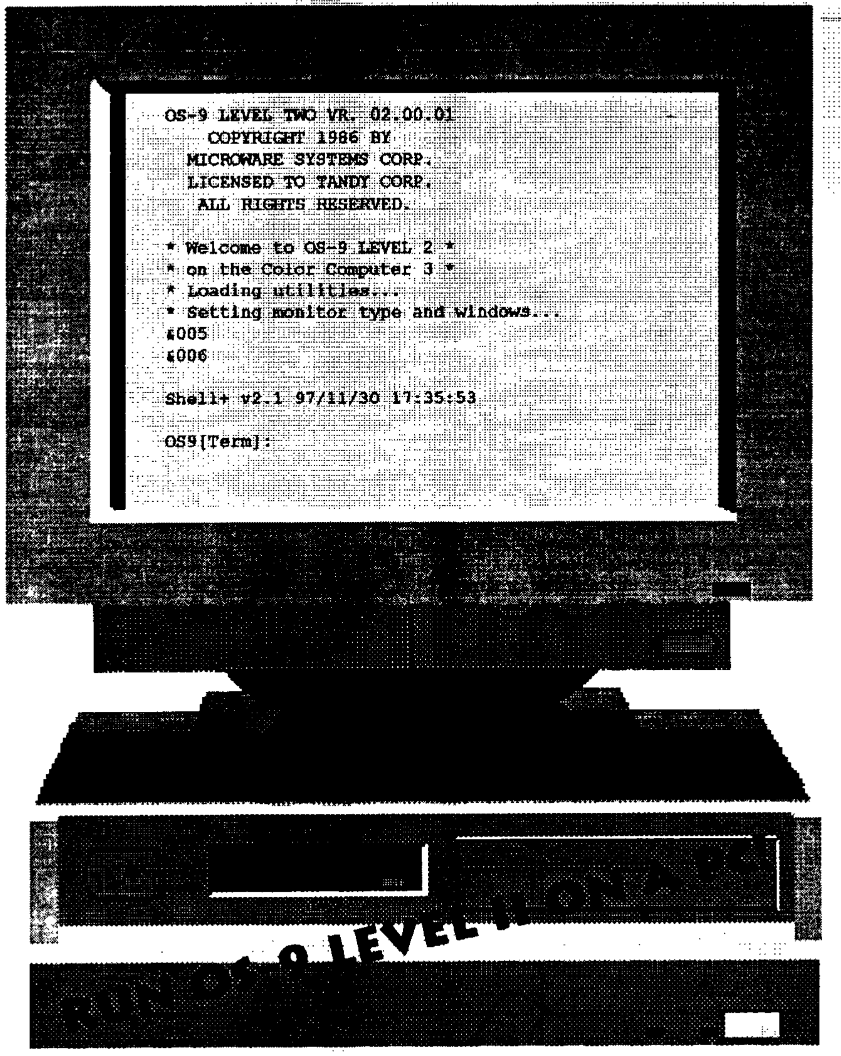
November/December 1997 Vol. 5 Number 3 $4.00US (North America)

the world of 68' micros

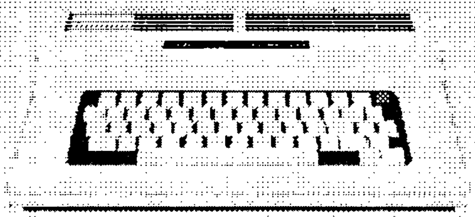
Support for Motorola based computer systems and microcontrollers, and the OS-9 operating system

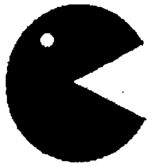

```
OS-9 LEVEL TWO VR. 02.00.01
   COPYRIGHT 1986 BY
  MICROWARE SYSTEMS CORP.
  LICENSED TO TANDY CORP.
   ALL RIGHTS RESERVED.

* Welcome to OS-9 LEVEL 2 *
* on the Color Computer 3 *
* Loading utilities...
* Setting monitor type and windows...
&005
&006

Shell+ v2.1 97/11/30 17:35:53

OS9[Term]:
```




Nicholas Marentes introduces a *NEW* Pac-Man game for the CoCo 3! (see page 8)

CONTENTS

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

With the advent of a new project, it has taken a bit longer to get this issue out than usual. The new project is a magazine supporting my other hobby, AMC (American Motors Corporation) and related cars. For those of you who have seen me at CoCoFests in recent years this should be no surprise. The little green and silver Rambler pretty much heralded my arrival! I'm sure Allen Huffman and Linda Podraza will never forget our venture into downtown Chicago in the Rambler a couple years back, nor will many of the other fest goers that year, as some maintained a vigil at the hotel while others searched for us... as we limped the Rambler back to the hotel. You see, we got in an accident downtown and called to have someone come get us, but they got lost (or were our directions flawed?) and we got tired of waiting. The biggest problem was a leak in the radiator, so we just stopped every 30 minutes to put more water in. Took us about three hours to make the drive from downtown to the hotel (usually takes just over an hour!), but we made it by five a.m.!!

The new magazine is coming along nicely, but not taking the place of this one. I'll restate now that the goal is to be publishing "68' micros" when the new century turns over, so don't lose faith! When I retire from service in about six years, maybe I'll have enough to keep me busy with the two magazines. That's doubtful, but sure does sound good!

There will be a CoCoFest in Chicago again this spring. If you've never attended a 'fest before, this is the one to attend! For the last few years attendence has been fair and stable. There will be a few new items available this year, both in hard and software. Nicholas Marentes announces a new game in this issue, and **Mark Marlette has developed a new 2MB board.**

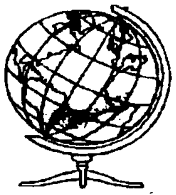
Working examples of the CoCo IDE interface should also be available this time. Unexpected problems caused developmental delays, but Carl Boll was quick to keep everyone appraised of the situation. He had offered to refund deposits, but all decided he should press on with squashing a few bugs first.

I must commend Carl for his actions, as simply not keeping people informed has hurt many CoCo businesses in the past. The CoCo community is well aware that there is a lot of time and little profit made in new hardware, and that unexpected delays are normal. As long as an effort is made to keep most of them informed (in Carl's case, it was through the Internet and various CoCo publications), they are very forgiving of delays.

Mark's 2MB board is a bit different from the usual CoCo projects. He has invested in a PAL programmer and sophisticated board layout software for other projects. This reduces the amount of chips and soldering required to build any particular project. This particular board will have surface mount devices (a first for third party CoCo peripherals!). And instead of using DRAM or 1MB SIMMs, Mark's new board uses eight 256K 30 pin SIMMs. The reason is that these use the same memory refresh cycle as DRAM, but take up less space. 1MB SIMMs would have been more compact, but would require additional refresh circuitry. By using 256K SIMMs, Mark was able to eliminate the second circuit board, making his upgrade easier to install. **A SCSI board should also be available.**

I may not be able to attend the fest this year (but will be represented by someone). My wife (Tiffany) is scheduled for surgery in February and may not be ready to travel by then. She will be out of work through March, so the budget will be stretched also. Don't worry, the surgery isn't anything life threatening, just something that needs to be done. Even though I'm military, we still have some red tape and "hoops" to go through to get them to pay for everything, just like anyone would have to do with their insurance company!

I hope you all had a great Thanksgiving, and that Christmas and the coming of the New Year are equally rewarding!



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.

Reader's Write...

Kudos...

Just a reminder of my appreciation to you for keeping the CoCo alive through 68' micros. I started in '80 with the chicklet CoCo, and have a "3" now; I'm doing this note on a Macintosh Performa 475. My NO MS-DOS belief was formed thereby!

By the way, could I go "online" (WWW) using my still-packed 300 Baud Volksmodem by Anchor Automation. I have OS-9, but use only programs in it and TandyDOS.

Dale Hawley
Dale_Hawley@dbug.org

Thanks for the kudos Dale! It really means a lot to know that the magazine is appreciated by readers. A lot of you declare an anti-DOS/Windows idiom, but I can't. I use a Win 95 based machine for things the CoCo just can't do efficiently, like the layout of this magazine. The CoCo is a hobby and a great machine for many tasks, but can't compete with more modern machines for ease of use in daily chores. These new machines are very poor for really learning about computers though. This the CoCo does extremely well! The CoCo is also much better suited for electronics hobbyists. It is robust and easy to fix as well as to program and "hack" hardware.

Now, to answer your question about surfing the web with your CoCo and 300 baud modem.... It CAN be done. You must have a shell account with an internet provider to do this, or belong to a service such as Delphi that still offers access through their system instead of a direct connection. This works well for general e-mail, but to "surf" the web (world wide web) you will need to run a text browser such as Lynx from the system you are connecting to. The CoCo can easily handle a 2400 baud modem, and with an RS-232 pak a 9600 baud. Again, you will need a UNIX shell account or an account with a service that still offers text terminal only access (like Delphi) to make this work. You can generally ask for more detailed help on the system, or in the CoCo Forum once on Delphi. To view graphics you will have to go with a Mac or Windows based system.

CoCo3 Secrets Made Clearer...

I have been reading Herbert Enzman's CoCo3 Secrets series with interest. In the latest issue (Sept '97) there are some errors of omission which may confuse the readers. The service manual makes the functions of $FF99 very clear. This register does not control the number of text lines per screen. It controls (among other things) the number of "lines per field" which is is a very different animal.

Think of a CoCo graphics screen which normally has 192 vertical lines. With $FF99, we can change this to 200 or 225 but this says nothing about the number of lines of text. Byte $FF98 controls (among other things) the number of lines per text character which can be 1, 2, 3, 8, 9, or 12.

To find the number of text lines per screen, we need to divide the lines per character into the lines per field. In Herbert's program listing #9, the characters are the default 8 lines per. 225/8=28.125 Unit number of text lines can be had only with 192/8=24 or 225/9=25. Other combinations will leave a fractional character on the last line with the stock screen print routines.

Herbert's Table 10 would be better presented as in the service manual. If done that way, some mistakes in Table 10 would have been prevented; $FF99 values 8,$28,$68 should be 0,$20,$60.

$FF99 video resolution register

bit 7	-	
bit 6	=	LPF1
bit 5	=	LPF0
bit 4	=	HRES2
bit 3	=	HRES1
bit 2	=	HRES0
bit 1	=	CRES1
bit 0	=	CRES0

LPFn = lines per field
HRESn = horizontal resolution
CRESn = color resolution

LPF1	LPF0	lines per field
0	0	192
0	1	200
1	0	reserved
1	1	225

In text screen mode:

	HRES2	HRES1	HRES0	CRES1
32 character	0	0	1	na
40 character	0	1	1	na
80 character	1	1	1	na

... and adding Herbert's discoveries

32 character	0	0	0	no attributes
40 character	0	1	0	n a
64 character	1	0	0	n a
64 character	1	0	1	w/ attributes
80 character	1	1	0	n a

This is not unexpected when you consider that $FF99 does not control the number of characters per line but the number of bits per line. Selections are 160, 256, 320, 512, 640 and with 8 bit wide characters the results are 20, 32, 40, 64, and 80. Unfortunately, in text mode you cannot select the 160 bit width.

One programming hint. Listing 9 has several entries such as:

```
LDD TEMP
STA $FF9D
STB $FF9E
```

The code should be:

```
LDD TEMP
STD $FF9D
```

Robert Gault
robert.gault@worldnet.att.net

Thanks Robert! I can always count on you to discover problems, which is good, as I don't always know what I'm looking at!! As a side note, ANYONE can write in and let me know when something goes awry or just doesn't look right. I can't improve without feedback from readers, both positive and negative.

More CoCo RS-232 Info

Robert Gault

Timing Loops and the CoCo RS-232 Port

As the quest for ever faster computers continues, these machines have become so fast that their operation leaves humans in the dust. Even the Coco can be made to list a directory or file faster than we can read the screen.

Some operations, such as data transfer, require precisely timed intervals for synchronization between computers. In short, some method is required to slow down computers in a controlled manner.

There are two methods for timing used on the Coco, interrupts and timing loops. Interrupts are breaks in the flow of code caused either by external events or a hardware clock. Timing loops are sections of code which are repeated to waste time. The number of repetitions controls the time wasted. Below are some examples of timing loops and their implications on the Coco RS-232 code used to send info to printers or modems.

Everyone who writes in Basic on the Coco has at one time used something like the following:

```
100 FOR CT =3D 1 TO NN: NEXT CT
```

This do-nothing For/Next statement wastes time. The amount of time depends on the value of NN. Similarly, the same thing can be done in assembly language:

```
  * timing subroutine
  timer   ldx  #nn
     load register X with a number
  t1      leax -1,x
     decrease the value of reg.X by 1
          bne  t1
      loop if not equal to zero
          rts
      return from the timing loop
```

What both these examples have in common is that the timer has three parts, the initialization, the main timing loop, and the conclusion. To accurately time something with a loop, the time required to execute the initialization and the conclusion can not be ignored. Using the assembly timer as an example, initialization takes 3 clock cycles, conclusion takes 1 clock cycle, and each pass through the loop takes 5 clock cycles.

It is easy to see that when nn is 1, the overhead is 80% of the loop time, 4 vs. 5. At the maximum value of 65535, the overhead is about 1/1000 of 1% of the loop time. This means that without careful planning, the effect of timing loops will not be linear with count. A perfect example of this is the loop used for the Coco bit-banger RS-232 port.

This topic came up during a discussion on the Coco listserver. Allen Huffman posted a complex equation for calculating the Coco baud rate constants:

```
POKE150,INT(.2175+5.7825^(5-(LOG(BD/600)/LOG(2)))-4.5)
```

The constants are the values the Coco owner=92s manual says should be POKEd into addresses $95 and $96. Here they are as listed for the Coco3 at 1MHz:

Baud Rate	Decimal Value
120	458
300	180
600	87
1200	41
2400	18
4800	not given
9600	not given

Ignoring the exact definition of baud, these values relate to bits transmitted per second. Therefore one would expect that when the baud rate changed by a factor of two, the timing constant would change by a factor of two; it doesn=92t. But, if your refer to the timer example above, this nonlinearity is not surprising. It is clear that the system overhead for sending one bit is not negligible at high baud rates.

With our new knowledge about timing constants, can we create a simpler formula for baud rates than the one given above? In particular, can we show a linear relationship between the constant and baud rate? Yes, we can.

Let's start by assuming the constant for 120 baud is correct and that the overhead for this value is insignificant.

Table 1 shows the ratio between the baud rates and calculates new constants by multiplying 458 successively by each ratio. The old constants are then subtracted from the new ones to get the last column. It is easy to see that the system overhead must be equivalent to five timing loops. We can then generate a new table of theoretical constants as shown in Table 2.

Table 1

Calculated Baud Rate	Decimal Value	Ratio	from Ratio	difference
120	458	-	458	-
300	180	0.4	183.20	3.20
600	87	0.5	91.60	4.6
1200	41	0.5	22.90	4.9
4800	-	0.5	11.45	"5"
9600	-	0.5	5.73	"5"

In the above table, we start by adding 458+5 to get our first total bit delay. Each subsequent delay value is calculated from the baud rate ratios. Finally the new constants are obtained by subtracting 5 from each total delay.

The above can be summarized into a formula which incorporates the effect of running the Coco at either 1 or 2 MHz speed:

timing constant =3D (clock * 9600 * 5.79 / baud) - 5; clock =3D 1 or 2

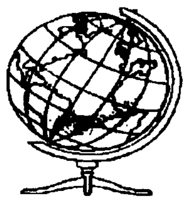
The above equation is both simple and demonstrates the linear relationship between the baud and timing constant. The system overhead is also clearly visible. Nevertheless, buyer beware, the formula assumes that the Tandy constant for 120 baud is correct. This may not be true particularly if the Coco is used with serial to parallel converters.

Table 2

Baud Rate	Total delay	New Constant
120	463	458
300	185.2	180
600	92.6	88
1200	46.3	41
2400	23.15	18
4800	11.58	7
9600	5.79	1

From a practical point of view, each reader should determine the high and low limits for the lowest functional baud rate by printing a line of text. The average value should be used as a reference constant to determine an optimal series assuming a system overhead of 5.

My internet address:
robert.gault@
worldnet.att.net



NEW PRODUCT ANNOUNCEMENT: Omni Basic

A compiler that converts BASIC to C executables!

Computer Design Lab (CDL) announces the availability of OmniBasic for MicroSoft Windows 95 and Windows NT (it is also available for OS-9/68000 systems). Now you can easily convert those old Microware basic programs to run on Win95/NT and many other platforms. All OmniBasic programs are FULLY portable to/from ANY of the supported platforms.

OmniBasic is available for Win95/NT, OS/2, MSDOS, Linux (ELF), OS-9/68000, OS-9/68020, and OS-9000 (v1.4). The OS/2 version requires EMX v0.9c. The MSDOS version requires DJGPP v2.1 and also runs under Win 3.xx.

The 68000 version should run on all "core" machines such as the AT306. It has been tested on the MM/1 with the 68070. It does not require a run time package but does require the Microware K&R C compiler for the final step of compilation.

The shareware (demo) version may be downloaded from:

http://www.bmtmicro.com/catalog/omnibasic.html

The GNU port required to run OmniBasic on Win95/NT may be downloaded from:

http://www.cygnus.com/misc/gnu-win32/

ftp://ftp.cygnus.com/pub/gnu-win32/gnu-win32-b18/

http://www.fu.is.saga-u.ac.jp/~colin/gcc.html

It is necessary to download the Cygnus gnu-win32 package plus the Minimalist package. A script file (included with OmniBasic) is then run which copies the required files from the Cygnus package to the Minimalist package. OmniBasic runs for Win95/NT runs ONLY with the Minimalist package, so after installation, the Cygnus package may be deleted (freeing 45-60MB from your hard disk). OmniBasic plus the Minamalist package consume approximately 10 MB total.

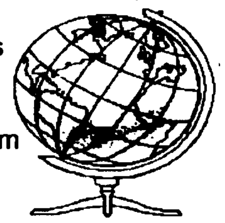
The Cynus file to download is cdk.exe which is a self-extracting executable file. The Minimalist file is a tar-zip and can be unzipped/untarred with WinZip.

OmniBasic is a C-output BASIC compiler which has a similar syntax to BASIC09 but also includes many advanced features such as pointers, based variables, dynamic memory allocation, macros, conditional compile, and more. It takes a .b (basic) file as input and outputs C code. It then automatically calls the C compiler which in turn calls the assembler which in turn calls the linker. The result is a program that is a binary executable that will run with no support required execept the operating system itself. You need the C package plus the OmniBasic package to develop OmniBasic programs.

MAIL:
BMT Micro
P.O. Box 15016
Wilmington, NC 28408

PHONE:
800-414-4268 –
Orders only (USA and Canada)
910-791-7052 –
Orders & questions

INTERNET:
bmt@bmtmicro.com



Chicago CoCoFest 98!

THE GLENSIDE COLOR COMPUTER CLUB OF ILLINOIS PRESENTS
THE SEVENTH ANNUAL "LAST" CHICAGO COCOFEST
April 18th & 19th, 1998 (Sat. 10am-5pm; Sun. 10am-3:30pm)

Elgin Holiday Inn

(A Holidome Indoor Recreation Center)
345 W. River Road
Right off intersection of I-90 & IL-31, Same location as past years!
Overnight room rate: $65.00 (plus 10% tax)

Call 1-847-695-5000 for reservations. Be sure to ask for the "Glenside" or "CoCoFEST!" rate. There is a limited number of rooms set aside for the CoCoFest. *These rooms will be released on March 31. They will not be available at the 'Fest rate after that date, so **make your reservations early!***

General Admission: $10.00, whole show (Children 10 and under are free)

For further information, general or exhibitor, contact:

Tony Podraza, VP, Spcl Evnts, GCCCI
847-428-3576, VOICE
847-428-0436, BBS
Tonypodraza@juno.com

Mike Knudsen, President, GCCCI
630-665-1394, VOICE
Mknudsen@lucent.com

Brian Schubring, Ast., Fest Coordinator, GCCCI
E-MAil theschu2@juno.com OR theschu3@aol.com

CoCoFEAST!

That's right a CoCo FAMILY DINNER at the HoliDay Inn. Why? So You don't have to drive 'here or there' or try to decide which group you want to spend time with. We can be all together to enjoy the food, and best of all, each others company. There my be a Keynote speaker present. This is planned for Saturday Night about 6:00pm. We need is a MINIMUM of 50 people to reserve a dining room. The tickets will only be available in advance, AT THIS TIME! People to conntact are listed below. The cost will be only $15 U.S. PER Person. We will be able to take paid reservations only up to March 28th, 1998. Please contact one of us for further details.

NOTE: THE CLUB IS NOT MAKING ANY MONEY FOR THE DINNER, NOR PLANS TO. THIS FUNCTION IS ONLY TO PROMOTE A COCO FAMILY GATHERING AT ONE GREAT LOCATION... THE COCOFEST!

If by the specified date we are lacking the number of attendees required, the dinner will be scrapped and a refund will be issued at the Fest.

Afterwards there may be a Musical Monk'O Rama Jam-Session like we had last year with Brother Jeramy, Allen Huffman, and anyone else who wants to bring and instrument and join in!

See Ya'll there in '98!!!

FARNA *Systems*
Your most complete source for Color Computer and OS-9 information!

Post Office Box 321
Warner Robins, GA 31099
Phone: 912-328-7859
E-mail: dsrtfox@delphi.com

ADD $3 S&H, $4 CANADA, $10 OVERSEAS

BOOKS:

Mastering OS-9 - $30.00
Completely steps one through learning all aspects of OS-9 on the Color Computer. Easy to follow instructions and tutorials. With a disk full of added utilities and software!

Tandy's Little Wonder - $25.00
History, tech info, hacks, schematics, repairs,... almost EVERYTHING available for the Color Computer! A MUST HAVE for ALL CoCo aficionados, both new and old!!! This is an invaluable resource for those trying to keep the CoCo alive or get back into using it.

Quick Reference Guides
Handy little books contain the most referenced info in easy to find format. Size makes them unobtrusive on your desk. Command syntax, error codes, system calls, etc.
CoCo OS-9 Level II : $5.00
OS-9/68000 : $7.00

Complete Disto Schematic set: $15
Complete set of all Disto product schematics. Great to have... needed for repairs!

SOFTWARE:

CoCo Family Recorder: Best genealogy record keeper EVER for the CoCo! Requires CoCo3, two drives (40 track for OS-9) and 80 cols.
DECB: $15.00 OS-9: $20.00

DigiTech Pro: $10.00
Add sounds to your BASIC and M/L programs! Very easy to use. User must make simple cable for sound input through joystick port. Requires CoCo3, DECB, 512K.

ADOS: Best ever enhancement for DECB! Double sided drives, 40/80 tracks, fast formats, extra and enhanced commands!
Original (CoCo 1/2/3) : **$10.00**
ADOS 3 (CoCo 3 only) : **$20.00**
Extended ADOS 3 (CoCo 3 only, requires ADOS 3, support for 512K-2MB, RAM drives, 40/80 track drives mixed) : **$30.00**
ADOS 3/EADOS 3 Combo: $40.00

Pixel Blaster - $12.00
High speed graphics tools for CoCo 3 OS-9 Level II. Easily speed up performance of your graphics programs! Designed especially for game programmers!

Patch OS-9 - $7.00
Latest versions of all popular utils and new commands with complete documentation. Auto-installer requires 2 40T DS drives (one may be larger).

TuneUp : $20.00
Don't have a 6309? You can still take advantage of Nitro software technology! Many OS-9 Level II modules rewritten for improved speed with the stock 6809!

Thexder OS-9
Shanghai OS-9 : $25.00 each
Transfer your ROM Pack game code to an OS-9 disk! Please send manual or ROM Pack to verify ownership of original.

Rusty : $20.00
Launch DECB programs from OS-9! Load DECB programs from OS-9 hard drive!

NitrOS-9:

Nitro speeds up OS-9 from 20-50% depending on the system calls used. This is accomplished by completely rewriting OS-9 to use all the added features of the Hitachi 6309 processor. Many routines were streamlined on top of the added functions! The fastest thing for the CoCo3! Easy install script! 6309 required.
Level 3 adds even more versatility to Nitro! RBF and SCF file managers are given separate blocks of memory then switched in and out as needed. Adds 16K to system RAM... great for adding many devices!
NitrOS-9 V.2.0: $35.00
NitrOS-9 Level 3: $20.00
SAVE $10! V.2.0 & Level 3: $45.00

The AT306 OS-9 Single Board Computer

AT306 Motherboard Specs:
16 bit PC/AT I/O Bus (three slots)
MC68306 CPU at 16.67MHz
Four 30 Pin SIMM Sockets
IDE Hard Drive Interface
Floppy Drive Interface (180K-2.88M)
Two 16 byte Fast Serial Ports (up to 115K baud)
Two "Terminal" Serial Ports (no modem)
Bidirectional Parallel Port
Real-time clock
PC/AT Keyboard Controller (five pin DIN)

Included Software Package:
"Personal" OS-9/68000 Vr 3.0
(Industrial with RBF)
MGR Graphical Windowing Environment
with full documentation
Drivers for Tseng W32i
and Trident 8900 VGA cards
Drivers for Future Domain 1680
and Adaptec AAH15xx SCSI cards
Many PD and customized utilities and tools

The AT306 is a fully integrated single board computer. It is designed to use standard PC/AT type components. Sized the same as a "Baby AT" board (approximately 8" square). Compact and inexpensive enough to be used as an embedded controller! Use with a terminal (or terminal emulation software on another computer) or with a video card as a console system. Basic OS-9 drivers are in ROM, making the system easy to get started with.

HACKERS MINI KIT (FARNA-11100): Includes AT306 board, OS-9 and drivers, util software, assembly instructions/tips, T8900 1MB video card. Add your own case, keyboard, drives, and monitor! *ONLY $500!*

Call for a quote on turn-key systems and quantity pricing.
Warranty is 90 days for labor & setup, components limited to manufacturers warranty.

Microware Programmers Package -
Licensed copies of Microware C compiler, Assembler, Debugger, and many other tools!
***With system purchase:* $65.00 *Without system:* $85.00**

operating system nine

Alan Dekok

More on Nitro V.2.00 and Nitro Level III.

NitrOS-9 V.2.00

1. What is NitrOS-9 (Nitro)?

NitrOS-9 is a 99% OS-9 compatible operating system for the Hitachi 6309E. It can be simply 'dropped in' to replace the OS-9 you currently use. The current version of Nitro is v2.00 and it is available exclusively from FARNA Systems.

2. Why is it 99% compatible, and not 100% compatible?

Because Nitro runs in HD6309E native mode, the CPU itself behaves in a slightly different manner than the MC6809E. The two differences are:

* The CPU runs faster. Software timing loops must be adjusted (such as serial port and printer drivers).

* Programs that use interrupts have to be patched to use 6309 native mode instead of 6809 mode (like Home Publisher).

Nitro includes patches for all known programs to ensure they work correctly on the 6309. If you have a program that Nitro does not patch, send it to the distributor and a patched program will be returned to you at no cost.

All other programs may be used as they are, and their behaviour will not change after installing Nitro.

3. Why should I use Nitro instead of OS-9?

The main reason to use Nitro is speed. Any OS-9 application running under NitrOS9 will run a minimum of 10-15% faster than the identical application running under stock OS-9. This speed increase is due to the 6309 native mode.

The kernel of Nitro has been written using the new capabilities of the 6309. As the 6309 is much more powerful than the 6809, Nitro is much more efficient and powerful than OS-9.

Many of the internal algorithms used by OS-9 have been changed in Nitro. The new algorithms execute much faster than the previous ones, but are otherwise identical in behaviour.

The graphics drivers (WindInt, Grfdrv) were also optimized for the 6309. Speed tests show that text screen updates are performed more than four times as fast under Nitro as under stock OS-9. Even the 'Xmas grfdrv' patches lag far behind NitrOS9 in terms of raw graphics speed.

The latest version of Nitro speeds up the entire system by 2 to 10 times over stock OS-9. Text writes, graphics fonts, line drawing, flood filling, system call overhead, serial I/O, and pipes are much faster than all previous versions of OS-9 or Nitro. The difference is so large now that as a developer, I refuse to use anything less than Nitro v2.00. Even v1.16 of Nitro is so slow as to be more comparable to a stock system than to v2.00.

These are not the only reasons to move to Nitro. Another advantage is that all of the bugs that existed in stock OS-9 have been found and fixed. This includes bugs that had not been discovered previously. All publicly available kernel 'fixes' and 'enhancements' are included in Nitro.

4. Does Nitro implement any of the rumored 'OS-9 Upgrade'?

Yes and no. Many of the same features rumored to be in the upgrade have been included in Nitro, but were developed independently of the upgrade. Other features in the upgrade may be added in the future.

There are many optimisations in Nitro, however, that never made it into the OS-9 Upgrade. These features cause Nitro to be much faster than the OS-9 Upgrade ever was.

For people who do want these optimisations without the potential HD6309E conflicts, FARNA also carries a product called TuneUp. TuneUp is a collection of new modules for your stock 6809 OS-9 system that drastically improves the speed of the system.

5. What level of support is there for NitrOS-9?

Both the authors and the distributors are fully committed to supporting Nitro. Any questions may be relayed to:

Software support: Alan DeKok
adekok@gandalf.ca
Distributor support: Frank Swygert
dsrtfox@delphi.com
912-328-7859
Write in care of 68' micros

With NitrOS-9, you find a level of support that was never possible with OS-9. Any software problems should be relayed to Software Support. You will be contacted about the problem, and a fix will be shipped as soon as possible.

6. Are there hardware problems with installing Nitro?

You must replace the MC68B09E chip inside the Color Computer with a HD63B09E. Directions for doing this are given in the Nitro manual. If you do not feel comfortable performing this modification yourself, any electronics shop can desolder and replaces chips for a charge.

Nitro pushes the hardware of the Color Computer much closer to its limit that did OS-9 or DECB. It is very important that the installation procedure is followed carefully and exactly in order to have a complete and therefore stable system.

No other hardware modifications are required. An HD63C09E may also be used.

7. How hard is it to install Nitro?

Nitro is much easier than to install stock OS-9. You must be able to build a boot disk, and as version 1.20 ships with the needed modules on the disk, minimal patching is required. You simply replace the OS-9 modules with the equivalent Nitro modules. The executable modules should NOT be mixed between OS-9 and Nitro, as they are incompatible. Device descriptors can be copied over unchanged.

If you have an OS-9 module and there is no Nitro equivalent, contact the distributor, and one will be shipped to you as soon as possible. After more than two years of shipping, however, we believe there are few, if any modules that do not have Nitro equivalents.

Once NitrOS-9 is installed, you may want to reformat your disks with a smaller interleave factor. Nitro can handle much higher data rates than OS-9.

8. What other enhancements are available for OS-9?

Users of Nitro can increase the amount of system RAM available by installing the Level III. Level III is compatible with all programs that will run under Nitro. The best 6809 enhancement package is TuneUp (see item 5).

9. Is other HD6309 specific software available?

There is a 6309 version of rma, and two 6309 versions of asm. There is not, however a 6309 C compiler.

Most authors appear to be continuing to be writing mainly 6809 software. This ensures that their software will run on ALL OS-9 LII platforms, although their software will run much faster under Nitro.

Nitro, therefore, benefits you the most. No one else has to be aware that you are running it, and you can buy any OS-9 software in the confidence that it will work on your Nitro system.

Nitro Level III

1. What is Level III?

Level III is one step beyond OS-9 Level II. Using Microware's definitions, we have:

Level I

System and User programs in system memory. The Coco 1 & 2 ran OS-9 with both the system and user programs executing out of the same 64K memory map.

Level II

User programs outside of system memory. The Coco 3 with it's MMU has one 64K memory map for the system, and each process gets it's own 64K memory map.

Level III

System modules running outside of the system map. Parts of the IO subsystem are task switched in and out of the system memory map, increasing the memory available to the entire system.

In short, Level III increases the amount of memory available in the OS-9 system map by 16K.

2. How exactly does Level III work?

Level III puts SCF and RBF each into their own 16K mini task. If the system is doing I/O to an SCF device, the SCF file manager and drivers are mapped into memory, and the RBF modules are mapped out. When RBF I/O is required, SCF gets mapped out.

The idea for Level III was born of the realization that it is possible to boot an OS-9 system with only SCF drivers, and likewise with only RBF drivers. Since the two file managers are completely independent in this way, there is no need for both of them to be in the system memory map at the same time.

In effect, the system is turned into a Kernel only process with 48K of RAM and 2 IO processes (RBF and SCF), each with 16K of RAM. The kernel process contains the minimum modules to run an OS-9 system and the descriptors. The RBF/SCF processes contain the IO modules and buffers.

Modifications to OS9p1, Clock, and IOMan were required in order to direct system calls, I/O requests, memory allocation requests, and IRQ's to the appropriate map. All other system modules remain unchanged.

3. How difficult is it to install Level III?

If you can create a new boot disk, you will find that installing Level III is no more difficult than that. Simply replace the specified modules in your OS9Boot file, re-order your OS9Boot file as described in the manual, and reboot your system.

As with all system upgrades, it is a good idea to install the upgrade onto a boot floppy, instead of over-writing your existing system configuration.

Nitro Level III is incompatible with the system modules in stock 6809 OS-9 and with pre v1.22 NitrOS-9. The latest version (v.2.00) is highly recommended.

4. What compatibility issues are there with Level III?

Level III will run all programs that will run under Nitro.

Distribution information for Nitro V.2.00 and Nitro Level III

For further information, contact the distributor (Frank Swygert) at:

FARNA Systems
Box 321
Warner Robins, GA 31099-0321
Phone: 912-328-7859
Internet: dsrtfox@delphi.com

Look for the ad in this issue for current prices. Do note that Nitro V.2.00 and Nitro Level III are SEPARATE products. There is a savings, however, if both are purchased together.

From: *Dennis Bathory-Kitsz*

Hi folks! I've been hiding out in Vermont, but since it's the 10th anniversary of my company Green Mountain Micro's demise, I thought it might be time to put in an appearance here.

About 150 copies of 'Learning the 6809' (book only) remain, which I'd be happy to offer at $10 postpaid to anyone interested. If at least 10 people also want the original tapes, I'd be pleased to make up a set of those as well.

One of these days I'll tell my own tale ... amusing indeed...

Dennis Bathory-Kitsz
RD 2 Box 2770
Cox Brook Road
Northfield, Vermont 05663

<bathory@maltedmedia.com>
Malted/Media:
http://www.maltedmedia.com/

NEW PRODUCT ANNOUNCEMENT:

Nickolas Marentes is proud to release

**** PAC-MAN ****

"A tribute to the great game"

For the Tandy Color Computer 3 with 512K RAM and Disk Drive

Finally! A version of the 1980 classic that is so similar to the original that you will think you ARE playing the original. Many of the original's features and characteristics have been included to make this game as faithful to the original Namco classic as possible. Fun, clean, violence free, 80's style entertainment for the whole family.

Features include:

* Most of the original sound effects
* Accurate replica of the original maze
* Accurate display of graphics and animations
* Many of the originals game play elements
* Coded in 100 percent 6809 assembly language
* Runs at 60 fps with 2 channel digital sound
* Keyboard and Joystick controls
* Reduced function DEMO version available as Freeware.
* Low price for full registered version ($20)

Get the best version of this historic game for your CoCo3 today!

Available from:

- USA -

Rick's Computer Enterprises, P.O. Box 276, Liberty, KY 42539
Internet Page: www.voicenet.com/~swyss/cfdm.html
E-mail: rcooper@kih.net

- AUSTRALIA -

Nickolas Marentes, P.O. Box 2003, Runcorn, 4113, QLD.
Internet Page: www.launch.net.au/~stauros/nickpage/ (FREE DEMO!!)
E-mail: N.Marentes@mailbox.uq.edu.au

Pac-man is the registered trademark and property of Namco/Midway.
Money collected is payment for the work involved in the development of the 6809 code.
The author has not seen or copied any of the original's Z-80 code.

MM/1 Survey!

Boisy Pitre

Dear MM/1 Owner,

I'm doing a survey to see how many MM/1 owners are out there, what kind of system they have, etc. This information is to be used to make a determination on what type of market there would be for certain products that I would like to release. Please take a few moments to fill out the following questions and send your responses via surface or e-mail.

Thank you!

Boisy G. Pitre

E-mail: boisy@microware.com

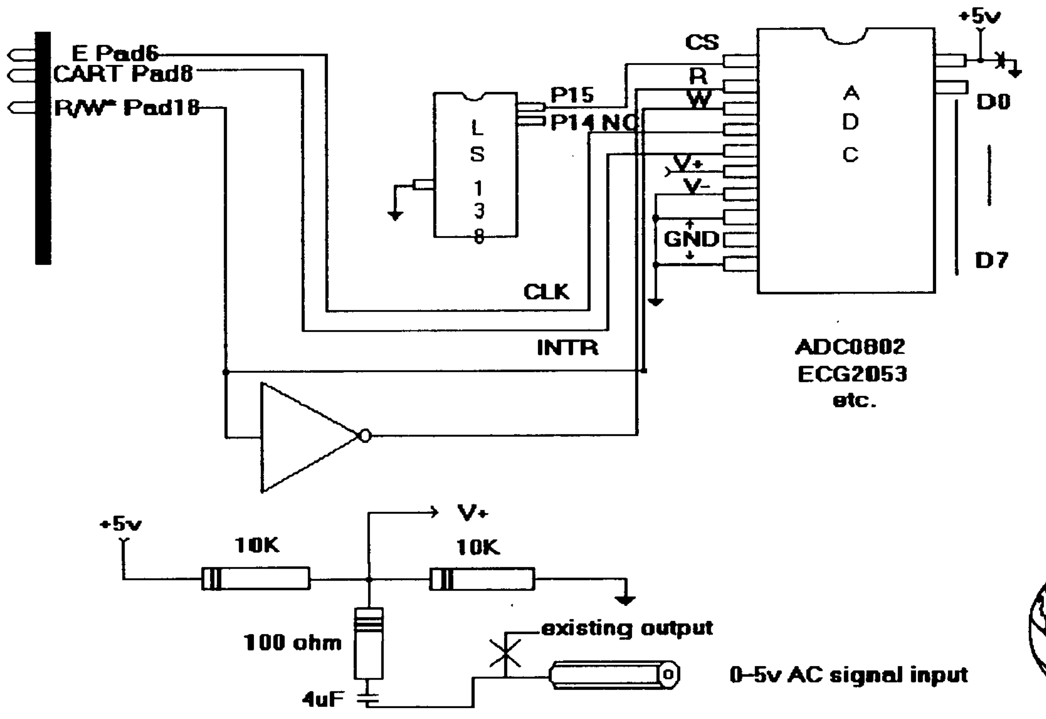
1. Which MM/1 do you own (68070 MM/1 or 68340 MM/1a)?
2. Do you have an I/O board with your MM/1?
3. How much RAM do you have? 1MB, 3MB or 9MB?
4. What size hard drive do you have?
5. Do you program in C or C++?
6. Do you own an Intel based PC? If so, what CPU type?
7. Please fill out the following fields.

Your Name:
Address:
City:
State:
ZIP:
Phone:
Preferred E-mail address:

Hacking Orchestra 90 Pak Part 2

Robert Gault

Editor: The following schematic was inadvertently left out of the "Hacking Orchestra 90 Pak" article by Robert Gault in the July/August issue (Volume 5 Number 1). What happened is that the schematic was embedded in a Word 6.0 document. The import filter that comes with PageMaker 5 doesn't recognize embedded graphics, so I didn't realize it was even there until Robert alerted me that it was missing. It took me a little time to figure out how to extract the file from the document so it didn't appear in the last issue. I'm sorry for the any inconvenience this may have caused! Robert informed me that the schematic would be needed by anyone interestd in the modification.



OS-9 Level II on a PC!

Walter Grossman

Setting up OS-9 to run from Jeff Vavasour's CoCo3 Emulator

What follows is an account of how I set up OS-9 on my CoCo III Emulator v1.6. The Emulator is an excellent product and should be used by everyone who likes the Color Computer and has a PC. The first thing that must be done is to make a complete print-out of the the file that comes with the emulator called COCO3.DOC. It is a very well written and complete manual for using the Emulator and should be read in its entirety before and while starting the Emulator.

I set up my Emulator in two directories: one I called COCO3X and the other OS-9. In this way I can access either DECB or OS-9 from the C:\ prompt easily via a simple BAT file which I will list later.

In the COCO3X directory I copied all the files on the Emulator distribution disk. Then following the instructions detailed in the manual (COCO3.DOC) I set up the ROM files to run my Extended ADOS 3 (or regular DECB) and any DECB applications. It is in this directory I set up all my "virtual disks," including all my OS-9 virtuals. (More about the OS-9 virtuals later.)

Once all the CoCo (or ADOS) ROM images are set up (again well covered in the manual) all CoCo functions including OS-9 can be run from this directory. OS-9 can be run by typing the customary "DOS" command from basic. To run the CoCo from the DOS c:\ prompt I wrote the following simple BAT file I named COCO.BAT and placed it in the C: directory:

```
@echo off
cd\coco3x
coco3x
cd\
```

This will return you to the c:\ prompt after quitting the Emulator. If you would rather stay in the COCO3X directory the last line can be eliminated. Staying in the COCO3X directory will allow you to backup virtual disks onto floppies via the dskini command in the COCO3X directory. However, it is often more convenient to write, copy, or backup to floppies right from disk basic because you can work with individual files on a floppy disk while still in the Emulator. However, after quitting the Emulator you are left only with the option of doing a complete backup of a floppy to virtual disk or from virtual to floppy. Individual files on either type of disk cannot be accessed from DOS.

After setting up the OS-9 directory I also copied all the files on the Emulator distribution disk to that directory, just as I did for the COCO3X directory. However, I neither set up nor copied any virtual disks to this directory for reasons I will explain later.

In the Emulator is a file named OS9BOOT.MOR. By renaming this file as described in the manual, you can start the Emulator booting immediately into OS-9 without needing the ROM images that DECB requires. Again, for this purpose I use a file named OS9.BAT which is placed in the C: directory as follows:

```
@echo off
cd\os9
coco3x
cd\coco3x
```

I also keep a copy of this BAT file in the COCO3X directory in case I want to start OS9 from there. This time, after quitting the Emulator you are placed in the COCO3X directory so that virtual disks can be backed up. Unlike the Emulator running DECB, the Emulator running OS-9 will not read from or write to a floppy disk. It will read from and write to virtual disks perfectly. Even though there may be no virtual disks in the OS-9 directory, the Emulator will automatically seek the virtual disks in the COCO3X directory. This way, all the virtual disks can be stored in one directory and be accessed by both the OS-9 and COCO3X directories.

Another consideration in setting up OS-9 on the Emulator is the fact that the Emulator will only read single sided floppies. So any OS-9 disk that is copied over to a virtual disk must be single sided. On the other hand, virtual disks can be formatted 40 or 80 track so that more data can be stored on one virtual disk. An 80 track virtual disk, however, must run in a virtual drive that has an OS-9 80 track descriptor in memory for that drive. More about this later.

OS-9 Level 2 can run at least three drives, so I set up my boot file to initialize /d0 as a 40 track double sided drive, /d1 as an 80 track single sided, and /d2 as a 40 track single. This way my boot-up and executables are in /d0, I can use either 40 or 80 track single sided disks in my data drives /d1 and /d2. These disks are readable both by the Emulator and the real CoCo.

Setting the system up can be a bit confusing. Most OS-9 systems on a real CoCo run double sided disks. To set up the same system on the Emulator requires these disks to be changed to single sided so the Emulator can read them. Then they can be changed back to double sided once in the virtual disk system.

The original disks to boot the Emulator into OS-9 must be prepared on a real CoCo. First a copy of 'os9boot' has to placed on a newly formatted single sided OS-9 disk with 'cobbler' or 'os9gen'. Then a startup file transferred to it, then a CMDS directory. In the commands directory must be 'format', 'shell', 'grfdrv', 'cobbler' and 'dsave' or 'wcopy' if you have it (Actually anyone serious about OS-9 really should have Level II Tools and Tools II by Keith Alphonso and at least the 'copy' utility from the Goldberg Utilities). In addition to this any additional commands or files that you can add to fill the single sided disk is fine. This disk can then be transferred to the COCO3X directory via the 'retrieve' command. Now you have a bootable virtual disk which if designated as D0 when the Emulator comes up will get you into OS-9. Bear in mind that the same descriptors will be placed in memory in the Emulator as would be placed in memory on a real CoCo.

At this point I formatted another virtual disk on the Emulator, from within OS-9, as a double sided 40 track disk. Next I made that disk into a boot disk using 'cobbler' and copying all the files and directories from the original single sided virtual boot disk (Once you verify that OS-9 will boot with the new double sided disk the single sided boot disk can be deleted from MS DOS.) After that, I brought over the rest of the files from my original CoCo double sided disk and copied them on to my new double sided virtual boot disk.

continued on page 19

HawkSoft

28456 S.R. 2, New Carlisle, IN 46552
219-654-7080 eves & ends MO, Check, COD; US Funds
Shipping included for US, Canada, & Mexico

MM/1 Products (OS-9/68000)

CDF $50.00 - CD-ROM File Manager! Unlock a wealth of files on CD with the MM/1! Read most text and some graphics from MS-DOS type CDs.

VCDP $50.00 - New Virtual CD Player allows you to play audio CDs on your MM/1! Graphical interface emulates a physical CD player. Requires SCSI interface and NEC CD-ROM drive.

KLOCK $20.00 - Optional Cuckoo on the hour and half hour!! Continuously displays the digital time and date on the /term screen or on all open screens. Requires I/O board, I/O cable, audio cable, and speakers.

WAVES vr 1.5 $30.00 - Now supports 8SVX and WAV files. Allows you to save and play all or any part of a sound file. Merge files or split into pieces. Record, edit, and save files; change playback/record speed. Convert mono to stereo and vice-versa! Record and play requires I/O board, cable, and audio equipment.

MM/1 SOUND CABLE $10.00 - Connects MM/1 sound port to stereo equipment for recording and playback.

GNOP $5.00 - Award winning version of PONG(tm) exclusively for the MM/1. You'll go crazytrying to beat the clock and keep that @#$%& ball in line! Professional pongists everywhere swear by (at) it! Requires MM/1, mouse, and lots of patience.

CoCo Products (DECB)

HOME CONTROL $20.00 - Put your old TRS-80 Color Computer Plug n' Power controller back on the job with your CoCo3! Control up to 256 modules, 99 events! Compatible with X-10 modules.

HI & LO RES JOYSTICK ADAPTER $27.00 - Tandy Hi-Res adapter or no adapter at the flick of a switch! No more plug and unplugging of the joystick!

KEYBOARD CABLE $25.00 - Five foot extender cable for CoCo 2 and 3. Custom lengths available.

MYDOS $15.00 - Customizable, EPROMable DECB enhancement. The commands and options Tandy left out! Supports double sided and 40 track drives, 6ms disk access, set CMP or RGB palettes on power-up, come up in any screen size, Speech and Sound Cartridge support, point and click mouse directory, and MORE OPTIONS than you can shake a stick at! Requires CoCo3 and DECB 2.1.

DOMINATION $18.00 - Multi-Player strategy game. Battle other players armies to take control of the planet. Play on a hi-res map. Become a Planet-Lord today! Requires CoCo3, disk drive, and joystick or mouse.

SMALL GRAFX ETC.

"Y" and "TRI" cables. Special 40 pin male/female end connectors, priced EACH CONNECTOR -		$6.50
Rainbow 40 wire ribbon cable, per foot -		$1.00
Hitachi 63B09E CPU and socket -		$13.00
MPI Upgrades for all small MPIs (satellite board) -		$10.00
Serial to Parallel Convertor with 64K buffer and external power supply -	**NOW ONLY**	**$28.00!!!**
Serial to Parallel Convertor (no buffer) and external power supply -	**ONLY**	**$18.00!!!**
2400 baud Hayes compatible external modems -		$15.00
Serial to Parallel Convertor or Modem cable (4 pin to 25 pin) -		$5.00

ADD $3.00 S&H FOR FIRST ITEM, $1.00 EACH ADDITIONAL ITEM

SERVICE, PARTS, & HARD TO FIND SOFTWARE WITH COMPLETE DOCUMENTATION AVAILABLE. INKS & REFILL KITS FOR CGP-220, CANON, & HP INK JET PRINTERS, RIBBONS & vr. 6 EPROM FOR CGP-220 PRINTER (BOLD MODE), CUSTOM COLOR PRINTING.

Terry Laraway
41 N.W. Doncee Drive
Bremerton, WA 98311
360-692-5374

The BlackHawk MM/1b

Based on the AT306 board from Kreider Electronics. Features built into the motherboard include:

- 16 bit PC/AT I/O bus with five slots
- MC68306 CPU at 16.67MHz
- 512K to 16MB of RAM with 30 pin SIMMs (4 sockets)
- IDE Hard Drive Interface (2 drives)
- 360K-1.44MB Floppy Drive Interface (2 drives)
- Two 16 byte fast serial ports (up to 115K baud)
- Bi-directional parallel printer port
- Real-time clock
- PC/AT keyboard interface
- Standard PC/AT power connector
- Baby AT size - fits standard PC case
- BASIC (resembles Microsoft BASIC)
- MGR Graphical Windowing Environment with full documentation
- "Personal" OS-9/68000 Vr 3.0 (Industrial with RBF)
- Drivers for Tseng W32i and Trident 8900 VGA cards
- Drivers for Future Domain 1680 and Adaptec AAH15xx SCSI cards
- OS-9/68000 Vr 2.4 with Microware C 3.2, Assembler, MW Basic (like Basic09), MW Debug, MW Programmers Toolkit
- UUCP from Bon Billson
- Ghostscript (software PostScript interpreter)
- Many other utilities and tools

Prices start at $400!
(motherboard,
Personal OSK, & MGR only)

BlackHawk
Enterprises, Inc.
756 Gause Blvd. #29
Slidell, LA 70458
E-mail: nimitz@stolY.com

The Embedded Programmer

Paul K. McKneely

Exploring the 68K's Advanced Interrupt Structure

This is one of the best features of the 68K which makes it more like high powered mini-computers (such the VAX) than simpler microprocessors (such as the Z80). An advanced interrupt structure is necessary if the system is to adequately handle a large number of devices efficiently. Indeed, the high-end 68K members such as the 68030, 68040 and 68060 do this rather well in comparison with much more expensive mini-computers from DEC and IBM.

Most computer systems have only a single main processor. Yet many of them seem to be able to keep a large number of complex programs running at once. It requires some sophisticated programming techniques to do this well and the 68K exception model does a lot to make this possible.

In this article we will be referring to some things that were discussed in the last article such as the Status Register (SR) and the Interrupt Stack Pointer (ISP). For quick reference, the Status Register is reproduced in figure 1.

Exception Vector Model

In most computers, software is usually divided into System Software and Application Software. System software generally runs in Supervisor Mode and application software generally runs in User Mode. Application software is written with the assumption that it has access to a set of abstract services that are provided by the system software. It is the system software that has to translate these requests into control of actual devices. In the performance of all of its duties, system software must be able to pre-empt application software to gain control of the processor. In many cases, higher priority system software can pre-empt lower priority system software to accomplish the same purpose. Any action that causes a program to be suspended to allow system software to gain control of the processor is called an Exception. There are two broad categories of exceptions:

1. Synchronous: This kind of exception happens when a program executes an instruction that causes it to be suspended. The most important kind of synchronous exception is called a System Call and is implemented in the 68K with the TRAP instruction. Other synchronous exceptions are caused by bus errors, page faults, divide-by-zero, etc.
2. Asynchronous: This kind of exception is called an Interrupt. This happens when a device needs prompt service. The servicing of the interrupting device is usually unrelated to the program that is suspended by the exception.

Each time an exception occurs, the processor goes through a sequence called Exception Processing that causes it to begin execution of the system software routine that has been designated to service the exception. The address of this routine is called an Exception Vector. The 68K supports up to 256 exception vectors and they reside in a location in memory called the Exception Vector Table (EVT). Each of these 4-byte values is identified by a 1-byte Vector Number which is its index into the table. Every time an exception occurs, a vector number is either generated internally by the processor or is obtained from an external device. During exception processing this vector number is multiplied by 4 (or shifted left by 2) before being added to the address of the base of the EVT. In the 68000, the EVT begins at location 0. In the 68010 and above, it can occupy any place in supervisor data space with the aid of the Vector Base Register (VBR).

A complete EVT occupies a contiguous block of 1024 bytes. The first 1/4 of these are either defined or reserved by Motorola. These first 64 locations generally contain vectors whose vector numbers are generated internally by the processor. The remaining 192 locations contain interrupt vectors. It is a good practice to place the vector of a default exception handler at all unused locations so that the processor will not crash when an unhandled exception occurs. If left blank, the processor will most likely begin execution at a location that contains arbitrary data and the programmer will not be able to recognize what happened. I use a little console routine that prints out a message and lets me know that an unhandled exception has occured.

Stack Frames

When an exception happens, the processor saves an internal copy of the SR then sets the Supervisor Bit (see reproduction, figure 1). If it is an interrupt, the IPM is updated with the IPL of the requesting device. It then pushes a block of data onto the Interrupt Stack (IStack) called a Stack Frame. This operation is done in hardware and it greatly simplifies programming of system software. There are a number of different formats but the last 6 bytes left on the top of the stack are always the same. The PC (Program Counter) and the saved copy of the SR are shown in figure 2.

A shortcoming of the original 68000 (and the 68008) is that this is the only information that is saved on the top of the IStack for many exceptions. For these processors, the exception handler is unable to determine the vector from which it was invoked. This makes it difficult to write exception handlers that can service more than one device. In all later 68K processors, two more bytes of information are always present in stack frames (figure 3, next page).

The Format code lets the processor know how much information needs to be restored and the Vector Offset allows the exception handler to determine the Exception Vector that it was invoked from. As it turns out, the 68000's shortcoming is not too serious and is only a nuisance when writing a default exception handler. Besides this problem, our kernel will not even need the Vector Offset. The Format code is useful for task switching because it implies the amount of extra data that was pushed

```
Status Register
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|T  S   | IPM |     |X N Z V C|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
^- System Byte -^* User Byte *^
```

figure 1

```
                 Stack Growth
                      ^
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ISP+00 |       Status Register         |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ISP+02 |                               |
       +-      Program Counter        -+
ISP+04 |                               |
       +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |   Additional Information      |
```

figure 2

```
                 Stack Growth
                      ^
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ISP+00 |        Status Register          |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ISP+02 |                                 |
        +-      Program Counter        -+
ISP+04 |                                 |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
ISP+06 | Format|     Vector Offset       |
        +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
       |    Additional Information       |
```

figure 3

onto the stack when the exception happened. When an exception handler finishes its job it can return to the suspended process by executing an RTE (ReTurn from Exception) instruction.

System Calls

Application programs typically depend on system services for communication with each other and the outside world. This is done in the 68K by using the TRAP instruction. There are actually 16 TRAP instructions and each one has its own exception vector in the EVT. The syntax for a TRAP call is of the form:

TRAP #n

The number n is an integer from 0 to 15. Corresponding vector numbers for the TRAP instructions are 32 to 47 ($20 to $2F). As with other kinds of synchronous exceptions, vector numbers are generated internally by the processor.

Interrupts

By their very nature, only one synchronous exception can be pending at a time since they are caused by the execution (or attempted execution) of the instructions of the currently running program. They can only happen as often as the instructions are executed and only one at a time. Interrupts differ from synchronous exceptions in that there may be many pending interrupt requests (IRQs) at one time. Servicing of interrupts is the key to efficient processing of a complex interactive software system. Because various interrupts typically have differing levels of urgency, the 68K uses an 8-level priority scheme which makes use of the Interrupt Priority Mask (IPM) of the Status Register (SR).

When a program runs, it has a pre-determined Interrupt Priority Level (IPL). That is, when it begins execution, it is provided with a pre-assigned 3-bit IPM value. Devices can only interrupt processes with an IPM that is less than their requested IPLs. Only programs running in supervisor mode can change the IPM. The 68K allows programs running in either supervisor or user modes to run at any of the 8 IPL levels (0 thru 7). Even though the hardware allows you to do this, our system will define some restrictions. In our system, when a user mode application runs, it will always have an IPL of 0. This is the lowest priority of any software in the system and is called Task Level. The IPL of system software will always be 1 or above (with the exception of the first few instructions of TRAP handlers when called from user mode). Level 1 is special and is called Kernel Level. This level can only pre-empt software that is running at Task Level. IPLs 2 thru 7 are called Interrupt Levels and they have the highest priorities.

Interrupt Vector Sourcing

An important difference between interrupts and synchronous exceptions is that interrupt vectors are not generated internally from the context of the instruction stream. In large minicomputers and mainframes, exception vectors are provided by the device that is requesting service. This is called Vectored Interrupts and is used a lot in larger 68K-based systems such as VMEbus based systems. Busses that are sophisticated enough to pass interrupt vectors to the processor during an interrupt acknowledge cycle are expensive. This mechanism is not trivial and it does not even exist in the ISAbus used in the IBM-type PC.

To reduce the complexity and cost in small and embedded systems, Motorola has provided a simpler method called Autovectored Interrupts. In the 68000, the IPL of the requesting device with the highest priority is passed into the processor at pins IPL0-IPL2. At every processor clock, these pins are sampled. When the same value appears on the pins for two clocks in a row, the requested level is recognized to be a valid request for service. At the end of each instruction's execution, this value is compared to the current IPM in the SR. If it is greater, then the interrupt is taken. When Vectored Interrupts are used, the device passes the processor an 8-bit number on the data lines during the interrupt acknowledge cycle. When Autovectored Interrupts are used, there is no interrupt acknowledge cycle and the processor uses the value on the IPL lines to look up one of seven vectors in the EVT. Value 0 is not interpreted as an interrupt request so the location that would otherwise be used for level 0 autovectored interrupt is instead, Spurious Interrupt.

Many embedded systems have greatly reduced needs for interrupting devices and some even do all of their I/O through polling. For those that do not need to implement Vectored Interrupts, the EVT can be reduced to only support the 64 reserved vectors (or fewer). This still leaves 7 Autovectored Interrupt vectors for the designer to use. This results in a savings of 768 (or more) bytes that would otherwise be wasted. Many of the 683XX embedded microcontrollers have pins that accept IRQ inputs directly from interrupting devices. Their IPLs are prioritized and encoded internally for comparison with the IPM of the SR.

Nested Interrupts

Many devices may be interfaced to a single 68K machine. Most devices represent physical processes going on in the real world that happen in parallel with program execution. When a device needs service, the currently running task is suspended (or interrupted) so that the device can be serviced. What the device is doing may be totally unrelated to the program that was interrupted. But most interrupt processing is very brief and provides that small amount of servicing that the device needs to get going again. Some devices require prompt servicing while others require servicing that is not so urgent. The 8-level IPL system allows the system designer and even the system owner to decide what the priorities are.

The 8-level IPL system allows mid-priority software to interrupt low-priority software and execute while still having high-priority interrupts remain enabled. This allows software to execute at its priority without blocking more urgent processing. In the example (figure 4), time procedes from left to right.

continued on page 19

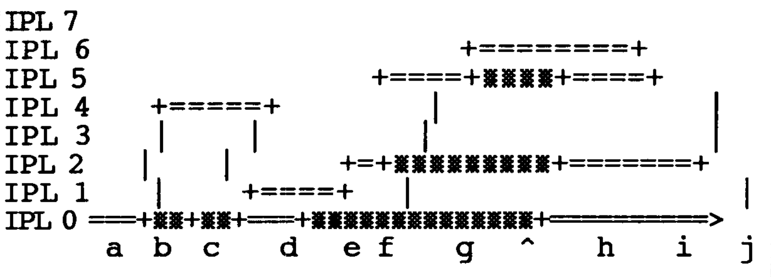


figure 4

CoCo3 Extended Memory Secrets Part 4

Herbert Enzman

Bit Plane Graphics Mode

The last installment covered the $FF9x registers and TEXT mode; now in this installment we will cover information that I have found about BIT PLANE GRAPHICS. I got into this because I wanted to do a graphics TITLE PAGE, and couldn't seem to find information on this subject either. So just like the previous information that I wanted, I was on my own again.

Included in this tutorial, is a MOD for standard DISK EDTASM to set it up for TR swapping, and a 'quick and dirty' GFX program, to demonstrate the various GFX modes. It is NOT a fancy mouse driven 'point and click' type program. You just have to use the arrow keys to move the cursor and enter the data VIA the keyboard; but it gets the job done. I used it to do the title page, and it worked OK. You can expand on it if you like, and maybe even add a mouse or joystick routine (I haven't the time).

Standard DISK EDTASM users will have to detour to the end of the tutorial to set up the EDTASM disk so it will TR swap; then return here to pick-up where you left off. EDT/ASM 6309 users can just continue.

To set the MMU for bit plane graphics mode, you just simply have to set $FF98 to $80. I'll cover a few more values for $FF98 during the GFX demo. The MMU will now interpret any data that it sees as GFX. TABLE 12 shows the various values to 'plug' into $FF99 to get the different color and resolution combinations. The 'line adjust' and the '$FF9E adjust' are just like that for the TEXT mode (as in the last installment), so keep them in mind when laying out a program. All this will fall into place as we walk thru the program. TABLE 13 is a list of the program commands and key usage. TABLE 14 demonstrates how to set up the data for the various color sets, which will also be explained later.

By changing the value in $FF9F, you can scroll the screen left and right, just like in PART 3; but there doesn't seem to be too much use for it in GFX mode, that I can see. The reason being that the entire picture will start to wrap-around (similar to a text screen); then a certain value will cause a double screen to appear. You will see what I mean during the "TRY" routine during the DEMO. The only way to see how any of this works, is to get your feet wet; so we will start with the the demo now.

If you are using DISK EDTASM and haven't made a modified disk, go to the end of this tutorial NOW and set it up! You will need it, as the standard DISK EDTASM will not work with this DEMO!!!!!!

GFX Demo Explanation

Type in LISTING 14 and save it to disk. Then just assemble it into memory, enter Z-BUG and execute it with "GDRAW". Leave all of the colors as they are for the moment so we will stay 'in sync'. Then, after the explanation, you can change the colors to what you like. Now that you have an error free assembly, enter Z-BUG, execute the program and follow along with the explanation.

The program has now set up the bit plane GFX mode and the screen is filled with "what ever". Press the <CLEAR> key to erase the screen. The cursor is the small orange dot in the upper left corner. Using the arrow keys, (E/A 6309 users can take advantage of the auto-repeat feature) move the cursor towards the center of the screen. The text inbetween the < > are KEYS to press (don't type them in). Now type the following: 'C0' <ENTER> <DOWN ARROW> '30' <ENTER> <DOWN ARROW> '0C' <ENTER> <DOWN ARROW> '03' <ENTER> <DOWN ARROW> and finally: 'FF' <ENTER> <DOWN ARROW>.

Keep in mind that the program is set up for a low resolution 4-color mode. The screen should look like example 1 (o = pixel x = background):

```
EXAMPLE 1
oxxx = $C0 = 11 00 00 00
xoxx = $30 = 00 11 00 00
xxox = $0C = 00 00 11 00
xxxo = $03 = 00 00 00 11
oooo = $FF = 11 11 11 11

EXAMPLE 2
oxxx = $80 = 10 00 00 00
xoxx = $20 = 00 10 00 00
xxox = $08 = 00 00 10 00
xxxo = $02 = 00 00 00 10
oooo = $AA = 10 10 10 10
```

The orange dots should match the location where the "O"s are in the text picture above. Now look at the BIN code that was placed there, and you will see a pattern. Everywhere there is a "11", there is an orange dot. The "11" equates to palette register $FFB3. What color is present in that register is what is going to be displayed at that particular location, of the 2 bit code (orange for this DEMO). The "00" equates to $FFB0, which is 'black', the background color. Now at the current cursor location, type the same sequence, but change the data to: 80 20 08 02 AA. You will see the same pattern, except that the dots are 'pink', which means that you have used palette register $FFB2 (example 2). Look at the "order/pixel" section in TABLE 14's 4 color mode, to see how the data byte for a screen location should be coded. In the 4 color mode, each 'byte' controls 4 pixels; each having a 2 bit code within the 'byte'. Move the cursor and type in the examples that are listed in the 4 color mode section of TABLE 14 and you should start to see the pattern emerge, of how to code the pixels.

Now for another example, we will try the 'psudo-zoom' feature. I call it the psudo-zoom because it is NOT a true zoom. As the picture enlarges, it gets squeezed in from the sides and just enlarges vertically. To use the psudo-zoom, press <SHIFT @>. If the display disappears, use the <SHIFT UP ARROW> key to scroll the screen up. By using the <SHIFT @> key, we are changing the value of $FF98 from $80 thru $83. The value $81 doesn't do anything, so the program skips it, but $82 and $83 causes the psudo-zoom.

Now that you have zoomed the picture, you can better see how the pixels are set. When you are finished with this 'mode', just press <SHIFT CLEAR> and you will be back to the normal screen. The use of the <SHIFT LEFT> and <SHIFT RIGHT> arrows will move the picture left / right. If you keep going in the same direction, the screen will start to scroll. This is done by adding / subtracting a value of 1 to the registers $FF9D / $FF9E. There isn't much use for it because the picture will 'wrap around' just like on a text screen, but it does show you something about these registers.

The real use of these registers is that by adding the "FF9E adjust" value to $FF9D / $FF9E; you can scroll the screen up or down (as seen during the psudo-zoom demo). As a matter of fact, you could scroll thru every memory block, just like the TEXT demo in the last installment. To see this, just use the <SHIFT UP> arrow key to start scrolling (E/A 6309's repeat key feature is great for this). As you scroll thru the blocks, the MMU will interpret all the data as just colorful pixels. If you get to an area that has a checkerboard pattern, that is caused by the data "FF / 00" that fills most unused memory blocks. So in reality, you could have pictures in ANY block (not just blocks $30 - $33) and display them at will, just by changing the value in $FF9D/FF9E. TABLE 8 showed this in the $FF9x text mode demo. I found out that the same thing is true for the GFX mode. By adding $04 to the value in $FF9D, you can have the start of a block in the upper left corner of the screen. This next demo will demonstrate this, so follow along with these instructions:

If you are running the program now, press <BREAK> to get back into text mode. Now, from Z-BUG's byte mode, type "START/" and change this value to 00. At the same time, change the 4 "MEMTAB" entries to 0, 1, 2, and 3. Now restart "DRAW", clear the screen and draw anything on the screen; anything at all. When complete, exit to text mode and change "START" to $10 and "MEMTAB" to 4,5,6, and 7. Restart "DRAW", clear the screen and draw another picture in a different location than the last one; anything at all, just so it is different. Exit back to Z-BUG again, change "START" to $20, "MEMTAB" to 8, 9, A, and B. Restart "DRAW" once again, clear the screen, and draw a third picture, again in a different location than the other two. The pictures can just be simple squares, rectangles or just lines. Now exit to text mode one last time, and type "GMOVIE". This routine will display all 3 pictures, one at a time so that they look like one. Yes, unfortunately there is a flicker, I was disappointed too! BUT it demonstrates that you can have pictures in other blocks of memory, other than the 4 that are reserved for GFX, just by changing the value of $FF9D. By setting the above data, we set up blocks 00,01,02 and 03 for GFX blocks, and told the MMU to start with block 00 in the upper left corner of the screen. The $20 told it to start with block 04, and the $30 told it to start with block 08 (examine TABLE 8).

I always wondered why $FF9D/9E contained $D800 for text and $C000 for GFX; but after making this table, I can now see why! Too bad Tandy didn't let us in on the secret in the service manual. It would have saved me a lot of time by not having me to experiment with the different values to make up the table. Oh well, such is life and the "powers that be" at Tandy.

You don't always have to reserve 4 blocks of memory for all GFX screens. That will depend on the resolution that you choose. The low resolution pictures will need less memory reserved, so you can use TABLE 12's screen end column to figure out how much memory to reserve for a particular size picture. For example: If you chose $11 as the resolution, the screen end is $AFFF. Since this table is based upon the screen start of $8000, then $AFFF-$8000=$2FFF (12K), only 1 and 1/2 blocks will be needed. Since you can't reserve 1 and 1/2 blocks, you must use 2; but this is still better than 4, if you need to save memory.

The TWO color mode will give you the BEST resolution because 1 byte controls 8 pixels (see TABLE 14 for pixel setup). To try it out, exit to Z-BUG and change "RESOL" to a 2 color code for $FF99; then restart "DRAW" (you DON'T have to re-assemble the program, just do it from Z-BUG). Now try some of the examples in the 2 color section of TABLE 14, and use the 'psudo-zoom' to examine the pixels to see how it works.

The 16 color mode gives you the lowest resolution; one byte controls only 2 pixels, but is very colorful. I kept the palette table short in TABLE 14's 16 color mode section, to save room. You can finish $FFB6 thru $FFBE, it is just simple 4 bit binary. To try it out, see the 2 color mode as above.

In TABLE 12, there are 2 resolution modes that have end addresses marked with "**". These two modes will need more than 4 blocks to display. The actual screen end address is $10C9F, which will extend $0C9F into the fifth block. DO NOT go past $FDFF in the demo when drawing, or you will crash the I/O area! You can't miss this area on the screen, it is where the "junk" is at the bottom of the screen. The "CLEAR SCREEN" routine for the 'DEMO' stops erasing at $FDFF, to leave the area marked and not crash the I/O area! However, if you map the blocks where the 5th block won't crash anything, then feel free to use it.

Once you have a picture that you would like to save, or work on later; you can do so by exiting to Z-BUG and save/load it with Z-BUG's disk commands (Pfilename.ext SSSS EEEE XXXX) and (Lfilename.ext). For the "P" command, use $8000 for the start (SSSS) and EXEC (XXXX) addresses. For the END filespec, use the "screen

TABLE 12 - GFX table information for $FF99

$FF99 VALUE (HEX)	LINE ADJUST (HEX)	LAST # SCREEN (HEX)	$FF9E COLORS (HEX)	ADJ (DEC)	screen size comments
14	50	BBFF	2	0A	640 x 191
10	40	AFFF	2	08	512 x 191
08	20	97FF	2	08	256 x 191
1D	A0	F7FF	4	14	640 x 191
19	80	DFFF	4	10	512 x 191
15	50	BBFF	4	0A	320 x 191
11	40	AFFF	4	08	256 x 191
1E	A0	F7FF	16	14	320 x 191
1A	80	DFFF	16	10	256 x 191
16	50	BBFF	16	0A	160 x 191
34	50	BE2F	2	0A	640 x 198
30	40	B1BF	2	08	512 x 198
28	20	98DF	2	08	256 x 198
3D	A0	FC5F	4	14	640 x 198
39	80	E37F	4	10	512 x 198
35	50	BE2F	4	0A	320 x 198
31	40	B1BF	4	08	256 x 198
3E	A0	FC5F	16	14	320 x 198
3A	80	E37F	16	10	256 x 198
36	50	BE2F	16	0A	160 x 198
74	50	C64F	2	0A	640 x 224
70	40	B83F	2	08	512 x 224
68	20	9C1F	2	08	256 x 224
7D	A0	**	4	14	640 x 224
79	80	F07F	4	10	512 x 224
75	50	C64F	4	0A	320 x 224
71	40	B83F	4	08	256 x 224
7E	A0	**	16	14	320 x 224
7A	80	F07F	16	10	256 x 224
76	50	C64F	16	0A	160 x 224

** end = $10C9F

NOTES:
1) LAST SCREEN = last screen address
2) ** = ADDRESS BEYOND $FDFF (don't mess with it passed here) I/O area passed this address!
3) SCREEN START = $8000 for this DEMO. Will be different depending upon where the GFX blocks ($30 - $33) are mapped.
4) SCREEN END = Will also differ depending upon where GFX blocks are mapped. Addresses shown in table are for this DEMO.
5) $FF99 VALUE for 80 column text screen = $1D.

TABLE 13 GFX DEMO Information

"DRAW" = draw a picture
- <BREAK> exits DRAW
- <arrow keys> move cursor about screen
- <ENTER> sets pixels at that location

"TRY" = to view GFX screen while playing with the $FF9x registers.
- 'GCLEAR' will clear the screen in this mode
- 'GTEXT' will exit this mode back to the text screen

KEYS
- <arrow keys> move cursor about screen
- <SHIFT @> toggles 'psudo-zoom'
- <SHIFT CLEAR> resets all registers to the default
- <SHIFT up/down arrows> scrolls picture up and down
- <SHIFT left/right> scrolls picture left and right (see text)

end" address for the resolution that you are using (from TABLE 12). FOR EXAMPLE: resolution $75 use:

Pgfx.pic 8000 C64F 8000

If using the 2 resolutions marked with "**", remember to use the address $FDFF for the end! When you want to load the picture, just use Z-BUG's "L" command. I would recommend to use the .PIC extension to remind you that it is a picture file (keeps down confusion).

The "TRY" routine is included so that you can change the $FF9x registers while viewing the GFX screen to see what happens when you change one. You will be "typing blind" in this mode, so TYPE carefully. Also, it will help a lot if you have something drawn on the screen to see what happens to the display. I used this mode quite a bit to collect the information in this installment. To use the "TRY" routine, run it from Z-BUG with "GTRY". To exit the "TRY" mode, just type the command "GTEXT" and you will be back into text mode (Z-BUG is in full charge when using the "TRY" mode, only the display is in GFX). To clear the screen in this mode, just type "GCLEAR". In this mode, you can change $FF9F to see what I meant earlier about how this register reacts. This routine should keep you busy for awhile. When changing $FF99 to see the different resolutions, I would recommend that you change the 'BORDER' register ($FF9A) to a color that is different from the background. This will aid you in seeing the drawing area of the screen, and how it changes with different resolutions.

HINT: When using the GFX routine to draw letters, buy some small square GRAPH paper and lay out the letter on it first. I used this method when doing the title page that I wanted and it worked out great. Just treat each square as a 'pixel'. Then, depending on the color mode that you choose, just group them into 2,4 or 8 sets across. This will aid you in getting the HEX code needed to put it on the screen. A little time consuming, but worth the extra time. I also found it useful to use\multiples of the 5x7 display. for example: to double the letter size use 10x14; triple size, 15x21; and so on.

Standard Disk EDTASM Setup

The first thing to do is make a back up copy of EDTASM just for this tutorial. Keep files on this disk to a minimum, so that you have plenty of room. Now start up EDTASM and enter LISTING 13. Save it to disk as "EDFIX.ASC" and when you have an error free listing; assemble it to disk as "EDFIX.BIN". Now you are ready to modify it. EXIT to BASIC and type in LISTING 12. Run LISTING 12 and when it is complete, restart EDTASM. You should now have an 80 column screen, with yellow text, on a black background. EDTASM will now be running in TR-1 instead of TR-0, so rembember this if you want to write programs using this copy. LISTING 3 from PART 1 will be needed to access the ROMs. I didn't bother to add a RESET routine or fix it to EXIT to DOS, so avoid doing this. I just wanted to set this up for you to use during this DEMO. Feel free to finish it if you like. You can EXIT to BASIC, with NO problems (it will cold start).

This MOD will not take up any space in the edit buffer, even though it is assembled there. Once EDTASM runs it (on start up); it is finished with it and it will be erased as soon as you start to write a program. The block swapping routines are moved down into $0500 (the 32 column screen area) which will not be needed any more, since you now have an 80 column screen. Keep this in mind if you decide to make this a permanent copy and add the reset routine.

You can change the screen colors to what ever you like by changing the listing and re-assembling (The listing is commented, so you easily find the section that sets the screen colors); But leave the colors the way they are

TABLE 14 Color mode information

2 COLOR MODE

PALETTE	EXAMPLES BIN	HEX	COLOR
0 = FFB0 = $00 = Black (B)	0000 0000	$01	BBBB BBBB
1 = FFB1 = $2C = Pink (P)	0000 0001	$01	BBBB BBBP
	0000 0010	$02	BBBB BBPB
	0000 0011	$03	BBBB BBPP
	1111 1010	$FA	PPPP PBPB

pixel order 1 2 3 4 5 6 7 8
palette code X X X X X X X X

4 COLOR MODE

PALETTE	EXAMPLES BIN	HEX	COLOR
00 = FFB0 = $00 = Black (B)	1000 1000	$88	PBPB
01 = FFB1 = $36 = Yellow (Y)	0100 0001	$41	YBBY
10 = FFB2 = $2C = Pink (P)	1100 1100	$CC	OBOB
11 = FFB3 = $34 = Orange (O)	0101 0101	$55	YYYY
	1001 0110	$96	PYYP
	1101 1000	$D8	OYPB

pixel order 1st 2nd 3rd 4th
palette code XX XX XX XX

16 COLOR MODE

PALETTE	EXAMPLES BIN	HEX	COLOR
0000 = FFB0 = $00 = Black (B)	0000 0001	$01	B Y
0001 = FFB1 = $36 = Yellow(Y)	0000 0010	$02	B P
0010 = FFB2 = $2C = Pink (P)	0011 0001	$31	O Y
0011 = FFB4 = $34 = Orange(O)	0010 0011	$23	P O
0100 = FFB5 = $12 = Green (G)	0100 0000	$40	G B
	0000 1111	$0F	B M

—> FFB6 - FFBE <— (same format)
(set the colors that you want)

1111 = FFBF = $2D = Magenta (M)

pixel order 1st 2nd
palette code XXXX XXXX

for the DEMO so we will stay "in sync" for the program explanation.

Well, this just about wraps up this installment. There is plenty of information for you to absorb, play with, and discover. I discovered a few new things myself while writing this, and had to edit the new material in, so I'm sure you will discover more. I hope you will share anything new that you find.

The next installment will cover TEXT and MENU windows that I have been using to dress up my routines. It will use almost all of the things covered by this tutorial series so far.

Herbert Enzman
memiser@delphi.com

LISTING 12 - BASIC program to modify DISK EDTASM (LISTING 13) on the disk.

```
10 PCLEAR1
20 LOADM "EDTASM.BIN"
30 LOADM "EDFIX.BIN"
40 SAVEM "EDTASM.BIN",&H1600, &H4B57,&H1600
```

LISTING 13 - DISK EDTASM MOD program

```
*** use filename "EDFIX.BIN" when assembling to disk.

**** Changes to EDTASM
     ORG   $1600
     LBRA  $4A6D     jump to setup routine
     FDB   LAST-GO+1 = new program 'LOAD' length
     ORG   $1D78
     JSR   $052A   addr. for 'clear screen' routine MOD
     ORG   $1D3F
     JSR   $0522   address for keyboard routine MOD
     NOP
     ORG   $1D99
     NOP
     JSR   $050C   address for CHROUT routine MOD
     ORG   $1DA4
     NOP
     JSR   $0504   address for keyboard routine MOD
     ORG   $2BFC
     FDB   $EF54   new offset for 'Q' command
     ORG   $2CDA
     FCB   $18     # of lines to display now = 24
**** Startup routine here
     ORG   $4A6D
     ORCC  #$50
     JSR   $F679   Setup 80 column screen
     LDD   #$3600  $36= (yellow) 00= (black)
     STA   $FFB8   set foreground
     STA   $FFB1
     STB   $FFB0   set background
     STB   $FF9A   set border
     LDD   #$2C34  $2C= pink  $34= orange
     STD   $FFB2
     LDX   #$0500  = where to move routines
     LEAY  SUBS,PCR  point to routines to move
     LDB   #$7F    = byte count
     LBSR  MOVEIT    move data now
     LDX   #$E0E9  = SECB register table
     LEAY  MEMTAB,PCR  table to move
     LDB   #8      = byte count
     PSHS  Y,B
     LBSR  MOVEIT    move data now
     PULS  Y,B
     LDX   #$FFA8  = hardware register table
     LBSR  MOVEIT    move data now
**** modify 'DOS' here
     LDD   #$7E8E  = 'JMP' 'LDX' opcodes
     STA   $0D1D
     STB   $0D2E
     STB   $0E80
     STB   $0CF4
     LDD   #$0532  = new disk routine addr. for TR swap
     STD   $0E9C
     STD   $0D1E
     LDD   #$12BD  $12=NOP  $BD= JSR
     STD   $0E9A
     LDD   #$00EA  = DSKCON buffer
     STD   $0D2F
     STD   $0E81
     STD   $0CF5
     LDD   #$053A  = NEW IRQ routine address
     STD   $0F56   set it for DOS
     LDA   #1      ##
     STA   $FF91   ## set to TR=1
     ANDCC #$AF
     JMP   $1605   back to standard EDTASM setup
SUBS   FDB  $A1B1  keyboard routine address
     FDB   $8C46   80 column 'clear screen' routine
KEY    PSHS  B,X,Y,U
     LDY   #$A000  get ready for keyboard routine
     BRA   DOIT    go do it
DISPLA PSHS  B,X,Y,U  save these registers for return
     LDY   #$A002  get ready for 'CHROUT' routine
DOIT   CLR   $FF91   swap TR to '0'
     JSR   [,Y]    do routine
     PSHS  CC
     LDB   #1      ##
     STB   $FF91   ## set TR=1
     PULS  CC
     PULS  B,X,Y,U,PC   back to EDTASM
KEY2   PSHS  B,X,Y,U
     LDY   #$0500  get ready for keyboard routine
     BRA   DOIT    go do it
CLEAR  PSHS  B,X,Y,U
     LDY   #$0502  get ready to clear screen
     BRA   DOIT    go do it
DISK   PSHS  B,X,Y,U
     LDY   #$C004  get ready for DSKCON
     BRA   DOIT    go do it
**** moved IRQ service routine here
IRQ    LDA   $FF02
     LDA   $0985
     BEQ   NO
     DECA
     STA   $0985
     BNE   NO
     LDA   $0986
     ANDA  #$B0
     STA   $0986
     STA   $FF40
NO     RTI
**** EXIT to BASIC routine here
EXIT   CLRB          ##
     TFR   B,DP    ## set DP for BASIC
     CLR   $0071   clear warm start flag
     CLR   $FF91   set TR=0
     LDS   #$03D7  set stack for BASIC
     JMP   $8C1B   to COLD start routine
**** move data routine
MOVEIT LDA   ,Y+     = source
     STA   ,X+     = destination
     DECB          count bytes moved
     BNE   MOVEIT  loop until all moved
     RTS
***** TR 1 memory table
MEMTAB FDB   $3839
     FDB   $3A3B
     FDB   $3031
     FDB   $3233
LAST   EQU   *
GO     EQU   $1600
     END
```

LISTING 14 - GFX DEMO program

```
GO     NOP
SAVE   RMB  1      storage TEMP
BUFF   RMB  4      data entry buffer
FLAG   RMB  1      flag for 'new data' entry
COUNT  RMB  1
TMP98  FCB  $80    $FF98 RAM image
TMP9D  FCB  $C0    $FF9D RAM image
TMP9E  FCB  00     $FF9E RAM image
RESOL  FCB  $11    picture resolution (we'll use this to
start)
ADJ    RMB  2      line adjust value (set by resolution
chosen)
RANGE  RMB  2      $FF9E adjust value (set by
resolution)
CURSOR FCB  $C0    Cursor value
START  FCB  $C0    screen start offset (see text)
MEMTAB FCB  $30    1st GFX block for "movie" (see
text)
     FCB   $31     2nd GFX block for "movie" (see text)
     FCB   $32     3rd GFX block for "movie" (see text)
     FCB   $33     4th GFX block for "movie" (see text)
     **** keyboard routine for TR swap
KEY    ORCC  #$50    disable interupts
     PSHS  B,X,DP  save these
     CLR   $FF91   set TR=0
LOOK   JSR   $A1B1   check for key
     BEQ   LOOK    loop until one down
     LDB   #1      ** set TR=1
     STB   $FF91   **
     ANDCC #$AF    enable interupts
     PULS  B,X,DP,PC  and return
     ** set GFX mode, registers, then clear picture area
CLRPIC LDA   #$80    = GFX value
     STA   $FF98   set GFX mode
     LDA   RESOL   get desired resolution
     STA   $FF99   set it
     LDA   #$C0    set screen display for GFX blocks
     STA   $FF9D   set it
     LDX   #$8000  point to start of screen
     LDB   #$0     = backgound color
AGAIN  STB   ,X+     erase screen
     CMPX  #$FDFF  end of block 4?
     BNE   AGAIN   no, loop until all erased
     LDD   #$0036  $00= black   $36= yellow
     STD   $FFB0
     LDD   #$2C34  $2C= pink    $34= orange
     STD   $FFB2
     RTS
           **** reset to text screen
RESET  LDA   #$03    = text
     STA   $FF98   set text screen
     LDA   #$15    text screen value
     STA   $FF99   set for text screen
     LDA   #$D8    ### set text screen
     STA   $FF9D   ###  display
     CLR   $FF9E   ###    address
     RTS
**** set GFX screen, figure line adjust and $FF9E adjust
GFX    LDA   RESOL   get chosen resolution
     ANDA  #$0F    drop 1st 4 bits
     LEAX  ADJTAB,PCR  point to lookup table
NEXT   CMPA  ,X++    match?
     BNE   NEXT    no, loop until a match found
     LDB   -1,X    get next table offset value
     LEAX  FIXTAB,PCR  point to 'adjust' table
```

```
       ABX          add offset
       LDD  ,X      get adjust values
       CLR  ADJ     ###
       STA  ADJ+1   ###
       CLRA         ###
       STD  RANGE ### set adjust values for resolution
       LDA  #$80    **
       STA  $FF98   ** set GFX mode
       LDA  RESOL   get resolution
       STA  $FF99   set it now
       LDA  #$C0    = GFX screen start
       STA  $FF9D   set it
       RTS          return
  *** CLEAR SCREEN from 'CLEAR' key during 'TRY'
CLEAR  BSR  CLRPIC
       SWI
  *** reset to TEXT screen during 'TRY'
TEXT   BSR  RESET
       SWI
  *** 'TRY' routine (see table 13)
TRY    BSR  GFX
       SWI
  *** DRAW picture routine entry point
DRAW   LBSR GFX     set GFX mode
       LDX  #$8000 = screen start
SHOW   LDA  CURSOR  get cursor
       CLR  FLAG
       LDB  ,X      get char. from screen
       STB  SAVE    save it for later
       STA  ,X      put cursor on screen
       LEAU BUFF,PCR  point to input buffer
POLL   LBSR KEY     go scan keyboard
       CLR  COUNT   clear table count
       LEAY TAB,PCR  point to 'key' table
P2     TST  ,Y      end of table?
       BMI  P4      jump if table end
       CMPA ,Y+     match?
       BEQ  P3      jump if match
       INC  COUNT   bump counter
       BRA  P2      loop for more
P3     LEAY TAB2,PCR  point to 'routine' table
       LSL  COUNT   divide by 2
       PSHS B       save 'B'
       LDB  COUNT   get table count
       LEAY B,Y     adjust pointer
       LDD  ,Y      get routine offset
       LEAY GO,PCR  point to constant
       LEAY D,Y     add offset for routine address
       PULS B       get 'B'
       JMP  ,Y      go do the routine
P4     CMPA #$30    ignore keys below '1'
       BLO  POLL
       CMPA #$39    if higher than '9', check for A - F
       BHI  LETTER
       STA  ,U+     store into input buffer
       BRA  POLL    back to keyboard
LETTER CMPA #$41
       BLO  POLL    exit if lower than 'A'
       CMPA #$46
       BHI  POLL    exit if higher that 'F'
       STA  ,U+     store into input buffer
       BRA  POLL    back to keyboard
BREAK  CLR  ,X      erase cursor
       LBSR RESET   go reset to TEXT screen
       SWI          back to Z-BUG
UP     TST  FLAG    was data put to screen?
       BNE  UP1     yes, then just move cursor
       LDB  SAVE    get what was there previous
       STB  ,X      put it back on screen
UP1    PSHS D       save registers
       TFR  X,D     current cursor location
       SUBD ADJ     adjust up 1 line
       TFR  D,X     set screen pointer
       PULS D       get previous data
       BRA  SHOW    go display cursor in new location
DOWN   TST  FLAG    (see 'UP')
       BNE  DWN
       LDB  SAVE
       STB  ,X
DWN    PSHS D
       TFR  X,D
       ADDD ADJ
       TFR  D,X
       PULS D
       LBRA SHOW
LEFT   TST  FLAG    (see 'UP')
       BNE  LEFT1
       LDB  SAVE
       STB  ,X
LEFT1  LEAX -1,X    move cursor left
       LBRA SHOW
RIGHT  TST  FLAG
       BNE  RGHT
       LDB  SAVE
       STB  ,X
RGHT   LEAX 1,X     move cursor right
       LBRA SHOW
  **** erase screen then go back to 'DRAW'
ERASE  LBSR CLRPIC  erase screen
       LBRA DRAW    back to draw
  **** psudo-zoom routine
PZOOM  PSHS D
       LDA  TMP98   get $FF98 RAM image
TF2    INCA         bump zoom
       CMPA #$81    since this one doesn't do anything,
       BEQ  TF2     bump one more time
       CMPA #$83    highest value?
       BLS  TF1     no, then ignore next
       LDA  #$80    reset to NO zoom
TF1    STA  $FF98   set register
       STA  TMP98   save RAM Image
       PULS D
       LBRA POLL    back to keyboard
  **** move picture right
MVLEFT PSHS D
       LDD  TMP9D   get RAM image of $FF9D/FF9E
       SUBD #1      move picture left
       STA  $FF9D   ** set register
       STB  $FF9E   **
       STD  TMP9D   save RAM image
       PULS D
       LBRA POLL    back to keyboard
  **** scroll picture down
DSCROL PSHS D
       LDD  TMP9D   get RAM image
       SUBD RANGE   scroll picture down
       STA  $FF9D
       STB  $FF9E
       STD  TMP9D   save RAM image
       PULS D
       LBRA POLL
  *** reset all registers from 'scroll', 'zoom' etc.
NEW    PSHS D
       LDA  #$80    ####
       STA  $FF98   ####
       STA  TMP98   #### reset to GFX mode
       LDD  #$C000  ***
       STA  $FF9D   ***
       STA  TMP9D   ***
       STB  $FF9E   ***
       STB  TMP9E   *** reset GFX screen start
       CLR  $FF9F   reset horizon. resol. register
       PULS D
       LBRA POLL    back to keyboard
  **** move picture right
MVRGHT PSHS D
       LDD  TMP9D
       ADDD #1
       STA  $FF9D
       STB  $FF9E
       STD  TMP9D
       PULS D
       LBRA POLL
  **** scroll picture UP
UPSCRL PSHS D
       LDD  TMP9D
       ADDD RANGE
       STA  $FF9D
       STB  $FF9E
       STD  TMP9D
       PULS D
       LBRA POLL
  **** change data from keyboard into hex, display on screen
SEND   LEAU BUFF,PCR  point to entry buffer
       PSHS B
       LDD  ,U      get ASCII data
       CMPA #$39    higher than 9?
       BHI  FIX1+1  must be letter
       SUBA #$30    make single digit
FIX1   CMPX #$8037  (SUBA #37)
       CMPB #$39    higher than 9?
       BHI  FIX2+1  must be letter
       SUBB #$30    make single digit
FIX2   CMPX #$C037  (SUBB #37)
       LSLA         **
       LSLA         **
       LSLA         **
       LSLA         ** move digit left
       ANDB #$0F    drop left 4 bits
       PSHS B       get ready for add
       ADDA ,S+     add the two
       STA  ,X      store it to screen
       INC  FLAG    flag that data was stored to screen
       PULS B
       LBRA POLL    back for more keys
                **** TABLES ****
  **** KEY TABLE ****
TAB    FDB  $0308
       FDB  $090A
       FDB  $0C0D
       FDB  $1213
       FDB  $155B
       FDB  $5C5D
       FDB  $5E5F
       FCB  $FF
  **** ROUTINE TABLE ****
TAB2   FDB  BREAK-GO   'break' key
       FDB  LEFT-GO    'left arrow' key
       FDB  RIGHT-GO   'right arrow' key
       FDB  DOWN-GO    'down arrow' key
       FDB  ERASE-GO   'clear' key
       FDB  SEND-GO    'enter' key
       FDB  POLL-GO    'shift 0' key
       FDB  PZOOM-GO   'shift @' key
       FDB  MVRGHT-GO  'shift right arrow' key
       FDB  DSCROL-GO  'shift down arrow' key
       FDB  NEW-GO     'shift clear' key
       FDB  MVLEFT-GO  'shift left arrow' key
       FDB  UP-GO      'up arrow' key
       FDB  UPSCRL-GO  'shift up arrow' key
ADJTAB FDB  $0800
       FDB  $0002
       FDB  $0102
       FDB  $0404
       FDB  $0504
       FDB  $0604
       FDB  $0906
       FDB  $0A06
       FDB  $0D08
       FDB  $0E08
FIXTAB FDB  $2008
       FDB  $4008
       FDB  $500A
       FDB  $8010
       FDB  $A014
```

```
**** "movie" routine demo - See text BEFORE
      trying to run this.
***  It will display 3 GFX pictures to look like 1.
      It will just run  for a short time and then exit
      to text mode.
MOVIE  LBSR  GFX     set GFX mode
       ORCC  #$50    disable interupts
       CLRB          clear offset to block "0"
       LDX   #0      clear counter
ANAM   STB   $FF9D   set GFX blocks
       LEAX  1,X     decrement counter
       CMPX  #$FFFF  done?
       BEQ   QUIT    yes, then quit
       ADDB  #$10    point to block "4"
       CMPB  #$20    done with display?
       BLS   ANAM    no, loop until all have been diplayed
       CLRB          recycle back to block "0"
       BRA   ANAM    loop until timer runs out
QUIT   LBSR  RESET   set back to text mode
       ANDCC #$AF    enable innterupts
       SWI           back to Z-BUG
       END
```

OS-9 Level II on a PC!

continued from page 10

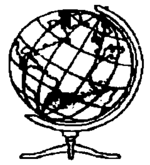
(Again, this is accomplished by making a single sided disk on the CoCo, transferring it over to a single sided virtual disk with 'retrieve', then copying the files to the permanent double sided virtual disk from within OS-9 and deleting the single sided virtual disk from MS DOS)

This ultimately gave me a virtual boot disk identical to the CoCo boot disk I started with. But remember that the characteristics of all the drives on the Emulator will be determined by the descriptors in memory. So you cannot format a drive on the Emulator that you do not have a descriptor for in memory. I always format data disks (/d1 and /d2) that I want to be compatible both with the Emulator and the CoCo as single sided so they are easily transferred back and forth between the two.

At this point you may ask what is the advantage of setting up two Emulator directories in the first place. Well, one major advantage is the ability to leave a boot-program virtual disk in D0 with most or all of your program files, utilities, procedure files, dictionaries, etc. and just about never have to change it. A large virtual disk such as a double sided 80 track can be formatted and used as the boot disk and can hold an enormous amount of OS-9 files. Better yet now with the Emulator v1.6, Mr. Vavasour gives us a virtual hard drive along with the utilities (driver and descriptor) to set it up. Now a "hard drive boot disk" can be made, placed in virtual drive D0 and left there just about permanently. From this the virtual hard drive drive can be accessed and once done becomes the default drive after bootup.

This may seem to be a bit of commotion to get the Emulator set up to do OS-9, but it is well, well worth the effort once it is set up. I have been using my Emulator this way for quite a while and have found it to be very efficient and enjoyable as well. If there are any questions please feel free to contact me.

Wally Grossman
17810 Allien
Cleveland, OH 44111



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 support, the longer it will be here!

Embedded Programmer

continued from page 13

Say a program is running at Task Level (IPL 0). The double-line represents the currently executing process while the single-line represents a suspended process. At the completion of the an instruction boundary (a), the processor sees that an interrupt of level 4 is being requested which is higher than the current IPM. An interrupt occurs and the level 4 ISR runs. Mid-way through its execution, an IRQ of level 1 shows up (b). Because it is lower priority, the level 4 ISR is allowed to run to completion (c) and the level 1 ISR is not acted on until a return from interrupt is executed returning the processor back to Task Level. At this time the level 1 interrupt occurs and runs to completion (d).

At (e) a level 2 interrupt is requested and is acted on at the next instruction boundary. Then at (f) a level 5 interrupt is requested and this exception is taken suspending the level 2 ISR. At (g) a level 6 interrupt occurs and the level 5 ISR is suspended. At this point (^) there are three processes stacked on the ISP and one is running (figure 5).

After this, the IPL6 ISR completes (h) allowing the IPL5 ISR to continue. After IPL5 ISR completes (i) then the IPL2 ISR resumes. Finally, the IPL2 ISR finishes (j) and the task (IPL 0) is running once again. These events typically happen very fast and a user is not able to sense any suspension of his program's execution. By adjusting the priorities of your various ISR's, you can tune your system to its highest performance.

```
      Stack Growth
           ^
+ ********+******** +
| IPL 6:Running     |
+ ***************** +
| IPL 5:Suspended   |
+ ***************** +
| IPL 2:Suspended   |
+ ***************** +
| IPL 0:Suspended   |
+ ***************** +
```

figure 5

Next time...

In the next article we'll look at the life and death of the boot process. If you have any comments or requests, please feel free to write me at either <gecko@onramp.net> or at the address given below:

Paul K. McKneely
technoVenture, Inc.
P. O. Box 5641
Pasadena, Texas 77508-5641

RGBoost - $15.00

If you want to speed up DECB easily, install an Hitachi 6309 and get RGBoost. This patch for DECB uses the extra 6309 functions for up to a 15% gain in overall speed. It is compatible with all programs tested to date! Save an additional $5 by purchasing RGBoost along with one of my other products listed below!

EDTASM6309 v2.02 - $35.00
Patches Tandy's Disk EDTASM to support Hitachi 6309 codes! Supports all CoCo models, including stock 6809 models. CoCo 3 version uses 80 column screen, runs at 2MHz. YOU MUST HAVE A COPY OF DISK EDTASM. This is a PATCH ONLY! It will not work with "disk patched" cartridge EDTASM

CC3FAX - $35.00
Receive and print weather fascimile maps from shortwave! The US weather service sends them all the time! Requires 512K CoCo3 and shortwave receiver. Instructions for simple cable included.

HRSDOS - $25.00
Move programs and data between DECB and OS-9 disks! Supports RGB-DOS - move files easily between DECB and OS-9 partitions! No modifications to OS-9 modules required.

DECB SmartWatch Drivers - $20.00
Access your SmartWatch from DECB! Adds function to BASIC (DATE$) for accessing date and time. *Only $15.00 with any other purchase!*

Robert Gault
832 N. Renaud
Grosse Pointe Woods, MI 48236
313-881-0335
Please add $4 S&H per order

STRONGWARE

Box 361 Matthews, IN 46957 Phone 317-998-7558

CoCo 3 Software:

Soviet Bloc	$15
GEMS	$20
CopyCat	$5
HFE- HPrint Font Editor	$15

MM/1 Software:

Graphics Tools	$25
Starter Pak	$15
BShow	$5
CopyCat	$10
Painter	$35

People, I have run across some tough times lately! Those who have outstanding orders please be patient... I'm working on them! I've recently moved (again!) and hope to get everything back on track by the beginning of the year. Thanks for all your support in the past. I know I've been unresponsive lately, but hope I can make up for it! I still have a lot of Tandy original software and some hardware, so let me know what you need! I may be working an arrangement soon with FARNA for distributing some of my hard and soft wares. I'm sure Frank will keep you posted! I still have the *fastest serial port ever available for the CoCo ...*

Fast 232 - $79.95
Daughter Board - $45.00
(for 2nd serial port)

Check with me for complete disk drive systems, misc. hardware items, hardware repairs, and hard to find new and used CoCo software!

CoNect 1629 South 61st Street
West Allis, WI 53214
(pulland@omnifest.uwm.edu)
414-328-4043

ADVERTISER'S INDEX

BlackHawk Enterprises	*11*
Chicago CoCoFest	*5*
CoNect	*BC*
FARNA Systems	*6, 19, BC*
Robert Gault	*BC*
Hawksoft	*11*
Dennis Kitsz	*8*
Small Grafx	*11*
StrongWare	*BC*

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!