed as it was originally written five to six years ago. At the time, the Intel 386SX/16MHz processor was the popular choice for home computers, with 286 models between 12 and 16 MHz the most numerous. This was edited from various sources.*

I. WHAT IS A COLOR COMPUTER?

The Tandy Color Computer 3 is a very inexpensive yet powerful computer for the home or small business. The original Color Computer, introduced in 1980, boasted eight colors at a time when all other Radio Shack computers in the TRS-80 line were monochrome (one color -- green or amber on a black background), hence the name "Color Computer". Unfortunately, the name has a toylike connotation. The Color Computer 3 is not, however, a toy. It can do all those things generally expected of personal computers. Technically, it compares quite favorably with computers priced much higher.

The Color Computer is probably the most versatile personal computer on the market. In its simplest form, it can be used with a TV set and Nintendo-like cartridges. In its most complex form, it can be configured with a multisync color monitor, a hard drive, and run several programs at the same time with the OS-9 operating system. The Color Computer uses standard peripherals (except monitors) such as serial printers, floppy disk drives (360K and 720K), hard disk drives, and modems. It offers an amazing combination of simplicity, power, and versatility.

II. WHAT CAN IT DO?

The Color Computer will not run software designed for other personal computers (i.e., it is not IBM- or Apple-compatible, although the BASIC computer language that is included with the standard Color Computer 3 is nearly identical to the GW-BASIC used by IBM-compatible computers). Nevertheless, the software necessary for doing all tasks commonly done on personal computers is available for the Color Computer 3. Professional quality programs exist for many varied applications, including the following:

WORD PROCESSING programs feature index and table of contents generation, mail merge, editing in multiple windows, programmable macros, spelling and punctuation checkers, with up to 160 pages in memory at one time (512K machine).

DATABASE programs for both general and specific needs (e.g., recipes).

SPREADSHEET programs offer up to 512 columns by 1024 rows, with graphics.

DESKTOP PUBLISHING programs include a tremendous variety of fonts and clip art, for use with any printer.

GRAPHIC DESIGN programs allow the Color Computer user to simply and inexpensively create custom title screens for home or business videos, or "slide shows" for business presentations.

MUSIC programs support MIDI-equipped music synthesizers.

TELECOMMUNICATIONS programs support all major protocols for going "on-line", and include VT-100 and VT-52 mainframe terminal emulations. One can get on the Internet through on-line services that still use a "shell" account (such as Delphi), but there are no graphical web browsers available. Electronic mail, broker services, travel services, information services, etc., are available through Delphi, which still has a text interface. Internet "use-net" groups are also available. While the text interface doesn't offer all the "pretty pictures" of the graphical web browsers, it has the advantage of being simple and fast.

EDUCATION programs range from learning the ABC's to calculus.

COMPUTER LANGUAGES available for the Color Computer include C, Pascal, LOGO, BASIC-09, and FORTH09.

Simplified word processing, file, and spreadsheet applications are available in cartridge form and are both less expensive and easier for the beginner to use than similar programs for other computers. Most software for the Color Computer is generally less expensive (sometimes quite a bit less) than similar software for other computers. In addition to standard software applications, the Color Computer 3 has built-in scientific and foreign language symbols, and arithmetic, trigonometric, and logarithmic math functions.

The Color Computer 3 can utilize the OS-9 disk operating system (optional), which is probably the most powerful operating system available for any personal computer. OS-9 is based on UNIX (used on large mainframe computers). It is the operating system used by NASA when communicating with satellites and the space shuttle. It transforms the Color Computer into a true multitasking system, which simply means it is capable of running several programs on the screen at the same time, in independent "windows", like a $2000 IBM PS/2. "Multi-Vue" is a program that makes the power of OS-9 readily accessible through the use of "point-and-click" icons (pictures), much like the Macintosh.

III. HOW DOES IT COMPARE?

The heart of the Color Computer is a so-called "eight-bit" microprocessor, the Motorola 6809E, which is a more powerful microprocessor than the 6502 used in the Apple II's and the Commodore 64/128. The 6809 possesses several sixteen-bit registers used for mathematical, logical, and graphics operations, giving it some of the power of sixteen-bit computers such as the Commodore Amiga, the Macintosh, and the IBM AT.

The maximum clock speed of the Color Computer is 1.788 MHz, almost twice as fast as an Apple IIe that sells for five times as much! Note that clock speed is NOT a good measure of computer speed, as the efficiency of the chip is more important. The Color Computer 3's exclusive GIME (Graphics Interrupt Memory Enhancement) chip, and the powerful OS-9 disk operating system, working together, allow the Color Computer 3 to surpass the speed of a typical IBM PC in many benchmark tests. Indeed, at its intro-

duction, Tandy officials demonstrated the Color Computer 3 beating Tandy's own 1000 SX running at 7.16 MHz. The Color Computer 3 is fast enough for any home or small business needs. It compares favorably to an Intel 80286 processor running at 12MHz. Remember, the Windows graphical user interface requires a lot of power and memory. If you've been around Intel machines for a long time, you should remember that it didn't take a lot of power to do most common tasks until Windows came along!

From the factory the Color Computer 3 comes with 128K of memory. This can be inexpensively increased to 512K with Tandy or third-party cards, and up to 1 megabyte with a third-party upgrade. Most available programs require only 128K, although some of the latest and larger programs require 512K of memory.

Screen resolution in the text mode may be either 40 columns by 24 lines or 80 columns by 24 lines. Screen resolution in the graphics mode may be either 320 dots horizontally by 192 dots vertically (320x192, with sixteen colors on the screen), or 640x192 (with 4 pure colors on the screen). Both text and graphics high resolution modes can be increased by software to 80x28 or 640x225 (12.5% more dots per screen than IBM CGA resolution of 640x200). In the highest resolution graphics mode, the four pure colors may be combined to form even more colors. Additionally, a total of 64 pure colors are available, which, through a technique called "palette-switching", provides animation abilities not possible on low-cost IBM-compatibles (for example, a flickering fire). All lower resolution graphics and text modes of the earlier Color Computers are also supported so that older programs will work. The large characters of the older 32x16 text screen are easy on the eyes of senior citizens or others with visual impairment.

The Color Computer 3 may be used with a TV set (color or b&w), a composite monitor (color or monochrome), or an analog RGB monitor, or even all three simultaneously! The RGB monitor is required for the high resolution text modes, but most programs support both TV (composite) and RGB modes. Other than the monitor, the Color Computer is compatible with IBM-standard peripherals such as floppy disk drives and disks (both 5.25 inch and 3.5 inch), hard disk drives, modems, and, with an inexpensive "serial-to-parallel converter", printers (even laser printers). Tandy 1000 series joysticks, mice, and printers plug right in. More exotic peripherals such as video digitizers, MIDI interfaces (for use with music keyboards), and sophisticated voice synthesizers are also available. Use of a cassette recorder is supported as a very inexpensive and simple alternative to a disk drive.

IV. WHY BUY A COLOR COMPUTER?

If you have a need for a specific program that is not available for the Color Computer, or you would like or need to maintain compatibility with computers at work, buy a computer that meets that need. HOWEVER, if you simply need "a computer" for writing a novel, tracking the stock market, putting out a Cub Scout newsletter, or predicting the next eclipse of the moon, the Color Computer 3 will fill the bill admirably. Consider these advantages:

LOW COST: A basic system consisting of computer, disk drive, and monochrome monitor is only $400. An absolute beginner, supplying his or her own TV and cassette recorder, could get started for only $100!

Tips for new users eager to get started

1. NEVER, EVER plug or unplug anything into the large side port while the computer is turned on! If you do, you could blow the processor. This isn't an expensive part, but it is tedious to replace. A 40 pin chip has to be desoldered and removed from the circuit board, a socket soldered in, and a new processor put in the socket. A TV shop would charge about $30 to pull the original processor and put in a socket, and a processor is about $20.

2. Game cartridges (program Paks) plug into the side port. Turn the computer off, plug in the pak, then turn it on. These games all self-start. Some have instruction screens, others you need the book for or will need to experiment with. Applications came in cartridges also, not just games.

3. Some games and such will ask what kind of monitor you have. Select either TV or Composite (Color) monitor if you have a TV or monitor that plugs into round connectors on the back of the CoCo. The RGB monitor is special for the CoCo3. They aren't real expensive, used ones are under $100. Multi-sync monitors work, but have to sync down to 15.75KHz for the CoCo. This is the same as the old IBM compatible "CGA" monitors. NEC Multi-Sync and Multi-Sync II monitros work fine.

4. Games on disk are pretty easy. With the disk drive controller plugged into the computer and the disk drive power on, put a disk in with the label side up. Now type DIR on the computer and press the ENTER key. You will have a directory of what is on the disk (that's why DIR... short for DIRectory).

5. Games or programs that end in .BIN are binary or machine language programs. To start one of them, type:
LOADM"PROGRAM <ENTER>
PROGRAM will be the name before the BIN, <ENTER> means press the ENTER key. The game will start. Some you end by pressing the red <BREAK> key. Others you have to press the RESET button on the back, next to the power button. If that doesn't end it, hold the <CTRL> and <ALT> keys down and press the RESET button. You will get a picture on the screen of three of the programmers who worked on the CoCo at Tandy. Press the RESET button again and the game will end. <CTRL> <ALT> RESET clears everything from memory. You can of course turn the power off, but if you do, count to FIVE before turning the power back on. It doesn't like sudden off/on cycles, could blow the processor!!

6. Games of programs that end in .BAS are BASIC programs written in the BASIC computer language. The CoCo has BASIC built in, so all you have to do is type:
RUN"PROGRAM <ENTER>
PROGRAM will be the name before the BAS, <ENTER> means press the ENTER key. The game will start.
All BASIC games should end by pressing the <BREAK> key, but if it doesn't do the same as with BIN programs.

7. You will need a few blank disks for later. What you want is 5.25" DOUBLE DENSITY disks, or 360K disks. DO NOT get 5.25" HIGH DENSITY disks (1.2MB), as they won't work right. Any Radio Shack will be able to get disks, and computer stores MIGHT have some.

continued on page 21

Color Computer Disk Basic Disk Structure

Robert Gault

And a binary disk editor -- ColorZap 93 -- to boot!

When Tandy finally marketed a disk system for the Color Computer, it did all users a big favor by releasing complete details on disk structure under Disk Basic. This made it possible for users to write some very interesting programs for the disk system and even find ways to overcome bugs present in DOS version 1.0. Happily Tandy and Microware were just as forthcoming when OS-9 was released. The complete details on disk structure under OS-9, different from Disk Basic, were documented.

Among the first programs written for the Coco disk systems were disk editors. These programs made it possible to read raw data from a disk, modify it if desired, and write it back to a disk ignoring the directory and file structure. This permitted users to repair the trashed disks, directories, and files which sooner or later happen to everyone.

My favorite disk editor for OS-9 is dEd by Doug DeMartinis. For some time I wanted an equivalent program for Disk Basic and finally wrote it myself. The source code for this Disk Basic look and workalike to dEd is part of this article. It can be compiled without any changes by my patch to Tandy's EDTASM (which I sell as EDTASM6309) or any other assembler that can use lower case, local labels, and multiple FCBs. Readers who have stock EDTASM can still assemble the program by replacing all labels of the form a@ with standard labels and using upper case for all source code. Replace any FCB 1,2,3 statements with separate FCB and FDB lines as needed. However, before looking at the source code let's see how Disk Basic organizes a disk.

Tracks, Sectors, and Granules

Regardless of your computer or disk operating system, disks must be formatted before use. Formatting is the process that takes a blank disk and partitions it into pieces that can be used by a computer. Disk Basic separates the disk into 35 pieces called tracks that are numbered 0-34. Each track is about 6,250 bytes (8 bit words) of which 6,084 are divided into 18 pieces called sectors, numbered 1-18, while the rest are system control bytes. Each sector contains 338 bytes where 256 bytes are for data and the rest are for system control.

Tandy describes the system control bytes in detail in the "Owners Manual & Programming Guide" for Disk Basic but this can be ignored by all but the most inveterate hackers. However, all users should know how the data portion of the disk is organized. Remember the data is stored in 256 bytes per sector, 18 sectors per track, and 35 tracks per disk side. This is a total of 161,280 data bytes per disk side.

Coco Disk Basic reserves track 17 for the directory. You can think of the directory as a special file that stores the names, location, and file types of all files on the disk side. The other 34 tracks are divided into 68 pieces called granules. Each granule is 2,304 bytes long or 9 sectors and there are two granules per track. You may be saying, "If the tracks have already been divided into sectors, why are they also divided into granules?"

Sectors are the low level structure of a disk, while granules are the low level structure of Disk Basic files. Putting it differently, the minimum space that a file can reserve under Disk Basic is 2,304 bytes even if the file is one byte long. This minimum reserved space is known as a granule in Disk Basic or a cluster under OS-9 where it can be as small as a 256 byte sector. Since the minimum space a Disk Basic file can reserve is one granule, it is easy to see that the maximum number of files per disk side is 68, the total number of granules.

Why are the Disk Basic clusters larger than OS-9 clusters? There is a trade off between wasted space and the possibility of file fragmentation. Large clusters can waste disk space but they help prevent the sectors of a file from being scattered all over a disk. Tandy made an arbitrary choice for large cluster size.

Directory and File Allocation Table (FAT)

I have already said that track 17 is reserved for the directory, the index for all files on the disk. In Color Disk Basic, the directory track has the following structure: sector #1 not used, sector #2 file allocation table (FAT), sectors #3-11 directory entries, sectors #12-18 not used. The FAT only uses the first 68 bytes of sector #2, corresponding to the 68 granules on the disk. The FAT is a map of the granules used by each file with the byte number equivalent to the granule used. If a FAT byte has the value $FF then it is not in use. A FAT byte value from 0-$43 points to the next granule in a file. A FAT byte value of $C0-$C9 indicates the number of sectors in the last granule.

If you have been following this closely, you may be asking, "Why have a FAT value of $C0?" This is a special case for zero length files that still are assigned one granule; $C0 indicates that no sectors were used. You may also have been wondering why Disk Basic does not use a 40 track disk. Clearly there is sufficient room in both the FAT and directory entry area to cover 78 granules worth of files. This unfortunately, is an example of Tandy marketing expertise. The disk structure can support 40 tracks, but Disk Basic software cannot. It is simple to patch Disk Basic to format and access 40 tracks per side but the file system still cannot make use of the extra tracks. So much for 40 track disks under Disk Basic, let's get back to the directory structure.

Each directory entry uses 32 bytes. The first 8 bytes (0-7) contain the left justified file name. The next 3 bytes (8-10) contain the extension (ex. .BAS). Byte 11 indicates the file type: 0 = BASIC program, 1 = BASIC data file, 2 = machine language program, 3 = text editor source file. Byte 12 indicates whether the file is ASCII ($FF) or binary format ($00.) You may remember that Basic programs are normally saved to disk in tokenized format. They can be saved in ASCII text format by the command SAVE"filename",A. Byte 13 is the number of the first granule in the file and bytes 14-15 indicate the number of bytes used in the last sector of the file. Finally, bytes 16-31 were reserved for future use and must be all zeros for compatibility with Tandy's EDTASM that does use these bytes.

The directory entry section starts with all bytes having the value $FF. As entries are added, the table grows and sooner or later some files will be deleted. When this happens, the first byte of the entry for the deleted file is set to $00. The next new file is added into this vacated slot.

Why a Disk Editor?

There are many reasons for using disk editors ranging from simple curiosity, to disk repair, to breaking copy protection schemes. Have you ever deleted a file and immediately had the sickening feeling of erasing the only copy of a 40 hour project? With a good disk editor, you stand a very good chance of being able to recover the file. This is because the file still exists on the disk until the space is reused. What has been lost is the chain of granule entries in the FAT and the first letter of the file name in the directory. If you change the first name byte from $00 to whatever it was and reenter the FAT information, your file will be recovered. You can find the first file granule by looking at the directory entry, to find the rest you must look at each granule on the disk for identifying text or code.

Colorzap-93 is a machine language disk editor for Coco3 only. With it you can select for examination drives, granules, tracks, or sectors and modify any byte. You can search a disk for a

pattern of bytes either hexadecimal or ASCII. One powerful option is the ability to link to any file on the disk and scroll through the file without having to know where the file sectors are located. Information displayed on screen can be sent to a printer. For those of you who use OS-9, Colorzap-93 looks and works like dEd so you only have to remember one set of commands.

Colorzap-93 works in an 80 column screen with a fast clock setting. However, the program tests your original clock speed and uses that speed for all disk I/O. The original clock speed is reset when you exit the program. Colorzap-93 supports DOS1.0, DOS1.1, and RGBDOS. The program has a repeat key function, so holding down an arrow key permits rapid scrolling through sectors. The program auto starts on loading with the first screen displayed being the instructions.

The Source Code

There are two ASM files used with Colorzap-93. There is a short file EDTDEFS that is a list of equates (equ) which I use with several of my programs. The main file, COLORZAP, is some 1700 lines long. Most of the critical lines have comments and I will not cover the code further except to point out several interesting routines that you may want to use in your own programs.

Lines 1160 - 1320 determine whether the CoCo is running at 1 or 2 MHz by counting loop cycles between interrupts. Lines 1730 - 1950 determine what screen colors are currently in use so that the program can use reverse colored letters regardless of the user's preference. Lines 16610 - 16990 are a repeat keys routine based on code by Roger Schrag. Lines 17020 - 17030 make the program auto-starting.

You will notice that whenever possible, I make calls to the Basic ROM routines. This makes the program much shorter than it otherwise would be. I am therefore indebted to the information contained in the Spectral Associates, "Basic Unravelled" series.

If you have questions about the code or cannot get it to compile and run, send questions by e-mail to:

robert.gault@ worldnet.att.net

```
00100 * Basic entry points for Coco3 DOS1.1
00110    title   EDTDEFS
00120
00130 cls   equ   $F6E0
00140 wdth32 equ   $F652
00150 wdth40 equ   $F65C
00160 wdth80 equ   $F679
00170 locate  equ   $F8F7  enter with reg.a=column reg.b=row
00180 printS  equ   $B99C
00190 decout  equ   $BDCC send # in reg.D as ASCII
00200 scrprt  equ   $A30A
00210 dskcon equ   $C004  POINTER TO DSKCON ROUTINE
00220 trkzro   equ   $D7B8
00230 getchr  equ   $A1B1  blink while waiting
00240 ikeyim  equ   $87   in key image
00250 waitky   equ   $ADFB  wait for key no blink; go to Basic on break
00260 hedtk0  equ   0
00270 read equ   2
00280 write equ   3
00290 charad  equ   $A6
00300 getnch  equ   $9F
00310 getcch  equ   $A5
00320 binval   equ   $2B
00330 linbuf   equ   $2DC   Basic line buffer
00340 CR   equ   $0D
00350 LF   equ   $0A
00360 blk   equ   '
00370 colon   equ   ':
00380 bkspc   equ   $08
00390 zero equ   $8a   two bytes are always fdb 0


00000    title   COLORZAP-93
00010 * (c) by Robert Gault September 1993; VR. 1.6
00020 * Full ml version of a disk editor program
00030 * Emulates dEd [by Doug DeMartinis (c) 1987] from the OS-9 world.
00040 * 9-29-93 Seek, Edit, Write, Find, Next, Copy, Repeat keys
00050 * 10-10-93 Link, Unlink, and correction needed to find, next, copy, etc.
00060 * 10-15-93 When linked, last sector stops printing at last byte.
00070 * 10-20-93 Corrected bug in link/edit: bad char does not leave edit
00080 *    Down arrow roll-over in last sector corrected.
00090 * 10-21-93 Added adjustable max values for track/sector.
00100 * 11-02-93 Corrected spelling of "hexadecimal".
00110 * 12-28-93 Handle incorrect file structure; ie. files where last sector
00120 *        contain zero bytes. Adjust maxgrn when adjusting maxtrk &
00130 *        maxsec.
00140 * 3-3-94   Added info to help screen.
00150 * 4-6-94   Changed repeat keys to K and records to R
00160 * 5-6-94   Added (P) screen dump to printer.
00170 * 11-2-95  Corrected error in xitopn routine which had an incorrect
00180 *        error trap. Added auto start routine.
00190
00200    org   0
00210 fgetnm rmb   2  get file name offset
00220 fopen   rmb   2  open file offset
00230 fstfcb   rmb   2  set file FCB offset
00240 fget  rmb   2  get record from file offset
00250 flof   rmb   2  get length of file
00260 fclose   rmb   2  close file offset
00270 fdir   rmb   2
00280
00290    org   $E00
00300 start bra    beginprogram has fixed exec loaction
00310 data equ   *
00320 drivermb   1  working values
00330 trackrmb   1
00340 sector   rmb   1
00350 gran rmb   1
00360 recnumrmb   2  record number of open file; max=612
00370 lof    rmb   2  length of file
00380 lstsec   rmb   1  last sector flag; 0=not FF= last sector
00390 fcblst   rmb   2  bytes in last sector+$ee; points to buffer
00400 linhdr   rmb   1  counter; holds $00,$10,$20,...,$F0
00410 color    rmb   1  0=normal; FF=reversed
00420 stndcl   rmb   1  normal attributes used by program
00430 revrcl   rmb   1  reverse color attrs
00440 hexflg   rmb   1  0=hex; FF=ascii
00450 mtcflg   rmb   1  0=no match; FF=match   used by next; set by find
00460 splits   rmb   1  0=no split; FF=split match across sector boundary
00470 mtctrk   rmb   2  this holds track & sector of last find
00480 frcnum rmb   2  this hold record number of linked match
00490 fndloc   rmb   2  find offset in buffer
00500 endmtc rmb   2  end of current match data in target buffer
00510 hexloc   rmb   1  x screen location; hex. table
00520 rownum    rmb   1   y screen location
00530 ascloc   rmb   1  x screen location; ascii table
00540 io1    rmb   1  temp. i/o storage
00550 io2    rmb   1
00560 repflg   rmb   1  0=repeat key not installed FF=installed
00570 cpyflg   rmb   1  copy active flag 0=no FF=yes
00580 drvmax rmb   1  filled by program; varies with 35-40 tk system
00590 maxtrk   rmb   1
00600 maxgrn rmb   1
00610 maxsecrmb   1
00620 dos   rmb   1   indicates DOS version
00630          * 0=DOS1.0, 1=DOS1.1, 2=RGB-DOS, 3=unknown
00640 drvflg   rmb   1  FF=drive set 0=drive not selected
00650 opnflg   rmb   1  file open flag; 0=none FF=open
00660 enddat equ   *  what follows does not get erased
00670 clock    rmb   1  0=slow 1=fast
00680
00690 ************************************************************************
00700 * Customize Colorzap-93 by changing the DOS jump tables.      *
00710 * Addition of functions to the table Must be accompanied by   *
00720 * simultaneous additions to the first RMB table above.       *
00730 ************************************************
00740
00750 dos10   fdb    $c8a4   get file name
00760    fdb    $c468   open file
00770    fdb    $c808   point to fcb
00780    fdb    $c2ccget record
00790    fdb    $cd5d   get LOF
00800    fdb    $ca3b   close
00810    fdb    $cbd2   dir
00820 dos11   fdb    $c952   get file name
00830    fdb    $c48d   open
00840    fdb    $c838   point to file fcb
00850    fdb    $c2e6   get record; reg.D=record number
00860    fdb    $ce39   get LOF
00870    fdb    $cae9   close all files
00880    fdb    $ccacdir reg.B=drive #;
                $eb = drive #
00890
00900 * system equates
00910 iobffr   equ   $989 i/o buffer for  find/next
00920 eiobuf   equ   iobffr+$100
00930 mtctrg   equ   $d00 match characters stored
                here; target
```

```
00940 tmpbuf equ $1da temporary buffer; 256 bytes
00950 secmax equ 18max. sector value; min.=1
00960 * table positions are 0 - n
00970 hextbl equ 5*$100+3 x=5; y=3
00980 asctbl equ 58x=58
00990 hexcel equ 3 size (in spaces) of a single
                table byte
01000
01010 * Include many standard defs..based on
        ROM Basic
01020 include EDTDEFS
01030 title COLORZAP-93
01040 page
01050
01060 begin ldx$ffffe
01070 cmpx #$8c1b
01080 beq b@
01090 leax a@-1,pcr
01100 jmp printS
01110 a@ fcc /SORRY! THIS PROGRAM IS FOR/
01120 fcb CR
01130 fcc /THE COCO3 ONLY!!/
01140 fcb CR,0
01150 b@ jsr wdth80
01160 orcc #$50 determine system clock rate
01170 clra
01180 ldb $ff00
01190 c@ ldb $ff01
01200 bpl c@
01210 d@ ldb $ff00
01220 ldb $ff01
01230 bmi d@
01240 e@ inca
01250 ldb $ff01
01260 bpl e@
01270 andcc #$af
01280 clrb
01290 cmpa#8
01300 blo sclock
01310 incb
01320 sclock stb clock
01330 stb $ffd9 set fast clock
01340
01350 clr$71 set for cold restart
01360 ldx#data
01370 ldb #enddat-data
01380 clra
01390 a@ sta ,x+set all data to 0
01400 decb
01410 bne a@
01420 ldx#iobffr
01430 stx$EE set drive buffer to FCB buf
                #1 location
01440 lda #secmax
01450 sta maxsec
01460 ldx#$322 max drive & track; 4 drives
                35 tracks
01470 stxdrvmax
01480 lda #$43 max gran; 35 track system
01490 sta maxgrn
01500 ldx$c002 pointer to disk basic
01510 cmpx #$2004
01520 beq init
01530 inc dos at least dos1.1
01540 ldx$c00a DOS
01550 cmpx #$df00
01560 beq init
01570 inc dos at least RGB-DOS
01580 * ldx#$2243 probably RGB system; 35 tracks
01590 * stxmaxtrk
01600 ldx[$d936] alt. RGB-DOS DSKCON entry
01610 cmpx #$3476 pshs d,x,y,u
01620 beq irgb
01630 inc dos
01640 bra init
01650 irgb lda $150 read max drive
01660 sta drvmax
01670 init inc sector sector can't be 0
01680 ldx#$c58f console in
01690 tst dos
01700 beq init2
01710 ldx#$c5bc
01720 init2 stx$16b
01730 lda $FE08 current attributes
01740 anda #%00111111
01750 tfr a,b
01760 anda #%00111000 keep foreground
01770 lsra normalize 0-7
01780 lsra
01790 lsra
01800 adda #8 foreground palettes start at 8-15
01810 andb #%00000111 keep background
01820 ldx#$FFB0 start of palettes
01830 lda a,xget foreground color
01840 anda #%00111111
01850 ldb b,xget background color
01860 andb #%00111111
01870 sta 14,x attr 6; store palette colors
01880 stb 15,x attr 7
01890 sta 6,x attr ,6
01900 stb 7,x attr ,7
01910 lda #%00110111 attr 6,7 normal; attr 7,6
                reverse color
01920 sta $FE08 set new attributes
01930 sta stndclkeep an image
01940 lda #%00111110 reverse color attributes
01950 sta revrcl keep an image
01960 ldxzero remove ON BRK and ON ERR
01970 stx$fe0c ON BRK
01980 stx$fe0e ON ERR
01990 leax drverr,pcr set new error driver
02000 lda #$7e
02010 sta $191
02020 stx$192
02030 lbsr ckdos
02040 jsr [fclose,x]
02050 sta $ffd9 fast clock
02060 lds #$7ffe set stack
02070 ldd #2 set two FCBs; #1 not active
02080 stb $95b active FCBs = 2
02090 ldx$928 point to FCB #1
02100 sta ,x closed
02110 decb =1
02120 std 7,xrecord number = 1; willprint as 0
02130 leax mcmd,pcr set return address
02140 pshs x
02150 lbra help display help screen then
                go to command
02160
02170 drverr leas 2,s pop return; entered via JSR
02180 cmpb#54 bad record
02190 beq xerr
02200 err2 cmpb#52 file not found
02210 bne xerr
02220 leax NEmsg-1,pcr
02230 jsr printS
02240 jsr getchr
02250 jsr unlink
02260 xerr lds #$7ffe
02270 bra main
02280 NEmsg fcb CR
02290 fcc /FILE does not exist!/
02300 fcb 0
02310
02320 clrkey pshs d,x speeds up arrow key
                functions; needed even with
02330 ldx#$152 repeat key routine
02340 ldd #$ff08
02350 cklp sta ,x+
02360 decb
02370 bne cklp
02380 puls d,x,pc
02390
02400 main ldx#iobffr needed because some disk
                routines change it
02410 stx$ee
02420 lbsr screen acquire data and show sector
02430 mcmd bsr wcmd print CMDS:
02440 clr cpyflg
02450 jsr getchr wait for a key press
02460 cmpa#'A
02470 blo mcmd2
02480 anda #.not.$20 make upper case
02490 mcmd2 bsr evlcmd evaluate key
02500 bcs mcmd loop if not command
02510 bsr cmdjmp execute command
02520 bcs mcmd if illegal arguments loop
                without read
02530 bsr clrkey
02540 bra main if legal arguments get new data
02550
02560 evlcmd leay cmds,pcr point tocommand table
02570 cmpa#'Q quit and
02580 beq ev0
02590 cmpa#'D drive select always available
02600 beq ev0
02610 tst drvflg all others must make sure
                drive was selected
02620 beq nodrv
02630 ev0 ldb #cmdend-cmds total # of commands
02640 ev1 cmpa,y+hunt
02650 beq ev2
02660 decb
02670 bne ev1
02680 ev3 comb indicate error
02690 rts return with carry set
02700 ev2 pshs b
02710 ldb #cmdend-cmds
02720 subb ,s+ reg.b=cmd# ( 0 to c-1)
02730 clra clear carry
02740 rts
02750 nodrv leax ndrmsg-1,pcr
02760 jsr printS
02770 jsr getchr
02780 bra ev3
02790 ndrmsg fcc /Please select a drive!
                Hit any key when ready/
02800 fcb 0
02810
02820 cmdjmp leax jmptbl,pcr
02830 lslb
02840 abx point to command
02850 jmp [,x]
02860
02870 wcmd ldd #21 col=0 row=22
02880 jsr locate
02890 leax blklin-1,pcr blank line
02900 jsr printS clear command line
02910 ldd #22
02920 jsr locate
02930 leax cmdtxt-1,pcr
02940 jmp printSprint CMDS:
02950 cmdtxt fcc /CMDS: /
02960 fcb 0
02970 cmds fcc "H/?DGTSCEFNWZQ^"
02980 fcb $0A down arrow
02990 fcc /KLU!RP/
03000 cmdend equ *
03010
03020 jmptbl equ *
03030 fdb help
03040 fdb help
03050 fdb help
03060 fdb setdrv
03070 fdb setgrn
03080 fdb settrk
03090 fdb setscS
03100 fdb copy
03110 fdb edit
03120 fdb find find string: hex or alpha numeric
03130 fdb next find next occurance
03140 fdb wrtsec write sector to disk
03150 fdb zap erase sector
03160 fdb quit return to Basic
03170 fdb secup increment sector
03180 fdb secdwn decrement sector
03190 fdb repkey
03200 fdb link
03210 fdb unlink
03220 fdb reset adjust max track & max sector
03230 fdb setscR actually set record#
03240 fdb print dump screen to printer
03250
03260
03270 blklin fcc / /40spaces
03280 fcc / /40spaces
03290 blkln2 fcc / /40spaces
03300 fcc / /40spaces
03310 fcc / /40spaces
03320 fcc / / 39spaces
03330 fcb 0
03340 twoblk fcc / / 2 spaces
03350 fcb 0
03360 betwn fcc / / 5 spaces between
                hex. & ascii tables
03370 fcb 0
03380
03390 sure leax surmsg-1,pcr
03400 jsr printS
03410 sure2 jsr getchr
03420 anda #.not.$20
03430 cmpa#'Y
03440 beq sr
03450 coma
```

```
03460 srrts
03470 surmsg fcb   CR
03480   fcc   / Write sector to disk; are you sure?/
03490   fcb   0
03500
03510 hprntS  jmp   printS
03520
03530 header pshs  y
03540   clr lstsec
03550   leax  copyrt-1,pcr
03560   bsr   hprntS
03570   leax  head1-1,pcr   drive
03580   bsr   hprntS
03590   lda   drive
03600   jsr decprt   print decimal number
03610   ldd   #12*$100+1   across, down
03620   jsr locate
03630   leax  head2-1,pcr   gran
03640   bsr   hprntS
03650   lda   gran
03660   bmi   h1 directory track does not have
                  gran value
03670   lbsr  hexprt   print hexadecimal number
03680 h1   ldd   #23*$100+1
03690   jsr locate
03700   leax  head3-1,pcr   track
03710   bsr   hprntS
03720   lda   track
03730   jsr decprt
03740   ldd   #36*$100+1
03750   jsr locate
03760   leax  head4-1,pcr   sector
03770   bsr   hprntS
03780   lda   sector
03790   jsr decprt
03800   ldd   #48*$100+1
03810   jsr locate
03820   leax  head4b-1,pcr
03830   bsr   hprntS
03840   ldx$928 point to FCB #1; recnum may
                  not = last sector
03850   ldd   7,x
03860   subd #1
03870   cmpd lof
03880   blo   h2
03890   com   lstsec
03900 h2   jsr decout
03910   ldd   #62*$100+1
03920   jsr locate
03930   leax  head4c-1,pcr
03940   bsr   hprntS
03950   ldd   lof
03960   jsr decout
03970   lda   #CR
03980   jsr scrprt
03990   bsr   flipcl  reverse colors
04000   leax  head5-1,pcr   print byte numbers
04010   jsr printS
04020   bsr   normcl  normal colors
04030   lda   #CR
04040   jsr scrprt
04050   puls  y,pc
04060
04070 normcl pshs  a,cc
04080   lda   stndcl
04090 norml2 sta   $fe08
04100   puls  a,cc,pc
04110
04120 flipcl pshs  a,cc
04130   lda   revrcl
04140   bra   norml2
04150
04160 copyrt  fcc   /              /
04170   fcc   /COLORZAP-93 (c) Sept.1993 by
                  Robert Gault VR. 1.6/
04180   fcb   CR,0
04190 head1  fcc   /DRIVE. #/
04200   fcb   0
04210 head2  fcc   /GRAN: $/
04220   fcb   0
04230 head3  fcc   /TRACK: #/
04240   fcb   0
04250 head4  fcc   /SECT: #/
04260   fcb   0
04270 head4b fcc   /RECORD #/
04280   fcb   0
04290 head4c fcc   /LOF #/
04300   fcb   0
```

page 10 the world of 68' micros

```
04310 head5  fcc   /   0 1 2 3 4 5 6 7 8 9
                        A B C D E F/
04320   fcc   /   / 6 spaces
04330   fcc   /0 2 4 6 8 A C E /
04340   fcb   0
04350 footr1  fcc   /   0 1 2 3 4 5 6 7 8 9
                        A B C D E F/
04360   fcc   /   / 6 spaces
04370   fcc   / 1 3 5 7 9 B D F/
04380   fcb   0
04390 footr2  fcb   CR
04400   fcc   /# = decimal number
                  $ = hexadecimal number/
04410   fcb   0
04420
04430 errorleax  errmsg-1,pcr
04440   jmp   printS
04450 errmsg fcb   CR
04460   fcc   'Disk I/O error! Drive not ready.'
04470   fcb   CR,0
04480 wrking  fcc   /Working .... Floppy drives
                        are slow./
04490   fcb   CR,0
04500
04510 screen  jsr clsclear screen
04520   leax  wrking-1,pcr
04530   jsr printS
04540   clr linhdrclear counter
04550   tst opnflg
04560   beq   secred
04570   ldx$928 pointer to FCBs
04580   lda   3,x
04590   sta   gran
04600   pshs  a
04610   lbsr  clcts
04620   decb
04630   puls  a
04640   bita  #1
04650   bne   noclr
04660   clrb
04670 noclr   addb  4,x
04680   stb   sector
04690   ldd   19,x  get bytes in last sector
04700   addd $ee   add buffer location
04710   std   fcblst
04720   bra   nxtln0
04730 secred  lda   #read   command for dskcon
04740   ldb   drive  tell dskcon the parameters
04750   std   $EA
04760   ldd   track
04770   std   $EC
04780   lbsr  diskon
04790   lbne  error
04800 nxtln0  ldd   zero
04810   jsr locate
04820   lbsr  header  print common header with
                  drive,gran,track,sect
04830   ldy$EE  point to dskcon buffer
04840 nxtlin  pshs  y
04850   lbsr  flipcl  flip colors
04860   lda   linhdr get row counter
04870   lbsr  hexprt   print ASCII hexadecimal
04880   lda   #colon
04890   jsr scrprt
04900   lbsr  normcl  reset colors
04910   leax  twobik-1,pcr
04920   jsr printS
04930   ldb   #16   print 16 ASCII hexadecimal bytes
04940 sloop1  lda   ,y+   print byte value
04950   tst opnflg
04960   beq   slup1b
04970   tst lstsec
04980   beq   slup1b
04990   cmpy fcblst
05000   bls   slup1b
05010   lda   #$ff
05020 slup1b  bsr   hexprt
05030   lda   #blk  print space
05040   jsr scrprt
05050   decb
05060   bne   sloop1
05070   leax  betwn-1,pcr   5 space gap
05080   jsr printS
05090   ldy,s  recover buffer pointer
05100   ldb   #16   print ASCII character or "."
05110 sloop2  lda   ,y+
05120   tst opnflg
05130   beq   slup2b
```

```
05140   tst lstsec
05150   beq   slup2b
05160   cmpy fcblst
05170   bls   slup2b
05180   lda   #$ff
05190 slup2b  bsr   ascprt
05200   decb
05210   bne   sloop2
05220   lda   #CR
05230   jsr scrprt
05240   lda   linhdr update line header; $10 per line
05250   adda #$10
05260   sta   linhdr
05270   puls  y
05280   leay  $10,y update line/buffer pointer
                  $10 per line
05290   cmpa#0 finished a sector?
05300   bne   nxtlin no?; then loop
05310   lbsr  flipcl
05320   leax  footr1-1,pcr
05330   jsr printS
05340   lbsr  normcl
05350   leax  footr2-1,pcr   message #=dec. $=hex.
05360   jmp   printS
05370
05380 ascprt  anda #$7f remove fcs bit
05390   cmpa#$7f  printer sees this as delete
05400   beq   period
05410   cmpa#blk
05420   bhs   norm
05430 period  lda   #'.exclude control codes
05440 norm    jmp   scrprt
05450
05460 hexprt  pshs  a  print binary# as ASCII
                  hexadecimal
05470   lsra     get MSN
05480   lsra
05490   lsra
05500   lsra
05510   bsr   digit
05520   puls  a
05530   anda #$F
05540 digit cmpa#9 is it a number
05550   bls   numb
05560   adda #7 must be letter; offset to letters
05570 numb    adda #'0   convert to ascii
05580   jmp   scrprt
05590
05600 decprt  tfr a,b   print byte as ASCII decimal
05610   clra
05620   jmp   decout  decimal out
05630
05640 cmdst2 ldx#linbuf+1
05650   ldd   ,x
05660   tsta     was there an entry?
05670   bne   cmd3
05680   andcc   #.not.4   no entry; indicate bad entry
05690   rts
05700 cmd3    tstb
05710   bne   hxDbin
05720   tfr a,b
05730   lda   #'0
05740
05750 hxDbin  bsr   hexbin   convert ascii hex
                  reg.D tp binary in reg.A
05760   bcs   hxDxit
05770   lsla
05780   lsla
05790   lsla
05800   lsla
05810   exg   a,b
05820   bsr   hexbin
05830   bcs   hxDxit
05840   pshs  b
05850   adda ,s+
05860   bra   notlet
05870 hxDxit  rts
05880
05890 hexbin  suba #'0   convert ascii hex. in
                  reg.A to binary in reg.A
05900   bcs   inval1
05910   cmpa#9
05920   bls   notlet
05930   suba #7
05940   cmpa#$F
05950   bhi   inval1
```

continued on page 12

operating system nine

Rick Ulland

CoCo IV ideas

Posted recently on the Internet CoCo List concerning talk about the possibility of creating a "CoCo4""

Come on people. The CoCo had it's day, it saw the light, but please get a grip and move on! There will not be a CoCo IV or V or VI or whatever. There is no money. You barely support the "little" people still giving...

ME:

I'm afraid I must agree with you to some extent. Gone are the days when a person could finance a good idea by making a few original CoCo addons. But this sort of hardware has low budget potential, in that you *don't* need the $5000 boardcutting or workstation class development system. Anyone with a pClone and $500 can make a small run of sub10 MHz parts.

Now what you do with them...if 'new' means 'take the CoCo design and tack a xyz on it' there would be no reason at all to build a CoCo4. It's original mission has been filled. For Joe User, budget computing consists of castoffs from the pClone wars. If you want stable multitasking, use linux. If you want lots of applications, use Win3.1 and reboot every few hours.

This brings us to CoCo4. Our community has a larger than normal share of hobbiests. These are the guys that used to build TV dazzlers and twist tie old teletypes together. The resulting machines weren't that useful in themselves, but they developed the techniques and people that got us (the computing public) where we are today.

But where have we gone? The fastest Pentium wonderbox is nothing more than a really big Altair. It's got huge drives, it's got wastelands of DRAM, perps to amaze the most jaded hacker. But it's still a one instruction wide path to a lone cpu -- CP/M with animated wizards. I refuse to believe this represents the ultimate in computing architecture.

But commercially viable computers aren't hackable. You pretty much run the motherboard they sold you. As long as 'commecially viable' means 'really fast Altair' they're kind of boring, so we should investigate a new paradigm that can later be scaled up in the Atlair->Wintel mold.

The solution proposed is true 'multiprocessing'. Rather than one overworked wafer laboring under Its Own Fan, a 'computer' would be a collection of cpus working towards a *possibly* common goal. This is going to require software a little beyond billyBASIC, bringing us to the CoCo3 and OS9. This alternative opsys has a smart scheduler, doesn't leak, and is already segmented in exactly the right places, with each 'process' nearly independant enough to move offboard already. And in hardware the CoCo has been using dual port DRAM (cpu, combined refresh/video) for years.

Great for the hackers involved, ignores the guy that needs a $50 upgrade path. So we've decided to stay as CoCo like as possible in the prototypes of any "CoCo IV". This way, anything useful can be drawn up as a CoCo version. Where it ends, we'll see. I've already designed a board that controls interupts through hardware, taking a big task away from the CPU under OS-9 (but pretty useless for DECB users).

But Frank recently told me about a fellow in Great Britain who has done what was discussed in Chicago a couple years ago -- well, almost. When the first seminar on the CoCo IV project was held at the Chicago CoCoFest in 1997, the general consensus was that the best way to pursue a prototype would be to make an I/O controller that took a lot of the general work tasks away from the main processor. This would plug into the side port and take care of the keyboard, interrupts, and anything else we could give it. The board would have a 6809, a PIA or two, and whatever necessary circuitry to do the jobs given it. Then OS-9 would be patched to use the added processor.

We weren't the first to get this idea! This Briton did almost the same thing with a Dragon and Dragon DOS (Tano's version of DECB) some years ago. Only he went an easier route -- let the CoCo be the I/O procesoor and the added 6809 the "main" one!

Sounds so logical it is hard to see why we didn't think of it! Since the CoCo processor is already programmed to do all the I/O functions, leave it alone! Pass the code crunching to another CPU, in this case clocked at 3MHz, and let the CoCo process the results! This chap says it works fine, and will be sending Frank some schematics and code later. Hope he comes through, I can't wait to see this stuff and start designing a board any OS-9 user would be proud to have!

ColorZap93 *(continued from page 10)*

```
05960 notlet  andcc  # not.1
05970  orcc #4
05980  rts
05990 inval1  orcc  #1
06000 hxxit rts
06010
06020 decbin  ldx#linbuf+1
06030  orcc #1
06040  lda  ,x
06050  beq  decxit
06060  ldy charad
06070  pshs y
06080  stx charad
06090  jsr $AF67
06100  puls  y
06110  sty charad
06120  tst opnflg
06130  beq  db
06140  tst cpyflg
06150  bne  db
06160  ldd  binval
06170  andcc  #.not.1
06180  rts
06190 db  tsta
06200  bne  baddrv
06210  lda  binval+1
06220  clrb
06230 decxit  rts
06240
06250 lnkmsg fcc  /Must link to file!/
06260  fcb  0
06270 unlmsg fcc  /Must unlink file!/
06280  fcb  0
06290 setdrv  tst opnflg
06300  beq  a@
06310 ulerr leax  unlmsg-1,pcr
06320 ulerr2  jsr printS
06330  jsr getchr
06340  orcc #1
06350  rts
06360 lnkerr  leax  lnkmsg-1,pcr
06370  bra  ulerr2
06380 a@  leax  drvnum-1,pcr
06390  lbsr  cmdset  print query; get answer
06400  bsr  decbin  convert to binary
06410  bcs  baddrv
06420  cmpa drvmax
06430  bhi  baddrv
06440  sta  drive
06450  clra
06460  deca
06470  sta  drvflg indicate drive selected
06480  bsr  setgrn
06490  bcs  b@
06500  bne  b@
06510  bsr  settrk
06520 b@  ldx zero
06530  stx mtctrk
06540  stx fndloc
06550  clr mtcflg
06560  rts
06570 baddrv  coma
06580  rts
06590 drvnum fcc  /Drive: #/
06600  fcb  0
06610
06620 setgrn  tst opnflg
06630  bne  ulerr
06640  leax  grnnum-1,pcr
06650  lbsr  cmdset
06660  lbsr  cmdst2
06670  bcs  nogrn
06680  bne  nogrn
06690  cmpa maxgrn
06700  bhi  baddrv
06710  sta  gran
06720  lbra  clcts
06730 nogrn  clra
06740  rts
06750 grnnum fcc  / Gran: $/
06760  fcb  0
06770
06780 settrk  tst opnflg
06790  bne  ulerr
06800  leax  trknum-1,pcr
06810  lbsr  cmdset
06820  lbsr  decbin
06830  bcs  baddrv
06840  cmpa maxtrk
06850  bhi  baddrv
06860  sta  track
06870  bsr  ssec
06880  clra
06890  rts
06900 trknum fcc  / Track: #/
06910  fcb  0
06920
06930 recmsg fcc  /RECORD #: /
06940  fcb  0
06950 setscS  tst opnflg
06960  beq  ssec
06970  lbra  ulerr
06980 x@  bra  baddrv
06990 setscR  tst opnflg
07000  bne  setrec
07010  lbra  lnkerr
07020 setrec  leax  recmsg-1,pcr
07030  lbsr  cmdset
07040  bcs  x@
07050  lbsr  decbin
07060  cmpd lof
07070  bhi  x@
07080  cmpd #0
07090  beq  x@
07100  std  recnum
07110 ssec0  ldx#iobffr
07120  stx $ee
07130  ldx $928 pointer to fcb #1
07140  stx $f1
07150  clr 15,x
07160  clr 16,x
07170  clr 17,x
07180  clr 18,x
07190  clr 6,x
07200  clr $d8  used as GET/PUT flag; 0=get
07210  lbsr  ckdos
07220  jsr [fget,x]
07230  sta  $ffd9
07240  clra
07250  rts
07260 ssec leax  secnum-1,pcr
07270  lbsr  cmdset
07280  lbsr  decbin
07290  bcs  x@
07300  cmpa #0
07310  beq  x@
07320  cmpa maxsec
07330  bhi  x@
07340  sta  sector
07350  bra  clcgrn
07360 secnum  fcc/ Sector: #/
07370  fcb  0
07380
07390 clcts lda  gran  calculate track/sector from
gran#
07400  ldb  #1 sectors start at 1
07410  bita  #1
07420  beq  a@
07430  ldb  #10  sector=10 on odd grans
07440 a@  cmpa #33  track 16
07450  bls  b@
07460  adda #2 compensate for track 17
07470 b@  lsra
07480  std  track
07490  andcc  #.not.5
07500  rts
07510
07520 clcgrn  lda  sector  calculate gran# from
track/sector
07530  clrb  even gran
07540  cmpa #9
07550  bls  a@
07560  incb  odd gran
07570 a@  lda  track
07580  cmpa #17
07590  beq  d@
07600  blo  b@
07610  deca  compensate for track 17
07620 b@  lsla  2 grans/track
07630  pshs b
07640  adda ,s+
07650 c@  sta  gran
07660  andcc  #.not.5
07670  rts
07680 d@  lda  #$FF directory no gran number
07690  bra  c@
07700
07710 quit  leax  a@-1,pcr
07720  jsr printS
07730  jsr sure2
07740  bcs  hxit
07750  lbsr  setclk
07760  jmp  [$fffe]
07770 a@  fcc  /QUIT Are you sure?/
07780  fcb  0
07790
07800 help jsr cls
07810  leax  helpms-1,pcr
07820 a@  jsr printS
07830  ldd  ,x
07840  bne  a@
07850 b@  jsr getchr
07860  beq  b@
07870  clra
07880 hxit  rts
07890
07900 helpms fcc  "H gets this message; also /
                  or ?"
07910  fcb  CR,0
07920  fcc  "Up/Down arrows move to next/
       previous sector"
07930  fcb  CR,0
07940  fcc  "D select drive number; [gran, track/
       sector]"
07950  fcb  CR,0
07960  fcc  /G select gran value/
07970  fcb  CR,0
07980  fcc  /T select track value; [sector]/
07990  fcb  CR,0
08000  fcc  /S select sector value; R record #
       if linked/
08010  fcb  CR,0
08020  fcc  /C copy current sector to  D,T,S/
08030  fcb  CR
08040  fcc  / enter each value separately with
       ENTER key/
08050  fcb  CR,0
08060  fcc  /E edit current sector/
08070  fcc  /; must Write sector to make changes
       permanent/
08080  fcb  CR,0
08090  fcc  /F find string; hex. or alphanumeric;
       case sensitive/
08100  fcb  CR
08110  fcc  /; quit search in progress with any key/
08120  fcb  CR,0
08130  fcc  /P print screen; preset BAUD for 2MHz
       from BASIC/
08140  fcb  CR,0
08150  fcc  /N next occurance of string/
08160  fcc  /; starts at last match regardless of
       current sector/
08170  fcb  CR
08180  fcc  ' no action if last find/next
          unsuccessful'
08190  fcc  /; quit searching with any key/
08200  fcb  CR,0
08210  fcc  /W write current sector to disk/
08220  fcb  CR,0
08230  fcc  /L link to disk file; 'ENTER' gives
       directory./
08240  fcb  CR,0
08250  fcc  /U unlink from disk file/
08260  fcb  CR,0
08270  fcc  /Z zap current sector with selected
       value/
08280  fcb  CR,0
08290  fcc  /K repeat key function. Use ONLY if
your ROM does not/
08300  fcc  / have built in repeats./
08310  fcb  CR,0
08320  fcc  /Q quit program for Basic/
08330  fcb  CR,0
08340  fcc  /! Adjust allowable maximum track
          and sector values /
08350  fcc  /for oddball disks;/
08360  fcb  CR
08370  fcc  / usually 34 or 39T & 18S.
          USE CAUTION!/
08380  fcb  CR,0
08390  fcc  /              /
08400  fcc  /[] indicates optional parameters/
```

```
08410   fcb   CR,0
08420   fcc   /                /
08430   fcc   /any key returns to main screen/
08440   fcb   0,0
08450
08460 secup   tst opnflg
08470   beq   a@
08480   ldd   recnum
08490   cmpd lof
08500   lbhs  baddrv
08510   addd #1
08520   bra   d@
08530 a@  ldd   track increment sector; track if
necessary
08540   cmpb maxsec
08550   beq   b@
08560   incb
08570   bra   c@
08580 b@  cmpa maxtrk
08590   lbeq  baddrv
08600   ldb   #1
08610   inca
08620 c@  std   track
08630   lbra  clcgrn
08640 secdwn tst opnflg
08650   beq   e@
08660   ldd   recnum
08670   subd #1
08680   beq   g@
08690 d@  std   recnum
08700   lbra  ssec0
08710 e@  ldd   track decrement sector; track if
necessary
08720   cmpb #1
08730   beq   f@
08740   decb
08750   bra   c@
08760 f@  tsta
08770 g@  lbeq  baddrv
08780   ldb   maxsec
08790   deca
08800   bra   c@
08810
08820 cmdset jsr printS   print command and get
answer
08830
08840 * Replacement for Basic line input. Needed
because Basic prints CR at
08850 * end of input.
08860
08870 linein  ldx#linbuf+1
08880 linin2  ldb   #1
08890 linlup  jsr $A171
08900   cmpa #bkspc
08910   bne   notbs
08920   decb
08930   beq   linein
08940   leax  -1,x
08950   bra   echo
08960 notbs   cmpa #$15 shift left arrow
08970   bne   noclin
08980 clin  decb
08990   beq   linein
09000   lda   #bkspc
09010   jsr scrprt
09020   bra   clin
09030 noclin  cmpa #3break
09040   orcc  #1
09050   beq   linxit
09060   cmpa #CR
09070   bne   inschr
09080   clra
09090 linxit pshs cc
09100   clr ,x
09110   puls  cc,pc
09120 inschr  cmpa #blk
09130   blo   linlup
09140   cmpa #'z+1
09150   bhs   linlup
09160   cmpb #250
09170   bhs   linxit
09180   sta   ,x+
09190   incb
09200 echojsr scrprt
09210   bra   linlup
09220
09230 wrtsec  ldb   drive write sector to disk
09240   stb   $EB
09250   ldx track
09260   stx $EC
09270 wsec2   lbsr  sure
09280   bcs   nowrt
09290   lda   #write
09300   sta   $EA
09310   lbsr  diskon
09320   bne   badcpy
09330 nowrt   rts
09340
09350 * Copy sector to any other sector at any drive
or track
09360
09370 copy com  cpyflg
09380   leax  cpymsg-1,pcr
09390   bsr   cmdset
09400   lbsr  decbin
09410   bcs   badcpy
09420   cmpa drvmax
09430   bhi   badcpy
09440   sta   $EB
09450   bsr   more
09460   cmpa maxtrk
09470   bhi   badcpy
09480   sta   $EC
09490   bsr   more
09500   cmpa maxsec
09510   bhi   badcpy
09520   sta   $ED
09530   bra   wsec2
09540 more    lda   #colon
09550   jsr scrprt
09560   jsr linein
09570   lbsr  decbin
09580   bcs   badcpy
09590   rts
09600 badcpy coma
09610   rts
09620 cpymsg fcc   /Enter destination Drive#<CR>
Track#<CR>  Sector#<CR>: /
09630   fcb   0
09640
09650 * Fill sector with any single character; ie. erase
sector
09660
09670 zap   leax  zapmsg-1,pcr
09680   lbsr  cmdset
09690   lbsr  cmdst2
09700   bne   badcpy
09710   bcs   badcpy
09720   ldx $EE
09730   clrb
09740 zloop   sta   ,x+   fill write buffer
09750   incb
09760   bne   zloop
09770   lbra  wrtsec
09780 zapmsg  fcc/Enter ZAP byte: $/
09790   fcb   0
09800
09810 * Find any hex. or ascii character string up to
125 hex or 250 ascii
09820 * bytes of data.
09830
09840 find  ldd   #21
09850   jsr locate
09860   leax  fndmsg-1,pcr
09870   jsr printS
09880 fndinp  ldd   #22
09890   jsr locate
09900   leax  blkln2-1,pcr
09910   jsr printS
09920   ldd   #22
09930   jsr locate
09940   leax  fndh-1,pcr
09950   tst hexflg
09960   beq   findh
09970   leax  fnds-1,pcr
09980 findhjsr printS
09990   ldy#tmpbuf
10000   lbsr  linein
10010   cmpa #3 BREAK key
10020   bne   fnd2
10030   com   hexflg
10040   bra   fndinp
10050 fnd2 cmpx #linbuf+1
10060   lbeq  badcpy  no input for find
10070   tst hexflg
10080   bne   fascii
10090 fhex ldb   ,-x
10100   lda   #'0
10110   cmpx #linbuf+1
10120   beq   h1byt
10130   lda   ,-x
10140 h1byt   lbsr  hxDbin
10150   lbcs  badfnd
10160   sta   ,y+
10170   cmpx #linbuf+1
10180   bne   fhex
10190   ldx#tmpbuf cassette buffer used as
temporary hold
10200   pshs x
10210   ldx#mtctrg
10220 h1lp lda   ,-y
10230   sta   ,x+
10240   cmpy ,s
10250   bne   h1lp
10260 fasci2  leas  2,s   yank temp data
10270   stx endmtc  save end of match data
10280   ldx zero
10290   stx mtcflg  clear match and split find
indicator
10300   stx fndloc
10310   leax  1,x
10320   stx frcnum  initialize to record #1
10330   bra   fndwds  now go get it
10340
10350 fascii  ldy#linbuf+1
10360   pshs x
10370   ldx#mtctrg
10380 h2lp lda   ,y+
10390   sta   ,x+
10400   cmpy ,s
10410   bne   h2lp
10420   bra   fasci2
10430
10440 * find the string
10450 fdwds0 ldd   zero
10460   std   fndloc
10470   tst mtcflg
10480   beq   fndwds
10490   com   splits
10500 fndwds tst opnflg
10510   beq   fnd0
10520   ldd   recnum
10530   cmpd lof
10540   bhi   fpk3
10550   pshs x,y,u
10560   lbsr  ssec0
10570   puls  x,y,u
10580   jsr $a1c1
10590   bne   fpk3
10600   ldd   recnum
10610   addd #1
10620   std   recnum
10630   bra   fp5
10640 fpk3 ldd   frcnum
10650   std   recnum
10660   lbra  ssec0
10670 fnd0 bsr   readsc
10680   jsr $a1c1   check keyboard break on any
key
10690   beq   fpk1
10700 fpk2 ldx track
10710   stx $ec
10720   clra
10730   rts
10740 fpk1 ldd   $ec  get track/sector
10750   cmpb maxsec max sector?
10760   beq   fp1
10770   incb
10780   bra   fp2
10790 fp1   cmpa maxtrk
10800   bne   fp3   end of disk; stop reading sectors
10810   lda   #$80+19
10820   sta   $ed   sector; will create illegal read
below
10830   bra   fp5
10840 fp3   ldb   #1
10850   inca
10860 fp2   std   $ec
```

```
10870 fp5  ldd   $ee  get start address of buffer
10880   addd fndloc  add last "found at" offset
10890 mtlp4  tfr d,y now try to match from new address
10900   tst mtcflg
10910   bne  mtlp2
10920 mtlp3  ldx#mtctrg  point to target buffer
10930   clr splits
10940   leau ,y  update buffer address
10950 mtlp2  cmpy #eiobuf  are we past end of
                           1 sector buffer?
10960   bhs  fdwds0  yes?; then advance 1 sector
10970   clr mtcflg   clear match flag; ie. no match
10980   lda  ,x+
10990   cmpa ,y+
11000   bne  mtlp3
11010   com  mtcflg
11020   cmpx endmtc  end of target buffer?
11030   beq  gotit
11040   bra  mtlp2
11050 badfnd  ldxzero
11060   stxmtctrk
11070   stxfndloc
11080   clr mtcflg
11090   rts
11100 readsc  lda   #2
11110 sta   $ea
11120 tst $ed
11130 bmi   bdread
11140 lbsr  diskon
11150 beq   xread
11160 bdread  lda   maxsec
11170 sta   $ed
11180 bread2  clra
11190 leas  2,s
11200 xread   rts
11210
11220 * now calculate markers for tables
11230 gotit lda   $ed
11240   anda #$7f  remove possible flag
11250   sta  $ed
11260   tfr u,d   reg.U = location in dskcon buffer
11270   subd $ee
11280   stb  fndloc+1   sector offset for find
11290   clr fndloc
11300   ldd  $ec
11310   cmpb#1
11320   beq  gt1
11330   decb
11340   tst splits
11350   beq  mark
11360   decb
11370   bra  mark
11380 gt1  deca
11390   ldb  maxsec
11400   tst splits
11410   beq  mark
11420   decb
11430 markstd   mtctrk
11440   std  track
11450   lbsr  clcgrn
11460   tst opnflg
11470   beq  mark2
11480   ldx$928
11490   ldd  7,x
11500   subd #1
11510   tst splits
11520   beq  mark3
11530   subd #1
11540 mark3  std   recnum
11550   std  frcnum
11560   lbsr  ssec0
11570 mark2  lbsr  screen  show correct sector data
11580   ldx#hextbl
11590   stxhexloc
11600   lda  #asctbl
11610   sta  ascloc
11620   ldb  fndloc+1
11630   lda  rownum
11640   deca
11650 mklp1  inca    divide reg.B by 16
11660   subb #$10
11670   bcc  mklp1
11680   sta  rownum
11690   addb #$10
11700   pshs b
11710   lda  #hexcel size
11720   mul
11730   addb hexloc
```

```
11740   stb   hexloc
11750   lda   ascloc
11760   adda ,s+
11770   sta   ascloc
11780   ldb   rownum
11790   jsr locate
11800   lbsr  flipcl
11810   jsr locate
11820   ldd   hexloc
11830   jsr locate
11840   ldd   #22
11850   jsr locate
11860   lbsr  normcl
11870   lbra  endchk
11880
11890 fndmsg fcc   /BREAK toggles hexadecimal
                   byte vrs. ASCII string/
11900   fcb   CR,0
11910 fndh fcc   /Search byte string: $/
11920   fcb   0
11930 fnds fcc   /Search character string: /
11940   fcb   0
11950 keys fcb   $c up arrow
11960   fdb   moveup
11970   fcb   $0a   down arrow
11980   fdb   movedn
11990   fcb   $09   right arrow
12000   fdb   movert
12010   fcb   $08   left arrow
12020   fdb   movelf
12030   fcb   CR
12040   fdb   endedt
12050   fcb   0
12060   fdb   edinp edit input
12070
12080 endedt clr color remove reversed color from
display
12090   lbsr  eddisp
12100   ldd   #$c5e   normalize CLEAR & up arrow
12110   stb   $a26e
12120   sta   $a27c
12130   ldx#$ed84  enable cursor
12140   stx$f812
12150 endchk orcc  #1
12160   rts
12170
12180 edtmsg fcc   /ENTER exits Edit   CLEAR =
ASCII uparrow/
12190   fcb   0
12200
12210 edit  tst opnflg
12220   beq   a@
12230   ldx$ee
12240   cmpx fcblst
12250   bne   a@
12260   tst lstsec
12270   bne   endchk
12280 a@  ldx#$1212  prevent cursor generation
12290   stx$f812
12300   ldd   #$c5e   swap CLEAR & up arrow
12310   sta   $a26e
12320   stb   $a27c
12330   ldd   #22
12340   jsr locate
12350   leax  fndmsg-1,pcr  print Edit messages
12360   jsr printS
12370   leax  edtmsg-1,pcr
12380   jsr printS
12390 tpleftdu   $ee  sector buffer   begin edit
12400   ldx#hextbl  row column
12410   stxhexloc
12420   lda   #asctbl  column only; row same as
hex. section
12430   sta   ascloc
12440 revbyt  com  color
12450   bsr   eddisp
12460 edinp   ldd   hexloc
12470   tst hexflg   are we changing hex. section or
ascii?
12480   beq   hexin
12490   lda   ascloc  if hex. adjust x location
12500 hexin   jsr locate   move cursor
12510   tst hexflg   what type of input do we need?
12520   bne   inchr
12530   bra   inbyte
12540
12550 eddisp ldd   hexloc
12560   jsr locate
12570   tst color
12580   beq   eddsp2
12590   lbsr  flipcl  make reverse
12600 eddsp2 ldd   hexloc
12610   jsr locate
12620   lda   ,u
12630   lbsr  hexprt
12640   lbsr  normcl  make normal
12650   lda   ascloc
12660   ldb   hexloc+1
12670   jsr locate
12680   tst color
12690   beq   eddsp3
12700   lbsr  flipcl  make reverse
12710 eddsp3 lda   ,u
12720   lbsr  ascprt
12730   lbra  normcl  make normal
12740
12750 check   pshs  a,x   test for special edit keys
12760   leax  keys,pcr
12770 chklp   lda   ,x+
12780   beq   gokey
12790   cmpa,s
12800   beq   gokey
12810   leax  2,x
12820   bra   chklp
12830 gokey   leas  3,s
12840   jmp   [,x]
12850
12860 inchr   bsr   read1
12870   cmpa#'
12880   blo   check
12890   sta   ,u
12900   lbsr  flipcl  make revers
12910   jsr scrprt
12920   lbsr  normcl  make normal
12930   bra   movert
12940
12950 inbyte  bsr   read1
12960   sta   io1
12970   lbsr  hexbin
12980   lda   io1
12990   bcs   check
13000   bsr   inbyt2
13010   bsr   read1
13020   sta   io2
13030   lbsr  hexbin
13040   lda   io2
13050   bcs   check
13060   bsr   inbyt2
13070   ldd   io1
13080   lbsr  hxDbin
13090   sta   ,u
13100   bra   movert
13110 inbyt2  lbsr  flipcl make reverse
13120   jsr scrprt
13130   lbra  normcl  make normal
13140
13150 read1   lbsr  flipcl
13160   jsr getchr
13170   lbsr  normcl
13180   cmpa#3
13190   bne   enread
13200   com   hexflg
13210   leas  2,s
13220   lbra  edinp
13230 enread rts
13240
13250 movert ldd   hexloc
13260   jsr locate
13270   clr color
13280   lbsr  eddisp
13290   leau  1,u
13300   tst opnflg
13310   beq   dspyrt
13320   tst lstsec
13330   beq   dspyrt
13340   cmpu fcblst
13350   blo   dspyrt
13360   lbra  tpleft
13370 dspyrt  inc   ascloc
13380   lda   ascloc
13390   cmpa#74
13400   bhs   rowdn
13410   lda   hexloc
13420   adda #hexcel size
13430   sta   hexloc
13440   lbra  revbyt
13450 rowdn   inc   rownum
13460   lda   rownum
13470   cmpa#18
13480   lbhi  tpleft
13490   ldd   #5*$100+58
13500   sta   hexloc
13510   stb   ascloc
13520   lbra  revbyt
13530
13540 movelf ldd   hexloc
13550   jsr locate
13560   clr color
13570   lbsr  eddisp
13580 mvlf2   leau  -1,u
13590   dec   ascloc
13600   lda   ascloc
13610   cmpa#58
13620   blo   rowup
13630   lda   hexloc
13640   suba #hexcel size
13650   sta   hexloc
13660   bra   mvlf3
13670 rowup   dec   rownum
13680   lda   #50
13690   sta   hexloc
13700   lda   #73
13710   sta   ascloc
13720   lda   rownum
13730   cmpa#3
13740   blo   gobot
13750   bra   mvlf3
13760 gobot   lda   #18
13770   sta   rownum
13780   leau  $100,u
13790 mvlf3   tst opnflg
13800   beq   lfdone
13810   tst lstsec
13820   beq   lfdone
13830   cmpu fcblst
13840   bhs   mvlf2
13850 lfdone  lbra  revbyt
13860
13870 moveup   lddhexloc
13880   jsr locate
13890   clr color
13900   lbsr  eddisp
13910   leau  -$10,u
13920   dec   rownum
13930   lda   rownum
13940   cmpa#3
13950   bhs   updone
13960   leau  $100,u
13970   lda   #18
13980   sta   rownum
13990   tst opnflg
14000   beq   updone
14010   tst lstsec
14020   beq   updone
14030 mvup3   cmpu fcblst
14040   blo   updone
14050   dec   rownum
14060   leau  -$10,u
14070   bra   mvup3
14080 updone lbra  revbyte
14090
14100 movedn   lddhexloc
14110   jsr locate
14120   clr color
14130   lbsr  eddisp
14140   leau  $10,u
14150   inc   rownum
14160   tst opnflg
14170   beq   mvdn2
14180   tst lstsec
14190   beq   mvdn2
14200   cmpu fcblst
14210   blo   mvdn2
14220   lda   rownum
14230   bra   mvdn5
14240 mvdn2  lda   rownum
14250   cmpa#18
14260   lbls  revbyt
14270 mvdn4  leau  -$100,u
14280   lda   #3
14290   sta   rownum
14300   lbra  revbyt
```

continued on page 19

Adventures in Assembly, Part 1

Art Flexser

Some assembly excercises and solutions by the creater of ADOS.

Here is an exercise in assembly language programming. It is all very well to read these tutorials and assemble the programs that go with them, but to learn assembly language, it is also necessary to practice writing programs by yourself. So, here's something to get you started. The "answers" are in the files TUTA1A.SRC and TUTA1B.SRC, with some commentary following the listings. But I encourage you to not look at these until you've had a whack at doing it yourself. Practice makes perfect!

Exercise #1

Write a program that will clear the screen with a particular color when you type the initial of the name of that color. When the screen has been cleared, the program should loop back to the beginning and await another color key. Keys that are not color initials should be ignored, except for the break key, which should cause the program to exit to BASIC. Since some colors share the same initial, we'll use the following as our color codes:

X = black ($80)
G = green ($8F)
Y = yellow ($9F)
B = blue ($AF)
R = red ($BF)
W = buff ($CF)
C = cyan ($DF)
M = magenta ($EF)
O = orange ($FF)

For an additional challenge, see if you can make the program work properly regardless of whether the input is in lower or upper case. It is possible to accomplish this with just a single added instruction (an AND instruction, if you must know!)

Exercise #2

This is not really a separate programming problem, but rather a more sophisticated approach to solving #1 than the most straightforward one. The straightforward approach to #1 involves using 9 separate CMPA #<byte value> instructions, one to check for each possible color key. This approach is shown in TUTA1A.SRC. It works very nicely when there are only 3 or 4 keys to be scanned for, but gets a bit cumbersome when there are more. The program in TUTA1A.SRC has a lot of repetitious code in it, and, as in Basic programming, that should hint that a more efficient programming approach may be called for.

So, see if you can write a program that accomplishes the same thing as described in #1, but which uses a lookup table containing pairs of bytes instead of multiple CMPA instructions. The first byte of each pair will be the ASCII value for the color name initial, and the second will be the byte that the screen will get filled with to produce that color. One advantage of this approach is that the program can very easily be modified to define a new key and the screen fill byte that goes with it, just by adding an additional pair of bytes to the lookup table. To allow maximum flexibility, do not have the program assume that the lookup table contains any fixed number of entries. Rather, just have the program look for a byte value of zero to tell it when it has come to the end of the lookup table (such a byte is referred to as a "terminator"). TUTA1B.SRC contains a program that uses this approach.

```
00100 *TUTA1A.SRC
00110 *ART FLEXSER
00120       ORG   $3000
00130 *FILLS SCREEN WITH APPROPRIATE
COLOR WHEN KEY IS
00140 *PRESSED THAT IS THE INITIAL OF
THE COLOR NAME
00150 START JSR   [$A000] GET
KEYPRESS
00160       BEQ   START  LOOP IF NO KEY
PRESSED
00170       CMPA  #3     BREAK KEY?
00180       BEQ   EXIT   YES, RTS
00190       ANDA  #$DF   ENSURE
UPPERCASE
00200       CMPA  #'X    BLACK?
00210       BNE   GRN
00220       LDA   #$80
00230       BRA   FILSCR
00240 GRN   CMPA  #'G    GREEN?
00250       BNE   YELO
00260       LDA   #$8F
00270       BRA   FILSCR
00280 YELO  CMPA  #'Y    YELLOW?
00290       BNE   BLUE
00300       LDA   #$9F
00310       BRA   FILSCR
00320 BLUE  CMPA  #'B    BLUE?
00330       BNE   RED
00340       LDA   #$AF
00350       BRA   FILSCR
00360 RED   CMPA  #'R    RED?
00370       BNE   BUFF
00380       LDA   #$BF
00390       BRA   FILSCR
00400 BUFF  CMPA  #'W    BUFF?
00410       BNE   CYAN
00420       LDA   #$CF
00430       BRA   FILSCR
00440 CYAN  CMPA  #'C    CYAN?
00450       BNE   MAG
00460       LDA   #$DF
00470       BRA   FILSCR
00480 MAG   CMPA  #'M    MAGENTA?
00490       BNE   ORNG
00500       LDA   #$EF
00510       BRA   FILSCR
00520 ORNG  CMPA  #'O    ORANGE?
00530       BNE   START  NO, GET
ANOTHER KEY
00540       LDA   #$FF
00550 *FILL SCREEN WITH BYTE VALUE IN
A REGISTER
00560 FILSCR LDX  #$400
00570       TFR   A,B    2 BYTES AT A TIME
00580 LOOP1 STD   ,X++
00590       CMPX  #$600
00600       BLO   LOOP1
00610       BRA   START  DONE, GET NEW
KEY
00620 EXIT  RTS          EXIT ON BREAK
KEY
00630       END   START
```

```
00100 *TUTA1B.SRC
00110 *ART FLEXSER
00120       ORG   $3000
00130 *FILLS SCREEN WITH APPROPRIATE
COLOR WHEN KEY IS
00140 *PRESSED THAT IS THE INITIAL OF
THE COLOR NAME
00150 START JSR   [$A000] GET
KEYPRESS
00160       BEQ   START  LOOP IF NO KEY
PRESSED
00170       CMPA  #3     BREAK KEY?
00180       BEQ   EXIT   YES, RTS
00190       ANDA  #$DF   ENSURE UPPER
CASE
00200       LEAX  TABLE,PCR   X=START
OF TABLE
00210 LOOP  CMPA  ,X     CHECK KEY
AGAINST TABLE ENTRIES
00220       BEQ   FILSCR FILL SCREEN IF
FOUND
00230       TST   ,X++   END OF TABLE?
00240       BEQ   START  YES, NOT IN
TABLE, GET NEW KEY
00250       BRA   LOOP   NO, CHECK
NEXT TABLE ENTRY
00260 EXIT  RTS          ADIOS, AMIGO
00270 FILSCR LDB  1,X    GET COLOR
BYTE VALUE
00280       LDX   #$400  START OF
SCREEN
00290 LOOP1 STB   ,X+    PUT COLOR
ON SCREEN
00300       CMPX  #$600  END OF
SCREEN?
00310       BLO   LOOP1  NO, CONTINUE
00320       BRA   START  YES, GET NEW
KEYPRESS
00330 *TABLE OF 2-BYTE ENTRIES
00340 * 1ST BYTE IS COLOR INITIAL
00350 * 2ND BYTE IS THE SCREEN DISPLAY
VALUE FOR THE COLOR
```

```
00360 TABLE  FCB   'X    BLACK
00370        FCB   $80
00380        FCB   'G    GREEN
00390        FCB   $8F
00400        FCB   'Y    YELLOW
00410        FCB   $9F
00420        FCB   'B    BLUE
00430        FCB   $AF
00440        FCB   'R    RED
00450        FCB   $BF
00460        FCB   'W    BUFF
00470        FCB   $CF
00480        FCB   'C    CYAN
00490        FCB   $DF
00500        FCB   'M    MAGENTA
00510        FCB   $EF
00520        FCB   'O    ORANGE
00530        FCB   $FF
00540        FCB   0     TERMINATOR
00550        END   START
```

Comments on TUTA1A.SRC

Line 190 ANDA #$DF

This instruction converts any lowercase input to uppercase. Lowercase letters have ASCII values starting with "a"=$61; uppercase letters start with "A"=$41. The difference between the ASCII codes for the upper and lowercase versions of the same letter is that the lowercase version has bit 5 equal to 1 and the uppercase version has this bit equal to 0.

The individual bits of an 8-bit byte are numbered 0-7, starting with the bit at the right. Thus, a bit's number corresponds to the power of two that that bit position represents.

76543210 bit

"A" = $41 = %01000001

"a" = $61 = %01100001

Note that the percent sign is used to signify a binary quantity. Some assemblers, though unfortunately not Edtasm+, will accept this notation.

(editor: Thus the easy explanation for bits and bytes -- a single byte is one character on the screen, so megabyte is a million characters on the screen, etc. Good explanation for novices!)

When you AND two binary quantities (call them M and N) together, each bit position of the result P is determined solely by the bit values in the corresponding bit positions of M and N. Bit 3 (say) of P will be a one if and only if Bit 3 of M AND Bit 3 of N are BOTH one. Looking at each bit position separately, we can then say that if we AND a particular bit position with a 0, the result must have a zero in that bit position, regardless of whether the original bit value in that position was a one or a zero. (1 AND 0) = 0; (0 AND 0) = 0. Also, if we AND a bit position with a one, its value will be unchanged. (0 AND 1) = 0; (1 and 1) = 1.

What good is all this? It allows us to reset a particular bit position to a zero while preserving all of the other seven bit positions unchanged, which is exactly what we need to convert lowercase to uppercase input. That is, if we AND the ASCII value of the keypress with %11011111 (=$DF), we will force Bit 5 to assume a value of zero while leaving the other bits alone. So, by inserting an ANDA #$DF instruction, we can then allow subsequent instructions of the program to check only for the uppercase forms of the letters. Incidentally, the AND operator works exactly the same in Basic. Try this little one-liner, which converts an input letter to uppercase:

```
10 INPUT A$:?CHR$(ASC(A$) AND &HDF):GOTO10
```

While we are on the subject, the OR operator is pretty much the mirror image of AND. If you OR a bit with a 1, the result is a 1. (0 OR 1) = 1; (1 OR 1) = 1. But if you OR a bit with a 0, you preserve its value: (0 OR 0) = 0; (1 OR 0) = 1. So, if you have a need to force a particular bit position to be a one, you do this by ORing with a quantity that consists of zeroes in all bit positions except the critical one. Thus, for example, to force lowercase instead of uppercase, which involves setting Bit 5 to a one, we would use an ORA #%00100000, (or ORA #$20, in language that Edtasm+ understands). Incidentally, forcing a particular bit to be 1 or 0 is called SETTING it or RESETTING it, respectively.

Lines 560-610

These lines contain the routine that fills the screen with the desired byte value. Note that in this version, a TFR A,B instruction is used to duplicate the contents of the A register into the B register. Thus, if A contained $80, the D register, which consists of the A and B registers taken together and considered as a 16-bit quantity, would contain $8080. Copying A into B allows use to use a STD, X++ instruction to fill the screen two bytes at a time, which is faster than if we had used a STA ,X+ instruction.

Comments on TUTA1B.SRC

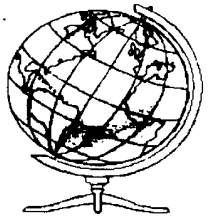
Line 200 LEAX TABLE,PCR

This instruction does the same thing as LDX #TABLE, as far as what value gets put into the X register. However, the LEAX TABLE,PCR form allows the program to be RELOCATABLE. That is, it will still work properly if we load it in with an offset (PCR stands for "position counter relative", by the way). TABLE, in this program, is at location $3029, which happens to be $19 bytes beyond the byte that follows the LEAX TABLE,PCR instruction (I can tell this by looking at my assembled printout, produced by A/LP/NO, and seeing that the LEAX instruction translated to a sequence of bytes ending with a $19).

The difference between the LDX #TABLE form and the LEAX TABLE,PCR form is that LDX #TABLE means that TABLE will be considered as being located at $3029, regardless of any offset that is used in loading the program, while LEAX TABLE,PCR means that TABLE is considered to be located $19 bytes beyond the start of the instruction that follows the LEAX. The latter location will be at $4029 instead of $3029 if we offset load the program by $1000 so that it loads in at $4000. A relocatable program will assemble to produce the same sequence of bytes, regardless of any ORG statement that is included with the source, so that the program will be equally happy anyplace in memory. It is a good idea to get into the habit of writing programs that are relocatable, since it is so easy to do, thanks to the structure of the 6809's instruction set. Use BRA and BSR instead of JMP and JSR to preserve relocatability, if the address you wish to jump to is not a fixed one, such as a ROM call.

Line 270 LDB 1,X

This instruction says to load the B register with the byte that is in the address one beyond that pointed to by the X register. That is, if X contains $4000, B is loaded with the contents of location $4001. The value of X is not changed by this instruction, but is left at $4000. In the program, X points to the ASCII value of a letter—the first byte of one of the pairs of bytes that make up the lookup table. LDB 1,X will therefore load B with the color byte that follows the ASCII code—the second byte of the corresponding byte pair. It is important to keep straight the difference between LDB 1,X and LDB ,X+. The latter instruction (if X = $4000) will load B with the contents of address $4000, and then change the X register so that it contains $4001. In the program, the incrementing of X to point to the next byte pair is taken care of by the TST ,X++ instruction in line 230. This instruction also checks for the zero terminator at the end of the lookup table.

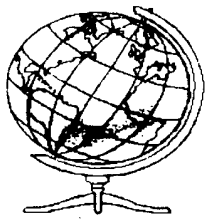


ROBOT ZAP BY ERIC STRIGER

This simple BASIC game puts the Tandy Speech/Sound Cartridge (SSC) to work if you have one! Those familiar with the SSC will realize the "misspellings" are intentional, as the cartridge "speaks" phoneticly. So spelling has to be "adjusted" to get the desired sounds. A good excercise in using the SSC in BASIC programs.

```
1 REM ROBOTZAP          V1.02
2 REM BY ERIC STRIGER     1986
5 PCLEAR4:PMODE4,1:SCREEN1,1:
SCREEN0,0: CLS0:PCLS1
9 REM CLEAR STRING SPACE
10 CLEAR1000:DIM MAP(32,19), FE(16),
RB(16),MN(16),MI(16),EX(16)
11 REM VARIBLE TABLE
14 REM SC=SCORE
15 SC=0
19 REM HS=HIGH SCORE
20 HS=1000
24 REM BS$=BLACK SPACE &
DS$=DOUBLE BLACK SPACE
25 BS$=CHR$(128):DS$=BS$+BS$
29 REM LV=LEVEL
30 LV=1
34 REM NM=NUMBER OF MEN
35 NM=3
50 ROB$="U3L2U3R3U2LU3R3D3L2D2
R3D4L2ND3L3E3"
55 MAN$="E3NF3U4NF2NHNG3UNRU"
60 FEN$="U4NRNLU2NE3NH3U3"
65 MIN$="NR2NL2NU3NE3NH3"
70 EXIT$="NR3U3NR2U3R3"
74 REM REMOVE REMARK IF YOU
HAVE RS-SPEECH AND SOUND PACK
75 V=&HFF00:V1=&HFF7E:V2=-1
90 GOSUB 700:IF V2=-1 THENA$=
"ROWBOT ZAP":GOSUB955:FORT=1TO
100:NEXTT
95 PLAY"V31T10O1L4CL200O4BA#AG
#AG#FEE-DCO3BA#AG#GF#FEE-DCO
2BA#AG#GF#FEE-DC#CO2BA#AG#GF
#FEE-DC#C01BA"
96 B$="..... PREPAIR FOR BATTAL
....."
99 NM=3:LV=1
100 REM TITLE SCREEN
105 CLS0:A$="BY ERIC STRINGER
1986  PRESS ANY KEY TO BEGIN
*INSTRUCTIONS HIT '@' * ":SC=0
110 PRINT@0,STRING$(32,CHR$(143+
RND(7)*16));:PRINT@64,STRING$(32,CHR$(143+
RND(7)*16));:PRINT@32,USING"SCORE
######  HIGHSCORE ######";SC,HS
115 C1=(RND(8)-1)*16
120 PRINT@96,DS$;STRING$(4,CHR$
(131+C1));DS$;STRING$(4,CHR$(131+C1));DS$;
STRING$(4,CHR$(131+C1));DS$;STRIN
G$ (4,CHR$(131+C1));DS$;STRING$(4,
CHR$(131+C1));
125 PRINT@128,DS$;CHR$(143+C1);
DS$; BS$;CHR$(138+C1);BS$;CHR$(143
+C1);DS$;CHR$(143+C1);DS$;CHR$(143
+C1);DS$;BS$; CHR$(138+C1);BS$;CH
R$ (143+C1);DS$;CHR$ (143+C1);DS$;
BS$;CHR$(133+C1);CHR$(138+C1);DS$;
130 PRINT@160,DS$;CHR$(143+C1);
DS$;BS$;CHR$(138+C1);BS$;CHR$(143+
C1); DS$;CHR$(143+C1); DS$;CHR$(143
+C1);DS$;BS$;CHR$(138+C1);BS$;CHR$
(143+C1);DS$;CHR$(143+C1);DS$;BS$;
CHR$(133+C1);CHR$(138+C1);DS$;
135 PRINT@192,DS$;CHR$(143+C1);ST
RING$(3,CHR$(140+C1));CHR$(130+C1);
BS$;CHR$(143+C1);DS$;CHR$(143+C1);
DS$;CHR$(143+C1);STRING$(3,CHR$(140+
C1));CHR$(130+C1);BS$;CHR$(143+C1);DS$;
CHR$(143+C1);DS$;BS$;CHR$(133+C1);CHR$
(138+C1);
140 PRINT@224,DS$;CHR$(143+C1);DS
$; BS$;CHR$(138+C1);BS$;CHR$(143+C
1);CHR$(131+C1);CHR$ (131+C1);CHR$
(143+C1);DS $;CHR$(143+C1);STRING$
(3,CHR$(131+C1));CHR$(136+C1);BS$;
CHR$(143+C1);CHR$(131+C1);CHR$(131+
C1);CHR$(143+C1);DS$;BS$;CHR$(133+C1);
141 PRINTCHR$(138+C1);:PLAY"CO1BA"
145 PRINT@256,STRING$(8,CHR$(128
));STRING$(4,CHR$(131+C1));DS$;CHR$
(129+C1);CHR$(131+C1);CHR$(131+C1);
CHR$(130+C1);DS$;STRING$(4,CHR$(131+C1));
150 PRINT@288,STRING$(11,CHR$(128
));CHR$(134+C1);DS$;CHR$(143+C1);DS$;
CHR$(133+C1);DS$;CHR$(143+C1);DS$;
BS$;CHR$(138+C1);
155 PRINT@320,STRING$(10,CHR$(128
));CHR$(134+C1);DS$;BS$;CHR$(143+C1);
STRING$(2,CHR$(131+C1));CHR$(135+C
1);DS$;CHR$(143+C1);STRING$(3,CHR$(131+C1));
CHR$(136+C1);
160 PRINT@352,STRING$(9,CHR$(128
));CHR$(134+C1);DS$;DS$;CHR$(143+C1);
DS$;CHR$(133+C1);DS$;CHR$(143+C1);
165 PRINT@384,STRING$(8,CHR$(128
));CHR$(135+C1);STRING$(3,CHR$(131+C1));
DS$;CHR$(143+C1);DS$;CHR$(133+C1);
DS$;CHR$(143+C1);
170PRINT@416,STRING$(32,CHR$(143+
RND (7) *16));175A$=RIGHT$(A$,LEN(A
$) -2)+LEFT$(A$,2):PRINT@448,LEFT$(
A$,32);
180 C2=RND(7)*16:PRINT@480,STRING
$(31,CHR$(143+C2));:POKE1024+511,143+C2
185 I$=INKEY$:IF I$="@" THENGOSUB
1000:GOTO110 ELSE IF I$="" THEN110
190 GOSUB200:GOSUB800
195 GOTO 300
200 REM DISPLAY SCORE AND LEVEL
205 CLS0
210 FOR Z=1 TO 25
215 PRINT@0,STRING$(32,CHR$(143+
RND (7)*16));
220 PRINT@32,USING"SCORE ######
HIGHSCORE ######";SC,HS;:PRINT@
64,STRING$(32,CHR$(143+RND(7)*16));
225 PRINT@224,STRING$(32,CHR$(143
+RND (7)*16));
230 PRINT@256,STRING$(32,CHR$(143
+RND(7)*16));:PRINT@256+12,USING"LEVEL
##";LV;
235 PRINT@288,STRING$(32,CHR$ (143
+RND(7)*16));
240 PRINT@320,STRING$(32,CHR$(143
+RND(7)*16));:PRINT@320+12,USING"MEN
##";NM;
245 PRINT@352,STRING$(32,CHR$(143
+RND(7)*16));
246 B$=RIGHT$(B$,LEN(B$)-2)+LEFT$(B
$,2):PRINT@384,LEFT$(B$,32);STRING$(32,
CHR$(143+RND(7)*16));
250 NEXT Z:RETURN
300 REM MAIN GAME CONTORL
305 JX=JOYSTK(0):JY=JOYSTK(1):PK=
PEEK (65280)
310 IF JX<20 THEN PX=PX-1
315 IF JX>42 THEN PX=PX+1
320 IF JY<20 THEN PY=PY-1
325 IF JY>42 THEN PY=PY+1
329 REM UP DATE MAN POSITION
330 GOSUB 400
334 REM UP DATE ROBOT POSITION
335 GOSUB 500
340 T=D(1)+D(2)+D(3)+D(4)+D(5):IF T=0
THENQ=5:GOTO600
345 IF SC>HS THEN HS=SC
390 GOTO 300
400 REM MAN POSITION UP DATE
405 IF MAP(PX,PY)=4
THENQ=1:GOTO600 ELSE IF MAP(PX,P
Y)=3ORMAP(PX,PY)=1THENQ=3:GOTO
600 ELSE IF MAP(PX,PY)=6 THEN Q=2:
GOTO600 ELSE IF MAP(PX,PY)=2 THEN
Q=4:GOTO 600
406 MAP(OX,OY)=0:MAP(PX,PY)=5
410 COLOR 1,1:LINE((OX-1)*8,(OY-1)*10
+10)-((OX-1)*8+8,(OY-1)*10),PSET,BF
415 PUT((PX-1)*8,(PY-1)*10+10)-((PX-1)
*8+8,(PY-1)*10),MN,PSET
416 IF PK=126 OR PK=254 THEN 430
420 OX=PX:OY=PY:RETURN
430 REM DROP MINE
435 MAP(OX,OY)=6:PUT((OX-1)*8,(OY-
1)*10+10)-((OX-1)*8+8,(OY-1)*10),MI,
PSET
440 GOTO 420
500 REMAN ROBOT UPDATE
```

```
505 FOR Z=1 TO 5
509 IF RND(INT(10/LV))=1THEN510
ELSE NEXTZ:RETURN
510 IF D(Z)=0 THEN NEXT Z:RETURN
511 LINE((PX(Z)-1)*8,(PY(Z)-1)*10+10)-
((PX(Z)-1)*8+8,(PY(Z)-1)*10),PSET,BF
512 MAP(PX(Z),PY(Z))=0
515 PX(Z)=PX(Z)+1*SGN(PX-PX(Z))
520 PY(Z)=PY(Z)+1*SGN(PY-PY(Z))
525 IF MAP(PX(Z),PY(Z))=3 AND LV<3
THENGOSUB550:NEXT Z:RETURN
530 IF MAP(PX(Z),PY(Z))=6 THEN GO
SUB 550:NEXTZ:RETURN
535 IF MAP(PX(Z),PY(Z))=5 THEN
Q=1:GOTO600
536 PUT((PX(Z)-1)*8,(PY(Z)-1)*10+10)-((
PX(Z)-1)*8+8,(PY(Z)-1)*10),RB,PSET
537 MAP(PX(Z),PY(Z))=4
545 NEXT Z:RETURN
550 SC=SC+50:MAP(PX(Z),PY(Z))=0:PL
AY "T1L255V31O1ADCFABGEDV16ACG
ADV4EABCAEDB":D(Z)=0:RETURN
600 REM MAN KILLED RUTINE
605 IF Q=1 THEN B$=" THEY GOT YOU
!!!! . . . . . .":NM=NM-1
610 IF Q=2 THEN B$=" STEPED ON YO
UR OWN MINE.....":NM=NM-1
615 IF Q=3 THEN B$=" ZAP !!! YOU HAV
E BEEN ELECTROFIDE... ":NM=NM-1
620 IF Q=4 THEN B$=" YOU HAVE ESC
APE OUT AN EXIT... "
625 IF Q=5 THEN B$=" YOU HAVE KILL
ED ALL THE ROBOTS ON THIS LEVEL..
BONUS "+STR$(LV*100)+"....":LV=LV+1:
SC=SC+100*LV
630 IF NM=0 THENB$=".....*** GAME
OVER ***.....":GOSUB200:GOTO95
635 IF V2=-1 THENA$=B$:GOSUB955
645 IF SC>HS THEN HS=SC
650 GOSUB200:GOSUB800
698 SCREEN1,1:GOTO300
699 END
700 REM DRAW PICES
705 PCLS1
710 DRAW"C0BM128,95;"+ROB$
715 GET(126,94)-(126+8,84),RB,G
720 PCLS1:DRAW"BM128,95;"+MAN$
725 GET(127,95)-(127+8,85),MN,G
730 PCLS1:DRAW"BM127,95;"+MIN$
735 GET(124,95)-(132,85),MI,G
740 PCLS1:DRAW"BM128,95;"+FEN$
745 GET(126,95)-(134,85),FE,G
750 PCLS1:DRAW"BM128,95;"+EXIT$
755 GET(127,95)-(135,85),EX,G
760 RETURN
800 REM SETUP SCREEN FOR PLAY
805 FORX=1TO32:FORY=1TO18:MAP(X,
Y)=0:NEXTY,X
810 PCLS1:POKE178,2:SCREEN1,1
815 REM SET UP BOUNDREIS
820 FOR X=0 TO 31
825 IF RND(10)=5 THENMAP(X+1,0)=2:
PUT(X+8*X,10)-(X+8+8*X,0),EX,PSET
ELSE LINE(X+8*X,10)-(X+8+8*X,0),PSET
,BF:MAP(X+1,1)=1
830 IF RND(10)=5 THENMAP(X+1,19)=2:
PUT(X+8*X,190)-(X+8+8*X,180),EX,PSE
T ELSE MAP(X+1,19)=1:LINE (X+8*X,19
0) -(X+8+8*X,180),PSET,BF
835 NEXT X
840 FOR Y=1 TO 17
845 MAP(1,Y+1)=1:LINE(0,20+10*(Y-1))-
(8,10+10*(Y-1)),PSET,BF
850 MAP(32,Y+1)=1:LINE(8*31,20+10*(Y
-1))-(8*32,10+10*(Y-1)),PSET,BF
855 NEXT Y
860 REM PUT FENCES ON BORD
865 NF=10*LV
870 FOR X=1 TO NF
875 X1=RND(32):Y1=RND(19)
880 IF MAP(X1,Y1)<>0 THEN 875
885 MAP(X1,Y1)=3:PUT((X1-1)*8,10+10*
(Y1-1))-((X1-1)*8+8,(Y1-1)*10),FE,PSET
890 NEXT X
895 REM PUT ROBOTS ON SCREEN
900 FOR X=1 TO 5
905 X1=RND(32):Y1=RND(19)
910 IF MAP(X1,Y1)<>0 THEN 905
915 MAP(X1,Y1)=4:PUT((X1-1)*8,10+10*
(Y1-1))-((X1-1)*8+8,(Y1-1)*10),RB,PSET
920 PY(X)=Y1:PX(X)=X1:D(X)=1:UX(X)=
X1:UY(X)=Y1
925 NEXT X
930 REM PUT THE MAN ON SCREEN
935 X1=RND(32):Y1=RND(19)
940 IF MAP(X1,Y1)<>0 THEN935
945 MAP(X1,Y1)=5:PUT((X1-1)*8,10+10*
(Y1-1))-((X1-1)*8+8,(Y1-1)*10),MN,PSET
950 PX=X1:PY=Y1:OX=PX:OY=PY:
RETURN
955 REM SPEECH OUTPUT
956 POKEV+1,52:POKEV+3,63:POKEV+
35,60
957 POKE65494,0
960 FOR I=1 TO LEN(A$)
965 IF PEEK(V1) AND 128=0 THEN 965
970 POKE V1,ASC(MID$(A$,I,1))
975 NEXT I
980 IF PEEK(V1) AND 128=0 THEN 980
985 POKE V1,13:FORT=1 TO30*LEN (A
$):NEXTT:POKE65495,0:RETURN
1000 REM INSTRUCTIONS
1005 CLS1
1010 PRINT"USEING RIGHT JOYSTICK
KEEP AWAY FROM ROBOTS AND FEN
CES AND WALLS."
1015 PRINT"PRESS FIRE BUTTON TO D
ROP MINES."
1020 PRINT"'E' ARE EXITS. BUT IF YOU
EXIT YOU DONT ADVANCE A LEVEL."
1025 INPUT"PRESS ENTER TO BEGAIN
";N$:CLS0:RETURN
```



ColorZap93 (continued from page 15)

```
14310 mvdn3 leau  $10,u
14320   inca
14330 mvdn5 cmpa #19
14340   bne   mvdn3
14350   bra   mvdn4
14360
14370 nonext coma
14380   rts
14390 next tst mtcflg
14400   beq   nonext
14410   ldxzero
14420   stxmtcflg   clear match and split find
14430   tst opnflg
14440   bne   nxlnxt
14450   ldxmtctrk
14460   stxtrack
14470   stx$ec   system track/sector
14480 nxl2 ldxfndloc   start search AFTER current
match
14490   leax  1,x
14500   stxfndloc
14510   lbra  fndwds
14520 nxlnxt  ldd   frcnum
14530   std   recnum
14540   bra   nxl2
14550
14560 repkey  tst repflg
14570   bne   endrep
14580   lda   dos
14590   beq   rpk1
14600   deca
14610   beq   d1
14620   deca
14630   beq   d1
14640   bra   norep
14650 d1   ldx#$d8ce  end of DOS1.1 irq
14660   bra   rpk2
14670 rpk1 ldx#$d7db  end of DOS1 0 irq
14680 rpk2 leay  REPEAT,pcr
14690   ldu   ,x
14700   cmpu#$8955
14710   bne   norep
14720   sty,x
14730   com   repflg
14740 endrep coma
14750   rts
14760 norep   leax   repmsg-1,pcr
14770   jsr printS
14780   jsr getchr
14790   bra   endrep
14800 repmsg fcc   /Sorry, can't help you with your
current system./
14810   fcb   CR,0
14820
14830 opnmsg   fcc/Link to file: /
14840   fcb   0
14850 ispn fcc   /A file is already open!/
14860   fcb   0
14870 isopn   leax   ispn-1,pcr
14880   jsr printS
14890   jsr getchr
14900   orcc  #1
14910 badlnk  rts
14920 quitln  andcc   #.not.1
14930   rts
14940 link  tst opnflg
14950   bne   isopn
14960   leax  opnmsg-1,pcr
14970   jsr printS
14980   lbsr  linein
14990   bcs   quitln exit on BREAK
15000   ldx#linbuf+1
15010   tst ,x
15020   beq   dir
15030   decb
15040   leay  xitopn,pcr
15050   pshs y
15060   clr ,-s
15070   lda   drive
15080   sta   $eb
15090   ldy#$94c
15100   pshs b
15110   ldd   #$200b
15120 nmclr   sta   ,y+   erase file name area
15130   decb
```

```
15140   bne   nmclr
15150   puls  b
15160   pshs  x
15170   bsr   ckdos
15180   leay  ,x
15190   puls  x
15200 sname jmp   [fgetnm,y]
15210
15220 dir   bsr   ckdos
15230   ldb   drive
15240   stb   $eb
15250   jsr [fdir,x]
15260   sta   $ffd9
15270   lda   #CR
15280   jsr scrprt
15290   bra   link
15300
15310 setclk  pshs  a,x
15320   ldx#$ffd8
15330   lda   clock get default clock speed
15340   sta   a,xset correct clock rate
15350   puls  a,x,pc
15360
15370 ckdos  bsr   setclk
15380   leax  dos10,pcr
15390   tst dos
15400   beq   xckdos
15410   leax  dos11,pcr
15420 xckdos rts
15430
15440 diskon bsr   setclk
15450   jsr [dskcon]
15460   sta   $ffd9
15470   tst $f0
15480   rts
15490
15500 xitopn  ldx#$1ff
15510   stx$957 file type
15520   ldx#$100
15530   stx$97c record length
15540   com   opnflg
15550   ldd   #'I*$100+1
15560   bsr   ckdos
15570   jsr [fopen,x]
15580   sta   $ffd9
15590   orcc  #1
15600   tst $973
15610   lbeq  badlnk
15620   bsr   ckdos
15630   leay  ,x
15640   ldx$928
15650   jsr [flof,y]
15660   sta   $ffd9
15670   jsr $b3ed
15680   std   lof
15690   bsr   ckdos
15700   jsr [fclose,x]
15710   sta   $ffd9
15720   ldx#$200    binary format
15730   stx$957 file type
15740   ldx#$100
15750   stx$97c record length
15760   ldd   #'R*$100+1
15770   bsr   ckdos
15780   jsr [fopen,x]
15790   sta   $ffd9
15800   ldd   #1
15810   std   recnum
15820   lbra  ssec0
15830
15840 unlink  tst opnflg
15850   beq   noulnk
15860   lbsr  ckdos
15870   jsr [fclose,x]
15880   sta   $ffd9
15890   ldd   zero
15900   sta   opnflg
15910   std   recnum
15920   std   frcnum
15930   std   lof
15940   ldx$928
15950   incb      =1
15960   std   7,xclear file record number
15970   andcc   #.not.1
15980   rts
15990 noulnk  orcc  #1
16000   rts
16010
16020 * Next routine used primarily for 35/40 track
disks but also can be
16030 * used to bypass certain copyright schemes.
16040 rsmsg   fcc   /Enter max values for
Track#<CR> Sector#<CR>: /
16050   fcb   0
16060 reset   leax  rsmsg-1,pcr   adjust max values
for track/sector
16070   com   cpyflg
16080   lbsr  cmdset
16090   lbsr  decbin
16100   bcs   badrst
16110   sta   maxtrk
16120   lbsr  more
16130   sta   maxsec
16140   clrb
16150   cmpa#9
16160   bls   a@
16170   incb
16180 a@  lda   maxtrk
16190   cmpa#17
16200   beq   d@
16210   blo   b@
16220   deca
16230 b@  lsla
16240   pshs  b
16250   adda  ,s+
16260 c@  sta   maxgrn
16270   rts
16280 d@  deca
16290   bra   c@
16300 badrst  orcc  #1
16310   rts
16320
16330 * PRINT: dump text screen to printer
16340 print lda   $ff22
16350   lsra
16360   bcs   z@
16370   ldd   #$36fe
16380   sta   $ffa2 $4000
16390   ldx#$4000
16400   stb   $6f
16410 a@  lda   ,x++
16420   jsr [$a002]     print to console out
16430   cmpx #$4fa0
16440   bne   a@
16450   lda   #CR
16460   jsr [$a002]
16470   clr $6f
16480   rts
16490 z@  leax  prterr-1,pcr
16500   jsr printS
16510   jsr getchr
16520   bra   badrst
16530 prterr  fcc   /printer not ready!/
16540   fcb   0
16550
16560 * REPEAT KEYS FOR THE RGB DOS
SYSTEM
16570 * Based on the code of Roger Schrag in
Rainbow
16580
16590 * ADJUST ONLY RATE1 or RATE2. Leave
everything else alone!
16600
16610 RATE1 EQU  60time for repeat key in IRQs
16620 RATE2 EQU  3  .05s repeat
16630 SCRUBEQU  $3F   row 6 not repeated
16640 KCLEAR  EQU $14A RGB variable area
16650 KHOLDEQU  $14Bmay need to be moved
16660 KEYBUF  EQU $152 KEYBOARD BUFFER
16670 KBFEND  EQU $15A
16680
16690
16700 REPEAT  LDX  #KEYBUF
16710 a@  LDA  ,X+
16720   ANDA   #SCRUB
16730   CMPA   #SCRUB
16740   BNE  A@
16750   CMPX   #KBFEND
16760   BNE  a@
16770   INC  KCLEAR
16780   LDA  KCLEAR
16790   CMPA   #7
16800   BLO  Z@
16810   CLR  KCLEAR
16820   CLR  KHOLD
16830 A@  INC  KHOLD
16840   LDA  KHOLD
16850   CMPA   #RATE1
16860   BNE  Z@
16870   SUBA   #RATE2
16880   STA  KHOLD
16890   LDX  #KEYBUF
16900 b@  LDA  ,X
16910   ORA #SCRUB
16920   STA  ,X+
16930   CMPX   #KBFEND
16940   BNE  b@
16950 Z@  JMP  $8955   EXBasic IRQ
16960 zendequ   *
16970
16980   ORG KCLEAR
16990   FDB  0
17000
17010
17020   org   $16a
17030   jmp   start
17040
17050   end   start
```

FINALLY!!
THE END OF
ColorZap93
LISTINGS!!

CoCo 3 Consumer Info

continued from page 5

SIMPLE: The simple keyboard is less confusing than those having many unfamiliar keys. Disk BASIC is much easier to learn than MS-DOS. And what could be simpler than inserting a Program Pak and turning on the computer?

VERSATILE: The Color Computer supports both TV sets and monitors, disk drives (floppy and hard) and cassette recorders, large character text screens and 80 column screens. It can be as simple or sophisticated as you want.

COMPATIBLE: with standard printers (serial port built in, parallel printers require a serial to parallel convertor), disk drives, and modems (external, maximum practical speed is 9600 bps).

POWERFUL: Multitasking, 64 colors, programming languages supported - a good hacker's computer

RELIABLE: Widely used for controlling industrial processes, the Color Computer has a long history of reliability. Service is available at any Radio Shack. The ROM-based Disk BASIC operating system is immune to viruses.

"the world of 68' micros": an excellent monthly magazine that, since 1992, has provided programs, help, product reviews, and instruction for users of the Color Computer.

DELPHI: a national telecommunication information service with a Color Computer Special Interest Group, for exchanging programs and information with "CoCoNuts" across the country.

COCO-LIST: An internet mailing list and use-net group of CoCo lovers.

Computers are playing an ever-increasing role in modern society. The Color Computer 3 is ideal for anyone wanting to learn about them and how to use them, without having to spend a lot of money or attend special classes. No other computer in the world provides so much power for so little cost.

NEW Hardware coming from Cloud Nine

c/o Mark Marlette
3749 County Road 30
Delano, MN 55328
email : mmarlett@isd.net
voice: 612-972-3261

512k - 2048k upgrade board

Just install SIMM memory in 512k increments (2x256K 8 or 9 chip SIMMs). Three chip SIMMs WILL NOT work! This is a timing requirement, as the 8/9 chip SIMMs use the same timing as the CoCo DRAM upgrades.

SCSI Host adapter interface

- Comes with OS9 Drivers, 6x09. 63b09e 1.78MHZ system "megaread" times are ~11 seconds with 512 byte sectors (Nitros 2.00 Level3).
- 256/512/1024 Sector size selection
- FULL SCSI ID supported
- Parity generation, enable/disable. Can use with parity devices such as ZIP drives!
- Gold plated card edge connector
- 50 pin SCSI header port
- Installation/Operation Manual
- Schematic package
- OS9 Utilities SCSItools, SCSIdesc, ZIP/JAZ Tools
- SCSItools - A BASIC09 utility that will do low level SCSI commands.
- SCSIdesc - A BASIC09 utility program that will create the SCSI descriptor for you based upon the menu drive options inputted.
- ZIPJAZtools - This utility will allow the features of the Iomega ZIP/JAZ drives. Eject disk, software protection are some. This utility isn't written yet, but I have the documentation needed from Iomega. Will do this soon!

These products should be available at the Chicago CoCoFest! Look for me there!!

A 512K SIMM upgrade is ready to ship. The unit will ship with the following items:

1 - 512K SIMM Memory Board with 8 or 9 chip 120ns or faster SIMMS

1 - Installation Manual

1 - Schematic package

1 - RSDOS Memory Test Program supplied on 5 1/4" disk.

$40 each including shipping, UPS ground, within the US. If you are outside of the US please indicate method of shipment desired and I will check into the added cost, if any.

BlackHawk Enterprises

New Products!

- Data Windows - $69.95 - A complete flat database program for OS-9/68K. Facilities include database creation, searching, maintenance and report generation. By Alpha Software Technologies.
- GNU TWO - $49.95 - This package include a new port of GNU M4, and the AUTOCONF automatic configuration macros. Together with the included port of BASH these tools make automatic configuration of software a much easier chore. Widely used on UNIX and other operating systems, use it now on your OS-9 platform! Includes two new manuals totaling about 110 pages.
- Model Rocketry Tools - $15 - Includes ports of tools for modeling and tracking the performance of various configurations of model rockets. Essential tools for those interested in designing rockets or achieving specified altitudes. Should run on any OS-9/68K machine.

MM/1, MM/1a and MM/1b hardware and other software still available, inquire!

P.O. Box 10552 • Enid, OK 73706-0552 • (405) 234-3911

ADVERTISER'S INDEX

What are you waiting for?

Get your friends to subscribe to the only magazine that still supports the Tandy Color Computer... *"the world of 68' micros"*!

The more people who want the support, the longer it will be here!