SUPERPET GAZETTE

The new 68000 machines are out--Atari's ST and the Amiga from Commodore. Three ISPUGgers have bought ST's with color monitors for $995; all report they are unable to penetrate below the Mac-like GEM interface, access the 68000, or use the underlying operating system. As with the C64, the computer is in the keyboard enclosure with 512K bytes of RAM. A separate power supply is required for the computer and for each disk drive. With the two-button mouse and two drives, you must cable together eight separate units. Have a large desk and a tolerance for a warren of cables. Two users report they don't like the mouse; at least you have an option with the ST; unlike Macintosh, it provides cursor controls on the keyboard. Atari, short of money, is trickling the machines out to dealers and supporting it with what little advertising it can afford.

The single cutesy wee 3.5-inch disk holds 360K bytes, which in the day of 200K byte programs and megabytes of data is about as useful as a glass of spit at a four-alarm fire. Atari has repeated Apple's bitter mistake with the Mac, but will, we hear, offer a double-sided 720K byte wee drive for more money.

Apple, of course, is upstaged. Its main act, the 512K byte Fat Mac, has no color and costs twice as much; its aging ballerina, the Apple II, can compete neither on performance nor price. Apple has a year, perhaps two, to arrange a new act and to reduce its always-high prices, until the ST attracts enough support to compete against Apple in software. As if that weren't enough, the new Amiga from Commodore is far superior to the ST and no more expensive than a Mac.

BYTE magazine has long disdained Commodore products, noting them (if at all) with quiet condescension. The August issue features the Amiga, a rave product review, and an editorial praising it as a machine to "rekindle the enthusiasm which drives personal computing." The September issue of COMPUTE! had an orgasm.

Why the enthusiasm? It's a home computer, a graphics computer, a music computer, a sound computer, a business computer, a game computer, a hacker's computer--all rolled into one. One reviewer said it's a general purpose machine which excels at special applications. It may cool you off to know that the operating system/library occupies 192K bytes (compared to 24K for SuperPET), and that the technical data to explain it is in like proportion.

The numbers following do not tell the story; read the reviews if you can locate them. Amiga, unlike Macintosh, has freed the 68000 from handling either graphics or sound by employing custom chips; the graphics are reported dazzling. At $1295 the basic machine has 256K bytes of RAM and one cutesy wee drive (880K bytes, double-sided), plus some bundled software on which the reviews disagree. You'll have to shell out about $2300 total for two drives, add-on RAM (512K bytes total in the system enclosure) and a color monitor. The architecture is open, the bus being extendible to a forthcoming expansion box which can hold 6 (maybe more) megabytes of RAM and some custom boards. Monitor, keyboard, and system enclosure are separate, in the current fashion. A 20-megabyte hard disk is promised from Tecmar. We're happy to see that the RETURN key is huge, Selectric-style, and that Commodore had the wit to put the four cursor keys tight against the main keyboard, where a touch-typist can use them without losing rhythm. (Ever had tennis elbow? Weep for cursor wrist.)

The problem for Commodore as for Atari is software, and in the U.S. at least, lack of a dealer network to sell and support Amiga--Kindly Uncle Jack Tramiel

having thrown his dealers to the wolves before he left Commodore. Last we heard, Commodore executives were about to safari into the commercial wilds, hunting dealers to stock the Amiga. Computerland and some other chains have refused to order it as a standard chain item, though their individual stores are free to stock it. It'll be months before we see the machine on dealer shelves; maybe more, for we don't know if the Amiga has passed the FCC tests for radio frequency interference (at last report, the the C-128 was hung up there).

As does the ST, the Amiga offers a mouse and a Mac-like user interface, named "Intuition." Commodore, unlike Atari and Apple, does not presume that mommy knows best; users are allowed to bypass the icon jazz and access the operating system (Amiga DOS) directly. The DOS is reported to have many of the features of Unix and MS DOS (it damn well better allow more than 8-character filenames!); it includes batch files. Finally we see a machine with a simple, easy-to-learn user interface for novices and direct access to DOS for experts--at which we utter three great cheers and cry, "About time!"

While we suspect our long search for a sensible successor to SuperPET may be over, we aren't about to divorce the old dear. She possesses too much useful software; we're going to keep her until the insulation rots off.

Caution before purchase is well advised on both the ST and the Amiga. New computers are never quite as good as the first, enthusiastic reviews. They always have bugs; time alone reveals their genetic mistakes. In addition, we hear that the operating system and the languages for both are being or have been written in 'C', not in assembly. Having been penalized by not-so-optimum compiled code generated by a C-like language (WSL) for the last few years, we will not walk innocent and unaware into a thicket of long, slow code which the innate speed of the 68000 is supposed to overcome. Remember that poor Lisa died from an overdose of an OS in Pascal despite her microprocessor; that the Mac's 68000 has never overcome the glacial slowness of a Basic written in C, and that the chief reason for a new computer is higher performance. We'll have to see a few benchmarks before we draw our checkbook from its scabbard.

We didn't mention Amiga's ability to perform several functions at the same time (concurrency and multi-tasking are the buzzwords). The machine is able to backup disks whilst processing words whilst printing hard copy whilst running a program or so in language. We can appreciate being able to compute while the printer churns out hard copy and the DOS chuckles away at a copy job, but are otherwise very skeptical of the virtues of schizophrenia. Inevitably the microprocessor must slow down as it rotates between tasks; the user, hopping from job to job, inevitably will become confused and make mistakes. In time, we suspect users will paint concurrency yellow for "danger, use with caution."

Having briefly looked at the option of a new computer, we present this issue some other options--a new, inexpensive megabyte drive; some add-on hardware and software to give SuperPET a 68000 microprocessor and up to two megabytes of RAM; options to use SuperPET's parallel port, a clear explanation of how to use the ACIA for telecommunication, and (if it arrives in time) a page on OS9.

REMARKS on REDMARKS If your address label is redmarked, your membership in ISPUG has expired, poor thing. Clip the application on the last page, check the RENEW block and favor us with a check made to ISPUG. Leave the address label on the other side so we know which J. Q. Schmidt sent it.

DRIVE FOLLY No reader of the Gazette can doubt that we dislike the wee 3.5-inch drives which the computer industry is trying to cram down our throats. If you question our logic, consider that Commodore recently demonstrated the Amiga in IBM PC compatible mode--running Lotus 1-2-3, but with add-on 5-inch drives! Splendid! Want PC compatibility? Buy an extra set of drives! The decision to use wee drives on a computer designed to sell in the business market, where the IBM PC is standard with 5-inch drives, strikes us as luminescent folly--especiially when IBM provides 1.2 megabyte 5-inchers on the AT.

As if that were not enough, ask how you will back up any hard disk. If you say by tape, we can only quote Bill Gates of Microsoft: "The makers of tape backup systems will burn in hell." If you say by disk, we hope you're happy while you back up 20 megs on 23 or more wee disks. The job is bad enough on 1.2 megabyte 5-inchers, of which you'll need 17. Only the makers of the expensive Bernoulli box have come up with a simple and reliable backup for hard disks. Until we see an inexpensive solution, we want no wee disks on any computer we own.

ONCE OVER LIGHTLY
Miscellany Last issue, we showed how a little structure, substituted for GOTO, could halve the time in which the sieve of Eratosthenes would execute. Later, guru Terry Peterson sent a HALGOL timing of the structured version--which executes in 1.79+ seconds (down from 2.18+ seconds in the GOTO version in HALGOL), an 18% increase in speed. It seems that good structure speeds up execution because it represents a minimal path through code--in any language, we suspect.

Whilst trying to reduce the run time in the Sieve, we inadvertently discovered that you may "next i%" as often as you wish in a FOR...NEXT loop in microBASIC without a syntax error. We show such a loop below to illustrate. At each NEXT i% execution jumps back to the FOR..NEXT line.

```
for i%=zero% to size%
  if flags%(i%) then NEXT i%                                    ! Why not?
  prime% = i%+i%+three% : k%=prime%+i% : count%=count%+one%
  if k% > size% then NEXT i%                                    ! Why not?
  for j% = k% to size% step prime%
    flags%(j%) = one%
  next j%
NEXT i%
```

SCHEMATICS AVAILABLE Thanks to the labors of a couple of ISPUGgers, we have in hand a 55-page set of schematics which they say define the circuits and chips for both 2 and 3-board SuperPETs; they also define sources and types for chips, including EPROMs to replace any bad ROMS. If you want a set, ORDER NOW! We'll hire somebody to slave over that hot Xerox machine and make enough copies to fill the orders. We will not stock this item. Order from ISPUG, PO Box 411, Hatteras, N.C. 27943. Price is reproduction cost plus postage: $11 U.S. We'll have to delay until all orders are in before we can print and ship.

MAN VERSUS PERSON We aren't unsympathetic to the feminists, for we do think that women have had a raw deal for a couple of millennia. On the other hand, the effort to change English so it isn't sexist may have taken the wrong path. Nowadays, anyone who releases news to the press is called a "spokesperson", even though she may be a spokeswoman, or he a spokesman. Shall we carry "person" to

its logical penultimates: horseperson, garbageperson, milkperson, bogeyperson? Gee, what do we do with "oneupsmanship?" Oneupspersonship?

Try personkind, personikin, personslaughter and personhole, or consider an attack by a person-eating shark. If you are still with us, ask if the Russians would be unpersoned by a threat to settle a dispute with a couple divisions of of infantrypersons, supported by some determined artillerypersons and a task force of tankpersons. Would they back down or would they personhandle us with their own armed personpower? Would both sides dispatch a fleet of well-personed persons-o'-war into the Persian Gulf? Does this note constitute unpersonly fun and gamespersonship? No, we draw your attention to what can happen when you begin to personipulate the language on the behalf of wopersons with the wrong word. "Person" is patent foolishness; it ain't gonna fly.

COPYING REL FILES ON MICROPOLIS DRIVES Those of you who own Micropolis-made 8050 drives may think you've crashed when you try to copy a long RELative file; the light on the copy-from drive comes on; the light on the copy-to drive often fails to light up for five minutes or so, and even then may only flicker a bit now and then. Fear not; be very patient; the file is being copied. We are continually surprised by the difference in performance between Tandon-made and Micropolis drives; the Tandons are noisy as a boiler factory but very fast; the Micropolis drives quiet but slow. Given a choice, we'll take the Tandons.

ANOTHER OBSCENITY Anent our complaint about folks hiding on/off switches and such in 3, II, Brad Bjorndahl writes that his favorite obscenity on SPET is the brightness control knob, which you 1) can't find, or 2) get hold of. Well, swipe a short piece of surgical tubing from your doctor, or a snippet of soft vacuum tubing from your service station; slip it over the knob, and you can find and fondle the hidden, obscene, and unmentionable device as often as you please.

HIGH RES GRAPHICS Brad also writes that a high resolution graphics board for SuperPET and 8032 and 4032 PETs was demonstrated to TPUG recently, having resolution of 640 x 250, two display pages which can be scrolled, 32K of RAM which may be used as a RAM disk with OS9, mixed graphics and text; price is under $300 Canadian from High Res Technologies, 16 English Ivyway, Toronto, Ontario, Canada M2H 3M4. Brad says users will have to write their own application software and that the board may be used in 6809 mode. If you are interested, get in touch with the maker.

CLEANING THAT COMPUTER In the June, '85 BYTE, Jerry Pournelle writes about dirt in computers; seems he'd never cleaned an old computer, moved it, and then suddenly found it wouldn't work. He called in an expert, who said the move had shifted the dirt and shorted out the board. The expert removed all boards and cleaned with compressed air and some commercial solvents [TV tuner cleaner, some Instant FD Zero Residue Cleaner (mostly trichlorotrifluoroethane), and a can of DE-OX-IDE]. All cables and sockets were cleansed with the cleaners above and then treated with DE-OX-IDE, which removes (what else?) oxide corrosion. Connectors and cables were then re-installed, all chips pressed firmly into sockets, and the computer ran. Shortly after we saw the article, Marlene Pratto wrote us that when her 3-board SPET turned flakey, her husband took it apart and cleaned it with Blue Shower from a local electronics store. When put back together, SPET worked perfectly. We've never done this, but pass the word.

Marlene Pratto also loves MacInker, a gadget which re-inks cloth ribbons without mess (see ads in COMPUTE! and BYTE); she calls it a "terrific machine". From the

dim print on the letters we get, about 80% of ISPUG members should buy one!

NEW SERVICE NUMBER AT COMMODORE In II, 3, Tony Klinkert gave a phone number to be used for repair service at Commodore's main plant in West Chester PA. Lee Seymour writes that it's wrong. There's a toll-free service number: (800) 247-9000, manned from 9 a.m. to 12 midnight E.S.T, Monday through Friday. We called, and the people on duty are courteous and knowledgeable. For repair, you must call (215) 431 9235 for a Repair Authorization (RA) number. Depending on where you live, you will be told where to ship your equipment (there are several service centers). The rumor that SuperPET is no longer being repaired at the main plant is not true; Commodore is directing the work to several sites. We strongly advise you not to write Commodore (you won't get an answer); call!

OOPS, AHEM, and WHOA DEPT. The EXEC file SCRCOPY:EXE which is printed in the tutorial on batch files on ISPUG Utility disk II has a minor error in it (all errors we make are minor; after all, we edit this rag). We ran and tested the file with an earlier BEDIT; the final version of BEDIT encloses all NOT searches and deletes in two backslashes (see above). The *\"/d on disk is wrong, and fails to delete all lines which do NOT have a quotation mark in them. Please change the file.

```
*\"\d
```

INEXPENSIVE 6502 ASSEMBLER We reviewed the excellent Waterloo 6502 Development system in II, 4 (page 104), which runs in 6809 mode and lets you write code in the microEditor. It costs $250. Reginald Wood of Hawaii writes that he uses the Label Assembler Development System (LADS) from COMPUTE! books [get "The Second Book of Machine Language, by Mansfield ($14.95) and the accompanying disk ($12.95)]. You write source code in the BASIC editor (you need the POWER chip or something similar) and assemble it with a SYS call. The disk holds versions for the PET/CBM, C64, and VIC 20; the book explains the assembler and holds the source code plus many examples. Reg says the assembler is pretty fast.

THE BOTTOMLESS PIT OF DICKY DUMBJOHN All our disk drives went bad on the very same day. We assembled and linked a program named "test" in Development, and tried to load it in the monitor with: >l test.mod, only to have the disk drive report "no such file". At left is what we saw on directory. "Bad disk," we sighed, and tried again. Same problem. "Bad drive," said we, and tried another drive. Oh, God. Same problem. We shut down and cold started. Same problem. We tried John Toebes' new linker. Same problem. For four hours we tried everything our mind could conceive. "Major bug," said we.

```
1     "
.mod"     PRG
```

Well, the major bug resided between our ears. As even Bodsworth knows, the first line in a command file must contain the name to be given to a .mod file; the name must be in quotation marks. We should have entered "test" on that first line; instead, it read: ;test.cmd. The poor linker, dumb beast, finding no title for the .mod file, obediently located a carriage return there and so named the file: CR.mod, which, in calmer retrospect, is splendidly logical. When we told John Toebes what had happened, he had hysterics, but finally did promise to fix his new linker to report and reject such follies. John asked if the error message could be an audible giggle.

```
"test"
  vs
;test.cmd
```

MAD ABOUT PASCAL Our APL Associate Editor, Reg Beck, is mad about Pascal; B.C. school authorities now demand that he teach Pascal; his summer was devoted

to learning the language. Reg doesn't like it, and comments "Never use 20 words when 200 will do. Worship the Great God Type! Never define a primitive function 'power', for what type would it be?" Somewhat bitterly, Reg tells us M. Montalbano, in "A Personal History of APL," reports that he gave a seminar a few years ago on programming languages; one of the people attending was Niklaus Wirth, the developer of Pascal. Montalbano says, "Unfortunately, Klaus didn't get the proper message from my talk. He went his own way and developed Pascal."

Reg notes that while grade 12 students who will be going on to college might as well learn Pascal before the college forces them to, most of his grade 11 students won't go to college, but rather are interested in learning BASIC, word-processing, and graphics for their home computers. Reg says that Pascal, in this light, is about as applicable as PL/1. We agree with him in spades, worry about the high school drop-out rate (read that both ways), remember the classmates who fled similar pedantic follies, and wonder if curriculum directors are promoted to that post only if they have credits for four semester hours in stupidity.

ON HALGOL, THE GRANDE, AND MEGABYTES for SPET

In Issue 1, Vol. II (Oct/Nov 1984) we directed your attention to HALGOL, a swift new language offered by Hal Hardenbergh, the president of Digital Acoustics, (DA) at 1415 E. McFadden, Suite F, Santa Ana CA 92705. At the time of that report, Hal had just bought a SuperPET (converted 8032), and was at work creating the hardware to attach a 68000 board (a 12.5 MHz 68000, please note) and the necessary auxiliary gear to SuperPET. Terry Peterson and Nick Solimene of ISPUG are equipped with the hardware and with the latest release of HALGOL (while the language is not yet finished, DA provides versions which run).

Speed of execution is primary in the design of the language and its hardware. Each line of the language is compiled as entered (no wait for compilation); it is as yet without an immediate mode but is interactive, as BASIC is. Hal has claimed for several months that HALGOL is the fastest Basic-like interactive language in the world; to date, nobody has disputed that claim. [Hal insisted that we insert the phrase "Basic-like" because somebody claimed FORTH was a language (we consider it a hallucinatory drug) and somebody else said 'C' was a language (from study, we conclude it is a form of shorthand used by gurus who can't type). Of course, we are kidding--a little.] The accuracy and the speed of HALGOL emerge from hand-written assembly language, including a fast set of floating-point routines, plus the 12.5 MHz microprocessor itself.

The equipment and software we describe below is available for SuperPET, PET, the C64 and for the Apple II. The SuperPET gear runs from the 8032 side of the machine, and works with 4040 or 8x50 drives. Terry Peterson uses a $50 kit from Skyles Electric Works (The 1541 Flash) to get the 1541 disk drive to load as quickly as a 4040 on the C64 whilst using HALGOL. Terry Peterson did the I/O routines and other mods needed for the Commodore versions of HALGOL, at no small expenditure of time and skill.

To give you a flavor of the language and some feel for its performance, we'll use the Sieve of Eratosthenes. Terry Peterson converted the best-performing version printed last issue to HALGOL, and comments that "What ho! Structure is advantageous even where it's hard to see! (at least, sometimes)." The previous best time on HALGOL had been 2.1 seconds; the version at left runs in 1.79 sec. Compare

```
5 REM Structured Sieve Benchmark
10 START TIMER : LET size%=7000
20 DIM flags%(7001)
```

```
30 SELECT PRINT 1 :
   PRINT "Start one iteration"
40 LET count%=0 : LET one%=1
50 FOR i%=0 TO size% STEP 1
60 MVA one% TO flags%(i%)
70 NEXT i%
80 FOR i%=0 TO size% STEP 1
90 IF flags%(i%)=0 GOTO "skip"
100 LET prime%=i%+i%+3
110 LET k%=i%+prime%
120 LET count%=count%+1
125 IF k% > size% GOTO "skip"
130 FOR j%=k% TO size% STEP prime%
140 MVA zero% TO flags%(j%)
150 NEXT j%
160 "skip" : NEXT i%
170 READ TIMER TO t : LET t=t/500
180 PRINT "Done: ",count%," primes found in ",t," seconds."
190 SELECT PRINT 0 : END
```

that to the times below on the IBM PC:

True Basic	BetterBasic	Turbo Pascal	PC Basic
21.2	31.4	15.4	190.7

In mBASIC in Superpet: 168 seconds...

The "MVA" instruction at left is a quick, temporary way to implement array assignment and "moves" the variable specified to the stated array element (we said that HALGOL is not quite finished).

Named subroutines are called by a GOSUB; a GOTO goes to a name (See "skip, at left), not to line numbers, which are meaningless except to the line editor. The timer on the latest version of the boards runs in increments of 1/250th of a second, and may be STARTed and READ as shown. Although you see LET statements and they are mandatory, you needn't type them in. When you enter an assignment, such as: a=a+4, the compiler inserts a LET for you; similarly, it capitalizes all keywords, which you may enter in lower case.

The Sieve is not the best way to compare number-crunching capabilities; we list below the results on BYTE's calculation benchmark (with no math chip support; no Commodore version of HALGOL yet handles the 32081 math chip, though we anticipate it will be supported later). All times are in seconds; programs were run on the machines shown:

BYTE Calculation Benchmark in Seconds

On an IBM PC				On SuperPET	
True Basic	BetterBasic	Turbo Pascal	PC Basic	MicroBasic	HALGOL
19.7	91.3	82.6	69.2	150.0	2.46

The Dr. Dobbs benchmark, as modified by Terry Peterson and published in Vol. II, issue 1, compares as follows (time in seconds). HALGOL time is on an Apple (it shouldn't make any difference except for math chip support):

		SuperPET		HALGOL	
Macintosh Basic	Apple II	mBasic	BASIC 4	(No math chip)	(32081 chip)
586	488	780	552	16.8	3.7

We do wish that the language optionally allowed indented lines for those with weak eyeballs and a liking for structure; we'd love to see the option of IF... ...ELSE...ELSEIF...ENDIF and various loop and case structures, having spent far too much of our life puzzling out and amending the code we wrote last month to ever go back to using unindented, unstructured sphaghetti code. We don't insist that everybody must use structure (as Pascal does), we simply want the option, for we dislike a language whose programs aren't both easy to read and easy to maintain (in our business, you expend far more effort maintaining programs than ever you did to write them--the doggone world keeps changing!).

We'd like to see a single symbol (! ', or whatever) to block off a comment line, instead of that darn REM. We miss the power of "index" to locate a substring within a string, though Hal tells us he definitely will implement it. We dislike the requirement that you must dimension every string (you default to strings of 16 characters if they are not dimensioned). Everybody else who sees HALGOL tells Hal how he ought to redesign it; why shouldn't we? After all, DA has only put $100,000 of its own money into creating it; why not ask Hal to have DA spend another tidy fortune to accomodate us? He can always say "No!"--and probably will.

Let us pass from the language to the hardware required to use HALGOL with SPET. We'll cover it in three sections:

1. Hooking Up: The hardware may be installed on machines of all vintages (2- and 3-board) without soldering. The instructions ask you to "tack" two power leads to the 8032 motherboard; these merely pick up +5 volts and ground, which you may do by poking the wires into the unused $9000 ROM socket pins #24 and #12, respectively, according to Terry P. [Terry notes that there may be one 8032 motherboard type which requires modification (no factory-made SPET used it); if you have one, Digital Acoustics supplies the parts for the change.] Everything needed for the hookup (cables, parts, instructions) is sent from DA as part of the Grande package, defined below:

2. The Beacoup Grande: The board on which the MC68000 and its associated RAM and support chips are mounted is named the Beacoup Grande, for reasons which'll soon be obvious. DA used to make a board named the Grande, which was equipped with 64K DRAM (Dynamic RAM) chips; the new 256K DRAM chips became so plentiful and cheap that Hal switched over to them, so that you may now equip SuperPET with up to two megabytes of RAM. There are two versions of the Grande, one with 150 nanosecond RAM, which is priced at left, and a second with 120 nsec RAM, with no wait state at all. Included in the prices are installation instructions, HALGOL and its documentation, some 8032 replacement EPROMS and demonstration software. Terry Peterson says the EPROMs are not of great interest to SPETters, for they require soldering/desoldering; if installed, they louse up 6809-side operations. Do not use them.

Vanilla Beacoup Grande With One Wait State	
0.5 Mbytes	$725
1.0 Mbytes	$815
1.5 Mbytes	$905
2.0 Mbytes	$995
Beaucoup Grande + No Wait State	
0.5 Mbytes	$741
1.0 Mbytes	$847
1.5 Mbytes	$953
2.0 Mbytes	$1059

Anybody who dives into this gear should subscribe to DTACK GROUNDED, Hal's newsletter, and should get back issues. It costs $15 for ten issues. Write DA at the address given in the first paragraph of this article.

3. Power Supply and Case: You must put the Grande in an enclosure and provide some power. DA makes a stainless-steel case and power supply which will hold any Grande plus an optional medium-high resolution graphics board and a fast, high-density (1 megabyte) disk-drive controller board (which is being worked on and is not yet available). The Grande works without either optional board. Case and power supply cost $195. Terry P. recommends that all but inveterate hardware hackers buy this rig. The power supply is reported adequate for the Grande with up to 2 megabytes plus the two optional boards noted (5 volts, 10 amps and line filter).

You should not expect to plug in the gear above and start using HALGOL and SPET as a 1 megabyte workaday computer. Though board design is firm and reported very

reliable, the only available software is HALGOL and Terry Peterson's ASSEM68K cross-assembler (see issue 2, Vol. II). This rig is for assembly language hackers, for those who want to use HALGOL for number-crunching, and those who see the 68000 as the microprocessor which will dominate the microcomputers of the next decade and who want a head start in writing programs for it.

USER COMMENTS ON HALGOL AND THE GRANDE

Nick Solimene of Woodhaven, N.Y. has had a Grande (early version, with 64K DRAM chips and one wait state) for several months; he's been using HALGOL quite a bit. We got a long letter from him, and summarize his comments below:

At first, I thought the BYTE calculations benchmark was too crude for a test of HALGOL, since within HALGOL it yields zero error and HALGOL does not use BCD (Binary Coded Decimal) as does BetterBasic. [Ed. The benchmark is at left. In many computer languages, numerous multiplications or divisions of decimal numbers cause serious errors, because the result of a single operation is not precise and this imprecision then is incorporated at each cycle into the result. The benchmark is designed to determine relative speed and relative precision, though somewhat crudely. We show in a table below errors generated by the math packages in several current languages.] I found that the routine runs in 104 seconds in BASIC 4.0 and also gives zero error; in microBASIC it requires 150 seconds and there is no reported error. This does not necessarily mean there was no actual error, but that the error, if any, is so small it cannot be stated. You can get an appreciation of this from the program, "Paranoia", by R. Kapinski, published in the Feb. 1985 issue of BYTE. He shows that in every math package there is a number he calls "ulpone"; it is the smallest number which, when added to or subtracted from one, can be recognized by the machine as different from one. Any error smaller than ulpone is invisible. For HALGOL, ulpone is 3.553E-15; for all of the SuperPET languages, 2.328E-10. Thus we know the errors are smaller than ulpone for the benchmark above whenever the error is reported as zero below.

```
100 t=time
110 number%=5000
120 a=2.71828
130 b=3.14159
140 c=1
150 for i%=1 to number%
160   c=c*a
170   c=c*b
180   c=c/a
190   c=c/b
200 next i%
210 print "Done"
220 print "Time =";time-t;"seconds"
230 print "Error =";c-1
```

No significance should be attached to the errors shown below unless they are related to ulpone and to the number of bits employed in the FP routines (in HALGOL, there are 48; in SuperPET, 32). The RMS (root mean square) error which was reported for Terry Peterson's version of the Dr. Dobbs Benchmark in Vol. II, No. 1, p. 4, more accurately reflects the quality of the transcendental math and FP arithmetic routines in HALGOL.

Errors Reported in Various Languages in Calculations Benchmark

HALGOL	microBASIC	BASIC 4.0	True Basic	BetterBasic	Turbo Pascal
0	0	0	-4.583...E-13	0	-1.3384...E-08

HALGOL is rapidly becoming more and more complete. The current CBM version from Terry Peterson is an excellent job of transporting HALGOL to SuperPET. I think his BIOS (basic input/output system) for the host and the 68000 board will prove to be useful as a nucleus for the I/O needed with languages or programs other than HALGOL.

Though HALGOL does not now have an immediate mode in the same sense that other BASICs do, you have operating commands, including LOAD, SAVE, HPR $aaaa,$aa... (that's hex, for a memory dump), HPOKE $aaaa,$aa... (poke to memory), LIST, RUN, SHIFT/RUN (a screen dump), RENUM, SELECT (to redirect listings, output, or error messages to printer and maybe to other devices in the future), as well as some others. The combination is a very convenient operating environment.

Line numbers in increments of 10 are automatically provided after entry of the first line unless you insert new lines by using new line numbers which fall beween existing line numbers. Syntax is checked as each line is entered; you get error messages if the line is in any way wrong. You can LIST a line with a "." followed by a line number if you wish, rather than using LIST #. With Terry's Commodore version, you may enter code in either upper or lower case; the compiller converts keywords to upper case.

Except for the use of labels instead of line numbers, GOTO and GOSUB are used in the customary way. I find it very convenient to be able to declare local variables in a subroutine, thus avoiding name conflicts. If only the language allowed parameters to be passed to subroutines, as we can do in microBASIC or in COMAL, HALGOL would be closer to being heavenly.

Concerning strings, I can understand your aversion to declaring string lengths. Early on, I had no particular need for strings as a result of FORTRAN upbringing. Then with CBM machines I got accustomed to them. But the garbage collection and contortions necessary to manipulate strings only prepared me for the change to dimensioned strings. I think this is a small price to pay to avoid garbage collection. [Ed. Maximum string length in HALGOL is 255 characters.]

HALGOL defaults to a string length of 16 characters. Since arrays must be dimensioned in any case, string arrays of other than default length require little additional effort [DIM a$(n%)100 instead of DIM a$(n%)]. Other strings require a DIM (i.e., DIM a$56). HALGOL provides STR(a$,7,4) to refer to the substring starting at character position 7 in a$ and extending for length 4, and a similar arrangement for arrays. STR may be used on either side of an assignment statement, and integer variables [e.g., STR(a$,i%,j%)] may be substituted for literal values. This rather nicely replaces left$, mid$ and right$. Of course, the usual + is used to concatenate strings on the right side of an assignment.

A nice feature of the current version of HALGOL is EDIT. The program statements "EDIT a$" or "EDIT a$(i%)" display the string and wait for you to insert, delete or overtype the string. On RETURN, the program continues to execute with the modified string. Because DATASAVE and DATALOAD let you store and retrieve string arrays from disk, and because of the large memory, EDIT should be very good for databases, mail lists, and such. The length function LEN is not yet implemented but I expect it will be. In summary, I do think HALGOL will be easy to use for fast string handling if we get both the LEN and the INDEX functions.

Unfortunately, at present it isn't possible to force indentation, even using statement separators and spaces. As for structure, only IF...THEN...ELSE is now available. I would like to have the sort of structure provided by either microBASIC or COMAL. At present, the lack of structure, the inability to pass parameters to subroutines, and things like MVA are a great pain. I know that some of these problems will disappear when HALGOL is finished.

In conclusion, however, I am very satisfied with the 512K Grande board, power supply, and case. Terry Peterson's ASSEMB68K cross assembler is very easy to use and quite fast. I have great expectations for HALGOL as it becomes more complete. Then the need for more memory and speed will inevitably arise. When it does, I expect to add the math board (which supports the National Semiconductor 32081 math chip) and a memory expansion board.

Hal Hardenbergh, in his newsletter, indicates that several new features will be forthcoming: 1) a DOS in 68000 to run the fast Mitsubishi 5-inch drives (1.2+ Megabytes) at very fast data transfer rates, 2) a high-resolution graphics board using a high-resolution monitor. Except for the expense and the fact that all this gear will be on an aging and discontinued SuperPET, the result should be an excellent system. [Ed. Just as we went to press, we learned that DA hardware and software will be adaptable to any terminal--not just to SuperPET--as soon as DA issues its DOS for the potent Mitsubishi drives and the high-res graphics board mentioned above. Which means that the DA gear is not tied forever to an aging and discontinued SuperPET, but may be moved, en bloc , to other terminals. At the moment, SuperPET is merely a host, providing a screen, keyboard, disk drives and some I/O routines.]

A SIMULATED CENTRONICS PORT FOR SUPERPET

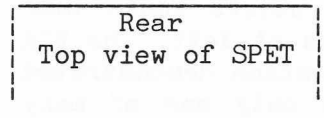
by P. J. Rovero
U.S. Naval Air Station FPO New York 09523

[Ed. Josh Rovero for some months has been stationed overseas, where telephone calls must pass through telephone systems owned and operated by governments (we refuse to use the words "run by" because the systems may crawl or refuse to run at all). Add to this notorious unreliability the costs of transocean transmission, and we aren't surprised by Josh's resorting to radio as a simpler and far more reliable way to communicate.]

I have a special modem connected to my serial port which allows me to telecommunicate by radio; it works well. But I needed a way to control the radio set itself from SuperPET; that set was configured to accept commands using Centronics parallel protocol, which isn't supported by our software. I therefore wrote such software to output through the User Port on SuperPET. With it, you may continue to use both the IEEE and Serial Ports and send Centronics output through the User Port--not only for my radio, but for printers which use Centronics output.

This article introduces you to the problems of simulating a Centronics port. The next article will provide a filter program which lets you couple any Centronics-compatible printer to the SuperPET user port.

The location of the User Port on SPET is sketched at left; it lies just to the right of the IEEE port most of us use (looking at SPET from the top or front). You may connect to it with the same type of push-on connector used on the IEEE. The pin identification is shown below, viewed from the rear:



1	2	3	4	5	6	7	8	9	10	11	12	Top Connections
A	B	C	D	E	F	H	J	K	L	M	N	Bottom Connections

The "G" and "I" identifications or pins are missing; the list is not in error. The table following correlates the pins and signals for the SuperPET User Port and for the Centronics printer plug. The dashed lines (----) show connections between pins. Use either the NACK or BUSY protocols, not both, for CA1!

User Port Signal	Pin		Centronics Plug Pin		Signal
CA1	Use	B	10\	Use	NACK
CA1	One!	B	11/	one!	BUSY
CB2		M	1		NSTROBE
GND		N	16		GROUND
PA0		C	2		DATA 1
PA1		D	3		DATA 2
PA2		E	4		DATA 3
PA3		F	5		DATA 4
PA4		H	6		DATA 5
PA5		J	7		DATA 6
PA6		K	8		DATA 7
PA7		L	9		DATA 8

The User Port signal names are those of SuperPET's 6522 VIA (Versatile Interface Adapter), which is already connected to the user port. When we discuss the VIA, we must employ the names of the VIA signals; likewise, when we discuss printer connections, we must use the Centronics names for the signals--even though they are wired together and comprise one line of the circuit.

CB1 of the VIA comes out on pin 8 of user port. Despite the programming manuals, I was never able to get CB1 to sense (input, read, or test) the logic level of a line I knew was changing. I therefore had to abandon any use of CB1, and used CA1 instead. It works well.

The terminology on protocols may be a little confusing. Both STROBE and ACK are negative logic (often shown by an underline above the names, as at left), but we can't print them that way on a Commodore machine, so we'll have to make do with NSTB or NSTROBE and NACK. BUSY is a positive logic signal (+5 volts when true). We wait for it to go "low" or false, which shows "unbusy". That is the same as waiting for NACK to go low (meaning "true, acknowledged"). When we get a negative transition (no +5 volts) on SPET's user port Pin B, the protocols noted below are satisfied; we can transmit.

$\overline{STB}$ $\overline{ACK}$

Many printers and other devices utilize the Centronics parallel protocol, which provides for 8 data bits in parallel with a two or three wire "handshake." Most devices use one of two different two-wire handshakes, as shown at left. My programs use the strobe/acknowledge method. They configure the 6522 VIA in SPET as an output port which can send a strobe pulse and can sense the acknowledge pulse sent back by the printer or other device.

Strobe/acknowledge
(both signals with negative logic)
Strobe/busy
(strobe negative, busy positive)

Port A of the 6522 VIA is already connected to the SPET user port. Pins C thru L are the eight data pins for the eight data bits; C=1 weight bit; L=128 weight bit; pin B is acknowledge (CA1), and pin M is strobe (CB2), an output. Always connect to the User Port with the computer turned OFF. Also note that negative logic means that the pulses occur when the lines are pulled low (nominal zero volts), and that the lines are high (+5 volts) in the absence of a pulse.

The 6522 VIA is a very flexible device. It can also be a very confusing device. A complete description of the VIA and its capabilities and quirks would take many pages. You'll find good background material in the books at left. The VIA configuration demonstrated here is only one of many that are possible. In the first application, I control a radio transceiver through a Centronics compatible interface on the transceiver itself, and do it from microBASIC with the assembly language program shown in listing 1.

(1) Programming the PET/CBM, Raeto West, pp 386-390
(2) PET Interfacing, Downey & Roberts, pp 33-50
(3) 6502 Software Design, Leo J. Scanlon, pp 192-218

Next issue, we'll see how to output any disk file from main menu to a Centronics printer hooked to the User Port (which probably interests more people than my radio application). Even so, the listings below may be useful for those who want to know how to use the User Port, as well as how to do it from language.

The program in listing 1 has only one purpose: to control the transceiver. It holds two subroutines: (1) set_via, which configures the VIA for Centronics simulation, and (2) send_it, which I call whenever I output a command string to the transceiver. The first subroutine is used only once, at start of session, to set up the VIA. The second calls a delay subroutine to time the strobe pulse. I need 21 milliseconds; most printers require at most 1 millisecond. The program, of course, also sends the commands from SuperPET to the transceiver.

The assembly language program in listing 1 is called from a microBASIC driver. The code gives you an idea of the bit-twiddling required to use the VIA. One interesting note: you can hear this program working. The CB-2 line is the same one used for the SPET "bell." When the strings are output, each CB-2 pulse is heard as a click.

Listing 1

```
;icom9.asm       ; Routine via_set sets up the VIA for communications by imitat-
                 ; ing Centronics protocol. CB2 sends NOT STB; CA1 senses NOT
                 ; ACKnowledge. This is listing 1.

                 ; Routine send_it sends commands to an icom 720 transceiver,
                 ; setting mode (USB, LSB, RTTY, CW, AM); VFO is set to "a";
                 ; frequency may be set from 0.1 to 29.9999 MHz.
                 ;
                 ; Delay sets the length of the NOT STB pulse; a value of $0A60
                 ; creates a 21 millisecond delay.

                 ; Versatile Interface Adapter (via) Addresses
port_Aca1        equ $e841 ; port A with CA1 handshake
ddra             equ $e843 ; Data direction register
acr              equ $e84b ; Auxiliary control register
perif            equ $e84c ; Peripheral control register
ifr              equ $e84d ; Interrupt flag register
port_A           equ $e84f ; Port A without handshake

set_via lda #255
        sta ddra     ;Make all port_A lines outputs
        lda acr
        anda #227
        sta acr      ;Disable shift register
        lda perif
        anda #254
        sta perif    ;Set ca1 for negative transition
        ora #128
        ora #64
        sta perif    ;Set up CB2 for proper pulse
        ora #32
        sta perif    ;Put cb2 high
        rts

send_it lda port_Aca1          ;Reading port_Aca1 clears ifr
        ldx #command
```

```
        loop
                lda ,x+
                quif eq
                sta port_A      ;Put data on output port
                lda perif
                anda #%11011111 ;NOT 32
                sta perif       ;Pull CB2 low for
                stx count
                jsr delay       ;about 21 ms.
                lda perif
                ora #32
                sta perif       ;Put CB2 high
                loop
                  lda ifr       ;Acknowledged ?
                  anda #2       ;2 weight bit of ifr should be set
                until ne        ;keep waiting if not
                lda port_Aca1   ;Clear ifr
                ldx count                               ; .CMD file shown below:
        endloop                                         ;
        rts                                             ; "icom9"
                                                        ; org $7F01
delay   pshs x                                          ; "icom9.b09"
        ldx #$0a60    ;causes 21 ms delay
        loop
           leax -1,x
        until eq
        puls x
        rts

command fcb 00,00,00,00,00,00,00,00,00,00 ; These buffers filled from language.
count   fcb 00,00
        end
```

Shown below is the heart of a microBASIC program which uses the ML module above. It configures the VIA, and then sets frequency by poking the necessary values to the buffers above and then SYSing the ML module itself. Anyone who wants a copy of the entire program: send a self-addressed, stamped envelope to Editor, PO Box 411, Hatteras, N.C. 27943.

```
100 ! machine language loader from SuperPET Gazette Vol. I, issue 13, p 220.
110 if loaded=0                                     ! Program by P.J. Rovero
120    loaded=1                                     ! April 4, '85.
130    poke (hex('22')),hex('7f'),hex('ff')         ! Set memend_ to $7fff
140    chain "icom9.mod,prg",names                  ! load ML routine
150 else
160    poke hex('22'),hex('7e'),254 ! MemEnd_ to $7EFE to protect ML routine.
170    open #40, "keyboard", output
180    print #40, "delete 100-200"+chr$(13)+"edit"+chr$(13)+"run"+chr$(13)
190 endif
200 stop
210 ! Make one call to set up the VIA properly.
220 call set_via
230 ! ...remainder of program displays on screen the various modes available
240 ! and asks for the frequency setting desired. When this data is input,
```

```
250 ! it is poked to the command buffer in the ML routine, which is then
260 ! SYS'd as shown below. The command buffer starts at $7F62.
270
280 for jj=1 to 9
290    poke (hex('7f61')+jj),ord(icom_freq$(jj:jj))
300 next jj
310 poke (hex('7f61')+10),0     ! End the string with a 00
320 print icom_freq$
330 call send_it
340
350 proc set_via                ! SYS the address of "set_via"
360   sys hex('7f01')
370 endproc
380
390 proc send_it                ! SYS the address of "send_it_" in the
400   sys hex('7f23')           ! ML routine
410 endproc
```

The technique is easy to extend. A simple "filter" which passes a disk file through the simulated Centronics port will be printed in the next issue of the Gazette; it may be used to send any disk file to a printer which accepts the Centronics protocol, and obviates the need for any hardware interface except a home-made cable.

AN ALARMING ARTICLE
by Loch Rose

Do you have a task that gets along pretty well without you most of the time but intermittently needs some operator help? I supply here a routine that will literally make SuperPET cry for your attention from the built-in speaker. Because that speaker is none too loud, I experimented to find the most aggravating sound, voted most likely to jerk you rudely from your hammock. To make the alarm "ring" twice, you need only insert 'call alarm(2)' into your program. To make the alarm ring until you press a key, insert 'call alarm(0).' With minor revisions, it should work in almost every SuperPET language.

```
9000 proc alarm(repeat%)      !'repeat%' is number of alarm signals requested
9010 if repeat% > 0
9020    for k = 1 to repeat% ! If repeat% <= 0, repeat alarm until user hits
9030       call noise        ! a key.
9040    next k
9050 else
9060    print : print "HIT ANY KEY TO STOP NOISE: ";
9070    poke 301, peek(303)                          ! empty keyboard buffer
9080    loop
9090       call noise
9100       get keypress
9110    until keypress
9120    print : print
9130 endif
9140 endproc
9150
9160 proc noise                                      ! actually makes the noise
9170 location = 59464
9180 poke 59467, 16 : poke 59466, 15
9190 for i = 1 to 8
```

```
9200     for j = 6 to 10
9210        poke location, j
9220     next j
9230 next i
9240 for i = 1 to 8
9250     for j = 71 to 75
9260        poke location, j
9270     next j
9280 next i
9290 poke 59467, 0
9300 endproc
```

[Those of you with 8250 drives may not be interested in a new 1001 drive; even so, read the section on REL files at the end of the the article following. Ed.]

AN INEXPENSIVE ONE MEGABYTE DRIVE, DEVICE SWITCHING, MEMORY WRITES, AND RUNNING PROGRAMS IN BEDIT

Most of you probably have seen ads for the Commodore SFD 1001 single drive; it seems to be, in performance, one-half of an 8250 drive, double-sided, with a formatted capacity of about one megabyte. We've seen it priced from $199 to $399. It's an IEEE device. We learned that Associate Editor Reg Beck had bought one; his SuperPET and he are happy with it, in event you're thinking about supplementing that old drive. It is sold by Progressive Peripherals & Software, 2186 South Holly, Suite 200, Denver CO 80222 (303) 759-5713, and is being advertised and discounted pretty heavily. See your favorite computer magazine.

Reg Beck sent us a short article describing his struggles to get up and running with the 1001 drive; because you face these problems with any second drive and we suspect a lot of you may get one, we expand on Reg's material below.

Once you buy a second drive, you gotta problem. Any 8x50 or 4040 and the new 1001 will default to device 8; you can't talk to either drive until you software reset one of them to a new device number. Then you become annoyed. Every time you turn the drives on, you must 1) Turn on the drive which will be device 9; turn off the one which will be device 8; 2) software set the active drive to device 9; and, 3) turn on the second drive, which then becomes device 8. If you crash, lose power, or reset to 6502, both drives reset to device 8; you must again trudge through the setup procedure.

The obvious solution is to hard-wire one of the drives as device 9 (see below), to leave both drive switches on, and to connect the power cords of both drives to a single outlet controlled by a master switch.

In the remainder of this article, we'll cover how to change device number by software and how to do it by hard-wiring. There are three ways to go about a software reset. The easiest is to employ the program on the first ISPUG Utility disk, "chgadrs.mod", written by Terry Peterson. It loads from main menu; keep it on your language disk and call it whenever you want a device reset. Oh, you don't have ISPUG Utility Disk I?

The second way to get a software reset is to write a program which sends the proper commands to the disk drive, using the Memory-Write (M-W) commands found in the various editions of the User's Manual for your disk drive. We simplify matters by printing below the easy way to determine the M-W code for any legal device number.

You may write your own M-W program, but it's easier to copy or to revise the one written by Terry Peterson and published on p. 115 of Vol. I, No. 9. We have extracted Terry's algorithm below; it will generate the M-W code to set any device number. The code values found below in angle brackets are the decimal ASCII codes to transmit. Write a program in any language to send them to your drive.

---

new_device = [choose] new_var = new_device+32 new_talk = new_device+64

The command to be sent is, in general form: M-W<12><0><2><new_var><new_talk> . We give a specific example for a change to device 9; the variables then take on these values: new_device = 9, new_var = 9+32, new_talk = 9+64, as shown at left.

M-W<12><0><2><41><73>

It is quite simple to use the algorithm to determine the commands for any legal device number. The command is sent to the command channel of the drive itself (to ieeeX-15, where X is the present device number). The Commodore manual says that the drive should be initialized before you send a device-change command, but we and Reg Beck have never found it necessary.

Now, having the code, ask yourself if you really want to load a language and a program each time you need to change device number. Obviously you won't. What are your other alternatives with software reset?

1. Well, poor Bodsworth sent program output to disk and then COPied the file to a drive. Gee, that works fine at the start, but when SuperPET tries to close the file on the old device, it can't. The old device isn't there any more. Poor Bodsworth is <u>still</u> bewildered by that one.

2. Write the program in BEDIT, BEDCALC, or DEVCALC, ISPUG's superEditors--and then PRINT it from any of them. Poor Bodsworth is still unaware that you can do this, as we show below by using some of Reg Beck's programs.

PRINTING MACHINE LANGUAGE FROM BEDIT Joe Bostic designed BEDIT in its various forms to send any possible decimal or hex value to any device--a drive, a plotter, a printer--with the PRINT command. First, let's define the difference between a character and a "literal." We all know that any editor sends the ASCII code for a character to any output device, "A" being transmitted as decimal 65, for example. You may send any "literal" value--the actual value itself-- by enclosing the value in angle brackets, <>; e.g., <$20> transmits hex 20, an <11> sends decimal 11. You <u>must</u> PRINT (not PUT) files containing <value> to transmit the literals. You may intermix normal text and <literals>.

The PRINT command sends as normal text all values in an editor file <u>except</u> the <literals>, which may be in either hex, decimal, or ASCII mnemonic; a CR may be PRINTed as <$0d>, <13>, or <CR>. Joe Bostic believes in being flexible.

Having this under our belts, let's send some M-W commands to change devices; we believe these commands will work on all 4040, 8x50 and 1001 drives. Put the screen cursor on the M-W command line and issue the PRINT commands shown, preceded by a dot (which sends only the M-W line to the disk command channel).

Change device from 8 to 9:
M-W<12><0><2><41><73>
. print ieee8-15

Change device from 9 to 8:
M-W<12><0><2><40><72>
. print ieee9-15

File (PUT) these commands to disk and GET them as you need them. Don't try to EXECute them from disk

in BEDIT, because the device number is changed in the midst of a file operation; SuperPET tries to close a file on the old device, not the new one--and can't, as Bodworth found out. In sum, PUT these commands to disk, GET them into BEDIT as needed, and then PRINT them to ieeeX-15.

SECOND HURDLE DEPT. You now decide to copy some old disks to the new, big 1001 disk. Sorry to tell you that there is no way to copy entire disks between devices in SuperPET in 6809 mode with the software which came with SPET. But you do have four ways to solve the problem:

1. You may rename all your files in capital letters and copy disks in 6502 mode; it's both impractical and slow.

2. You can buy MicroPIP and its manual from Waterloo (a recommended buy if you have a lot of disk work to do) but somewhat expensive at $75. See the review of that program in Vol. I, No. 9, p. 129. It copies whole disks flawlessly between devices, but is both slow and complex if you must copy selected files.

3. Write a program which will read an edited directory list and copy the files between devices. It's easy to make the original list with: "di disk to index" or similar commands; it's easy to edit the list; writing the program is equally simple. But you must first load an Editor to make and edit the list, and then load a language and a program to copy the files. That's slow and tedious.

4. Use BEDIT's batch file capability. This is the fastest and easiest way. In the tutorial and instructions for BEDIT are two programs: COPYFROM9 and COPYTO9. You make a list of the files to be copied by editing a directory listing, file it as "list:bed", and then execute either of the EXEC files above. The copying is swift, accurate, and simple.

Gee, who would have thought that buying a new disk drive could get so involved? Sorry it sounds that way; we suspect it will take the average user about an hour to set up and try the M-W commands and the COPY9 files. After that, handling files between devices is simple and easy; you'll find yourself switching disks far less often with the added disk capacity aboard.

THIRD AND LAST HURDLE DEPT. We repeat Reg Beck's sage words on how to copy or read REL files in 8250 or 1001 drives. If 8050-formatted REL files are to be copied (as from an ISPUG 8050 disk) to an 8050 formatted disk which is in an 8250 or 1001 drive, you must disable the Expanded Relative File (ERF) capability of the 8250 or the 1001--or the files will not copy. Reg says the commands at left do the trick. You may PRINT them directly from BEDIT or BEDCALC or stuff them into a batch file which uses EXEC. Be sure to reset the 8250 or 1001 to expanded ERF when you are through copying. You must disable ERF if you want to read 8050 formatted REL files on an 8250 or 1001. Last, a warning: You must initialize the 8250 or the 1001 with: p ieeeX-15.i <RETURN> before you send the ERF enable/disable. The lower case "i" initalizes both drives of an 8250 or the single drive of a 1001. You can send it in upper case if you want, or you can PRINT it instead of PUTting it. Doesn't matter, thanks to Joe Bostic.

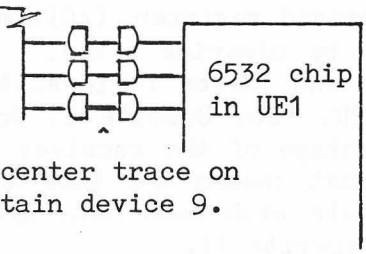
Disable ERF	
M-W<164><67><1><255>	
Enable ERF	
M-W<164><67><1><0>	

* * *

The early disk drive user manuals we received from Commodore held no material at all on changing device numbers on drives; for those in the same boat, we summarize the information below.

HARDWARE DEVICE NUMBERS We always work with two drives; the second is hard-wired as device 9 for good and obvious reasons. Unless you know your way around electronic gear, you should have the modifications we discuss done by someone who does. Lift the lid on any 8x50 or 4040 drive; look at the top board--the one inside the lid itself. Near the bottom of the board, you'll see a large chip (ours is 6532) in socket UE1. Just left of the socket are three circular spots in the traces on the board itself:

Device No. Wanted:	Pin 22	Pin 23	Pin 24
8	0	0	0
9	0	0	1
10	0	1	0
11	0	1	1
12	1	0	0



0 means leave alone. 1 means cut trace or bend this pin so that it doesn't enter the socket when chip is replaced.

Cut the center trace on 24 to obtain device 9.

Pins 22, 23, and 24 of the chip in UE1 are normally grounded by the traces.

You have three choices: 1) remove the chip, bend the appropriate pin, and reinstall the chip, 2) scratch out (cut) the trace as shown in the table above for permanent change. We've marked the place to cut for device 9 in the sketch, at the small center trace between the large, circular spots shown (see the carat ^ for exact position); last, 3) if you feel this will reduce the resale value of a drive, you can set in a switch, but it is not a job for amateurs. The switch cannot "bounce" (make several contacts); it cannot affect the circuit it's in, and it should never get flipped accidentally. It should only be "switched" when the drive is off unless you use the UJ command defined in the drive manual. Pick the method which suits your needs.

THE ACIA AND HOW TO USE IT FOR INTERRUPT-DRIVEN TC

by Loch Rose
Associate Editor

The 6551 ACIA in SuperPET is the chip resposible for running SuperPET's serial port. (ACIA means Asynchronous Communications Interface Adapter; we'll later see that the title is appropriate.) This article outlines how to use the ACIA when you write interrupt-driven telecommunications programs; it owes a great deal to Terry Peterson's article in the April, 1983 issue of MICRO, to which you should refer if you need more detail.

Basic Information & Terminology Dept.: a byte consists of eight bits, numbered (as a matter of convention) 7 through 0 from left to right. A bit is set if it contains 1, and is clear if it contains 0. On a terminological note, RECEIVED data is that which comes into the computer through the serial port, from a modem for example, and TRANSMITTED data is that sent by the computer, such as text to a serial printer.

I will now describe some of the functions of the ACIA registers. There are four registers in all:

The data register (ACIAdata for short) at hexadecimal address $EFF0 is the register through which both received data and transmitted data pass. To transmit a byte of data, you need only store it in ACIAdata; the ACIA then takes care of actually transmitting it. But since the ACIA places received data bytes in ACIA-

data, you should check ACIAdata for a received byte before placing an outgoing byte in it. ACIAstat (below) allows you to do so.

The status register (ACIAstat) at $EFF1 gives you the status of various operations. (1) Bit 3 is set when the ACIA receives a data byte, and cleared when you write to or read from ACIAdata. (2) Bit 4 is cleared when you store an outgoing byte in ACIAdata, and set when that byte is actually transmitted. (3) Bit 7 is set when the ACIA requests an interrupt (of which more below).

The command register (ACIAcmd) at $EFF2 allows you to request a receiver interrupt by clearing bit 1. If you do so, then whenever the ACIA receives a data byte and places it in ACIAdata, it sends an interrupt request signal (IRQ) to the CPU. (See Gazette I, No. 15, p. 276ff for articles on interrupts.) To take advantage of the receiver interrupt, you must supply an interrupt-handling routine that checks the ACIA and removes the received byte from ACIAdata. This method deals with each data byte as soon as it arrives, so any following bytes cannot overwrite it.

The control register (ACIAcont) at $EFF3 controls serial port parameters such as baud rate. You can set it with SETUP or with subroutine sioinit_, which is documented on p. 185 of the Assembler manual.

I shall now outline some machine language routines that perform interrupt-driven telecommunications. At the end of this article, "main.asm" will weave them together. If "main" and the subroutines following are assembled, you may link this package into a working telecommunications package which loads at main menu.

Should you choose to use the package, the command file is printed at the left.

```
"tc"
include "disk/1.watlib.exp"
org $1000
"main.b09"
"init.b09"
"signoff.b09"
"send.b09"
"handler.b09"
```

```
;init.asm - Initializes input buffer, connects user interrupt-handling routine.

xref  IRQhndlr,conbint_,nextread,nextstore,inbuffer
xdef  init
ACIAcmd  equ   $eff2

init     sei                    ;Mask off interrupts.
         ldd   #inbuffer        ;Initialize our own input buffer.
         std   nextread         ;Set read pointer to start of the buffer.
         std   nextstore        ;Set write pointer to the same address.
         ldd   #8               ;Offset of 8 bytes into IRQ handling table.
         pshs  d
         ldd   #IRQhndlr        ;Address of our interrupt-handling routine.
         jsr   conbint_         ;Connect user interrupt-handling routine.
         leas  2,s
         lda   ACIAcmd          ;Command register--
         anda  #%11111101       ;Clear bit 1, enabling receiver interrupt
         sta   ACIAcmd
         cli                    ;Reenable interrupts.
         rts
```

```
;handler.asm  - A user interrupt-handling routine.

xdef IRQhndlr,input,nextread,nextstore,inbuffer,endbuffer

regIRQ   equ   $de0b          ;Address of usual interrupt-handling routine
ACIAdata equ   $eff0
ACIAstat equ   $eff1

IRQhndlr bsr   input          ;Is there a received byte in ACIA?
         if    ne             ;NO, interrupt was generated by clock.
           jsr   regIRQ       ;Go to usual interrupt-handling routine.
           bsr   input        ;Recheck for received byte (one might arrive
         endif                ;                  during the previous routine.)
         rts                  ;Return from interrupt process.

input    lda   ACIAstat       ;Check ACIA status:
         anda  #%00001000     ;Is bit 3 set? (= byte received).
         if    eq             ;NO, byte not received.
           andcc #%11111011   ;Clear zero flag (means 'nonzero result found')
         else                 ;YES, byte received, so
           ldb   ACIAdata     ;read received byte.
           andb  #%01111111   ;Clear bit 7 of received byte.
           ldx   nextstore    ;Load input buffer store pointer,
           stb   ,x+          ;store received byte in buffer.
           cmpx  #endbuffer   ;Is pointer past end of buffer?
           if    hs
             ldx   #inbuffer  ;Yes, wrap pointer to start of buffer.
           endif
           cmpx  nextread     ;Has pointer caught up to read pointer?
           if    ne
             stx   nextstore  ;No, store new pointer value.
           endif
           orcc  #%00000100   ;Set zero flag (means 'zero result found')
         endif
         rts

nextstore  rmb   2            ;Input buffer store pointer.
nextread   rmb   2            ;Input buffer read pointer.
inbuffer   rmb   80           ;An 80-byte buffer - you can make it any size.
endbuffer  fcb   0            ;Just a marker for end of buffer.
```

NOTE: I do not recommend simply printing received bytes directly on the screen, because the printing process can be too slow when data arrives at 1200 baud; you may lose bytes. That is the reason we employ a buffer.

Next is a routine that transmits one byte (to send a series of bytes, you simply call it repeatedly). You'll note that it checks for received bytes; this should not be necessary, because received bytes should trigger an interrupt which deals with them immediately. However, I find it necessary to do it this way to avoid losing an occasional byte, and will be delighted if someone can tell me why.

```
;send.asm - Subroutine transmits a byte supplied in register B.
```

```
xref  input
xdef  send

bputscn  equ   $d714            ;Routine prints char. in B register to screen.
ACIAdata equ   $eff0
ACIAstat equ   $eff1

send     pshs  b                ;Save a copy of outgoing byte on stack.
         clra
         jsr   bputscn          ;Print byte on screen.
         loop
           loop
             lda   ACIAstat     ;Load status.
             bita  #%00001000   ;Is ACIAstat bit 3, 'byte received', set?
             quif  eq           ;NO, byte not received, quit.
             sei                ;YES, received. Mask off interrupt,
             jsr   input        ;deal with received byte.
             cli
           endloop
           anda  #%00010000     ;Is bit 4, 'byte transmitted', set?
         until ne               ;YES, go ahead and transmit new byte.
         puls  b                ;Restore outgoing byte, and transmit
         stb   ACIAdata         ;by storing outgoing byte in data register.
         rts
```

Note that it is essential to wait until the 'byte transmitted' flag is set. You otherwise might put a new byte into ACIAdata before its predecessor is sent.

Finally, when you are through with TC, you should disenable the receiver interrupt and return interrupt-handling to the usual routine:

```
;signoff.asm - Disconnects user interrupt-handler, disenables receiver interrupt

xdef  signoff

regIRQ   equ   $de0b
ACIAcmd  equ   $eff2

signoff  sei                    ;Mask off interrupts.
         lda   ACIAcmd
         ora   #%00000010       ;Disenable receiver interrupt.
         sta   ACIAcmd
         ldd   #regIRQ          ;Address of usual interrupt-handler.
         std   $0108            ;Restore usual interrupt-handler.
         cli                    ;Reenable interrupts.
         rts
```

If you want to combine these routines for TC, use "main.asm", below. It employs the subroutines to create a working telecommunications program.

```
;main.asm - Mainline routine, which calls the subroutines above.

xref nextread,nextstore,inbuffer,endbuffer,init,signoff,send
```

```
kyputb   equ   $dd82          ;Gets character from keyboard into B register.
bputscn  equ   $d714          ;Prints character in B on screen.
service_ equ   $32            ;See end of program.

main     jsr   init           ;Connect user interrupt-handler.
         loop
           jsr   kyputb       ;Get character from keyboard.
           if    ne           ;If character found in keyboard,
             cmpb  #139       ;is it "shifted period" on the keypad (PF.)?
             beq   endit      ;YES, end program;
             jsr   send       ;else, transmit character.
           endif
           ldx   nextread     ;Load input buffer read pointer.
           cmpx  nextstore    ;Is buffer empty?
           if    ne           ;NO, input buffer not empty, so
             ldb   ,x+        ;load character from buffer, increment pointer.
             cmpx  #endbuffer ;Is pointer past end of buffer?
             if    hs         ;YES, wrap pointer to start of buffer.
               ldx   #inbuffer
             endif
             stx   nextread   ;Store new value of read pointer.
             guess
               tstb           ;Test character taken from input buffer,
               quif  eq       ;if character is a null, skip.
               cmpb  #10
               quif  eq       ;If character is a <line feed>, skip.
               jsr   bputscn  ;Print character to screen.
             endguess
           endif
         endloop
endit    jsr   signoff        ;Disconnect interrupt-handler.
         clr   service_       ;Causes program to return to main menu.
         rts
```

B I T S B Y T E S & B U G S by Gary Ratliff, Sr.

215 Pemberton Drive, Pearl, Mississippi 39208

We have completed a milestone in this column--all the details of how to create assembly language programs using the SuperPET have now been covered, the two-part series on math processing routines (printed last issue) being the end.

Now we are ready to turn our attention to the development of larger programs, using the ability of the Development system to create separate modules. We first introduce the concept of table processing and then advance to the study of command tables and their drivers.

Table processing is the heart of any complex system. A table may consist of simple one-character commands, as in the monitor, where the commands are, for example, m for modify, t for translate, d for dump, and so on. In high-level languages, a command table again must be processed, but the commands consist of full words such as: print, for, next, etc. How do we set up and use the tables?

We will introduce the art of processing tables with an example which is familiar to all. When you were a child, you probably sent encrypted messages to a friend.

To translate a message into code, each of you had a table which consisted of the alphabet and the code symbol which replaced it in your code. You converted the encrypted messages into English by looking up the coded letter and replacing it with its table value.

To keep the coding of this example simple, we'll restrict our message to lower case characters of the alphabet. The program follows. Before you go through it, please look at the label TABLE, at the end of program, for that governs what characters are substituted for those in our original message.

```
xref getrec_, putrec_, islower_, printf_          ; System routines to be used

buf1 equ $1500  ; Buffer 1 will hold the original message.
buf2 equ $1600  ; Buffer 2 will hold the encrypted message.
msize equ 80    ; Maximum size of the message will be one screen line.

; Get the message from the screen and store in buffer 1.
ldd  #msize     ; Get one line or less. This is P2 for GETREC.
pshs d
ldd  #buf1      ; The address of buffer 1 is P1 for GETREC.
jsr  getrec_    ; Get record from terminal. Number of chars read returns in B.
leas 2,s

; End the string with a null and print it for verification.
ldx  #buf1
pshs x          ; This address is the substitution address P2 of printf_.
abx             ; Add the number of characters read to address of buffer.
clr  ,x         ; This ends our original message with a null.

ldd  #msg1      ; Message 1 prints "Your text message was..."
jsr  printf_    ; and prints the original message.
leas 2,s

; Check message for illegal characters; abort if any found.
ldx # buf1
loop
  ldb ,x+
  quif eq
  cmpb #" "             ; Check for spaces.
  if ne                 ; If not a space...
    jsr islower_
    if eq               ; 0000 returns for FALSE--we found a character NOT in
      ldd # msg2        ; lower case alphabetics.
      jsr printf_
      swi               ; Error, so abort program.
    endif
  endif
endloop                 ; Well, if there was no error...

; We begin translating the message into code, using the TABLE at end program.
ldu  #buf1              ; Load address of original message.
ldx  #buf2              ; Set up to store the enciphered message in buffer 2.
ldy  #table -1          ; Load address of TABLE (and offset, since "a" will
                        ; be 1, "b" will be 2, etc.).
```

```
pshs x              ; We stack the address of the enciphered message.
loop                ; so we can print it at the end of program.
  ldb ,u+           ; Get a character from buffer 1,
  quif eq           ; Stop if it's the endstring null,
  cmpb #" "         ; bypass spaces;
  if ne             ; if not a space, encipher
    andb #$1f       ; AND ASCII code for character with Decimal 31.
    ldb  b,y        ; Offset to desired position in TABLE
  endif             ; (see comments below on effect of AND)
  stb ,x+           ; Store enciphered character in buffer 2.
endloop
stb ,x+             ; Store endstring null as last character.
```

[Ed. If you're confused by the ANDB #$1f, above: it changes the ASCII code number for a character to its alphabetic place; e.g., a (ASCII $61) becomes 1, and z (ASCII $7a) becomes 26. You may find it clearer to subtract: SUBB #$60.]

```
; print out the converted message.

ldd # msg3          ; We have already stacked the address of the encipherment
jsr printf_         ; and now print the that converted message.
swi

; data area
msg1 fcc "Your text message was:%n%s%n"
     fcb 0
msg2 fcc "text must contain only lower case letters.  ABORTED THE PROGRAM.%n"
     fcb 0
msg3 fcc "your coded message is:%n%s%n"
     fcb 0

table fcc "qwertyuiopasdfghjklzxcvbnm"   ; TABLE which enciphers the message.
      fcb 0
      end
```

```
"trans1"
org $1000
include "disk/1.watlib.exp"
"trans1.b09"
```

To the left we have our .cmd file for the program above. Note in particular the use of the index registers to save the places within the tables. One is needed for the input text buffer, another for the output buffer, and a third is required for the place in the translation table itself. This is a fairly simple but often-used technique for table translation.

We could expand this idea to a game in which we try to identify the characters contained in a hidden message. This idea is similar to that of the popular TV show "Wheel of Fortune." This exercise will greatly increase your understanding of the techniques of table handling, so we'll have a contest. Submit your solution to find a hidden message and to display it.

For example, have a screen line which displays the alphabet and highlights the letters which have been used; display a blank message line, which fills in as you guess correctly: ___ __ ___ ____ ___ ___ _____ ___ ____ ___ _____.
The hidden message might be "now is the time for all ispuggers to solve this problem." If you guess an "a", then "a" is highlighted on the alphabet line, and

the message line shows an "a" wherever one appears in the hidden message. We'll award a small prize for the best solution; if it isn't too long, we'll print the best program in a future issue. In any event, we'll name the winners.

No, I'm not going to type in each entry! Send the solution both as an .asm file and as an executable .mod file to me at the address shown at the top of this column, on 8050 disk. If you have a 4040, send the disk to ISPUG to be converted to 8050 format. ISPUG will return your disk and forward the copy to me.

Next issue, we will continue our treatment of table handling by taking a closer look at the structure of a command table and its driver.

THE APL EXPRESS

by REG BECK
Box 16, Glen Drive, Fox Mountain, RR#2, Williams Lake, B.C., Canada V2G 2P2

The absence of the diamond statement separator in SuperPET's APL can frustrate the user (when implemented, it allows several statements on one line). It probably was left out because Waterloo ran out of banks for APL interpreter code and because our APL was designed for schools, where beginning programmers are encouraged to write one statement per line.

I recently worked on a function which set the cursor sequentially to positions in a grid. A loop was slow and resulted in a sluggish cursor. I wound up with a 51-line function which could have been made quite compact using the diamond statement separator; each statement was only a few characters long. Readability would not have suffered because each pair of statements did the same thing, but at a different place on the screen. Unfortunately, there does not seem to be a way to synthesize a statement separator for statements which do not return results. Yet a simple function will work as a separator <u>if</u> each statement to its right returns a result and each statement to its left requires an argument. The separator shown below throws away each result returned to it as a right argument but permits any variables which are defined to be retained.

```
      ∇Δ[□]∇
[  0]    R ← A Δ B
[  1]   R←A

Δ:α                          ⍝IN DIRECT DEFINITION

      6 Δ 55 Δ ¯22
6
      ∇EXAMPLE[□]∇
[  0]    R ← EXAMPLE ;A;B;C
[  1]   R←(A+B-C)*2 Δ C←⍳4 Δ B←□ Δ A←□
      EXAMPLE
□:
      1 5 8 7
□:
      2 0 6 5
4 9 121 64
```

EXAMPLE is a simple demonstration of how statement separators can be employed to avoid several lines of code. These statements obviously could be written as a one-liner; appropriate parts of long one-liners are, however, often written as

separate statements for readability. Del is a good choice for this function, as it resembles the diamond and is the only non-alphabetic symbol which can be used as a function name.

* * *

The following pair of functions will remove and replace the blanked lines which are normally found between text lines on screen to make the text easy to read. Data for doing this in all Waterloo languages is found on page 69 of the Super-PET System Overview manual.

```
      ∇NOBLANK[□]∇
[  0]    NOBLANK
[  1]   □AV[□IO+4 40 5 5 7 33 9 7] □POKE 8ρ59520+0 1
      ∇BLANK[□]∇
[  0]    BLANK
[  1]   □AV[□IO+4 32 5 3 7 29 9 9] □POKE 8ρ59520+0 1
```

These functions are useful for graphics. You can't draw a decent box or grid without first removing the blank lines. Writing functions to draw grids can be a formidable task, since a vector must be created which contains the code for each character to be POKEd to the screen. This can run to several hundred numbers for a large grid. The function GRID, which follows, does all this for you. I received it some time ago from Jim Swift of Nanaimo, BC. GRID takes its time to draw a grid of the required size and sets up two global variables, CHAR9 and PK9. CHAR9 is the vector of characters and PK9 is a matrix of POKE locations required by QUAD POKE. Once these have been generated, FASTGRID will create the grid in a flash and GRID is no longer necessary for that particular job. [The reverse field box graphics (+128 supplements) need not be poked; they may be printed to screen freely. The box graphics which must be poked are the low CONTROLS, from ASCII 1 through 11. Ed.]

```
      ∇GRID[□]∇
[  0]    SIZE GRID CELL ;□IO;LINE;START;R;C;S;CL;TYPE
[  1]   ⍝DRAWS A GRID   SIZE IS THE NUMBER OF CELLS
[  2]   ⍝CELL IS THE SIZE OF A CELL
[  3]   ⍝START IS THE POKE NUMBER OF TOP LEFT CORNER
[  4]   ⍝  THE SYNTAX IS  4 6 GRID  3 4  FOR 24 CELLS EACH  3 BY 4
[  5]   ⍝  NOTE -  A SQUARE CELL SHOULD BE  2 4 OR 3 6 OR 4 8 ETC
[  6]   ⍝RECEIVED FROM JIM SWIFT
[  7]   □IO←1
[  8]   →0 ×⍳∨/  24 80 ≤ S←SIZE×CELL ←CELL-1
[  9]     NOBLANK
[ 10]       START←32768
[ 11]       □TC[5,(1+R←S[1])ρ3]
[ 12]       LINE  ← (4,1+C←S[2])ρ0
[ 13]       LINE[1;]←5,((C-1)ρ(CL←(CELL[2]-1)ρ2),9),6
[ 14]       LINE[2;]← (C+1)ρ1,CL+30
[ 15]       LINE[3;]←8,((C-1)ρCL,11),10
[ 16]       LINE[4;]←4,((C-1)ρCL,7),3
[ 17]     TYPE← 1,((R-1)ρ((CELL[1]-1) ρ2),3),4
[ 18]     (CHAR9←□AV[□IO+,LINE[TYPE;]]) □POKE PK9←START+(0,80×⍳R)∘.+0,C
      ∇FASTGRID[□]∇
[  0]    FASTGRID
[  1]    NOBLANK
[  2]   □TC[5,(1+1↑ρPK9)ρ3]
```

```
[  3]  CHAR9 ⎕POKE PK9
```

If you wish the program to fill the grid with information or to let a user enter information in the grid, you can write a function which calls up SETCURS, which will set the cursor to any screen row and column (not grid row and column).

```
     ∇SETCURS[⎕]∇
[  0]    R SETCURS C
[  1]   ⍝SETS POS OF CURSOR ON SCREEN TO ROW=R,COL=C
[  2]   0 0⍴(256 1+.×R,C)⎕SYS 45191
```

* * *

The Fibonacci series interests mathematicians and programmers. Writing a program to generate Fibonacci numbers is a good exercise. If you're not familiar with this series: The first and second Fibonacci numbers are both 1. Each subsequent Fibonacci number is the sum of the two immediately preceding it in the series. Thus, the first 10 numbers in the series are 1 1 2 3 5 8 13 21 34 55. We can easily write a recursive function in direct definition which will return a specific Fibonacci number (try it yourself before reading on). FIB 10 returns 55, the 10th Fibonacci number.

```
FIB:(FIB ⍵-1)+FIB ⍵-2:(⍵=1)∨⍵=2:1
```

Recursive functions can be SLOW in SuperPET APL. Try generating some larger Fibonacci numbers and see for yourself how the system slows down to a crawl.

* * *

In the last column we discussed APL idioms. Since then, The APL Idiom List, by Perlis and Rugaber, arrived. It is published by Yale University, Department of Computer Science, as its Research Report #87, 1977. Perlis and Rugaber have extended the meaning of idiom (from that given last column) to refer to any expression which occurs repeatedly in various contexts. The idioms which follow are from their list. (The FINNAPL Idiom Library from APL Press arrived today and is an excellent companion to Perlis and Rugaber.)

```
(+/∧\L=' ')⌽L                ⍝LEFT JUSTIFY A WORD LIST.
(1-(L=' ')⊥1)⌽L              ⍝RIGHT JUSTIFY A WORD LIST.
(∨\S≠' ')/S                  ⍝ELIMINATE LEADING BLANKS.
(⌽∨\⌽S≠' ')/S                ⍝ELIMINATE TRAILING BLANKS.
V←A,A+(×B-A)×⍳|B-A           ⍝A VECTOR OF INTEGERS FROM A TO B.
∧/(V⍳V)=⍳⍴V                  ⍝ARE ALL ELEMENTS OF V UNIQUE?
∧/1=+⌿M∧.=⍉M                 ⍝ARE ANY ROWS OF MATRIX M DUPLICATED?
S←(A,B)[↓(⍳⍴A),(⍴B)⍴N]       ⍝PUT THE STRING B INTO THE STRING A
                             ⍝AT POSITION N.
```

* * *

I return to writing this column after a couple of weeks away from the keyboard. School is almost over and summer holidays are near. We have been informed that the Computing Science curriculum guide will be ready for next school year and that, yes, the language will be Pascal-- sigh. Good APL books keep turning up but go on the shelf for later as all available time must be devoted to Pascal.

I received a letter from a happy owner of the direct function definition disk. He wanted to know if it was possible to extend the compiler to handle multiple conditions without rewriting it. This is readily accomplished through multiple functions. A direct definition of the signum function serves to demonstrate how this works. Sgn A has the value -1 if A<0, 0 if A=0 and 1 if A>0.

SGN: 1:ω≤0:*ELSE* ω
ELSE: 1:ω=0:0

The function ELSE handles the condition that A is non-positive.

WINDOW INTO OS/9
by Gerry Gold and Avy Moise

What the User Receives with Super-OS/9 Version 1.1: In mid-August, four disks will be distributed to all OS/9 owners. This provides the TPUG release of V1.1 [disk 1] (V1.0 with XCom9 telecom software), a subset of the Super-OS/9 ".asm" and ".b09" files that permit user redefinition of many defaults (e.g., the keyboard and I/O ports)[disks 2-3] and an extensively revised manual that includes sample source codes for various system utilities. All other source files which are not copyrighted by Microware Inc. and which are not included on the release disks, may be obtained on request from Avygdor Moise for $49, at 120 Torresdale Ave., Apt #1207 Willowdale, Ontario, Canada M2R 3N7, Tel. 1 (416) 667 9898 at home, or with the same prefixes 3954 at work.

XCOM9 is a telecommunication program with terminal emulation mode, upload and download capabilities, supports X-on/X-off and X-modem protocols. Since XCOM9 is a compact program, it can co-reside with any other OS/9 program.

Commercial Software available from TPUG: Although TPUG does not normally distribute commercial software, OS/9 programs, on Commodore disks, are being sold on a cost recovery basis. All of this software is tested, configured and transferred to CBM disk format before release. The following programs are available or in work; status is as shown. Cost is in $ U.S. and in ($ Canadian). Quoted prices for asterisked items is subject to a minimum number of orders; single quantity prices are higher. If you wish to obtain your software at a lower cost, you may have to wait until we fill enough orders (typically 5).

BASIC09: $110 (152). Three times faster than BASIC 4.0 with features of mBASIC.

FULL C COMPILER: $120 (166). Comes with a macro assembler/linker; supports 8/16 floating point digits precision.

PASCAL COMPILERS*: $175 (242). Two packages are being tested.

FORTRAN 77 COMPILER: $120 (166). Under test by Microware; release Oct. 1st, '85.

KANSAS CITY BASIC: $25 ($34). Approximate price. TRS80 compatible BASIC. Now being tested.

Other languages such as COBOL, Introl-C, etc. will be available on demand.

STYLOGRAPH III: $149 (189). Includes a full screen editor/word processor, spelling checker and mail merge...better than Wordstar & Wordpro, etc.

DYNASTAR: $90 (122) - approximate price Full featured screen editor/text formatter. Two versions are being tested. Spelling checker is optional.

DYNACALC: $99 (129)*. Full-featured screen-oriented spread sheet.

We have decided not to carry SCRED until a new version is available. TPUG will convert any OS/9 Level I software that is purchased from any vendor, provided

that the software is distributed on an 8" SS SD disk. There is a $25 charge per package for this conversion which includes the mailing, media and testing. Mail to TPUG at the address below; include your own disk format, name, and address.

PUBLIC DOMAIN SOFTWARE (Our Major Interest):

XLISP: (avail. July, 1985) A subset of LISP and SMALL TALK 80, artificial intelligence language. Includes object code (20K ready to run), full source in C and manual.

XCOM9: (avail. August, 1985) A 'Freeware' terminal program includes executable object, full assembler source, manual.

Utilities Package, to be available in September, 1985:

RatAsm: OS/9 filter that converts Waterloo structured assembler to OS/9 format.

COP: copy and replace a file.

TREE: Display hierarchical directories.

WC: A word count facility.

GREP: String search and display facility.

AT: Submit jobs at specified times.

STRIP: Remove/add file control characters.

DATES: Personal secretary and reminder service.

Many OS/9 programs are available on the OS/9 SIG on Compuserve, if you are a member of the OS/9 National Users Group ($25 to P.O. Box 7586, Des Moines, Iowa, 50322). Before you spend time and money downloading this software, remember that TPUG already has much of this library and is converting it to run on SuperPET. The user's group is still worth belonging to if only for MOTD, the newsletter.

As for our efforts, please remember that the SuperPET portion of TPUG is small and that it sometimes takes a few weeks to process an order or to return a request for information. The work is done by volunteers such as Avygdor (Avy) Moise, Gerry Gold, Bill Dutfield and others. In other words, please be patient and call if there is any problem.

To our knowledge no two board SuperPETs have had any problem with OS/9. Almost all problems with 3-boarders have been solved; we are still working on a few, especially on the early 3-board machines with soldered memory chips.

As of September, TPUG members will have also be members of DELPHI information service where TPUG will store public domain SuperPET and OS/9 software. There will be monthly workshops to bring SuperPET and Super-OS/9 users anywhere in the world in contact with persons who have the expertise to help and to advise.

If you do not receive your copy of V1.1 by mid-September, please call or write TPUG. Observe TPUG's new address and telephone number listed below. Toronto Pet Group Inc., 101 Duncan Mill Rd. Suit 7G, Don Mills Ontario, Canada M3B 1Z3, Tel. 1-416-445-4524.

* * *

Tips on Using OS/9 : Part I

1. Formatting a new floppy on your 2031, 4040, 8050, 8250 etc.: The CBM Disk utility cannot format a disk since the OS-9 file system is contained in one CBM

RELATIVE file, "OS9 DRIVE A".

To format a new disk, use the BASIC 4.0 program, 'FORMAT.OS/9', which is on your system disk. When the disk is ready, insert it in drive 1 and type the following commands:

```
OS9: dir /d1  ¦ This will initialize the system pointers (you have to do it
ERROR #241    ¦ only once before you access a drive for the first time).

OS9: format /d1 ¦ This will build an OS-9 file system inside the RELATIVE
                ¦  file. At this point, you are done.
```

The number of blocks (sectors), which are allocated by 'FORMAT.OS/9' and the OS-9 'FORMAT' utility should be equal to the number of tracks requested x 16. The following values are recommended:

CBM Drive Model	Sectors	Tracks(cylinders)	Capacity
2031,4040,8050	640	40	160 K
2031,4040,8050	624	39 (system disk)	156 K
8250	4000	250	1000 K

2. If the above procedure is too lengthy, you may speed it up in two ways. The first is to follow the above procedure only once, then use the CBM DOS DUPLICATE command to create additional file systems. The second is to wait until we produce second generation direct access disk driver software that will eliminate most of the difficulties. Meanwhile, as a temporary solution, users may build two logical file systems on each 8050 drive, extending the storage capacity of a single 8050 to about 350K. We'll cover this matter in more detail next issue.

This journal is published by the **International SuperPET Users Group** (ISPUG), a non-profit association; purpose, interchange of useful data. Offices at PO Box 411, Hatteras, N.C. 27943. Please mail all inquiries, manuscripts, and applications for membership to Dick Barnes, Editor, PO Box 411, Hatteras, N.C. 27943.

Table of Contents, Issue 6, Volume II

SuperPET Gazette
PO Box 411
Hatteras, N.C. 27943
U.S.A.

First Class Mail
in U.S. and Canada.
Air Mail Overseas.