INSIDE OS9 LEVEL II

THE *INSIDE* STORY OF OS9 FOR THE TANDY COLOR COMPUTER 3

by
Kevin K. Darling

INSIDE OS9 LEVEL II

THE *INSIDE* STORY OF OS9 FOR THE TANDY COLOR COMPUTER 3

by
Kevin K. Darling

PUBLISHED BY
Frank Hogg Laboratory, Inc.
770 James Street
Syracuse, New York 13203

Printed and bound in the United States of America

Distributed by Frank Hogg Laboratory, Inc.

Disclaimer

The opinions in this book are strictly those of the author and do not necessarily represent the views of Frank Hogg Laboratory, Inc.

The authors have exercised due care in the preparation of this book and the programs contained in it. Neither the author or the publisher make any warranties either express or implied with regard t o the information and programs contained in this book. In no event shall the authors or publisher be liable for incidental or consequential damages arising out of the furnishing, performance, or use of any information and/or programs.

First edition published in April 1987

table of contents

5-MISCELLANEOUS

6-SOURCES

7-REFERENCE

INSIDE OS9 LEVEL II

Introduction

FOREWORD

Around the middle of Febuary, Frank Hogg asked me to do a "little something" on Level Two OS-9 for the CoCo-3. This is the result, a compilation of old and new notes I and others had made for ourselves.

Organizing anything about OS-9 is tough, since each part of it interacts closely with the rest. In the end, I decided to simply present information as a series of essays and tables. Some of these are ones that I had made for L-I, but apply equally well to L-II. Maybe in a half year or so we'll come out with a second edition, but we really wanted to help people out NOW.

To me, at least, it is very like being blind not knowing exactly what occurs during the execution of a program that I have written. For that reason, I have taken a look at OS-9 on the CoCo from the inside out.

The idea is that if you can figure out what's happening on the inside, you have a better chance of knowing what to do from the user level. In essence, this whole collection is a reference work for myself and my friends out there like you.

Level-II wasn't out yet at the beginning of this writing, and I had not seen the Tandy manual until the end, so please bear with me if things have changed somewhat.

In general, I will not duplicate explanations provided by the Tandy manuals, Microware manuals or the Rainbow Guide. Instead, my intention is to enhance them. You should get them, too. Dale Puckett and Peter Dibble are working now on a book about windows for the user. I will be doing more on drivers soon.

This reference work is the result of many hours of studying and probing by myself and others. Hopefully, it will save you at least some of the time and trouble that we have had. Since this is meant as part tutorial, part quick reference, some tables may occur more than once as I felt necessary.

Special thanks are due to Frank Hogg, for publishing this and for being "patient" with delays. I also owe a lot to the many people on CompuServe's OS-9 Forum, who keep asking the right questions.

Thanks also to Pete Lyall for letting me use his excerpts on login, Kent Meyers for much help on internals, and to Chris Babcock for delving into the fonts for us.

And, of course, none of this would have been done without the support and love of my dearest friend and sweetheart, Marsha. Thank you, Sweet Thang!

I hope it helps. Best wishes, and Have Fun.
Kevin K Darling - 30 March 1987

OVERVIEW OF OS9

The following is all of OS9 in one spot:

UNIVERSAL SYSTEM TABLES:

Direct page vars -	table pointers, interrupt vectors
Memory bitmaps -	maps of free / in-use memory
Service dispatch tables -	vectors for SWI2 system calls
Module directory -	pointers to in-memory modules
Device table -	info on used devices (/D0,/P,etc)
IRQ polling table -	vectors interrupts to drivers

PROCESS INFORMATION:

Process descriptors -	process specific information
Path descriptors -	I/O open file information
Driver static storage -	device driver constant memory

PROGRAM MODULES:

User programs -	your program
Kernal -	handles in-memory processing
Ioman -	controls I/O resources
File Mgrs -	file handling and editing
Drivers -	data storage and transfer
Device descriptors -	device characteristics

SIMPLE SYSTEM MEMORY MAP

```
00000-01FFF   System Variables
02000-        Free memory, bootfile
     -7DFFF    video memory
7E000-7EFFF   Kernal
7F000-7FFFF   I/O and GIME
```

THE MAIN PLAYERS:

Modules	Responsibilities
REL,BOOT	. Reset hardware and Boot
OS9P1 OS9P2	. Initialization of system Handling of most SWI2 service calls (except I/O) Memory management and process control Module directory upkeep, module searching Allocation of process descriptors
IOMAN	. Handling I/O related SWI2 service calls Allocation of path descriptors IRQ polling table entries Device IRQ polling Device table entries for desc, driver, filmgr Queuing processes trying to use same path desc Allocation of driver static memory Copying device desc init table to path desc Calling file mgr for I/O calls
RBFMAN SCFMAN PIPEMAN	. Allocation of data buffers File & directory allocation and management Edit, seek, read, write of file Queuing processes trying to use same device
CC3DISK CC3IO PRINTER RS232	. Allocation of verify buffer Read / write of data buffers from / to device Device interrupt handling Device status / error monitoring

---

REL	- Resets hardware, calls OS9p1
INIT	- Data module containing system constants
BOOT	- Load OS9Boot if initial dir's, paths fail
CC3GO	- CHX CMDS, Startup, Autoex, Shell
CLOCK	- System timekeeping, VIRQ's, Alarm calls

Process Descriptors	- info on each process
Path Descriptors	- info local to each I/O path
Device Table	- device memory, desc, filmgr, driver
Polling Table	- device status address, driver IRQ vector
Module Directory	- address, user cnt of program modules

MULTI-TASKING PRINCIPLES

The power of the 6809's addressing modes enables the m/l programmer to easily write code that will execute at any memory address. Furthermore, if the code is written to access program variables by offsets to the index registers, more than one user can execute that code as long as he has his own data area.

The point of all this is that the 6809 made it easy for Microware to write an operating system that can load a program anywhere there is enough contiguous memory, assign the user a data space, and through SWI2 (trap) calls, access system I/O and memory resources.

Now, since we know that we can be processing code and sharing the 64K memory space with other programs, we can allow more than one program / user at more or less the same time by switching between the processes fast enough to appear to each user that he has his own computer.

How often is fast? In some other multi-tasking systems, each process is responsible for signaling to the operating system kernal that it was ready to give up some of its CPU time. The advantage of this method was that time-critical code wasn't interrupted. (OS9 users can simply shut off interrupts if this is necessary.) But this method depends on the user to write the switching signal into his code so that it was hit often enough to give other processes a chance to run.

In OS9, there is always a system 'clock' that interrupts the 6809 about 10 times a second, and causes the next process to be given a CPU time slice.* Other interrupts from any I/O devices needing service cause the system to execute the interrupt service routine in the driver for that device, and quickly resume the original process.

Switching between processes is the easy part. Each process has a process descriptor, holding information about it. When the 6809 is interrupted, the current address it is at in the program, and the CPU's registers are saved on the system stack in the process's data area. The stack pointer's value is saved in the current program's process descriptor for later retrieval.

The kernal then determines who gets the next time slice according to age and priority. The stack pointer of the new main process is loaded from its process descriptor, and since the stack pointer is now pointing to a 'snapshot' of its process's registers, a RTI instruction will cause the program to continue as if nothing had ever stopped it.

So, in essence, each process thinks that it is alone in the machine with its own program and data area limits defined, although if needed, it can find limited info on the others. Besides device interrupts and normal task-switching, two other events may have an effect on a program's running without its knowing about it: I/O queuing and untrapped signals.

* Actually 60 times/second on the CoCo, but a process time slice is considered to be 6 'ticks', or 1/10th second.

MULTI-TASKING PRINCIPLES
PROCESS QUEUES/STATES

PROCESS QUEUES

These are just what they sound like - an ordered arrangement of programs. They are kept in a linked list, that is, each has a pointer to the next in line. When a process changes queues, the process descriptor itself isn't moved, just the pointers are.

A process is always in one of three major queues (except for the current process):

Active - Normal running; gets its turn in varying amounts of the total processor time according to its age, priority, and state.

Sleeping - A program has put itself to Sleep for a specified tick count, or until it gets a signal. (As in waiting for its I/O turn)

Waiting - Special Sleep state that terminates on a signal or child's death / F$Exit. Entered via F$Wait.

STATES

The P$State byte in a process's descriptor has different bits set depending on what the program is doing, where it is currently executing, and what external occurences have affected it.

A process has one or more of these state attributes:

SysState	%1000 0000	Is using system resources, or is being started/aborted by the kernal.
TimSleep	%0100 0000	Asleep: awaiting signal, sleep over.
TimOut	%0010 0000	Has used up its time slice. This is a temporary flag used by the kernal.
Suspend	%0000 1000	Continues to age in active queue, but is passed over for execution. Used in place of Sleep and Signal calls in someL-II drivers.
Condem	%0000 0010	Has received a deadly signal, dies by a forced F$Exit call as soon as it is no longer in a system state.
Dead	%0000 0001	Is already unexecutable, as its data and program areas have been relinquished by an F$Exit call. The process descriptor is kept so that the death signal code may be passed to the parent on F$Wait.

The System State is a privileged mode, as the kernal doesn't make the process give up the next time slice, but instead lets it run continuously until it leaves the system state.

The reason for this is that the process is servicing an interrupt, changing the amount of free memory, or doing I/O to a device, and thus should be allowed to run until it is safe to change programs, or it has released the device for other use.

It is because of the System State that interrupts are allowed almost always. Any driver interrupt code acts as an "outside" program that temporarily takes over the CPU, but the current process is not changed and will continue when the driver is finished taking care of the interrupt source.

MULTI-TASKING PRINCIPLES
I/O

If two or more processes want to do input/output/status operations on the same device, all except the first will have to wait in line (queue). Under OS9, IOMan and the file managers are responsible for this control.

Each open path has a path descriptor associated with it. This is a 64-byte packet of information about the file. Because OS9 allows a path that has been opened to a file or device to be duplicated, and used by another process, several programs may be talking about the same path (and path descriptor). Provision must be made to queue an I/O attempt using the same path. (The most common instance of this is with /TERM.)

Since all I/O calls pass through the system module IOMAN, the I/O manager, it checks a path descriptor variable called PD.CPR to see if it is clear, or not in use. If it is in use, the process in inserted in a queue to await it's turn.

Here the process descriptor plays a part. Two of its pointers are used here: P$IOQP (previous link) , and P$IOQN (next link). P$IOQP is set to the ID of the process just ahead of this one, and the P$IOQN of the process ahead in line is set to this one's ID, forming a chain (linked list) of process ID pointers waiting to use this particular device.

When a process has made it through a manager to the point that the manager must do I/O through a device driver, it checks a flag in the driver's static (permanent) storage called V.BUSY. If it is clear, no one is using the device at that instant, and V.BUSY is set to the process's internal ID number.

If V.BUSY is not clear (another process got there first and is waiting for it's call to finish), the manager inserts the process in an I/O queue to wait its turn.

When the process (executing the file manager) is through with the device, it clears V.BUSY, and all the processes waiting in line are woken up to try again. As far as I know, V.Busy only becomes very important if a driver has put it's process to sleep, as otherwise the program would have exclusive access while within a system call anyway.

Thus a process seeking use of a device and its driver must wait FIRST for the path to be clear, and THEN for the device used by that particular path. If two processes are talking to two different files, or have each opened their own paths and the file is considered shareable, they will only have to wait in line for device use.

Again, it should be noted that once one process has started I/O operations, it has near-total use of the CPU time, except of course for interrupt routines or if it goes to sleep in the driver or a queue.

MULTI-TASKING PRINCIPLES

SIGNALS

Signals are communication flags, as the name implies. Since processes operate isolated from each other, signals provide an asynchronous method of inter-process flagging and control.

Commonly used signals include the Kill and the Wakeup codes. Wakeup is essential to let the next process in an I/O queue get its turn in line at a path or device.

OS9 has a signal-sending call, F$Send, which sends a one byte signal to the process ID specified, and causes the recipient to be inserted in the active process queue. Any signal other than Kill or Wake is put in the P$Signal byte of its process descriptor.

If it was the Kill signal, the P$State byte in the process descriptor has the Condemned bit set to alert the kernal to kill that process. A Wake signal clears the P$Signal byte, since just making the destination an active process was enough.

Signals are not otherwise acted upon until the destination process returns to the User state. (It'd be unwise to bury a process in the midst of using the floppy drives, for instance.) However, drivers and the kernal may take note of any pending signals and alter their behavior accordingly.

When the kernal brings a process to the active state, the P$Signal byte in the descriptor is checked for a non-zero value (Kill=0, but the Condemned bit was set instead, causing a rerouting to the F$Exit 'good-bye' call as soon as the killed process enters a non-system state). The process is given a chance to use the signal right off.

If the program has done a F$Icpt call to set a signal trap, a fake register stack is set up below the process's real one, holding the signal, data area and trap vector: P$Signal, P$SigDat, P$SigVec. The kernal then does its usual RTI to continue the program where it left off.

Instead, the program picks up at the signal vector where it usually stores the signal in the data area for later checking when convenient (totally up to the programmer, though). The trap routine is itself expected to end with a RTI, thus finally getting back to the normal flow of execution by pulling the real registers that are next on the stack.

If the program has NOT done a F$Icpt call, the kernal drop-kicks it into F$Exit, the same as a Kill signal does.

```
SIGNALS:

 0  S$Kill     Abort process (cannot be trapped)
 1  S$Wake     Insert process in Active process queue
 2  S$Abort    Keyboard abort (Break Key)
 3  S$Intrpt   Keyboard interrupt (Shift-Break)
 4  S$Window   Window has changed
 5-255         user defineable so far
```

INITIATING A PROCESS

P$-- process descriptor D.- Direct Page Variable

#	VAR	MOD	ACTION
1	P$ID	OS9	Allocates a 64-byte process descriptor.
	P$User		Copy parent's user index
	P$Prior		and priority.
	P$Age		Age set to zero.
	P$State		State of process is System State.
	D.Proc		Current process desc is now this one.
	P$DIO		Copies parent's default directory ptrs.
2	P$PATH	IOMAN	Called three times to I$Dup the first 3 paths of the parent (std in, out, error).
3	P$SWI	OS9	Make these 3 vectors = D.UsrSvc (0040).
	P$SWI2		
	P$SWI3		
	P$Signal		Clear process's signal, signal vector.
	P$SigVec		
4	P$PModul		F$Link to desired program module.
4a	P$PModul	IOMAN	F$Load from xdir if not in memory.
5		OS9	Error end if not Program/System module.
6	P$ADDR	OS9P2	F$Mem request to >= data area needed.
	P$PagCnt		
7		OS9	Copy parameters to top of new data area.
	P$SP		Set stack pointer to RTI stack registers.
			Set up RTI stack with register values:
			PC - module entry point
			U - start of data area
			Y - top of data area
			X - parameters pointer
			DP - start of data area
			D - length of parameters passed
			SP-> CC - interrupts okay, E flag for RTI
8	D.Proc		Put back parent as current process.
	P$CID		Get PARENT's other child, and
	P$SID		make it new proc's sibling link.
			(PARENT's new P$CID = new P$ID)
	P$PID		Copy parent's ID to new proc desc.
9	P$State		State of new is no longer System State.
			Return new child's ID to parent.
	P$Queue		F$AProc - insert process in active queue.

OPENING A FILE/DEVICE

```
=============================================================
   PD.- path descriptor vars        V$-- device table
   V.-- device static storage       Q$-- IRQ poll table
                                    P$-- process descriptor
-------------------------------------------------------------
Opening an OS9 device/file takes the following general steps:

 #  VAR      MOD    ACTION
 -  -----    -----  -----------------------------------------
 1  PD.PD    IOMAN  Allocates a 64-byte block path descriptor
    PD.MOD          Sets access mode desired.
    PD.CNT          Sets user cnt=1 for this path desc.

 2  PD.DEV   IOMAN  Attaches the device (drive) used.
    V$STAT          Allocates memory for device driver (CCDisk)
    V.PORT          Sets device address in driver static memory

 3  V.xxxx   DRVER  The driver's init subroutine is called to
    V.xxxx            initialize the device and static memory.
                    If device uses IRQ's, uses F$IRQ call:
4  Q$POLL   OS9    Sets up IRQ polling table entry.
     ...              ( address, flip & mask bytes, service add,
    Q$PRTY              static storage, priority of IRQ )

 5  V$DRIV   IOMAN  Sets up rest of device table.
    V$DESC            ( module addresses of desc, driver, mgr)
    V$FMGR
    V$USRS          Sets user count of device=1

 6  PD.OPT   IOMAN  Copies device desc info to path desc.
     ...              ( default values: drive #, step rate,
    PD.SAS              sides, baud rate, lines/page, etc. )
                    Calls file managr Open subroutine:

 7  PD.BUF   FLMGR  Allocates buffer for file use.
    PD.DVT          Copies device table entry for user.
    PD.FST-         Opens file for use, and sets up
    PD.xxx            file mgr pointers and variables.

 8  P$PATH   IOMAN  Puts path desc # in proc desc I/O table.
                    Returns table pointer to user as path nmbr.
_____________________________________________________________
2,3,4,5  only if first time for that device,
              else V$USRS = V$USRS + 1
                   PD.DEV = Device table entry
4        only if device uses IRQ's
```

GIME DAT

The memory management abilities of the CoCo-3 are the source of it's ability to run Level-II. To help explain what a DAT is, and it's usefulness, here's a text file I first posted on the OS9 Forum on 5 August 86.

Q: What is the difference between the 512K boards that are sold now and the 512K CoCo-3?

LOGICAL vs PHYSICAL ADDRESSES ---

To understand the difference, you must first keep in mind that the 6809, having 16 address lines, can only DIRECTLY access 64K of RAM. The only way for the CPU to use any extra memory is to externally change the address going to the RAM.

The address coming from the CPU itself is called the Logical Address. The converted address presented to the RAM is called the Physical Address.

For instance, the CPU could read a byte from $E003 in it's 64K Logical Address space, but external hardware could translate the $E003 into, say, a Physical Address of $1B003, by looking up the entry for the 4K block $E in a fast RAM table.

A coarser, but more familiar, example to CC owners is the $FFDF (64K RAM) 'poke'. The SAM chip can address 96K of Physical memory (64K RAM and 32K ROM). When that register was written to, the SAM translated all accesses to memory in the Logical (CPU) range of $8000-$FEFF to Physically point to the other 32K bank of RAM, instead of the ROM. A similar example is the use of the Page Bit register, to translate Logical accesses to $0000-$7FFF into using the other Physical 32K bank of RAM.

MEMORY MANAGEMENT ---

The hardware that does the actual translation between the Logical --> Physical addresses is called a Memory Management Unit (MMU). In the case above, the SAM was the MMU. One common type of hardware MMU is called a DAT, for Dynamic Address Translation. A DAT consists of a Task Register and some fast look-up RAM. It's called Dynamic partly because the translation table is not fixed, but can be modified. I'll go into more detail on a DAT later.

THE COCO-2 BOARDS ---

The memory expansions sold for the CC2 are an extremely simple form of a DAT. Most only allow the upper or lower 32K of Logical Addresses to access a different upper or lower 32K bank of Physical Memory. Leaving out I/O addresses and ROM for the moment, their 64K modes simplistically look like: (for 256K)

```
Logical (CPU) XXXX
 Address:
 $FFFF +-----+-----+-----+-----+
       I     I     I     I     I
       I U0  I U1  I U2  I U3  I   Upper 32K Banks
       I     I     I     I     I
       I     I     I     I     I   example: CPU access of
                         $0100
 $8000 +-----+-----+-----+-----+    using Bank 2 = L2+$0100
       I     I     I     I     I    is RAM address $20100.
       I     I     I     I     I
       I L0  I L1  I L2  I L3  I   Lower 32K Banks
       I     I     I     I     I
 $0000 +-----+-----+-----+-----+
      $0XXXX 1XXXX 2XXXX 3XXXX  Physical (RAM) Hex Address
```

The Physical memory that the CPU addressed is chosen from a combination of (L0 or L1 or L2 or L3) AND (U0 or U1 or U2 or U3). Some boards would mostly only allow the selection of Banks in number pairs (eg: L1+U1, L2+U2), or keeping L0 constant, and varying the Upper (U0-U3).

The important point here is that you could not 'mix & match' the Banks (Upper appear as Lower, Lower as Upper, or say, map U2 from $0000-$7FFF and U3 as $8000-$FFFF).

To use data from one bank to another generally required the copying of that data. This is why most applications of the extra memory were as RamDisks, or extra data storage, NOT as programs. (Tho you could have four different copies of the Color Basic ROMS for example, or four different OS9 '64K machines' running one at a time.)

THE COCO-3 DAT ---

To make the most economical use of the available RAM, and make the most use of reentrant (used by more than one process at a time) and postion-independent (runnable at any address, possibly using a different data area) programs or sections of data, the DAT has to be much more flexible than the Bank switching schemes above.

For instance, in the example given of four copies of the Basic ROMS, what if you had not modified the Extended Color ROM? You would have wasted 24K of RAM (3 banks x 8K) on extra copies. (Actually, you wasted 32K, since it'd be even better just to keep the original ROM 'in place'.) Or what if you really wanted one ROM copy and seven 32K RAM program spaces? Or you need to temporarily map in 32K of video RAM? Or keep seven different variations of the Disk ROM, which would all (at least on a CC2) need to made to appear at $C000 up?

And we haven't even discussed OS9 yet!

What have we figured out? We need both smaller translation 'blocks' and a way of making those physical blocks appear to the CPU at any logical block size boundary.

What size should a block be? So far, it seems that the smaller the better for a programmer or operating system, because that could leave more 'free blocks' left over for other use. This will become apparent later, in the Level-II discussion. Many Level-II machines use a 4K block. The CoCo-3 uses an 8K block size. In most cases, this may not be restrictive, except perhaps on a base 128K machine.

And so we come to the CoCo DAT. Here's a simple diagram:

```
+------+                 +-------+
 I     I                 I Task# I          VIDEO ADDR
 I CPU I A13-A15   R0-R2 I-------I               1 (19 addrs)
 I     I--------/------->I  DAT  I  P13-P18   +--1---+
 I     I    (3 addrs)    I  RAM  I------/---->I      I
 I     I                 I       I  (6 data)  I RAM  I   512K
 I     I                 +-------+            I ADD  I-->RAM
 I     I     A0-A12                P0-P12     I MUX  I
 I     I-------/------------------------/---->I      I
 +-----+                (13 addrs)            +------+

                        /...........GIME............../
```

As shown, the DAT RAM would be 8 six-bit words x 2 tasks (explained below).

From left to right, the Logical Addresses from the CPU are translated into a extended Physical Address to access the RAM.

The upper 3 CPU lines (A13-A15) are used to tell the DAT which 8K Logical Block is being used (1 of 8 in a 64K map) and act as DAT RAM address (R0-R2) lines. At that Logical Block address in the DAT is a 6-bit data word, which forms the extended Physical Address lines P13-P18. The lower CPU address lines are passed thru as is to point within the 8K RAM block (out of the 512K RAM) selected by P13-P18.

Note that 6 bits can form 64 block select words. Multiply 64 possible blocks by 8K per block, and there's your 512K RAM. You may write any 6-bit value to each of the 8 DAT RAM locations, thus choosing which of the 64 8K-blocks you wish to appear within the 8K address block the CPU wishes to access. You could even write the same value several times, making the same 8K physical RAM show up at different logical CPU addresses.

The Task number acts as the DAT R3 address line, and simply allows selection between 2 sets of eight DAT RAM words. This makes it simpler to change between 64K maps. Normally, you can software select the Task number.

AN ANALOGY ---

Okay, this has been rough on some of you, and my explanation may need some explaining <grin> so a simpler analogy is in order:

Let's say you have a fancy new TV cabinet with 8 sets from bottom to top in it. You can watch all 8 at a time. (This makes you the CPU, and each screen is 8K of your logical 64K address space.)

Ah, but each set also has 64 channels! So you can tune each set to ANY of the channels, or several to the SAME channel. (Each channel is like one 8K block out of the 64 available to you in a 512K machine.) When you tune in a program, you are said to have "mapped it in".

An analogy to the Task Register would be if each set had TWO channel selectors A and B, and you had one switch to select whether ALL the sets used their A or B setting. This is generally called "task switching". If you wanted to switch to a C,D, or E task, you'd have to get up and retune all 8 sets on their A or B selectors (all A or all B), possibly from a list (called a "DAT Image") you had made from TV Guide.

Get it now? The CC2 512K expansions would then be like the same cabinet, only the top or bottom four sets always tune together and only have 8 selector positions; the same eight channels per same position. Which would you buy?

NOW I HAVE IT! -- BUT WHAT USE IS ALL THIS?

So far, we've seen that the 64- 8K blocks can be arranged any which way that you'd like to see them, 8 at a time. As a quick example of what could be done, let's see how a text editor might work. We'll assume the upper 32K is RSDOS always, and not to be touched, to keep this simple.

This leaves us with 32K, or four 8K blocks for our program and data (the text). In our example, we'll make the editor code itself just under 24K long, which leaves us only 8K for text. So, here's the map:

```
E000-FFFF logical block 7     hires cmds & I/O
C000-DFFF               6     disk basic
A000-BFFF               5     color basic
8000-9FFF               4     extended basic
6000-7FFF               3     editor
4000-5FFF               2     editor
2000-3FFF               1     editor
0000-1FFF               0     text
```

(Note that this is kind of unrealistic, since you'd probably not want to have the text down in RSDOS variable territory, but this is just an extremely simple example, okay?)

Okay, you type in 8K of text. Normally, that'd be all you could do, but remember that we can make any Physical 8K Block map into any Logical 8K Block. So the editor, when it realizes that it's buffer is almost full, could tell the GIME MMU to make a different RAM block (out of the 64, minus those used by Basic for text, etc) appear to the CPU in our logical block 0 (from $0000-$1FFF).

Even if Basic uses up 8 actual RAM blocks for it's own use, and the editor uses 3, we still could use (64-11) or 53x8K blocks. That's over 400K of text space! By swapping real (physical) RAM into our 64K (logical) map like this, the only limitation on spreadsheets, editors, etc, is that the programmer must respect the 8K block boundaries.

Hmmm... you say. I could even swap in different editor programs, if I had to, couldn't I? You bet. Now you're starting to get an inkling of how Microware did Level-II.

OK, WHAT ABOUT OS9 LEVEL-II?

L2 gives each process up to 64K to work with. It allocates blocks of memory (you got it - up to eight 8K blocks!) for that process to use as program or data areas.

Having 512K of memory does NOT mean you could do a "basic09 #200k" command line. The CPU can still only access 64K at a time, but the space not used by Basic09 (which itself is about 24K long) is usable for data. So about 64K minus 24K is about 40K, which is very big for a Basic09 program.

Notice a gotcha here, though. If Basic09 was 25K long, then you'd have much less data area possible. Why? Remember the 8K blocks! A 25K program would map in using four 8K blocks (three wouldn't be enough), using up 32K of your 64K map. The same goes if you asked for 9K of data space. You'd get two 8K blocks of RAM mapped in, taking up 16K of CPU space. Aha! Now you understand why the smaller the block size the better.

Back to the good parts. Remember that most OS9 programs are reentrant and position-independent. This means that no matter how many processes or terminal-users want to use a certain program, only ONE copy needs to be in memory. (Check the difference: if you had 10 Basic09 programs running, each needing 30K of data space - they'd need only 24K for B09 + 10*30K, versus 10*(24K+30K), a 216K savings!) The Amiga's programs, for example, aren't reentrant. It'd need 540K.

As far as making 200K virtual programs, there ARE ways of doing that. You could start other processes (Forking), or map in different data modules. Even better, you can pre-Load modules, and by Linking and Unlinking them, they will swap in and out of your 64K address space, a technique much faster than using RamDisks. (A Loaded module is off in RAM somewhere, but not in your map until Linked to.) This is what Basic09 does, by the way, so by writing a program that calls lots of small subprograms, each would get swapped in automatically as you needed them. Instant 400K basic!

TOO MUCH TO SAY ---

Well, there's about a zillion other things I wanted to put in here, like how the page at $FE00-$FEFF is across all maps, to make moving data easier (some move code is there); or how each Level-II process or block of programs has a DAT Image associated with it, that can be swapped into the DAT RAM; or that up to 64K is allocated to the System Task, where the Kernal and Drivers and buffers are; or the neat tricks you could do using the DAT; or show you a possible memory map using the DAT; or about how interrupts switch to the System Task.

(Some of this IS covered in this new collection - Kevin)

DAT IMAGES and TASKS

It may seem that we're spending a lot of space on the DAT, but it's very important to the whole of L-II. So...

As you now know, the DAT in the CoCo-3 allows you to specify which of up to eight blocks will appear in the 6809's logical address map when their numbers are stored and enabled in the GIME's MMU or DAT.

Ideally, an MMU would have enough ram to handle the maps for any conceivable number of programs, modules or movement. But ram that fast is expensive and uses lots of power. So a compromise was made -- in the GIME's case, two sets of DAT registers. That is, two complete 64K maps can be stored and switched between at will.

You will surely need one map for the system plus another for a shell at least. So how does OS9 handle the needs of all the other programs you want to run? By swapping sets of block numbers into the DAT as needed.

The set of block numbers is stored in a packet of information called a DAT Image. Because various OS9 machines use different size blocks (2K, 4K, 8K, are most frequent) and have differing amounts of memory blocks available, a DAT Image can vary in size even though a process descriptor has 64 bytes available for one.

On the CoCo-3, it's 16 bytes long, made up of 8 two-byte entries. The first byte of each entry is usually zero, while the second byte is the physical block number. The exception is when an entry contains a special value of $333E, which is used to indicate that that logical block is unused as memory for that map.

When expanding the amount of blocks allocated to a map, OS9 checks for the special $333E flag bytes. That's how it knows where to place new blocks in the DAT Image.

DAT Images are created for several purposes. The one that affects you the most is the image stored in a process descriptor. Whenever a process comes up in the queue for running, it's DAT image is copied to one of the two sets of GIME task map registers. Then that set is enabled by setting the task register select. Instantly the new logical map is the one seen by the CPU. When a process' timeslice is up, it also gives up the use of the task number.

The task register number used for the process DAT image is usually the same number stored in the P$Task byte in other L-II computers. On the CoCo-3 however, P$Task contains the number of a virtual or fake DAT task map. There are 32 of these, which make it appear as though the GIME had 32 sets of map registers.

If the images are already in the process descriptors, why have virtual tasks? Because it's simpler for the system to look them up in a known table versus searching all over.

The first two virtual DAT tasks (0 and 1) are reserved for the system's use. The first is for the usual kernel, drivers, descriptors, buffers. The second is for GrfDrv's screen and buffer access.

So on the CoCo-3, the task number refers to a table entry that points to the DAT Image to be used. Except for special cases, the pointer is to the image within a process descriptor.

Another use for the images is in the module directory. Unlike Level One, where the entry could simply contain the module's address within the 64K you had, Level Two entries point to a DAT Image of the block or blocks containing the module and any others loaded with it.

While a module file is being loaded, OS9 temporarily allocates a process descriptor and a task number for it. The file is then read into blocks of memory that F$Load has requested. Then the descriptor & task are released, leaving the modules in a kind of "no-man's-land", waiting to be mapped into a program's space.

The visible residue of loading a file of modules is that the free memory count goes down, and any new modules found are entered into the system map's module directory. Otherwise, they don't directly affect a process map until linked into it.

Each Module Directory entry is made up of:

00-01 MD$MPDAT -	Module DAT Image Pointer
02-03 MD$MBSiz -	Block size total
04-05 MD$MPtr -	Module offset within Image
06-07 MD$Link -	Module link count

A program such as Mdir can use these to display what it does about the modules in memory. First, it gets the module directory using F$GModDr. Then by using the DATImage and offset associated with an entry, Mdir F$Move's the header and name from the blocks where the module has been loaded.

The Mdir example illustrates a third common usage of images, moving data into your program's map for inspection.

Anytime you need to "see" memory external to your process (sorry, you can only legally read it; no writes), you can create a DAT image of your own and use it with F$Move. OS9 will take the offset and amount you pass, and copy that amount over to your map from the offset within the image you made.

In the case of Mdir, the image was moved over by F$GModDr along with the module directory entries. So there's no need to build an image in that case. Just use the MD$MPDAT pointer.

You may also in some cases request movement of data between maps using a reference to a Task number instead. OS9 itself will internally index off the tasks' images for you.

Notice that throughout this section, the image is used over and over simply to allow the cpu to read or write to extended memory.

In the next section, we'll see some examples of DAT Images and maps.

LEVEL TWO IN MORE DETAIL

I will be using "L-II" for Level Two, and "One" for Level One, so as to make differentiating the names a little easier as you read. Other word definitions I use here are (loosely):

space - any 6809 logical 64K address area.
mapping, mapped in -causing blocks to appear in a space.
a map - a space containing mapped-in modules/RAM blocks.
system map - the 64K map containing the system code.
task - a particular map with a certain program and data area
task number - number of a particular task map.
DAT map - a task ready to use thru the hardware/software
enable of the task number's map.
task register - task number stored here to enable a DAT map.

user code - the programs/data you use (applications).
system code - the programs/data the system uses (file mgrs,
drivers, descriptors, and the kernal F$ & I$Calls,
IRQ handlers, and scheduling codes).

LEVEL TWO vs ONE: General

The core of understanding L-II is in understanding the separation and handling of 8K blocks, and their use in logical 64K spaces. And why.

DAT -

Under One, you only had 64K of contiguous physical RAM in one 64K logical map. L-II uses the DAT to map any physical 8K blocks of RAM containing program and data modules into a 64K logical address map. When a program's turn to run comes up, the block map data (called a DAT Image) for it's 64K space is copied to and/or enabled in the GIME's DAT.

SWI's -

L-II was designed to run most programs written for One, which is possible since system calls are made using a software interrupt call, passing parameters (via cpu registers pushed on a memory stack) that are pointed to by the 6809's SP register. This gives two advantages over Level One:

1: Virtually none of the system code has to reside in the 64K space containing the user's program and data areas. The system map is switched in place of the caller's map.

2: OS/9 needs only to know the caller's SP and task number (both kept in the caller's process descriptor in the system map) to access the parameters passed, or to move data between the two maps.

(Note that a kernal could be written to do simply this on any CoCo that had the Banker or DSL Ram expansion, etc. But you'd lose the advantage of the smaller flexibly-mapped blocks provided by the GIME's DAT.

The corollary advantage, and the "why" of L-II, is that each user program can have almost an entire 64K space to itself and it's data area, as can also the system code.

THE SYSTEM TASK MAP:

Up to 63.75K of kernel, bootfile (drivers, mgrs, etc).
I/O buffers.
Descriptors.
System vars & tables.

System calls and other interrupts temporarily "flip" the program flow into this task map. User parameters and R/W data copied from/to system ram for drivers and file managers to act upon.

EACH USER TASK MAP:

Up to 63.5K total for each program and it's pgmdata area. Each task map made out of up to 8 module or pgmdata blocks (8K each) that are mapped in from the 64 (minus those used by the system task or other user tasks) blocks available in a 512K machine.

THE SYSTEM MAP

Oddly enough, the system map is close to what you're used to under Level One. Memory is allocated for buffers and descriptors in pages just as before. The main difference is that no user programs (should) share space here, as they did under Level One.

You still have the Direct Page variables from $0000-00FF along with other system global memory just above it up to $1FFF Towards the top (????-FEFF) we run into descriptors, buffers, polling tables, and finally the I/O modules and the kernal. A CoCo-III Level Two System Map looks like this:

```
0000-0FFF  Normal L-II System Variables
1000-1FFF  New CC3 global mem and CC3IO tables
2000-xxxx  free ram
xxxx-DFFF  Buffers, proc descs, bootfile
E000-FDFF  REL, Boot, OS9
FE00-FEFF  Vector page (top of OS9p1)
FF00-FFFF  I/O and GIME registers
```

Some areas of special interest include the ...

Vector Page RAM:

This page of RAM is mapped across ALL 64K maps. This "map-global" RAM is necessary so that no matter what other blocks are mapped in place of the system code, there is always a place for interrupts (hardware or software) to go and execute the special code in OS9p1 that switches over to the system task.

THE SYSTEM TASK MAP:

Up to 63.75K of kernel, bootfile (drivers, mgrs, etc).
I/O buffers.
Descriptors.
System vars & tables.

System calls and other interrupts temporarily "flip" the program flow into this task map. User parameters and R/W data copied from/to system ram for drivers and file managers to act upon.

EACH USER TASK MAP:

Up to 63.5K total for each program and it's pgmdata area. Each task map made out of up to 8 module or pgmdata blocks (8K each) that are mapped in from the 64 (minus those used by the system task or other user tasks) blocks available in a 512K machine.

THE SYSTEM MAP

Oddly enough, the system map is close to what you're used to under Level One. Memory is allocated for buffers and descriptors in pages just as before. The main difference is that no user programs (should) share space here, as they did under Level One.

You still have the Direct Page variables from $0000-00FF along with other system global memory just above it up to $1FFF Towards the top (????-FEFF) we run into descriptors, buffers, polling tables, and finally the I/O modules and the kernal. A CoCo-III Level Two System Map looks like this:

```
0000-0FFF  Normal L-II System Variables
1000-1FFF  New CC3 global mem and CC3IO tables
2000-xxxx  free ram
xxxx-DFFF  Buffers, proc descs, bootfile
E000-FDFF  REL, Boot, OS9
FE00-FEFF  Vector page (top of OS9p1)
FF00-FFFF  I/O and GIME registers
```

Some areas of special interest include the ...

Vector Page RAM:

This page of RAM is mapped across ALL 64K maps. This "map-global" RAM is necessary so that no matter what other blocks are mapped in place of the system code, there is always a place for interrupts (hardware or software) to go and execute the special code in OS9p1 that switches over to the system task.

BlockMap:

In a 512K CoCo OS/9 has 64 RAM blocks of 8K each to choose from (8K x 64 = 512K). Each is known by a number from 00-3F. The blockmap is a table of flags indicating the current status of each of these blocks, which could be ...

FREE RAM = Ram blocks not in use as Module/ PgmData areas.

RAM IN USE = Ram blocks in use for either:

Modules - Blocks that contain program, subroutine, or data modules. MDIR will show these. Before a module is used, it will have been loaded into free ram blocks. On link or run, those blocks are then mapped into (made to appear in) any task's space. A data module mapped into several maps can provide inter-task vars. Subroutine mods (like for RUNB) can be linked/unlinked, in/out of a task map.

Data - Free ram that has been mapped into a task space for use as pgm data areas. Normally these blocks are only mapped into one task space (unlike module blocks). These blocks will be released to the free RAM pool when the program using them exits.

DAT Images:

Since each task map requires knowing which (of up to 8) blocks are to be mapped in for that process (yes- system code execution is also a process), AND since OS/9 must know in which blocks that program modules have been loaded into, OS/9 keeps individual tables or "images" of those block numbers.

Each Image has 8 slots, two bytes each. A special block number, $333E, is used to designate an unused logical block for that task.

Module Directory:

In Level One, the module directory simply had to point to the module's address. Under L-II, it points to the DAT Image table showing the block(s) the module is physically in and it's beginning offset within the DAT Image logical 64K map.

Process Descriptors:

A descriptor contains pretty much the same info as it did under L-One, but adds the DAT Image for that process, which will be set into the DAT when it's turn to run comes up.

There is also a local process stack area, used while in the system state (executing system code after a system call). This is because the process's real stack is of course in another map, and a local stack is needed if the process were interrupted or went to sleep.

SYSTEM MEMORY ALLOCATION

As I said above, the system map is still allocated internally in pages. However, when you first boot up, it usually will only have about 5 blocks mapped in. Something like:

```
Logical    Physical
Address    Block(s)
---------  --------
0000-1FFF  00        - block 00 is always here
2000-7FFF            - no ram needed here yet
8000-DFFF  01,02,03  - this is your bootfile, first vars
E000-FEFF  3F        - block 3F always contains the kernal
```

The system process descriptor of course has the DAT Image that corresponds to this block map.

Any RAM left over in blocks allocated for loading the bootfile is taken by page for system use. For instance, the device table normally is just below the bottom of the boot.

Once you begin running several processes and opening files, the system must allocate more RAM for descriptors and buffers. When all the pages that are free in the blocks already mapped in are used up, OS9 maps in another block, which is then also sub-allocated by page.

Page allocation is still used because buffers, descriptors and tables usually are a page or two size, just as under Level One. So it's still the best use of available memory.

USER MAPS

MODULE and DATA AREAS

Each user process has the use of a map made up of up to eight 8K blocks. However, it is seldom that all eight are in use (certain basic09 and graphics programs excepted).

More likely, each task map will look like:

```
Logical    Physical
Address    Block(s)
---------  --------
0000-1FFF  ??        - 8K data area
2000-DFFF            - no ram needed here yet
E000-FEFF  ??        - block containing program
```

Again, the process descriptor DAT Image has a copy of the block numbers actually used (instead of ??).

Unlike Level One, RAM for a user process is NOT allocated by page. There's no need to, for two reasons. First, the data area is not shared with any other process.

Second, no memory can be used from any left over in the program block. Many people ask why not? Hey, they say, since you can map a block anywhere, why can't some other program take advantage of the unused RAM? The answer is basically that it would just take too many resources to keep track of what module should stay because part of the block was being used for data.

Even more importantly, what if a program requested more memory while it was running? You'd be stuck, as data areas must be contiguous and any modules within that block would be in the way. One more reason: Level Two was designed to take advantage of modules in ROM. So there's no way to assume that RAM is available in that block.

So, the upshot is that data areas are allocated from any free RAM blocks in the machine, and always 8K at a time. Even if your program only needed two pages to run in, it still gets a block. Now you can see that the smaller the block the better, as in this case having 4K blocks would leave more free RAM for other programs to use.

Just like in Level One, programs end up at the highest logical address possible in a map, and data areas at the bottom. For the same reason as in One, this is done to allow the data area to grow as much as possible if needed.

One very important point to make at this time: since all modules that were loaded together are also mapped into spaces together, it pays to keep module files close to an 8K boundary. More details on this are in the MISC TIPS section at the end of the book

SWITCHING BETWEEN MAPS

Okay, now we come to the nitty-gritty of Level-Two. This is where we tie together all we've talked about so far. But it's not tough, so don't worry.

Let's say that a program is running in it's own map, and wishes to use a system call for I/O. How does the code get over to the system map where the drivers are?

An OS9 system call is simply a software interrupt. What that means is that what the program is doing and where it's at is saved in the process' memory on a stack of variables.

Then, like all interrupts, program flow is redirected (by reading the CoCo's BASIC ROM, specially mapped in just long enough to get the addresses) to the vector page at logical address FE00 which is at the top of all maps.

The code within that page is part of OS9p1 and it knows that it should change the GIME task register select to task 0, which is always the system map. As soon as it does that, all the kernal, file managers, drivers etc are accessible to the CPU, which will come down out of the vector page to complete your system call. If needed, OS9 will go back to code located in the vector page where it can map in your user task long enough to get and put data.

At the end of the call, the system code jumps back up into the vector page, maps your process' DAT Image back into the GIME's task map 1, then enables task register 1 which allows your program space to reappear to the CPU.

Then the saved registers are taken back off the stack in your map, and your program continues.

If you want to, you can think of Level Two as really giving your program 128K of RAM, as the net effect compared to Level One is just that... under One, your program had to share space with the drivers and kernal, and any system calls stayed within the same old 64K map. Under Two, your program jumps between 64K maps when you make a system call.

One side note: because of the manipulation of the GIME's MMU and the necessity of copying much data between maps, L-II is normally slower than Level One. However, the CoCo-3 makes up for this as it runs at twice the speed of our older CoCo's.

EXAMPLE MAPS

Here are some example process, module and memory maps generated by the programs I've included in the back of this book. Study them and you can see the relationship between what is reported by each utility. They should help give you a better feel as to what's going on in your machine.

EXAMPLE ONE:

I had two shells running, and of course the particular utility that was printing out at the time.

```
 ID Prnt User Pty Age  St Sig ..  Module    Std in/out
--- --- ----- --- ---  -- --- --  --------  -----------
  2   1    0  128 129  80   0 00  Shell     <TERM >TERM
  3   2    0  128 129  80   0 00  Shell     <W7    >W7
  4   3    0  128 128  80   0 00  Proc      <W7    >D1
```

Below's my PMAP output. The numbers across the top (01 23 etc) are short forms of (0000-1FFF, 2000-3FFF) addresses in each task's logical map. Notice that there are indeed eight 8K block places in each map, but only those blocks that are needed are mapped in (and are in the DAT Image of that process, which by the way, is where the map information is gotten by PMAP).

```
 ID      01 23 45 67 89 AB CD EF  Program
----     -- -- -- -- -- -- -- --  -----------
  1      00 .. .. 04 01 02 03 3F  SYSTEM
  2      05 .. .. .. .. .. 06 ..  Shell
  3      07 .. .. .. .. .. 06 ..  Shell
  4      0A .. .. .. .. .. .. 08  PMap
```

Now, notice that in the SYSTEM map is Block 00 = system global variables, Block 3F = kernal, Blocks 01,02,03 = bootfile, and Block 04 plus probably part of 01, = system data and tables.

In the shell and pmap lines, we see that Blocks 05,07,0A are being used for data. Block 06 must contain the Shell, and Block 08 must contain Pmap. We can confirm all this by looking at the module directory output below and comparing block numbers:

```
 Module Directory at 00:03:51
Blk Ofst Size Ty Rv At Uc  Name
--- ---- ---- -- -- -- --  ------
 3F  D06  12A C1  1 r  00 REL     - the kernal
 3F  E30  1D0 C1  1 r  01 Boot
 3F 1000  ED9 C0  8 r  00 OS9p1
 01  300  CAE C0  2 r  01 OS9p2   - boot modules
 01  FAE   2E C0  1 r  01 Init
 .. ....  ... ..  . .  .. ....
 01 6947  1EE C1  1 r  02 Clock
 01 6B35  1AE 11  1 .  01 CC3Go
 06    0  5FC 11  1 r  03 Shell   - the Shell file
 06  5FC  2E7 11  1 r  00 Copy
 .. ....  ... ..  . .  .. ....
 06 1E10   2D 11  1 r  00 Unlink
 08    0  28E 11  1 r  01 Proc    - my cmds file
 08  435  1B1 11  1 r  00 MMap
 08  5E6  1F8 11  1 r  00 PMap
 08  7DE  1D5 11  1 r  00 SMap
 08  9B3  136 11  1 r  00 DMem
 08  AE9  240 11  1 r  00 Dump
 09    0 1FFC C1  1 r  01 GrfDrv - grfdrv is alone
```

Using my MMAP command, we can see below how many blocks are left for the OS9 system to use. Take notice of the block 3E being allocated... that's the video display ram block.

RAM for video is allocated from higher numbered blocks, since there is a better chance of finding contiguous RAM that way. Normally, blocks don't have to be together for OS9 to use them, but the GIME requires that screen memory be that way for display.

```
    0 1 2 3 4 5 6 7 8 9 A B C D E F
#   = = = = = = = = = = = = = = = =
0   U U U U U U M U M M U _ _ _ _ _  <--blocks 00-0A
1   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
2   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
3   _ _ _ _ _ _ _ _ _ _ _ _ _ _ U U  <--3E, 3F
```

Number of Free Blocks: 51
Ram Free in KBytes: 408

EXAMPLE TWO

This real example I ran off the other day. I had five shells, all of which had started another process (by me typing it in).

```
 ID Prnt User Pty Age  St Sig ..  Module   Std in/out
--- ---- ---- --- ---  -- --- --  -------- -----------
  2   1    0  128 129  80   0 00  Shell    <TERM >TERM
  3   2    0  128 130  80   0 00  Shell    <W7   >W7
  4   3    0  128 129  80   0 00  Shell    <W4   >W4
  5   4    0  128 129  80   0 00  pix      <W4   >W4
  6   2    0  128 129  80   0 00  pix      <TERM >TERM
  7   3    0  128 129  80   0 00  Shell    <W5   >W5
  8   7    0  128 128  80   0 00  pix      <W5   >W5
  9   3    0  128 129  80   0 00  Shell    <W6   >W6
 10   3    0  128 128  80   0 00  Proc     <W7   >D1
 11   9    0  128 129  C0   0 00  Ball     <W6   >W6
```

Note the high block numbers in most of the programs. Each window was showing an Atari ST picture in it, and process #11 had Steve Bjork's bouncing ball demo running.

True windows that use GrfInt and Grfdrv are NOT mapped into a program's space. But this was special, as I was running many VDGInt screens, which usually ARE mapped in (on purpose) so that the programs could directly access the video display.

Notice also that my System task had fully been allocated by block. The SMAP later shows what part of them was free.

```
ID      01 23 45 67 89 AB CD EF   Program
----    -- -- -- -- -- -- -- --   -----------
  1     00 31 11 04 01 02 03 3F   SYSTEM
  2     05 .. .. .. .. .. 06 ..   Shell      -see note below
  3     07 .. .. .. .. .. 06 ..   Shell
  4     09 .. .. .. .. .. 06 ..   Shell
  5     0E .. .. 3A 3B 3C 3D 0D   pix
  6     0F .. .. 36 37 38 39 0D   pix
  7     10 .. .. .. .. .. 06 ..   Shell
  8     12 .. .. 32 33 34 35 0D   pix
  9     13 .. .. .. .. .. 06 ..   Shell
 10     18 .. .. .. .. .. .. 19   PMap
 11     14 16 17 .. .. .. 31 15   Ball
```

The other point to note is that the Tandy-provided shell file (block 06) goes over the block size-512 byte limit, and thus cannot be mapped into the top block slot, because it would fall on top of the vector page and I/O area from FE00-FFFF.

Here's the MMAP output. Lots of video ram allocated, huh?

```
    0 1 2 3 4 5 6 7 8 9 A B C D E F
#   = = = = = = = = = = = = = = = =
0   U U U U U U M U M U M M M M U U
1   U U U U U M U U U M _ _ _ _ _ _
2   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
3   _ U U U U U U U U U U U U U U U
```

Number of Free Blocks: 23
Ram Free in KBytes: 184

And just to show how close I was to a real limit, here's the SMAP utility output. It shows in pages how much memory is left in the system task map. The 32x16 old-style VDG text screens and all the process descriptors (two pages each!), plus a page for each window's SCF input buffer made things rather tight.

```
    0 1 2 3 4 5 6 7 8 9 A B C D E F
#   = = = = = = = = = = = = = = = =
0   U U U U U U U U U U U U U U U U
1   U U U U U U U U U U U U U U U U
2   U U U U U U U U U U U U U U U U
3   U U U U U U U U U U U U U U U U
4   _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
5   _ _ U U U U _ U U U U U U U U U
6   U U U U U U U U U U U U U U U U
7   U U U U U U U U U U U U U U U U
8   U U U U U U U U U U U U U U U U
9   U U U U U U U U U U U U U U U U
A   U U U U U U U U U U U U U U U U
B   U U U U U U U U U U U U U U U U
C   U U U U U U U U U U U U U U U U
D   U U U U U U U U U U U U U U U U
E   U U U U U U U U U U U U U U U U
F   U U U U U U U U U U U U U U U .
```

Number of Free Pages: 19
Ram Free in KBytes: 4

INSIDE OS9 LEVEL II

The System

L-II PROCESS DESCRIPTOR VARIABLES

```
00      P$ID      Process   ID
01      P$PID     Parent's  ID
02      P$SID     Sibling's ID
03      P$CID     Child's   ID
                    The family proc id numbers.
04-05   P$SP      Stack Pointer storage
                    SP position within Process map
06      P$Task    Task Number
                    Virtual DAT task number
07      P$PagCnt  Data Memory Page Count
08-09   P$User    User Index
0A      P$Prior   Priority
0B      P$Age     Age
                    The age always begins at Priority.
0C      P$State   Status
                    System, Image Changed, Dead, etc.
0D-0E   P$Queue   Queue Link (next process desc ptr)
                    For active, waiting, sleeping procs.
0F      P$IOQP    Previous I/O Queue Link (Proc ID)
10      P$IOQN    Next     I/O Queue Link (Proc ID)
                    Path or driver queues.
11-12   P$PModul  Primary Module pointer
                    Offset within proc map to program.
13-14   P$SWI     SWI  Entry Point
15-16   P$SWI2    SWI2 Entry Point
17-18   P$SWI3    SWI3 Entry Point
                    May be changed to point to proc map.
19      P$Signal  Signal Code
1A-1B   P$SigVec  Signal Intercept Vector
1C-1D   P$SigDat  Signal Intercept Data Address (U)
                    Signal storage and user-defined vector.
1E      P$DeadLk  Dominant proc ID for locked I/O
20-2F   P$DIO     Default I/O ptrs (chd, chx)
                    Drive table and LSN entries.
30-3F   P$Path    I/O Path Table (real path numbers)
                    User path numbers 0-F index here to the
                    actual path descriptor number involved.
40-7F   P$DATImg  DAT Image (only 16 used in CoCo-3)
                    The block map of this 64K process space.
80-9F   P$Links   Block Link counts (for user map)(8 used)
                    To keep track of map-internal links.
A0-AB             Network variables?
AC                Path number (0,1,2) for selected window

         rmb $200-. Local stack
P$Stack  equ 512 Top of Stack
P$Size   equ 512 Size of Process Descriptor
```

There are three main differences between a L-I and Level Two process descriptor. The L-II additions are:

. DAT Image - so OS9 knows what to map in for the process.
. Link Cnts - so an unlink won't unmap blocks with other still-linked-into-this-map modules.
. Stack area- used while in the system state.

The link counts apply to that process map only, and are counts of block links, not individual modules. Say you had a merged module file loaded with Runb, Syscall and Inkey all together taking up two blocks. The first logical block number of the whole group will have a link count of one.

Then perhaps your program calls Inkey. Inkey is found in your map already, and the first block number link count is incremented in the process descriptor. The module directory link count is incremented also.

Now Inkey finishes and is unlinked. The link count is decremented in the module directory and could easily now be zero. But you don't want Runb and Syscall to go away, too! And they won't because the process map block link now only goes down to one again, and so both blocks mapped will stay mapped.

The stack area is needed when an interrupt (software or hardware) occurs. The initial register save will be within the process' stack area. Then OS9 flips over to the system map, where, in case this process' time is up and it's whole state must be saved, OS9 begins using the process descriptor stack area instead.

In a way, the process descriptor stack is an extension of the process data area into the system map.

Under L-I, of course, there was no need for this, as everyone's stack was available at all times.

L-II Direct Page Variable Map $00XX

* Names are standard L-II. Defs with no name are new CC3 vars.

```
Addrs  Name      Use
-----  --------  ----------------------------
20-21  D.Tasks   Task Proc User Table
                  Points to 32 byte task# map.
22-23  D.TmpDAT  Temporary DAT Image stack
                  Used to point to images used in moves.
24-25  D.Init    INIT Module  ptr
                  Points to the Init module.
26-27  D.Poll    Interrupt Polling Routine
                  Vector to IOMan sub to find IRQ sources.
28     D.Time    System Time Variables:
28     D.Year      Year
29     D.Month     Month
2A     D.Day       Day
2B     D.Hour      Hour
2C     D.Min       Minute
2D     D.Sec       Seconds
```

2E	D.Tick	Tick countdown for slice 60 Hz IRQ count. (60 ticks = 1 second)
2F	D.Slice	Current slice remaining Ticks left for current process normal run.
30	D.TSlice	Ticks per Slice constant Set to 6 = 1/10 second per process slice
32	D.MotOn	Drive Motor time out
36-37		Boot start address
38-39		Boot length New variables for use by os9gen & cobbler.
40-41	D.BlkMap	Memory Block Map Points to 64 byte physical block flag array.
44-45	D.ModDir	Module Directory Points to the 8 byte dir entries start.
48-49	D.PrcDBT	Process Descriptor Block Table Points to 256 byte array of msb addresses.
4A-4B	D.SysPrc	System Process Descriptor Points to proc desc used while in SysState.
4C-4D	D.SysDAT	System DAT Image Points to the image within D.SysPrc desc.
4E-4F	D.SysMem	System Mem Map Points to 256 byte page table for systm map.
50-51	D.Proc	Current Process Desc Points to the proc desc in use now.
52-53	D.AProcQ	Active Process Queue First proc desc link of procs ready to run.
54-55	D.WProcQ	Waiting Process Queue First proc desc link of procs that F$Wait'd.
56-57	D.SProcQ	Sleeping Process Queue First proc desc link of procs sleeping.
58-59	D.ModEnd	Module Directory end
5A-5B	D.ModDAT	Module Directory DAT image end
6B-6C		"Boot Failed" REL vector Vector to display of this message.
71-7C		CoCo reset code 55 NOP NOP B7 FF DF 7E F00E
80-81	D.DevTbl	I/O Device Table Points to array of 9-byte device entries.
82-83	D.PolTbl	I/O Polling Table Points to array of 9-byte IRQ poll entries.
88-89	D.PthDBT	Path Descriptor Block Table ptr Points to base 256-byte path descs table.
8A	D.DMAReq	DMA Request flag (MPI slot use) Set= MPI slot has been changed. CC3Disk flag.
90		GIME register copies:
91		Init Reg $FF91 shadow for tasks
92		IRQEN $FF92 shadow IRQ enables
93-9F		other GIME shadows
A0		Speed flag (1=2Mhz)
A1-A2		Task DAT Image Ptrs Table ptr Pointer to 32 image pointers for task #'s.

A3		0=128K, 1=512K temp flag
A4		FF91 Task Reg Bit (which system state task)
A5-A6		Global CC3IO memory Pointer to $1000: global mem.
A7-A8		Grfdrv SP storage Pointer to end of global mem. sysmap 1 stack.
A9-AA		Grfdrv ->kernal return vector
AB-AC		Kernal ->grfdrv second sysmap
AD-AE		Clock SvcIRQ vector for VIRQ
AF		GIME IRQ bits status Set bit = unpolled interrupt as yet.
B0-B1		VIRQ table Pointer to the Virtual Interrupt table.
B2-B3		CC3IO Keybd IRQ vector Vector to keyboard scan sub... used by Clock.
C0-C1	D.SysSvc	System Service Routine entry
C2-C3	D.SysDis	System Service Dispatch Table
C4-C5	D.SysIRQ	Sys State IRQ Routine entry
C6-C7	D.UsrSvc	User Service Routine entry
C8-C9	D.UsrDis	User Service Dispatch Table
CA-CB	D.UsrIRQ	User State IRQ Routine entry
CC-CD	D.SysStk	System stack
CE-CF	D.SvcIRQ	In-System IRQ service
D0	D.SysTsk	System Task number
E0-E1	D.Clock	Secondary Vectors:
E2-E3	D.XSWI3	
E4-E5	D.XSWI2	
E6-E7	D.XFIRQ	
E8-E9	D.XIRQ	
EA-EB	D.XSWI	
EC-ED	D.XNMI	
F2-F3	D.SWI3	Primary Interrupt Vectors:
F4-F5	D.SWI2	(most point to their D.X form above)
F6-F7	D.FIRQ	
F8-F9	D.IRQ	
FA-FB	D.SWI	
FC-FD	D.NMI	

OTHER SYSTEM RAM USAGE

(from above pointers- for info only)

0100-011F	D.Tasks	Task table
0120-015F	00A1-A2	Virtual dat tasks ptr
0200-023F	D.BlkMap	Block usage map ($80=notram, $01=in use, +$02=module)
0300-03FF	D.SysDis	Sys call dispatch table
0400-04FF	D.UsrDis	User call dispatch table
0500-05FF	D.PrcDBT	Proc Desc ptrs table
0600-07FF	D.SysPrc	System proc desc
0800-08FF	D.SysStk	(0900) system stack space
0900-09FF	D.SysMem	System page ram map ($01=in use)
0A00-0FFF	D.ModDir	Module DATImages
1000-1FFF		Global cc3io mem, alarm & system use

SAMPLE SYSTEM LOW MEMORY DUMP (00000-00FFF)

```
     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0000 A001000000000000 0000000000000000  System Direct Page
0010 0000000000000000 00000000FFFF0000   Variables
0020 010000008FAE967E 0000000006233904
0030 0601000000008300 69E3000000000000
0040 020002400A001000 0500060006400900
0050 6D00760000007800 0BF80E8600000000
0060 0000000000000000 0000007FFF917EED
0070 55550074127FFFDF 7EED5F0000000000
0080 8100825F00000000 8000000000000000
0090 6C00080009000000 0315000000F80000
00A0 0101200100100020 00FE69FE7DE9D500
00B0 82E6B98400000000 0000000000000000
00C0 F3160300FE12F27E 0400FD370900E9D5
00D0 0000000000000000 0000000000000000
00E0 FCD2F274F316F000 FE12F287F0000000
00F0 0000F271F271F271 E971F271AD9B0000

     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0100 0101010000000000 0000000000000000  Task Numbers Use Table
0110 0000000000000000 0000000000000000
0120 064011876D406D40 0000000000000000  Virtual Dat: pointers
0130 0000000000000000 0000000000000000   to task # DAT Images
0140 0000000000000000 0000000000000000
0150 0000000000000000 0000000000000000   The ones here are:
0160 0000000000000000 0000000000000000    task 0 (0640)= system
0170 0000000000000000 0000000000000000    task 1 (1187)= grfdrv
0180 0000000000000000 0000000000000000    task 2 (6D40)= dump
0190 0000000000000000 0000000000000000
01A0 0000000000000000 0000000000000000
01B0 0000000000000000 0000000000000000
01C0 0000000000000000 0000000000000000
01D0 0000000000000000 0000000000000000
01E0 0000000000000000 0000000000000000
01F0 0000000000000000 0000000000000000

     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0200 0101010101010301 0303010000000000  Block Map (64 bytes)
0210 0000000000000000 0000000000000000   80 = not ram
0220 0000000000000000 0000000000000000   02 = contains module
0230 0000000000000000 0000000000000101   01 = ram in use
0240 0000000000000000 0000000000000000   03 = module,ram-in-use
0250 0000000000000000 0000000000000000
0260 0000000000000000 0000000000000000  "Mfree" would check
0270 0000000000000000 0000000000000000    this map using
0280 0000000000000000 0000000000000000    F$GBlkMp call.
0290 0000000000000000 0000000000000000
02A0 0000000000000000 0000000000000000
02B0 0000000000000000 0000000000000000
02C0 0000000000000000 0000000000000000
02D0 0000000000000000 0000000000000000
02E0 0000000000000000 0000000000000000
02F0 0000000000000000 0000000000000000
```

```
     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0300 F39397278439852F 863486AE884488DF  System Dispatch Table
0310 894089F68A040000 8AC18AA58ACD98FD   (SWI2)
0320 F72EF7C38BEF8B24 8B98EADB8AE8F636
0330 8C4A8C638C7E8CA8 8CA08D03EAA40000
0340 000096BF96A00000 000000000000EA60
0350 F820F89795FC9945 FD0FFD86F4548D50
0360 8D738DF4F3689062 F386F8F4F8208E24
0370 FB23F967F9BA8E46 FA86FA3FFA25FC56
0380 FC66FC77FCA1FCC1 FAA60000FABD0000
0390 FAF6FB12FB1C85DE 9530F38BF6799D74
03A0 8EAE8EEB8F13F99C 0000000000000000
03B0 0000000000000000 0000000000000000
03C0 0000000000000000 0000000000000000
03D0 0000000000000000 0000000000000000
03E0 0000000000000000 0000000000000000
03F0 0000000000000000 00000000000090DE  (I$call vector)

     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0400 F39396FA8439852F 863486AE884488DF  User Dispatch Table
0410 894089F68A040000 8AC18AA58ACD98FD   (SWI2)
0420 F72EF7B88BE28B17 8B8BEADB8AE8F636
0430 8C4A8C638C7E8CA8 8CA08D03EAA40000
0440 000096BF96A00000 000000000000EA60   Notice that many
0450 0000000000000000 0000000000000000   calls are not
0460 00000000F3680000 0000000000000000   available to the
0470 0000F96700000000 0000000000000000   user.
0480 0000000000000000 0000000000000000
0490 0000000000000000 0000000000008E74
04A0 8EAE8EEB00000000 0000000000000000
04B0 0000000000000000 0000000000000000
04C0 0000000000000000 0000000000000000
04D0 0000000000000000 0000000000000000
04E0 0000000000000000 0000000000000000
04F0 0000000000000000 00000000000090D9  (I$call vector)

     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0500 060678766D000000 0000000000000000  Process Descriptors
0510 0000000000000000 0000000000000000   Base Table (PrcDBT)
0520 0000000000000000 0000000000000000
0530 0000000000000000 0000000000000000  Here: 0600 - n/a
0540 0000000000000000 0000000000000000        0600 - id 1
0550 0000000000000000 0000000000000000        7800 - id 2
0560 0000000000000000 0000000000000000        7600 - id 3
0570 0000000000000000 0000000000000000        6D00 - id 4
0580 0000000000000000 0000000000000000        ...
0590 0000000000000000 0000000000000000
05A0 0000000000000000 0000000000000000
05B0 0000000000000000 0000000000000000
05C0 0000000000000000 0000000000000000
05D0 0000000000000000 0000000000000000
05E0 0000000000000000 0000000000000000
05F0 0000000000000000 0000000000000000
```

```
     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0600 0100000200000000 0000FFFFA0000000  The System (id 1)
0610 0000000000000000 0000000000000000   Process Descriptor
0620 8100000000028100 0000007500000000
0630 0101010000000000 0000000000000000
0640 0000333E333E0004 000100020003003F  - DAT Images
0650 0000000000000000 0000000000000000
0660 0000000000000000 0000000000000000
0670 0000000000000000 0000000000000000
0680 0000000000000000 0052000000000001
0690 0000000000000000 0000000000000000
06A0 0000000000000000 0000000000000000
06B0 0000000000000000 0000000000000000
06C0 0000000000000000 0000000000000000
06D0 0000000000000000 0000000000000000
06E0 0000000000000000 0000000000000000
06F0 0000000000000000 0000000000000000

     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0700 0000000000000000 0000000000000000  - and it's stack area
     ....
07F0 003F004000410042 0043004400450046


     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0800 0000000000000000 0000000000000000  System Stack Page
     ....
08F0 10FEFEF400026D00 FD026D0012E7FE52

     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0900 0101010101010101 0101010101010101  System 64K Page Map
0910 0101010101010101 0101010101010101
0920 0000000000000000 0000000000000000  Each byte = one page
0930 0000000000000000 0000000000000000   01 = in use
0940 0000000000000000 0000000000000000   00 = free
0950 0000000000000000 0000000000000000   80 = not ram
0960 0000000000000000 0000000000010101
0970 0101010101010101 0101010101010101
0980 0101010101010101 0101010101010101
0990 0101010101010101 0101010101010101
09A0 0101010101010101 0101010101010101
09B0 0101010101010101 0101010101010101
09C0 0101010101010101 0101010101010101
09D0 0101010101010101 0101010101010101
09E0 0101010101010101 0101010101010101
09F0 0101010101010101 0101010101010180  (top page is I/O)
```

```
     0 1 2 3 4 5 6 7  8 9 A B C D E F
==== +-+-+-+-+-+-+-+- +-+-+-+-+-+-+-+-
0A00 0FF41ED90D060000 0FF41ED90E300001  Module Directory
0A10 0FF41ED910000000 0EF66CE303000001
0A20 0EF66CE30FAE0001 0EF66CE30FDC0001  Each entry is 8 bytes
0A30 0EF66CE319CF0014 0EF66CE32BFD0014   and contains:
0A40 0EF66CE330510008 0EF66CE33081000C   DAT Image Ptr - 2
0A50 0EF66CE330B10000 0EF66CE330E10004   Block Size    - 2
0A60 0EF66CE336C40004 0EF66CE342FA0001   Offset to Mod - 2
0A70 0EF66CE34FDF0001 0EF66CE35D1C0002   Link Count    - 2
0A80 0EF66CE35D610000 0EF66CE35DA30000
0A90 0EF66CE35DE60000 0EF66CE35E290000  "Mdir" gets this table
0AA0 0EF66CE35E6C0000 0EF66CE35EAF0000    using F$GModDr call.
0AB0 0EF66CE35EF20000 0EF66CE35F350002
     ....
0E80 0000000000000009 0000000800000000   and towards the end
0E90 0000000000000009 0000000000000000    is the temporary
0EA0 0000000000000006 0000000000000000    DATImage stack.
0EB0 0000000000000000 0000000000000000
0EC0 0000000000000000 0000000000000000
0ED0 0000000000000000 0000000000000000
0EE0 0000000000000000 0000000000000000
0EF0 0000000000000001 00020003003F0000
     ....
0FF0 00000000003F0000 0000000000000000  - end system vars.

1000 ---------------- ----------------  Begin CC3 global mem
```

OS9 SYSTEM CALLS

The OS9 system service calls, a SWI2 opcode followed by the call number, are the only recommended means to utilize memory, I/O and program control. A process inherits the SWI vectors from its parent, but may change them by the F$SSWI call.

Most of the calls are handled by the OS9 or OS9P2 modules. Any I/O call is vectored to IOMAN, which does its own internal table look-up. Another exception is the get-time call, which is dealt with by the Clock module.

There are two tables that contain the call vectors. The first table is from $00300-003FF, and is the table for calls made while in the system state. The user call table is at $00400-004FF.

To be in the system state, a program must currently be executing code within a system, manager, or driver module. This mainly occurs because of a system call. In other words, once a SWI call is made, all calls made within that call are vectored by the system table.

There are three main reasons for having a system mode. First, if a program is aborted while doing I/O (system mode), the program must be allowed to release I/O resources for other programs to use. Second, path numbers used while in the system mode are the actual path desc block number, and so must be distinguished from a process's path table pointer. And third, since new SWI and IRQ vectors are set on entry to the system mode, time is saved by bypassing this set-system-mode sub.

When a SWI2 call is made, the registers are placed on the current process's stack, and the stack pointer is saved in the process descriptor for easy access by the system modules. This way, the modules can use all the registers (except the SP) with impunity, and they all know where to get parameters passed and where to return values. Each module may do a fair amount of SWI2 calls itself. Under Level One, that meant that you needed to keep a large stack area for your program. That's not so important under Level Two, as the system or process descriptor stack is used mostly instead.

The calls from $28-$33 are regarded as privileged calls, since they have resource allocation powers that would be dangerous if used by a passing (non-system) program module. They may only be used while in the system state.

```
================================================================
                    SWI2 SERVICE REQUEST                  OS9
================================================================
          USER SWI2                     SYS SWI2
              l                            l
          State=sys                        l
          DP = 0                        DP = 0
          U=SP, store P$SP              U = SP
          Table=user (D.UsrDis)         Table=sys (D.SysDis)
              l                            l
          BSR Docall                    BSR Docall
          State=user                       l
              l                            l
             END                          END
```

```
================================================================
Docall                    Subroutine                         OS9
================================================================
    Get PC off IRQ stack
    Get next byte (call)
    Inc stack PC past call byte
             l
    (I/O call >=$80 ?) n---------------->.
             ly                          l
    Vector at table-2 (I/O)      (Call >= $37?) y-------->.
             l                           ln               l
             l                  Get call vector          l
             l                   (vector=0?) y---------->l
             l<--------------------------l                l
             l                                            l
    JSR the call vector                         'Illegal SVC'
             l
.<--n (C set for err?)
l            ly
l   Return Reg.B=err code in B
l----------->l
             l
    Return lower 4 bits of CC
             l
          END SUB
================================================================
I/O Vector              I/O SERVICE CALL                   IOMAN
================================================================
           USER                SYS
            l                   l
       Table=CBC8          Table=CBEA
            l<------------------l
            l
       (call>$90?) y------------------>'Illegal SVC'
            ln
       Get call vector
       JMP to vector        (Hidden RTS to OS9 Docall above)
```

```
===============================================================
 SWI 01                      F$LOAD                       IOMAN
===============================================================
    Alloc temp proc desc
    Totram=0, Totmod=0
    Set proc prty=caller's
    Open EXEC. path to file
    F$AllTsk, D.Proc=temp
.------->l
l   Call ReadMod header
l        l
l   (M$ID 87CD okay?) n--------------->.
l        ly                            l
l   Call ReadMod rest                  l
l        l                             l
l   F$VModul into moddir err---------->l
l        l                             l
l   (known module?) n---->update       l
l        ly              TotMod        l
l        l<----------------l           l
l   Set FoundMod flag                  l
l<-------l                             l
                                       l
         .<----------------------------l
         l
    D.Proc=caller proc
    Close EXEC. path
    Check TotRam-TotMod
    Release blocks unused
    Dealloc temp proc desc
    (FoundMod flag set?) n-->return err
         ly
    Return ptr to first module
         l
        END
===============================================================
 Subroutine                  ReadMod                      IOMAN
===============================================================
           sub
            l
       ModSiz=ModSiz+request
.<--n (ModSiz >TotRam?)
l           ly
l      Calc # of blocks needed
l      Find free blocks and set=$01
l      Set into temp proc desc datimg
l      TotRam=TotRam+new blocks
l      F$SetTsk: update datimg
l           l
l---------->l
            l
       Read in header/module
            l
           RTS
```

```
=================================================================
                         Verify Module
=================================================================
    Call CRC check
    F$FModul in ModDir
.<-n (find same name?)
l         l
l   (revision higher on new?) n---> E$KnwMdl
l         ly
l------->l
    Set ModImg
    MPDAT,MPtr,MDLink=0
    MBSiz=up to and including module
         l
.<-n (module in another block?)
l        ly
l   Free other entry
l------->l
    Mark BlkMap with "ModBlock"
         l
        END
=================================================================
 SWI 02                     F$UNLINK                       OS9P2
=================================================================
    Calc proc desc datimg block #
    (does BlkMap show module?) n---->okay end
         ly
    Decrement P$Link cnt
    Search ModDir
.<-------l
l
l        .<----------------.
l   Next ModDir entry      l
l        l                 l
l------->l                 l
         l                 l
    (same MD$MPtr?) n----->l
         ly                l
    (same block #?) n----->l
         ly
    MD$Link cnt-1
.<-n (link cnt=0?)
l        ly
l   Do IODEL if needed
l   Call ClearDir sub
l------->l
    Decrement P$Link cnt
.<-n (link cnt=0?)
l        ly
l   Mark P$Datimg blocks as free
l------->l
        END
```

```
=================================================================
 Subroutine                ClearDir                        OS9P2
=================================================================
        sub
         l
    Get dir entry block #
    Check BlkMap flag ------->end if already clear
    Pt to ModDir
         l<---------------.
.<-n (blk=this entry?)    l
l        ly               l
l   End if MD$Link<>0     l
l------->l                l
    Next ModDir entry     l
    (last entry?) n------>l
         ly
    Free BlkMap flags
    CLear DatImg
    Clear ModDir entry
         l
        RTS
=================================================================
 SWI 03                    F$FORK                          OS9P2
=================================================================
    F$AllPrc desc
    Copy parent's P$User,Prior,DIO
    I$Dup std 0,1,2 paths
    Call MakeProc
    F$AllTsk for child
    F$Move parameters to child map
    F$Move register stack from proc desc to map
    F$DelTsk of child
    Return child id to caller
    Set P$CID of parent, P$PID, P$SID of child
    Clear SysState of child
    F$AProc: activate child
         l
        END
=================================================================
 Subroutine                MakeProc                        OS9P2
=================================================================
        sub
         l
    F$SLink to module -ok--->.
         lok               l
    F$Load module          l
         l<----------------l
         l
    (Prgrm/Systm+Objct?) n-----> err
         ly
    Set P$PModul
    F$Mem for new D.Proc
    Set new register stack in proc desc
         l
        rts
```

```
===============================================================
 SWI 4B                   F$AllPrc                        OS9P2
===============================================================
    Check D.PrcDBT table for free entry
    F$SrqMem 512 byte proc desc
    Set D.PrcDBT entry
         l
    Set P$ID in proc desc
    Clear P$DATImg
    State = SysState
         l
        END
===============================================================
 SWI 3F                  F$ALLTSK                         OS9P1
===============================================================
    Quick End if has P$Task
    Call ResTsk
    Call SetTsk
         l
        END
===============================================================
 SWI 42                  F$RESTSK                         OS9P1
===============================================================
    Point to D.Tasks table
    Skip first two (reserved for systm)
    Find free entry, mark it used
    Return entry number as task
         l
        END
===============================================================
 SWI 43                  F$RELTSK                         OS9P1
===============================================================
    Point to D.Tasks table
    Clear task entry
     unless is SysTsk
         l
        END
===============================================================
 SWI 41                  F$SETTSK                         OS9P1
===============================================================
   Clear ImgChg flag in P$State
   Get P$Task
   Copy P$DATImg's to task map
         l
        END
===============================================================
 subroutine             Check Task
===============================================================
   P$State has ImgChg flag set? n-->rts
         ly
   Call SetTsk
         l
        rts
```

```
===============================================================
 SWI 04                       F$WAIT                       OS9P2
===============================================================
      (children?) n-------------> 'No Children' error
           ly
      (any dead yet?) y----------------------->.
           ln                                  l
     Return Regs.A=0                      Regs.D= ID/code
     Stop IRQ's                           Fix sibling links
     Place proc at front of W.Queue       Dealloc. child desc
     Make a fake RTI stack                    l
     F$Nproc:start next process              END
          l
        . . .
          l
  <F$Exit of child wakes parent>
   <Regs.D has child ID/code>
          l
      Get real SP
          l
         END
==============================================================
 SWI 08                     F$SEND                        OS9P2
==============================================================
      (dest ID=0?) y------>Send signal to all!
           ln
     Send to ID only
          l
         END

         <->
          l
       Stop IRQ's
          l
.--n  (code=abort?)
l          ly
l    Make proc condemned state
l--------->l
      (has signal?) y--------.
           ln               (signal=wake?) n---->error
           l<----------------l
     Store signal
     Wake up proc
     Signal=0 if signal=1
     Insert proc in A.Queue
          l
      END of SUB
```

```
===============================================================
 SWI 06                     F$EXIT                        OS9P2
===============================================================
      P$Signal = Regs.B
      Close all I/O paths
      Return data memory
      Unlink primary module
           l
      Point to our last child
.<---------l
l
l          .<------------.
l          l             l
l  .<-n (is it dead?)    l          Return proc desc's of all
l  l       ly            l           dead (F$Exit'd) children.
l  l  Dealloc proc desc  l
l  l------>l             l
l     Zero parent ID     l          Live kids are now orphans.
l     Point to sibling   l
l          l             l
l--------->l             l
     (any children?) y-->l
           ln
           l
     (we have parent?) y------>.    If we are orphan ourselves
           ln                  l      we exit quickly.
      Dealloc our proc desc    l
.<---------l                   l
l                              l    If parent hasn't F$Waited,
l                              l     we are marked as Dead for
l                              l     parent's Wait or Exit.
l                              l
l                              l
l        .<-----------n (parent waiting?)
l        l                     ly
l    Mark us as       Take parent out of W.Queue
l       Dead          F$Activate parent
l        l            Put ID/code in parent's Regs.D
l        l            Fix sibling links
l        l            Dealloc child proc desc
l        l                     l
l------->l------------------->l
                               l
                         D.Proc = 0000
                               l
                              END
```

```
=================================================================
 SWI 00                    F$LINK                        OS9P1
=================================================================
     Type=Reg.A
     Name ptr=Reg.X
     Find module dir entry -err------>$DD error
         l
.<--y (reentrant?)
l        ln
l    (link cnt=0?) n---------->$D1 error
l        ly
l------->l
         l
     Inc link cnt
     Return type/lang/hdr/entry
         l
        END
=================================================================
 SWI 0C                    F$ID                          OS9P2
=================================================================
     Get ID from Proc Desc
     Get User from Proc Desc
         l
     Return ID in Reg.A
     Return User in Reg.Y
         l
        END
=================================================================
 SWI 0D                    F$SPRIOR                      OS9P2
=================================================================
     ID# = Reg.A
     Find Proc Desc for ID  -err------>'Not Found'
         l
     (same index?) n-------------->'Not Yours'
         ly
     New proc priority=Reg.B
         l
        END
=================================================================
 SWI 0E                    F$SWI                         OS9P2
=================================================================
     Point to Proc Desc's SWI table
     Type= Reg.A
     (type>3?) y-------------->'Illegal SWI Code'
         ln
     New vector=Reg.X
         l
        END
=================================================================
 SWI 0F                    F$PERR                        IOMAN
=================================================================
     Get Error Path (#2) from Proc Desc table
     Convert Reg.B code to ASCII number
     Print 'ERROR #'
     Print err number
         l
        END
```

```
==============================================================
 SWI 15                    F$TIME                       CLOCK
==============================================================
     Destination=Reg.X
     F$Move D.Time to dest
         1
        END
==============================================================
 SWI 16                   F$SETIME                      OS9P1
==============================================================
     Source=Reg.X
     Move source to D.Time
     F$Link to 'Clock'
         1
      (error?) y------------->'Unknown Module'
         1n
     Jmp to Clock init  (after this, Clock usually sets it's
         1                   own F$Setime call - see below)
       (END)
==============================================================
 System Module             Init                         CLOCK
==============================================================
     Set constants/vars
     Insert Clock vector at D.IRQ
     F$SSVC new Time call
         1
        END
```

```
==================================================================
 SWI 2A                        F$IRQ                        IOMAN
==================================================================

      Get packet values
      Get max # IRQ entries from INIT
      Point to poll table (<$62)
          l
.<--y  (Reg.X=0?)
l          ln
l       (mask=0?)  y---------->error
l          ln
l       Search for empty
l          l
l       (no empties?)  y------>'Poll Table Full'
l          l
l       Sort by priority
l       Insert new entry
l          l
l          END
l
l------->.   * KILL ENTRY *
           l
       Find entry by data address          At INIT Module+$0C
       Delete it                            is max # entries.
       Move rest up in table
           l
           END

------------------------------------------------------------------
POLLING TABLE ENTRY FORM:

00-01   Address of status port
02      Flip byte
03      Mask byte
04-05   IRQ service address
06-07   Storage memory address
08      Priority (0-low,255-high)

==================================================================
 System Module         IRQ Polling Routine                  IOMAN
==================================================================

    Point to polling table
     Get max # entries from INIT
         l--------------------->.
.-------->.                     l
l    Point to next entry        l
l     (end of table?) y-------- l ------------->.
l        ln                     l               l
l        l<---------------------l               l
l    Get status byte                      'Table Full Err'
l    Flip and Mask                          Return error
l<--n (found it?)
l        ly
l    Do service routine
l<--y  (error?)
         ln
        END
==================================================================
```

```
==================================================================
 SWI 80/81              I$ATTACH/DETACH                    IOMAN
==================================================================

   F$link--device desc
     (get header, device address) err---------------->.
   F$link--device driver                                l
     (get driver entry address)   err---------------->l
   F$link--file manager                                 l
     (get mgr entry address)      err---------------->l
          l                                             l
   Get max # of entries (INIT)                     I$DETACH
   Get device table add (<$60)                          l
          l                                        Dec user cnt
.-->(entry empty?) y--------------->.                  Unlink desc
l   (same desc?)   n------------>.  l                  Unlink driver
l   (mem alloc'd?) y-------->.  l  l                   Unlink mgr
l        l                   l  l  l                        l
l<--n (any user?)            l  l  l                       END
l        ly                  l  l  l
l   Insert in I/O queue      l  l  l
l<-----wakeup                l  l  l
l                            l  l  l
l        .<------------------l  l  l              DEVICE TABLE ENTRY:
l        l                      l  l               0 - Driver mod
l   Save entry ptr              l  l               2 - Static mem
l        l<---------------------l  l               4 - Desc mod
l        l                         l               6 - File mgr mod
l   (same port add?) n------------>l            8 - User count
l   (same driver?)  n------------->l
l   (user cnt=0?)  y-------------->l
l        l                         l
l   Save user cnt                  l
l        l<------------------------l
l   Point to next entry
l<--n (last?)
         ly
     (entry found?) y-------------->l
         ln                         l
    Find empty spot                 l
       (error?) y--->'Table Full'   l
         l                          l
.<--y (mem alloc'd?)                l
l        ln                         l
l   Allocate drvr mem & clear       l
l   Set V.Port add in mem           l
l   Do driver init sub              l
l------->l                          l
    Insert device tble data         l
         l<-------------------------l
         l
     (Check desc/drvr modes) err--------> 'Illegal Mode'
         l
.<--y (user cnt=0?)
l      (device shareable?) n------------> 'Device Busy'
l------->l
    Increment user cnt
    Return table entry in Regs.U
         l
        END
```

```
=================================================================

=================================================================
SWI 83                       I$DUP                          IOMAN
=================================================================

   Get free path # from Proc Desc   err---->'Path Table Full'
   Find path desc of old path       err---->'Unknown Path'
   Increment path desc image cnt
   Return new Proc path ptr in Regs.B


=================================================================
SWI 83/84               I$CREATE/OPEN                       IOMAN
=================================================================

   Get free path # from Proc Desc
        l
   Get requested mode
   Allocate path desc
   Do File Manager Create/Open
        l
   Put path desc # in Proc path table
   Return Proc path number in Regs.A
        l
       END

=================================================================
SWI 8F                       I$CLOSE                        IOMAN
=================================================================

   Get Proc path ptr for A=path#
   Zero that path ptr in Proc Desc
   Find path desc
   Decrement # of open images
        l
.<--y (current proc ID?)
l       ln
l    Update I/O queue
l    Save caller's stack in PD.REGS
l    Do File Manager Close
l       l
l    Wake up proc's in pd.links
l       l
l      (proc.ID=path.ID?) n--->.
l       ly                     l
l   Clear path.ID              l
l------->l<--------------------l
        l
     (open images=0?) n---->.
        ly                  l
    I$DETACH device         l
    Kill path desc          l
        l                   l
        l<------------------l
       END

=================================================================
```

```
==============================================================
SWI 86                        I$CHGDIR                   IOMAN
==============================================================

    Save SWI code for later use
    Allocate temp path desc
        l
    Do File Manager Chgdir sub      (RBFman finds dir desc LSN &
        l                             dr# and puts in Proc Desc)
    .<---l------------>.
   l                   l
  data dir            exec dir
   l                   l
 (dec user cnt in device table for old dir's device)
 (inc user cnt in device table for new dir's device)
 (set new device table entry into Proc Desc)
        l
    Point to device table entry for this temp path
    I$Detach drive
    F$Dealloc64 - kill this temp path desc
        l
        END

--------------------------------------------------------------

  PROCESS DESCRIPTOR DEFAULT DIR ENTRIES:
   data     exec                                  from
   20-21    25-26    Device table entry ptr      (IOMAN)
   22       27       Drive number (not used)     (RBFman)
   23-24    28-29    Dir file desc LSN           (RBFman)


==============================================================
SWI 89                        I$READ                     IOMAN
==============================================================

    Find path desc
        l
      (read attr?) n----------------> 'No Permission'
        ly
         l<-----------------.
.<--n (path desc in use?)    l
l        ly                  l
l   Place in I/O Queue       l
l       wakeup ------------->l
l
l------->.
         l
    Do File Manager Read sub
    Wake up others in I/O Queue
    Clear path user if still us     (PD.CPR)
        l
        END

==============================================================
```

```
===============================================================
                         Subroutine                      IOMAN
===============================================================
ALLOCATE PATH DESCRIPTOR   (Open, Create)

     Get pd's base (D.PthDBT)
     Allocate 64 byte block
     Set user cnt=1, mode=mode requested
           l
     Point to pathname
     Skip blanks
           l
.--y (1st char='/'?)          If '/', it's full pathname;
l          ln                  Else use default dirs for this
l          l                    process descriptor.
l      dir type?
l    ---------------
l    l               l
l    data dir         exec dir
l    (get device tble entry from Proc Desc)
l          l
l      (entry=0?)  y------------------------------>.
l          ln                                      l
l    Point to device desc name                     l
l          l                                       l
l-------->l                                        l
           l                                       l
     Parse name of device                          l
        (error?)   y------------------------------>l
           ln                                      l
           l                                       l
     Attach device                                 l
     Save table ptr in path desc                   l
           l                                       l
        (attach err?)  y-------------------------->l
           ln                                      l
           l                                       l
     Get device desc init size                'Bad Pathname'
     Move up to 32 bytes to path desc         Deallocate pd block
           l                                       l
          END                                   Error End
```

==

IRQ HANDLING

I have included this general text for the hackers out there.

Technical notes on the flow of hardware interrupt handling in OS9 L-I CoCo ver 1.X or 2.0, and OS9 L-II Gimix ver 2.0 or CoCo 1.X.

The 6809 has three hardware interrupt lines, NMI, FIRQ, and IRQ. This doc concentrates on the IRQ, which is the one used by OS9 for it's clock and I/O device polling routines.

I'll cover the various paths OS9 may take when it receives an IRQ, which don the current level, revision & system state. Note that because I only touch on IRQ-related code, other variables are involved.

IRQ'S - CLOCKS and DEVICES

There are two main source catagories of IRQ's: clock and device. They're both vectored to the same handler at their start, but branch differently. (CoCo OS-9 adds the VIRQ and FIRQ, but they end up being treated as an IRQ.)

The timesharing type has to do with updating the D.Time variables and calling the kernal's D.Clock process-switching algorithm. It comes from a regular timed interrupt source, such as the 60Hz Vertical Sync on the CoCo, or a clock chip or timer on other systems.

The other type is from a device asking for service. Usually that device's driver has entered an F$IRQ request, so that the OS will know where to vector, after the polling routine has found that IRQ source device.

BASIC INTERRUPT HANDLING

All 6809 machines fetch their cpu interrupt vectors from a ROM that can be read at logical addresses FFFX. The IRQ vector is at FFF8-F9.

Level-I CoCo 1/2

The ROM in these computers vectored to 010C, which contains a BRA to 0121, which does a JMP [D.IRQ].

Level-I Coco 3

The new ROM vectors IRQs to FEF7, where it does a LBRA to 010C, maintaining compatability with 1.X or 2.0 OS-9. See CoCo 1/2 above. L-II of course needs the FEXX page pseudo-vectors so that there is always IRQ handling code across all task maps.

Level-II Task Switching

In Level-II, interrupts are ROM-vectored to the code at the top of OS9p1. This code lies within the page that is mapped across all task maps (on some systems, an interrupt causes a hardware reset of the task register to the system map instead, so a user has the full 64K available). In either case, the task register is set to the SysTask, the Direct Page register is set to zero, and then- JMP [D.IRQ] D.IRQ defaults to the IntXfr (interrupt transfer) code in OS9p2, which does what boils down to a JMP [D.XIRQ]. This is changed by the Clock module.

OS-9 VECTOR INITIALIZATION

```
When OS9 first cranks up, it sets the following:

 D.UsrIRQ  - kernal user-irq routine
 D.SysIRQ  -    "    system  "
              both of which will and up JSR'ing [D.Poll]
 D.SvcIRQ  - has D.SysIRQ in it
 D.IRQ     - kernal JMP [D.SvcIRQ]
 D.Poll    - kernal COMB, RTS
```

This means that initially all IRQ's go thru the kernal to [D.SvcIRQ] back to the kernal's own Sys/UsrIRQ code, which then calls [D.Poll] to find the source. As the kernal does not do polling, and IOMan isn't initialized yet, D.Poll returns an error. The Sys/UsrIRQ code then shuts off IRQ's by setting the CC bits as a precaution.

TRANSFER TO SYSTEM STATE - Level-I or II

Whether a program is in the user or system state when an interrupt occurs affects what D.SvcIRQ contains.

If in user state, it contains the vector constant copied from D.UsrIRQ. The routine in OS9p1 at that address saves the task's SP, sets SWI vectors to use system vectors, and copies D.SysIRQ into D.SvcIRQ.

The OS9p1 routine at [D.SysIRQ] does not save or set up anything as you are already in the system state. This helps speed interrupt handling.

IOMAN INIT

When the first I$Call is made, the kernal links to and initializes IOMAN (I/O MANager). Ioman inserts a vector to itself in D.Poll. From then on, IRQ's still go thru the kernal [D.SvcIRQ] to the Sys/UsrIRQ code, but their call to
[D.Poll] is now honored by ioman, which does the source searching (polling).

Also on the init call, ioman sets up several tables. These are the device table [D.DevTbl], polling table [D.PolTbl], and on the CoCo the VIRQ (virtual irq) table [D.CltTab].

These tables will be used by ioman for keeping track of active devices, inserting and deleting F$IRQ entries, and by ioman's D.Poll routine in finding the source of an IRQ.

CLOCK INIT and OPERATION

We must include Clock modules here because they are important in the IRQ heirachy. A side note: some clock modules keep their device address in the M$Size (data size) portion of their module header.

Clock modules keep track of the real time. Interrupts usually are vectored almost directly to them, and they decide for themselves if a clock IRQ was involved. In effect, a special device driver IRQ routine.

They are not in a polling table because a) the clock must be serviced quickly, and b) they may jump directly or thru another module to the kernal's timesharing routine (D.Clock) and so cannot be called as a subroutine such as device IRQ handlers are.

When the first F$STime call is made (best from SysGo), OS9p1 links to any module called "Clock", and JSR's to it's entry point. There the Clock module inserts itself into the system D.IRQ vector, so that it gets called first.

After that, IRQ's come to Clock, who checks to see if it's timer was the source. If so, it updates the time variables as needed, and jumps via D.Clock to the kernal (L-II jumps via D.XIRQ to the kernal).

If the timer or clock chip was NOT the IRQ source, then Clock jumps [D.SvcIRQ] so that OS9 can check for the correct device.

Exception #1: on the CoCo L-I ver 1.X, the IRQ's go first to CCIO (so it could time the disk motors), then to Clock via [D.AltIRQ], then Clock continued by [D.Clock].

Exception #2: on the CoCo L-I ver 2.0, Clock jumps via [D.AltIRQ] to the CCIO keyboard scan. CCIO finishes the jump to [D.Clock].

IOMAN IRQ POLL SYNOPSIS

As we know now, when the Clock's D.IRQ code finds that an IRQ has occurred from other than it's IRQ, the IOMan D.Poll vector is eventually called.

IOMan looks thru the Polling Table, which has been presorted by device priority. Each Q$POLL address is read, XOR'd with the Q$FLIP byte, AND'd with the Q$MASK byte, and if is not=$00 after all that, the Q$SERV routine in the driver for that device is called to service and clear that IRQ.

If the driver service code finds that a mistake has been made in it's selection, it can set the C bit, and IOMan will continue the search thru the table. See D.SvcIRQ above.

IRQ FLOWCHARTS

```
 CoCo Level I
                IRQ
                 l
            ROM:jmp [D.IRQ]
           (was it clockirq?) y------------------->update time
                nl                                      l
           jmp [D.SvcIRQ]                           jmp [D.AltIRQ]
 (D.UsrIRQ)----- or ------(D.SysIRQ)                scan keyboard
      l                        l                    jmp [D.Clock]
 D.SvcIRQ = D.SysIRQ      jsr [D.Poll]              update ticks
 jsr [D.Poll]                  l<-----------y (ticks>0 or SysState?)
      l                      rti                       nl
      l                                                 l
        l<----------------------------------------------l
choose next proc
D.SvcIRQ = D.UsrIRQ
     rti


    D.Poll
scan devices, do driver irq sub
    rts

==================================================================================
Level II
                IRQ
                 l
            ROM:jmp to allmap page (XFEXX)
            TaskReg = SysTask
            jmp [D.IRQ]:                  (old L-II):            (CoCo L-II):
           (was it clockirq?) y----------->update time - - - - ->update time
                nl                               l                      l
           D.SvcIRQ = D.Poll             D.SvcIRQ = D.Clock     D.SvcIRQ = D.Virq
                   l<---------------------------l<- - - - - - - - - -l
            jmp [D.XIRQ]:
  (D.UsrIRQ)----- or ------(D.SysIRQ)
      l                        l
SP = D.SysStk             jsr [D.SvcIRQ]
D.XIRQ=D.SysIRQ                rti
jsr [D.SvcIRQ]
      l
  (slice up?) n----------.
       ly                l
choose proc to run       l
        l<---------------l
D.XIRQ = D.UsrIRQ                                                   D.Virq:
switch task to user                                             update Virq table
      rti                                                       call D.Poll if Virq
                                                                jsr [D.KbdIRQ] scan
     D.Poll:                             D.Clock:               check & do alarm sig
find source, driver IRQ sub           update ticks              jmp [D.Clock]
     rts                                   rts                      (rts)
```

NOTES:

All code is OS9p1, except D.IRQ/D.Virq-->Clock, and D.Poll-->IOMan.
In most cases, IRQ's (and FIRQ's) are not reenabled until the RTI.

The L-II D.Clock is a subroutine, but the L-I D.Clock both updates the ticks, and then falls through to the timeshare routine.

Notice that if an interrupt occurs while in , other processes get achance to run if the current process is out of time.

GENERAL NOTES:

virqs end up as irqs

```
 Just after the end of the OS9p1 module are the offsets to the following
default code within it:
  D.Clock routine
  D.SWI3  (these are D.X... in Level-II)
  D.SWI2
  D.FIRQ
  D.IRQ
  D.SWI
  D.NMI
```

IRQ-RELATED DP.VARS and SYSTEM TABLES

The following are the Direct Page ($00XX) variables that have to do with interrupt processing, and their addresses on the CoCo and GIMIX machines. Each contains a two-byte vector to the code within a System module that handles it, or point to a table.

Your system may vary, so check your OS9Defs file, if you don't own one of those computers. Addresses are included simply to give a rock to cling to.

NAME	L-I	L-II	
D.Init	2A-2B	24-25	Init Module pointer
D.DevTbl	60-61	80-81	I/O Device Table pointer
D.PolTbl	62-63	82-83	I/O Polling Table pointer
D.FIRQ	30-31	F6-F7	FIRQ handler
D.IRQ	32-33	F8-F9	IRQ
D.NMI	36-37	FC-FD	NMI
D.SvcIRQ	38-39	CE-CF	IRQ vector set by Clock depending on IRQ type
D.Poll	3A-3B	26-27	Source device polling routine
D.AltIRQ	6B-6C		Alternate IRQ hook
D.Clock	81-82	E0-E1	Kernal timeshare routine
D.ClTb	86-87	B0-B1	VIRQ device entry table ptr
D.KbdIRQ		B2-B3	Keyboard scan
D.XIRQ		E8-E9	Secondary IRQ vector set to D.UsrIRQ or D.SysIRQ

Then there are the Direct Page variables that contain initialized vector constants, so that interrupts may be handled differently depending upon the OS state:

```
D.UsrIRQ  3C-3D  CA-CB   User   state D.SvcIRQ vector
D.SysIRQ  3E-3F  C4-C5   System state D.SvcIRQ vector
```

IOMAN TABLES --

The size of these tables is calculated from the DEVCNT and POLCNT entries in the system INIT module.

```
  DEVICE TABLE ENTRIES
V$DRIV   00-01   Driver module addrss
V$STAT   02-03   Device static storage
V$DESC   04-05   Device Descriptor
V$FMGR   06-07   File Manager
V$USRS   08      Device User Count
DevSiz   equ .

  POLLING TABLE ENTRIES
Q$POLL   00-01   Polling address (device status byte address)
Q$FLIP   02      Flip byte for negative logic IRQ bits
Q$MASK   03      Mask byte for IRQ status bit
Q$SERV   04-05   Driver IRQ service routine
Q$STAT   06-07   Device static memory pointer
Q$PRTY   08      Device polling priority (position in table)
PolSiz   equ .
```

INSIDE OS9 LEVEL II

Devices

```
OS9 I/O                                                    OS9 I/O
                          RBFMAN FILE
==================================================================

   PD.- path descriptor vars       V$-- device table
   V.-- device static storage      Q$-- IRQ poll table
   DD.- drive tables (LSN 0)       P$-- process descriptor
------------------------------------------------------------------
 Opening a disk (RBF) file takes the following steps:

#  VAR       MOD     ACTION
-  -----     -----   ---------------------------------------------
1  PD.PD     IOMAN   Allocates a 64-byte block path descriptor.
   PD.MOD            Sets access mode desired.
   PD.CNT            Sets user cnt=1 for this path desc.

2  PD.DEV    IOMAN   Attaches the device (drive) used.
   V$STAT             Allocates memory for device driver (CCDisk).
   V.PORT             Sets device address in driver static memory.

3  V.NDRV    CCDISK  The driver's init subroutine is called to
   V.TRAK                initialize the device, and static memory
   DD.TOT                (drive tables) to default values.

4  Q$POLL    OS9     Sets up IRQ polling table entry.
    ...                ( address, flip & mask bytes, service add,
   Q$PRTY                static storage, priority of IRQ )

5  V$DRIV    IOMAN   Sets up rest of device table.
   V$DESC                ( module addresses of desc, driver, mgr)
   V$FMGR
   V$USRS            Sets user count of device=1

6  PD.OPT    IOMAN   Copies device desc info to path desc.
    ...               . ( drive #, step rate, density, tracks,
   PD.SAS                 sides, interleave, seg alloc size )

7  PD.BUF    RBFMN   Allocates buffer for file use.
   PD.DVT             Copies device table entry for user.
   PD.DTB             Calc's drive table add for quick ref'rnce.

   DD.TOT    CCDISK  Copies LSN 0 init info to drive table.
    ...                ( diskette's format, root dir, ID, attr's,
   DD.RES                number of tracks, sectors, bitmap size )

   PD.DSK    RBFMN   Gets disk ID and finds the file:
   PD.DFD             LSN of directory file desc
   PD.DCP             Entry # of pathname in directory file
   PD.FD              LSN of pathname's file desc
   PD.CP              Current file pos
   PD.SIZE           File size
   PD.SBL             Offset from beginning of file segment
   PD.SBP             LSN of file segment
   PD.SSZ             Segment size in sectors
   PD.ATT             File attributes (DSEWR)

8  P$PATH    IOMAN   Puts path desc # in proc desc I/O table.
                      Returns table pointer to user as path number.
==================================================================
```

```
==============================================================
                    DEVICE DRIVER ENTRIES              RBFMAN
==============================================================
INIT             U =device static memory       CC,B <error code
                 Y =device descriptor
--------------------------------------------------------------
  . Set V.NDRV to number drives controller handles
  . Set DD.TOT to non-zero value so RBFman can read LSN 0
  . Set V.TRAK to high number if driver controls seek op code
  * Use F$Irq to place driver IRQ service routine in poll table
  . Init controller
  * Copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0

==============================================================
READ             U =device static memory       CC,B <error code
                 Y =path descriptor
               B,X =LSN
--------------------------------------------------------------
  . Get PD.BUF  buffer address from path descriptor
  . Get PD.DRV  drive number from path descriptor
  . Send LSN converted to track and sector to controller
  * Copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0
  . Read the data into the buffer if not a DMA controller
  . If LSN 0, copy DD.SIZ bytes into drive table

==============================================================
WRITE            U =device static memory       CC,B <error code
                 Y =path descriptor
               B,X =LSN
--------------------------------------------------------------
  . Get PD.BUF  buffer address from path descriptor
  . Get PD.DRV  drive number from path descriptor
  . Send LSN converted to track and sector to controller
  . Write the data into the buffer if not a DMA controller
  * Copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0

==============================================================
GETSTT           U =device static memory       CC,B <error code
PUTSTT           Y =path descriptor
                 A =status call
--------------------------------------------------------------
  . Do wildcard driver call if not handled by IOMAN/RBFman

==============================================================
TERM             U =device static memory       CC,B <error code
--------------------------------------------------------------
  * Wait for any I/O to complete
  * Disable any device IRQ's
  * Remove device from IRQ polling table

==============================================================
 * IRQ Service Routine

  . Kill IRQ request if necessary
  . Send wakeup signal to V.Wake
  . Clear V.Wake
  . Clear Carry bit
  . RTS
```

```
==================================================================
 * Interrupt driven devices only !
==================================================================


==================================================================
DEVICE VARIABLES           Static Memory                    RBFMAN
==================================================================

Name      Offset   Description
------------------------------------------------------------------
V.PAGE    00       Port extended address
V.PORT    01-02    Device address
V.LPRC    03       Last active process ID (not used
V.BUSY    04       Active process ID (dev busy flag)  0=not busy
V.WAKE    05       Process ID to awake after command completed
V.USER    .        Beginning of file mgr/driver var's
------------------------------------------------------------------
V.NDRV    06       Number drives controller can handle
          07-0E    Reserved
------------------------------------------------------------------
DRVBEG    .        Beginning of drive tables
                    (One table for each drive, up to V.NDRV)

==================================================================
This section of each table copied from LSN 0 of disk.    Dr#0
------------------------------------------------------------------
DD.TOT    00-02    Number of sectors                     0F-11
DD.TKS    03       Number of tracks                      12
DD.MAP    04-05    Number bytes in allocation map        13-14
DD.BIT    06-07    Sectors/bit in map (sectors/cluster)  15-16
DD.DIR    08-0A    LSN of root directory                 17-19
DD.OWN    0B-0C    Owner's user number                   1A-1B
DD.ATT    0D       Disk attr (D S PE PW PR E W R)        1C
DD.DSK    0E-0F    Disk ID                               1D-1E
DD.FMT    10       Disk format                           1F
DD.SPT    11-12    Sectors/track                         20-21
DD.RES    13-14    Reserved                              22-23
DD.SIZ    .        Size of bytes to copy from LSN 0      .
------------------------------------------------------------------
V.TRAK    15-16    Current track                         24-25
V.BMB     17       Bit map in use flag                   26
V.FileHd  18-19    Open file list                        27-28
V.DiskID  1A-1B    Disk ID                               29-2A
V.BMapsz  1C       Bitmap size in sectors                2B
V.MapSct  1D       Lowest reasonable bitmap sector       2C
V.ResBit  1E       Reserved bit map sector               2D
          1F-25    Reserved                              2E-34
------------------------------------------------------------------
DRVMEM    .        Drive table size
                    (other drive tables follow)
==================================================================

 Drive table address = DRVBEG + ( PD.DRV * DRVMEM )
 Also found in PD.DVTB

==================================================================
```

```
=====================================================================
Module                  DEVICE DESCRIPTOR                      RBFMAN
=====================================================================

Name        Offset    Description
---------------------------------------------------------------------
M$ID        00-01     Sync bytes ($87CD)
M$Size      02-03     Module size
M$Name      04-05     Offset from start to module name string
M$Type      06        Type/lang  ($F1)
M$Revs      07        Attr/revision
M$Parity    08        Header parity

M$FMgr      09-0A     File manager name offset
M$PDev      0B-0C     Driver name offset
M$Mode      0D        Device capabilities
M$Port      0E-10     Device extended address
M$Opt       11        Number of options in initialization table

IT.DTP      12        Device type  (1=RBF)
IT.DRV      13        Drive number (0...n)
IT.STP      14        Step rate:    0- 30 ms
                                    1- 20 ms
                                    2- 12 ms
                                    3-  6 ms

IT.TYP      15        Device type: bit0- 0=5 1/4    1=8 inch
                                    bit5- 0=noncoco 1=coco
                                    bit6- 0=os9std  1=nonstd
                                    bit7- 0=floppy  1=hard

IT.DNS      16        Density:      bit0- 0=single  1=double
                                    bit1- 0=48 tpi  1=96 tpi

IT.CYL      17-18     Cylinders  (tracks)
IT.SID      19        Sides
IT.VFY      1A        0= verify disk writes
IT.SCT      1B-1C     Sectors/track
IT.TOS      1D-1E     Sectors/track (track 0)
IT.ILV      1F        Sector interleave
IT.SAS      20        Minimum #sectors/segment alloc
            .         End of option table.
            21-       Name strings here.
```

PATH DESCRIPTOR — PD.Variables — RBFMAN

Name	Offset	Description^h
PD.PD	00	Path number
PD.MOD	01	Access mode 1=read 2=write 3=update
PD.CNT	02	Number of paths using this path desc
PD.DEV	03-04	Device table entry address
PD.CPR	05	Current Proc ID using this path for I/O
PD.RGS	06-07	Address of user's register stack
PD.BUF	08-09	Data buffer (256 bytes) address for RBF/drvr
PD.SMF	0A	Buffer state: see below
PD.CP	0B-0E	Current file position
PD.SIZ	0F-12	File size
PD.SBL	13-15	File beginning segment number (FSN)
PD.SBP	16-18	Actual segment beginning LSN
PD.SSZ	19-1B	Segment size in sectors
PD.DSK	1C-1D	Disk ID
PD.DTB	1E-1F	Drive table address for this drive^h

This section copied by IOMAN from the Device Desriptor:

Name	Offset	Description
PD.OPT	20	Device class 0=SCF 1=RBF 2=PIPE
PD.DRV	21	Drive number 0-n
PD.STP	22	Step rate
PD.TYP	23	Device type 5,8,hard
PD.DNS	24	Disk density
PD.CYL	25-26	Number of tracks (cylinders)
PD.SID	27	Number of sides
PD.VFY	28	Verify flag 0=do verify on write
PD.SCT	29-2A	Sectors/track
PD.TOS	2B-2C	Sectors/track (track 0)
PD.ILV	2D	Sector interleave
PD.SAS	2E	Segment allocation size in sectors^h
PD.TFM	2F	DMA transfer mode
PD.Exten	30-31	Path extension (not used)
PD.SToff	32	Sector/track offset
PD.ATT	33	File attributes (D S PE PW PR E W R)
PD.FD	34-36	File descriptor LSN (list of segments)
PD.DFD	37-39	Dir file desc LSN (of dir holding file)
PD.DCP	3A-3D	File dir ptr (filename entry in dir file)
PD.DVT	3E-3F	Device table entry address for user^h

Buffer state flag bits:

$01 - buffer modified
$02 - current sector
$04 - file desc in buffer
$08 - end of file sector
$10 - end of file
$20 - in disk driver
$40 - buffer busy^h

==

Template DEVICE DESCRIPTOR RBFMAN

==

```
 IFP1
 USE DEFS/OS9defs
 USE DEFS/RBFdefs
 ENDC

type SET Devic+Objct
revs SET ReEnt+1

 MOD  rend,devnam,type,revs,fmnam,drvnam
 FCB $FF              all access modes^b
 FCB $FF,$FF,$40      device address^b
 FCB optl             option length^b

optns    EQU *

 FCB DT.RBF   type = 1 for RBFman devices^b
 FCB $03      drive number (0...n)^b
 FCB $02      step rate ^b
 FCB $40      device type:        bit0- 0=5 1/4    1=8 inch
*                                 bit5- 0=noncoco  1=coco
*                                 bit6- 0=os9std   1=nonstd
*                                 bit7- 0=floppy   1=hard

 FCB $01      density:            bit0- 0=single   1=double
*                                 bit1- 0=48 tpi   1=96 tpi

 FCB $00,$23    cylinders  (tracks)
 FCB $01        sides
 FCB $01        0= verify disk writes
 FCB $00,$12    sectors/track
 FCB $00,$12    sectors/track (track 0)
 FCB $01        sector interleave
 FCB $01        minimum #sectors/segment alloc

optl EQU *-optns

devnam FCS /D3/
fmnam  FCS /RBF/
drvnam FCS /CCDisk/
       EMOD
rend   EQU *
```

==

This is a typical RBF device descriptor. You may modify the constants and names (devnam, drvnam) to suit your device name, driver, and characteristics.

==

Ron - ok, ok <heh-heh>. Have you tried formatting the disk anyway? I can't remember now, but I don't think the desc extensions are used there. Anyway try one of these:

```
  DIVA     0A   or   09
  DIVY   0100      0080
  DIVU   0302      0101
```

DIVA is the # of bits used for the cylinder number.

DIVY is the # of heads * sectors/trk * shift value.

DIVU mask (# of bits set) is (DIVA-8) bits. The DIVU shift is DIVA-8.

If you've disassembled the driver, you'll see that you end up with the sectors remaining in D (shifted to the left), with the cyl hi in the last one or two bits of B. They mask off those bits, and put them as the cyl hi value. Then D must be shifted right to get back in the correct position. Thus the shift value is dependent upon how many cylinders you have.

I THINK either of the two sets of values above will work. Also I think your drive is 15meg, not 20.

```
===========================================================
Disk Format                LSN FORMATS                RBFMAN
===========================================================

LSN 0 (ID sector)  DD.vars               FILE DESC Sector:
---------------------------               --------------------
DD.TOT  00  Number disk sectors           FD.ATT 00 DSPEPWEWR
DD.TKS  03  Number tracks                 FD.OWN 01 Owner ID
DD.MAP  04  Bytes in alloc map            FD.DAT 03 Last YMD:HM
DD.BIT  06  Sectors / cluster             FD.LNK 08 Link count
DD.DIR  08  Root dir LSN                  FD.SIZ 09 File #bytes
DD.OWN  0B  Owner's user num              FD.DCR 0D Date create
DD.ATT  0D  Disk attributes               FD.SEG 10 Segment list
DD.DSK  0E  Internal disk ID              --------------------
DD.FMT  10  Format,dens,sides             Seg list:
DD.SPT  11  Sectors / track                Up to 48 5-byte
DD.RES  13  Reserved                        entries: 3LSN,2size
DD.BT    15  Bootstrap LSN strt           --------------------
DD.BSZ  18  Boot size in bytes            Dir file:
DD.DAT  1A  Create time YMD:HM             29 bytes-name
DD.NAM  1F  Disk name(32 bytes)             3 bytes-LSN desc

LSN 1 (Bit map)
 Each bit = 1 cluster of the number of sectors from DD.BIT.
-----------------------------------------------------------
```

Each disk file has at least one sector: the File Desc. This sector (see format above) contains the segment list, which is a list of the sectors used by that file. Each 5 byte entry (in order) points to the next block of sectors: the beginning LSN of the block, and the number of contiguous LSN's from and including the beginning block LSN.

Thus, if your disk files got so fragmented that the file could not be held in 48 blocks of any number of neighboring sectors, the File Desc couldn't handle it. This is extremely unlikely, of course.

The sectors pointed to in the segment list contain the file itself, which might be a m/l program, an ASCII file, or a list of other files.

A file that consists of a list of other files is assigned (by the Attr or Makdir commands) the Directory attribute. The list of files, and THEIR File Desc sector, is kept in a special order (see Dir file above right).

The directory file can have an essentially unlimited number of 32-byte entries consisting of the file name (up to 29 char) and the 3-byte LSN of the filename's File Desc sector. Note that the first two filenames are automatically inserted by RBFman and they are '.' and '..', which point respectively to the dir file's own File Desc, and the File Desc of the dir file just above it in heirarchy.

DD.DIR points to the LSN of the first File Desc which has the Directory attribute, and is a list of all the files and directory files that you see when you do a 'Dir' of the device holding the disk.

```
==============================================================
CREATE                   File Mgr Entry                 RBFMAN
==============================================================

     Drop bit 7 of attr parm
     Find file LSN
       (file exits?) y------------->'File Exists'
          ln
       (dir found?)  n------------->'Path not Found'
          ly
     Get segment PSN of dir file
     Get size of dir file
     Allocate >=one sector (segment)
     Save number of sectors alloc'd
     Save new segment PSN
     Seek start of dir file
           l<----------------.
     Get 32 byte entry       l              * Make new dir entry *
.<--y (empty spot?)          l
l         ln                 l
l    Point to next 32        l
l       (error?) n---------->l
l         ly
l       (eof ?)  n---------------->Error End
l         ly
l    Extend file by 32
l    Update file size
l    Read new sector
l------->l
         l
     Clear 32 bytes
     Move <=29 name chars to buffer
     Move alloc'd desc LSN to buffer
     Write out updated dir file LSN
         l
     Clear buffer                           * Make desc sector *
     State=file desc
     Insert file attr, user ID, time, date
     Set link count=1
         l
     Check number sectors alloc'd
         l
.<--y (any sectors left?)
l         ln
l    Set first seg LSN=desc LSN+1
l    Set first seg size=sectors-1
l------->l
     Write out file desc LSN
     Put file desc LSN in path desc
     Zero file size, pos in path desc
     Seek 0 in new file
         l
        END
```

```
================================================================
OPEN                      File Mgr Entry                   RBFMAN
================================================================

    Find dir LSN
         l
.<--y (file desc PSN?)
l        ln (@ - open whole device)
l        l
l     (mode=dir?) y------------> '$D6 error'
l        ln
l    Zero seg begin PSN,FSN
l    Get #sectors from drv table
l    Store as pd.segment size
l    Store*256 as pd.file size
l        l
l       END
l
l------->.
         l
     Check file attr  err------> 'No Permission'
         l
     PD.pos, FSN, msb seg size=0
     Move file attr fm buffer to pd.attr
     Move first LSN & segment size to path desc
     Move file size to pd.file size
         l
        END

----------------------------------------------------------------

Path desc var's:

PD.CP    0B  4  Current file position
PD.SIZ   0F  4  File size
PD.SBL   13  3  Segment beginning file sector (FSN)
PD.SBP   16  3  Segment beginning disk sector (LSN)
PD.SSZ   19  3  Segment size in sectors
PD.ATT   33  1  File attr (D S PE PW PR E W R)
PD.FD    34  3  File desc PSN  (the list of sectors for file)
PD.DFD   37  3  Dir file desc PSN  (one level up from 34)
PD.DCP   3A  4  Dir file entry pointer to this filename

----------------------------------------------------------------
```

The FSN, as I call it, is the offset in sectors from the beginning of the actual file position.
The LSN is the actual disk sector that the FSN is equal to.
The PSN is also the actual disk LSN.

```
==============================================================
CLOSE                     File Mgr Entry                RBFMAN
==============================================================
                                                        $D3EB


      (images=0?) n--------->END
         ly
       (mode=write?) n----------------------->.
         ly                                   l
       (file desc ?) n----------------------->l
         ly                                   l
      Insert date in desc buff              Return buffer
      Move file size to desc buff             l
      Check disk ID & write buff             END
      Check EOF status
         l
        END
```

```
==============================================================
CHGDIR                    File Mgr Entry                RBFMAN
==============================================================
                                                        $D43A


      Open pathname
          l
      .---l---.
      l       l
    data     exec
          l
    Put dr# & file desc LSN in Proc Desc
    Return buffer
          l
         END
```

```
==============================================================
SEEK                      File Mgr Entry                RBFMAN
==============================================================
                                                        $D4FA


.<--n (sector in buffer?)
l        ly
l    Get pos of buff start
l<--y (seek within buff?)
l        ln
l    Get buff within seek
l------->l
     Set new pd.pos
         l
        END
```

```
=============================================================
Find File                  Subroutine                   RBFMAN
=============================================================

      State=altered
      Request buffer, set PD.BUF
      PD.file desc PSN=0
      PD.disk ID=0
          l
      (1st char='/'?) n------>.
          ly                  l
      Get device name         l
           l<----------y (1st char='@'?)
          l                   ln
          l             PD.file desc PSN= Proc Desc default
          l              data/exec dir desc PSN
          l<------------------l
      PD.DVT=PD.DEV
      PD.DTB=static mem+drvbgn+(dr# * drvmem)
          l
.<--y (was 1st char '@'?)
l         ln
l     Read LSN 0
l     PD.disk ID=disk ID
l         l
l<--y (PD.file desc PSN=0?)
l         ln
l     PD.file desc PSN = root dir PSN
l         l
l------->l
      Save ptr to pathname
.------->l
l     Read file desc LSN
l         l
l      (next char '/'?) n------------------------>.
l         ly                                      l
l     Check file attr  err--->'No Permsn'         l
l     Read 32 bytes                               l
l         l--------------->.              (end of name?) y-->.
l                          l                F$Parsenam        l
l .----->.                 l                      l<----------l
l l  Pt to next filename   l              Save ptr to name
l l       l<---------------l                      l
l l<-y (unused entry?)                           END
l l       ln
l l<-n (same names?)      * FOUND NAME *
l         ly
l     Set PD.dir file PSN & entry ptr
l     PD.file desc PSN=this LSN
l         l
l     (at end of file?) y------->'EOF error'
l<-------ln

=============================================================
  Returns last dir file PSN & entry found.
  File desc PSN = the LSN at that dir file position.
  IF '@', PSN=0, size= entire disk
=============================================================
```

```
OS9 I/O                                                    OS9 I/O
                          SCFMAN FILE
============================================================

   PD.- path descriptor vars        V$-- device table
   V.-- device static storage       Q$-- IRQ poll table
                                     P$-- process descriptor
------------------------------------------------------------
Opening a serial (SCF) file takes the following steps:

#  VAR      MOD    ACTION
-  -----    -----  -----------------------------------------
1  PD.PD    IOMAN  Allocates a 64-byte block path descriptor.
   PD.MOD          Sets access mode desired.
   PD.CNT          Sets user cnt=1 for this path desc.

2  PD.DEV   IOMAN  Attaches the device (driver) used:
   V$STAT           Allocates memory for device driver (RS232).
   V.PORT           Sets device address in driver static memory.

3           RS232  The driver's init subroutine is called to
                     initialize the device.

4  Q$POLL   OS9    Sets up IRQ polling table entry.
    ...              ( address, flip & mask bytes, service add,
   Q$PRTY              static storage, priority of IRQ )

5  V$DRIV   IOMAN  Sets up rest of device table.
   V$DESC              ( module addresses of desc, driver, mgr)
   V$FMGR
   V$USRS           Sets user count of device=1

6  PD.OPT   IOMAN  Copies device desc info to path desc.
    ...                ( upper/lower case, lf, lines/page,
   PD.XOFF               file chars, baud rate, echo device )

7  PD.BUF   SCFMN  Allocates 1 byte buffer.
   V.LINE           Copies desc lines/page to lines left var.
   PD.DV2           I$Attach echo device, set dev table ptr.

8  P$PATH   IOMAN  Puts path desc # in proc desc I/O table.
                    Returns table pointer to user as path number.
------------------------------------------------------------

2,3,4,5  only if first time for that device,
              else V$USRS = V$USRS + 1
                   PD.DEV = device table entry.

4        only if device uses IRQ's.

============================================================
                    DEVICE DRIVER ENTRIES              SCFMAN
============================================================
INIT             U =device static memory      CC,B <error code
                 Y =device descriptor
------------------------------------------------------------
  . Initialize any static vars, constants
  . Use F$Irq to place driver IRQ service routine in poll table
  . Init controller
```

```
================================================================
READ             U =device static memory          A <char
                 Y =path descriptor            CC,B <error code
----------------------------------------------------------------
  . Get next char from input buffer in static memory
  . If none: copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0


================================================================
WRITE            U =device static memory          A <char
                 Y =path descriptor            CC,B <error code
----------------------------------------------------------------
  . Put char into static memory output buffer
  . Enable ready-to-transmit interrupt
  . If full: copy V.BUSY to V.WAKE, F$Sleep 0, check V.WAKE=0


================================================================
GETSTT           U =device static memory       CC,B <error code
PUTSTT           Y =path descriptor
                 A =status call
----------------------------------------------------------------
  . Do wildcard driver call if not handled by IOMAN/RBFman


================================================================
TERM             U =device static memory       CC,B <error code
----------------------------------------------------------------
  . Wait for output buffer to empty
  . Disable any device IRQ's
  . Remove device from IRQ polling table


================================================================
IRQ Service Routine

  . Read data if necessary into input buffer.
  . If pause char read, set V.PAUS of memory area V.DEV2 <>0.
  . If quit or keybd interrupt is read, send appropriate signal
     to the last user (V.LPRC) and error code=char.
  . Write the output buffer to device until it is empty,
     disable ready-to-transmit interrupt.
  . Send wakeup signal to V.WAKE
  . Clear V.WAKE
```

DEVICE VARIABLES		Static Memory	SCFMAN

Name	Offset	Description
V.PAGE	00	Port extended address
V.PORT	01-02	Device address
V.LPRC	03	Last active process ID
V.BUSY	04	Active process ID (dev busy flag) 0=not busy
V.WAKE	05	Process ID to awake after command completed
V.USER	.	Beginning of file mgr/driver var's
V.TYPE	06	Device parity type
V.LINE	07	Lines til end of page
V.PAUS	08	Pause request 0=none
V.DEV2	09-0A	Echo device memory area
V.INTR	0B	Interrupt char
V.QUIT	0C	Quit char
V.PCHR	0D	Pause char
V.ERR	0E	Error collector
V.XON	0F	X-ON char
V.XOFF	10	X-OFF char
	11-15	used by Japanese computers
V.PDLHD	16-17	Path desc's head link for device users
	18-1C	reserved
V.SCF	.	End of SCFman vars
	1D-	Free for device driver vars

V.LPRC is used by the IRQ routine. If a quit or interrupt char is received, the routine should signal the last process to use the device with the signal associated with that char.

This is why the Shell usually catches your <shft-brk> or <brk> multi-task/ abort keystrokes, and takes the appropriate action. Note that if your program uses the device itself, you get the strange alternating set of Shell/ program messages.

```
==============================================================
Module                  DEVICE DESCRIPTOR                 SCFMAN
==============================================================
```

Name	Offset	Description
M$ID	00-01	Sync bytes ($87CD)
M$Size	02-03	Module size
M$Name	04-05	Offset from start to module name string
M$Type	06	Type/lang ($F1)
M$Revs	07	Attr/revision
M$Parity	08	Header parity
M$FMgr	09-0A	File manager name offset
M$PDev	0B-0C	Driver name offset
M$Mode	0D	Device capabilities
M$Port	0E-10	Device extended address
M$Opt	11	Number of options in initialization table:
IT.DTP	12	Device type (0=SCF)
IT.UPC	13	Case: 0= U/l 1=Upper only
IT.BSO	14	Backspace: 0=bsp pnly 1=bsp, space, bsp
IT.DLO	15	Delete: 0=bsp over line 1=<cr>
IT.EKO	16	Echo: 0=no echo
IT.ALF	17	Auto linefeed: 0=no auto linefeed
IT.NUL	18	Null: number of delay nulls sent after <cr>
IT.PAU	19	Pause: 0=no pause at end of page
IT.PAG	1A	Lines per page
IT.BSP	1B	Backspace code char from device
IT.DEL	1C	Delete-line code from device
IT.EOR	1D	End of record code from device
IT.EOF	1E	End of file code fm dev ('EOF' is echoed)
IT.RPR	1F	Reprint line code from device (buffer echoed)
IT.DUP	20	Duplicate line code (all buffer echoed)
IT.PSC	21	Pause code from device
IT.INT	22	Interrupt code from device
IT.QUT	23	Quit code from device
IT.BSE	24	Backspace code echoed to echo device
IT.OVF	25	Line too long code to echo (bell)
IT.PAR	26	Parity: init byte for ACIA control register
IT.BAU	27	Baud rate
IT.D2P	28-29	Echo device name offset
IT.XON	2A	X-on char
IT.XOFF	2C	X-off char
IT.COL	2C	Number of columns
IT.ROW	2D	Number of rows
		End of option table.
	2E-	Name strings here.

PATH DESCRIPTOR	PD.Variables	SCFMAN

Name	Offset	Description
PD.PD	00	Path number
PD.MOD	01	Access mode 1=read 2=write 3=update
PD.CNT	02	Number of paths using this path desc
PD.DEV	03-04	Device table entry address
PD.CPR	05	Current Proc ID using this path for I/O
PD.RGS	06-07	Address of user's register stack
PD.BUF	08-09	Data buffer (256 bytes) if used
PD.FST		Beginning of SCFman vars
PD.DV2	0A-0B	Echo device table ptr (output)
PD.RAW	0C	Edit flag 0=read/write 1=readln/writln
PD.MAX	0D-0E	Readline max char cnt
PD.MIN	0F	Device use flag 0=my devices
PD.STS	10-11	Status routine module address
PD.STM	12-13	Reserved for status routine
	14-1F	Reserved
		This section copied by IOMAN from the Device Descriptor:
PD.OPT	20	Device class 0=SCF 1=RBF 2=PIPE
PD.UPC	21	Case 0=upper and lower 1=upper only
PD.BSO	22	Backspace 0=bsp 1=bsp,space,bsp
PD.DLO	23	Delete 0=bsp over line 1=cr/lf
PD.EKO	24	Echo 0=no echo
PD.ALF	25	Auto lf 0=no auto line feed after cr
PD.NUL	26	Null cnt nulls sent after cr/lf for delay
PD.PAU	27	Pause lines left before pause; 0=no pause
PD.PAG	28	Lines / page
PD.BSP	29	Backspace char
PD.DEL	2A	Delete-line char
PD.EOR	2B	End of line char (normally $0D, 0=til EOF)
PD.EOF	2C	End of file char (read only)
PD.RPR	2D	Reprint line char
PD.DUP	2E	Duplicate last line char
PD.PSC	2F	Pause char
PD.INT	30	Keyboard interrupt char (ctrl-C)
PD.QUT	31	Keyboard abort char (ctrl-Q / Break)
PD.BSE	32	Backspace echo char
PD.OVF	33	Line overflow char (Bell code)
PD.PAR	34	Device init byte (parity)
PD.BAU	35	Baud rate code
PD.D2P	36-37	Offset to DEV2 name string
PD.XON	38	X-ON char for ACIA
PD.XOFF	39	X-OFF char
PD.ERR	3A	Most recent I/O error status
PD.TBL	3B-3C	Device table entry copy for user

Input of a keyboard INT/QUT character returns that char as the I/O error code, and sends an interrupt/abort signal to the last active user process of this path.

```
================================================================
Template                DEVICE DESCRIPTOR                 SCFMAN
================================================================

 ifpl
 use /dd/defs/defsfile
 endc

type SET    DEVIC+OBJCT
revs SET    REENT+1

 MOD    len,nam,type,revs,mgr,drvr

 FCB   READ.+WRITE.  mode
 FCB   $FF          ext'd add
 FDB   $FF00        device address
 FCB   opt-*-1      option byte cnt
 FCB   DT.SCF       SCF device

 FCB   0       case= UPPER and lower
 FCB   1       backspace=bs sp bs
 FCB   0       delete=bs over line
 FCB   1       auto echo
 FCB   1       auto linefeed
 FCB   0       no nulls on CR
 FCB   0       no page pause
 FCB   24      lines per page
 FCB   08      backspace char
 FCB   $18     delete line char
 FCB   $0D     end of record char
 FCB   0       no end of file char
 FCB   04      reprint line char
 FCB   01      dup last line char
 FCB   $17     pause char
 FCB   3       abort char
 FCB   5       interrupt char
 FCB   $08     backspace echo char
 FCB   07      line overflow (bell)
 FCB   0       printer type
 FCB   4       baud rate=2400
 FDB   echo    echo device
opt    EQU     *

nam    FCS     "Remote"
       FCS     "  "      patch space
mgr    FCS     "SCF"     file mgr name
drvr   FCS     "CCIO"    driver name
echo   FCS     "T1"      echo device

       EMOD
len    EQU     *
       END
```

==

Using 'Shell </remote >/t1 >>/t1' allows you to use the CoCo keyboard while visual output is redirected (and input echoed) to a terminal display connected to the RS-232 port.

==

```
OS9 I/O                                                         OS9 I/O
                           PIPEMAN FILE
=========================================================================

   PD.- path descriptor vars      V$-- device table
   V.-- device static storage     Q$-- IRQ poll table
                                  P$-- process descriptor
-------------------------------------------------------------------------
Opening a pipe (PIPEMAN) file takes the following steps:

#  VAR      MOD     ACTION
-  -----    -----   ------------------------------------------
1  PD.PD    IOMAN   Allocates a 64-byte block path descriptor.
   PD.MOD           Sets access mode desired.
   PD.CNT           Sets user cnt=1 for this path desc.

2  PD.DEV   IOMAN   Attaches the device used:
   V$STAT           Allocates memory for device driver (none).
   V.PORT           Sets device address in driver static memory.
                     ( address = 00 0000 )

3           PIPER   The driver's init subroutine is called to
                      initialize the device (does nothing).

4                   No interrupts used by PIPER.

5  V$DRIV   IOMAN   Sets up rest of device table.
   V$DESC             ( module addresses of desc, driver, mgr)
   V$FMGR
   V$USRS           Sets user count of this pipeman=1

6  PD.OPT   IOMAN   Copies device desc info to path desc.
                     ( just type= Pipe )

7  PD.BUF   PIPMN   Allocates 256-byte buffer.
                    Sets begin, end, nextchar ptrs in PD.

8  P$PATH   IOMAN   Puts path desc # in proc desc I/O table.
                    Returns table pointer to user as path number.

-------------------------------------------------------------------------

2,3,5    only the very first time a pipe is used,
              else V$USRS = V$USRS + 1
                   PD.DEV = device table entry
4        not used at all

  Note that both the driver and descriptor (Piper, Pipe) are
  only dummy modules, there just to make IOMAN happy.

=========================================================================
```

```
================================================================
PATH DESCRIPTOR           PD.Variables                  PIPEMAN
================================================================

Name      Offset   Description
----------------------------------------------------------------
PD.PD     00       Path number
PD.MOD    01       Access mode   1=read  2=write  3=update
PD.CNT    02       Number of paths using this path desc
PD.DEV    03-04    Device table entry address
PD.CPR    05       Current Proc ID using this path for I/O
PD.RGS    06-07    Address of user's register stack
PD.BUF    08-09    Data buffer (256 bytes) address each Create
PD.FST             Beginning of Pipeman vars
----------------------------------------------------------------

          0A       Read user
          0B       Number read users
          0C       Read signal
          0D       End of line char

          0E       Write user
          0F       Number write users
          10       Write signal
          11       Not used

          12-13    End of buffer
          14-15    Pointer to next address to store char
          16-17    Pointer to next address to read char
          18       Data flag  0=no data in circular buffer

================================================================
```

Pipeman uses no static memory. Instead, it allocates a 256 byte buffer each time a 'file' is created. This buffer is returned when the last user has closed a path to it, or there are no more readers.

Note: these are for Level One. I haven't had a chance to check on L-II vars, but the concept will be the same, with the exception that Pipeman will do an F$Move of the data between process maps.

GENERAL DRIVER NOTES

LEVEL TWO DEVICE ADDRESSES

(Message from me to CompuServe OS9 Forum 24Mar87:)

Finally went looking for the reason why I've been telling everyone that their extended device addresses had to be $07FXXX instead of the old L-I $FFXXXX. Here's the dope:

L-II IOMan (just like a GIMIX) takes the address ($07FF) top bytes, and converts it to an I/O block number... on the CoCo, it translates to block $3F. Well, this makes sense as far as it goes, as extended address $07FXXX is indeed the top of mem; that is, the last block or $3F block.

It then looks to see if that block is already mapped into the system 64K map...if it's block $3F, it already is, cuz that's the kernal and I/O area from $E000-FFFF.

BUT! If the extended address does NOT translate out to $3F ($FFFF = block number $FF!!), then it maps that block into the system map. And ignores it as RAM cuz it's obviously I/O, right? So you just lost 8K in your System 64K map.

8K is a lot to take away from the system map, and that's when those of you using Rogue got the dreaded 207 error for no seeming reason.

You also got the error if it couldn't map the block in. This error number has been changed to 237 (no ram), in the latest versions, btw.

Since the converted logical address would also be wrong, some things died. Devices with hard coded addresses had fewer problems.

That's the scoop, guys.. so make sure to use the $07FXXX if writing up new device descriptors. That is, offset $0E in your device descriptor must be = $07 and the next =$FX.

On the other hand, $00 0XXX should be okay also, as block 00 is also always in the system map.

SCF SPECIAL CHARS

As you know, SCF drivers are responsible for sending either an S$Abort (for character matching V.QUIT) or S$Intrpt (char = V.INTR) signal to the last process (V.LPRC) that used the device.

A note about the above... character matching is done against the V.xxx static memory variables, NOT against the path descriptor PD.yyy equivalents. This is even though the V.xxx were set by SCF to the PD.yyy characters when the process gained the use of the device.

Why not just use the PD stuff? Because most devices are IRQ-driven, and there's no easy way for OS9 to get the path descriptor pointer to the asynchronous IRQ code that is servicing that driver. Hence they are copied to the V.xxx driver memory which IS known, as IOMan has it in it's interrupt polling table.

RBF THINGS

The Device Descriptor describes the maximum capabilities of the device; the Path Descriptor is used for variables pertaining to the file itself (pos, length, lsn's, dirs, etc); and the Drive Tables are for info about THAT one diskette currently in the drive (format, tracks, sectors, bitmap size, root dir, id, which track the head is pointing to, whether a process is changing the bit map, etc).

Those of you who write RAMdisk drivers usually follow the lead of the floppy drivers. Okay, but some parts are different. For example in your Init, you should probably set the DD.TOT to the actual sector size of the "drive". And unless you wish to use it as some kind of flag, there is NO need to do anything to DD.TRAK. That's done there only so floppy drives can restore to track zero the first time they're called. If your driver doesn't need it, don't mess with it.

IRQ's On LEVEL TWO

Let's take a quick look at how ACIAPAK sets up for interrupts, to give other driver writers some help.

```
ACIAPAK Init Routine:

  Does an F$IRQ call
  Stops all interrupts
  Resets the CART PIA line for no Multi-Pak FIRQ's
  Gets Direct Page 0092 (GIME IRQ register shadow)
  OR's it with 01 to enable CART-->IRQ conversions
  Stores that value back at 0092 and FF92
  Restores the CC register
  Sets the MPI slot for CART from slot 0

What CLOCK Does on Interrupt:

  On an IRQ, Clock read GIME FF92 IRQ register
  OR'd that value into Direct Page 00AF
  JSR'd the Interrupt Polling Routine...

  ACIAPAK Interrupt Routine:

  Get Direct Page 00AF (contains FF92 IRQ read by Clock)
  NOT with 01 to indicate that CART IRQ was read
  Store that value back at 00AF
  Do the interrupt routine
  Go back and check for another IRQ before RTS
```

OTHER L-II DRIVER CHANGES

Because the system map is so much like under L-I, only a few changes must be made. The most obvious is the interrupt handling, as discussed above. Timing loops have to compensate for the 2Mhz speed, also.

For RBF devices that must change slots, the main (and sometimes almost only) change is that D.DMAReq has moved from 006A to 008A.

The file managers take care of moving data between system maps, so many old drivers will work fine (once the descriptor address is changed as pointed out). For example, once the address has been changed the Disto Parallel Printer port driver works.

One last note: CC3DISK no longer turns on precompensation on the inner tracks. Supposedly most drives never needed it.

INSIDE OS9 LEVEL II

Windows

INSIDE 0S9 LEVEL II
Windows
Section 1

THE WINDOW DRIVERS

The windowing system on the CoCo-3 is composed of the window device descriptors, the main driver CC3IO specified in those descriptors, and several co-modules that handle window output.

```
The modules and a schematic of their relationship:

Term    - Actually, the W0 descriptor OR a VDG descriptor
 W1-W7  - Window descriptors
 W      - Special window descriptor

CC3IO   - Keyboard scanning (60 times a second if key down)
          Joystick/mouse reads
          Some stat calls

VDGInt  - Emulates L-I v2.0 gfx environment
          Adds hires gfx screens mapped into proc space

WindInt- Preprocessor for hi-level windowing/menu calls
          plus window codes

GrfInt  - Preprocessor for window codes
          Some stat calls

GrfDrv  - Text/gfx display
```

```
----------------------------------------------------------------

               IOMAN
                 l
               CC3IO - Term W W1 W2 ..     Main driver/desc
                 l
    .<----either-l-or-----------.
    l              l            l
    l            GrfInt      WindInt       Output processing
    l              l-----.------l
  VDGInt                 l
                      GrfDrv               Screen data

        (video output)
```

COMPARISON WITH OTHER I/O DEVICES

Like other OS9 devices, reading and writing and stat calls are done through a main driver. Each device has it's own address, static memory, and has an input buffer for type-ahead. Outputted characters are not queued, but go straight to the screen.

Unlike others, though, each window also shares the same input device (the keyboard or mouse). They also share use of the GIME chip. This means that some way must be used to keep track of which window sets up it's display on the GIME, and which window gets the input from the keyboard. For this purpose, all of the window devices also share a common or global memory.

This global memory is located at in block 00, extended address 001000-001FFF, and is always mapped in for the CoCo terminal driver modules to use. A very preliminary and cursory look at this memory area is provided in the next section of the book.

The /W descriptor also introduces a new technique. This wildcard device flags CC3IO to open the next free window in place of it. I think that requesting the name from a path opened using /W will instead return /Wx instead (x=number).

Instead of hardcoding window numbers, good L-II programs that need to open another virtual terminal should use /W.

CC3IO

CC3IO is very similar to it's L-I (ver 2.0) counterpart, CCIO. Some of it's code is even the same for the keyboard, lo-res mouse read, and so on. However, where CCIO used CO80 or CO32 as comodules to handle the screen output, CC3IO now passes codes on to the GrfInt/GrfDrv or VDGInt comodules. (The name "CO80" can still be found within CC3IO, but was probably there just for debugging purposes, as it is no longer used.)

VDGINT

VDGInt contains the equivalent of the Level One CO32 and GRFO modules. It handles the 32x16 text screens, semi-graphics and original VDG-style graphics screens.

Because of this emulation, you can still run many older programs that ran on the CoCo-1/2's, including TSEDIT.

In addition, VDGInt provides for new screens that allow speed-dependant programs to take advantage of the CoCo-3's high resolution graphics. Unlike the GrfInt screens that are not mapped into a program's space, VDGInt graphics screens are. This means that games like Koronis Rift can directly access the screen memory to be displayed, allowing much faster updating of the screen than by using escape codes.

VDG text screens are normally allocated from the system map, as allocating a full 8K block just for a 512 byte display would be wasteful. To provide compatibility, the use of the SS.AlfaS GetStat call WILL map the screen into the caller's task space (since it returns the address within a logical 64K area), along with any other system variables that just happened to be in the same system map block. For this reason, programs that use this call should be careful to stay within the 32x16 screen area, lest they accidentally write over crucial system data.

Windows within a screen are not provided for, although it is possible to set up more than one VDG screen. And, you can still <CLEAR-key> between these screens and normal windowing screens.

Character and graphics functions are not provided for the CoCo-3 specific modes. The only text output is through use of the 32x16 character display.

GRFINT/WINDINT

GrfInt takes the parameters passed with a window code (as when you do a "display 1b 31 5 38"), checks them for values exceeding limits or specifications, and stores the possibly converted parameters in the system map global memory and window tables.

GrfInt then calls GrfDrv with an internal code, which is used as a table index to call the appropriate GrfDrv subroutine for any screen manipulation.

WindInt will be included with the Multiview graphics shell package. It will take the place of GrfInt, providing the same calls plus adding new ones for creating pull-down menus, boxed windows, scroll bars and other hi-level windowing abilities.

GRFDRV

GrfDrv is the module that does any actual storage or drawing of data on the screen. It also handles allocation of screen memory and buffers. In other words, anything specific to the CoCo-3.

Both GrfInt and WindInt will use GrfDrv as the driver that manipulates the video data. By breaking things up this way, it's possible for perhaps just a new GrfDrv to be written for other display devices, or the next CoCo.

The most unique aspect of the GrfInt/GrfDrv combination for lovers of L-II is that it's code size, and the need to have direct access to so much memory (like 32K for each gfx screen), caused the authors of CoCo-3 L-II to adopt what amounts to an extension of the 64K system map into another 64K space to handle the memory needed.

A CLOSER LOOK:

CC3IO

On initialization, CC3IO inserts it's IRQ handler vector into D.AltIRQ at $00B2 in the direct page variables. It also sets vectors for window select, mouse reads and the terminal bell (this is used by CLOCK's F$Alarm call).

Depending on the device type ($80= window, else= VDG), it will link or load, and inititialize the Interface module required. Obviously, VDG device types use VDGInt. Window devices cause CC3IO to first try locating WindInt. If that fails, it then goes after GrfInt.

On IRQ's, CLOCK calls CC3IO as a subroutine to read the keyboard, check for fire buttons, decrement the mouse scan delay, and send signals to processes needing them.

The Write routine passes all the characters onward to the Interface modules, but can be requested by them to read more than one parameter for escape codes.

The CLEAR key flip between windows is also caught during interrupts, which you can see by holding CLEAR down while doing disk access. Be careful, though - this causes my machine to crash.

Other than that, CC3IO really knows very little about windows.

CC3IO also handles these:

```
GETSTATS        SETSTATS
 SS.ComSt        SS.ComSt
 SS.Mouse        SS.Mouse
 SS.Montr        SS.Montr
 SS.KySns        SS.KeySns
 SS.Joy          SS.Tone
                 SS.GIP
                 SS.SSig
                 SS.MsSig
                 SS.Relea
                 SS.Open
```

GRFINT

GrfInt has six entry points, Init, Write, Getstt, Setstt, Term, and SetWindow. At offset 0026 begins the window escape code table, each entry made up of a parameter count, vector, and a code byte to be used for internal GrfDrv calls.

On initialization, GrfInt links or loads "grfdrv" or "../CMDS/grfdrv". GrfDrv MUST end up on an 8K block exact boundary, which is why it should be loaded off disk. GrfInt calls GrfDrv's Init routine and then unlinks it. This causes GrfDrv to be unmapped from the system task, which is okay as GrfDrv has already moved itself over to the second system map.

GrfInt moves a default palette into global memory where other modules may find it. This table is listed later.

GrfInt sets up the window entry tables, screen tables, and requests system memory for the graphics cursor tables.

As said before, it handles the task of getting all the parameters for the window display codes. It checks for a valid window destination. Parameters are collected and passed onto GrfDrv for execution.

Loading of Get/Put buffers is partially taken care of here, too. GrfInt reads in up to 72 bytes at a time into a global buffer for GrfDrv to read from.

It also sets the page length according the window size, does most of the window Select routine, and computes relative coordinates.

GRFINT also handles these:

```
GETSTATS        SETSTATS
  SS.ScSiz        SS.Open
  SS.Palett       SS.MpGPB
  SS.FBRegs       SS.DfPal
  SS.DfPal
```

GRFDRV

After being loaded by GrfInt or WindInt, GrfDrv is called to initialize itself. It sets up the second task map (Task One, which is reserved, as is task zero, for the system use) to contain itself, global system memory, and areas for swapping in buffers and screens to access. This map looks like:

```
 Logical
Block Addrss     Use
----- ---------  ---------------------
 0    0000-1FFF  System Global Memory
 1    2000-3FFF  Buffers mapped in here
 2    4000-5FFF  -
 3    6000-7FFF  Grfdrv
 4    8000-9FFF  Screens mapped in here
 5    A000-BFFF    "        "
 6    C000-DFFF    "        "
 7    E000-FDFF    "        "
```

To get to GrfDrv, GrfInt sets up a new stack with GrfDrv's entry point as the PC, then jumps via direct page vector 00AB to OS9p1. OS9p1 copies the reserved Task One DAT Image into the GIME's second DAT set, flips over to the GrfDrv map, and does a RTI.

Returning to the normal system map (back to GrfInt) is just the opposite, except the vector at 00A9 is used to flip back to the always set up Task Zero system map.

Interrupts are still enabled on the GrfDrv map, and OS9 saves which system map (0 or 1) it was in when the interrupt occurred. After servicing the interrupt, OS9 resets the DAT to the correct task number.

GrfDrv handles all character writing (text or graphics) and graphics routines (line, point, etc).

It checks for window collisions, sets the GIME, translates colors, handles buffers, and executes terminal codes such as CLS, INSLINE, etc.

Allocation and release of buffer and video memory is also done within GrfDrv.

SCREEN MEMORY

Screen memory is allocated using F$AlHRAM (from high block numbers at the top of memory), because the GIME requires contiguous physical memory for display, and there's a better chance of finding such up there. The OS9 kernal gets program and data blocks from the lower end.

Actually, it really shouldn't matter all that much where you found contiguous RAM, but perhaps they felt it was safer up high. Since we have no ROM blocks to map into DAT Images as a safe area (for blocks not used in a program map), the DAT.Free marker used by the CoCo (333E) means that a video page (3E) is all that should get clobbered if a bad program runs amuck through it's logical address space. (That is, unless it should run into the GIME and I/O page at XFFXX!)

Each new window doesn't necessarily take up a lot more memory. If you open a window on a previously allocated screen, it's still going to use that screen memory. It's inside that screen, and so is also inside that memory block or blocks.

Graphics screens are allocated by blocks, since the smallest form uses 16K or two blocks. When all the windows on a screen are closed, all the blocks are returned to free memory.

Text screens are allocated a block at a time, and that block is divided up into at least two screens, if they are both 80 column (4K each) screens. So you can have two 80's, one 80 and two 40's, or four 40's per 8K RAM block. That is, you can if you apply the patch to GrfDrv that's in the BUGS section of this manual. See it for more details.

Obviously, it makes more sense, memory-wise, to use text screens where feasible.

MISC WINDOW TIPS

The keyboard mouse toggled on and off by <CTRL-CLEAR> changes the arrow keys into a hires joystick, and the function keys into fire buttons. I believe that it takes over in place of the external right-hand joystick. In this mode, the arrow keys are set up as:

```
Arrow       - move 8 positions
Shift-arrow - move 1 position
Ctrl-arrow  - move to far edge
```

If you've set the proportional switch and are using the stdfonts character set, change the font to C8 02 for a better display.

Each device (TERM, Wx) has a 128 byte input queue. This means that you can go to an inactive window, type something blindly on it. Then if you started a program on that window, what you typed previously will be immediately read. For example, if you typed "dir" on W3, then went back and "shell <>>>/w3&", the dir command would be executed by the new shell.

In most cases, it might be better to use the Forgnd, Backgnd text color set commands, instead of the Palette command. There are eight colors already provided for, and except for two color graphics windows, should be easier to use and remember.

Want to see what your StdPtrs file looks like? Merge them into a window. Open a 320x192 graphics window for best results. Then "display 1B 4E 0100 0050" to move the graphics cursor to an open spot. Now you can "display 1B 39 CA p", where p=1-7 to see how the various pointers look.

AREAS OF INTEREST

For those who might wish to customize their system by changing some of the module defaults, and could use a quick reference to the tables used, here are some helpful assembly areas:

```
CC3IO

        *          Keyboard & Mouse Delay Init (1st device):

 007D 861E             lda    #30         1/2 second
 007F A78861           sta    $61,x       set keybd delay constant
 0082 A78829           sta    $29,x       and first delay
 0085 8603             lda    #$03        1/20 second
 0087 A78862           sta    $62,x       secondary delay

 008A 4A               deca               A=02
 008B A784             sta    ,x          ($1000)=02
 008D 6C883C           inc    $3C,x       mouse flag
 0090 8601             lda    #$01
 0092 A7883D           sta    $3D,x       right joystick
 0095 8678             lda    #120        2 seconds
 0097 A7883E           sta    $3E,x       set button timeout
 009A CCFFFF           ldd    #$FFFF
 009D ED8828           std    $28,x       init keyboard vars
 00A0 ED882B           std    $2B,x
 00A3 CC0078           ldd    #$0078      set ss.mouse for device
 00A6 EDC828           std    U0028,U     (scan rate & timeout)

--------

        *          Keyboard Mouse Coord Deltas:
        *                   Normal, Shift, Control

 00F4 0801             fcb    8,1         right
 00F6 027F             fdb    639
 00F8 F8FF             fcb    -8,-1       left
 00FA 0000             fdb    0
 00FC 0801             fcb    8,1         down
 00FE 00BF             fdb    191
 0100 F8FF             fcb    -8,-1       up
 0102 0000             fdb    0

--------

        *          Special Key Code Table:
        *                   Normal, Shift, Control

 05A2 406000           fcb    $40,$60,$00 @
 05A5 0C1C13           fcb    $0C,$1C,$13 up
 05A8 0A1A12           fcb    $0A,$1A,$12 down
 05AB 081810           fcb    $08,$18,$10 left
 05AE 091911           fcb    $09,$19,$11 right
 05B1 202020           fcb    $20,$20,$20 space
```

```
05B4 303081          fcb    $30,$30,$81 0      0      case
05B7 31217C          fcb    $31,$21,$7C 1      !
05BA 322200          fcb    $32,$22,$00 2      "
05BD 33237E          fcb    $33,$23,$7E 3      #
05C0 342400          fcb    $34,$24,$00 4      $
05C3 352500          fcb    $35,$25,$00 5      %
05C6 362600          fcb    $36,$26,$00 6      &
05C9 37275E          fcb    $37,$27,$5E 7      '
05CC 38285B          fcb    $38,$28,$5B 8      (      [
05CF 39295D          fcb    $39,$29,$5D 9      )      ]

05D2 3A2A00          fcb    $3A,$2A,$00 :      *
05D5 3B2B7F          fcb    $3B,$2B,$7F ;      +
05D8 2C3C7B          fcb    $2C,$3C,$7B ,      <
05DB 2D3D5F          fcb    $2D,$3D,$5F -      =      _
05DE 2E3E7D          fcb    $2E,$3E,$7D .      >
05E1 2F3F5C          fcb    $2F,$3F,$5C /      ?      \

05E4 0D0D0D          fcb    $0D,$0D,$0D enter
05E7 828384          fcb    $82,$83,$84 clear
05EA 05031B          fcb    $05,$03,$1B break
05ED 313335          fcb    $31,$33,$35 F1
05F0 323436          fcb    $32,$34,$36 F2
```

GRFINT

```
      *          Default Palette Table:
      * whi, blu, blk, grn, red, yel, pur, cyn

02F2 3F090012        FCB    $3F,$09,$00,$12,$24,$36,$2D,$1B
02FA 3F090012        FCB    $3F,$09,$00,$12,$24,$36,$2D,$1B
```

GRFDRV

```
L03C?   ldd #$C801        set default font for gfx windows
```

```
L08CC   equ    *          Translate Color For RGB:
        pshs   x
        tst    >x1009     check monitor type
                           (0=comp color, 1=RGB, 2=mono)
        bne    L08D9      ..skip if not composite color
        leax   >L08DB,pcr translation table
        ldb    b,x        get new gime palette byte
L08D9   equ    *
        puls   pc,x       rts.
```

```
L08DB    equ    *         64 Color Translation Table:

         FCB    $00,$0C,$02,$0E,$07,$09,$05,$10
         FCB    $1C,$2C,$0D,$1D,$0B,$1B,$0A,$2B
         FCB    $22,$11,$12,$21,$03,$01,$13,$32
         FCB    $1E,$2D,$1F,$2E,$0F,$3C,$2F,$3D
         FCB    $17,$08,$15,$06,$27,$16,$26,$36
         FCB    $19,$2A,$1A,$3A,$18,$29,$28,$38
         FCB    $14,$04,$23,$33,$25,$35,$24,$34
         FCB    $20,$3B,$31,$3E,$37,$39,$3F,$30
```

```
  * System and CC3IO Memory Map (block 00)
    * Our personal disasm variable map from Rogue.
    * Kevin Darling     14 Feb 87, 30 Mar 87
    * Kent Meyers
    * Not necessarily accurate for latest versions.
    * ------------------------------------------
    * Global and CC3IO Memory Starts at $01000:

1000          rmb 1
1001          rmb 1
1002          rmb 1  map side (grfdrv)
1007          rmb 2  grfdrv stack pointer
1009          rmb 1  monitor type (0,1,2)
100A          rmb 1  same as active dev flag
100B          rmb 1  v.type of this dev
100C          rmb 2  device static memory ptr
100E          rmb 1  WindInt map flag?

100F          rmb 6  F$Alarm time packet
1015          rmb 1  F$Alarm process id
1016          rmb 1  F$Alarm signal code
1017          rmb 2  terminal bell vector
1019          rmb 2  ptr to default palette ptr
101B          rmb 1  tone duration in ticks
101C          rmb 1  bell flag
101D          rmb 3

1020          rmb 2  active window devmem
1023          rmb 1  screen changed flag
1024          rmb 1  $80=grf/windint,$02=vdg found

1025          rmb 2
1027          rmb 1  last keybd row fnd
1028          rmb 1
1029          rmb 1  repeat delay cnt now
102A          rmb 5

102F          rmb 1  grfdrv init'd flag
1030          rmb 1  SHIFT key down
1031          rmb 1  CTRL  key down
1032          rmb 1
1033          rmb 1  ALT   key down
1034          rmb 1  keysns byte
1035          rmb 1  same  key flag
1036          rmb 1  SHIFT/CLEAR flg
1037          rmb 1
1038          rmb 1  grfdrv init'd flag
1039          rmb 2
103B          rmb 1  mouse sample tick counter

    * ------------------------------------------
    * Mouse Packet: ($20 bytes)
```

```
103C        rmb 1                    00
103D        rmb 1  fire bit#,rdflg 01
                   bit 0=fire button #
                   bit 1=side (0=right,1=left)
                   bit 6=set if was keybd mouse
103E        rmb 1  timeout constant02
103F        rmb 1  keybd flag        03
1040        rmb 1                    04
1041        rmb 1  cntr              05
1042        rmb 2  0-FFFF cnt        06
1044        rmb 1  fire chg bit      08
1045        rmb 1  fire chg bit      09
1046        rmb 1  up time           0A
1047        rmb 1  up time           0B
1048        rmb 1  chg counter       0C
1049        rmb 1  chg counter       0D
104A        rmb 1  down time         0E
104B        rmb 1  down time         0F
104C        rmb 2                    10
104E        rmb 2  returned X        12
1050        rmb 2  returned Y        14
1052        rmb 1                    16
1053        rmb 1  0=old,1=hires     17
1054        rmb 2  X coordinate      18
1056        rmb 2  Y coordinate      1A
1058        rmb 2  X window          1C
105A        rmb 2  Y window          1E

    * --------------------------------------------
1060        rmb 1  mouse sample rate
1061        rmb 1  first key delay ticks
1062        rmb 1  secondary repeat ticks
1063        rmb 1  enable kbdmouse toggle flag
1064        rmb 1  one shot ignore CLEAR key flag
1065        rmb 1  fire button dwn (F1=01 F2=04)
1066        rmb 1  mouse to use (AND 66+67<>0:update packet)
1067        rmb 1  mouse coord changed flag
1068        rmb 6  comodule entry vectors...
106A        rmb    vdgint entry
106E        rmb    grfdrv entry
1070        rmb 1  move data cntr for buffers
1071        rmb 4  32 bit window alloc map
1075        rmb 2  ptr to 576 byte gfx tables

10BF        rmb 1  cc3io L0116 flag (chg mouse?)
10C2        rmb 2  cc3io shift-clear key sub (L0614)
10C4        rmb 2  cc3io set mouse sub (L06AE)
10C6        rmb 1  fire not read: zero if ssig sent
10C7        rmb 16 palette reg data (sys default)
10E7        rmb

1100        rmb x  grfdrv variables
1200        rmb x  data buffer for gpload
1280        rmb x  window tables ($40 each)
1290               window table base offset used
1A80        rmb x  screen tables
```

```
* ---------------------------------------------
* GrfInt/GrfDrv Vars:

grfdrv   equ $0100 use for global offset

110E        rmb 1  char bsw bits

1120        rmb 2  ellipse parms:
1122        rmb 2  .
1124        rmb 2  .
1126        rmb 2  .

112E        rmb 2  windentry now
1130        rmb 2  screen table now
1132        rmb 3  3 byte buffer table
1135        rmb 3  grp,offset
1138        rmb 3  grp,offset returned (new)
113B        rmb 2  end of vars ptr?
113D

1147        rmb 2  HBX,LBX
1149        rmb 2  HBY,LBY
114B        rmb 2  current X
114D        rmb 2  current Y
114F        rmb 2  HSX,LSX
1151        rmb 2  HSY,LSY
1153        rmb 2  Circle, ellipse, arc
1155        rmb 2  Ellipse, arc
1157        rmb 1  GRP
1158        rmb 1  BFN
1159        rmb 1  SVS
115A        rmb 1  PRN
115B        rmb 2  BX putgc
115D        rmb 2  BY putgc
115F        rmb 1
1160        rmb 1  STY marker
1161        rmb 1  fore rgb data WE:06
1162        rmb 1  back rgb data WE:07
1163        rmb 1  bytes/row     SC:04
1164        rmb 2  lset vector?  WE:16
1166        rmb 2  Pset offset   WE:0F
1168        rmb 2  grfdrv lset   WE:14
116A        rmb 2  max x coord   WE:1B
116C        rmb 2  max y coord   WE:1D
116E        rmb 2  X pixel cnt
1170        rmb 2  Y pixel cnt
1172        rmb 2  get/put ow save screen strt

117D        rmb 1  buffer block # (get block)
117E        rmb 2  buffer offset grp/bfn
1180        rmb 2  HBL,LBL
1182        rmb 2  3 byte extended screen address
1185        rmb 2  temp

1187        rmb 16 grfdrv (sysmap 1) DAT Image
1197        rmb 1  temp
1199        rmb 2  this windentry ptr
```

```
119B          rmb 1  counter temp
119C          rmb 1
119D          rmb 2  offset to buffer in block

1280          rmb x  windentries: base=1290

    * -----------------------------------------
    * Window Entry: ($40 each)

              org -$10
-10 W.        rmb 2  screen table ptr
-0E           rmb 1  back wind# link
-0D           rmb 2  screen logical start
-0B           rmb 2  CPX, CPY
-09           rmb 2  SZX, SZY
-07           rmb 2  x,y sizes?
-05           rmb 2  cursor address
-03           rmb 1
-02           rmb 1
-01           rmb 1
 00           rmb 1  sty marker byte
 01           rmb 1
 02           rmb 1  X byte cnt (cwarea)
 03           rmb 1  cwarea temp
 04           rmb 2  bytes/row
 06           rmb 2  fore/back prn
 08           rmb 1  def attr byte FUTTTBBB
 09           rmb 1  char bsw bits: (default=$89)
                      80 TChr
                      40 Under
                      20 Bold
                      10 Prop
                      08 Scale
                      04 Invers
                      02 NoCurs
                      01 Protect
 0A           rmb 1  LSET #
 0B           rmb 1  GRP for font
 0C           rmb 2  font offset
 0E           rmb 1  GRP for PSET
 0F           rmb 2  pset offset?
 10           rmb 1  LCD mode
 11           rmb 1  overlay grp
 12           rmb 2  overlay offset
 14           rmb 2  ptr to grfdrv LSET table
 16           rmb 2  vector (1FDE/1FF4)
 18           rmb 1  gcursor BFN
 19           rmb 2  gcursor offset
 1B           rmb 2  max X coord (0-79,0-639)
 1D           rmb 2  max Y coord (etc txt/gfx)
 1F           rmb 2  BLength
 21           rmb 3  grp/offset for next gpload
 24           rmb 2  screen logical start default
 26           rmb 2  cpx,cpy defaults
 28           rmb 2  szx,szy
 2A           rmb 6  reserved
```

```
    * -------------------------------------------
    * Screen Table: ($20 each)

00  S.      rmb 1  sty marker
01          rmb 1  first block # (used flag)
02          rmb 2  screen logical start
04          rmb 1  bytes/row
05          rmb 1  border  prn
06          rmb 1  foregnd prn (software border)
07          rmb 1  backgnd prn
08          rmb 8
10          rmb 16 palette regs (00RGBRGB)

    * -------------------------------------------
    * Gfx Table (32 of 18 bytes each) pt'd to by .75-6

00          rmb 1
01          rmb 2  BX of graphics cursor
03          rmb 2  BY
05          rmb 13

    * -------------------------------------------
    * Internal Screen TYpe marker byte:
    * User  STY =>Mark ...
            FF   FF   current screen
            00   FF   current screen
            05   01   640 two color
            06   02   320 four
            07   03   640 four
            08   04   320 sixteen
            02   85   80 col
            01   86   40 col

    * -------------------------------------------
    * Device Memory:

            rmb V.SCF
1D  V.      rmb 1 0=window,2=vdg,4=?? ,6=grfdrv
1E          rmb 1
1F          rmb 2 parity, baud (also char temp)
21          rmb 1 case flag
22          rmb 1 keysns enable
23          rmb 1 screen change flag
24          rmb 2 keybd ssig id,signal
26          rmb 2 mouse ssig id,signal
   *              SS.Mouse (X):
28          rmb 1 init'd to $00 mouse sample rate
29          rmb 1 init'd to $78 mouse fire timeout
   *              SS.Mouse (Y):
2A          rmb 1 mouse to use
2B          rmb 1  ""
2C          rmb 1 parm cnt
2D          rmb 2 parm vector
2F          rmb 2 ptr to parms start
31          rmb 2 ptr to next parm storage
33          rmb 1 last char read buff offset
34          rmb 1 next char read
```

```
35          rmb 1 window entry number
36          rmb 1 dwnum from descriptor
37          rmb 1 internal comod call number
38          rmb x parm storage
51          rmb x
80          rmb $80 read buffer

    * -------------------------------------------
    * Device Descriptor:

2C  DXSiz   rmb 1 SZX
2D  DYSiz   rmb 1 SZY
2E  DWNum   rmb 1 window number
2F  DWIni   rmb 1 0=no defaults, 1=use defaults
30  DSTyp   rmb 1 STY
31  DXPos   rmb 1 CPX
32  DYPos   rmb 1 CPY
33  DFCol   rmb 1 Foregnd PRN
34  DBCol   rmb 1 Backgnd PRN
35  DBord   rmb 1 Border  PRN

    * -------------------------------------------
    * Get/Put Buffer Header ($20 each?):

00  B.Block  rmb 1  block link
01  B.Offset rmb 2  offset in block
03  B.Grp    rmb 1  group number
04  B.Bfn    rmb 1  buffer number
05  B.Len    rmb 2  BL length
07  B.XDots  rmb 2  # x dots in char
09  B.YDots  rmb 2  # y dots in char
0B  B.RowsC  rmb 1  # rows in char
0C           rmb 1
0D           rmb 1
0E  B.STyp   rmb 1  sty marker byte
0F  B.BlkSiz rmb 1  number of blocks
10           rmb $10 reserved
20           ...    data

    * -------------------------------------------
    * Internal GrfDrv Call Numbers (from Grfint)
```

#	What	Escape	#	What	Escape
00	Init		2C	DEFGB	29
02	Terminate		2E	KILLBUF	2A
04	DWSET	20	30	GPLOAD	2B
06	DWPROTSW	36	32	Move buffer	
08	DWEND	24	34	GETBLK	2C
0A	OWSET	22	36	PUTBLK	2D
0C	OWEND	23	38	Map GP Buffer	
0E	CWAREA		3A	Alpha put	
10	SELECT	21	3C	Control codes	
12	PSET	2E	3E	05 xx cursor calls	
14	BORDER	34	40	1F codes	
16	PALET	31	42	Goto xy	
18	FONT	3A	44	PUTGC	4E
1A	GCSET	39	46	Set Window	

1C	DEFCOLR	30	48	POINT	42,43
1E	LSET	2F	4A	LINE	44-47
20	FCOLOR	32	4C	BOX	48,49
22	BCOLOR	33	4E	BAR	4A,4B
24	TCHRSW	3C	50	CIRCLE	50
26	PROPSW	3F	52	ELLIPS	51
28	SCALE	35	54	ARC	52,53
2A	BOLD	3D	56	FFILL	4F

* ---

CHARACTER FONTS -

by Chris Babcock

Each font has a maximum size of $400 bytes.

The first $100 bytes are broken up and scattered around in the area $80 to $FF.

The next $300 bytes contain the definitions for the area $20 to $7F.

Each character is represented by 8 bytes. If the bit is 1 the pixel will be set and if it is 0 the pixel will not be set (as you would expect.) The graphic mode is always interpreted as mode five for the fonts.

The font color is the foreground palette. This means the font can not be more than two colors, the foreground palette and the background palette for the on/off conditions of the bits.

A font always uses exactly 8 scan lines per character row. The number of pixels across per character can be either 6 or 8. Using a size of six allows up to 53 characters across in 40 column graphic windows and 106 in 80 column graphic windows. Eight pixels allow 40 or 80 in the corresponding graphic windows.

The following is the breakup of the file:

```
Position in file        Character codes represented
 ----------------        -----------------------------------
 $0000 - $00CF           $C1-$DA and $E1-$FA stored here
 $00D0 - $00FF           $AA-$AF and $BA-$BF stored here
 $0100 - $03FF           $20-$7F stored in this area

 $0170 - $0177 ($2E)      $A0-$A9 $B0-$B9 $C0 $DB $E0 $FB-$FF
   Note: All the above reference $2E ('.')
```

Proportional spacing uses a different method of putting characters on the screen. The 8 bytes are checked to find the range of bits used. Then a blank bit is added to the range at the end. This range is used as the character. The driver is not smart enough to do a proper backspace; it always uses a backspace of the number of pixels selected when the buffer was loaded. A text graphic example of this is below using the word "Mistake."

Normal:

```
.        .        .        .        .        .        .        .
7654321076543210765432107654321076543210765432107654321076543210
 X   X                                  X
 XX XX                    X             X
 X X X     X     XXXXX  XXXXX    XXX    X  X     XXXX
 X X X          X         X     X   X   X X     X    X
 X   X     X     XXXX     X     X   X   XX      XXXXXX
 X   X     X         X    X X   X   X   X X     X
 X   X     X    XXXXX      X     XXX X  X  X     XXXX
.        .        .        .        .        .        .        .
```

Proportional:

```
.        .        .        .        .        .        .        .
7654321076543210765432107654321076543210765432107654321076543210
X   X                       X
XX XX            X          X
X X X X  XXXXX XXXXX  XXX   X  X  XXXX
X X X   X        X   X   X  X X  X    X
X   X X  XXXX    X   X   X  XX   XXXXXX
X   X X      X   X X X   X  X X  X
X   X X XXXXX     X   XXX X X  X  XXXX
.        .        .        .        .        .        .        .
```

The transparent character option causes only the set bits to be placed on the screen. Bits already set are not removed from the screen as they would be without this option selected. Using this mode allows the text to overlay graphics on the screen without erasing the character block area.

If moving the cursor, change to fonts you're going to use before moving, otherwise the cursor ends up one line down. Unless you're going from 6-6 or 8-8, then okay.

Note that fonts don't have to be real text. You could for example, set up a font of small objects. The ROGUE game uses special fonts to represent people, gold, trapdoors, etc.

```
                              nam    Window Descriptor - CC3 LII
                              ttl    INSIDE OS9 LEVEL II
        * SRC for /W1-W9
        * roll your own descriptors
        * 1st version -24Jan87
        * Copyright 1987 by Kevin Darling

        * -------------------------------------------
        * Change these to make a new /Wx descriptor:
        * (only "window" need really be changed)
        * For Window numbers great than 9, you must
        *  manually set the dnam at the end.
        * The following is just a sample...

0001             window   set    1          the window number
0000             cpx      set    0          begin col
0000             cpy      set    0          being row
001B             cols     set    27         number cols
000B             rows     set    11         number rows
0001             mode     set    1          (1=40 col text, 2=80 col text,
0002             fore     set    2          foregnd and cursor palettes
0000             back     set    0          backgnd palette
0004             bord     set    4          border palette

        * cols should be <= the mode maximum.
        * fore+8 is actual foregnd palette, fore is cursor.

        * -------------------------------------------
00F0             devic    equ    $F0        quicker than defsfile
0001             objct    equ    $01
0080             reent    equ    $80
0001             READ.    equ    $01
0002             WRIT.    equ    $02
0000             DT.SCF   equ    0

0000 87CD0044             mod    len,dnam,devic+objct,reent+1,mgr,drv

000D 03                   fcb    READ.+WRIT. device mode
000E 07                   fcb    $07
000F FFA1                 fdb    $FFA0+window port address
0011 1A                   fcb    opts-*-1   option byte count
0012 00                   fcb    DT.SCF     device type

0013 00                   fcb    0          case=upper and lower
0014 01                   fcb    1          backspace mode
0015 00                   fcb    0          delete mode
0016 01                   fcb    1          echo on
0017 01                   fcb    1          auto line feed on
0018 00                   fcb    0          no nulls after cr
0019 00                   fcb    0          no pause
001A 18                   fcb    24         lines per page default (MW)
001B 08                   fcb    $08        backspace char
001C 18                   fcb    $18        delete line char
001D 0D                   fcb    $0D        end of record char
001E 1B                   fcb    $1B        end of file char
001F 04                   fcb    $04        reprint line char
```

```
0020 01                 fcb   $01         dup last line char
0021 17                 fcb   $17         pause char
0022 03                 fcb   $03         interrupt character
0023 05                 fcb   $05         quit character
0024 08                 fcb   $08         backspace echo char
0025 07                 fcb   $07         line overflow char
0026 80                 fcb   $80         type=window
0027 00                 fcb   $00         baud
0028 0036               fdb   dnam        echo device
002A 00                 fcb   $00         xon character
002B 00                 fcb   $00         xoff character
002C           opts     equ   *           End of Path Desc Options

002C 1B                 fcb   cols
002D 0B                 fcb   rows
002E 01                 fcb   window      window #
002F 01                 fcb   1           use defaults option
0030 01                 fcb   mode
0031 00                 fcb   cpx
0032 00                 fcb   cpy
0033 02                 fcb   fore        forgnd and cursor palette
0034 00                 fcb   back        backgnd palette
0035 04                 fcb   bord        border palette

0036 576E      dnam     fcc   "Wn"
0038 B1                 fcb   $B0+window
0039 5343C6    mgr      fcs   "SCF"       file manager
003C 43433349  drv      fcs   "CC3IO"     driver

0041 3EF9CA             emod
0044           len      equ   *
                        end

00000 error(s)
00000 warning(s)
$0044 00068 program bytes generated
$0000 00000 data bytes allocated
$0160 00352 bytes used for symbols

0000 S BACK     0004 S BORD     001B S COLS     0000 S CPX      0000 S CPY
00F0 E DEVIC    0036 L DNAM     003C L DRV      0000 E DT.SCF   0002 S FORE
0044 E LEN      0039 L MGR      0001 S MODE     0001 E OBJCT    002C E OPTS
0001 E READ.    0080 E REENT    000B S ROWS     0001 S WINDOW   0002 E WRIT.
```

===

These are the Tandy-supplied options:
(in same order as descriptor)

OPTION	W	W1	W2	W3	W4	W5	W6	W7
cols	00	1B	0C	28	3C	13	50	50
rows	00	0B	0B	0C	0B	0B	0C	18
wind#	FF	01	02	03	04	05	06	07
deflt	00	01	01	01	01	01	01	01
mode	00	01	FF	FF	02	FF	FF	02
cpx	00	00	1C	00	00	3D	00	00
cpy	00	00	00	0C	00	00	0C	00
fore	00	02	00	02	00	02	02	00
back	00	00	01	07	01	07	00	01
bord	00	04	01	01	04	04	04	01

Note that a descriptor with TYPE=1 is a VDG window instead of these (TYPE=80).

INSIDE OS9 LEVEL II

Miscellaneous

SHELL

INFORMATION

CoCo-3 Level Two has a new shell, derived from the original that was used before for both L-I and L-II systems. The changes made were done mostly because of windows and our 8K blocks.

To the user, there are four main new features:

. The ability to redirect multiple paths to the same file, using the <>, <>>, <>>>, >>> options.

. The usage of a path number as a device reference: that is, you can redirect a command's standard input, output or error to the current in/out/err paths. To do this, you use the pseudo device names "/0, /1, or /2".

The main use that you'll see of this is inside shell script files. An example should be in your Startup file, where you'll find "setime </1" instead of "setime </term" like you're used to seeing. Since path 1 (standard output) is still the device that you're viewing, the effect is the same, but now the same Setime script will also work with say, an external terminal. This feature gives you more flexiblility and less hard-coding of device names.

. The "i=/devicename" option. This is known as the immortal option. What it does is open all three standard paths to the device named, and sets a flag in the shell's data area.

The flag indicates that the shell should not end operations on an End-of-File. This is needed because CC3GO would have no idea where to restart a shell, unlike the older SysGo which could pretty well assume /TERM.

This also provides a quick and dirty tsmon-like way to use an external terminal without it dying on you. Just use something like "shell i=/T2 &" to keep a shell on /T2. You could also have done "shell <>>>/t2", but that one could die on an EOF.

A related new feature is that if a new shell starts up but gets back an error printing "Shell", then it does die. This might happen if you start a shell and the open-window call fails. The reason is to keep from having phantom shells laying about with no paths open... they'd be impossible to kill.

. The ability to send special shell characters as parameters. Before, if you tried an: echo hello!   , the shell would send 'hello' (without the quotes) to echo, but then take the '!' and try to pipe to the next command, which wasn't there of course.

Now, you can type: echo "hello!"   , and what echo gets and prints out is: "hello!", but including the quote marks, unfortunately.

A SMALL PROBLEM

As seen in the flowchart, if the shell can't find a program in memory, it tries reading it's header from the current execution directory. If that fails, it tries to use a file from the data directory as a shell script for a new shell.

The older shells would first F$Link a module into it's own map to get the header information needed for a F$Fork of the new process. Unfortunately, with our 8K blocks, it's possible that this link might fail because the new program was too large to fit in the blocks left in a shell's map (normally 5 under ver 2.00.01).

The new L-II shell uses two new OS9 system calls to get around this: F$NMLink and F$NMLoad, both of which do NOT link a module into the caller's map, but instead just return some information from the module's header (like Data Size).

To keep the module link count straight, the shell also does an F$UnLoad, which uses a module's NAME to call unlink.

This is fine. A minor problem can occur, though, if the name of the module that shell wants to unload differs from the module's real name. This can happen if, for example, you had the Ident command on your disk under the filename "Id". What would happen is that when you typed "id", the shell would end up F$NMLoad'ing Ident from your commands directory and executing it. This is normal. But then shell would try to Unload "id", as that's the name it saved from the command line.

The net effect is that Ident would stay linked in the module directory until you manually unlinked it.

Another way this could occur is if you used a partial or full pathname. Examples: "/d1/cmds/bob" or " ../bob". In neither case will the F$Unload call work since those "names" do not match any in memory.

As I said, this is minor, and the shell can be rewritten someday to also read in the real name after it reads the header from disk. I suspect a later version will have this. The point is that you should be aware of this and so not be surprised.

KILLING WINDOW PROCESSES

While we're on the shell, I want to bring up another "gotcha" that makes perfect OS9 sense, but that still took a while to figure out.

Let's say that you began with a shell on TERM. Then you started one on W2 with "shell i=/w1&" and you went over to that one. Now you start another one with "shell i=/w7&" and then moved back to the original TERM window.

There let's say that you kill the shell on W7. You do a Procs and that shell continues to show up with an error 228.

The "gotcha" is that the shell on W1 was the parent of the dead W7 shell, and until you go to W1 and hit a key, the dead shell can't get thru to W1 to report it's death.

A similar thing can bite you worse. If you had started a process on W7 using the same method and it dies while you're doing something important (like editing a file) on the parent's window (W1), then you'll be confused by the death message popping up in the middle of your session.

Now this quirk has been around OS9 forever, but unless you used a lot of terminals, it didn't matter too much. With many windows now, it becomes more important and aggravating.

The partial solution that I use is to always start all my shells on other windows from my first window. That way, I at least know where their deaths will show up (-005 etc). This would go for any program I wanted to run in the bacground mostly unseen (using "&").

Typing "w" <enter> on the parent shell's window after killing a child is another good idea, as that causes that shell to Wait for the death report without messing up your screen.

MISC

Just wanted to add a couple of things about the shell that don't seem to be well-documented.

Many people falsely assume that "OS9" recognizes that a module is, say, a Basic09 packed I-Code procedure and so "OS9" calls up RUNB to execute it. The truth is that this is all done by the Shell. Trying to fork an I-Code module from a machine language program would fail unless you yourself specified the module as a parameter to RUNB and forked RUNB.

The other small point is that using parenthesis starts a sub-shell. For example, the command " (((echo hi; sleep 500))) " would cause 3 sub-shells to be formed, each calling the next. Try this sometime with a Procs command running on another window so you can see all the shells formed.

```
===============================================================
 SHELL                    Flowchart                       SHELL
===============================================================
        l                                      Data Area:
    Clear vars                                redirected pths
    Set signal intercept                      #pages
    Store parm size                           pathname ptr
.<--y (parm size=0?)                          parm size
l    Gosub DOCMD                              parm ptr
l    (end of parms?) y------->END             mem for mod
l------->l                                    this char
    Print 'Shell'                             '(' count
.-------->l                                   signal storage
l    Print 'OS9'                              P flag
l    I$Readline                               T
l    (end of file?) y-------->END             X
l        l<-------------.                     Setpr ID #
l    (-X flag?)         l                     19 byte buff
l<-- Print err msg      l                     input buffer
l    (-T flag?)         l
l    Echo in to #2      l
l    Gosub DOCMD        l
l<--n ( error?) y------>l

---------------------------------------------------------------
    * DOCMD SUB *
        l
     Exec W,*,CHD,CHX,EX,KILL,X,P,T,SETPR,;
     Find ()'s
     Exec & , ! ; # < > >>
     Start Process
     Undo redirection
     Wait if required
        RTS

---------------------------------------------------------------
   * START PROCESS *
        l
     Link to name   err--------->.
     Unlink                      l
        l                    Open xfile  err-------->.
        l                    Read hdr                l
        l                    Close file              l
        l<-----------------------l                   l
.<--y (M/L code?)                            Cmd = 'Shell
l    Else find lang (Runb, PascalS)             < name'
l    Cmd=lang, parm=name                             l
l------->l<------------------------------------------l
     Link to cmd/language
     Load if necessary
     Set mem size
     F$fork
     F$sleep 1
     F$unload cmd name
        RTS
```

This section is not really needed any more, as L-II will be out by the time this gets published. However, for those those who are getting started with L-II by way of the Tandy game disk "Rogue" cat # 26-3297,

USING Rogue TO MAKE A SYSTEM DISK:

```
1- under L-I, format a disk.
2- os9gen that disk using the OS9boot file on Rogue *.
3- copy over CMDS dir with grfdrv and shell **.
4- drop back to RSDOS and copy over the L-II kernal with:

    5 REM Rogue in drive 0, new disk in drive 1.
   10 CLEAR 10000
   20 FOR SE = 1 TO 18
   30 DSKI$ 0,34,SE,A$,B$
   40 DSKO$ 1,34,SE,A$,B$
   50 NEXT SE
```

* LR Tech owners may include their driver and desc after copying the new "shell" file and "grfdrv" to it, OR after changing the desc name from "H0" to something else so that the bootup gets shell/grfdrv from the floppy. Then CHX /H0/CMDS.

You should also change the H0 desc byte at $0E from $FF to $07 and reverify that module. That's the extended device address.

** You may include other utilities merged into the Rogue shell file (do an ident on it first!), to be included at startup. The total length of your shell file should be under $1E00 long.

You MUST have Grfdrv and Shell in your CMDS dir. They must also have the "e" attribute set on the files.

Since L-II will map in the entire block of cmds loaded in a file, you should try to keep things on an n*8K+(8K-512) boundary.

Your L-I mfree, mdir, and procs will NOT work.

PRINTER will work if you change the baud rate to 1/2 before.

One other thing: do NOT unlink Shell in memory. Crash-o!

MAKING WINDOWS:

Examples are also in Rogue's MAKE40, MAKE80, MAKEGW shell files.

However, because Rogue does not include the W, and W1-W7 device descriptors, you cannot make more than one window or screen of windows with it. Solution: make a set of window descriptors using the source code elsewhere in this text.

Don't worry too much about the default size and palettes, you can send the escape codes to override them anyway. Example:

```
 iniz w1 (if you have iniz cmd)
display 1b 20 2 0 0 30 c 9 0 1 >/w1
shell i=/w1 &
(now hit the CLEAR key: you should flip to that screen)
```

Read the Sept 86 RAINBOW article on windows, plus try out the later examples they give if you have 512K.

[]

Be aware that your CLEAR and @ keys are no longer the same as the CTRL and ALT keys!

BUGS - SOFTWARE

Level Two for the CoCo-3 has gone through many revisions, and most of the bugs have been ironed out over the months. What are left in version 2.00.01 are relatively minor. Not all are listed here. Check the electronic forums for recent updates.

MODULE: Clock
PROBLEM: Bad error code return.
SPECIFICS: Somebody left the '#' sign off of a LDB #E$error.

SOLUTION: Patch and reverify.

Offset	Old	New
0191	D6	C6

MODULE: IOMan
PROBLEM: Sorts queues wrong.
SPECIFICS: Change first made in L-I 2.0 to insert processes in I/O queues according to priority. Used wrong register.

SOLUTION: Patch and reverify.

Offset	Old	New
09A6	10	12
09A7	A3	E1

MODULE: GrfDrv
PROBLEM: Non-efficient use of screen memory.
SPECIFICS: Opening a 40 column screen should use the last 2K of an 8K screen block if it's free for use. However, apparently a bad Def was used in MW's source code and GrfDrv cannot match an internal code as a 40 column screen.

SOLUTION: Patch and reverify.

Offset	Old	New
033A	84	86

MODULE: IOMan
PROBLEM: Cannot have more than one VIRQ device at a time.
SPECIFICS: While Clock gets the size of the VIRQ table from the Init module (as it should), IOMan has a different size hard-coded in. Clock inserts the first entry at the front of the VIRQ table, but the next call starts searching at the end of the table...which turns out to usually be the header of the first module in your bootfile. Symptoms: if your disk drive is still going (waiting for motor time-out), you cannot Iniz a ModPak device. Or, if you Iniz a ModPak device, your drives will never shut off.

SOLUTION: Easiest patch is to the INIT Module, to change the number of IRQ/VIRQ devices down from 15 to say, 12.

Offset	Old	New
000C	0F	0C

MODULE: CC3IO
PROBLEM: SS.Montr getstat call bad.
SPECIFICS: Although the manual doesn't mention it, CC3IO also supports getting the current monitor type set by montype. The value (0,1,2) is returned in the X register. The code in CC3IO should have been a STD R$X instead of STB R$X though.

SOLUTION: Patch and reverify.

Offset	Old	New
07D2	E7	ED

BUGS - HARDWARE

The GIME chip itself, on many machines, has problems with map changes causing "snow" on the screen, horizontal scrolling difficulties, and a few other items.

The most basic problem is one of bus-timing, and a fix is expected soon from Tandy. This is all I can say right now.

The Speech/Sound Cartridge, because it uses the clock signals generated from the 6809E, is driven too fast at the 2MHz speed of L-II to operate correctly. This is also true of several third-party interfaces and ramdisk paks.

Information on hacking the SSC can be had on the electronic forums. Users of other gear should contact their suppliers for updates or patches to their hardware.

Many of us with the original Tandy floppy disk controllers have found that they simply cannot handle the 2Mhz speed. There are two things you can do about this.

You can try replacing the Floppy Disk Controller chip or data separator chips, and hope you bought a faster part than before. Or you can opt for one of the third-party controllers.

Both Disto and J&M controllers seem to work fine so far. The newer, the better, seems to be the rule of thumb.

As far as hard disk set-ups go, the ones at this time that I know will work at 2MHz is the LR Tech from Owlware, FHL's QT CoCo, and perhaps the J&M.

BUGS - MANUAL

At the last moment before this went to press, several people with Level Two called to ask about some mistakes in the manual. I won't point out the ones like misspellings, just the ones that might confuse you.

==

SUBJECT: Creating GFX Windows
SECTION: BASIC09 Reference
PAGE: 9-37

Here they tell you how to create a graphics window, but show the "merge sys/stdfonts >/w1" AFTER the wcreate. Nope. All you get is dots on the screen. You must merge stdfonts BEFORE opening any gfx windows, unless you care to do a FONT command to that window after merging. They had it correctly on the page before (9-35) about merging so that you can type later.

==

SUBJECT: F$FORK, F$LINK, F$LOAD, I$CREATE, I$MAKDIR, I$OPEN
SECTION: OS9 Tech Reference
PAGE: 8-16, 8-23, 8-26, 8-49, 8-56, 8-58

On all of these, after the call X should be pointing to the $0D (carriage return) at the end of the string.

==

SUBJECT: F$FORK
SECTION: OS9 Tech Reference
PAGE: 8-15

The Y register contains the parameter area size in BYTES, not in pages.

==

SUBJECT: F$TIME
SECTION: OS9 Tech Reference
PAGE: 8-40

To be exact, on exit X points to the time packet returned to the area at (X) that you had originally passed for the call.

==

SUBJECT: I$DELETE
SECTION: OS9 Tech Reference
PAGE: 8-50

On return, X should be pointing to the beginning of "MEMO".

==

SUBJECT: F$ALARM
SECTION: OS9 Tech Reference
PAGE: 8-66

F$Alarm is a user call, too. And they left out how to use it. Here's the info:

This call has several variations, which have to do with setting time variables that the Clock module will try to match once a second. You may clear the alarm setting, read it, or set it for one of two exclusive actions.

```
--------------------------------------------------
D = 0000 : clear the setting
--------------------------------------------------
X = ptr to 5-byte time packet (YYMMDDHHMM)
D = 0001 : cause the CC3IO "beep" for 16 seconds
   after the time packet sent matches system time.
--------------------------------------------------
X = ptr to spot for time packet return
D = 0002
X < current alarm setting packet returned
D < current proc id and signal pending
--------------------------------------------------
X = ptr to 5-byte time packet (YYMMDDHHMM)
A = proc id to signal on time match
B = signal to send on time match
```

==

SUBJECT: F$DATLOG
SECTION: OS9 Tech Reference
PAGE: 8-78

Actually, not a bad example, but only if you're running on a machine with 4K blocks. On the CoCo-3, Ouput X = $4329. The actual code just multiplies B*$2000 and adds it to X.

==

SUBJECT: SS.RDY
SECTION: OS9 Tech Reference
PAGE: 8-113

On devices that support it, the B register will return the number of characters that are ready to be read. Both CC3IO and ACIAPAK support this feature.

==

SUBJECT: SS.MOUSE
SECTION: OS9 Tech Reference
PAGE: 8-125 on

Somebody forgot the two reserved bytes between Pt.ToTm and Pt.TTTo. As printed, offsets after ToTm are wrong. So insert a "rmb 2 - reserved" after Pt.ToTm.
Also ignore the system use note at the end after Pt.Siz.

==
SUBJECT: SS.DSCRN
SECTION: OS9 Tech Reference
PAGE: 8-143

Also, if you specify screen number zero (Y=0000), then you will return to the normal VDG (32x16) screen. This should be done before a SS.FScrn if you wish to return to a text screen.

==

SUBJECT: INSIDE OS9 LEVEL II BOOK
SECTION: All
PAGES : Many

This is such a great book that the minor errors can be explained by the authors desire to get the information out to you quickly. You should send them lots of money and good wishes. By the way, this portion of the book is being written very close to April 1st.

PS The word 'them' in the second sentence should be changed to FHL.

PPS Remember it's real close to April 1st.

FONT CONVERSION

This is an RSDOS program from Chris Babcock that converts Graphicom-II font files to the format required by OS9. After conversion, you must copy the file over to an OS9 disk.

You must also specify the group/buffer numbers that you will later use to access the font using the FONT commands. We've been personally using group D0, and buffers 1-8 or so.

```
10 CLEAR 500,&H7B00:POKE&H95C9,&H17:POKE&HFF22,PEEK(&HFF22)OR&H10:CLS:PRINT"Graphi
com II Font to OS-9 Font  Copyright 1987 by Chris babcock - Program for Coco 3"
20DATA141,83,134,27,141,59,134,43,141,55,182,14,0,141,50,182,14,1,141,45,134,5,141
,41,204,0,8,141,46,141,44,204,4,0,141,39,79,16,142,1,0,141,22,49,63,38,250,142,124,
0,16,142,3,0,236,129,141,17,49,62,38
30 DATA 248,126,164,45,141,28,38,3,126,206,217,126,207,181,141,18,38,3,126,206,215,
126,207,179,141,8,38,3,126,201,86,126,202,4,52,2,182,193,66,129,48,53,130
40 FOR I=&HE04 TO &HE04+103:READ DT:POKE I,DT:NEXT
50 PRINT"What is the filename of the font(Maximum 8 Chars. Ext is
"+CHR$(34)+"SET"+CHR$(34)+")":PRINT"Use #:FILENAME if other drive."
60 LINEINPUT";";F$:PRINT@235,".SET"+CHR$(13):F$=LEFT$(F$,10)+".SET"
70 PRINT"New filename for the font       (Maximum 8 Chars. Ext is
"+CHR$(34)+"OS9"+CHR$(34)+")":PRINT"Do NOT enter a drive # now."
80 LINEINPUT":";G$:PRINT@393,".OS9":G$=LEFT$(G$,8):G$=G$+STRING$(8-
LEN(G$),32)+"OS9"
90 INPUT"Drive number for OS-9 file";D
100 LOADM F$
110 CLS:PRINT"Group number for the OS-9 Font  (Give in hexadecimal 00-
FF)":LINEINPUT":";GR$
120 GR=VAL("&H"+GR$):IF GR<0 OR GR>255 THEN 110
130 PRINT"Buffer/Font number (Hex also)":LINEINPUT":";BF$
140 BF=VAL("&H"+BF$):IF BF<0 OR BF>255 THEN PRINT@96,"";:GOTO 130
150 POKE&HEB,D:POKE&H95A,D
160 POKE&HE00,GR:POKE&HE01,BF
170 X=&H94C:FOR I=1 TO 11:POKE X,ASC(MID$(G$,I,1)):X=X+1:NEXT
I:POKE&H957,1:POKE&H958,0
180 PRINT"Saving..."
190 EXEC&HE04
200 CLS:PRINT"Use XCOPY or TRSCOPY to move thefile over to an OS-9 Level II   disk.
MERGE the file and type   DISPLAY 1B 3A GROUP BUFFER <cr>"
210 END
```

TIPS, GOTCHAS, and LAST MINUTE STUFF

Using L-I VDG Programs

Many of you may want to run programs such as TSEDIT or Steve Bjork's bouncing ball demo within a L-II screen. Fortunately, Microware provided for this. However, your disk only comes with one VDG-type descriptor, TERM-VDG.

For programs that don't have "/TERM" hard-coded in them, you can set up a window device as a VDG screen using the following method (where wX= any window number):

```
deiniz wX
xmode /wx type=1 pag=16
shell i=/wX &
```

This will give you another screen that you can flip to, where you can run TSEDIT or other older programs.

OS9Boots

Under L-I, many of us only loaded drivers and other modules as needed, to save memory. Level Two acts a bit differently, and your methods must change.

You should put ANY and ALL drivers and descriptors that you plan to use, IN your OS9Boot file. If you don't, then each time you load a separate driver, you will take up 8K of your 64K system map... doesn't take more than a couple to really limit the number of tasks or open files that you can have.

When using OS9Gen or Cobbler to make a new boot disk, be sure that you have a CMDS directory with a Shell file and the GrfDrv module. The execution attributes should also be set on these two files. Otherwise, you'll get the dreaded "OS9BOOT FAILED".

Merged Module Files:

If you ident your /D0/CMDS/shell, you'll see that more than one command is included in that file. The reason is that it pays to get as close to an 8K block boundary as possible, so that you use less memory. If you separately loaded each of those commands, each would take an 8K block. Even with 512K, you'd lose memory very quickly.

OS9 will try to fit a block of modules into the upper part of a 64K task map... but remember that the FEXX page and our I/O is from FE00-FFFF in all maps. So the ideal size of a merged file is:

(8K * N) - 512 bytes, where N ranges from 1-7)

Actually, N should be kept around 1, if possible. So a Shell file for instance, should ideally be just under $1E00 long. That's (8K * 1)-512 = $2000-$200 = $1E00.

RUNB is 12K, so it takes up 2 blocks, but you still have room for about 5K of things like syscall, inkey, gfx2, etc.

To create a new shell file, for example, you might do:

```
merge shell dir free mdir procs ... etc >newshell
rename shell shell.old; rename newshell shell
attr shell e pe
```

A "dir e " can tell you the size of merged files or you can print out an Ident of all your commands and use that as a reference to calculate from.

F$Load from system state:

Requires an extra parameter if done from a driver or other module that will be in the system map. The U register must point to the process descriptor of the process who's map you want the new module loaded into. Example for loading module file into the system space:

```
leax modnam,pc  point to name of module to load
 ldu D.SysPrc    get system proc desc pointer
 OS9 F$Load      load file "modnam" into system map
```

F$Link from system state:

Will put the module into the map of the current process (D.Proc). It also gets the name (X points to it) from the D.Proc map. So to link a module into system space, you must "trick" OS9:

```
ldd D.Proc      get current process
 pshs d          save it
 ldd D.SysPrc    get system proc desc
 std D.Proc      make it current proc temporarily
 ...             (set up link parms)
 OS9 F$Link      link module(s) into system map
 puls d          retrieve true user process
 std D.Proc      and reset as current process
```

Forking RUNB modules:

Pete Lyall and I just figured this one out, and even though it's fully explainable, it's still a gotcha...

Let's say that you have a Basic09 I-code (packed) module named "Bob", and it requires 10K of data area. Typing "bob" from the shell command line causes shell to check Bob's header. There it finds that Bob needs 10K and also needs RUNB. So the shell effectively does a "runb bob #10k". Fine.

But! If you have the need to fork "RUNB BOB" from within a m/l program and don't know what data size Bob (or any I-code module) needs, you'll probably try just using a F$Fork RUNB with Bob as a parameter - which will fail because RUNB's header only has a default data size required of 4K (possibly 8K for
CoCo-3). And 4K isn't enough for Runb to use Bob.

(note: just doing a "runb bob" from the shell cmd line would fail, too)

Moral is that you should either check an I-code's header yourself, or you could instead do a "F$Fork Shell bob" and let shell handle everything.

Using L-I Debug on Level Two:

There is no debug included on the L-II disk set. It will be on the Developer's Pak disk. In the meantime, if you can't use Modpatch for what you need to do, you can partially patch your current debug to at least let you modify modules in memory.

Debug will link to a module, but does so just to get the module address. It immediately unlinks the same module to keep the system link count correct. Under L-II, this means that the module is mapped into debug's space, then mapped out right after that.

As debug is now, you CAN use it on any modules that were in your bootfile, but that's because those cannot be unlinked. To debug other loaded modules, you have to change debug while under Level ONE:

```
Offset Old New
 06CC   10  12   this changes F$Unlinks to NOP's
 06CD   3F  12
 06CE   02  12

 06D0   10  12    "       "
 06D1   3F  12
 06D2   02  12
```

Then save it and reverify, of course. The only gotcha now is that since modules are not unlinked at all, then if you try debugging all sorts of modules at one time, you could get an error #207 from the debug map getting filled up. No problem, just Quit and enter Debug again.

Login II Patch

This patch will allow you to use your level I 'LOGIN' command (which currently crashes on a level II system) on a level II system. It corrects the code so that it uses the F$suser call instead of trying to manipulate the system's direct page, which is inaccessible under level II for writing (in USER mode). This patch is a joint effort of Kent Meyers and Pete Lyall.

```
display c
 t
 * LOGIN2.DBG - A patch script by Pete Lyall
 *
 * This is a shell procedure to use DEBUG to patch the LOGIN
 * command for use on a Level II OS9 system. Note: If you HAVE
 * NOT already patched your DEBUG command for use on a level II
 * system then either do THAT first, or run this script on a
 * LEVEL I system where DEBUG will work.
 *
 *
  -t
  tmode .1 -pause
  load login
  debug
  l login
```

```
. .+52
=49
=20
=32
l login
. .+57
=30
l login
. .+5a
=31
l login
. .+69
=49
=20
=32
l login
. .+6e
=30
l login
. .+71
=31
l login
. .+234
=1f
=02
=10
=3f
=1c
=12
l login
. .+49b
=66
=15
=73
q
save login.II login
display c
t
* The patch is completed.
*
* Now simply UNLINK LOGIN until it is out of memory
*
* The updated LOGIN command has been saved as 'login.ii' in
* the current directory.
*
* To use it, simply copy it to a LEVEL II disk's CMDS
* directory and rename it to 'login'. Also ensure that all
* the attributes are set properly for execution.
*
* Enjoy!
```

INSIDE OS9 LEVEL II

Sources

```
Microware OS-9 Assembler RS Version 01.00.00    03/30/87 00:15:04      Page 001
Alarm - INSIDE OS9 LEVEL II

                              nam   Alarm
                              ttl   INSIDE OS9 LEVEL II
                * alarm - test that sets alarm for next minute.
                * causes beep from coco sound output for 15 secs.
                * just for fun.
                * Copyright 1987 by Kevin Darling

   0006               F$Exit   equ   6
   0015               F$Time   equ   $15
   001E               F$Alarm  equ   $1E

   0054               D.Time   equ   $54
   0057               D.Min    equ   $57

   0000 87CD0026               mod   len,name,$11,$81,entry,msize

 D 0000               time     rmb   10
 D 000A                        rmb   200
 D 00D2               msize    equ   .

   000D 416C6172      name     fcs   "Alarm"
   0012 01                     fcb   1
   0013               entry
   0013 30C4                   leax  time,u
   0015 103F15                 OS9   F$Time
   0018 6C1D                   inc   D.Time-D.Min,x next minute (bad on 59)
   001A CC0001                 ldd   #$0001
   001D 103F1E                 OS9   F$Alarm     set alarm time
   0020 103F06                 OS9   F$Exit

   0023 A9F133                 emod
   0026               len      equ   *
                               end

00000 error(s)
00000 warning(s)
$0026 00038 program bytes generated
$00D2 00210 data bytes allocated
$00CA 00202 bytes used for symbols

0057 E D.MIN    0054 E D.TIME    0013 L ENTRY    001E E F$ALARM   0006 E F$EXIT
0015 E F$TIME   0026 E LEN       00D2 E MSIZE    000D L NAME      0000 D TIME
```

```
DMEM - dmem  <block> <offset> [<length>] ! dump
       dmem -<proc#> <offset> [<length>] ! dump
```

Dmem writes up to $1000 bytes to standard out, that it has copied **over** for you from other maps. If no length is given, it defaults to **256** ($0100) bytes. Examples using data above:

```
dmem  4 0 ! dump            : dumps first 256 bytes of GrfDrv
dmem  2 1CA 1AE ! dump      : dumps CC3Go
dmem  0 0 1000 >/d1/file    : file contains lower sysmem vars

dmem -3 0 20 ! dump         : dump first 32 shell data bytes
dmem -3 E000 5FA ! dump     : another way of dumping Shell
dmem -1 0 1000 >/d1/file    : file contains lower sysmem vars
```

Good use of PROC, PMAP, MDIR, and DMEM depends on the data you get from each. Open a graphics window and recheck the MMAP. Kill a Shell, and notice the status and signal codes. Look up the status bits in your old DEFS file, signal from Error codes. Watch how modules get mapped in using PMAP and MDIR.

Figure out system data use by knocking out the blocks you know are in other use, with PMAP and MMAP.

INSIDE OS9 LEVEL II

SOURCES

DMem

```
Microware OS-9 Assembler RS Version 01.00.00    03/30/87 00:15:20     Page 001
DMem - INSIDE OS9 LEVEL II

                                  nam    DMem
                                  ttl    INSIDE OS9 LEVEL II
          * DMEM - display block/mem offset
          * "dmem blk offset [len]! dump"
          * "dmem #id offset [len]! dump"

          * 08feb87 - change page offset to byte or id.
          * 22jan87 - version 1

          * Copyright 1987 by Kevin Darling

     0000 87CD0136                mod    len,name,$11,$81,entry,msize
     000D 444D65ED   name         fcs    "DMem"
     0011 02                      fcb    2

     0006            F$Exit       equ    $06
     0018            F$GPrDsc     equ    $18
     001B            F$CpyMem     equ    $1B
     008A            I$Write      equ    $8A
     008C            I$Writln     equ    $8C

     1000            buffsiz      set    $1000

   D 0000            acc          rmb    2
   D 0002            input        rmb    1
   D 0003            offset       rmb    2
   D 0005            dlen         rmb    2
   D 0007            id           rmb    1
   D 0008            prcdsc       rmb    512
   D 0208            buffer       rmb    buffsiz
   D 1208            stack        rmb    200
   D 12D0            msize        equ

     0048            dat          equ    prcdsc+$40

     0012            hexin
     0012 0F00                    clr    acc
     0014 0F01                    clr    acc+1
     0016            hex01
     0016 A680                    lda    ,x+
     0018 8120                    cmpa   #$20
     001A 272A                    beq    hexrts
     001C 810D                    cmpa   #$0D
     001E 2726                    beq    hexrts
     0020 8030                    suba   #$30
     0022 810A                    cmpa   #10
     0024 2504                    bcs    hex2       0-9
     0026 8407                    anda   #7         A-F
     0028 8B09                    adda   #9
```

```
     0028 8B09                    adda  #9
     002A            hex2
     002A 48                      asla
     002B 48                      asla
     002C 48                      asla
     002D 48                      asla
     002E 9702                    sta   input
     0030 DC00                    ldd   acc         get accumulator
     0032 0902                    rol   input
     0034 59                      rolb
     0035 49                      rola
     0036 0902                    rol   input
     0038 59                      rolb
     0039 49                      rola
     003A 0902                    rol   input
     003C 59                      rolb
     003D 49                      rola
     003E 0902                    rol   input
     0040 59                      rolb
     0041 49                      rola
     0042 DD00                    std   acc
     0044 20D0                    bra   hex01
     0046            hexrts
     0046 301F                    leax  -1,x
     0048 DC00                    ldd   acc
     004A 39                      rts

     004B            entry
     004B 1700DA                  lbsr  skipspc     skip leading
     004E 102700C7                lbeq  badnum      ..was <cr>
     0052 812D                    cmpa  #'-         else is it #id ?
     0054 2617                    bne   entry0      ..no

     0056 3001                    leax  1,x         yes, skip '-'
     0058 8DB8                    bsr   hexin       get id number
     005A 1F98                    tfr   b,a
     005C 3410                    pshs  x
     005E 30C90008                leax  >prcdsc,u
     0062 103F18                  OS9   F$GPrDsc    get that proc desc
W    0065 10250053                lbcs  error
     0069 3510                    puls  x
     006B 2006                    bra   entry1

     006D            entry0
     006D 8DA3                    bsr   hexin       get block #
     006F 0F48                    clr   dat         set in fake datimg
     0071 D749                    stb   dat+1

     0073            entry1
     0073 1700B2                  lbsr  skipspc     get offset
     0076 1027009F                lbeq  badnum
W    007A 17FF95                  lbsr  hexin
     007D DD03                    std   offset

```

```
  007F 1700A6               lbsr  skipspc    get possible length
  0082 270E                 beq   entry2
W 0084 17FF8B               lbsr  hexin
  0087 10831000             cmpd  #$1000
  008B 2308                 bls   entry3
  008D CC1000               ldd   #$1000
  0090 2003                 bra   entry3
  0092           entry2
  0092 CC0100               ldd   #$0100
  0095           entry3
  0095 DD05                 std   dlen

  0097 30C90048             leax  >dat,u
  009B 1F10                 tfr   x,d        D=dat image ptr
  009D 109E05               ldy   dlen       Y=count
  00A0 9E03                 ldx   offset     X=offset within dat image
  00A2 3440                 pshs  u
  00A4 33C90208             leau  buffer,u
  00A8 103F1B               OS9   F$CpyMem
  00AB 3540                 puls  u
  00AD 250D                 bcs   error

  00AF 109E05               ldy   dlen
  00B2 30C90208             leax  buffer,u   point within buffer
  00B6 8601                 lda   #1
  00B8 103F8A               OS9   I$Write
  00BB           bye
  00BB 5F                   clrb
  00BC           error
  00BC 103F06               OS9   F$Exit

  00BF           help
  00BF 5573653A             fcc   "Use: DMem <block> <offset> [<length>] |
  00EB 0A                   fcb   $0A
  00EC 206F723A             fcc   " or: DMem -<id>   <offset> [<length>] |
  0118 0D                   fcb   $0D
  005A           helplen    equ   *-help
  0119           badnum
  0119 308DFFA2             leax  help,pc
  011D 108E005A             ldy   #helplen
  0121 8602                 lda   #2
  0123 103F8C               OS9   I$Writln
  0126 2093                 bra   bye

  0128           skipspc
  0128 A680                 lda   ,x+
  012A 8120                 cmpa  #$20
  012C 27FA                 beq   skipspc
  012E 301F                 leax  -1,x
  0130 810D                 cmpa  #$0D
  0132 39                   rts
```

```
        0133 979412            emod
        0136           len     equ   *
                               end

$0136 00310 program bytes generated
$12D0 04816 data bytes allocated
$0223 00547 bytes used for symbols

0000 D ACC      0119 L BADNUM   0208 D BUFFER   1000 S BUFFSIZ  00BB L BYE
0048 E DAT      0005 D DLEN     004B L ENTRY    006D L ENTRY0   0073 L ENTRY1

0092 L ENTRY2   0095 L ENTRY3   00BC L ERROR    001B E F$CPYMEM 0006 E F$EXIT
0018 E F$GPRDSC 00BF L HELP     005A E HELPLEN  0016 L HEX01    002A L HEX2
0012 L HEXIN    0046 L HEXRTS   008A E I$WRITE  008C E I$WRITLN 0007 D ID
0002 D INPUT    0136 E LEN      12D0 E MSIZE    000D L NAME     0003 D OFFSET
0008 D PRCDSC   0128 L SKIPSPC  1208 D STACK
```

```
================================================================
MMAP - Show memory block map, display mfree.
        U = used, M = loaded module, . = no RAM, else FREE.
       Of course, add at least one free block, since
        MMap's using one for data! This is my 128K map:

 ..  0 1 2 3 4 5 6 7 8 9 A B C D E F
 #   = = = = = = = = = = = = = = = =
 0   U U U U M U M U M _ _ _ _ _ U .
 1   . . . . . . . . . . . . . . . .
 2   . . . . . . . . . . . . . . . .
 3   . . . . . . . . . . . . . . . U

 Number of Free Blocks:   5
    Ram Free in KBytes:  40
```

```
Microware OS-9 Assembler RS Version 01.00.00    03/30/87 00:15:48    Page 001
MMap - INSIDE OS9 LEVEL II

                                   nam    MMap
                                   ttl    INSIDE OS9 LEVEL II
           * mmap - memory blockmap for cc3
           * 01 feb 87
           * Copyright 1987 by Kevin Darling

        0006              F$Exit   equ    6
        0019              F$GBlkMp equ    $19
        008A              I$Write  equ    $8A
        008C              I$Writln equ    $8C

        0000 87CD01B1              mod    len,name,$11,$81,entry,msize
        000D 4D4D61F0     name     fcs    "MMap"
        0011 03                    fcb    3

        0400              buffsiz  set    1024
      D 0000              leadflag rmb    1
      D 0001              number   rmb    3
      D 0004              free     rmb    1
      D 0005              row      rmb    1
      D 0006              spc      rmb    1
      D 0007              out      rmb    3
      D 000A              mapsiz   rmb    2
      D 000C              blksiz   rmb    2
      D 000E              blknum   rmb    1
      D 000F              buffer   rmb    buffsiz
      D 040F              stack    rmb    200
      D 04D7              msize    equ    .

        0012              header
        0012 20202020              fcc    "    0 1 2 3 4 5 6 7 8 9 A B C D E F"
        0035 0D                    fcb    $0D
        0024              hdrlen   equ    *-header
        0036              hdr2
        0036 20232020              fcc    " #  = = = = = = = = = = = = = = = ="
        0059 0D                    fcb    $0D
        0024              hdrlen2  equ    *-hdr2

        005A              entry
        005A 1700EF                lbsr   crtn
        005D 308DFFB1              leax   header,pc
        0061 8601                  lda    #1
        0063 108E0024              ldy    #hdrlen
        0067 103F8C                OS9    I$Writln
        006A 308DFFC8              leax   hdr2,pc
        006E 108E0024              ldy    #hdrlen2
        0072 103F8A                OS9    I$Write
        0075 304F                  leax   buffer,u     get block map
        0077 103F19                OS9    F$GBlkMp
        007A 1025009B              lbcs   error
```

```

007E 0F0E                  clr   blknum
0080 0F04                  clr   free
     * std blksiz
     * sty mapsiz
0082 304F                  leax  buffer,u
0084 8630                  lda   #$30
0086 9705                  sta   row
0088 8640                  lda   #$40
008A 3402                  pshs  a          save count
008C            loop
008C A6E4                  lda   ,s
008E 850F                  bita  #$0F
0090 261F                  bne   loop2

0092 3410                  pshs  x
0094 1700B5                lbsr  crtn
0097 3046                  leax  spc,u
0099 108E0004              ldy   #4
009D 9605                  lda   row
009F 9707                  sta   out
00A1 0C05                  inc   row
00A3 CC2020                ldd   #$2020
00A6 9706                  sta   spc
00A8 DD08                  std   out+1
00AA 8601                  lda   #1
00AC 103F8A                OS9   I$Write
00AF 3510                  puls  x

00B1            loop2
00B1 E680                  ldb   ,x+        get next block
00B3 270A                  beq   freeram
00B5 2B12                  bmi   notram
00B7 C502                  bitb  #2
00B9 260A                  bne   module
00BB C655                  ldb   #'U        ram-in-use
00BD 200C                  bra   put
00BF            freeram
00BF C65F                  ldb   #'_        not used
00C1 0C04                  inc   free
00C3 2006                  bra   put
00C5            module
00C5 C64D                  ldb   #'M        module
00C7 2002                  bra   put
00C9            notram
00C9 C62E                  ldb   #'.        not ram
00CB            put
00CB D707                  stb   out
00CD C620                  ldb   #$20
00CF D708                  stb   out+1
00D1 3410                  pshs  x
00D3 3047                  leax  out,u
00D5 108E0002              ldy   #2
00D9 8601                  lda   #1
```

```
      00DB 103F8A           OS9   I$Write
      00DE 3510             puls  x
      00E0 6AE4             dec   ,s
W     00E2 1026FFA6         lbne  loop
      00E6 3502             puls  a

      00E8 8D62             bsr   crtn
      00EA 8D60             bsr   crtn
      00EC 308D002C         leax  freemsg,pc
      00F0 108E0018         ldy   #freelen
      00F4 8601             lda   #1
      00F6 103F8A           OS9   I$Write
      00F9 D604             ldb   free
      00FB 4F               clra
W     00FC 170071           lbsr  outdec
      00FF 8D4B             bsr   crtn

      0101 308D002F         leax  rammsg,pc
      0105 108E0018         ldy   #ramlen
      0109 8601             lda   #1
      010B 103F8A           OS9   I$Write
      010E D604             ldb   free
      0110 8608             lda   #8
      0112 3D               mul
W     0113 17005A           lbsr  outdec
      0116 8D34             bsr   crtn
      0118         bye
      0118 5F               clrb
      0119         error
      0119 103F06           OS9   F$Exit

      011C 204E756D freemsg fcc   " Number of Free Blocks: "
      0018         freelen  equ   *-freemsg
      0134 20202020 rammsg  fcc   "    Ram Free in KBytes: "
      0018         ramlen   equ   *-rammsg

      014C         crtn
      014C 3412             pshs  a,x
      014E 860D             lda   #$0D
      0150 9707             sta   out
      0152 3047             leax  out,u
      0154 108E0001         ldy   #1
      0158 8601             lda   #1
      015A 103F8C           OS9   I$Writln
      015D 3592             puls  a,x,pc

      015F         print
      015F 9707             sta   out
      0161 3410             pshs  x
      0163 3047             leax  out,u
      0165 108E0001         ldy   #1
      0169 8601             lda   #1
      016B 103F8A           OS9   I$Write
      016E 3590             puls  x,pc
```

```

0170                outdec    equ   *            D=number
0170 3041                     leax  number,u
0172 0F00                     clr   leadflag
0174 6F84                     clr   ,x
0176 6F01                     clr   1,x
0178 6F02                     clr   2,x
017A                hundred
017A 6C84                     inc   ,x
017C 830064                   subd  #100
017F 24F9                     bcc   hundred
0181 C30064                   addd  #100
0184                ten
0184 6C01                     inc   1,x
0186 83000A                   subd  #10
0189 24F9                     bcc   ten
018B C3000A                   addd  #10
018E 5C                       incb
018F E702                     stb   2,x
0191 8D08                     bsr   printled
0193 8D06                     bsr   printled

0195                printnum
0195 A680                     lda   ,x+
0197 8B2F                     adda  #$30-1       make ascii
0199 20C4                     bra   print

019B                printled
019B 0D00                     tst   leadflag     print leading zero?
019D 26F6                     bne   printnum     ..yes
019F E684                     ldb   ,x           is it zero?
01A1 0C00                     inc   leadflag
01A3 5A                       decb
01A4 26EF                     bne   printnum     ..no, print zero's
01A6 0F00                     clr   leadflag     else surpress
01A8 8620                     lda   #$20
01AA 3001                     leax  1,x
01AC 20B1                     bra   print

01AE 42D247                   emod
01B1                len       equ   *
                              end
```

```
00000 error(s)
00003 warning(s)
$01B1 00433 program bytes generated
$04D7 01239 data bytes allocated
$02B9 00697 bytes used for symbols

000E D BLKNUM    000C D BLKSIZ   000F D BUFFER   0400 S BUFFSIZ  0118 L BYE
014C L CRTN      005A L ENTRY    0119 L ERROR    0006 E F$EXIT   0019 E F$GBLKMP
0004 D FREE      0018 E FREELEN  011C L FREEMSG  00BF L FREERAM  0036 L HDR2
0024 E HDRLEN    0024 E HDRLEN2  0012 L HEADER   017A L HUNDRED  008A E I$WRITE
008C E I$WRITLN  0000 D LEADFLAG 01B1 E LEN      008C L LOOP     00B1 L LOOP2
000A D MAPSIZ    00C5 L MODULE   04D7 E MSIZE    000D L NAME     00C9 L NOTRAM
0001 D NUMBER    0007 D OUT      0170 E OUTDEC   015F L PRINT    019B L PRINTLED
0195 L PRINTNUM  00CB L PUT      0018 E RAMLEN   0134 L RAMMSG   0005 D ROW
0006 D SPC       040F D STACK    0184 L TEN
```

```
==================================================================
PMAP - Process DAT Image Maps. The best. Shows blocks in use
        by processes. Lower is data, top is modules.
       Example: block 09 is mapped into $6000-7FFF in the
        system dat map. Note that Shell in block 06 (see DIRM
        above!) is simply mapped into both procs 2 and 3 at
        $E000-FEFF along with any other modules in that block.

 ID      01 23 45 67 89 AB CD EF  Program
----     -- -- -- -- -- -- -- --  -----------
  1      00 .. .. 09 01 02 03 3F  SYSTEM
  2      05 .. .. .. .. .. .. 06  Shell
  3      07 .. .. .. .. .. .. 06  Shell
  4      0A .. .. .. .. .. .. 08  PMap
```

```
Microware OS-9 Assembler RS Version 01.00.00    03/30/87 00:16:17     Page 001
PMap - INSIDE OS9 LEVEL II

                               nam    PMap
                               ttl    INSIDE OS9 LEVEL II
             * PMap - CC3 proc datimg display
             * 08 feb 87 : derived from my Proc cmd.
             * Copyright 1987 by Kevin Darling

   0088                D.PthDBT equ    $0088
   0003                PD.DEV   equ    $03
   0004                V$DESC   equ    $04

   0006                F$Exit   equ    6
   0018                F$GPrDsc equ    $18
   001B                F$CpyMem equ    $1B
   008A                I$Write  equ    $8A
   008C                I$Writln equ    $8C

   0004                M$Name   equ    4

   0000                P$ID     equ    0
   0001                P$PID    equ    1
   0004                P$SP     equ    4
   0006                P$Task   equ    6
   0007                P$PagCnt equ    7
   0008                P$User   equ    8
   000A                P$Prior  equ    $0A
   000B                P$Age    equ    $0B
   000C                P$State  equ    $0C
   0010                P$IOQN   equ    $10
   0011                P$PModul equ    $11
   0019                P$Signal equ    $19
   0030                P$Path   equ    $30
   0040                P$DATImg equ    $40

   0000 87CD01F8                mod    len,mname,$11,$81,entry,msize
   000D 504D61F0       mname    fcs    "PMap"
   0011 01                      fcb    1

   0200                buffsiz  set    512

 D 0000                umem     rmb    2
 D 0002                sysimg   rmb    2         pointer to sysprc datimage
 D 0004                datimg   rmb    2         datimg for copymem
 D 0006                lineptr  rmb    2
 D 0008                number   rmb    3
 D 000B                leadflag rmb    1
 D 000C                path     rmb    2
 D 000E                pid      rmb    1
 D 000F                hdr      rmb    12        header
 D 001B                out      rmb    80
 D 006B                buffer   rmb    buffsiz   each proc desc
 D 026B                stack    rmb    200
```

```
D 0333                msize    equ
  0012                header
  0012 20494420                fcc    " ID     01 23 45 67 89 AB CD EF
Program
  003E 0D                      fcb    $0D
  002D                hdrlen   equ    *-header
  003F                header2
  003F 2D2D2D2D               fcc    "----    -- -- -- -- -- -- -- --   -------
  006B                hdrcr
  006B 0D                      fcb    $0D
  002D                hdrlen2  equ    *-header2
  006C                entry
  006C DF00                    stu    umem
  006E 8601                    lda    #1
  0070 0F0E                    clr    pid
  0072 308DFFF5                leax   hdrcr,pc
  0076 108E0001                ldy    #1
  007A 103F8C                  OS9    I$Writln
W 007D 10250034                lbcs   error
  0081 308DFF8D                leax   header,pc
  0085 108E002D                ldy    #hdrlen
  0089 103F8C                  OS9    I$Writln
W 008C 10250025                lbcs   error
  0090 308DFFAB                leax   header2,pc
  0094 108E002D                ldy    #hdrlen2
  0098 103F8C                  OS9    I$Writln
  009B                main
  009B DE00                    ldu    umem
  009D 30C81E                  leax   out,u
  00A0 9F06                    stx    lineptr
  00A2 0C0E                    inc    pid           next process
  00A4 270E                    beq    bye           ..>255 = exit
  00A6 960E                    lda    pid           proc id
  00A8 30C86B                  leax   buffer,u      destination buff
  00AB 103F18                  OS9    F$GPrDsc      get proc desc
  00AE 25EB                    bcs    main          ..loop if not one
  00B0 8D0C                    bsr    output        report proc data
  00B2 20E7                    bra    main          ..loop.
  00B4                bye
  00B4 5F                      clrb
  00B5                error
  00B5 103F06                  OS9    F$Exit
  00B8 53595354       sysnam   fcs    "SYSTEM"
  0006                syslen   equ    *-sysnam
  00BE                output
  00BE A684                    lda    P$ID,x        process id
  00C0 1700E6                  lbsr   outdecl
  00C3 1700C1                  lbsr   space
```

```
00C6 1700BE                lbsr  space
00C9 1700BB                lbsr  space
00CC 1700B8                lbsr  space
     * Print Process DATImage:
     * X=proc desc
00CF 3410                  pshs  x
00D1 308840                leax  P$DATImg,x
00D4 C608                  ldb   #8
00D6 3404                  pshs  b
00D8             prntimg
00D8 EC81                  ldd   ,x++
00DA 4D                    tsta
00DB 2710                  beq   prntimg2
00DD 109E06                ldy   lineptr
00E0 CC2E2E                ldd   #"..
00E3 EDA1                  std   ,y++
00E5 109F06                sty   lineptr
00E8 17009C                lbsr  space
00EB 2005                  bra   prntimg3
00ED             prntimg2
00ED 1F98                  tfr   b,a
00EF 170093                lbsr  outhexl
00F2             prntimg3
00F2 6AE4                  dec   ,s
00F4 26E2                  bne   prntimg
00F6 3514                  puls  b,x
     * Print Primary Module Name:
00F8 17008C                lbsr  space
00FB 318840                leay  P$DATImg,x
00FE 1F20                  tfr   y,d         D=dat image in proc desc
0100 DD04                  std   datimg
0102 AE8811                ldx   P$PModul,x X=offset in map
0105 2614                  bne   doname
0107 308DFFAD              leax  >sysnam,pc
010B 109E06                ldy   lineptr
010E C606                  ldb   #syslen
0110             copy
0110 A680                  lda   ,x+
0112 A7A0                  sta   ,y+
0114 5A                    decb
0115 26F9                  bne   copy
0117 8D43                  bsr   name2
0119 2002                  bra   printlin
011B             doname
011B 8D19                  bsr   printnam
011D             printlin
011D 9E06                  ldx   lineptr     now print whole line:
011F 860D                  lda   #$0D
0121 A784                  sta   ,x
0123 DE00                  ldu   umem
0125 30C81B                leax  out,u
```

```
  0128 108E0050             ldy   #80
  012C 8601                 lda   #1
  012E 103F8C               OS9   I$Writln
W 0131 1025FF80             lbcs  error
  0135 39                   rts

       * Find and Print a Module Name:
       * X=mod offset, U=data area, datimg=ptr
  0136            printnam
  0136 3440                 pshs  u
  0138 334F                 leau  hdr,u        destination
  013A DC04                 ldd   datimg       proc datimg ptr
  013C 108E000A             ldy   #10          Y=length
  0140 103F1B               OS9   F$CpyMem     get header
  0143 1025FF6E             lbcs  error

  0147 EC44                 ldd   M$Name,u     get name offset from header
  0149 DE06                 ldu   lineptr      move name to output line
  014B 308B                 leax  d,x          X=offset in map to name
  014D DC04                 ldd   datimg
  014F 108E0028             ldy   #40          max char len
  0153 103F1B               OS9   F$CpyMem     get name
  0156 3540                 puls  u
  0158 1025FF59             lbcs  error

  015C            name2
  015C 3410                 pshs  x
  015E 9E06                 ldx   lineptr
  0160 5F                   clrb               B is length
  0161            name3
  0161 5C                   incb
  0162 A680                 lda   ,x+
  0164 2AFB                 bpl   name3
  0166 C128                 cmpb  #40
  0168 2411                 bcc   name5

  016A 847F                 anda  #$7F         fix it up, then
  016C A71F                 sta   -1,x
  016E C109                 cmpb  #9
  0170 2409                 bcc   name5
  0172 8620                 lda   #$20
  0174            name4
  0174 A780                 sta   ,x+
  0176 5C                   incb
  0177 C109                 cmpb  #9
  0179 25F9                 bcs   name4
  017B            name5
  017B 9F06                 stx   lineptr
  017D 3590                 puls  x,pc

       *----------------------------------------

  017F            outhex2
  017F 3404                 pshs  b
```

```
0181 8D08                  bsr    hexl
0183 3502                  puls   a
0185            outhexl
0185 8D04                  bsr    hexl
0187            space
0187 8620                  lda    #$20
0189 2014                  bra    print

018B            hexl
018B 1F89                  tfr    a,b
018D 44                    lsra
018E 44                    lsra
018F 44                    lsra
0190 44                    lsra
0191 8D02                  bsr    outhex
0193 1F98                  tfr    b,a
0195            outhex
0195 840F                  anda   #$0F
0197 810A                  cmpa   #$0A        0-9
0199 2502                  bcs    outdig
019B 8B07                  adda   #$07        A-F
019D            outdig
019D 8B30                  adda   #$30        make ASCII
019F            print
019F 3410                  pshs   x
01A1 9E06                  ldx    lineptr     ++++
01A3 A780                  sta    ,x+
01A5 9F06                  stx    lineptr
01A7 3590                  puls   x,pc

     *---------------------------------------------------
01A9            outdecl    equ    *           A=number
01A9 1F89                  tfr    a,b
01AB 4F                    clra
01AC            outdec     equ    *           D=number
01AC 0F0B                  clr    leadflag
01AE 3410                  pshs   x
01B0 9E00                  ldx    umem
01B2 3008                  leax   number,x
01B4 6F84                  clr    ,x
01B6 6F01                  clr    1,x
01B8 6F02                  clr    2,x
01BA            hundred
01BA 6C84                  inc    ,x
01BC 830064                subd   #100
01BF 24F9                  bcc    hundred
01C1 C30064                addd   #100
01C4            ten
01C4 6C01                  inc    1,x
01C6 83000A                subd   #10
01C9 24F9                  bcc    ten
01CB C3000A                addd   #10
01CE 5C                    incb
01CF E702                  stb    2,x
```

```
    01D1 8D0F                 bsr    printled
    01D3 8D0D                 bsr    printled
    01D5 8D05                 bsr    printnum
  W 01D7 17FFAD               lbsr   space
    01DA 3590                 puls   x,pc

    01DC           printnum
    01DC A680                 lda    ,x+
    01DE 8B2F                 adda   #$30-1      make ascii
    01E0 20BD                 bra    print

    01E2           printled
    01E2 0D0B                 tst    leadflag    print leading zero?
    01E4 26F6                 bne    printnum    ..yes
    01E6 E684                 ldb    ,x          is it zero?
    01E8 0C0B                 inc    leadflag
    01EA 5A                   decb
    01EB 26EF                 bne    printnum    ..no, print zero's
    01ED 0F0B                 clr    leadflag    else surpress
    01EF 8620                 lda    #$20
    01F1 3001                 leax   1,x
    01F3 20AA                 bra    print

    01F5 474519               emod
    01F8           len        equ    *
                              end

00000 error(s)
00004 warning(s)
$01F8 00504 program bytes generated
$0333 00819 data bytes allocated
$0499 01177 bytes used for symbols

006B D BUFFER    0200 S BUFFSIZ   00B4 L BYE       0110 L COPY      0088 E D.PTHDBT
0004 D DATIMG    011B L DONAME    006C L ENTRY     00B5 L ERROR     001B E F$CPYMEM
0006 E F$EXIT    0018 E F$GPRDSC  000F D HDR       006B L HDRCR     002D E HDRLEN
002D E HDRLEN2   0012 L HEADER    003F L HEADER2   018B L HEX1      01BA L HUNDRED
008A E I$WRITE   008C E I$WRITLN  000B D LEADFLAG  01F8 E LEN       0006 D LINEPTR
0004 E M$NAME    009B L MAIN      000D L MNAME     0333 E MSIZE     015C L NAME2
0161 L NAME3     0174 L NAME4     017E L NAME5     0008 D NUMBER    001B D OUT
01AC E OUTDEC    01A9 E OUTDEC1   019D L OUTDIG    0195 L OUTHEX    0185 L OUTHEX1
017F L OUTHEX2   00BE L OUTPUT    000B E P$AGE     0040 E P$DATIMG  0000 E P$ID
0010 E P$IOQN    0007 E P$PAGCNT  0030 E P$PATH    0001 E P$PID     0011 E P$PMODUL
000A E P$PRIOR   0019 E P$SIGNAL  0004 E P$SP      000C E P$STATE   0006 E P$TASK
0008 E P$USER    000C D PATH      0003 E PD.DEV    000E D PID       019F L PRINT
01E2 L PRINTLED  011D L PRINTLIN  0136 L PRINTNAM  01DC L PRINTNUM  00D8 L PRNTIMG
00ED L PRNTIMG2  00F2 L PRNTIMG3  0187 L SPACE     026B D STACK     0002 D SYSIMG
0006 E SYSLEN    00B8 L SYSNAM    01C4 L TEN       0000 D UMEM      0004 E V$DESC
```

```
==================================================================
 PROC - Like procs, but shows standard in/out devices:
        St = status byte, Sig = pending signal in hex and dec.

Example:

OS9: dirm >/w7 & (setpr 2 255; proc >/dl/test)

  ID Prnt User Pty Age  St Sig ..  Module   Std in/out
 --- ---- ----- --- ---  -- --- --  -------- ----------
   2    1     0 255 255  80   0 00  Shell    <Term >Term
   3    2     0 128 128  80   0 00  Shell    <W1   >W1
   4    2     0 128 128  00   0 00  DirM     <Term >W7
   5    2     0 128 130  80   0 00  Shell    <Term >Term
   6    5     0 128 129  80   0 00  Proc     <Term >D1
```

```
Microware OS-9 Assembler RS Version 01.00.00    03/30/87 00:17:04     Page 001
Proc - INSIDE OS9 LEVEL II

                                  nam   Proc
                                  ttl   INSIDE OS9 LEVEL II
              * Proc - L-II Procs for coco 3

              * 06 feb 87 : add std out also
              * 03 feb 87 : add path name display
              * 01 feb 87 : working version
              * Copyright 1987 by Kevin Darling

        0088            D.PthDBT equu   $0088
        0003            PD.DEV   equ    $03
        0004            V$DESC   equ    $04

        0006            F$Exit   equ    6
        0018            F$GPrDsc equ    $18
        001B            F$CpyMem equ    $1B
        008A            I$Write  equ    $8A
        008C            I$Writln equ    $8C

        0004            M$Name   equ    4

        0000            P$ID     equ    0
        0001            P$PID    equ    1
        0004            P$SP     equ    4
        0006            P$Task   equ    6
        0007            P$PagCnt equ    7
        0008            P$User   equ    8
        000A            P$Prior  equ    $0A
        000B            P$Age    equ    $0B
        000C            P$State  equ    $0C
        0010            P$IOQN   equ    $10
        0011            P$PModul equ    $11
        0019            P$Signal equ    $19
        0030            P$Path   equ    $30
        0040            P$DATImg equ    $40

        0000 87CD028E            mod    len,mname,$11,$81,entry,msize
        000D 50726FE3   mname    fcs    "Proc"
        0011 09                  fcb    9

        0200            buffsiz  set    512

      D 0000            umem     rmb    2
      D 0002            sysimg   rmb    2         pointer to sysprc datimage
      D 0004            datimg   rmb    2         datimg for copymem
      D 0006            lineptr  rmb    2
      D 0008            number   rmb    3
      D 000B            leadflag rmb    1
      D 000C            path     rmb    2
      D 000E            pid      rmb    1
```

```
D 000F               namlen   rmb    1
D 0010               hdr      rmb    64          header
D 0050               out      rmb    80
D 00A0               buffer   rmb    buffsiz     each proc desc
D 02A0               sysprc   rmb    buffsiz     sys proc desc
D 04A0               stack    rmb    200
D 0568               msize    equ    .

  0012               header
  0012 20494420                 fcc   " ID Prnt User Pty Age  St Sig ..  Module
  0048 0D                     fcb    $0D
  0037               hdrlen   equ    *-header
  0049               header2
  0049 2D2D2D20                 fcc   "--- --- ----- --- ---  -- --- --  ------
  007F               hdrcr
  007F 0D                     fcb    $0D
  0037               hdrlen2  equ    *-header2

  0080               entry
  0080 DF00                   stu    umem
  0082 8601                   lda    #1
  0084 970E                   sta    pid
  0086 308DFFF5               leax   hdrcr,pc
  008A 108E0001               ldy    #1
  008E 103F8C                 OS9    I$Writln
W 0091 10250045               lbcs   error
  0095 308DFF79               leax   header,pc
  0099 108E0037               ldy    #hdrlen
  009D 103F8C                 OS9    I$Writln
W 00A0 10250036               lbcs   error
  00A4 308DFFA1               leax   header2,pc
  00A8 108E0037               ldy    #hdrlen2
  00AC 103F8C                 OS9    I$Writln

  00AF 8601                   lda    #1
  00B1 30C902A0               leax   >sysprc,u   get system proc desc
  00B5 103F18                 OS9    F$GPrDsc
  00B8 2520                   bcs    error
  00BA 308840                 leax   P$DATImg,x just for it's datimg
  00BD 9F02                   stx    sysimg

  00BF               main
  00BF DE00                   ldu    umem
  00C1 30C850                 leax   out,u
  00C4 9F06                   stx    lineptr
  00C6 0C0E                   inc    pid         next process
  00C8 270F                   beq    bye         ..>255 = exit
  00CA 960E                   lda    pid         proc id
  00CC 30C900A0               leax   buffer,u    destination buff
  00D0 103F18                 OS9    F$GPrDsc    get proc desc
  00D3 25EA                   bcs    main        ..loop if not one
  00D5 8D06                   bsr    output      report proc data
  00D7 20E6                   bra    main        ..loop.
```

```
00D9               bye
00D9 5F                      clrb
00DA               error
00DA 103F06                  OS9    F$Exit

00DD               output
00DD A684                    lda    P$ID,x        process id
00DF 17015D                  lbsr   outdecl
00E2 A601                    lda    P$PID,x       parent's id
00E4 170158                  lbsr   outdecl
00E7 170133                  lbsr   space
00EA EC08                    ldd    P$User,x      user id
00EC 170153                  lbsr   outdec
00EF 17012B                  lbsr   space
00F2 A60A                    lda    P$Prior,x     priority
00F4 170148                  lbsr   outdecl
00F7 A60B                    lda    P$Age,x
00F9 170143                  lbsr   outdecl
     * lda P$Task,x task number
     * lbsr outhexl
00FC 17011E                  lbsr   space
00FF A60C                    lda    P$State,x     state
0101 170117                  lbsr   outhexl
0104 A68819                  lda    P$Signal,x    signal
0107 170135                  lbsr   outdecl
010A A68819                  lda    P$Signal,x    signal in hex
010D 17010B                  lbsr   outhexl

0110 17010A                  lbsr   space
0113 EC8830                  ldd    P$Path,x      save proc stdin path #
0116 DD0C                    std    path          and pathl stdout

     * Print Primary Module Name:
     * X=proc desc
0118 318840                  leay   P$DATImg,x
011B 1F20                    tfr    y,d           D=dat image in proc desc
011D DD04                    std    datimg
011F AE8811                  ldx    P$PModul,x    X=offset in map
0122 C609                    ldb    #9
0124 D70F                    stb    namlen
0126 1700A2                  lbsr   printnam

     * Print Std Input Device:
0129 863C                    lda    #'<
012B 8D21                    bsr    device
012D               stdout
012D 960D                    lda    path+1
012F 970C                    sta    path
0131 863E                    lda    #'>
0133 8D19                    bsr    device

0135               printlin
0135 9E06                    ldx    lineptr       now print whole line:
0137 860D                    lda    #$0D
```

```
  0139 A784                 sta    ,x
  013B DE00                 ldu    umem
  013D 30C850               leax   out,u
  0140 108E0050             ldy    #80
  0144 8601                 lda    #1
  0146 103F8C               OS9    I$Writln
W 0149 1025FF8D             lbcs   error
  014D 39                   rts

  014E             device
  014E DE00                 ldu    umem
  0150 1700E2               lbsr   print       ("< >")
  0153 960C                 lda    path
  0155 2610                 bne    device2
  0157 8620                 lda    #$20
  0159 C605                 ldb    #5
  015B 109E06               ldy    lineptr
  015E             device0
  015E A7A0                 sta    ,y+
  0160 5A                   decb
  0161 26FB                 bne    device0
  0163 109F06               sty    lineptr
  0166 39                   rts

  0167             device2
  0167 33C810               leau   hdr,u       get path table offset:
  016A DC02                 ldd    sysimg      in system map
  016C 8E0088               ldx    #D.PthDBT   from direct page ptr
  016F 108E0002             ldy    #2
  0173 103F1B               OS9    F$CpyMem
  0176 1025FF60             lbcs   error

  017A 9E10                 ldx    hdr         get path descriptor table:
  017C 108E0040             ldy    #64
  0180 DC02                 ldd    sysimg
  0182 103F1B               OS9    F$CpyMem    (X was D.PthDBT ptr)
  0185 1025FF51             lbcs   error

  0189 D60C                 ldb    path        point to path block:
  018B 54                   lsrb               four paths / sub-block
  018C 54                   lsrb
  018D A6C5                 lda    b,u         A=msb block address
  018F 3402                 pshs   a
  0191 D60C                 ldb    path        then point to path within
  0193 C403                 andb   #3
  0195 8640                 lda    #$40
  0197 3D                   mul
  0198 3502                 puls   a           D=path descriptor address

  019A CB03                 addb   #PD.DEV     and get device table ptr
  019C 1F01                 tfr    d,x
  019E DC02                 ldd    sysimg
  01A0 108E0002             ldy    #2
  01A4 103F1B               OS9    F$CpyMem
```

```
01A7 1025FF2F            lbcs   error

01AB 9E10                ldx    hdr          X=device table entry sys addrs
01AD C604                ldb    #V$DESC      but we want it's desc ptr
01AF 3A                  abx
01B0 DC02                ldd    sysimg
01B2 108E0002            ldy    #2
01B6 103F1B              OS9    F$CpyMem
01B9 1025FF1D            lbcs   error

01BD 9E10                ldx    hdr          then get desc address:
01BF DE00                ldu    umem
01C1 DC02                ldd    sysimg
01C3 DD04                std    datimg
01C5 C605                ldb    #5
01C7 D70F                stb    namlen
01C9 2000                bra    printnam     and get device name

     * Find and Print a Module Name:
     * X=mod offset, U=data area, datimg=ptr
01CB           printnam
01CB 3440                pshs   u
01CD 33C810              leau   hdr,u        destination
01D0 DC04                ldd    datimg       proc datimg ptr
01D2 108E000A            ldy    #10          Y=length
01D6 103F1B              OS9    F$CpyMem     get header
01D9 1025FEFD            lbcs   error

01DD EC44                ldd    M$Name,u     get name offset from header
01DF DE06                ldu    lineptr      move name to output line
01E1 308B                leax   d,x          X=offset in map to name
01E3 DC04                ldd    datimg
01E5 108E0028            ldy    #40          max char len
01E9 103F1B              OS9    F$CpyMem     get name
01EC 3540                puls   u
01EE 1025FEE8            lbcs   error

01F2 3410                pshs   x
01F4 9E06                ldx    lineptr
01F6 5F                  clrb                B is length
01F7           name3
01F7 5C                  incb
01F8 A680                lda    ,x+
01FA 2AFB                bpl    name3
01FC C128                cmpb   #40
01FE 2411                bcc    name5

0200 847F                anda   #$7F         fix it up, then
0202 A71F                sta    -1,x
0204 D10F                cmpb   namlen
0206 2409                bcc    name5
0208 8620                lda    #$20
020A           name4
```

```
0A0A A780                     sta    ,x+
020C 5C                       incb
020D D10F                     cmpb   namlen
020F 25F9                     bcs    name4
0211               name5
0211 9F06                     stx    lineptr
0213 3590                     puls   x,pc

          *---------------------------------------

0215               outhex2
0215 3404                     pshs   b
0217 8D08                     bsr    hexl
0219 3502                     puls   a
021B               outhexl
021B 8D04                     bsr    hexl
021D               space
021D 8620                     lda    #$20
021F 2014                     bra    print

0221               hexl
0221 1F89                     tfr    a,b
0223 44                       lsra
0224 44                       lsra
0225 44                       lsra
0226 44                       lsra
0227 8D02                     bsr    outhex
0229 1F98                     tfr    b,a
022B               outhex
022B 840F                     anda   #$0F
022D 810A                     cmpa   #$0A        0-9
022F 2502                     bcs    outdig
0231 8B07                     adda   #$07        A-F
0233               outdig
0233 8B30                     adda   #$30        make ASCII
0235               print
0235 3410                     pshs   x
0237 9E06                     ldx    lineptr     ++++
0239 A780                     sta    ,x+
023B 9F06                     stx    lineptr
023D 3590                     puls   x,pc

          *---------------------------------------
023F               outdecl    equ    *           A=number
023F 1F89                     tfr    a,b
0241 4F                       clra
0242               outdec     equ    *           D=number
0242 0F0B                     clr    leadflag
0244 3410                     pshs   x
0246 9E00                     ldx    umem
0248 3008                     leax   number,x
024A 6F84                     clr    ,x
024C 6F01                     clr    1,x
024E 6F02                     clr    2,x
```

```
   0250                   hundred
   0250 6C84                      inc    ,x
   0252 830064                    subd   #100
   0255 24F9                      bcc    hundred
   0257 C30064                    addd   #100
   025A                   ten
   025A 6C01                      inc    1,x
   025C 83000A                    subd   #10
   025F 24F9                      bcc    ten
   0261 C3000A                    addd   #10
   0264 5C                        incb
   0265 E702                      stb    2,x

   0267 8D0F                      bsr    printled
   0269 8D0D                      bsr    printled
   026B 8D05                      bsr    printnum
W  026D 17FFAD                    lbsr   space
   0270 3590                      puls   x,pc

   0272                   printnum
   0272 A680                      lda    ,x+
   0274 8B2F                      adda   #$30-1      make ascii
   0276 20BD                      bra    print

   0278                   printled
   0278 0D0B                      tst    leadflag    print leading zero?
   027A 26F6                      bne    printnum    ..yes
   027C E684                      ldb    ,x          is it zero?
   027E 0C0B                      inc    leadflag
   0280 5A                        decb
   0281 26EF                      bne    printnum    ..no, print zero's
   0283 0F0B                      clr    leadflag    else surpress
   0285 8620                      lda    #$20
   0287 3001                      leax   1,x
   0289 20AA                      bra    print

   028B 015EAF                    emod
   028E                   len     equ    *
                                  end
```

```
00000 error(s)
00004 warning(s)
$028E 00654 program bytes generated
$0568 01384 data bytes allocated
$047B 01147 bytes used for symbols

00A0 D BUFFER   0200 S BUFFSIZ  00D9 L BYE      0088 E D.PTHDBT 0004 D DATIMG
014E L DEVICE   015E L DEVICE0  0167 L DEVICE2  0080 L ENTRY    00DA L ERROR
001B E F$CPYMEM 0006 E F$EXIT   0018 E F$GPRDSC 0010 D HDR      007F L HDRCR
0037 E HDRLEN   0037 E HDRLEN2  0012 L HEADER   0049 L HEADER2  0221 L HEX1
0250 L HUNDRED  008A E I$WRITE  008C E I$WRITLN 000B D LEADFLAG 028E E LEN
0006 D LINEPTR  0004 E M$NAME   00BF L MAIN     000D L MNAME    0568 E MSIZE
01F7 L NAME3    020A L NAME4    0211 L NAME5    000F D NAMLEN   0008 D NUMBER
0050 D OUT      0242 E OUTDEC   023F E OUTDEC1  0233 L OUTDIG   022B L OUTHEX
021B L OUTHEX1  0215 L OUTHEX2  00DD L OUTPUT   000B E P$AGE    0040 E P$DATIMG
0000 E P$ID     0010 E P$IOQN   0007 E P$PAGCNT 0030 E P$PATH   0001 E P$PID
0011 E P$PMODUL 000A E P$PRIOR  0019 E P$SIGNAL 0004 E P$SP     000C E P$STATE
0006 E P$TASK   0008 E P$USER   000C D PATH     0003 E PD.DEV   000E D PID
0235 L PRINT    0278 L PRINTLED 0135 L PRINTLIN 01CB L PRINTNAM 0272 L PRINTNUM
021D L SPACE    04A0 D STACK    012D L STDOUT   0002 D SYSIMG   02A0 D SYSPRC
025A L TEN      0000 D UMEM     0004 E V$DESC
```

SMAP - Show system page memory map. As above, except in pages.
Important info adding drivers, starting many procs, etc.

```
   0 1 2 3 4 5 6 7 8 9 A B C D E F
#  = = = = = = = = = = = = = = = =
0  U U U U U U U U U U U U U U U U
1  U U U U U U U U U U U U U U U U
2  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
3  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
4  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
5  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
6  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
7  _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
8  _ U U U U _ U U U U U U U U U U
9  U U U U U U U U U U U U U U U U
A  U U U U U U U U U U U U U U U U
B  U U U U U U U U U U U U U U U U
C  U U U U U U U U U U U U U U U U
D  U U U U U U U U U U U U U U U U
E  U U U U U U U U U U U U U U U U
F  U U U U U U U U U U U U U U U .

Number of Free Pages:  98
  Ram Free in KBytes:  24
```

```
Microware OS-9 Assembler RS Version 01.00.00    03/30/87 00:17:48     Page 001
SMap - INSIDE OS9 LEVEL II

                                  nam    SMap
                                  ttl    INSIDE OS9 LEVEL II
              * SMap - system memory blockmap for cc3
              * 08 feb 87
              * Copyright 1987 by Kevin Darling

      004E               D.SysMem equ    $004E       system mem map

      0006               F$Exit   equ    6
      001B               F$CpyMem equ    $1B
      008A               I$Write  equ    $8A
      008C               I$Writln equ    $8C

      0000 87CD01D5               mod    len,name,$11,$81,entry,msize
      000D 534D61F0      name     fcs    "SMap"
      0011 01                     fcb    1

      0100               buffsiz  set    256

    D 0000               leadflag rmb    1
    D 0001               number   rmb    3
    D 0004               free     rmb    1
    D 0005               row      rmb    1
    D 0006               spc      rmb    1
    D 0007               out      rmb    3
    D 000A               mapsiz   rmb    2
    D 000C               blksiz   rmb    2
    D 000E               blknum   rmb    1
    D 000F               buffer   rmb    buffsiz
    D 010F               stack    rmb    200
    D 01D7               msize    equ    .

      0012               header
      0012 20202020               fcc    "    0 1 2 3 4 5 6 7 8 9 A B C D E F"
      0035 0D                     fcb    $0D
      0024               hdrlen   equ    *-header
      0036               hdr2
      0036 20232020               fcc    " #  = = = = = = = = = = = = = = = ="
      0059 0D                     fcb    $0D
      0024               hdrlen2  equ    *-hdr2

      005A 00000000      datimg   fdb    0,0

      005E               entry
      005E 17010F                 lbsr   crtn
      0061 308DFFAD               leax   header,pc
      0065 8601                   lda    #1
      0067 108E0024               ldy    #hdrlen
      006B 103F8C                 OS9    I$Writln
      006E 308DFFC4               leax   hdr2,pc
      0072 108E0024               ldy    #hdrlen2
```

```
0076 103F8A                  OS9    I$Write

     * Get SysMap Ptr:
0079 308DFFDD                leax   datimg,pc
007D 1F10                    tfr    x,d
007F 8E004E                  ldx    #D.SysMem
0082 108E0002                ldy    #2
0086 3440                    pshs   u
0088 334F                    leau   buffer,u
008A 103F1B                  OS9    F$CpyMem
008D 3540                    puls   u
008F 102500AC                lbcs   error

     * Get SysMap:
0093 AE4F                    ldx    buffer,u      get map address
0095 108E0100                ldy    #buffsiz
0099 3440                    pshs   u
009B 334F                    leau   buffer,u
009D 103F1B                  OS9    F$CpyMem
00A0 3540                    puls   u
00A2 10250099                lbcs   error

00A6 0F0E                    clr    blknum
00A8 0F04                    clr    free
     * std blksiz
     * sty mapsiz
00AA 304F                    leax   buffer,u
00AC 8630                    lda    #$30
00AE 9705                    sta    row
00B0 6FE2                    clr    ,-s           save count
00B2               loop
00B2 A6E4                    lda    ,s
00B4 850F                    bita   #$0F
00B6 2627                    bne    loop2

00B8 3410                    pshs   x
00BA 1700B3                  lbsr   crtn
00BD 3046                    leax   spc,u
00BF 108E0004                ldy    #4
00C3 9605                    lda    row
00C5 813A                    cmpa   #$3A
00C7 2604                    bne    oknum
00C9 8641                    lda    #$41
00CB 9705                    sta    row
00CD               oknum
00CD 9707                    sta    out
00CF 0C05                    inc    row
00D1 CC2020                  ldd    #$2020
00D4 9706                    sta    spc
00D6 DD08                    std    out+1
00D8 8601                    lda    #1
00DA 103F8A                  OS9    I$Write
00DD 3510                    puls   x

```

```
   00DF               loop2
   00DF E680                  ldb   ,x+         get next block
   00E1 270A                  beq   freeram
   00E3 2B04                  bmi   notram
   00E5 C655                  ldb   #'U         ram-in-use
   00E7 2008                  bra   put
   00E9               notram
   00E9 C62E                  ldb   #'.         not RAM
   00EB 2004                  bra   put
   00ED               freeram
   00ED C65F                  ldb   #'_         not used
   00EF 0C04                  inc   free
   00F1               put
   00F1 D707                  stb   out
   00F3 C620                  ldb   #$20
   00F5 D708                  stb   out+1
   00F7 3410                  pshs  x
   00F9 3047                  leax  out,u
   00FB 108E0002              ldy   #2
   00FF 8601                  lda   #1
   0101 103F8A                OS9   I$Write
   0104 3510                  puls  x
   0106 6AE4                  dec   ,s
 W 0108 1022FFA6              lbhi  loop
   010C 3502                  puls  a

   010E 8D60                  bsr   crtn
   0110 8D5E                  bsr   crtn
   0112 308D002C              leax  freemsg,pc
   0116 108E0017              ldy   #freelen
   011A 8601                  lda   #1
   011C 103F8A                OS9   I$Write
   011F D604                  ldb   free
   0121 4F                    clra
 W 0122 17006F                lbsr  outdec
   0125 8D49                  bsr   crtn

   0127 308D002E              leax  rammsg,pc
   012B 108E0017              ldy   #ramlen
   012F 8601                  lda   #1
   0131 103F8A                OS9   I$Write
   0134 D604                  ldb   free
   0136 4F                    clra
   0137 54                    lsrb
   0138 54                    lsrb
 W 0139 170058                lbsr  outdec
   013C 8D32                  bsr   crtn
   013E               bye
   013E 5F                    clrb
   013F               error
   013F 103F06                OS9   F$Exit

   0142 204E756D      freemsg fcc   " Number of Free Pages: "
   0017               freelen equ   *-freemsg
```

```
0159 20202052   rammsg   fcc    "   Ram Free in KBytes: "
0017            ramlen   equ    *-rammsg

0170            crtn
0170 3412                pshs   a,x
0172 860D                lda    #$0D
0174 9707                sta    out
0176 3047                leax   out,u
0178 108E0001            ldy    #1
017C 8601                lda    #1
017E 103F8C              OS9    I$Writln
0181 3592                puls   a,x,pc

0183            print
0183 9707                sta    out
0185 3410                pshs   x
0187 3047                leax   out,u
0189 108E0001            ldy    #1
018D 8601                lda    #1
018F 103F8A              OS9    I$Write
0192 3590                puls   x,pc

0194            outdec   equ    *           D=number
0194 3041                leax   number,u
0196 0F00                clr    leadflag
0198 6F84                clr    ,x
019A 6F01                clr    1,x
019C 6F02                clr    2,x
019E            hundred
019E 6C84                inc    ,x
01A0 830064              subd   #100
01A3 24F9                bcc    hundred
01A5 C30064              addd   #100
01A8            ten
01A8 6C01                inc    1,x
01AA 83000A              subd   #10
01AD 24F9                bcc    ten
01AF C3000A              addd   #10
01B2 5C                  incb
01B3 E702                stb    2,x
01B5 8D08                bsr    printled
01B7 8D06                bsr    printled

01B9            printnum
01B9 A680                lda    ,x+
01BB 8B2F                adda   #$30-1      make ascii
01BD 20C4                bra    print

01BF            printled
01BF 0D00                tst    leadflag    print leading zero?
01C1 26F6                bne    printnum    ..yes
01C3 E684                ldb    ,x          is it zero?
01C5 0C00                inc    leadflag
01C7 5A                  decb
```

```
01C8 26EF                bne   printnum   ..no, print zero's
01CA 0F00                clr   leadflag   else surpress
01CC 8620                lda   #$20
01CE 3001                leax  1,x
01D0 20B1                bra   print

01D2 1F9F9F              emod
01D5           len       equ   *
                         end

00000 error(s)
00003 warning(s)
$01D5 00469 program bytes generated
$01D7 00471 data bytes allocated
$02D7 00727 bytes used for symbols

000E D BLKNUM    000C D BLKSIZ    000F D BUFFER    0100 S BUFFSIZ   013E L BYE
0170 L CRTN      004E E D.SYSMEM  005A L DATIMG    005E L ENTRY     013F L ERROR
001B E F$CPYMEM  0006 E F$EXIT    0004 D FREE      0017 E FREELEN   0142 L FREEMSG
00ED L FREERAM   0036 L HDR2      0024 E HDRLEN    0024 E HDRLEN2   0012 L HEADER
019E L HUNDRED   008A E I$WRITE   008C E I$WRITLN  0000 D LEADFLAG  01D5 E LEN
00B2 L LOOP      00DF L LOOP2     000A D MAPSIZ    01D7 E MSIZE     000D L NAME
00E9 L NOTRAM    0001 D NUMBER    00CD L OKNUM     0007 D OUT       0194 E OUTDEC
0183 L PRINT     01BF L PRINTLED  01B9 L PRINTNUM  00F1 L PUT       0017 E RAMLEN
0159 L RAMMSG    0005 D ROW       0006 D SPC       010F D STACK     01A8 L TEN
```

INSIDE OS9 LEVEL II

Reference

```
--------------------------------------------------------------------------
I          COCO-3 MEMORY, and GIME REGISTER MAP    (1 Sept 86)          I
--------------------------------------------------------------------------

SYSTEM MEMORY MAP:
 RAM      00000 - 7FFFF 512K bytes
 ROM      78000 - 7FEFF when enabled
 I/O      XFF00 - XFFFF I/O space and GIME regs

64K PROCESS MAP:
 RAM       0000 - FEFF (possible vector page FEXX)
 I/O       FF00 - FFFF (appears in all pages)

Note: the Vector Page RAM at 7FE00 - 7FEFF, when enabled, will appear instead
 of the RAM or ROM at XFE00 - XFEFF. (see FF90 Bit 3)

  XFF00-0X  PIA0         (not fully decoded)
  XFF10-1F  reserved
  XFF20-2X  PIA1         (not fully decoded)
  XFF30-3F  reserved
  XFF40-5F  SCS          (see note on FF90 Bit 2)
  XFF60-7F  undecoded    (for current peripherals)
  XFF80-8F  reserved
 ============================================================================

 FF90  INITIALIZATION REGISTER 0
        Bit 7 - CoCo Bit  1= Color Computer 1/2 Compatible
        Bit 6 -           1= MMU enabled
        Bit 5 -           1= GIME IRQ output enabled to CPU
        Bit 4 -           1= GIME FIRQ  "       "
        Bit 3 -           1= Vector page RAM at FEXX enabled
        Bit 2 -           1= Standard SCS
        Bit 1 -           ROM mapping      0 X - 16K internal, 16K external
        Bit 0 -            "    "          1 0 - 32K internal
                                           1 1 - 32K external

 CoCo bit set = MMU disabled, Video address from SAM, RGB/Comp Palettes = CC2.
 Interrupt bits 5 and/or 4 must be set for FIRQ/IRQ FF92-3 to pass to CPU.
 Access and moves throughout mem are usually done from constant RAM at FEXX.
 If Bit2=0, then XFF50-5F is SCS, and XFF40-4F will be internal to CoCo.
 ----------------------------------------------------------------------------

 FF91  INITIALIZATION REGISTER 1
        Bit 5 - TINS      Timer INput Clock Select:  0= 70 nsec, 1= 63 usec
        Bit 0 - TR        MMU Task Register Select (0/1 - see FFA0-AF)
 ----------------------------------------------------------------------------
```

```
FF92  IRQENR    Interrupt Request Enable Register (IRQ)
 FF93  FIRQENR   Fast Interrupt Request Enable Reg (FIRQ)
   (Note that the equivalent interrupt output enable bit must be set in FF90.)
   Both registers use the following bits to enable/disable device interrupts:
       Bit 5 - TMR        Timer
       Bit 4 - HBORD      Horizontal border
       Bit 3 - VBORD      Vertical border
       Bit 2 - EI2        Serial data input
       Bit 1 - EI1        Keyboard
       Bit 0 - EI0        Cartridge (CART)

   I have no idea if both IRQ & FIRQ can be enabled for a device at same time.
 -------------------------------------------------------------------------------

 FF94  Timer MSB    Write here to start timer.
 FF95  Timer LSB
  Load starts timer countdown. Interrupts at zero, reloads count & continues.
  Must turn timer interrupt enable off/on again to reset timer IRQ/FIRQ.

 FF96  reserved
 FF97  reserved
 -------------------------------------------------------------------------------

 FF98  Alpha/graphics Video modes, and lines per row.
        Bit 7 = BP        0 is alphanumeric, 1= bit plane (graphics)
        Bit 6 = na        ...
        Bit 5 = BPI       1= color burst phase change
        Bit 4 = MOCH      MOnoCHrome bit (composite video output) (1=mono)
        Bit 3 = H50       50hz vs 60hz bit
        Bit 2 = LPR2      Number of lines/char row:
        Bit 1 = LPR1       (Bits 2-1-0 below:)
        Bit 0 = LPR0
                          000 - 1 line/char row    100 - 9  lines/char row
                          001 - 2                  101 - 10
                          010 - 3                  110 - 11 (??)
                          011 - 8                  111 - 12 (??)
 -------------------------------------------------------------------------------

 FF99  VIDEO RESOLUTION REGISTER
        Bit 7 - na      ...                          (bits 6-5):
        Bit 6 - LPF1    Lines Per Field:        00= 192 lines    10= 210 lines
        Bit 5 - LPF0     "    "    "            01= 200 lines    11= 225 lines
        Bit 4 - HR2     Horizontal Resolution
        Bit 3 - HR1         "         "
        Bit 2 - HR0         "         "         (see below for HR, CRES bits)
        Bit 1 - CRES1   Color RESolution bits
        Bit 0 - CRES0     "       "
```

TEXT MODES:

Text: CoCo Bit= 0 and FF98 bit7=0. CRES0 = 1 for: attribute bytes are used.

	HR2	HR1	HR0	(HR1 = don't care for text)
80 char/line	1	X	1	
64 "	1	X	0	
40 "	0	X	1	
32 "	0	X	0	

GRAPHICS MODES:

X	Colors		HR2	HR1	HR0	CRES1	CRES0	Bytes/line	
640	4	-	1	1	1	0	1	160	
640	2	-	1	0	1	0	0	80	
512	4	-	1	1	0	0	1	128	
512	2	-	1	0	0	0	0	64	
320	16	-	1	1	1	1	0	160	Other combo's are
320	4	-	1	0	1	0	1	80	possible, but not
320	2	-	0	1	1	0	0	40	supported.
256	16	-	1	1	0	1	0	128	
256	4	-	1	0	0	0	1	64	
256	2	-	0	1	0	0	0	32	
160	16	-	1	0	1	1	0	40	

Old SAM modes work if CC Bit set. HR and CRES are Don't Care in SAM mode.
Note the correspondence of HR2 HR0 to the text mode's bytes/line. -Kev

---

FF9A Border Palette Register (XX00 0000 = CoCo 1/2 compatible)
FF9B Reserved

FF9C Vertical Fine Scroll Register
FF9D Screen Start Address Register 1 (bits 18-11)
FF9E Screen Start Address Register 0 (bits 10-3)
FF9F Horizontal Offset Register

```
Bit 7 - horizontal offset enable bit (128 char width always)
Bit 6 - X6  ... offset count (0-127)
Bit 5 - X5    for column scan start.
Bit 4 - X4
Bit 3 - X3
Bit 2 - X2
Bit 1 - X1
Bit 0 - X0
```

If Bit 7 set & in Text mode, then there are 128 chars (only 80 seen)/line. This allows an offset to be specified into a virtual 128 char/line screen, useful for horizontal hardware scrolling on wide text or spreadsheets.

```
  FFA0-AF  MEMORY MANAGEMENT UNIT (MMU)
  FFA0-A7  Task #0 Map Set  (8K block numbers in the 64K map)
  FFA8-AF  Task #1 Map Set  (Task map in use chosen by FF91 Bit 0)
```

Each register has 6 bits into which is stored the block number 0-63 ($00-$3F) of the Physical 8K RAM block (out of 512K) that you wish to appear at the CPU Logical address corresponding to that register.

Also can be shown this way: the 6 register bits, when the Logical Address in the range of that register, will become the new Physical RAM address bits:

```
         18  17  16   15  14  13
```

MMU Register: Task0	Task1	CPU: Logical Address	/ Block#
FFA0	FFA8	0000 - 1FFF	0
FFA1	FFA9	2000 - 3FFF	1
FFA2	FFAA	4000 - 5FFF	2
FFA3	FFAB	6000 - 7FFF	3
FFA4	FFAC	8000 - 9FFF	4
FFA5	FFAD	A000 - BFFF	5
FFA6	FFAE	C000 - DFFF	6
FFA7	FFAF	E000 - FDFF	7

The 6-Bit Physical Block Number placed in a MMU register will become the A13-A18 lines when the corresponding Logical Add is accessed by the CPU.

Ex: You wish to access Physical RAM address $35001. That Address is:

```
A- 18  17  16  15  14  13  12  11  10   9   8   7   6   5   4   3   2   1   0
   .....3....  .......5......  .......0......  .......0......  .......1......
    0   1   1   0   1   0   1   0   0   0   0   0   0   0   0   0   0   0   1
```

Taking address bits 18-13, we have: 0 1 1 0 1 0, or $1A, or 26. This is the physical RAM block number, out of the 64 (0-63) available in a 512K machine.

Now, let's say you'd like to have that block appear to the CPU at Logical Block 0 (0000-1FFF in the CPU's 64K memory map).

You would store the Physical Block Number ($1A) in either of the two Task Map registers that are used for Logical Block 0 (FFA0 or FFA8). Unless your pgrm doing this is in the Vector RAM at FEXX (set FF90 Bit 3, so ALWAYS there), you would want to use your current Task Map Register Set. If the TR bit at FF91 was 0, then you'd use MMU register FFA0 for the $1A data byte.

To find the address within the block, use Address Bits 12-0 plus the Logical base address (which in this case is $0000):

Now you could read/write address $1001, which would actually be $35001.

```
FFB0-BF  COLOR PALETTE REGISTERS  (6 bits each)
 FFB0 - palette 0
 FFB1 - palette 1        The pixel or text attribute bits in video memory
 ...                      form the address of a color palette (0-15).
 FFBF - palette 15       It is the color info in that palette which is seen.

 Reg bits- 5  4  3  2  1  0
 CMP ...  I1 I0 P3 P2 P1 P0    Intensity and Phase (16 colors x 4 shades)
 RGB ...  R1 G1 B1 R0 G0 B0    Red Green Blue      (64 RGB combo's)
```

When CoCo Bit is set, and palette registers preloaded with certain default values (ask, if you need these), both the RGB and CMP outputs appear the same color, supposedly.

40/80 Column Text Screen Bytes are Even=char, Odd=attribute, in memory.
Characters selected from 128 ASCII. NO text graphics-chars.

```
  Char Attributes- 8 bits...  F U T T T B B B
       Flashing, Underline, Text foregrnd, Backgrnd colors 0-7.
```

```
FFC0-DF  SAM : same as before (mostly compatible Write-Only Switches)
 FFD8 = CPU .895 MHz   (no address-dependent speed)
 FFD9 =     1.79 MHz
 FFDE = Map RAM/ROM
 FFDF =     all RAM
```

ADDENDUM

This is an addendum to the GIME information.
Thanks to Greg Law and his friend Dennis Weldy for much register info.

==

GIME Register Corrections:

$FF91 - Bit 5, Timer Input Select. Looks like 0=slower speed, instead. Haven't had time to put a scope on it to check actual clocks, yet. Not sure.

$FF92-3 - Interrupt Request Regs: You can also read these regs to see if there is a LOW on an interrupt input pin. If you have both the IRQ and FIRQ for the same device enabled, you read a Set bit on both regs if that input is low.

For example, if you set $FF02=0 and $FF92=2, then as long as a key is held down, you will read back Bit 1 as Set.

The keyboard interrupt input is generated by simply AND'ing all the matrix pins read back at $FF00. Therefore, you could select the key columns you wished to get by setting the appropriate bits at $FF02 to zero. Pressing the key drops the associated $FF00 line to zero, causing the AND output to go low to the GIME. Setting $FF02 to all Ones would mean only the Joystick Fire buttons would generate interrupts.

$FF94-95 - Storing a $00 at $FF94 seems to stop the timer. Also, apparently each time it passes thru zero, the $FF92/93 bit is set without having to re-enable that Int Request.

$FF98 - Bit 5 is the artifact color shift bit. Change it to flip Pmode 4 colors. A One is what is put there if you hold down the F1 key on reset. POKE &HFF98,&H13 from Basic if your colors artifact the wrong way for you.

$FF9F - Horz Offset Reg. If you set Bit 7 and you're in Gfx mode, you can scroll across a 128 byte picture. To use this, of course, you'd have to write your own gfx routines. On my machine, tho, an offset of more than about 5 crashes.

$FFB0-BF - As I originally had, and we all know by now, FFB0-B7 are used for the text mode char background colors, and FFB8-BF for char foreground colors, in addition to their other gfx use.

CoCo-3 Internal Tidbits:

The 68B09E address lines finally have pullup resistors on them. Probably put in for the 2MHz mode, they also help cure a little-known CoCo phantom: since during disk access, the Halt line tri-states the address, data, and R/W lines, some old CoCo's would float those lines right into writing junk in memory. Now $FFFF would be presented to the system bus instead.

Since the GIME catches the old VDG mode info formerly written to the PIA at $FF22, those four now-unconnected lines (PB4-7 on the 6821) might have some use for us.

Also, Pin 10 of the RGB connector is tied to PB3 on the same PIA. Shades of the Atari ST. Could possibly be used to detect type of monitor attached, if we like.

Data read back from RAM must go thru a buffer, the GIME, and another buffer. Amazing that it works at 2 MHz.

In case you didn't catch the hint from GIME.TXT on FF90 Bit 2, the option of an internal SCS select opens up the possibility of a CoCo-4 with a built-in disk controller.

```
==============================================================================
GIME PINS:

   61 63 65 67 01 03 05 07 09              09 ------- 01 68 ------ 61
60 62 64 66 68 02 04 06 08 11 10        10                           60
58 59                      13 12        1                            1
56 57                      15 14        1                            1
54 55                      17 16        1                            1
52 53       Bottom         19 18        1            Top             1
50 51                      21 20        1                            1
48 49                      23 22        1                            1
46 47                      25 24        1                            1
44 45 42 40 38 36 34 32 30 28 26        26                           44
   43 41 39 37 35 33 31 29 27              27 ------------------- 43
```

01 - GND	18 - D6	35 - +5 Volts	52 - A13
02 - XTAL	19 - D7	36 - Z3	53 - A14
03 - XTAL	20 - FIRQ* ->CPU	37 - Z4	54 - A15
04 - RAS*	21 - IRQ* -->CPU	38 - test(+5)	55 - VSYNC*
05 - CAS*	22 - CART* Int in	39 - Z5	56 - HSYNC*
06 - E	23 - KeyBd* Int in	40 - Z6	57 - D7 (RAM)
07 - Q	24 - RS232* Int in	41 - Z7	58 - D6
08 - R/W*	25 - A0 (fm CPU)	42 - Z8	59 - D5
09 - RESET*	26 - A1	43 - A4 (fm CPU)	60 - D4
10 - WEn* 0	27 - A2	44 - A5	61 - D3
11 - WEn* 1	28 - A3	45 - A6	62 - D2
12 - D0 (CPU)	29 - S2 .	46 - A7	63 - D1
13 - D1	30 - S1 .	47 - A8	64 - D0
14 - D2	31 - S0 .	48 - A9	65 - Comp Vid
15 - D3	32 - Z0 (RAM)	49 - A10	66 - Blue
16 - D4	33 - Z1	50 - A11	67 - Green
17 - D5	34 - Z2	51 - A12	68 - Red

```
Notes: WEnx = Write Enables for Banks 0 and 1 RAM
       S2-0 = (address select code -> 74LS138) :
 000 -0- ROM      010 -2- FF0X, FF2X     100 -4- int SCS    110 -6- norm SCS
 001 -1- CTS      011 -3- FF1X, FF3X     101 -5- n/a        111 -7- ??ram??
```

CONNECTORS:

(CN5,6 - top to bottom, CN2 - left to right)

```
CN6 - Gnd, +5, D1, D0, D2, D3, D6, D7, D5, D4, WEn1, Gnd
CN5 - Gnd, D2, D3, D1, WEn0, D0, CAS, D6, D5, D4, D7, Gnd
CN2 - Gnd, RAS, Z0, Z1 , Z2, Z3, Z6, Z5, Z4, Z7, Z8, Gnd
```

Tho as far as the CN's go, even if I have messed up all but the CAS, RAS, WEn's, and +5, you could connect the extra RAM Dx and Zx pins in parallel to each bank in any order. Most RAM's don't care.

CN6 and CN5 data lines go to separate 256K banks, of course.

==

General Info:

```
Data is written to the RAM by byte thru IC10 or IC11, selected by WEn 0 or 1.
     (write enable 0 = even addresses, write enable 1 = odd addresses)
  Two bank RAM data is read back to the GIME thru IC12 & IC13, byte at a time.
  The CPU can then get it from the GIME by byte.
   IC 10, 11, 12 = 74LS244 buffer.   IC13 = 74LS374 latch clocked by CAS* rise.
     RAM Read --> IC12 --> GIME enabled by CAS low. (read first)
     RAM Read --> IC13 --> GIME enabled by CAS hi.  (latched & read)

 Test Points:
   TP 2 = E          TP 4 = RAS         TP 6 = Comp Video      TP 9 = Green
   TP 3 = Q          TP 5 = CAS         TP 8 = Red             TP10 = Blue
 ============================================================================
```

===

IRQ POLLING TABLE

===

A list of 9-byte entries, one for each device controller / driver that has used the F$Irq call. When an IRQ comes, IOMAN uses this list to find the device that is requesting service.

IOMAN then JSR's to the driver's interrupt routine, which is expected to clear the IRQ, and do whatever I/O is required. The driver normally will wake up V.WAKE, the process that was using the device. (The driver had put the process to sleep.)

===

DEVICE TABLE

===

When a device is first called upon, IOMAN inserts quick reference info about the device in the table, and calls the device's INIT subroutine that first time only.

Table used by IOMAN for making path desc's & calling the device's file mgr; by file mgr to call device's driver.

===

MODULE DIRECTORY

===

Table of modules in memory, at 00A00-00FFF. Contains info on their physical address, and used by OS9 for quick lookup of module names. Also used to keep track of the number of users.

===

PATH DESCRIPTORS

===

Each open path has a Path Descriptor, which is shared by all processes that got the path desc by I$Dup'ing a path, or by having the path passed to it by the F$Fork call, which dup's the first 3 standard path's of the parent to the child.

The desc block number is NOT the number you use in a program to access the path. The block number is stored in the process desc I/O path table in the order in which the paths are opened (they take the first empty spot found in the proc path table).

Your number is simply an index into the path desc I/O table in your process descriptor, which is then used by IOMAN to get the real path desc block number.

The base address of all path desc's is in D.PthDBT.

===

```
==============================================================
Entry Format            IRQ POLLING TABLE
==============================================================

Q$POLL   00-01   Polling address (status byte)
Q$FLIP   02      Flip byte for negative logic
Q$MASK   03      Mask byte for IRQ bit
Q$SERV   04-05   Service routine
Q$STAT   06-07   Static storage address
Q$PRTY   08      Priority of device
POLLSIZ  .       Size of each entry
==============================================================
Entry Format              DEVICE TABLE
==============================================================

V$DRIV   00-01   Driver module
V$STAT   02-03   Driver static storage
V$DESC   04-05   Descriptor module
V$FMGR   06-07   File manager module
V$USRS   08      Device user count
DEVSIZ   .       Size of each entry
==============================================================
Entry Format            MODULE DIRECTORY
==============================================================

MD$MPDAT 00-01   Module's block(s) DAT Image Pointer
MD$MBSiz 02-03   Memory Block Size
MD$MPtr  04-05   Offset pointer in block to module
MD$Link  06-07   Module Link Count

==============================================================
Block Format           PROC/PATH DESRIPTORS
==============================================================
```

Descriptors (process/path) are allocated in 64-byte blocks, out of 256-byte pages. The very first block is dedicated as pointers to this and any other pages needed to hold the max # of descriptors in use.

```
00-3F    MSB's of pages allocated to this type of descriptor
40-7F    Descriptor #1
80-BF    Descriptor #2
C0-FF    Descriptor #3
```

Therefore, byte $01 in the first page above points to the next page of four 64-byte blocks:

```
00-3F    Descriptor #4
40-7F    Descriptor #5
80-BF    Descriptor #6
C0-FF    Descriptor #7
```

The descriptor # is used as the proc ID / path pointer by the system. If the descriptor is not in use (killed/closed), the first byte of the block is cleared as a flag, else it is equal to the descriptor number itself.

MODULE TYPES

```
$10  Prgrm  Program module        $C0  Systm  System module
$20  Sbrtn  Subroutine mod        $D0  FlMgr  File manager
$30  Multi  Multi-module          $E0  Drivr  Device driver
$40  Data   Data module           $F0  Devic  Device descriptor
```

UNIVERSAL MODULE HEADER

```
M$ID      00-01   Sync bytes ($87CD)
M$Size    02-03   Module size
M$Name    04-05   Offset from start to module name
M$Type    06      Type / language nibbles
M$Revs    07      Attributes / revision nibbles
M$Parity  08      Header parity
          .....   Rest of header, program, and CRC value.
```

INIT MODULE

```
          00-08   Universal header
Maxmem    09-0B   Top of free memory
PollCnt   0C      IRQ polling table max entry count
DevCnt    0D      Device table max entry count
InitStr   0E-0F   Startup module name offset     ('CC3GO')
SysStr    10-11   Default device name offset     ('/D0')
StdStr    12-13   Standard I/O pathlist          ('/TERM')
BootStr   14-15   Bootstrap module name          ('Boot')
ProtFlag  16      Write-protect enable flag
          .       Name strings
```

PROGRAM MODULE

```
          00-08   Universal header
M$Exec    09-0A   Execution entry offset
M$Mem     0B-0C   Data memory size required
          .       Program
```

SUBROUTINE MODULE

```
          00-08   Universal header
M$Exec    09-0A   Subroutine entry point  (may be elsewhere)
M$Mem     0B-0C   Stack space required  (optional for pgm use)
          .       Subroutine(s)
```

```
===============================================================
                         FILE MANAGER
===============================================================

         00-08   Universal header
M$Exec   09-0A   Offset to Execution Entries Table
                 Name string, etc.

                 Execution Entries Table   (all LBRA xxxx)
FMCREA   00-02   Create new file
FMOPEN   03-05   Open file
FMMDIR   06-08   Make directory
FMCDIR   09-0B   Change directory
FMDLET   0C-0E   Delete file
FMSEEK   0F-11   Seek position in file
FMREAD   12-14   Read from file
FMWRIT   15-17   Write to file
FMRDLN   18-1A   Read line with editing
FMWRLN   1B-1D   Write line with editing
FMGSTA   1E-20   Get file status
FMSSTA   21-23   Set file status
FMCLOS   24-26   Close file
                 File manager program


===============================================================
                         DEVICE DRIVER
===============================================================

         00-08   Universal header
M$Exec   09-0A   Offset to Execution Entries Table
M$Mem    0B-0C   Static storage required
M$Mode   0D      Driver mode capabilities
                 Name string, etc.

                 Execution Entries Table   (all LBRA xxxx)
D$INIT   00-02   Initialize device
D$READ   03-05   Read from device
D$WRIT   06-08   Write to device
D$GSTA   09-0B   Get device status
D$PSTA   0C-0E   Put device status
D$TERM   0F-11   Terminate device
                 Device driver program
```

==
DEVICE DESCRIPTOR
==

	00-08	Universal header
M$FMgr	09-0A	File manager name offset for this device
M$PDev	0B-0C	Driver name offset
M$Mode	0D	Device capabilities
M$Port	0E-10	Device extended address
M$Opt	11	Number of options in initialization table
M$DTyp	12	Device type 0=SCF 1=RBF 2=PIPE 4=NFM
	13-	Initialization table (copied to path desc)
	..	Name strings

VIDEO DISPLAY CODES

```
All codes are hex (natch) and are sent to the desired device window.
    (see also pages 20 on, in September 86 RAINBOW for examples)
Parameters with H** L** parts are the High (msb) and Low (lsb) bytes.
Device windows are the /Wx's, overlay windows go within device windows.
Visible screens will change to the one containing the current active window.
   (each displayable screen can have several windows in it)
------------------------------------------------------------------------------
DWSET     1B 20 STY CPX CPY SZX SZY PRN PRN (PRN)
   Device Window Set - set up a device window (/Wx)

DWEND     1B 24
   Device Window End

SELECT    1B 21
   Select Active Window - send this code to the device window whose screen you
                wish to become visible and the new active keyboard user.

OWSET     1B 22 SVS CPX CPY SZX SZY PRN PRN
   Overlay Window Set - set up an overlay window within a device window

OWEND     1B 23
   Overlay Window End

CWAREA    1B 25 CPX CPY SZX SZY
   Change Window Area - changes active window portion

Notes:
 /Wx     - up to 31 windows, plus /W and /TERM
 CPX CPY - starting char col & row
 SZX SZY - size in rows & cols
 PRN     - palette register number (00-0F)
 SVS     - save switch (0=no, 1=yes) to save data under OW
 STY     - window screen type
            0 = current type: allows multiple windows in a screen
            1 = 40x24 text
            2 = 80x24 text
            5 = 640x192 two  color gfx
            6 = 320x192 four color
            7 = 640x192 four color
            8 = 320x192 sixteen color

------------------------------------------------------------------------------
DEFGPBUF  1B 29 GRP BFN HBL LBL
   Define Get/Put Buffer - preset a buffer size

KILBUF    1B 2A GRP BFN
   Kill Buffer - return buffer to free mem

GPLOAD    1B 2B GRP BFN STY HSX LSX HSY LSY HBL LBL DATA...
   Get/Put Buffer Load

GETBLK    1B 2C GRP BFN HBX LBX HBY LBY HSX LSX HSY LSY
   Get Graphics Block
```

```
PUTBLK    1B 2D GRP BFN HBX LBX HBY LBY
    Put Graphics Block


-

 Notes:
    GRP     - Get/Put Buffer Group Number 00-FE
    BFN     - Get/Put Buffer Number 01-FF (within Group)
    HBL/LBL - 16 bit length
    -SX -SY - size X Y
    -BX -BY - buffer X Y
```

Get/Put Groups and their Buffer subsets are used to store screen data, fonts, and pattern ram info.

Certain Group numbers are pre-defined as reserved, or as fonts, patterns, etc. Within those Groups, specific Buffer numbers are set aside.

For your own use, you should do an F$ID call to get your process id, kill the group, then open it for your use. This keeps things separated.

```
  The standard Groups and Buffers within those groups:
    C8 - fonts                01 - 8x8 font
                              02 - 6x8 font
                              03 - 8x8 gfx

    C9 - clipboards

    CA - pointers             01 - arrow
                              02 - pencil
                              03 - large cross-hair
                              04 - wait
                              05 - stop!
                              06 - text )(
                              07 - small cross-hair

CB - patterns ( 2 color)      01 - dot
CC - patterns ( 4 color)      02 - vertical lines
CD - patterns (16 color)      03 - horz lines
                              04 - cross-hatch
                              05 - left slant
                              06 - right slant
                              07 - small dot
                              08 - big dot

 ---------------------------------------------------------------------------
 PSET      1B 2E GRP BFN     Pattern Set - select buffer as pattern ram array
 LSET      1B 2F LCD         Logic   Set - select mode for pattern display
                                 0 = store data on screen as is
                                 1 = AND pattern data w/screen data
                                 2 = OR      "
                                 3 = XOR     "
```

```
--------------------------------------------------------------------------
DEFCOLR    1B 30             Default Color Reset
PALETTE    1B 31 PRN CTN     Set Palette Reg
FCOLOR     1B 32 PRN         Foreground Color - use palette # PRN
BCOLOR     1B 33 PRN         Background Color - use palette # PRN
BORDER     1B 34 PRN         Border Color     - use palette # PRN

Notes:
                 CTN         color (00-3F RRRGGGBBB xslated by monitor type)

--------------------------------------------------------------------------
SCALESW    1B 35 BSW        Scaling - 01 = drawing is relative to window size
DWPROTSW   1B 36 BSW        Window Protect Switch (boundary detection)
GCSET      1B 39 GRP BFN    Set source of Graphics Cursor data
FONT       1B 3A GRP BFN    Select Font - previously loaded into buffer
BCHRSW     1B 3B BSW        Block Char - draw char font as full char block
TCHRSW     1B 3C BSW        Transparent Char - draw char dots only
BOLDSW     1B 3D BSW        Boldface Char
PROPSW     1B 3F BSW        Proportional

Notes:
                 BSW        option switch (00=off, 01=on)

--------------------------------------------------------------------------
CURSOR     1B 40 HBX LBX HBY LBY        RCURSOR  1B 41  (Relative Coords)
POINT      1B 42 HBX LBX HBY LBY        RPOINT   1B 43 - use relative coords
LINE       1B 44 HBX LBX HBY LBY        RLINE    1B 45  HBXo LBXo HBYo LBYo
LINEM      1B 46 HBX LBX HBY LBY        RLINEM   1B 47  for these cmds
BOX        1B 48 HBX LBX HBY LBY        RBOX     1B 49  ...
BAR        1B 4A HBX LBX HBY LBY        RBAR     1B 4B  ...
PUTGC      1B 4E HBX LBX HBY LBY
FFILL      1B 4F
CIRCLE     1B 50 HBR LBR
ELIPSE     1B 51 HBRx LBRx HBRy LBRy
ARC        1B 52 HBRx LBRx HBRy LBRy HX01 LX01 HY01 LY01 HX02 LX02 HY02 LY02
RARC       1B 53 HBRxo  "    "  etc

--------------------------------------------------------------------------
Other Terminal Codes:

HOME          01                  ERASEEOS        0B
GO XY         02                  CLSHOME         0C
ERASELINE     03                  RETURN <CR>     0D
ERASEEOL      04                  REVERSEON       1F 20
CURSOROFF     05 20               REVERSEOFF      1F 21
CURSORON      05 21               UNDERLINEON     1F 22
RIGHT         06                  UNDERLINEOFF    1F 23
BELL          07                  BLINKON         1F 24
LEFT          08                  BLINKOFF        1F 25
UP            09                  INSLINE         1F 30
DOWN          0A                  DELLINE         1F 31
```

```
==============================================================
          Keyboard Definitions with Hex Values
==============================================================

NORM   SHFT   CTRL       NORM   SHFT   CTRL       NORM   SHFT   CTRL
--------------------------------------------------------------
0 30   0 30   _ 1F       @ 40     60   NUL 00     P 50   p 70   DLE 10
1 31   ! 31   ‾ 7C       A 41   a 61   SOH 01     Q 51   q 71   DC1 11
2 32   " 22     00       B 42   b 62   STX 02     R 52   r 72   DC2 12
3 33   # 23     7E       C 43   c 63   ETX 03     S 53   s 73   DC3 13
4 34   $ 24     00       D 44   d 64   EOT 04     T 54   t 74   DC4 14
5 35   % 25     00       E 45   e 65   EMD 05     U 55   u 75   NAK 15
6 36   & 26     00       F 46   f 66   ACK 06     V 56   v 76   SYN 16
7 37   ' 27   @ 5E       G 47   g 67   BEL 07     W 57   w 77   ETB 17
8 38   ( 28   [ 5B       H 48   h 68   BSP 08     X 58   x 78   CAN 18
9 39   ) 29   ] 5D       I 49   i 69   HT  09     Y 59   y 79   EM  19
: 3A   * 2A     00       J 4A   j 6A   LF  0A     Z 5A   z 7A   SUM 1A
; 3B   + 2B     7F       K 4B   k 6B   VT  0B
, 2C   < 3C     7B       L 4C   l 6C   FF  0C      BREAK   05 03 1B
- 2D   = 3D   _ 5F       M 4D   m 6D   CR  0D      ENTER   0D 0D 0D
. 2E   > 3E   ‾ 7D       N 4E   n 6E   CO  0E      SPACE   20 20 20
/ 2F   ? 3F   \ 5C       O 4F   o 6F   CI  0F      LEFT    08 18 10
                                                   RIGHT   09 19 11
<CLR><0> = shift u/l case                          DOWN    0A 1A 12
                                                   UP      0C 1C 13
```

The only new key code generated is the 7F rubout key.
<control>-;

System Error Codes

001		01	Exit
002		02	Keyboard abort
003		03	Keyboard interrupt
200	E$PthFul	C8	Path Table full
201	E$BPNum	C9	Bad Path Number
202	E$Poll	CA	Polling Table Full
203	E$BMode	CB	Bad Mode
204	E$DevOvf	CC	Device Table Overflow
205	E$BMID	CD	Bad Module ID
206	E$DirFul	CE	Module Directory Full
207	E$MemFul	CF	Process Memory Full
208	E$UnkSvc	D0	Unknown Service Code
209	E$ModBsy	D1	Module Busy
210	E$BPAddr	D2	Bad Page Address
211	E$EOF	D3	End of File
212		D4	Attempt to return memory not assigned
213	E$NES	D5	Non-Existing Segment
214	E$FNA	D6	File Not Accessable
215	E$BPNam	D7	Bad Path Name
216	E$PNNF	D8	Path Name Not Found
217	E$SLF	D9	Segment List Full
218	E$CEF	DA	Creating Existing File
219	E$IBA	DB	Illegal Block Address
220	E$HangUp	DC	Carrier lost
221	E$MNF	DD	Module Not Found
222		DE	Sector out of range
223	E$DelSP	DF	Deleting Stack Pointer memory
224	E$IPrcID	E0	Illegal Process ID
225		E1	
226	E$NoChld	E2	No Children
227	E$ISWI	E3	Illegal SWI code
228	E$PrcAbt	E4	Process Aborted
229	E$PrcFul	E5	Process Table Full
230	E$IForkP	E6	Illegal Fork Parameter
231	E$KwnMod	E7	Known Module
232	E$BMCRC	E8	Bad Module CRC
233	E$USigP	E9	Unprocessed Signal Pending
234	E$NEMod	EA	Non Existing Module
235	E$BNam	EB	Bad Name
236	E$BMHP	EC	Bad module header parity
237	E$NoRam	ED	No Ram Available
238	E$BPrcID	EE	Bad Process ID
239	E$NoTask	EF	No available Task number
240	E$Unit	F0	Illegal Unit (drive)
241	E$Sect	F1	Bad SECTor number
242	E$WP	F2	Write Protect
243	E$CRC	F3	Bad Check Sum
244	E$Read	F4	Read Error
245	E$Write	F5	Write Error
246	E$NotRdy	F6	Device Not Ready
247	E$Seek	F7	Seek Error
248	E$Full	F8	Media Full
249	E$BTyp	F9	Bad Type (incompatable) media
250	E$DevBsy	FA	Device Busy
251	E$DIDC	FB	Disk ID Change
252	E$Lock	FC	Record is busy (locked out)
253	E$Share	FD	Non-sharable file busy
254	E$DeadLk	FE	I/O Deadlock error

INSIDE OS9 LEVEL II

INDEX

INSIDE OS9 LEVEL II

INDEX

Congratulations on your purchase of the first of what we hope will be many books on the most powerful operating system available, OS-9!

This copy of 'INSIDE OS9 LEVEL II' will be upgraded as time goes by and as more information is collected.

To get your discounted copy of the next edition of 'INSIDE OS9 LEVEL II' fill out the coupon below and mail it in.

COUPON

This coupon entitles the bearer to edition 2 of 'INSIDE OS9 LEVEL II' when it becomes available for 50% of the cover price of this copy. The price of $20.00 includes COD UPS charges. This coupon is good for shipment in the USA only.

Name____

Address(no PO BOX)____________________________

City____________________State__________ZIP________

Date______________

Mail to:

Frank Hogg Laboratory, Inc.
770 James Street
Syracuse, New York 13203

Frank Hogg Laboratory is the largest supplier of software for OS9. We have also been supporting OS9 longer than any other company. If you would like to get on our mailing list to receive free newsletters and catalogs please fill out the coupon below and mail it in.

Thank You

Frank Hogg Laboratory, Inc.

770 James Street
Syracuse, New York 13203