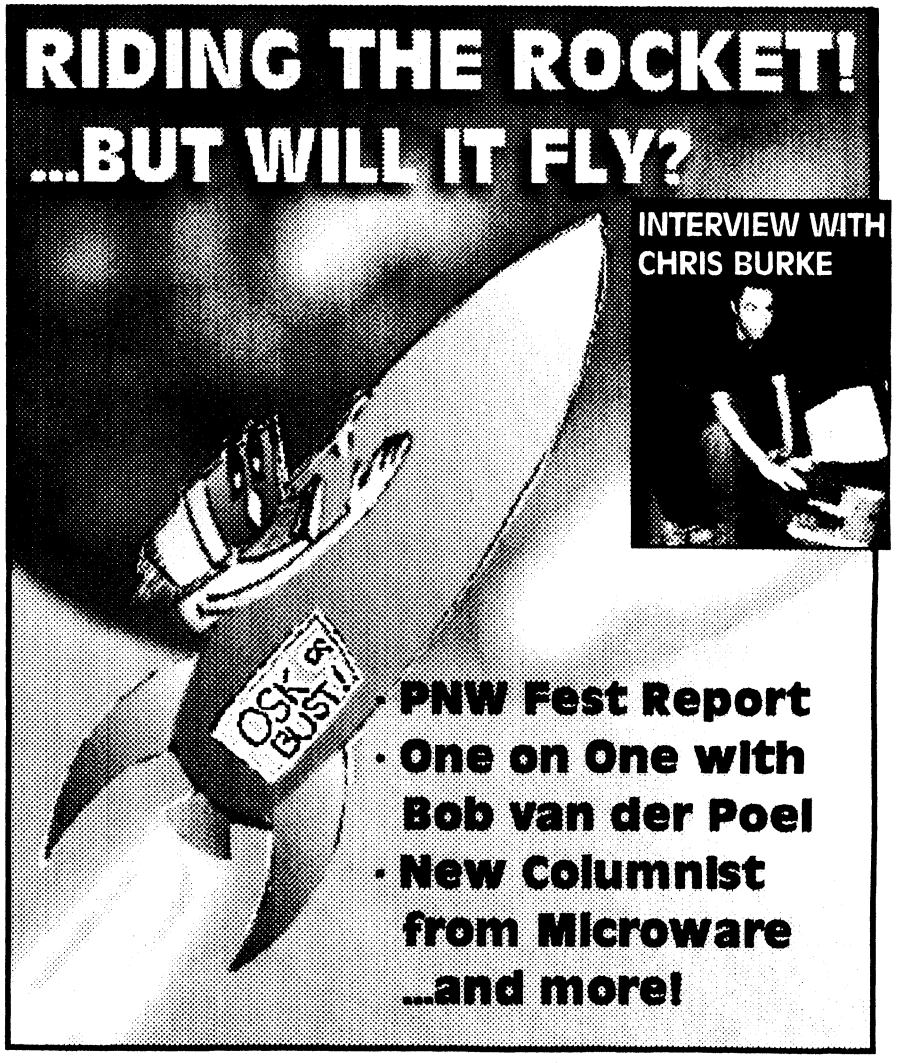
Volume 1, Issue 8

$3.00

The "International" OS9 Underground®

A Fat Cat® Publication

Magazine Dedicated to OS-9/OSK Users Everywhere!



New Lower Prices!
from ColorSystems

Variations of Solitaire

Includes FIVE Variations, Pyramid, Klondike, Spider, Poker and Canfield. Complete documentation shows how to create your own games boot disk using the special menu program which is included.

CoCo3 Version $29.95
MM/1 Version $39.95

WPShel

A Word Processing Oriented Point and Click Shell for all your word processing needs. Requires WindInt from your Multi-Vue Disk. Does not include Editor, Formatter or Spelling Checker.

CoCo3 Only! $20.00

NEW!

Using AWK With OS-9

A description of the AWK Programming Language with an emphasis on GNU AWK for OSK. Includes the latest version of GNU AWK.

OSK Only! Just $19.95

OS-9 Game Pack

Includes FIVE complete games, Othello, Yahtzee, Minefield, KnightsBridge and Battleship. Includes special menu program and step by step instructions on creating your own games boot disk.

CoCo3 Version $29.95
MM/1 Version $39.95

All CoCo3 Programs require at least 256K of memory.

Coming SOON! Indexed Files for OS-9 Level 2, OS-9/68000 and OS-9000!

We accept Personal Checks or Money Orders drawn from US Banks or International Postal Money Orders. NC residents please add 6% Sales Tax. Call or write for a FREE catalog!

Please add $3 per item for shipping outside of the Continental United States.

ColorSystems

P. O. Box 540
Castle Hayne, NC 28429
(919) 675-1706

Quality OS-9 Software for the Color Computer 3 and the MM/1 from IMS

CoNect
New Hardware

Mini-RS232 Port: If you are in the market for a Tandy-compatable serial port, check out our Mini! This ROMPak sized unit supports all seven control lines available, and can supply more output current than even the Tandy Pak! Jumper selectable address and cd swap.
Only $49.95 (Y Cable use requireds 12 volt power supply, add $9.95)

XPander: The XPander allows you to assemble the most compact CoCo3 system possible using a stock motherboard. For example, the electronics of my 2 Meg CoCo3 with Tandy Floppy Controller, Burke and BurkeXT, WD1002 Hard Drive controller, rs232 port, Puppo and Hi-Res adaptors forms a block 12 inches long, 7 inches deep, and 3 inches at its highest point. Not only will this fit in even the smaller PC cases, but in a modified CoCo case.

Obviously this is not a full tilt MultiPak clone- there just isn't room. The two internal slots may both contain /scs decoded devices, but only one slots ROM may be used. The external slot may be used either as a ROMPak port (disables internal hardware when Pak is inserted), or as an undecoded buss slot. 12v is available at all slots.

The no-slot RS232 port is a virtual clone of the mini-rs232 described above, and saves not only a slot but quite a bit of room in the finished package.

The XPander is available in two versions. If a PC type case/power supply will be used, order just the board. CoCo Kit includes a new lower case shell and 450ma +/-12v power supply.
CoCo Kit $124.95 **Board Only $99.95**

CoNect Custom CoCo Cables!

Cassette- (mini or rca)	$4.95	Comp. Video (6 feet)	$3.95
Disk Power (either way)	$5.95	Printer (DIN to DIN)	$4.95
2 Drive Floppy Data	$14.95	3 Drive Floppy Data	$19.95
RS232 (db25,db9,din4 to db25,db9 - normal,nulmodem,or dcd swap)			$9.95

RAM Upgrades:

64K CoCo1 (F board), or CoCo2 (2 or 8 chip) with instructions			$7.95
512K CoCo3 (various makes) not always available			$49.95
2 Meg CoCo3 (Disto) with SIMMs	$194.95	Installed (add $11.95 for 6309)	$219.95

Hitachification Install CPU socket and 6309. 6809 returned $29.95

OS9 Underground Magazine Member Card

Participating Vendors

These vendors offer the following discounts for OS9 Underground Member Card Holders...

- CoNect ..... 10% off any order
- Sub-Etha Software ..... 10% off any order
- Canaware ..... 10% off WristSavers, 10% off WristSavers Mouse Pad & 15% off ENC9
- AniMajik Productions ..... 20% off All Software
- Farna Systems ..... 10% off any order

Software/Hardare Vendors... you can be listed here FREE! Contact The OS9 Underground for details. You need not be an advertiser for this service.

Be sure to give your card number when you place an order with these fine vendors
(Not responsible for typos or mis-prints)

Advertiser's Index

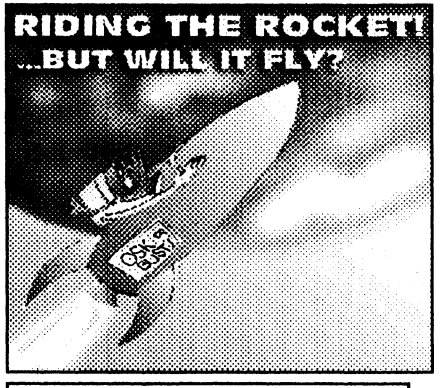
Vendor	Page
Color Systems	IFC
Peripheral Technologies	5
AniMajik Productions	5
Computer Design Services	7
JWT Enterprises	10
OS9 Underground Submissions	15
BlackHawk Enterprises	17
Sub-Etha Software	20
Farna Systems	21
Bob van der Poel Software	22
OS9 Underground Subscriptions	25
Disto	28
CoNect	IBC
Delmar Co	BC

The "International" OS9 Underground® Magazine

Dedicated to OS-9/OSK Users Everywhere

Volume 1, Issue 8

CONTENTS



Don't Let This Happen! (Editorial by Alan Sheltra) 4

Under it All (Editor's Column) 6

Software Engineering
Scalar Data Types
by Leonard Cassady 8

BASIC Training
The END or the BEGINNING?
by Jim Vestal 18

Dealing with Memory Protection
by Bob van der Poel 19

Writing a Device Driver
by Boisy Pitre 23

One on One with Bob van der Poel
Interview by Richard Albers 26

The Man Behind the Rocket
Interview by Richard Albers 29

PNW Fest Report! 33

Chicago Fest Pictures 36

Ad Index and Member Card Listing 38

The Underground Staff

Editor/Publisher: Alan Sheltra

Assistant Editors: Jim Vestal

Associate Editor: Steve Secord

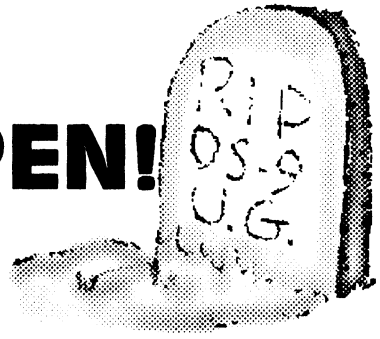
Contributing Editors: Leonard Cassady, Andy DePue, Wes Gale, Scott Griepentrog, Allen Huffman, Scott McGee, Boisy Pitre, Bob van der Poel

Art and Typesetting: Alan Sheltra

Printing: Pink Printers

 Subscription rates are $18.00 for 12 issues ($23.00 Canada, $27.00 overseas US Funds). Single or back-issues are available at the cover price (please call or write for availability). Fat Cat Publications is located at 4650 Cahuenga Blvd., Ste #7, Toluca Lake, CA 91602 • (818) 761-4135 (Voice), (818) 365-0477 (Fax) or (818) 769-1938 (Modem).

DON'T LET THIS HAPPEN!



Commentary by Alan Sheltra

It's a damn crime and I am pissed! (Pardon my French). I have been following the recent events on both the networks and have been in touch almost daily by those very close to this. I have really tried to stay out of this, but I can't. This newest turn of evens just sickens me. But first let's step back a bit give this a little background.

Some Background

The newly formed OS-9 Users Group really started something good and accomplished a great deal. I refer to the User Group under the presidency of Boisy Pitre.

Since Boisy and many of the officers of the OS-9 UG had to step down, for various reasons. Boisy and Scott McGee, because conflicts of interest with their employer, Microware, others because their term of office was up (April 1st, 1993). I served as editor of the OS-9 UG newsletter, the MOTD for that year.

Prior to the April 1st letters were sent out to all members to vote for new officers. Only 2 or 3 ballots were returned out the 125+ that were sent out to members. A pretty dismal response indeed.

Well, it left Boisy no choice but to find someone to take over the group. One person did take it, and the group, it seemed was well in it's way in it's second year.

Back to the Present...

The current OS-9 User Group President, (Jim DeStafeno) has sent out a letter to User Group members. He has in this letter decided to resign as UG President (which is fine), but has also decided (in his mind) that there is no more need for the OS-9 Users Group. I have a one word rebuttal to that... **Bull Puckey!** If he wants to step down or can't handle the job, that's one thing, but to advocate killing the UG is not showing the mark of a true leader.

Then Jim decided that he would pick a new set of officers from qualified persons wanting the job. Several excellent people volunteered. This would certainly allow Jim to step down and pass the group on, since he didn't want to carry through with his commitments.

Well, that never happened. Just recently, a second letter was sent out to all members stating that the User Group was dissolved and returned the unused portion of the dues to each user. Who gave this man the right to single-handedly dissolve the UG?!! Excuse me Jim, what is color of the sky on your planet?

Never in OS-9's history have we had more need for a National (or International) Users Group and we can't just let it die the way Mr. DeStafeno would like it to.

Where do we go from here...

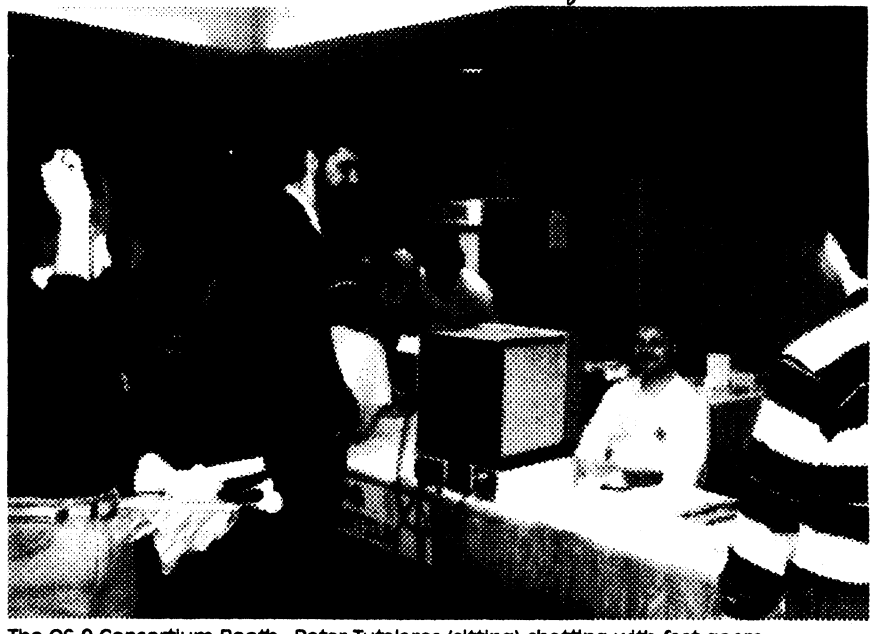
Sending comments in mail or email to Jimmy-D would certainly do no good.

In my opinion, I feel the watch-dog directors of the UG should step in and hand the Group over to a will able body (There are several will and able) and make Jim pay back all UG monies..

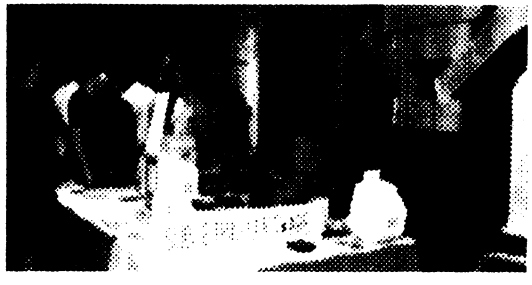
In the meanwhile, I would urge all members to hang in there. A new leader will step forward soon, and the OS-9 UG will continue once again. It MUST! We need the UG! ... and good riddance Jimmy-D.

I normally don't go on tirades like this in print, but what this man has done to UG members is criminal. Your comments to this are invited.

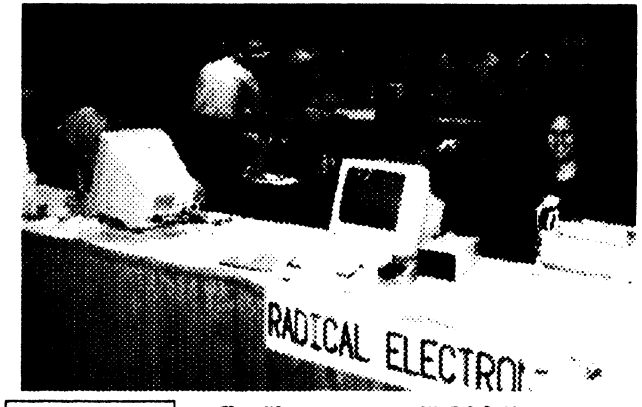
-Alan Sheltra



The OS-9 Consortium Booth. Peter Tutalares (sitting) chatting with fest-goers.



The CoNect, Farna Systems and Sub-Etha booths

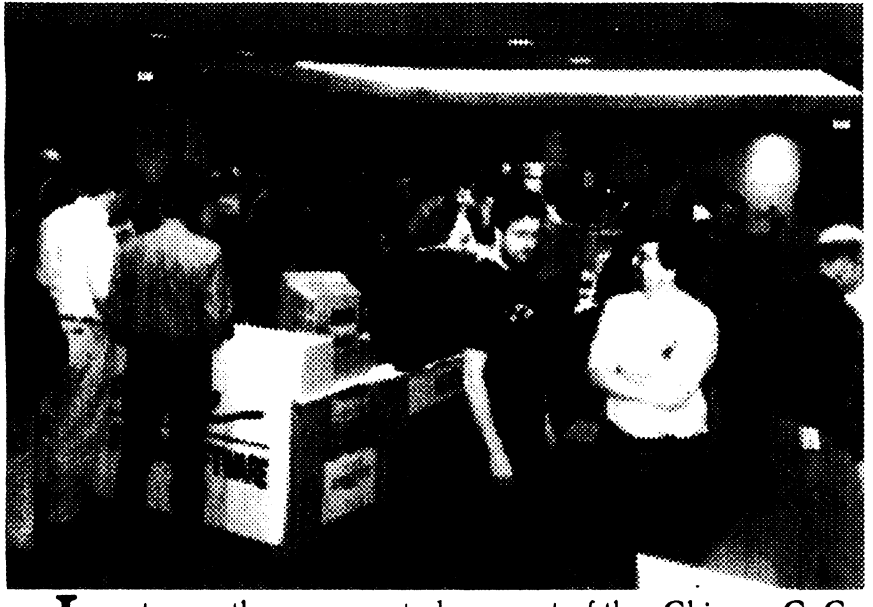


Al Bayco of Radical Electronics (Smiling at camera). This was the first time attending this show.

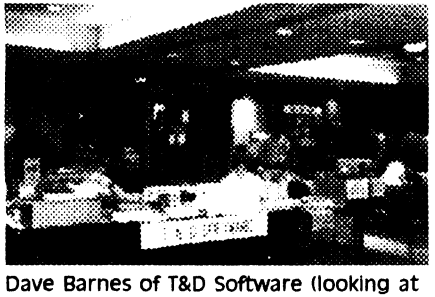
and more pictures from...

The Chicago CoCo Fest:

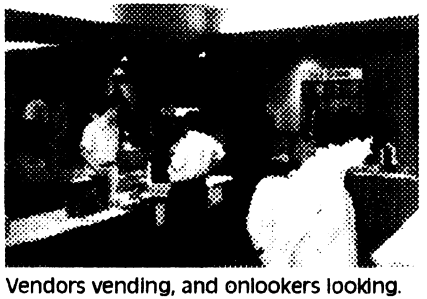
(Pictures Courtesy of Allen Huffman)



Last month we presented a report of the Chicago CoCo Fest, but were not able to get the pictures in time before we went to press. We'd like to present a few of those that we received with you now.



Dave Barnes of T&D Software (looking at camera)



Vendors vending, and onlookers looking.

Under it All...

Alan Sheltra

Two New Columnists

Both Boisy Pitre and Scott McGee have joined the staff at the Underground. See Boisy's article "*Writing a Device Driver*" in this issue. Scott's will be writing a monthly column staring in next month's issue.

Boisy, as some of you may know was the OS-9 User Group President from the previous term. He single-handedly brought together several good solid OS-9 people and re-formed the new OS-9 UG from the ashes of the old (Carl and Debi Krieder, Scott McGee and later, Zack Sessions and myself. If nothing else, Boisy will be remembered for being responsible for lowering the price of OS-9000 for User Group members for 1/3 of it's price. Welcome aboard Boisy!

BASICally Speaking

We also have a another B09 writer who will be joining the Underground staff soon. This should make some of you BASIC09 hounds happy! Eric Levinson has written for "The OSKer" and is a co-writer of the RCIS net software. At least Jim Vestal won't feel so lonely writing about B09 now. ☺

Surgery for Leonard

Leonard Cassady, who writes our Software Engineering column recently had major surgery. He is fine now and recuperating at home in Canoga Park, CA With nothing to do but get well, I know Leonard will be writing up a storm for future issues. Glad to hear you're doing well Leonard.

Comments and Questions...

As always, your comments and questions are always welcome (good or bad) in our letter to the Editor column. But staring next month Leonard Cassady, as well as writing his regular Software Engineering column, will also be fielding questions for our revised "OS-9: The Q&A", question and answer column.

Keep those questions coming. Drop us a postcard or letter to:

The OS9 Underground
OS-9: The Q&A
4650 Cahuenga Blvd., Ste #7
Toluca Lake, CA 91602

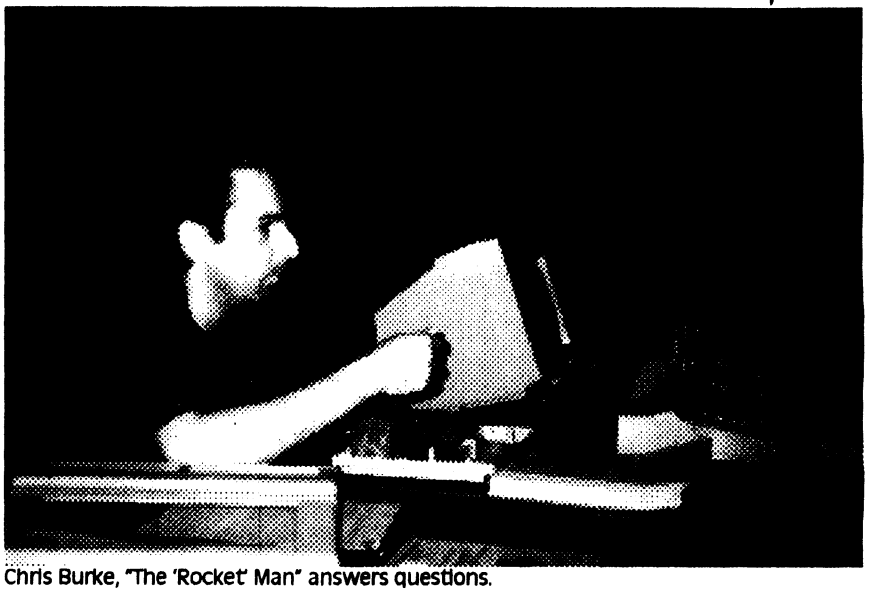
or email to:

ZOGster@delphi.com

The "Mail Room" will also return next month. I took over those pages for my comments this month.

Till next month,
"Keep on hacking in Real Time!"

Editor/Publisher

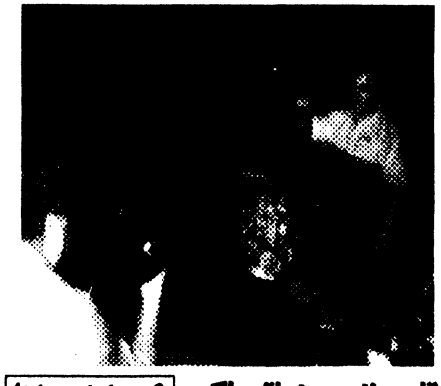


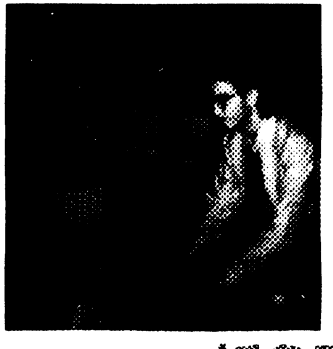
Chris Burke, "The 'Rocket' Man" answers questions.

Bob van der Poel (Left) takes break to listen to seminar.

Mike Bussmantalks about databasing with the Mac.

Andre LaVelle (Right) of SBUG (South Bay Users Group) listens in.

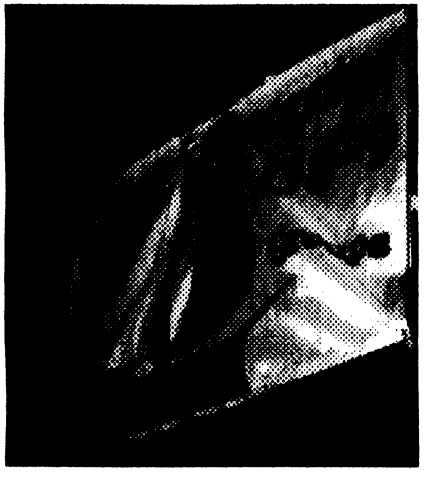


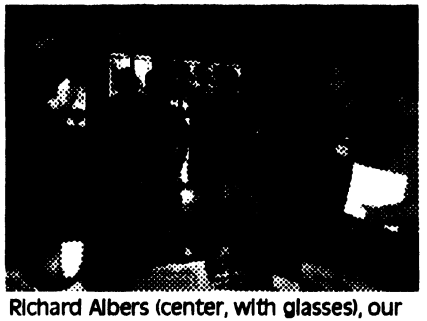


The attendance was slightly better than the previous year according to Donald and the computer swap meet raised quite a bit of change for the Computer Bank Charity. The Computer Bank Chirity donates used equipment to needy individuals or worthwhile organizations. A very good cause.

Everyone who attended had a good time and got a chance to meet old friends.

Chris Johnson demos the Kix30.





Richard Albers (center, with glasses), our Underground Roving Reporter listens in.

John Schliep (Right), who was selling 6309's, takes a lunch break.

C Software Engineering

by Leonard Cassady

SCALAR DATA TYPES

At the byte and bit level, the computer may not understand the differences between integers, characters, and floating-point numbers, which comprise the different data types. The term, "data type", is at best, ambiguous. A "data type" is really an interpretation of a string of bits.

INTERNAL REPRESENTATION

Computers, or at least the majority of them, use "base 2" or "binary" to internally represent data as a string of two-valued quantities. Each single quantity is known as a "bit" or (B)inary (D)ig(T). This represents the smallest value computers understand. Sometimes known as the "native language" of computers, the hardware uses binary to implement data and instructions to the CPU, memory, and I/O devices.

A string of bits, organized into groups of four, are known as a "nibble". A "byte" is comprised of eight bits or two nibbles. The largest number that may be represented by a byte is decimal 255. Numbers bigger than 255 use two or more bytes, and the common combinations are two bytes, (16 bits), or a "word", and four bytes, (32 bits), known as a "long word".

Binary (nibble)	Octal	Decimal	Hex
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

OPERATIONS ON BINARY NUMBERS

The same operations that may be performed on decimal numbers, may also be performed on binary numbers. The computer's ALU, (A)rithmetic and (L)ogic (U)nit, which is a major component of the CPU, is capable of performing a number of simple operations including:

One's Complement:

Simply invert the value of all the binary bits. All 0's become 1's and vice versa.

original number: 0000 0001 ($01 hex)

one's complement: 1111 1110 ($FE hex)

Binary *AND*:

The AND operation is commonly used to obtain the a remainder for a division by the power of 2 minus 1. It produces a third binary number where the original numbers BOTH had 1's.

Pacific North West Computer Festival

(Pictures Courtesy of Donald Zimmerman)

Chris Burke, "The 'Rocket' Man", shows off his prototype to Fest-goers in Port Orchard. The "Rocket", an add-on hardware board, will allow a CoCo to run OSK.

The PNW Fest

While everyone in the OS-9 Community knows about the Chicago and Atlanta Fest, the West has it's Fest too. The Pacific North West Fest, held June 25th and 26th, 1993 in the resort town of Port Orchard, Washington.

The PNW Fest, while not strictly a CoCo Fest shared it's attendance with speakers and attendees from other computer platforms.

Most notable for the OS-9 and CoCo community were seminars given by Chris Burke, of Burke & Burke, and Bob van der Poel (Our very own contributing editor), Rodger Alexander (Editor of "The OS-9 Newsletter") and Allen Morgan.

The show was a success according to the show's presenter, Donald Zimmerman. Donald and the Port O' CoCo Users Group, the hosts of the fest has definate plans to return next year. June 24 and 25 1994 (Satuday and Sunday) are the scheduled dates

```
original number: 0000 0001  ($01 hex)

      ANDing: 1111 1110  ($FE hex)

              0000 0000  ($00 hex)
```

In this example neither number has 1's in the same bit position, so the result of the operation is zero.

To find the remainder when $3D hex is divided by $8 hex, AND with the power of 2 minus 1, ($08 - $01 = $07). The result is $05.

```
Decimal Example:

61 / 8 = 7  with a remainder of 5.

Binary Example:

original number: 0011 1101  ($3D hex)

      ANDing: 0000 0111  ($07 hex)

              0000 0101  ($05 hex)
```

Binary *OR*:

The OR operation takes two binary numbers and produces a third binary number that has a 1 where EITHER of the original numbers had a 1.

```
original number: 0000 0001  ($01 hex)

      OR-ing: 1111 1110  ($FE hex)

              1111 1111  ($FF hex)
```

Binary *XOR*:

The XOR, e(X)clusive OR), operation takes two binary numbers and produces a third number that has 1's in the bit positions where ONLY one, of the original numbers have a 1. "XOR-ing" a number with itself produces zero.

```
original number: 0000 0010  ($02 hex)

      XORing: 1111 1110  ($FE hex)

              1111 1100  ($FC hex)
```

Binary Addition:

Binary Addition is similar to decimal addition. The difference is simply the result carry of 1 to the next column, which is determined by the base value system. In decimal, the carry occurs after the column value exceeds 9. In binary, this carry occurs after the column value exceeds 1.

```
original number: 0000 0010  ($02 hex)

      plus  + 0000 1111  ($0F hex)

              0001 0001  ($11 hex)
```

Two's Complement:

The two's complement is obtained by taking the one's complement of number and adding 1 to it.

```
   original number:  0000 0001  ($01 hex)

one's complement:  1111 1110  ($FE hex)
        add one: + 0000 0001  ($01 hex)

two's complement:  1111 1111  ($FF hex)
```

Adding the two's complement of a number is the same as subtracting the number.

Two's Complement Addition:

```
original number:    0000 0010  ($02 hex)
two's complement
    of $01 hex: + 1111 1111  ($FF hex)

                  0000 0001  ($01 hex)
```

The carry out of the high order bit is discarded as all numbers kept in a computer have exactly the same number of bits, (in this case eight bits).

To represent negative numbers internally, the computer uses "two's complement notation". The leftmost bit, called "MSB" or (M)ost (S)ignificant (B)it because it represents the

There is one report that the OS-9 kernel would be in these ROMs. That's not true.

14 mhz. The Rocket board will run at up to 16.7 mhz and to do that you will need 70 ns memory and no wait states. We're running it at 14 mhz, no wait states and 100 ns memory. Someone else that wants to put fast memory on it can change the crystal and change the memory out and take care of that.

AM That would be the only change required?

CB Yeah, that's the only change. There's no other change, there's nothing timing dependent on the board at all, so all you do is change the crystal and drop the new SIMMs on and you'd have 15% faster throughput. I thought of selling the board that way for $50 more or something, but people can do that themselves. I think that's a good overview of the hardware.

AM You said you had a socket for adding a chip on back there?

CB Yes, the way it's set up it plugs into the 6809 socket on the CoCo and then it will actually bring all 40 pins up to another socket on top of the Rocket. You can plug the 6809 or 6309 in there and there's a 2 pin header which you can either put a jumper on or you can put a switch on it. When the switch is closed, you will disable the rocket circuitry and enable the 6809 or 6309 and it will then act just like a stock CoCo3. The 68000 will be completely off line. Actually, the 68000 will still be running, it will just be kicked off the CoCo buss.

RA Essentially running in a NOP loop.

CB Although, if you were processing and not actually processing the CoCo box, it could still run.

RA You mean if you actually start a process running in 68000 and use a software switch and fire up the 6809...

CB Well, a software switch, probably not, but if you wanted to leave the 68000 running to do something else, I don't know.

RA Even a "bouncy" hardware switch would be okay.

CB It affects that the tri-state control so they don't really recover from it. The point is, is that the 68000 is really an independent computer and the CoCo is just a peripheral, so as long as you don't have to access that peripheral the 68000 ideally will just keep chugging along. We're not doing any special hardware or software to support that, it's just kind of a description of the circuit. So you have your processor and you can use it like a stock CoCo or you can use it as a 68000 and disable the 6809

But really, all you'd need is 128K of memory on your CoCo mother board to run the Rocket.

The memory on the CoCo mother board, since it's accessed through the GIME and all that and so on, you can use it for your graphic screens. That's where all your graphics windows would be stored and you could put a ram disk or something on there also. If you had the extra memory, or you could put the ram disk in the main 68000 memory, which would be a little faster. But really, all you'd need is 128K of memory on your CoCo mother board to run the Rocket. You could run it with more and you'd have more room for graphic screens, but 128K is certainly enough.

AM What is in these EPROMs?

CB The EPROMs have boot-up code which just initializes the hardware. They will also have the 6809 and 6309 simulator in them. When the simulator is running it will be actually copied from the ROM into the RAM because it will run faster. That will be stored in there and so will utility routines be stored in there and a boot loader that will do 2 different kinds of boots. One is, you just put something in the floppy controller that says "DK", BASIC will transfer out to that automatically and run it. If you put something in the floppy controller that says "RK", this will kick out to that and start executing 68000 code. So you put your own boot ROM in your floppy controller, also it will go out and do a floppy disk boot. It will boot OS-9 or anything else you want to run off of floppy

There is one report that the OS-9 kernel would be in these ROMs. That's not true. If you need to patch your kernel, or whatever, you can do that because it will be loaded off the disk.

(EOF)

standard 68000 2 serial port and timer peripheral. Then it has a completely programmable DRAM controller with programmable zero or 1 wait state and a MUX from 8 to 16 bits, so it can accommodate really any size SIMM, including some that haven't been invented yet.
It'll run 2 banks, it's full 16 bit wide memory. It also has 16 parallel I/O lines on it and all the interrupt control logic and the DTack generation for the 68000 asynchronous buss and so on, is right on that chip.
It's 132 pin chip and it's a surface mount type chip. It's got something else on it, well a device selects, it'll generate programmable device selects. The last thing of interest is that's it's got a full 32 bit address buss, whereas the straight 68000 has a 24 bit address buss, so it can access, what is it? 4 gigabytes.

RA A lot of memory

CB a lot of memory!

RA More than most CoCo's use.

CB A So it really only has 24 external address lines, but it has 32 to the DRAM. SO you can put a ton of dynamic ram on. The other things on the board are 2, eight bit wide EPROMs. They are 32K byte EPROMs so you've got 64K bytes of EPROM and a buss synchronization circuit which will match the 68000 buss up to the ENQ clocks of the CoCo...

RA That lets the GIME do it's job.

CB ...and it also lets the GIME interleave just the way it is. So this circuit makes the Rocket board just like a 6809 or a 6309 to the rest of the CoCo, including the GIME. For instance, when you're not doing a buss cycle into the CoCo, it puts all "one's" on the address buss like a 6809 would. It also makes the entire CoCo, including the GIME chip and all the on-board memory, it views it just like it would look like to the address and data lines of the 6809. If you have 256K or 512K or 1 Megabyte of memory on your CoCo, the 68000 sees that through the GIME registers, the 8K blocks and everything. Anything out on the Multi-Pak, it sees. The entire CoCo is at address FF80 XXXX. The floppy controller is at FF80 FF40, that's where the floppy controller would be. So anything that the 6809 can do to the CoCo, the Rocket board can do to the CoCo also. But because of the way memory works, we put memory onto the Rocket board also, the 2 SIMM sockets, although the chip can handle 4 SIMMs we put 2 on there, so you can put 512K total memory, or 2 Megabytes or 8 megabytes.

AM Is this an economic reason for doing that?

CB Yeah, economic and space. We could put a second set of SIMM sockets on and take up more board space, but another more important consideration is that we want this to fit in a PAL CoCo. The European CoCo has the processor mounted at right angles and since the Rocket board plugs right into the Processor socket, we have to turn the whole board 90 degrees without interfering with the keyboard or sticking out the back side, So we're trying to keep the parts count real small.
So that's kind of the hardware of it.

AM Are you using the DUARTs as one of the other features or are you talking to everything through the CoCo?

CB The Rocket board does not.... the board that we're advertising doesn't even mention that those other things are on there, the DUART and the parallel I/O. Our plan is to bring them to a single connector. SO all of those I/O lines will come to this connector and it will be the experimenter's connector. It's almost like an after market board. There's enough I/O there to put a MIDI port or 2 serial ports or a SCSI interface or whatever, but that is not part of the basic Rocket offering. It'll be other boards that we do, or that other people do as projects. We're also not planning right now, to put a buss connector on it , although if there's room we'll bring out the 16 data lines and one device select and read/write signals, and you'll be able to plug things in there. One thought was to make it look just like a Disto mini-expansion buss, only bring out 8 data lines and do something like that.
But anyway, the Rocket, again as we're advertising it, we're not promising that the buss connector will be there. It's just something we'll do if there's room. A couple of people have mentioned that there's a bottle-neck because the CoCo itself is slow, but the bottle-neck applies only during the actual cycles when you're accessing the CoCo peripherals or the peripherals in the multi-pak. All of the computation in between, for instance if you're doing graphics. It's true if you're putting a pixel on the screen, it's going to happen at 2 mhz. But calculating the coordinates of the pixel, or scaling it or rotating it or calculating the font, that will all happen at

largest value, is the SIGN BIT.

If the sign bit is set to zero, it is positive. If it is set to one, the number is negative. Without using two's complement, the range of 0 to 255 may be represented with eight bits. This is known as "unsigned" arithmetic.

Using two's complement, or "signed" arithmetic, the range of -128 to 127 may be represented.

When copying an 8-bit quantity into a 16-bit quantity, or copying a 16-bit quantity into a 32-bit quantity, there is the possibility of losing the two's complement properties of the number. The way to avoid this problem is to copy the sign bit, (MSB), into all the "extra" bits of the larger number.

This is called "sign extension". For example, if we sign extend an 8-bit quantity, $FF hex (-1 in two's complement form), into a 16-bit quantity, we get $FFFF hex (which is -1 in two's complement form):

```
two's complement 8-bit:
      1111 1111  (-1)

 sign extend to 16-bits:
      1111 1111 1111 1111  (-1 two's complement)
```

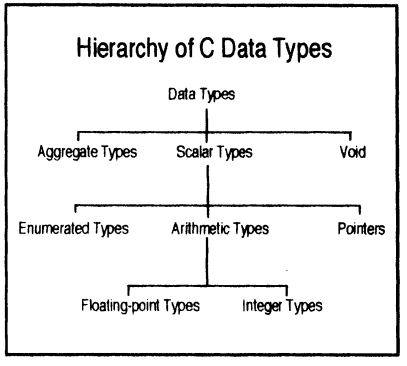
Personally, I find working with binary numbers obnoxious at best. However it is necessary to have at least a fundamental understanding of how the computer represents numbers internally and resulting side effects of certain operations on scalar types at the bit level.

C DATA TYPES

The C language offers a small, but powerful set of data types. There are two floating-point types, (three in the ANSI standard), and eight types of integers.

Integers may be decimal, octal, or hexadecimal. These data types, along with floating-point numbers are "arithmetic types". Together with pointers and enumerations, they are known as "scalar types". Scalar values are either less than, equal to, or greater than any other scalar value, because all of the values lie along a "linear scale".

In addition to "scalar types", we have "aggregate types". Aggregate types are built by combining one or more scalar type and can be arrays, structures, and unions. Aggregate types are useful for organizing related variables into physically adjacent groups. The last type is neither scalar or aggregate, but a null or "void type".



THE BASICS

There are nine reserved words for scalar data types. The first five, "char", "int", "float", "double", and "enum" - are the basic types.

The qualifiers, "signed", "unsigned", "long", and "short", modify these basic types. The variable declaration tells the compiler how many bytes should be allocated for the variable and how to interpret the bytes. While "int" is a basic data type, it has different sizes on different machines.

A byte is comprised of eight bits and with most 16-bit processors, an "int" data type is four bytes, or thirty two bits in length. The older 8-bit processors have a default "int" size of two bytes, or sixteen bits. When the size of the data type becomes important, such as porting code from a larger processor to a smaller one, we can use one of the qualifiers to modify the allocation.

On most 16-bit machines, a "long int" is fours bytes, which is the same length as the default size for an "int". Since there is no difference in length, the use of the qualifier, "long" on 16-bit machines is redundant and generally used for informational purposes. Conversely, the use of the qualifier, "short", on an 8-bit machine is not needed as the default is 2 bytes.

In the case of an 8-bit machine, the use of the qualifier "long" is important. The default size of an "int" is two bytes and the variable declaration, "long int", modifies the size to four bytes. The length in bytes of an "int" type

represents the "natural" size, or default of the computer.

DATA REPRESENTATION			
Data Type	8-Bit	16-Bit	Internal Representation
char	1 byte	1 byte	two's complement binary
unsigned char	--	1 byte	unsigned binary
short	--	2 bytes	two's complement binary
unsigned short	--	2 bytes	unsigned binary
int	2 bytes	4 bytes	two's complement binary
unsigned	2 bytes	4 bytes	unsigned binary
long	4 bytes	4 bytes	two's complement binary
float	4 bytes	4 bytes	binary floating point
double	8 bytes	8 bytes	binary floating point
"pointer to.."	4 bytes	4 bytes	address

MIXING TYPES

To make sense of an expression containing mixed scalar types, C performs conversions automatically. Sometimes called "quiet conversions" or "automatic conversions", these implicit conversions occur under four circumstances:

1.) Assignment Conversions:

In assignment statements, the value on the right side of the assignment is converted to the data type on the left side

2.) Integral Widening or Integral Promotion Conversions:

A "char" or "short int" appearing in expressions are converted to an "int" type. "Unsigned chars" and "unsigned shorts" are converted to an "int" type provided it can hold the size of their value, otherwise they are converted to an "unsigned int".

3.) Operator Conversions:

In an arithmetic expression, objects of the expression are converted to conform to the conversion rules of the operator.

4.) Function Argument Conversions:

Scalar arguments smaller than an "int" type are converted to an "int". "Float" type arguments are converted to a "double" type.

Note: "Prototyping", a new feature introduced in ANSI C, enables function allusions to include data type information of arguments, and will turn off automatic function argument conversions.

The following rules for implicit conversions occur after all integral widening, or integral promotion conversion has occurred.

- If a pair of operands contain a long double, the other value is converted to a long double.
- Else, if one of the operands is a double, the other is converted to a double.
- Else, if one of the operands is a float, the other is converted to a float.
- Else, if one of the operands is an unsigned long int, the other is converted to an unsigned long int.
- Else, if one of the operands is a long int, the other is converted to long int.
- Else, if one of the operands is an unsigned int, then the other is converted to an unsigned int.

Essentially, implicit conversions will traverse the hierarchy of C scalar data types in an upward direction, from smaller to larger byte allocation.

The Man Behind The "Rocket"

An Interview with Chris Burke

(Interview by Richard Albers our OS9 Underground roving reporter at the PNW Fest in Port Orchard Washington

Transcribed by Alan Sheltra)

Chis Burke showing off a prototype of the "Rocket" mounted in a CoCo case.

(Editors' Note: There were 2 people asking questions in this interview, Richard Albers (RA) and Allen Morgan (AM).)

RA I have a little bit of a cold and my throat is just about gone. So you ask the question and I'll just stand here ☺

CB The chip itself on the Rocket board is a 68306, it's a brand new chip. In fact this is an engineering sample that's on here, but they are supposed to be in production soon. This chip has a 68000, it's a straight 68000 on it, not a CPU 32. It has what is called a DUART on it which is the

supplied tools.
Hopefully, the OS-9 Users group will reach out to some of these people and put some of these people on the list. It'll be interesting to see what happens, but it's a tough, tough thing to crack. I think what we need is the cooperation of somebody like Microware and some of the major board manufacturers like Gespac.

RA One wonders if Microware know who is using OS-9, 'cause they license to others who resell to the industrial users.

BvP That's right and I don't know if Gespac, for example reports to Microware who they sell their products to, I don't know?

RA I would know of no reason why they would?

BvP Yup, even though when you buy a machine, you're supposed to get a card that's supposed to give you some support, so maybe? There's a lot of big companies using OS-9, the list is Fortune 500 type.

RA We have NASA on that list.

BvP Yeah, and there's a lot of other one's too.

RA Have you contacted them?

BvP Yeah, I have been contacted by them. I didn't make a sale because my product was too low in price.

RA

BvP If I'd know it was NASA, I would have added a zero or two to the price, when I was talking to them.
but that's one of the reasons we have a dual pricing strategy because, frankly, industrial user's are just not prepared to pay $50 for a piece of software.

RA $50 is not even enough to hit the petty cash fund.

BvP That's right, that's right and if it's only "$50" it can't be much good.

RA Especially when your custom software costs you $50,000. Somebody knew to add an extra zero.

BvP Well, custom software is expensive. When I look at the hours... I shouldn't look at the hours... that go into a program like Ved, it's unbelievable the number of man-hours that went into it. When you look at some of the programs that are being developed, they have teams of developers working on this stuff. The man-hours, it's incredible!

RA They can sell those for 4 or 5, $600.

BvP Yes!

RA I am glad, among other OS-9 users that you don't charge us by the hour.

BvP A Well, if I charged you by the hour I'd be in a different business.

(EOF)

Hierarchy of C Scalar Data Types

long double
|
double
|
float
|
unsigned long int
|
long int
|
unsigned
|
int

MIXING INTEGERS

The four sizes of integers, "char", "short", "int", and "long", may be mixed freely in expressions. The compiler converts "chars" and "shorts" to an "int" data type due to the integral widening or promotion rules, before evaluating the expression.

To convert an 16-bit "short" to a 32-bit "int" data type, all that is needed is to add two additional bytes of zeros.

short 5 = 00000000 00000101 = 2 bytes in length.

After conversion to a 32-bit "int" type:

int 5 = 00000000 0000000 00000000 00000101
= 4 bytes in length.

For negative values, the compiler must ensure the converted value is also negative, by filling the additional bytes with ones instead of zeros. This is known as "sign extension". Using two's complement notation, the internal representation of a 16-bit negative value must retain the sign bit.

short -5 = 11111111 11111011 = 2 bytes in length.

After conversion to a 32-bit "int" type:

int -5 = 11111111 11111111 11111111 11111011

When an implicit conversion "shortens" a value in assignment conversion, problems arise.

For example, assume the variable "a" is a "char" data type with one byte, or eight bits allocated. If we attempt to assign a value requiring two bytes, or sixteen bits of storage, the two upper bytes fail to be assigned. This will cause an "overflow" condition and the result will be erroneous.

After conversion to a shortened data type of "char", one byte in length, the compiler usually ignores the extra bits and the assignment of the right-most eight bits are made:

16-bit value:

882 = 00000011 01110010 = 2 bytes in length.

After conversion to an 8-bit "char" type:

882 = 01110010 = 1 byte in length. /* WRONG */

Thus, variable "a" is erroneously assigned the value of 114 rather than the value 882. This overflow condition applies to "shorts", "ints", and "long ints". It becomes important not to exceed the size limits in assignment expressions when traversing the hierarchy in a downward direction.

MIXING SIGNED/UNSIGNED TYPES

Simply stated, a "unsigned" value is a positive value and a "signed" value is negative. Signed and unsigned integer types are allocated the same amount of memory for storage. The difference between them is the way they are interpreted.

A "signed char" with the decimal value of "-22", assuming two's complement notation, has the 8-bit binary representation of:

8-bit signed char a = -22 = 11101010

An "unsigned char" with the same 8-bit bit pattern has the decimal value of 234.

8-bit unsigned char a = 234 = 11101010

Most pre-ANSI compilers convert "unsigned chars" and "unsigned shorts" to "unsigned ints", 16-bits in length, preserving the the unsigned quality:

```
16-bit signed char a   = -22 = 11111111 11101010
16-bit unsigned char a = 234 = 00000000 11101010
```

When "signed" and "unsigned" values are mixed in assignment expressions, the result is always "unsigned". This "sign preserving" feature may sometimes produce unusual results.

For example:

```
int b;
unsigned short a = 2;

b = a - 3;
```

This will evaluate to a very large number, rather than the signed value of "-1" you would expect.

The ANSI standard adopted a different conversion method known as "value-preserving" to avoid this problem. By promoting "unsigned chars" and "unsigned shorts" to the larger "int" data type, the sign extension of variable "a" is preserved and the result of the expression will be "-1". If the "int" data type is not larger, the value is promoted to an "unsigned int".

The difference between "sign preserving" and "value preserving" comes into play only when an "unsigned" data type is shorter than the an "int" data type. If both operands are "unsigned int", the result is unsigned.

We need to be careful when using "unsigned" values for program flow control. Using unsigned conversion rules, in following example, the expression "a - b" will never evaluate to less than zero.

```
unsigned a;
int b;

if( a - b < 0 )   /* This expression will never be true */
  delay();
```

The compiler should be able to diagnose this problem as a bug and issue a warning.

CASTS

It is also possible to "explicitly" convert a value to a different type. This called "casting". To "cast" an expression, enclose the target data type in parentheses directly before the expression.

For example:

```
int a, b = 5;

a = (float) b;
```

This converts the variable "b" to a "float" data type before assigning it to the variable "a". Since variable "a" is declared to be an "int" data type, it will be converted back to an "int" type BEFORE the assignment is made. The way to avoid this problem is to cast "a" as a "float" type. There would then be reason to retain the promotion of variable "b" to a "float" data type.

Consider the following:

```
int a = 5, b = 3;
float c;

c = (float) b / a;
```

This expression explicitly converts variable "b" to a "float" data type. Now the expression contains two floats, "b" and "c". The result is a "float" type assignment as variable "a" is automatically promoted to the data type of the variable on the left side of the assignment expression.

As a general rule, avoid mixing data types of different sizes. Assigning a floating point value to an integer variable will cause the fractional part to be discarded. This causes a loss of precision which could dramatically impact your program. Another serious scenario occurs when the floating point value cannot fit into an integer data type.

For example:

```
int a;

a = 999999999999.888888;
```

This will cause an overflow condition and possibly halt the program execution.

RA A megabyte of C is a lot of SRC!

BvP There's probably in the whole program, about 15 lines of assembly. Very little assembly, and that's just because it's very easier to write those things in assembly because of speed. It's very fast. I'm amazed by C, just how fast it is. VPrint is the other one I'm porting, but it's not as problematic 'cause there's already so many options in it.

RA It's an option interpreter.

BvP That's right. It's a programming language. A print formatting, programming language.

RA I've seen a similar item on UNIX.

BvP The amazing thing is, that Ved is so complicated, there are so many options, that I as the author, when ever I'm writing something, I have to refer to the manual.

RA ☺

BvP You know it's crazy. So what's nice about this is we have that MM/1 market and the Tomcat , not the Tomcat, the TC70 and some other industrial users. We hope that Chris has some real good success with the Rocket because those are lots of customers. But we a have a great untapped market in the industrial users. Now I have some industrial clients, but very few. The reason I don't have more is because I don't know who they are! They are very difficult to contact.

RA There's no Users Group for industrial users for OS9?

BvP No. Now I do have a European reseller and I'm working on, negotiating a contract with a Japanese reseller, and these are strictly industrial people.

RA Does it work as well in Japanese also?

BvP Well, that's one thing we're working on.

RA Or they'll have to write in English.

BvP Hopefully it will handle there extended character sets. We'll see. My Japanese is nonexistent.

RA VPrint is going to need some...

BvP Well, I don't think that is going to work at all. Don't they go up and down or left to right, or right to left? I don't know that much about it.

RA I don't know that much about it.

<Editor's note: Yes, Japanese is written right to left, up and down>

BvP But we are getting that industrial market in Europe through our resellers and that's kind of nice because we can sell the product for a bit more money because we sell an industrial license rather than a personal license.

RA Multi-User.

BvP That's right. We charge some more money, but of course they get some more benefits, they get to make copies and they get some additional support. There are a lot of people in this country, in America, a lot of industrial users and I don't know where they are? And I talk to other people who are also in this OS9 business and it's a real problem. How do you reach these people?

RA There's probably a similar problem for other operating systems too in the industrial environment.

BvP Oh, I'm sure there is.

RA There's no national registry operating systems.

BvP But this applies not only to my stuff, but it also applies to things like the EFFO public domain stuff and the other things that are on places like Compuserve. I had a developer I was talking to a while ago, not a developer, an industrial user, one of my customers. He found me on Compuserve but he was stumbling around Compuserve personally and stumbled on the OS-9 forum and he said there's all this neat stuff here and he downloaded it and took to work where he's using OS-9 and saved himself all kinds of money because all these little utilities and things, they were going to write themselves.

RA They were going to write themselves or have to have custom written which is terrible expensive.

BvP That's right, but they didn't know that all these products were available, a lot of it free and a lot of other things, like my editor for a very low price. So there's a real vacuum there on how to reach these people. Not only does it help developers like myself, but it also helps these people who are using our favorite operating system and not having all the tools, all they have are the Microware supplied tools, and as you know, there's a lot more to life than the Microware

One on One with Bob van der Poel

(at the Pacific Northwest Fest on June 26th, 1993)

Interview by Richard Albers

Bob van der poel speaks at the Pacific NorthWest Fest in Port Orchard, WA

RA We're talking with Bob van der Poel

BvP <Falsetto voice> Yeah, he talks with a high squeaky voice

RA Essentially I was going to do the same thing I did with Chris, is let you talk about what you're doing, what your software is doing.

BvP Well as you know, I'm spending most of my time on OSK stuff right now and basically we got 2 major products we're supporting, one is Ved text editor, the other is our VPrint formatter and that's taking up an awful lot of time.

RA A lot of time re-writing on that?

BvP Well, we're spending time doing 2 things, one is fixing up bugs that keep occurring and we're running the software now on a lot of different platforms. We have European sales...

RA And you have a different version for each one?

BvP No, that's the beauty of OS-9, you run the same version, but different platforms exercise different parts of the same program. You discover different bugs.

RA Ah, a bug doesn't appear until it's actually in use.

BvP Well, for example on the more sophisticated platforms, they have memory management which tracks all your pointers and if you have a pointer, pointing to something that doesn't belong to you, it objects strenuously, so...

RA It's good that it objects.

BvP Well it certainly is, but if you're running on something like an MM/1, it doesn't care about those things so that's the other thing I find with these other processors, you have a program that runs fine on your MM/1, they don't run on these other machines because of the problems with the stray pointers. Fortunately, you become a better programmer. And the amazing thing is that a program like that has so many options, so many innards, that I found bugs 2 weeks ago that had been in Ved for a year and a half that had never been noticed.

RA I've never seen Ved for the 68K.

BvP Well, it was running over on the Kix, you should have come over and seen it.

RA I saw them trying to get it started..

BvP Oh they'll be running it later. It was just a matter of trying to get the right files on the disk. Oh and plus we have customers, I love them... and I also hate them because they keep saying "well, could you add this...", and I usually end up doing it!

RA Too good of an idea to ignore.

BvP Yes, and that terrible number of options to program, since the initial port of V1.0 it has probably doubled in size.

RA How big is it now?

BvP Well, there's almost a million bytes of SRC.

RA And this is written in C?

BvP It's all C.

ENUMERATED DATA TYPE

Enumerated data types allow you to store a unique set of values to a variable by using the "enum" keyword. They are particularly useful as each "enum" constant name declaration receive a default integer value based on their position in the enumeration list. The compiler need only allocate as much memory as is necessary for the enumerated value.

The default values start at zero and are incremented by one with each new constant name.

For example:

```
enum boolean { NO, YES };
```

The first constant name, "NO", has the value of zero, the next, "YES", a value of one. We can override the default values by specifying other values. In this case, all subsequent default values begin at one more than the last defined value.

For example:

```
enum fruit { APPLES, ORANGES = 10, LEMONS,
     GRAPES = -5, MELONS };
```

Is the same as:

```
enum fruit { APPLES = 0, ORANGES = 10,
     LEMONS = 11, GRAPES = -5, MELONS = -4 };
```

For readability, the assigned values should be written in ascending order, although it is not necessary to write them that way.

For those compilers that do not support an enumerated data type, it is possible to create a "pseudo" enumerated data type using char arrays. While the subsequent values are not automatically incremented, it is possible to assign values to this "pseudo" data type, the default values are assigned by the position in the array list.

For example:

```
#define APPLES 0
#define ORANGES 1
#define LEMONS 2
#define GRAPES 3
#define MELONS 4

char fruit[] = { 0, 10, 11, -5, -4 };

int qty;

qty = fruit[LEMONS];
```

In the previous example, the variable, "qty", will be assigned the value of eleven. Since the macro, "LEMONS", has a definition of two, "fruit[LEMONS]" evaluates as "fruit[2]" and is assigned the value stored at that array position.

VOID DATA TYPE

While not originally part of the K & R standard, the void data type has become accepted as part of the C Language. The void data type has two important features. To indicate that a function does not return a value, or to declare a generic pointer.

A void data type, when used as a part of the function declaration, informs the compiler that the function does not return any useful value and any attempt to use the returned value should be flagged as an error.

For example:

```
extern void delay();

main()
{
  int x, y;

  x = 0, y = 1000;

  delay( x, y);

}

  void delay( a, b)
   int a, b;
   {
      int i;

      for( i = a; i < b; i++ )
        ;

   }
```

However, you cannot any assign the returned value to a variable. The following assignment should produce an error:

```
elapsed = delay( x, y );
```

A void data type used as a pointer generally applies to memory allocation. Prior to the ANSI standard, it was the programmer's responsibility to cast a returned memory allocation pointer to a pointer to the correct data type. Since memory functions are implemented differently in ANSI and the K & R standards, the ANSI standard can make use of the void data type by returning a pointer to a void. The "void pointer" is automatically cast to the correct data type when it is assigned a pointer value. However, with ANSI C, you must include the "stdlib.h" file as it contains the function prototypes.

K & R standard call to malloc():

```
int nmbr, *pt;

pt = (int *) malloc(sizeof(nmbr));
```

In the previous example it is necessary to cast the pointer, "pt", explicitly. With the ANSI C version, we do not need to cast the function result.

ANSI standard call to malloc:

```
#include <stdlib.h>

int nmbr, *pt;

pt = malloc(sizeof(nmbr));
```

Since this new syntax will not work on older compilers, it is best to use the K & R standard call to malloc for portability to non-ANSI platforms, or at least be aware of the difference when porting code to an ANSI compiler.

-Leonard Cassady

Offset	Name	Maintained By	Description
$00	PD_PD	Kernel	Path number
$02	PD_MOD	Kernel	Access mode (R W E S D)
$03	PD_CNT	Kernel	Number of paths using this PD (obsolete)
$04	PD_DEV	Kernel	Address of related device table entry
$08	PD_CPR	Kernel	Requester's process ID
$0A	PD_RGS	Kernel	Address of caller's MPU register stack
$0E	PD_BUF	File Manager	Address of data buffer
$12	PD_USER	Kernel	Group/User ID of original path owner
$16	PD_PATHS	Kernel	List of open paths on device
$1A	PD_COUNT	Kernel	Number of paths using this PD
$1C	PD_LProc	Kernel	Last active process ID
$20	PD_ErrNo	File Manager	Global "errno" for C language file managers
$24	PD_SysGlob	File Manager	System global pointer for C language file managers
$2A	PD_FST	File Manager	File manager working storage
$80	PD_OPT	Driver/File Man.	Option table

Figure 2 - Fields in an OS-9 path descriptor

Once copied into in the path descriptor, these options can be changed using the tmode utility. Likewise, using xmode will change the values in the device descriptor itself, causing subsequent paths to inherit the modified options.

Thus far, we've examined the OS-9 I/O structure and the modules and data structures that comprise it. Information in this article is prerequisite to writing a driver, so learn it.

I'll leave you to ponder the information presented thus far. Next month we'll delve deeper into the bowels of OS-9 I/O and study the hardware which will be the target of our driver.

-Boisy Pitre

Note...

① In OS-9 V2.4, IOMan is actually part of the kernel module whereas in V3.0 it is a stand-alone module. When referenced in the context of these articles, we assume it to be part of the kernel.

Offset	Name	Maintained By	Description
$00	V_PORT	Kernel	Device base address
$04	V_LPRC	File Manager	Last active process ID
$06	V_BUSY	File Manager	Active process ID
$08	V_WAKE	Driver	Process ID to awaken
$0A	V_Paths	Kernel	Linked list of open paths
$0E	Reserved		
$2E	V_DEV2	Kernel	Addr. of attached device storage
$32	V_TYPE	File Manager	Device type or parity
$33	V_LINE	File Manager	Lines left until end of page
$34	V_PAUS	Driver/File Man.	Pause request
$35	V_INTR	File Manager	Keyboard interrupt character
$36	V_QUIT	File Manager	Keyboard abort character
$37	V_PCHR	File Manager	Pause character
$38	V_ERR	Driver	Error accumulator
$39	V_XON	File Manager	X-ON character
$3A	V_XOFF	File Manager	X-OFF character
$3B	Reserved		
$3C	V_Presvd	Reserved	
$46	V_Hangup	Driver/File Man.	Path lost flag
$54	Driver Variables begin here		

Figure 1 - Fields in an SCF device static storage area

The Device Driver

The OS-9 device driver is made up of eight entry points which handle various aspects of the device. Each routines (with the exception of the ISR) are referenced by a jump table at the entry point of the driver. The eight entry points are:

1. INIT (called by IOMan)
This routine initializes the hardware and any driver variables. Interrupts are also enabled and the device is added to the IRQ polling table if necessary.

2. READ (called by the file manager)
This routine contains code to read data from the device.

3. WRITE (called by the file manager)
Writing data to the device is done here.

4. GETSTAT (called by the file manager)

5. PUTSTAT
These routines retrieve and send a specific status regarding the device.

6. TERM (called by IOMan)
The device is deinitialized, interrupts are turned off, and the IRQ polling table entry is cleared if necessary.

7. TRAP
This routine is not currently implemented.

8. ISR (called by the kernel's IRQ Polling Routine)
This routine is the Interrupt Service Routine, and is called when an interrupt is received from the device. If a device is not interrupt driven, this routine may be excluded from the driver.

The Device Descriptor

As noted above, device descriptors are special tables that contain no executable code. Parameters such as end-of-line character, end-of-file character, echo toggle and related options are stored here.

When a new path to a device is opened, OS-9 copies the options from the device descriptor into a newly created path descriptor.

BASIC TRAINING

BY JIM VESTAL

The End or the Beginning?

This installment of the Basic Training series will end the discourse on structured programming in Basic09 that was started in the first installment of Basic Training last year. Following this installment this column will focus on all aspects of OS-9, system commands and utilities, user interfaces, and more on programming, both in Basic09 and C.

On Error is still a *GOTO*:

Basic09 has many features that make it a very good structured language, but one feature that can cause problems is the ON ERROR GOTO statement. I wish Microware would have had a generic ON ERROR THEN or at least an ON ERROR RUN statement in place of the OR ERROR GOTO statement. My advise to Basic09 programmers would be to avoid ALL On Error usage, but in some cases errors do have to be trapped. In such cases use only one ON ERROR GOTO per procedure if possible. And if possible do not use other GOTO statements within the error trap. I have debugged other people's buggy code and in several cases have traced the problem back to an improperly trapped error. After removing the Error trapping it was simple to find out the cause of the problem and fix it. The main problem that programmers have is that they TRAP not just the expected error but ALL errors which itself causes many problems. Please make use of the ERR variable when you have to trap errors in your program do not assume (ass-u-me) that the error trapped is the one that is expected.

Overview of structured programming discourse

Here's a summary the main points that have been covered so far in this series.

Top-Down programming and the use of the REM statements — Learn how to write your code using a Top-Down structured design and make liberal use of REMark statements in your programs.

Goto statements, line numbers and loop structures in Basic09 — Plan ahead and sketch an outline of what your program is to accomplish BEFORE you code program Avoid line numbers if possible. Use separate procedures in your program as opposed to number subroutines. Take advantage of the looping structures built-in to Basic09 (Repeat, While, For and Loop). Avoid ALL use of GOTO statements.

ON ERROR GOTO statements — If error trapping is needed in your program, use only one error trap per procedure and avoid GOTO statements within the error trap.

Coming in future installments of Basic Training:

In the next few months I will be presenting a tutorial on the commands and utilities that came with OS-9. I will also talk about how to make Basic09 code portable to other systems.

Please contact me via email or the postal service. My email addresses follow this article. Till next month I'll leave you with this:

Murphy's Law No. 75 STEINBACH'S GUIDELINE FOR SYSTEMS PROGRAMMING
Never test for an error condition you don't know how to handle.

Email: ——Delphi: 09Online
Internet: sysop@narnia.citrus.sac.ca.us or 09Online@Delphi.com
StG: Sysop@Narnia
UUCP:
uunet!csusac.ecs.csus.edu!citrus!narnia!sysop
Postal mail:
Jim Vestal
221 E. 17th #31 Maryrsville, CA 95901

-Jim Vestal

Writing a Device Driver

Boisy G. Pitre

When taken to task, many OS-9 programmers can explain the function of a driver, but have never written one (or more to the point, don't really understand all there is to know about writing one).

We will discover the secret art of writing a device driver by taking a step-by-step approach into the process. At the end of this series you should have a basic understanding of the inner workings of a device driver as well as a stronger awareness of their role in

OS-9's structure. Unless otherwise noted, version 2.4 of OS-9/68K will be assumed.

To begin, let's familiarize ourselves with concepts relating to driver writing.

OS-9 I/O Concepts

OS-9 observes each device as a file. To maintain this high level of abstraction on the user level, a hierarchal system of modules have been defined. The I/O paradigm consists of:

IOMan ① *(See Note at end of article)*

IOMan (the I/O Manager) is responsible for all I/O system calls. The kernel calls IOMan. It in turn looks at the device descriptor of the device (by referencing the passed path number) and passes the I/O request to the appropriate manager.

Data returned from the file manager is passed to the kernel and finally to the user program.

File Manager

The file manager is tailored for a specific class of devices. It narrows the characteristics of the device down to a certain set of criteria (sequential vs. random access, block access vs. byte access, and so on). The file manager calls the specified device driver to do the actual data read or write and then does the logical processing. Processed data is then passed back up to IOMan.

Device Driver

Physical interaction with the device is ultimately the job of the device driver. Once the data requirements have been fulfilled, it returns the raw data to the file manager.

Device Descriptor

This is a non-executable module which holds various parameters related to the device (see Figure 1). It is normally referenced by the file manager. In addition to these modules, several system data structures play a part in completing a device I/O request.

Device Static Storage *(Figure 1)*

Upon a device's initial use, the kernel allocates an area of memory for the device's static storage. This area is used to store "static" variables that are needed for the duration of the driver's life. The exact fields in a device's static storage is dependent upon the file manager.

Device Table

The kernel maintains a list of initialized devices in a structure called the device table. In addition to other information, the device table includes a pointers to the device's static storage and the device descriptor.

Path Descriptor *(Figure 2)*

Each open path has a corresponding path descriptor. While the first 128 bytes of the path descriptor are used by the kernel and file manager, last 128 bytes make up the option area used for the path's operating parameters.

With the above in mind, let us look closer at the two key modules we will need to write ourselves.

by the super-user and processes in system-state.

There are many more cases where memory accesses can cause problems. If you use a language like Basic or Pascal you're probably safe; with assembler or C you have to be careful. Following careful programming guidelines will ensure that your programs run on your current system as well as other systems:

1. Make sure that you initialize pointers before using them,
2. Avoid passing NULL pointers to library routines,
3. Make sure that the system has given permission to access any memory.

The above is a very brief overview of memory protection. Having it can, at times, be a real bother. But, with multi-user, multi-tasking computers it is a must—without it you can damage other processes data and cause very hard-to-debug errors. Writing all your programs with memory protection in mind will create better, bug-free programs.

It has been very nice to hear from so many readers; keep those cards and letters coming. I can be reached at CIS 76510,2203;

PO Box 355, Porthill,
ID, USA 83853;
- or -
PO Box 57,
Wynndel, BC, V0B 2N0 Canada.

-Bob van der Poel

(EOF)

Dealing With Memory Protection

by Bob van der Poel

A number of computers available today support memory protection schemes though software and hardware. I'm am not going to delve into the details of how this is done...there are a number of different methods. Instead, I'm going to relate a few of the problems I've encountered in porting my programs to hardware of this type.

When you write in a language like C, you have the power to access just about any memory location in the computer—whether you should or not. For example, assume that "p" is a pointer to char:

```
p=anyvalue;

*p=44;
```

and you run the program on your Coco or MM/1 or even on a MSDOS machine the value 44 will be stored at memory location "anyvalue". If "anyvalue" has been allocated by the operating system to your program this is probably fine. But what if "anyvalue" points to another user's program, or points to non-existent memory? You either get a program crash (maybe now, maybe later), or you might be lucky and have nothing at all happen.

Running the same program on a platform with memory protection hardware will cause an error. For example, on a MVME system running OS9/68000 with SSM (system security module) an error 102 or 103 will be reported. SSM monitors all memory accesses and only permits those to memory which has been allocated to your program.

So far this seems to be just ideal. It is, until you try to run programs which work fine on your old machine.

One of the most common problems I have found is in doing comparisons on NULL pointers. For example, if we wish to compare two strings it is easy to use the strcmp() function:

```
char *str1=NULL;
char *str2="foo";
int t;

t=strcmp(str1,str2);
```

On systems without memory protection this will work. However, in a protected system it will fail. The reason is simple: strcmp() does not test to see if the pointers passed to it are NULL; it blindly checks to see if the character at str1 matches that at str2. Without memory protection, this test will fail and a "non-equal" will be returned. However, with memory protection the first test will cause a system error—your program does not have permission to read or write to the memory at address 0. You have to write your code to take this into account:

```
if(str1 && str2) t=strcmp(str1,str2);
```

will work (however, you'll have to set the value of "t" another way).

Memory protection knows about read and write protection—some memory you can only read, only write, or both. In one program I wrote I used a data module to share information between different processes. This data module contained constant data, plus shareable variables. Each of the different program modules contained code which first attempted to do a modlink() to the data module, and if that failed, it attempted to do a modload(). Sometimes it worked....sometimes it didn't. Why? Well, modload() let's you specify the access permission (and I'd used READ); modlink() gives read and write permission. So, if the data module was in memory and the modlink() worked all was fine; if the module had to be loaded I got an error (but running the program a second time was successful).

Under OS9/68000 you run into a neat situation if you try to access an I/O port. Again, the system treats I/O ports like memory...and your program doesn't own it. So, special system calls exist which will grant and deny permission (F$Permit and F$Protect) to areas of memory. These calls are only available to processes owned

Continued Page 22