Ipso Facto

The A-C-E Magazine

FEBRUARY 1981 **ISSUE 21**

INDEX INDEX PAGE

ASSOCIATION OF COMPUTER EXPERIMENTERS 1981 EXECUTIVE

President:		Program Co-ordinator:	
John Norris	416-239-8567	Jeff Davis	416-643-1578
Past President:		Training Co-ordinators:	
Ken Bevis	416-277-2495	Fred Feaver	416-637-2513
		Ken Bevis	
Secretary/Treasurer:			
Mike Franklin	416-878-0740	Software Co-Ordinator:	
		Wayne Bowdish	416-388-7116
Hardware Co-ordinator:			
Anthony Tekatch	416-957-7556	Editor:	
		Fred Pluthero	416-389-4070
Hardware Production and Sales:			
Fred Pluthero	416-389-4070	Editorial Staff:	
Publishing Committee:			
Dennis Mildon	416-385-0798	Sharon Swindells	
John Hanson	416-637-1076	Consultant:	
		Bob Silcox	416-681-2848
Membership Co-ordinator:			
Bernie Murphy	416-845-1630	Draughtsman:	
Don MacKenzie	416-676-9084	John Myszkowski	416-529-0250

ARTICLE SUBMISSIONS:
We can always use lots of software and hardware related articles of all types. Inasmuch as editing consists of taking the path of least resistance, 'camera ready' articles stand the best chance of getting in. Camera ready means typed, single spaced, reasonably error free and done with a dark ribbon. Diagrams should be large and clear (we can reduce them) and clearly labelled. Don't let camera ready scare you off. If you don't have access to a typewriter, by all means send in what you have, we still want to see what you've been up to.

Some important notes: First, please send us your original manuscript, not a photocopy. The quality of most photocopies is invariably poor and such articles get pushed to the back of the editorial 'stack'. Second, make sure your diagrams and programs are accurate. We have enough trouble with errors on our part; there's no way we'll ever catch yours.

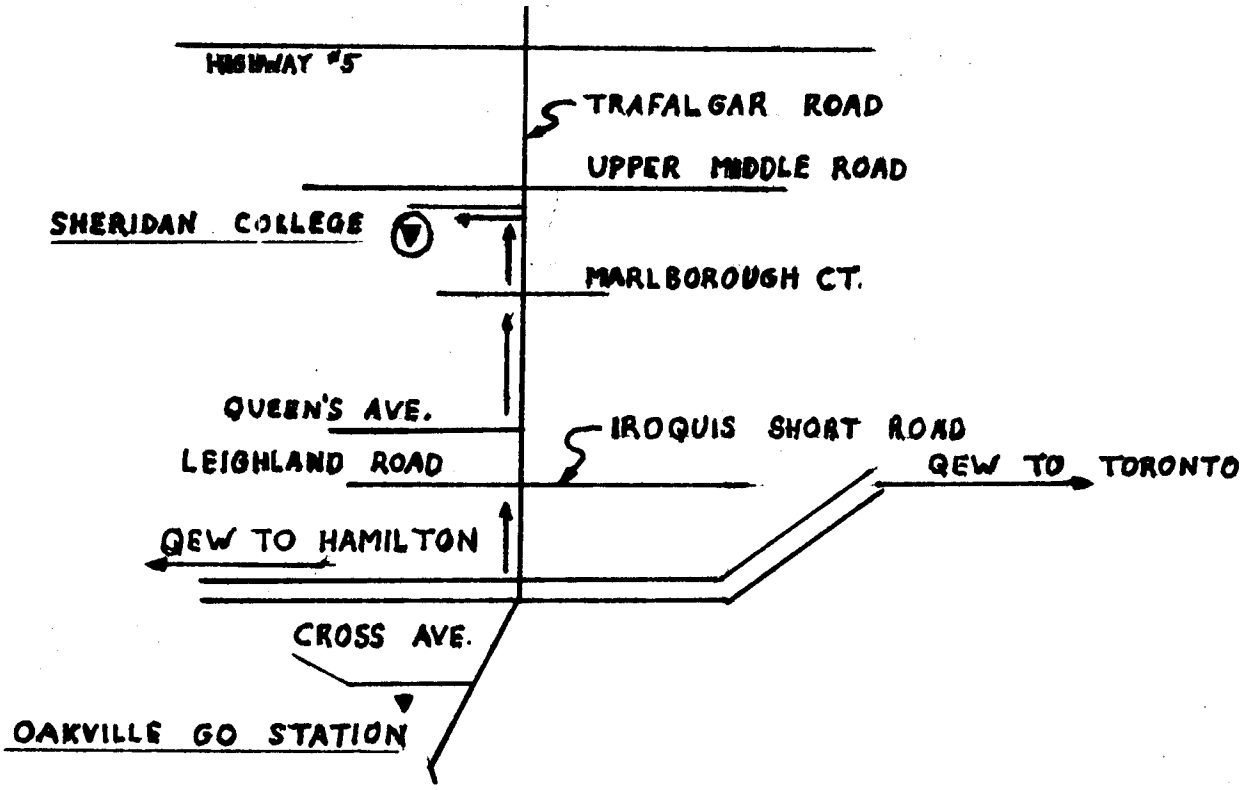
MEMBERSHIP RENEWALS:
This is a bit pointless; if you got this issue, you're fully paid up. To help you keep track, a note designating the status of your subscription should appear on the mailing label of your issue. This will most probably take the form of '2 of 6'.

US MEMBERS:
We understand that a number of our US members are having trouble obtaining Canadian currency. Don't bother, send in your membership renewal (or order for DeFacto, hint, hint) in US funds. This also applies to overseas members, if you can't come up with Canadian money easily enough, by all means send cheque or money order in US funds.

SEND ALL A.C.E. CORRESPONDENCE TO: Bernie Murphy,
102 McCraney Street,
Oakville, Ontario. L6H 1H6
Canada.

MEETINGS:
Meetings are held on the second Tuesday of every month. The dates are March 10, April 14, May 12, June 9.

Hardware tutorials start at 6:45 p.m. and the actual meeting starts at 7:30-7:45 p.m. Meetings will take place at Sheridan College on Trafalgar Road in Oakville. Once in, go through the main entrance and across from the cafeteria is room B123, the meeting site. Don't drink the machine coffee.



DEFACTO:
DeFacto is now printed and ready to go. For new members who haven't heard, DeFacto is essentially a collection of reprints from the from the first three years of Ipso Facto. What makes DeFacto better than back issues (which are not available anyway) is that all known corrections have been incorporated. This makes DeFacto probably the best source of hardware and software articles around! DeFacto consists of over 600 pages, pre-punched for easy insertion in a standard three ring binder. Also, (for what it's worth) the collection comes with a pretty impressive three colour cover. See our back page for order form and price.

CONTESTS

The last page of the bulletin has been re-designed to facilitate:

a) voting for the best article in order of preference;

b) order form for P.C.B.'s and Defacto;

c) membership application form;

d) return mail by folding on dotted line, taping or stapling and adding appropriate postage.

Contest for a Club Monitor:
We would like a lot of suggestions as to what it should do.

Standard CPU Board for the A.C.E. Buss:
Again, we need many ideas and comments on the board structure.

EDITORIAL LISTING

Yes, Bob, Chuck and Ed, there really is an IPSO FACTO, and hopefully I will get the rest of this year's issues out on time.
I need articles large or small written for all levels of experience, so sit down right now and write that article you've always been meaning to write.
The ACE VDV board has been out now for some time; where are those graphics and text editor programs?
Does anyone have a VIP III to Ace buss interface, or any interface to Ace buss?

COLUMNS: I need material for the new columns we started in the last issue: Hints and Fixits - helpful hard- and software information
Nestings - small, useful programs in any language
HELP - in this column you can ask for or give HELP
Letters of Contact - list what you would like to communicate about
What's New for the 1802 - send information, product reports.

The votes have not been tabulated yet for the Best Article, the result will be in the next issue. Keep those votes and comments coming.
Where are all those hams, with their call letters and suggested net frequencies?

EXECUTIVE FILE

Hardware Sales: we have contracted with a new supplier for our P.C.B. and all outstanding boards will be manufactured and shipped by the first week of March, 1981. I thank all of you who have been waiting so patiently, I know what it's like to be kept waiting to get going on a project because the material isn't there. (I haven't been able to update my system yet. -F.P.) The prices have been raised due to increases in the cost of production and the ever-rising postal rates.
Ordering P.C.B.'s and DE FACTO: all orders must be accompanied by a certified cheque or money order. This will greatly speed deilvery and any orders not accompanied by one of the foregoing will be held up until the cheque has cleared, at which time shipment will be made. Thank You.
We have two disK controller boards in development by T. Crawford and Ken Bevis. Both use the 1771B controller chip and the same cable connections so hardware is virtually interchangable. Ken has a 3K DOS running with his prototype board which includes a working interface with Quest Basic Ver. 5. Both of these systems are for the Ace buss. F.P.

HELP

Eugene Shepard, 903 12th St., Huntsville, Texas 77340 would like a parallel input for Super Elf Tiny Basic V 1.1, a character generator for T.B.(similar to Netronics 4K T.B.) and output from the 1861. He has an RCA keyboard.

Cor Bouwhais, Bentrot-Straat 28, 7531 AB ENSCHEDE, the Netherlands wants to know if there is a PASCAL compiler for the 1802 in development. He also wants to know where to get a copy of PALO ALTO tiny basic.
In IPSO FACTO issue 16, page 55, Robert Edwards, 104 Montreal Lane, Oak Ridge, TN, USA 37830, offered a source listing for a PALO ALTO tiny Basic interpreter called TBI1802 for $2.00 (US).

E.L. Smothers, 5022 Judy Lynn, Memphis, Tenn., USA 38118
I have a homebrew Elf II with 17K RAM, I'm also using a Netronics VIP board and running Tiny Basic with no problems. I tried running Quest Super Basic and had trouble after making the patches listed for Elf II. Anyone who has Super Basic running on the Elf II please write and give me some HELP.

LETTER OF CONTACT: Cor Bouwhais, Bentrot-Straat 28, 7531 AB Enshede, The Netherlands
Cor would like to correspond with 1802 users in California.

Bert de Kat, Lyndon, Ontario, Canada, LOR 1TO
Bert would like to contact members who have an Explorer 85 system.
(I can't imagine why anyone would want anything but an 1802 - Ed.)

SELL OR SWAP

1. Paper tape reader, has motor takeup and drive electronics. $20.00
2. T.I. 2716 s 2KX8 EPROM's, require 3 voltages, used once. $5.00 ea
3. Dual 22 pin connectors, gold solder tail, $1.50 each.

- contact Gary Banko, 407 Southmoor Dr., Arlington, Texas 76010

HINTS AND FIXITS

Alfred Pascheiu, 2500 Geo. Washington Way #233, Richland, Washington 99352 - FIXIT for Netronics tape controller board jitters:
The controller board operated erratically and seemed to be undependable. I solved the problem by first removing the resistors and replacing them with jumpers. Then I moved the board farther away from my video monitor; apparently the monitor generated a strong magnetic field that kept the relays from operating properly.
FIXIT for Netronics Assembler - the assembler came in two parts, and the second part was supposed to prepare the assembled program into machine code. I could not get it to operate as described in the manual, and I discovered the problem was that the memory has to be partitioned so that the last two pages ahve the high address as follows: YXOO-YZFF, where X & Z could be anywhere in memory. I accomplished this by removing the two original 2101 chips. -A.P.

DANGER - ELECTRICITY CAN KILL YOU

"The most dangerous effect of electric shock is a derangement of heart action known as ventricular fibrillation. In the fibrillating condition, the pumping action of the heart stops, and death is likely in about five minutes. Treatment consists of prompt and continuous application of artificial respiration, preferably the mouth-to-mouth resuscitation method."

"If the rescuer has been trained, artificial respiration should be alternated with closed chest cardiac massage and the resuscitation process continued as the victim is taken to a hospital and given a defibrillating treatment. It is believed that currents of only 100 milliamperes are sufficient to cause the heart to fibrillate."

"Many persons have a fear of electric shock from instinct, hearsay, past experience, or education. Moveover, this fear is often surrounded by mystery and lack of knowledge. For those familiar with the technical aspects of electricity, perhaps part of this confusion might be dispelled by defining the limiting quantitative values controlling the hazards of electric shock."

a) Lethal Shock Hazard: Those a-c and d-c circuits capable of passing through a 500 chm resistor, an uninterrupted a-c current in excess of 100 milliamperes, an uninterrupted d-c current in excess of 500 milliamperes, or an inpluse discharge having an energy in excess of 50 joules.

b) Shock Hazard: Those a-c and d-c circuits capable of passing through a 500 chm resistor, an uninterrrupted a-c current in excess of 10 milliamperes, an uninterrrupted a-c curent in excess of 10 milliamperes, an uninterrupted d-c current in excess of 60 milliamperes, or an impluse discharge in excess of 1/4 joule.

c) Negligible Shock Hazard: Shocks of an intensity less than those producing shock hazard defined above, or equipment and circuits operating at 25 volts or less.

"As stated earlier, electrical safety is achieved by isolation, insulation, grounding, and current or shock limitation. Isolation means placing high voltage wires high overhead and far out of reach, or designing structures with ample clearance so that workmen can work or climb with safety. Unfortunately safety by isolation is often circumvented by kites with metalized strings, booms of cranes and derricks, and by storing objects in the work spaces in front of or behind switchborads, etc."

"Electricity is kept in its proper place by electric insulation. Adequate insulation assures that our elecrical machines, appliances, hand tools, and portable extension cords, etc., will permit safe operation with long and essentially trouble-free life. However, all mechanical contrivances fail eventually and the public must be ever alert to spot potentially dangerous conditions."

"Many insulations become brittle and develop small cracks when they become old or worn; on the other hand, abuse can damage any insulation. Fortunately, many pending failures are obvious, such as frayed insulation or extension cords, broken extension cords, broken receptacles, or broken attachment plugs. Water and moisture often bridge small defects in the insulation of nearly worn out devices and cause the metal frame of the device to become electrified."

SIMULATED WORD INSTRUCTIONS
FOR THE RCA 1802 MICROPROCESSOR

 Volker Raab, Ramtenvej 30
DK 8581 Nimtofte, Denmark

Like most other 8 bit microprocessors, the 1802 has only a limited instruction set. If you start out to write any bigger software system for your micro, you need more sophisticated instructions than those provided by your micro.

The common way to expand the instruction set of any computer is to add a suitable set of procedures. So, that's exactly what I've done.

Some special 1802 features make these procedures look rather like expanded instructions,though. Careful planning and a bit of tricky code allowed for more than 60 instructions using less than 512 bytes. There is even room for extensions and naturally, the code can run from a PROM. Furthermore, the instructions are compatible with the 1802 hardware instructions and the usual 1802 software conventions.

Naming conventions:
The first, or the first two letters of each instruction indicates the type of instruction; the next letters are mnemonic names and the last letter sometimes indicates special handling of the carrybit. The mnemonic should be rather obvious and easy to understand, but the type indicators need some further explanation:

u or x	are 3 byte instructions, the third byte contains an immediate operand.
v or y	are the corresponding 2 byte instructions, they expect a byte operand placed in the hardware stack, at location (R2).
a	the instruction effects register RA
b	the instruction effects register RB.
w	the instruction effects register RF.
x and y	are index register instructions, they change the hardware stack register; x by executing a SEP RF instruction, while computing the operand address.
z	are special instructions, effecting the X-register in different ways.

Note: The second type letter happens always to be a.

Use of instructions:
The word instructions are sequences of either 2 or 3 bytes, like:

D7 asign (=47) D7 reasign (=45) byte operand

After the SET R7 (=D7), the next instruction executed will be the LDA R3 placed at address R7INIT with P=7, (register R7 as program counter). This instruction will pick up (to D) the address of the word instruction (here: asign or reasign) and increase R3. The next instruction PLO R7 is actually an inside page jump to the address of the word instruction.

3 byte instructions start with another LDA R3, picking up their operand and increasing R3 further. All word instructions return to the instruction just in front of R7 init, thus resetting register R7. Their last instruction is the SEP R3 which restablishes R3 as the program counter and advances R7. The next instruction is then the one following the word instruction. Note, however: Just like the SCRT's, the word instructions require the use of R3 as the program counter.

Register utilization:
The word instructions expect a conventional use of the 16 1802 registers, meaning:

R0,R1,R2 — are used by the hardware, DMA, Initialization and Interrupt. Consequently, R0 and R1 are never used, while R2 is used in a way that is compatible with conventional usage. The byte addressed by R2 is used as a temporary or intermediate storage location and the R2-. stack is used for storage of return addresses.

R4,R5,R6 — and R3 and R2 are conventionally used by the standard call and return technique, SCRT. SCRT routines are included in this package, they are fully compatible with other SCRT routines,but slightly more elaborate.

In other words, utilization of register R0 to R6 is more or less predefined. Therefore, the word instructions use R7 as their dedicated program counter, similar to the way R4 and R5 are used in SCRT routines.

As the word instructions are implemented by software rather than hardware, they have to make a choice of the registers they do effect:

RA — is used as a 16 bit accumulator

RB — is used as a 16 bit base address for index computation

RF — is used as a 16 bit general work register

This may seem an awful waste of registers, but RA, RB and RF are not reserved, when no word instructions are used. On the other hand, if you want to do 16 bit arithmetics, you have to use a register as a 16 bit accumulator anyway and if you want to do index addressing, you do need both a base address and a computed address.
Note: R4,R5 (due to conventional SCRT) and R7 (due to the word instructions) must be initialized before useage; their initial values are R4INIT, R5INIT, R7INIT, extended with a high byte for the page of memory where they reside.

Installation:
The word instructions may be placed anywhere in memory. The listing shows them on pages 6C and 6D for one reason only, to identify all byte locations even in a binary listing only where correct page addresses should be inserted.

Selection criteria:
In general, word instructions have been included, if their calling sequence (2 or 3 bytes) corresponds to more than twice as many bytes of inline-code.

Stack handling:
Like most other 1802 instruction, the word instruction assume their operands either in RA (the equivalent to D) or at a stack (defined by the current value of X). Word instructions cannot use the R2 stack, as R2 would sometimes point to the high part and sometimes to the low part of a word. Any possible placement of words in the R2 stack could be damaged by an interrupt, occurring while pointing to the byte with the higher address. (It would be possible to change the interrupt routines, though).

After some experimentation, I did choose the most natural stack implementation, an upward growing stack with the high byte of a word preceding the low byte of the word. The word address is always the address of the high byte, and the stackpointer always points to the word just placed into the stack or to the word that could be taken off the stack. (That is the same word, the alternative always points to the next free stack location). (R2!!) This indicates the following use of the free RAM area, the word stack growing upwards from the lowest address, and the R2 stack growing downward from the highest address.

The stack pointer may be any register, preferably R8,R9,RC,RD,RE, (or RF and RB in some cases).

Index addressing:
The word instructions provide a limited form of index-addressing, using a 16 bit base address and an 8 bit offset (in the range 0-255). This form is useful for table lookup. The base address must be present in RB and the offset can be an immediate offset (x-type) or a computed offset (y-type). The immediate offset is just the third byte of the word instruction, the computed offset must be placed at (R2).

Index computation is done at R5IDX, base address and offset are added and the resulting index address is placed in RF.

Index computation is done with R5 as program counter. This allows index computation as a procedure from within other word instructions. As the SCRT-EXIT routine uses R5 as well, there might be a conflict. The solution is rather simple,but a bit tricky:

EXIT is SEP R5, INDEX is INC R5, SEP R5

Index computation has a side effect; it executes a SEX RF making RF the new current stack register. In other words, in some application it may be necessary to change the current stack register after use of an index-addressing word instruction.

Listing of word instructions:
All word instructions are listed together with corresponding byte (hardware) instructions or mnemonics of possible but non-existing byte instructions and sometimes an explanation.

WORD - INSTR,		BYTE-INSTR		EXPLANATION
AMIN	0	SDI	0	arithmetic complement
AMINB	2	SDBI	0	
XAADD	5	ADD	(X=F)	arithmetic sum
YAADD	7	ADD	(X=F)	
AADD	9	ADD		arithmetic sum
AADDC	0B	ADC		
XASUB	0E	SM	(X=F)	subtraction
YASUB	10	SM	(X=F)	
ASUB	12	SM		
ASUBB	14	SMB		
XAREV	17	SD	(X=F)	reverse subtraction
YAREV	19	SD	(X=F)	
AREV	1B	SD		
AREVB	1D	SDB		
ANOT	20	XRI	FF	logical complement
XAAND	23	AND	(X=F)	logical AND
YAAND	25	AND		
AAND	27	AND		

WORD - INSTR,		BYTE - INSTR		EXPLANATION
XAIOR	2A	OR	(X=F)	logical (inclusive) OR
YAIOR	2C	OR	(X=F)	
AIOR	2E	OR		
XAXOR	31	XOR	(X=F)	exclusive OR
YAXOR	33	XOR	(X=F)	
AXOR	35	XOR		
ALEFT	38	SHL		left shift
ADOUB	3A	SHLC		
ARIGHT	3D	SHR		right shift
AHALF	3F	SHRC		
ACOMP	42			D=0 if RA=0 (and only then)
UASIGN	45			extend low byte (sign extension)
ASIGN	47			
UABYTE	4A			extend low byte with zero
ABYTE	4C			
XAPUT	4F	STX	(X=F)	store RA
YAPUT	51	STX	(X=F)	
APUSH	54	„ASTX"		(advance AND) store RA
APUT	56	STX		store RA
BPUSH	5D	„ASTX"		(advance AND) store RB
BPUT	5F	STX		store RB
WPUSH	66	„ASTX"		(advance AND) store RF
WPUT	68	STX		store RF
XAGET	6F	LDX	(X=F)	load RA
YAGET	74	LDX	(X=F)	
AGET	73	LDX		
BGET	79	LDX		load RB
WGET	7F	LDX		load RF
BPULL	85	„LDXD"		load RB (and decrease X)
WPULL	8B	„LDXD"		load RF (and decrease X)
APULL	91	„LDXD"		load RA (and decrease X)
ZDEC	96	„DRX"		decrease X
WACALL	A3			call, RF points to address
WAJUMP	A5			jump, "
WCALL	AB			call, RF contains address
WJUMP	AD			jump, "
XADDR	B3		(X=F)	compute index address
YADDR	B5		(X=F)	
ZSAVE	B9	MARK		save X,7 in R2-stack, X gets 7!!
ZKEEP	BD	„MARK"		save X,7 in R2-stack and load X again
ZLOAD	BF	„RET"		load X from R2-stack without destroying
ZTAKE	C5	RET		load X from R2-stack and destroy R2-stack
ZHALT	CA	„SWI"		software interrupt: P=1, X=2
ASAVE	D0	STXD		save RA in R2-stack
ATAKE	D3	„ALDX"		take RA from R2-stack
BSAVE	D6	STXD		save RB in R2-stack
BTAKE	D9	„ALDX"		take RB from R2-stack
WSAVE	DC	STXD		save RF in R2-stack
WTAKE	DF	„ALDX"		take RF from R2-stack
BASWOP	E2			exchange RA and RB
WASWOP	E5			exchange RA and RF

Arithmetic and logical instructions:
Arithmetic and logical instructions are comparable and compatible to the corresponding byte (hardware) instructions. They use RA (instead of D) as their first operand and for the result, and if necessary, take their second operand from the stack as defined by X.

The resulting carrybit is the same as the resulting carrybit would be for an extended version of the corresponding byte instruction.

If the result is zero, and only then, D will be zero too. This group includes all instructions from AMIN to AHALF. Sign extension and index computation always destroy the carrybit, all other instructions: BYTE, PUSH, PUT, PULL, GET, DEC, CALL, JUMP, SAVE, TAKE, KEEP, LOAD, and SWOP do not affect the carrybit, neither do (R4) CALL or EXIT or HALT, but naturally subroutines may do so.

The X-Register and parameter passing:
Both 1802 and word instructions use the X-Register to define the second operand for some instructions. Unfortunately, the 1802 provides only inadequate means of manipulating the X-register. Furthermore, 1802 software conventions only support inline parameters or special agreements between program and subroutines. A general software system requires a general method of parameter passing and full control of the X-register, especially the possibility to save and restore the X-register.

The following 4 word instructions save and load the X-register from a 2 byte record saved at the R2-STACK.

ZSAVE	saves a record into the R2-stack with the current X-register and 7 as the current P-register. The instruction sets the X-register to 7.
ZKEEP	same as ZSAVE, but continues with ZLOAD and ZTAKE.
ZLOAD	copies the X-record at the R-2 stack and uses the copy to restore the X-register, leaving the original X-record at the R2-stack.
ZTAKE	takes the X-record off the R2-stack and restores the X-register.

These four instructions allow use of the X-register as a pointer to the parameters of a subroutine without knowing the actual value of the X-register.

A single parameter may be passed via RA, but in general, parameters should be passed via the X-stack. The subroutine may then use RA and RF as accumulator and general work register.

Both RA, RB and RF can be saved and restored at the R2-stack in order to make subroutines transparent to the program using them. As subroutines have no way of knowing the value of the X-register, they should always save other registers and restore them, if they need more registers themselves.

Call and return technique:
The word instructions use a version of the Standard Call and Return Technique that keeps the X-register intact. For convenience, two different forms of computed jumps and calls are added, they all use RF either as a pointer to the jump/call address or as the jump/call address and follow otherwise the same conventions as the standard call (SEP D4, address). Still, only one exit (SEP D5) is necessary. Note: Programs can only use computed calls and jumps, when RF and RB are not containing parameters. This is quite acceptable as far as RF is concerned, but may be a bit awkward if both program and subroutine work on a common table (placed in RB).

FULL BASIC FOR THE 1802 (REALLY!)

- By D. Shroyer - May 1980

As an 1802 user for about three years now, I have often thought wistfully of running a full Basic and all that it would allow on my home system. As advertisements for full Basic programs began to appear, I eagerly anticipated the delivery of the products. I hoped it would allow me to join the rest of the hobby computing world. As I worked with the full Basics, I realized that their respective strengths and weaknesses were very different. I also realized that the information available from the advertisements wasn't really sufficient to make an intelligent decision as to which Basic to choose. The purpose of this article is to present sufficient information to allow 1802 users to decide which of the two currently available versions is right for them. The products reviewed are Netronics Research and Development's Full Basic with RPN hardware math, Level #3 written by L. Sandlin and Quest Electronic's Super Basic Release 3.0 written by R. Cenker.

My System.

Since both products were tested on my own home system with its particular limitations, a word about it is in order. My 1802 system is the Elf II with Netronic's Video Display Board, Giant Board, a total of 22K of RAM from various makers, Quest's Super Expansion and Elf II Adaptor Board, a home built ASCII keyboard, and a portable TV monitor with RF modulator hookup. A home built bus buffer allows me to operate all of this with no known weaknesses in hardware. For mass storage I use a portable cassette tape recorder.

BASIC COMPARISON

Netronics' Full Basic with RPN Hardware Math, Level #3

Hardware Requirements

1. Memory: 8 K of RAM minimum. (6 K plus the last ¼ K for the interpreter and 1¾ K for user programs)

2. Input/output: parallel via ports 7 or serial RS-232 C (both are included on the Giant Board)

3. Terminal: Netronic's Video Board and ASCII encoded keyboard or equivalent

4. Display: video monitor, converted TV or RF modulator, or printer, etc.

5. Cassette: tape read via EF2 and write circuit via Q (also included on the Giant Board)

6. Math: MM57109 interface with input port at N = 2 and output port at N = 5 (a 86 pin buss board is included with the BASIC package).

The Math/ROM Board

Since this item probably accounts most for the difference in price between the Basic's discussed, I will deal with it first. This circuit board, which plugs conveniently into an Elf II bus connector, is responsible for the Full Basic's math capability. It is high quality epoxy glass with plated through holes. The board includes all parts required for the hardware math interface. The board also includes an area for 8 K of ROM/EPROM and the associated decoding, addressed at 0000 - 1FFF. The format for the firmware accommodates both the 2317 and 2716 ICs. It is therefore possible to have the Basic reside in ROM or EPROM.

A modification to the Giant Board is necessary since both the hardware math interface and the Giant Board cassette read circuits use EF2. The modification will clear up any conflict in the multiple use of EF2. If you have difficulty reading cassettes after the mod, try substituting a germanium diode.

Advantages and Disadvantages

Netronics' Basic requires a mere 6½ K of memory, definitely an advantage in small systems. The Basic's other principal advantages are the math in hardware, with a tremendous memory saving and the fact that variable names are not limited to A to Z single characters. A variable name can be anything using alphanumerics but not space or punctuation.

The Basic's principal disadvantages are the lack of dimensional variables, (arrays) very limited string handling, the fact that strings cannot contain spaces or punctuation for the most part (see the manual additions and corrections below), the fact that null strings are not allowed, and the very slow execution time of programs written in this Basic.

The RPN (reverse polish notation) logic used may be considered by some 1802 users to be a disadvantage. My opinion is that in the long run RPN is more powerful, less complicated, and more efficient to use than algebraic logic. For those unfamiliar with it, it does take some getting use to, especially the version utilized by this Basic.

Most existing Basic programs will require conversion of the math functions to RPN before being usable.

The power of RPN comes in where a complex problem can be written and solved in terms of the same "stack oriented" method actually used by the computer. Such a problem is less complicated because parentheses are never used to artificially group the computations which must be done first. In RPN, the problem is solved in the order in which it is presented.

Listing 1. gives examples of the difference between RPN and algebraic logic.

Some of the other interesting characteristics of this Basic are discussed now. The Basic features automatic baud rate detection. The range of baud rates is not specified in the manual. Even if it were, I could not test it since my VID is hardwired to 300 baud. At 300 baud, the baud rate detection feature performed flawlessly. Automatic terminal line length limiting and selection is also featured. This means that at power up you can select either 32 or 64 character width and Basic configures output accordingly. As the end of the line width is approached, Basic decides whether or not to continue printing or to insert a CR and LF to begin a new line. Unfortunately, there seems to be a bug here. See below in the section on manual additions and corrections. Character editing is also included using the delete key, however, no code is provided to cancel a line.

Printouts can include control characters such as VT (verticle tab), HT (horizontal tab), LF (line feed) BS (back space) and HOM (home cursor). With a display such as Netronic's Video Board with its special characters, erase, cursor control moves, and cursor addressing sequence, all of these can be done using the Basic's CTL statement. This was not discussed in the manual but is set forth below in the manual additions and corrections section.

Program control of I/O is possible using the DEVICE and SET statements. This means that the user program can select the nature of communication with Basic: parallel to parallel, serial to parallel, parallel to serial, or serial to serial, selecting desired baud rates where appropriate.

LOAD and SAVE in this Basic is independent of the monitor, that is, once Basic is running, user programs can be loaded from cassette or saved on cassette using Basic's built in routines rather than the Giant Board's ROM monitor. A new feature of Basic's cassette routines is that the size of the dump is part of the tape data so that Basic will not permit loading a program too large for memory available in user space.

The Basic has machine language capability using USR, POKE, and PEEK statements. I feel that PEEK should have been kept as a function rather than a statement but the resulting limitations are not substantial. At most, two or three extra program lines may be needed in some cases.

Basic has verticle as well as horizontal TAB capability. This is a good feature but not without limitations because of incomplete screen coverage. There seems to be a bug here in that a CR and LF may be injected unexpectedly, defeating the purpose of a TAB statement to define where a printout should be placed. See the manual additions and corrections section and Table 3. TAB arguments below.

Since Basic allows great flexibility in defining variable names, an EXAM statement was included to printout the variables which have been defined. This is very useful in debugging user programs. If more than 15 or so variables are printed out, however, some are lost off the top of the screen. There is no way to freeze it at any point except perhaps by putting the 1802 into the Pause mode (CLEAR high and WAIT low). That may result in garbage on the screen when the 1802 is again allowed to run. EXCL is a statement which allows you to eliminate a defined variable once it is no longer needed. You can EXCL only one variable at a time though.

HELP is a cute statement which displays all statements available in Basic. It doesn't show the match functions available however. I feel this capability is "cute" because though surprising, its usefulness fades as you become familiar with the statements available.

RELIST is a helpful statement which lists whatever was last listed. So if you alter a set of program lines in debugging a program, rather than LIST with arguments, use RELIST to look the corrected lines over.

INPORT and OUTPORT statements allow you to input or output data using the 1802's N line decoder control feature. Once again I feel that INPORT would be more versatile as a funtion but the point is a small one.

The Basic's hardware math capabilities deserve independent discussion. This capability is very similar to that available in a hand held scientific calculator. Floating point math to eight digit accuracy or scientific notation with an exponential range of 10 to $\pm$ 99 is user selectable. The mantissa of the scientific notation has a five significant digit capability. Trigonometric and inverse trigonometric functions, memory, inverse memory, square, common and natural log, roots, pi (3.1415927), e^X, Y^X, degrees, and radians are all available.

Some of the MM57109's capabilities built into Basic were left out of the manual. See the manual additions and corrections section. The speed of the math function while not overwhelming, is pretty good, and may be increased by decreasing the size of C1 on the math board. Math functions are also available at the machine language level if you know how to implement them. The National Semiconductor's spec sheets for the MM57109 Number Oriented Micro processor will assist interested persons to do this.

Some of Basic's small but annoying faults are now discussed. Using an INPUT statement, you are limited to a ten character maximum for each variable. You also can input only one variable at a time, typing CR after each. TAB may erase along its horizontal path which makes creation of table alphanumerics and graphics displays more difficult. Basic is very "space sensitive" in that spaces are a required part of statement/command and function syntax. Some very strange results can occur if required spaces are not used. The manual does not indicated this for every command. See the manual additions and corrections section below.

Operators are inflexible, and <>, = < and ≤ >, do not always work. See the manual additions and corrections section following. The break capability of Basic does not say where the break occurred, if it did, debugging would be easier. GOSUB can be nested but the nesting is limited to five deep. There is only a very limited multi-statement program line capability. LET can assign as many variables as the 72 character line buffer will allow in a single line provided a math operation using "#" (enter) is not involved. The result of an IF operation can be another statement. Outside of that, every statement needs its own line number.

Two not so small faults are: first, there is no string handling except for comparison for equality, and second, Basic's FOR NEXT loops. STEP can't be negative, the loops run slower than a common loop, there are limitations in the extent of the arguments, each loop counter must be a different variable, the deeper the loop is in a program the slower it runs, and there seems to be a bug in that a READ statement as part of the loop will read only one DATA statement.

Other things which I would like to have had in full Basic, but which Netronic's Basic lacks, are hex number capability, logic operators (AND, OR, EXCLUSIVE OR, and NOT), and a tape load append capability. The latter would allow creation of a new user program by loading subroutines of different tapes together as long as there was not line number conflict.

Manual Additions and Corrections

line numbers

Don't use 13 as a line number. It will foul up GOTOs, GOSUBs, and LIST.
p.4

constants

Spaces and commas are not allowed in string constants but HTs (horizontal tab) and periods are accepted.

variables

Before a variable is used by a program, its value must be assigned (the assumption of 0 should not be made). HTs and periods are allowed in variable names.
p.5

overflow and division by zero

A complete list of the error/overflow conditions of the MM57109 is given in the spec sheets as: (adapted to the Basic format)

1) (x)#LNX where (x)≤=0
2) (x)#BTX " " "
3) (x)#TAN where (x)=90^0, 270^0, 450^0, etc.
4) (x)#SIN where |(x)|≥=90^0
5) (x)#COS " " " "
6) (x)#TAN " " " "

7) (x)#SQRT where (x)<0
8) /, INV /, or IX where division by 0 results
9) (x)#INV SIN where |(x)|>1 or |(x)|<= 10^{-50}
10) (x)#INV COS " " " " " " "
11) any scientfic mode result <10^{-99} or >=10^{100}
12) any floating point mode result where the number of mantissa digits to the left of the decimal point is> 8 or the number of 0's to the right of the decimal point> 6.

relational operators

<> doesn't always work, use ><. Don't use =< or =>.
p.7

format notation - 3

Spaces between statements and arguments must be included where shown.

ABS

Use a space between ABS and its argument, eg. ABS (variable)
p.8

CLS

The format: PRINT CLS also allowed.

CTL (this was entirely omitted from my manual)

Format: CTL (character)
CTL (characterlist...)
PRINT CTL(character)
PRINT CTL(characterlist...)
Note where there are and are not spaces!

Purpose: To print the control/shift ASCII code of the characters. The action is to subtract 40 hex from the ASCII code of each character.

Remarks: If the PRINT format is used, CTL... must be the end of the program line. With the Netronic's VID or equivalent most of the downshift characters, using CTL P..., cursor moves, escape sequence, using CTL [..., and erases (CTL E and CTL F) can be outputted. See the VID manual.

Example:
```
10 CTL POPK
20 END
RUN
Ω Σ
```
(symbols for ohms and summation)
p.9

END

DATA and FOR NEXT cannot be beyond an END.

EXAM

Format PRINT EXAM also allowed.
p.10

FOR NEXT

The loop continues until the final value Y is exceeded (not just equalled). STEP value can't be negative. Each loop must use a different variable for the loop counter.
p.11

HELP

Format PRINT HELP also allowed.

INPUT

While there is no defined exit from an input prompt, typing ",," will produce an error stop.

IF THEN

Format: IF (variable) <= >(variable) THEN (line number)
IF (variable) <= > (variable) (command/statement)

Remarks: THEN makes the IF a conditional branch statement. If another command/statement is needed, don't use THEN.
p.12

LET

As long as "#" isn't needed, any number of variables up to the 72 character input line buffer limt can be assigned. A calculation using "#" can be the last such assignment, however,

Examples: 10 LET A=1,B=2,C=3...,X=100
20 LET A=1,B=2...,X=44#C+

LOAD

If there is already a program in user memory, NEW may be necessary before LOAD will work.
p.13

PRINT OR PR

A space is needed after the statement unless " is used. Any other statements which can be used with PRINT can be combined in one line.

Examples: 10 PR A;B,C
20 PR"HELLO"
30 PR EXAM;CLS;BYTE;HELP;CTLY
p.16

SET

SET Q prints out the measured baud rate code without a CR or LF.

STOP

FOR NEXT can also be located after STOP.

TAB

Format PR TAB(argument),(argument); is also allowed but will not calculate arguments. The cursor will erase along its horizontal path. (see also error code 5469 and arguments in Table 2.).
p.17

USR

All three arguments may be needed for a proper return via a D5. Use ~~0~~ if otherwise unneeded.
P.3 - 2

EE

Either EE or E is valid.

INV

INV can also be used with +, =, *, /. Used this way the math processor registers X, Y, Z, and T are unchanged, previous M is lost.

Example: PR 13#INV + (13 is added to memory register M)

LOG

Format for common (base ten) logarithm is BTX (not LOG).

Appendix - error codes

5469 TAB value for horizontal argument must be < =23 for 32 line width, < = 54 for 64 line width. See also the Table 2. arguments.

5676 INPORT 1 is missing the * required.

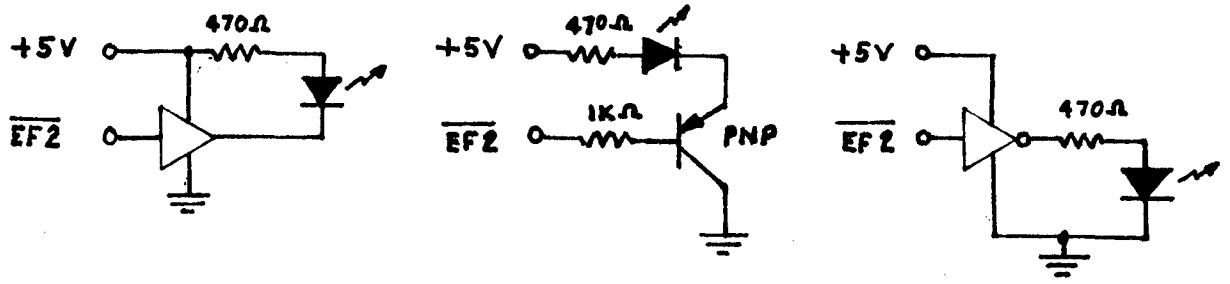
4596 Cassette tape read error.

Operating Notes.

I have found the following useful whle working with this Basic. The terminal line length limiting feature may issue a LF and CR while typing. To assure the program line will execute properly, press delete once, then retype the last character and continue. The line length limiter only operates toward the end of the line width and only when a non-alphabetic, non-numeric character is typed.

As the 72 character input line buffer limit is approached, a series of underscores will indicate the spaces remaining. Press delete twice. The printout of underscores will repeat. Be sure to finish the line with at least two underscores remaining or the entire line will be rejected.

Before attempting to LOAD a program from cassette, be sure EF2 is not grounded by the math interface circuit. See Figure 1. for simple circuits which will give a visual indication of this.



An alternative is to type the following:

PR#

This will clear the EF2 line for use by the cassette circuit.

The last page (256 bytes) of available memory is used by Basic for stack storage. Don't POKE around there unless you know what you are doing.

One of the byproducts of the RPN format math is the CS function/operator. CS (change sign) is used to negate a number or quantity as opposed to - which is subtraction.

Use with other systems.

Netronic's Basic was on course designed for use with the Elf II. An inquiry to Netronics about use with other sytems would only get the reply that the Basic is "hardware compatible" with other systems. If you have a system which makes different port assignments or EF line uses, you would be on your own in adapting basic to your system as it is not covered in the manual.

Since Netronics Basic is in straight machine code, it is relatively easy, although time consuming to disassemble it.

Quest's Super Basic Release 3.0.

Hardware requirements.

You will need the following to run this Basic:

1. Memory: 14 K of RAM minimum (12¼ K plus the last ¼ K for the Basic and 1½ K for user programs)

2. Input/output: parallel line port 5/3 or serial via RS-232 C (both are available on the Super Expansion Board)

3. Terminal: ASCII encoded keyboard and video display

4. Display: video monitor, converted TV or RF modulator, or printer, etc.

5. Cassette: tape read via EF3 and write circuit via Q.

Advantages and disadvantages.

This Basic's principal advantages are: first, it allows multistatement program lines, second, it has string variables and several string handling functions and operators, third, it allows one and two dimensional numeric variables and one dimensional string variables, fourth, interrupt handling capability on the Basic level, fifth, fast execution considering the 1802's SCRT limitation, and sixth, user programs are compressed and coded for insertion in memory. The last feature saves memory and includes a decode operation for LIST so that the coding is invisiable to the user.

The principal disadvantages are that 12½ K is needed for the Basic interpreter due in some part to the math functions being located all in software, the limited math functions available, and the usual A to Z single character names for variables.

Algebraic logic is use for this Basic's math package. This means it takes no getting used to but is efficient only for relatively simple math problems.

This Basic has quite a few other interesting capabilities. It too has automatic baud rate detection. Line editing is provided using the BS (back space) character. CAN (cancel) is the line cancel code provided. Printouts can have VT, HT, LF, HOM directly and BS, and DEL using the CHR$ function. With a display such as Netronic's VID, CHR$ can be used to output all of the special characters, erases, cursor control moves, and cursor addressing sequence.

PLOAD, PSAVE, DLOAD, and DSAVE is monitor independent. Note also the ability to load or save a program or program data independently of the other. There is a memory protect feature to prevent loading a program or data too large for available memory.

This Basic has machine language capability using the CALL and POKE statements and the PEEK and USR functions. If desired DEFUS is available to clear memory space for machine code programs. Also, machine code programs are saved or loaded along with the associated Basic program using PSAVE and PLOAD.

TAB in this Basic has only the horizontal argument. A verticle TAB can be effected using PRINT and LF or VT but to do so is rather clumsy. In a complex program, it is difficult to keep track of the cursor position. Of course with a VID, you can use the relative or absolute escape sequence via CHR$. This Basic's TAB also erases along its horizontal path.

Several string functions are provided. LEN returns with the number of characters in a string. ASC returns with the decimal equivalent of ASCII value of the first character of a string. MID$ extracts portions of strings. Strings may be tested for equality and substrings can be "added" together to form new strings. There are no limitations in string content.

This Basic has three statements to assist with keeping track of how much memory remains in user space. MEM prints the number of bytes left. EOP prints out the hex address where the current user program ends. EOD prints out the end of memory used to hold program data in hex.

The INP function and the OUT statement allow input and output using the N line controlled ports of the 1802. In this Basic, INPUT allows input of up to six or so numerical variables at a time. String variables must be input one at a time since strings can contain spaces, punctuation, etc. and one does not want to have to type " for a string. You can also input null strings. This is useful where a program is only waiting for a signal to continue rather than for a value or variable. A message can be included with an input prompt.

The GOSUB and FOR NEXT nesting is limited only by available memory. Also of interest is the EXIT statement which allows premature exit from a GOSUB or FOR NEXT, however, Basic expects to be at the next higher nested level after EXIT.

As mentioned before, Quest's Basic handles interrupts. ENINT enables interrupts, DISINT disables them. Multi-level interrupt handling is possible by re-enabling interrupts within each interrupt routine level. One application for this capability is program debugging. A properly written interrupt routine can diagnose program problems at any time the INT line is brought low. Such a routine could, for example, print all program variables or other program parameters. Manipulation of such could also be provided as appropriate. Another program debugging aid is the TRACE statement. When operating, this results in a printout of each currently executing line number.

Other interesting features are SGN, which returns with the algebraic sign of an expression, CLD which clears accumulated program data, WAIT which is used to create variable program delays, hex memory address capability using @, and hex number capability using #. Basic's input line buffer is 95 characters long and an overflow does not reject the whole line. Instead, only the overflowing characters are rejected and BS's outputted. The user then can edit back to a place in the line where it can be conveniently terminated.

A BREAK will print the line number being executed when the break occurred. This is useful for diagnosing endless loops in the user program. Break and interrupt are both limited in that they can occur only at the end of a

program line which can contain many statements. Also, the user program cannot be resumed after a break or error stop at the point of the break or error. The interrupt capability compensates for the former.

An interesting feature of this Basic is that it tests on insertion for missing " in all statements where quotes are needed. This is the only error tested during program entry. It is unfortunate that missing (or) are not tested also in all functions and statements where parentheses are required.

The math capability is available in either floating point or integer mode. The advantages of floating point are greater number range and program writing simplicity. The advantages of interger are absolute accuracy and increased speed. The normal mode on power up is floating point. The mode of all functions except RND(expr) is floating point. All numbers are floating point even though entered as an integer (no decimal point). The only way to get into the integer mode in release 3.0 is to use DEFINT. Even then, integer mode expressions are converted if necessary to floating point to avoid the mixed mode error problem of prior releases of this Basic. In integer mode, floating point mode expressions are converted to avoid the same problem. The only cause for concern remaining in mixed mode situations is the RND function. Even then no problems will arise so long as the proper form of RND for each situation is used: RND without argument for floating point, RND(expr) for integer. The most obvious case where integer would be desireable is, for instance, a FOR NEXT counter where the program is long and complicated and speed must be maximized.

The integer range is $\pm$ 214748367 whereas the floating point range is $\pm$ 17 x $10^{\pm 38}$. Floating point numbers have 6 significant digits. Whether floating point numbers are printed in scientific notation or not is determined by the value of the number (rather than by the mode as in Netronic's Basic). The same is true for integer numbers, but remember that integer numbers can only be whole numbers. The only space sensitivity in Quest's Basic is with numbers. Whether floating point, scientific notation, or integer, there can be no spaces within a given number. Also, in scientific notation, the decimal point is always printed first, as opposed to standard notation where the decimal point goes after the first digit.

The scientific function capability of Basic is adequate except for dedicated math applications. The speed of Quests' math calculation is just a little slower than Netronic's Basic. Note that trig functions in this Basic use radian arguments. This improves its speed, but is a little clumsy to work with if you are not used to radians, however, Pi is included to ease the problem. See manual additions and corrections section below.

As discussed in the Netronic's section, I would like to see Basic have an append cassette tape capability and logic operators. Quest's Basic does not have these either.

Manual additions and corrections.

INT The INT function always rounds down.

INUM INUM on the other hand rounds to the nearest integer, eg. = .5 rounds up.

INPUT Typing ",," allows an undefined exit from input (numerical variables only).

Operating notes.

You may find the following information useful in working with this Basic: first, don't use PR, the abbreviated PRINT statement, contiguous with the INT function (spaces do not help). Second, I have found that Quest's cassette circuits are more reliable with an Elf II running Quest's Basic. Both tape read write circuits can be used as required by switching between the two "Q" outputs and EF inputs. Third, the following conversion factors can be used when working with the trig functions:

```
degrees = radians * 180/PI
radians = degrees * PI/180
```

For those using the built in serial I/O routines but do not need the delay after a LF, change address 2D58 hex from 3A hex to 30 hex.

Use with other systems.

While Quest's Basic was designed to run on the Super Elf, all changes necessary to run it on other 1802 systems are supplied by Quest. This includes the different external flag and N line port assignments. Special instructions are provided for Elf II users and owners of Netronic's Video Display Board/Terminal. Quest will also supply a cassette read circuit schematic and a short cassette read routine using the Quest format so that owners of other 1802 systems can load the program cassette which uses Quests own system monitor.

Basic Performance Comparison

Execution Times

This section is included to give some support to my statements concerning the speed of execution of the two Basics. Several statements were selected for timing in order to give some idea of the difference in running times of similar programs using each Basic. To this end, statements were chosen which I feel account for a significant portion of total user program running times: FOR NEXT, GOSUB, LET. Also tested was an ordinary program loop (defined below) and a call to and return from a machine language level routine.

The times given are probably accurate to the nearest tenth of a second but the procedures used were designed only to test the two Basics against each other and may not be valid in terms of other microprocessor Basic benchmarks. In all cases the tests described were completed ten or more times and the measured execution times averaged to obtain the values given in Listings 1 through 9. The times record from when CR was hit after typing RUN to when the command prompt reappeared.

FOR NEXT was chosen because it is the most used program loop. The programs used to test FOR NEXT are given in Listing 1. Note that two different versions were done in Quest's basic. This is because of the difference in running times between integer and floating point math modes. The Listing's programs do nothing but loop so only looping time is measured. Since a single loop would be very difficult to measure, the loops were set to repeat 50 times. To ascertain whether the location of a loop in the program made any difference in execution time, the same tests were done with the loops located at the end of a 1600 odd byte program. See Listing 2.

A test on "ordinary loops" was also done. This was done to test whether a FOR NEXT loop was faster than an ordinary loop. An ordinary loop is defined simply as setting a counter variable, incrementing it, and testing to see if a certain value has been reached. If not the variable is incremented and tested again. If so the loop is complete. See Listing 3 for the actual programs used. Again two were used for Quest's Basic because of the two math modes. As with the FOR NEXT loops, they were set to repeat 50 times. The same 1600 odd byte programs were used to find out if depth in the program made any difference. See Listing 4.

GOSUB RETURN was tested the same ways as discussed above. See Listing 5 and Listing 6.

LET, and the abbreviated form with just the = sign, were tested since assignment of values is also and integral part of all programs. Only one listing of times is given since location within a program produced no significant differences. In Quest's Basic, the integer / floating point mode alternatives caused no differences either. See Listing 7.

Since both Basics include a machine language level capability, I decided to see how much time it took to call and return from such a routine. See Listing 8 for the programs used. Note that Netronic's Basic USR statement is the equivalent of Quest's Basic CALL. Since the arguments made a difference in execution times, this is shown. The machine code routines did nothing but return via the D5 instruction.

The GO SUB, LET and machine code calls were all done 50 times to ease the time measurement. This was done by placing each given program within a FOR NEXT loop. The execution times given are total execution time minus what the respective FOR NEXT loops required themselves.

Math was also tested for execution time. Because of the differing math capabilities between the two Basics, it was difficult to decide what to include. My decision was that since only the time required was of interest here and that the ability to directly compare the two Basics' times was desired, only those capabilities common to both Basics were tested. See Listing 9 for the problems used. Since math took more time in both Basics than the loops tested, each operation was executed only once, and the time recorded. Since the mode of a math calculation in Quest's Basic will almost always be floating point, no attempt was made to create or test an integer mode math problem set.

	Netronics	Quest (floating point)	(integer)
Listing 1			
	10 FOR A=0 TO 50 20 NEXT A 30 END	10 FOR A=0 TO 50 20 NEXT 30 END	5 DEFINT A 10 FOR A=0 TO 50 20 NEXT
Time (Sec)	17.8	1.7	0.9
Listing 2			
	1000 FOR A=0 TO 50 10010 NEXT A 10020 END	10010 FOR A=0 TO 50 10020 NEXT 10030 END	10005 DEFINT A 10010 FOR A=0 TO 50 10020 NEXT
Time (Sec)	69.0	1.6	0.9 sec
Listing 3			
	10 LET A=0 20 LET A=A# 1+ 30 IF A< =50 GO TO 20 40 END	10 LET A=0 20 LET A=A+1 30 IF A< = 50 GOTO 20 40 END	5 DEFINT A 10 LET A=0 20 LET A=A+1 30 IF A< =50 GOTO20 40 END
Time (Sec)	13.5	3.0	1.6
Listing 4			
	10000 LET A=0 10010 LET A=A# 1+ 10020 IF A <=50 GO TO 10010 10030 END	10010 LET A=0 10020 LET A=A+1 10030 IF A <=50 GOTO 10020 10040 END	10005 DEFINT A 10010 LET A=0 10020 LET A=A+1 10030 IF A <=50 GOTO 10020
Time (Sec)	29.1	4.3	3.0
Listing 5			
	20 GOSUB 1000 ... 1000 RETURN	20 GOSUB 1000 ... 1000 RETURN	5 DEFINT Z ... 20 GOSUB 1000 ...
Time (Sec)	2.4	0.5	0.5

	Netronics	Quest (floating point)	(integer)
Listing 6			
	10010 GO SUB 11000 ... 11000 RETURN	10020 GOSUB 11000 ... 11000 RETURN	10005 DEFINT Z ... 10020 GOSUB 11000 ... 11000 RETURN
Time (Sec)	18.1	1.9	1.9
Listing 7			
	20 LET N=1	20 LET N=1	20 N=1
	20 LET N=1.2345678E50	20 LET N=1.23456E25	
Time (Sec)	1.0	0.5	0.5
	20 N=1.2345678E50	20 N=1.23456E25	
Time (Sec)	1.6	0.2	0.2
Listing 8			
	20 USR 0,0,0 M(00 00)=D5 20 USR 57344,57344,57344 M(E0 00)=D5	20 CALL (0,0,0) M(00 00)=D5 20 CALL (57344,57344,57344) M(E0 00)=D5	
Time (Sec)	6.1/11.6	0.8/1.5	

Listing 9

Problem # and Logic	The Calculation Program		Answers, format used: Netronics Time (Sec)		Quest Time (Sec)	
1. RPN	22.574 SQRT 37 * 29.6 - 4.38 / LNX 12 + SQRE		237.47196	1.6		
algebraic	(LOG ((SQR (22.574) * 37 - 29.6) / 4.83) + 12) 2				237.47	2.0
2. RPN	2.2574E1 SQRT 3.7E1 * 2.96E1 \- .483E1 / LNX 1.2E1 + SQRE		2.3747E02	1.7		
algebraic	(LOG ((SQR (2.2574E1) * 3.7E1 - 2.96E1) / .483E1) + 1.2E1) 2				237.47	2.3
3. RPN	1.2345678 CS E13 CS # 9.8765432 CS E13 CS +		1.1111-E12-	0.6		
algebraic	-1.2345678E-13 + (-9.8765432E-13)				-.111111E-11	1.6
4. RPN	0.001 COS	in degrees	1.			
algebraic	COS(.0000174)	in radians		1.0	1.	0.9
	COS(.174E-04)	scientific notation radians			1.	1.0
5. RPN	89.99 COS	in degrees	0.0001745	1.5		
algebraic	COS(1.5706)	in radians			.0001962	0.9
	COS(89.99*PI/180)	degrees converted to radians			.0001758	1.1

Trig functions were included in the tests because these quite often tell the tale of how fast a math package is. Note, that in Quest's Basic, radians rather than degrees are used. Since many users will be more familiar with degrees, extra math tests were done to indicate what extra time might be needed to do the conversion. See Listing 9.

Note that the execution times given for the math problems are from when CR was hit after typing in the problem to when the complete answer was printed out. Listing 9. also gives the answers to the problems arrived at by each Basic.

Company Comments

Netronics states that an improved version of their Basic is in the works. It will include dimensional variables and some limited string handling. No specifics or a definite release date are yet available. (Ed note: as of Jan 81.) I have been assured that owners of the original version can obtain the improved one for a service charge as yet to be fixed. Netronics also tells me that an improved manual will be available soon but I have not seen it. (Ed note: corrections were mailed to early buyers.) Finally, the bugs I have mentioned are being checked by Netronics. As of the date I submitted this article, no patches or corrections have been suggested to me. (Ed note: some minor corrections have been mailed by Netronics.)

Quest states that version 3.0 of their Basic will be offered to owners of earlier releases for a service charge of $5.00. The one bug I did find in this Basic, in the INT function where the argument was from 0 to 0.5, was promptly corrected by Ron Cenker. I am told that Quest's Basic will work with baud rates up to 3000 baud. I was not able to test this however as I am limited to 300 baud as used by Netronics' VID board. (Ed note: Version 5.0 became available in Nov. 1980. Several corrections and program enhancements, including DISK and printer drivers are included.

Summary

Well there you have it! All the information any 1802 user should require to make an informed and intelligent decision as to which full Basic will best meet their system and needs.

If I may be permitted the luxury of expressing my opinion and general impressions of these programs, consider the following: Quest's Basic is so close to a standard Basic that I was able to type in many programs with very little or no changes. The math package is adequate. The speed of execution, while still not as good as others, for example 6502 Microsoft, is fairly fast. With integer loop counters speed has been considerably improved. The program compression feature is easy on memory space and typing in abbreviated statement lines still produced full program lines. This Basic also had some features seldom seen in standard Basic's, such as interrupt handling. In short, it is an adequate and useful product. I must point out however that Quest's Basic uses a lot of memory just to hold the interpreter, memory which many 1802 systems simply do not, as yet, have.

Netronics' Basic takes a different tack. In a lot less memory, the math package is developed to the capabilities of a scientific calculator. If you plan to do a lot of heavy math applications this cannot be ignored. What is most disappointing is the sacrifice of features almost universally thought of as implicit in a full Basic: dimensional variables and string handling. What is down right annoying is the very slow execution time. An investment analysis program I tried in this Basic took nearly one hour to give me accrued interest on the balance outstanding for a loan with a repayment period of 10 years. Dimensional variables can be faked by POKEing into the user program, the string weakness is partially made up for by unlimited alphanumeric variable naming, but a slow program is a slow program. Most of the Basic's speed problems can be attributed to the FOR NEXT loops' mode of execution. If this could be changed, DIM and string handling added, Netronics' Basic could do a lot more than turn your 1802 into a glorified hand calculator. More would still be required before it would be comparable to Quest's Basic, at least to my mind. It goes without saying that many changes were required before I could run programs in this Basic. Some programs just could not be made to run at all. The problem there was not just DIM or strings. There were problems with limitations on statement arguments and the fact that functions in other Basics are statements here. Also, this Basic does not allow multiple statement program lines. Often, there were simply not enough line numbers available to accomplish a satisfactory conversion. Once again I must point out the memory consideration. Netronic's Basic does quite a bit for the memory it uses. For those with a number of different output capabilities, Netronics' DEVICE and SET statements are very useful. This and other features make Netronics' Basic a substantial improvement over Tiny Basic but, in my opinion, do not make it a truely "full Basic"; almost, but not quite.

Conclusion

Both products, companies, and authors must be given due credit for helping to advance the 1802 as a viable alternative to higher priced systems. Both programs have their advantages and disadvantages. Neither is without room for improvement. The individual capabilities are quite different and should be carefully examined before making a choice. Of course you could use both as I have but my expereince is that this is extravagant and even wasteful as one or the other will likely be ignored as a favorite emerges. In short, decide before you spend your money, not after.

Post Script

Netronics Full Basic owners may be interested in the various patches and improvements I and others have come up with to fix various problems with the basic, such as the slow math board, line limiting bug, etc. Send $5.00 to cover shipping, handling, and copying costs or your own patch to get a list of all tested so far. Correspondence should be sent to the author's address (Don Shroyer, 209 Brinker Street, Latrobe, PA, USA, 15650), and include a self-addressed and stamped envelope.

THOUGHTS ON A SIMPLE FLOPPY DISK I/O SYSTEM
(with cassette tape options)

Wayne Bowdish
149 EAST 33
Hamilton,Ont
L8V 3T5

Floppy disk drives are now being acquired by many club members. This new hardware again presents the problem of standards. There are two main areas of concern, hardware compatability and software compatability. In this article I will propose some hardware standards for 8 inch soft sectored disks and some standard software routines.

It should be noted that the software standards, really a set of calling conventions for some subroutines, can be applied to cassette tape I/O as well as disk I/O. The reasons for implimenting the software for cassette based systems are:

1. the user is provided with 'blocked' I/O
2. software using the standard can be easily upgraded to run on disk based systems
3. software exchanges among members is faciiitated by providing a common set of I/O primitives
4. people having both cassette tape and floppy disks have a common set of I/O routines allowing programs to be run on either tape or cassette

To use the software with cassette tapes the drives must be equiped with some form of computer controlled tape start/stop capability. Rewind functions can be done manually. Another desirable feature, but not a requirement, would be to have two tape drives, one for READ operations and one for WRITE operations. This facilitates tape copy operations and large input/process/output type operations.

HARDWARE CONSIDERATIONS

An 8 inch soft sectored disk consists of 77 tracks. Each track is further divided into sectors. The number of sectors per track is dependent on the way the disk was formatted (ie. various sector header and trailer information was written onto the track). The disk controller (usually based on a disk controller chip such as the 1771) must be set up to handle the format of the disk.

A pseudo standard for sectoring exists which was developed by IBM. This standard divides each track into 26 sectors, each of which contains 128 data bytes. This standard is used by many vendors including IBM and DEC. Most CP/M* type systems also conform to this standard.

Since 256 bytes seems like a more reasonable size (ie. exactly one page) the proposed standard software will always read or write two sectors. These double sectors will be called blocks. I'll reserve the term 'page' for 256 byte blocks of memory. Cassette tape systems will always read or write 256 byte blocks.

* CP/M is a trademark of Digital Research

SOFTWARE CONSIDERATIONS

The software components will now be described. I envision this as being a part of a ROM monitor in most systems although it could be booted in from disk or tape. To facilitate the discussion I have named the subroutine package FSTCIO (pronounced FaST Cassette I/O). This stems from the fact that the user can view the disks as an extremely fast cassette tape.

The following material was generated as a seperate document and is included as-is to provide a definition of the software standards.

FSTCIO is a collection of user callable subroutines which provide a simplified means of reading and writing data on floppy disks. The package has been designed to work with the floppy disk controller board developed by Tom Crawford. If required, the actual disk access primitives portion could be modified for use with a different controller board while still maintaining the same user interface.

FSTCIO was developed as an interim package to provide floppy disk I/O capability as quickly as possible. The disk system could then be used in the development of a more comprehensive disk operating system. For this reason FSTCIO does not provide any file control services. In fact, the floppy disk may be viewed as a very high speed cassette tape. This fact allows existing software, which uses cassettes, to be quickly modified for floppy disk use.

Before describing the user interface (ie. the subroutine calling conventions) the structure of the floppy disk will be dicussed. Each disk contains 1001 blocks, numbered 0 through 1000. Each block contains 256 bytes of data. Any data which is stored on a disk (ie. a file) will occupy one or more consecutive blocks. It is the users responsibility to remember where each file starts and the number of blocks which are used.

To access data on the disk the user must therefore specify a block number. The FSTCIO routines convert this block number to a track and sector number and perform the desired operation, either a read or a write operation. The unit number must also be specified if there is more than one disk drive. FSTCIO can support up to four disk drives.

FSTCIO consists of nine user callable subroutines. These routines allow the user to initialize the disk controller, open 'files' for I/O, read and write data and 'rewind' back to the beginning of a file. For each file which is to be accessed, there must be a file control block (FCB). An FCB is a nine byte block of RAM which is maintained by the FSTCIO routines. These areas must be allocated by the users program but are not normally modified by the user. The contents of the FCB's are described in APPENDIX A.

The FSTCIO subroutines will be described in the following sections. These routines use the SCRT call and return technique.

FCINIT - Initialize The Floppy Disk Controller

This routine should be called before any disk I/O operations are performed. FCINIT is used to ensure that the controller is in a known state, ready for use. The calling sequence is:

```
        CALL      FCINIT
```

OPEN - Open a File On Disk

This routine 'opens' a file on a specified disk drive for subsequent I/O. The calling sequence is:

```
        CALL      OPEN
        .DBYTE    FCB       ; address of FCB
        .DBYTE    start     ; starting block number of file
        .DBYTE    buffer    ; address of data buffer in memory
        .BYTE     unit      ; drive unit number ( 0 to 3 )
```

RDBLK - Read a Disk Block

This routine reads a block of data into a RAM buffer. The block number and buffer address must have been previously placed into the FCB by an OPEN request. The calling sequence is:

```
        CALL      RDBLK
        .DBYTE    FCB       ; address of FCB
```

RDINC - Read and Increment Block Number

This routine performs a read operation similar to the RDBLK routine. After the read operation the current block number, in the FCB, is incremented. This routine is used to read sequentially through a file. The calling sequence is:

```
        CALL      RDINC
        .DBYTE    FCB       ; address of FCB
```

RDSEQ - Read Sequential Blocks

This routine reads a specified number of blocks into sequential pages of memory. RDSEQ is normally used to read programs or large blocks of data. The calling sequence is:

```
            . . .           ; D-reg contains block count on entry
        CALL      RDSEQ
        .DBYTE    FCB       ; address of FCB
```

WRBLK - Write a Disk Block

This routine writes a data block (256 bytes) onto the disk. The calling sequence is:

```
        CALL      WRBLK
        .DBYTE    FCB       ; address of FCB
```

WRINC - Write and Increment Block Number

The WRINC routine performs a write operation similar to the WRBLK routine. After the operation the current block number, in the FCB, is incremented. This routine is used to sequentially write multiple blocks of a file. The calling sequence is:

```
        CALL      WRINC
        .DBYTE    FCB       ; address of FCB
```

WRSEQ - Write Sequential Blocks

This routine writes a specified number of contiguous pages of memory to the file. The calling sequence is:

```
            . . .           ; D-reg contains block count on entry
        CALL      RDSEQ
        .DBYTE    FCB       ; ADDRESS OF FCB
```

REWIND - Rewind File

This routine rewinds a file by setting the current block number to the first block of the file. The calling sequence is:

```
        CALL      REWIND
        .DBYTE    FCB       ; address of FCB
```

APPENDIX A
FCB CONTENTS

FCB's are nine byte blocks of RAM formatted as follows:

BYTE(s)	NAME	DESCRIPTION
0,1	FBLOCK	starting block number of the file (0 to 1000 decimal)
2,3	CBLOCK	current block number for read and write operations
4,5	ADDRES	address of A RAM buffer which will receive data from a read operation or which contains data for a write operation
6	UNIT	contains the drive unit number (0 through 3) of the drive which contains the file
7	TRACK	track number for current I/O operation
8	SECTOR	sector number for current I/O operation

APPENDIX B
PROGRAMMING EXAMPLES

B.1 FILE COPY EXAMPLE

This program copys a file from drive 0 to drive 1. This would be useful for duplicating disks for backup or generating copies for distribution to other users. Of course this example is only valid for users with two disk drives.

```
COPY:   CALL    FCINIT      ; INITIALIZE CONTROLLER

        CALL    OPEN        ; OPEN INPUT FILE
        .DBYTE  INPUT       ; -- FCB ADDRESS
        .DBYTE  100         ; -- STARTING BLOCK NUMBER
        .DBYTE  BUFFER      ; -- BUFFER ADDRESS
        .BYTE   0           ; -- DRIVE UNIT NUMBER

        CALL    OPEN        ; OPEN OUTPUT FILE
        .DBYTE  OUTPUT      ; -- FCB ADDRESS
        .DBYTE  0           ; -- STARTING BLOCK NUMBER
        .DBYTE  BUFFER      ; -- BUFFER ADDRESS
        .BYTE   1           ; -- DRIVE UNIT NUMBER

        LDI     10          ; R7.LO IS THE BLOCK COUNTER
        PLO     R7          ; (10 BLOCKS WILL BE COPIED)

LOOP:   CALL    RDINC       ; READ A BLOCK
        .DBYTE  INPUT

        CALL    WRINC       ; WRITE A BLOCK
        .DBYTE  OUTPUT
```

```
            DEC     R7          ; DECREMENT COUNT
            GLO     R7
            BNZ     LOOP        ; AND IF NOT DONE LOOP FOR NEXT

            LBR     MONITR      ; RETURN TO MONITOR

INPUT:      .BLOCK  9           ; FCB FOR INPUT FILE
OUTPUT:     .BLOCK  9           ; FCB FOR OUTPUT FILE
BUFFER:     .BLOCK  256         ; BUFFER AREA

            .END
```

This routine could be modified to accept the input and output files starting block numbers and the file size from the users terminal. This would then become a more general and very useful file copy routine.

Alternately, if the counter (register 7) was initialized to 1001 (decimal) and the starting block numbers were set to 0 this routine could be used to copy complete disks.

Of course, in either case the loop control code would need to be modified since the count value is to large for 1 byte.

B.2 PROGRAM LOAD EXAMPLE

This program loads a file into memory. It assumes that some user written code exists which asks the user for, and accepts, a file starting block number, starting memory address and a block count.

```
LOAD:       CALL    FCINIT      ; INITIALIZE CONTROLER
            CALL    OPEN        ; OPEN THE FILE ON DISK
            .DBYTE  FCB         ; - ADDRESS OF FCB
            .DBYTE  0           ; - DUMMY STARTING BLOCK
            .DBYTE  0           ; - DUMMY MEMORY ADDRESS
            .DBYTE  1           ; - FILE EXISTS ON UNIT 1

               .    user code to read load parameters into FCB
               .     -- beginning block number read into CBLOCK
               .     -- memory address read into ADDRES
               .     -- block count read into D-register

            CALL    RDSEQ       ; READ THE FILE
            .BYTE   FCB

            LBR     MONITR      ; EXIT BACK TO MONITOR

FCB:        .BLOCK  9

            .END
```

TEXT FINDER AND DISASSEMBLER FOR 1802

by H. B. STUURMAN

Both of these programs make use of sophisiticated possibilities of the instruction set. Textfinder will search a block of bytes for ASCII characters, while the Disassembler has the options for SEP- or SCRT- techniques by changing one byte.
It is best to use the Textfinder in order to detect ASCII strings. Next use the Disassembler and you'll save a lot of confusion because you'll know already what bytes represent ASCII strings. The Disassembler makes rubbish from these bytes and it is very funny to use, for example, the textfinder on the assembler. You see the structure of data tables evolve, and of course you can use the Textfinder to get insight into the table of the disassembler, where you can see how it is organized into groups of mnemonics.

When I used the Textfinder on the Netronics editor/assembler (by the way very badly documented on how to get them running on non-Elf II machines) I found that somebody is extremely fond of a guy called George.

1802 Textfinder

M0000 uptill and incl. M01FF. The stack is located on the 7 memory places immediate above the program.
The Text Finder starts in R0 and is page relocatible.
Own I/0 included 300 Baud with Q and EF4.

For inverted EF4 (ELF II) change the following addresses.

Address	Byte			
0125	37	(B4)	3F	(BN4)
012D	3F	(BN4)	37	(B4)
01EA	37	(B4)	3F	(BN4)
01EE	37	(B4)	3F	(BN4)
01F8	3F	(BN4)	37	(B4)

After carriage Return (0D) follow 2 null's.
If you want his to be after Line Feed (0A) change M 01C3 in 0A.
The number of null's is in M 01C7 (See note by disassmbler).

If you have your own I/0 you can make the linkage by changing.

Address	
006B	F8 (LDI(
006C	high byte Type
006F	low byte Type
0087	F8 (LDI)
0088	high byte Read
008B	low byte Read

Type and Read start in R3 and return with SEP R5 (D5).
Data exchange in R(F)1.
Available registers for Type and Read: R1, RC, RE, R(F)0 and for Type R(D)0.
This conforms to UT4

If you make a mistake when entering an address keep pushing untill the last 4 entered numbers are correct before hitting Return.

Pushing "Break" halts finding; pushing again lets it continue again.

One last note on Q and EF 4.

The chosen parameters are conform UT 4. That is when Q = 0 20 mA line current is flowing (MARK).
When current is flowing the EF4-flag line is tied low.
The following condition is true: $\overline{EF4}$ = 0 or EF4 = 1; the latter conform the instruction set. But remember the pin is pulled low!

When you want to change the Baudrate of the built in 0/I routines here are the addresses of the timing constants.

Disassembler		Text Finder	
03B8	76	01CF	76
03BC	76	01D3	76
03D6	38	01ED	38
03DD	76	01F4	76

For 110 Baud you could multiply these values by $\frac{300}{110}$ (in Hex).

1.74 MHz. Clock.

1802 Disassembler

M0000 uptill and incl. M03FF included stack.
Start in R0 and is page relocatible.
Has its own I/0 routines 300 Baud with Q and EF4.
For inverted EF4 (ELF II) change the following addresses.

Addresses	Byte			
0125	37	(B4)	3F	(BN4)
012D	3F	(BN4)	37	(B4)
03D3	37	(B4)	3F	(BN4)
03D7	37	(B4)	3F	(BN4)
03E1	3F	(BN4)	37	(B4)

After carriage Return (OD) follow 2 null's.
If you want this to be after Linefeed (0A) change M 03AC in 0A. The number of null's is on M 03B0. You may change this in 6 for an Teletype (also change then M 03AC).

If you have your own 1/0 in ROM or Eprom (for example UT-4), you can link the disassembler with these by changing the following addresses.

Address	
006B	F8 (LDI)
006C	high byte Type
006F	low byte Type
0087	F8 (LDI)
0088	high byte Read
008B	low byte Read

Type and Read start in R3 and return with SEP R5 (D5).
Date exchange in R(F)1.
Available registers for Type and Read: R1, RC, RE, R(F)0 and for Type R(D)0.
This is conform UT4 from RCA.
The disassembler makes no use of the SCRT-technique.

When you make an error by entering an address. Keep entering untill th last4 entered numbers are correct, before pushing Return.
Pushing "Break" (cuts the 20 mA line current) halts the disassembler. Pushing again gets the program going again.

SEP disassembling M 01AA = F8
SCRT disassembling M 01AA = 32.

```
*1802 DISASSEMBLER VER 0
 0000 90B4 B5B6 FC03 B2F8 ABA4 F853 A5AA F817;
 0010 A6F8 F8A2 D424 D492 B8F8 F9A8 4432 16F8;
 0020 06B7 A728 8732 40F6 3331 1808 F6F6 F6F6;
 0030 3808 FA0F FCF6 C7FC 07FF C652 2227 3024;
 0040 97A7 FCF9 A828 1202 5827 873A 4530 1512;
 0050 02AD D48D 5222 44A7 7912 02F6 F6F6 F6F9;
 0060 E0B7 22F8 D552 2297 52D2 90FC 03B3 F8A4;
 0070 A372 BFD3 2787 324F 3062 FC07 3382 FC0A;
 0080 3397 FC00 9FD4 90FC 03B3 F8D3 A3D3 BFFF;
 0090 413B 7AFF 0633 82FE FEFE FEFC 08FE AE8D;
 00A0 7EAD 9D7E BD8E FE3A 9E30 84F8 00A0 E4D5;
 00B0 240D 0A31 3830 3220 4449 5341 5353 454D;
 00C0 424C 4552 0D0A 5645 5220 300D 0A0A 5354;
 00D0 4152 543F 20F8 86A5 BAD5 33D9 FB0D CED0;
 00E0 C49D BB8D AB8A A5FE FF01 3AE8 E4D5 090D;
 00F0 0A45 494E 4445 3F20 9AA5 D533 FAF8 0DCE;
 0100 C4D0 8AA5 FEFF 013A 05E4 D502 0D0A E28D;
 0110 528B F59D 529B 7533 25E4 D508 0A45 494E;
 0120 4445 210A D037 3386 B323 933A 293F 2780;
 0130 FBFF A080 3A25 90FC 02B9 BA80 AAD6 9B58;
 0140 188B 5818 0B58 D6E8 D504 EAD5 02E8 D502;
 0150 F80E A90B 3268 FB60 3267 FAF0 3A86 80B8;
 0160 0BFF 6833 6619 1919 09A9 EAD5 0AE9 D503;
 0170 982A 83EA D502 D60B FF68 CFFC 6858 D618;
 0180 E8D5 011B 3009 F81E A980 A3EB C813 1909;
 0190 328D F3FA F032 A109 F332 BE19 1919 1930;
 01A0 8E83 3A97 0BFB D4CE FF01 329B 19EA D50A;
 01B0 E9D5 04E4 D502 2052 28E8 D501 3083 1923;
 01C0 833A CBEA D50A E9D5 0430 83E8 23D6 1848;
 01D0 5818 0B58 D683 3AEC D502 EAD5 08E9 D504;
 01E0 E4D5 0220 23D6 00E8 D502 3084 D504 EAD5;
 01F0 06E9 D504 E4D5 0220 23D6 00E8 D504 3083;
 0200 2020 2020 2020 2020 2020 2020 2020 1215;
 0210 181B 4944 4C49 5258 494E 504F 5554 0F4C;
 0220 444E 2010 494E 4320 2044 4543 2040 4C44;
 0230 4120 5053 5452 2080 474C 4F20 9047 4849;
 0240 20A0 504C 4F20 B050 4849 20E0 5345 5820;
 0250 D053 4550 2000 D552 4554 4E71 4449 5303;
 0260 7052 4554 0379 4D41 5248 7853 4156 037A;
 0270 5245 5103 7853 4551 03C4 4E4F 5003 3853;
 0280 4B50 0373 5354 5844 724C 4458 41F0 4C44;
 0290 5803 7E53 484C 43FE 5348 4C03 7653 4852;
 02A0 43F6 5348 5203 F241 4E44 03F3 584F 5203;
 02B0 F14F 5203 0377 534D 4203 F753 4D03 0375;
```

```
02C0 5344 4203 F553 4403 0374 4144 4303 F441;
02D0 4444 03CC 4C53 4945 C54C 534E 51CD 4C53;
02E0 5120 C74C 534E 46CF 4C53 4446 C64C 534E;
02F0 5ACE 4C53 5A03 C84C 534B 5000 F84C 4449;
0300 2030 4252 2020 3242 5A20 203A 424E 5A20;
0310 3342 4446 203B 424E 4620 3142 5120 2039;
0320 424E 5120 3442 3120 203C 424E 3120 3542;
0330 3220 203D 424E 3220 3642 3320 203E 424E;
0340 3320 3742 3420 203F 424E 3420 FC41 4449;
0350 207C 4144 4349 FD53 4449 207D 5344 4249;
0360 FF53 4D49 207F 534D 4249 F94F 5249 20FB;
0370 5852 4920 FA41 4E49 2000 C04C 4252 20C2;
0380 4C42 5A20 CA4C 424E 5AC3 4C42 4446 CB4C;
0390 424E 46C1 4C42 5120 C94C 424E 51D4 4753;
03A0 5542 9FD5 93BC F8EA AC9F BEFB 0D3A B2F8;
03B0 0138 80A0 F80A AFDC 76FC 00DC 76C7 7A30;
03C0 7B9E FF00 76BE 2F8F 3ABB 8032 A220 F800;
03D0 BE30 B437 D3DC 3837 D3F8 7F8F DC76 9FF6;
03E0 BF3F E5F9 8033 DB30 A9D3 43AE 2E8E 3AEC;
03F0 30E9 4332 34E4 D545 3043 3033 3034 3200
```

1802 TEXT FINDER.
AUTEUR HB STUURMAN.

```
?M0 200
0000 90B4 B5B6 FC02 B2F8 ABA4 F853 A5AA F817;
0010 A6F8 03A2 D424 D492 B8F8 04A8 4432 16F8;
0020 04B7 A728 8732 40F6 3331 1808 F6F6 F6F6;
0030 3808 FA0F FCF6 C7FC 07FF C652 2227 3024;
0040 97A7 FC04 A828 1202 5827 873A 4530 1512;
0050 02AD D48D 5222 44A7 7912 02F6 F6F6 F6F9;
0060 E0B7 22F8 D552 2297 52D2 90FC 01B3 F8BB;
0070 A372 BFD3 2787 324F 3062 FC07 3382 FC0A;
0080 3397 FC00 9FD4 90FC 01B3 F8EA A3D3 BFFF;
0090 413B 7AFF 0633 82FE FEF8 FEFC 08FE AE8D;
00A0 7EAD 9D7E BD8E FE3A 9E30 84F8 00A0 E4D5;
00B0 240D 0A31 3830 3220 5445 5854 2046 494E;
00C0 4445 5200 0D0A 5645 5220 300D 0A0A 5354;
00D0 4152 543F 20F8 86A5 BAD5 33D9 FB0D CED0;
00E0 C49D BB8D AB8A A5FE FF01 3AE8 E4D5 090D;
00F0 0A45 494E 4445 3F20 9AA5 D533 FAFB 0DCE;
0100 C4D0 8AA5 FEFF 013A 05E4 D502 0D0A E28D;
0110 528B F59D 529B 7533 25E4 D508 0A45 494E;
0120 4445 210A D037 3386 B323 933A 293F 2780;
0130 FBFF A080 3A25 F810 B9A9 D69B 5818 8B58;
0140 D6E8 D504 E4D5 0120 3818 D600 0B58 D6E4;
0150 D501 20E8 D502 2989 3A49 8BFF 0FAB E4D5;
0160 070D 0A20 2020 2020 99A9 E40B FE33 94F6;
0170 FC81 3394 FFA2 3887 D502 2020 EBD5 0138;
0180 1B29 893A 6A30 090B FB0D 3298 FB07 32A2;
0190 FB2A 32A9 D503 2020 2030 80D5 0320 4352;
01A0 3080 D503 204C 4630 80D5 0320 205F 3080;
01B0 D343 AE2E 8E3A B330 B09F D593 BCF8 B1AC;
01C0 9FBE FB0D 3AC9 F802 3880 A0F8 0AAF DC76;
01D0 FC00 DC76 C77A 387B 9EFF 0076 BE2F 8F3A;
01E0 D280 32B9 20F8 00BE 30CB 37EA DC38 37EA;
01F0 F87F BFDC 769F F6BF 3FFC F980 33F2 30C0
```

Tom Jones
409 Springdale ave
Enterprise, Alabama
36330

Editor/Assembler/Monitor for video

What seems trivial may not be so in practice-at least so it is with me. This describes the tieing together of George Millar's Editor/Assembler, a video driver, and Steve Nies's monitor to form a program development package that works well.

The named file load/save functions make keeping track of source files and assembled routines much easier, and the debug functions (break, Trace,Register exam/dep) find the bugs in the assembled code faster. Video output is faster than a teletype for the initial editing, and saves paper too. I hope to interface John Beringer's "disassembler" to the package as well, soon. Working together, this software puts any 1802 in a whole new class. The package runs in 6k ram, 1k refresh, and 256 bytes for the monitor scratch-pad.(I use ELFII's original).

1. Load "The Monitor" to 1000-17FF, using the ElfII Monitor.
2. Load "Kbscrn.RCA" to E200-E2FF, using the big monitor. Changes included in KBSCRN.RCA are:

E218 - E0	E260 - E2	E2b9 - E1
E222 - E0	E27B - E2	E208 - 9F
E251 - 03	E286 - 9E	E295 - E0
E26B - E0	E289 - BE	E2C1 - E0
E28F - E1	E28B - E0	E2B5 - DF
E2A2 - E2		

3. Load "Editor" to 0000-0FFF. (Assembler will overlay the same memory area later to assemble the source file.)New code included is:

```
0F40 - 3F40          KBIN:    BN4 KBIN ;
0F42 - 6A                     INP 2;
0F43 - BF                     PHI RF;
0F44 - D4E20E                 CALL KBSCRN ;
0F47 - D5                     RETN;
;
0F48 - 3F48          WHOA:    BN4 WHOA ;
0F4A - C01000                 LBR MONITOR ;
```

NOTE: My ELFII uses INP 2 and EF4 for ascii Kbd.

4. Linkage assignments:

Editor	Assembler
0263 - C0E20D	01B6 - C0E20D
0266 - C00F40	01B9 - C00F40
03B9 - C00F48	0080 - C00F48

5. Operating Notes; see references. Some differences are:

a.) Cursor controls, clear, Clr to end of screen, Home cursor.
b.) When messages from the assembler/editor are displayed, press any ascii key to continue to monitor.
c.) You really must put a space after a label in an op field, or crazy things can happen. Other "snafus" are using DBYTE for DBYT, and RET for RETN.
d.) The "FFF" command does show the end of the source code, but the address is for the space preceding the last statement (the END statement). Add 5 to the address when saving the source area on tape.
e.) The Assembler exits to Monitor if pass 1 has an error, after displaying the address in source of the error. It is often possible to list the area in question and correct the error without reloading the editor, but pass 1 always changes location 0700. You should reinsert a space (20) before return.
f.) The Editor list command "LOF" is useful for video output because it lists one screen, 16 lines. It would be nice if the assembler full list mode also paused in the same manner until a key was pressed, but until the disassembler works I cannot do it. This would only be of interest to printerless people.
g.) The available source area allows about one page of assembled object code to be created. This is more than most subroutines in a program need, and version 2 of the monitor allows any length file to be loaded to any location of memory, thereby lending itself to a manual linking of program modules into a larger one. Anyone want to write a "Linker"?

6. Cassette copys: I have enjoyed using these programs and feel they would encourage software production on 4K or more 1802's. I hope most members will try them, but keying in the full 19 pages of machine code is a dauting task. I will, therefore, attempt to supply a copy of the whole package to anyone who sends a cassette and return postage. I only have the Netronics Monitor and Steve's Monitor, but by putting the former on the fornt of the tape you can u use the "ELFII Keyin loader" to boot it all in on any 1802. I would be interested to see how many other people's tapes I can read also, so please put something on the tape you send. I could use Palo Alto TBI, Pittman's TBI, or games.

REFERENCES:

IF# 8, p. 10 "Software for the 16*32 Video"
IF #12, p 60 "1802 Editor/Assembler"
IF #13, p26 "1802 Editor/Assembler Fixes"
IF #14, p26 "1802 Editor/Assembler Fixes"
IF #9, p.34 "Key-in Loader for ElfII Tapes"
IF #8, p.21 "Tape conversion, ELF to VIP/VIP to ELF"
Microcomputing, July 1980, p.196 "Disassembler for 1802"
IF #16, p.4 "The Monitor"
IF #17, p.17 "The Standardized 1802"

ELF II SERIAL INTERFACE

IF YOU HAVE THE NETRONICS ELF II WITH ASCII TERMINAL AND VIOEO BOARD, AND YOUR TERMINAL IS SET UP AS AN RS-232 OEVICE RUNNING AT 300 BAUD, THEN THIS ARTICLE MAY BE OF INTEREST TO YOU.

EVER SINCE I FIRST STARTED RUNNING TINY BASIC ON MY MACHINE, I WONOERED HOW THE TERMINAL COMMUNICATEO WITH THE CPU. WHAT I DION'T UNDERSTANO WAS HOW THE SERIAL BIT STREAM REPRESENTING THE ASCII CHARACTER WAS PUT BACK TOGETHER AS A BYTE OF INFORMATION ONCE IT GOT INTO THE MACHINE AND VICA VERSA. WHEN I RECEIVED MY OE FACTO IN THE MAIL A FEW WEEKS AGO, I FOUNO AN ARTICLE BY MR. B. MURPHY IN IPSO FACTO #5 P.41 , WHERE HE OESCRIBES SUBROUTINES WHICH PERFORM SERIAL RECEIVING AND TRANSMITTING FOR HIS TEC-1802. WITH A FEW MODIFICATIONS TO AOAPT TO THE ELF II ENVIRONMENT I HAO THE SUBROUTINES WORKING AS LISTED BELOW. THE MOST TIME CONSUMING PART OF THE OPERATION WAS FINDING A TIME CONSTANT FOR THE DELAY SUBROUTINE. I FINALLY DISCOVERED THAT A VALUE BETWEEN HEX 73 ANO HEX 80 WOULD WORK ON MY SYSTEM, HENCE THE VALUE HEX 78. I KEEP THESE SUBROUTINES ON TAPE SO THAT THEY CAN BE LOAOED INTO CORE WHEREVER NEEDEO. THE AODRESSES IN THE LISTINGS BELOW START WITH 'XX' WHICH MAY BE CHANGEO TO ANY PAGE TO SUIT YOUR NEEDS. A TEST PROGRAM IS INCLUDED IN THE LISTINGS. IT WILL REAO AN ASCII CHARACTER FROM THE KEYBOARD, DISPLAY THE CHARACTER HEX EQUIVALENT ON THE HEX DISPLAY, AND TRANSMIT THE CHARACTER BACK TO THE VIDEO OISPLAY.

ONCE THESE SUBROUTINES ARE OPERATING ON YOUR SYSTEM, THE POSSIBILITIES FOR CLEVER PROGRAMS ARE ENDLESS. YOU CAN FINALLY BEGIN WORKING ON THAT MONITOR/EOITOR YOU'VE BEEN DREAMING ABOUT. SUDDENLY THE FULL CHARACTER SET OF THE KEYBOARD IS AVAILABLE TO YOUR MACHINE CODE PROGRAMS, ANO YOUR PROGRAMS CAN TALK TO THE VIDEO OISPLAY. ALSO DON'T FORGET THE ESC KEY ON YOUR TERMINAL, AND ITS CAPABILITIES FOR POSITIONING THE CURSOR ON THE VIOEO SCREEN. HAVE FUN !!!

DIRK J. JORENS
315-645 SHELTER CREEK LANE
SAN BRUNO, CALIFORNIA, 94066

TRANSMITTER SUBROUTINE

REGISTERS R5 PROGRAM COUNTER
R4.0 COUNTS 8 BITS
R4.1 HOLDS ASCII CHARACTER

INPUTS : CHARACTER TO BE SENT MUST BE IN ACCUMULATOR
OUTPUT : Q LINE SERIAL OUTPUT

```
XX00   D3          RETURN TO MAIN PROGRAM
 01    B4          SAVE ASCII CHARACTER IN R4.1
 02    7A          RESET Q
 03    F8 08 A4    SET UP R4.0 TO COUNT 8 BITS
 06    7B          SET Q (START BIT)
 07    D6          CALL DELAY SUBROUTINE
 08    94          LOAD ASCII CHARACTER INTO D
 09    76          SHIFT D RIGHT INTO DF
 0A    B4          SAVE D IN R4.1
 0B    CF          LONG SKIP IF DF=1
```

```
          0C    7B             SET Q  ( DF=0 )
          0D    38             SKIP NEXT INSTRUCTION
          0E    7A             RESET Q  ( DF=1 )
          0F    D6             CALL DELAY SUBROUTINE
          10    24             SUBTRACT 1 FROM R4.0
          11    84 3A 08       IF R4.0 > 0 , REPEAT PROCESS
          14    7A             RESET Q  (STOP BIT)
          15    D6             CALL DELAY SUBROUTINE
          16    D6             CALL DELAY SUBROUTINE
          17    30 00          RETURN
```

```
RECEIVER SUBROUTINE              REGISTERS  R5 PROGRAM COUNTER
------------------                          R4.0 COUNTS 8 BITS
                                            R4.1 CONTAINS ASCII CHARACTER

                                 INPUTS:  FLAG EF4
                                 OUTPUT:  ACCUMULATOR CONTAINS THE
                                          ASCII CHARACTER

        XX1D    D3               RETURN TO MAIN PROGRAM
          1E    F8 00 B4         CLEAR CHARACTER STORAGE
          21    F8 08 A4         SET UP R4.0 TO COUNT 8 BITS
          24    3F 24            WAIT FOR START BIT
          26    F8 3C            LOAD D WITH 1/2 BIT TIME
          28    16               ADD 2 TO R6 SO CALL TO DELAY WILL
          29    16               ENTER AT XX43 INSTEAD OF XX41
          2A    D6               CALL DELAY SUBROUTINE
          2B    3F 24            IF BAD START BIT , GO BACK TO WAITING
          2D    D6               CALL DELAY SUBROUTINE
          2E    FC 00            RESET DF
          30    37 34            IF BIT, LEAVE DF RESET
          32    FF 00            SET DF
          34    94               GET R4.1 CHARACTER STORAGE
          35    76               SHIFT DF INTO ACCUMULATOR
          36    B4               PUT CHARACTER BACK INTO R4.1
          37    24 84            SUBTRACT 1 FROM BIT COUNTER
          39    3A 2D            IF R4.0 > 0 , REPEAT PROCESS
          3B    D6               CALL DELAY SUBROUTINE (STOP BIT)
          3C    94               PUT ASCII CHARACTER INTO D
          3D    30 1D            RETURN
```

```
DELAY SUBROUTINE                 REGISTERS  R6 PROGRAM COUNTER
----------------                            R7.0 DELAY COUNTER

        XX40    D5               RETURN TO CALLER
          41    F8 78 A7         LOAD 1 BIT TIME DELAY INTO COUNTER
          44    27               SUBTRACT 1 FROM COUNTER
          45    87 3A 44         IF R7.0 > 0 , REPEAT PROCESS
          48    30 40            RETURN
```

TEST PROGRAM

REGISTERS R1 INITIAL PROGRAM COUNTER
R2 STACK POINTER
R3 MAIN PROGRAM COUNTER
R5 RX/TX PROGRAM COUNTER
R6 DELAY PROGRAM COUNTER

```
0000    C0 F0 00        CALL ROM MONITOR
  03    E2
  04    F8 00 B2 B3
  08    F8 25 A2        LDAD R2 WITH STACK ADDRESS
  0B    F8 16 A3        LOAD R3 WITH MAIN PROGRAM ADDRESS
  0E    F8 01 B5 B6     SUBROUTINES ARE IN PAGE 01
  12    F8 41 A6        LOAD R6 WITH DELAY SUBR. ADDRESS
  15    D3              CALL MAIN PROGRAM
  16    F8 1E A5        LOAD R5 WITH RECEIVE SUBR. ADDRESS
  19    D5              CALL RECEIVER
  1A    52              PUT D ON STACK
  1B    64              DISPLAY ON HEX OUTPUT
  1C    22              DECREMENT STACK DUE TO '64' INSTRUCTION
  1D    F8 01 A5        LOAD R5 WITH TRANSMIT SUBR. ADDRESS
  20    02              LOAD D FROM STACK
  21    D5              CALL TRANSMITTER
  22    30 16           REPEAT
  24    00 00           STACK
```

RECYCLING THE NETRONICS MATH BOARD. M. Franklin, Feb. 1981.

The Math Board supplied by Netronics with their Full Basic package was designed to provide a calculator math function in hardware and accommodate 8k of 2716 type Eprom, single volt, addressed from 0000.

I don't use Full Basic much anymore, certainly not enough to invest in 3 Eproms, but I would like to re use the $40. worth of hardware. Tnis article will outline the modifications necessary to re use the Eprom space, and raise the challenge for someone to write a callable subroutine to access the Math package from any program.

Recycling the Eprom space.

The Eprom space is currently configured to accommodate 4 single volt 2716 Eproms addressed from 0000 to 1FFF. All three parameters may be changed.

Changing the Chip.

THe following table gives the functional assignments of the 4 pins that differ between the uni and tri volt Eproms.

pin	2716(5v)	2716(3xv)	2758	2708
18	CS	CS	CS	GND
19	A10	+12v	GND	+12v
20	GND	A10	GND	CS
21	+5v	-5v	+5v	-5v

Notice that the uni volt pair have only one pin different, thus it would be relatively easy to mix the two chips if desired. If 1k Eproms are used, the method of decoding will also have to be changed, else the chip would be enabled for 2k.

Changing the Address Space.

The Eprom space is decoded as 4 adjacent 2k spaces using a 74LS138-3 to 8 decoder as follows:

LS138 pin #	4	5	6	1	2	3	15	14	13	12
function	enable decoder			address decoder			chip select(low enable)			
name	E1	E2	E3	A1	A2	A3	Q0	Q1	Q2	Q3
logic level	L	L	H	L	L	L	L	H	H	H
				L	L	L	L	H	H	H
				L	H	L	H	L	H	H
	REPEATS			L	H	L	H	L	H	H
				H	L	L	H	H	L	H
				H	L	L	H	H	L	H
				H	H	L	H	H	H	L
				H	H	L	H	H	H	L

In the LS138, the 3 enable pins are ANDED to enable the chip. The address decoder acts as a 2 to 4 decoder since pin #3,(A2) is held low, effectively using only the lower half of the chip selects available, but each chip select is held low for 2k.

To use 1kEproms, address decode pin #3 must be changed to A10 from ground. The LS138 will now decode 8 adjacent 1k blocks of memory.

To change the address location, it is necessary to change the logic level of the inputs to the enable decoder, A13, A14, and A15 respectively, to permit the enable decoder to enable at a specific address. The Math Board latches the upper addresses with a 74LS174 for A8 throughA13, providing a Q output only, and with a 74LS74 flip flop for A14 and A15, providing both a Q and $\overline{Q}$. A spare inverter is available on U4, pins 14/15, to provide a $\overline{Q}$ for A13 if necessary.

The following table lists the logic levels for A13 through A15 for the eight 8k blocks of memory. The enable logic required is E1=0(A13), E2=0(A14), and E3=1(A15). The appropriate Q or $\overline{Q}$ is listed on the right.

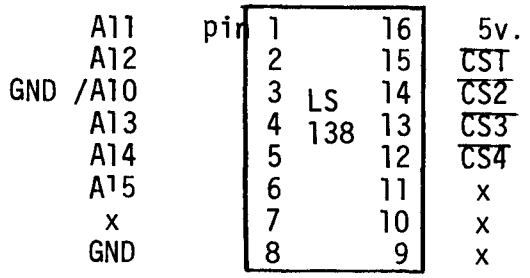
8k block	buss address logic level			enable decode logic required		
	A13	A14	A15	A13	A14	A15
0-1	0	0	0	Q	Q	$\overline{Q}$
2-3	1	0	0	$\overline{Q}$	Q	$\overline{Q}$
4-5	0	1	0	Q	$\overline{Q}$	$\overline{Q}$
6-7	1	1	0	$\overline{Q}$	$\overline{Q}$	$\overline{Q}$
8-9	0	0	1	Q	Q	Q
A-B	1	0	1	$\overline{Q}$	Q	Q
C-D	0	1	1	Q	$\overline{Q}$	Q
E-F	1	1	1	$\overline{Q}$	$\overline{Q}$	Q

SOURCE:
A13 Q U8 pin 10
$\overline{Q}$ U8 pin 10 to U4 pin 14, out U4 pin 15
A14 Q U7 pin 5
$\overline{Q}$ U7 pin 4
A15 Q U7 pin 9
$\overline{Q}$ U7 pin 8

For example, if you wanted the Eprom located from C000 to DFFF, it would be necessary to connect the Q of A13, U8 pin 10, to U9 pin 4, the $\overline{Q}$ of A14 from U7 pin 4 to U9 pin 5, and the Q of A15 from U7 pin 9 to U9 pin 6.

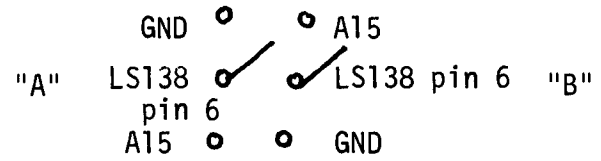
Splitting the 8k.

The next option requires that an additional LS138 be wired onto the board in the kluge space. By decoding the address lines to enable the decoder at another location, it is possible to split the 8k block into two units, from 2 to 6k each. Just connect the chip select lines for the appropriate socket to the second LS138. Again if 1k Eproms are used, it will be necessary to connect A10 to the new LS138 pin 3. In this manner, one or more Eproms could be placed into high memory to accommodate monitors, while the remainder placed at a low address for a program. The following pin out gives the appropriate connections.



Shadowing the Eprom

By adding a DPST switch to the board to disable the enable circuit of the LS138, it is possible to accommodate two programs at the same address in separate Eproms and access only the one desired at any given time. The key to this option lies in the requirement of the LS138 that pins 4,5, and 6 be at logic levels 0, 0, and 1 respectively before the chip will decode the three address lines. By preventing one of the three from being at the correct logic level, it is possible to allow a second LS138 decoder to work while preventing the first from operating. The easiest manner in achieving the above is to connect A15 to the switch such that pin 6 of one LS138 will be held to ground while the second is connected to A15. The following diagram illustrates a suitable connection.



In the above diagram, the "B" LS138 will be enabled while the "A" LS138 will be disabled.

A variation of this option that I have not tried would be to hold both LS138 decoders in the disabled mode and enable a third RAM board to allow direct access to the same address area for other programs. One possible method of achieving this would be to use a centre off DPST switch and tie both LS138 pin 6 inputs to ground through a resister of about 2k. With the switch in the centre off position, both LS138s would be disabled, allowing the operator to bring a RAM board to the same address location. With the switch in either pole position, A15 would defeat the pull down of the resister when it was driven high, thus enabling the decoder. Be sure that A15 is protected with a buffer to prevent damage to the 1802 and to provide sufficient drive capability to defeat the resistor.

The Math Package

I think it would be useful to be able to access the math package without using Full Basic. Consider options such as a scientific calculator directly accessible from the keyboard, precise calculations for real time games, better math for Tiny Basic, conversion of Quest Basic to hardware math, whatever. I remember reading an article on this chip in an old Kilobaud article, and there is an article in Radio Electronics, Dec. 1979 I think, about using this chip. Perhaps one of the club members is already using the 57109 in this mode and would provide an article on the program required.

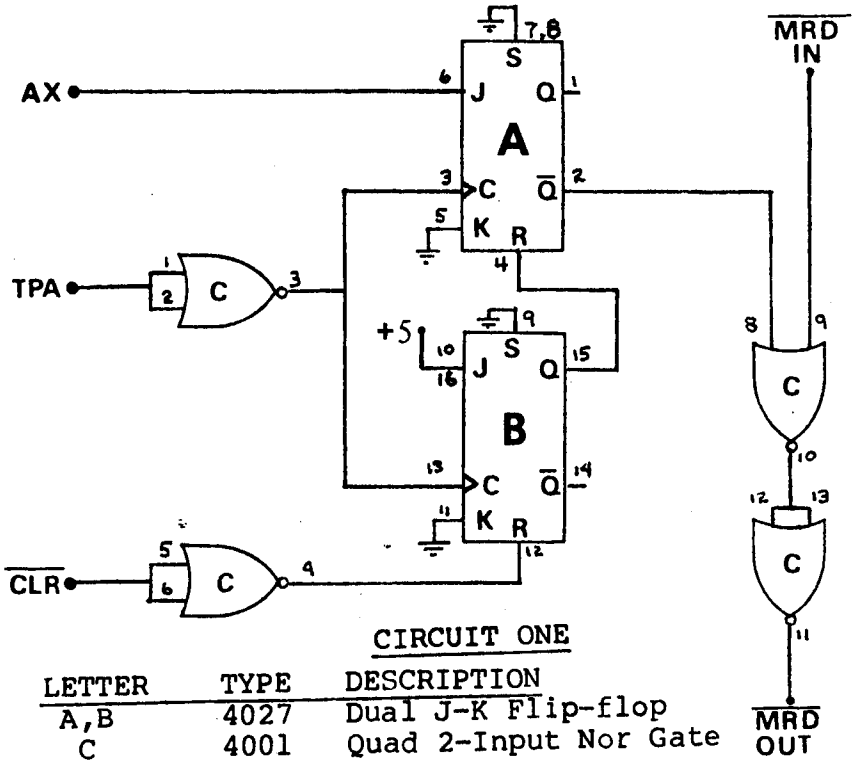
HARDWARE RAMBLINGS

by Tony Hill
1202-165 Queen St. S.
Hamilton, Ontario
L8P 4R3

This article contains a bunch of little doodads and whatnots I've had sitting around for a while. While I don't take any resposibility for what happens to your system (or you) if you try them, they've all worked for me.

1) ANOTHER BOOT CIRCUIT

Having read, with great interest, Tom Crawford's article in I.F.15 on boot circuits for the 1802, I found myself still unhappy with his final solution. I really did not want to have to use an EPROM to build the circuit he designed, because of the cost and board space required. While these are minor points, I looked for a different solution. The circuits shown below resulted.



CIRCUIT ONE

LETTER	TYPE	DESCRIPTION
A,B	4027	Dual J-K Flip-flop
C	4001	Quad 2-Input Nor Gate

As everyone knows, the 1802 starts program execution at location 0000 after a reset is performed. For the benefit of those who did not read Tom's article, a boot circuit is a way of starting execution at somewhere other than 0000H. (Please don't write me irate letters for the last statement. I know it's an oversimplification, but its correct enough for now.)

The basis of my circuits is to disable the $\overline{\text{MRD}}$ line to memory after a reset is performed. This may seem like a stupid idea, but what it does is force the CPU to fetch whatever value the data buss floats at and execute that value as an instruction. If you have pull up resistors on the buss (you'll need them if you don't - try 33K) , then that

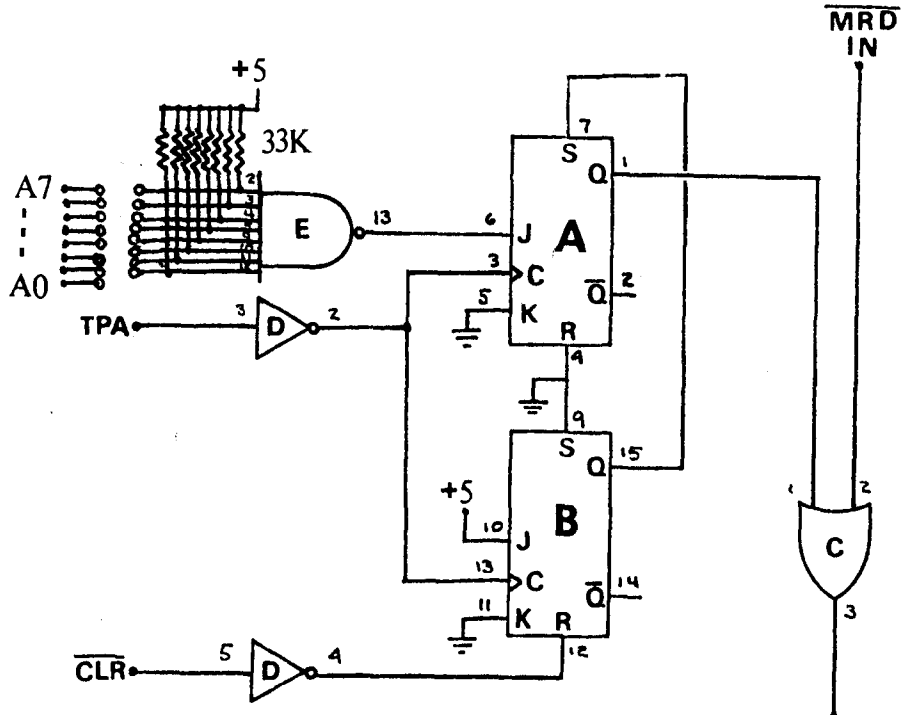
instruction will be an FF, known to the 1802 as a Subtract Memory Immediate. If you leave the $\overline{MRD}$ line disabled the 1802 will advance through memory and execute SMI's forever. (Or until the D register becomes so negative that the number of borrows performed empties the bit bucket and shuts the CPU down).

However, long before CPU shutdown, the rest of my boot circuit will step in and save the day (or at least the 1802, which may not be a blessing at all). What the circuitry does is monitor the address buss, looking for the first occurance of a high order bit or bits. As the CPU works its way up through memory, merrily subtracting away, the address sent out advances. "So what?" you say. Well, it means that if you want to boot to 8000H, you only need to watch for address line 7 to be high when TPA is high. You don't need to watch the whole buss, and as a result may need less chips. I built circuit 1 with $1.00 worth of chips to boot my system to 0200H.

TABLE OF POSSIBLE BOOT ADDRESSES
Connect AX from circuit to required address line.

ADDRESS LINE	BOOT ADDRESS
A0	0100H
A1	0200H
A2	0400H
A3	0800H
A4	1000H
A5	2000H
A6	4000H
A7	8000H

CIRCUIT TWO



LETTER	TYPE	DESCRIPTION
A,B	4027	Dual J K Flip-flop
C	4071	4x2 input Nand gate
D	4049	Hex inverter
E	4068	8 input Nand gate

So much for the theory. Here's how circuit 1 works. At reset, flip-flip B is reset, and it resets flip-flop A. When flip-flop A is reset it disables the $\overline{MRD}$ line through the OR gate connected to its output. During the 1802 initialization cycle, flip-flop B sets, thus enabling flip-flop A to be set on the next instruction cycle. If, during any execution cycle the input to flip flop A is a 1, it sets and allows $\overline{MRD}$ pulses to pass through the OR gate combuination. 1802 operation then runs normally (if you call an 1802 normal). If you reset the system again, the whole cycle starts over.

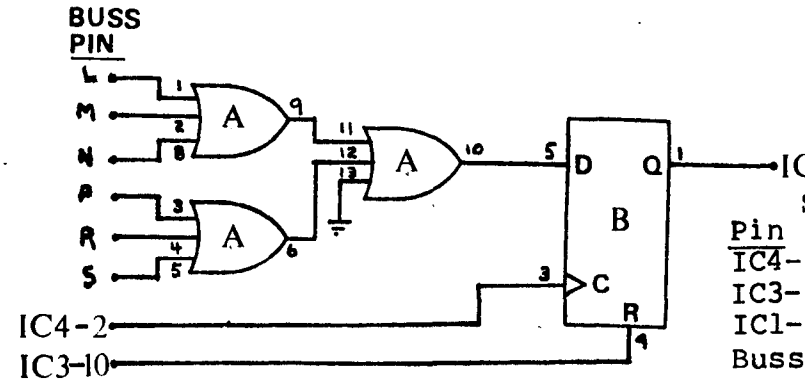
Circuit 2 is a fancy version of circuit 1, that allows you to boot to any 256 byte page you desire. I haven't built it, because I don't need it, but I include it for interest sake.

For both circuits, open up the $\overline{MRD}$ line before it reaches any memory chips (i.e. on the CPU board probably). Make sure you have pull-ups on both sides of any buss buffers. Other than that, there's not too much too it. A buck is not too much to spend if it saves you entering a LBR to your monitor by hand every time you reset.

2) TEKTRON 3/4K CMOS RAM MODIFICATIONS

The following is a list of some modifications that may be useful to those people who wish to the use their TEKTRON 3/4K CMOS Ram board in their expanded systems. They are designed to make the board compatable with the ACE buss structure.

Full 64K memory decoding should be added, as the TEKTRON board only decodes to 1K and cannot be used in an expanded system. I use the circuit shown below to do this.



SIGNAL CONNECTIONS

Pin	Name
IC4-2	TPA (inverted)
IC3-10	CLEAR*WAIT
IC1-5	Board select line
Buss L-U	Address buss A2 - A7

IC	#	Description	Pins tied to +5	Pins tied to GND
A	4075	3x3 input OR	14	7, 3
B	4013	Dual D flip-flop	14	6,7,8,9,10,11

As drawn, it maps the board at address 0000H. Any other 1K page could be used by inserting inverters in the appropriate connection between the address buss and the OR gate (i.e. the IC marked A, pins 1,2,8,3,4 or 5). For example, one inverter gate from a 4049 inverter pack inserted between buss pin L and the OR gate would map the board at 8000H.

I built the circuit on a small piece of PC board and glued it to some empty space near the top of the memory board (so it extends above the board). Then I ran wires along the bottom of the boards to connect the required points together. I etched a PC board for the IC's, but the connections to the TEKTRON board are all point to point wiring, so the whole circuit could probably have been built this way.

The connections shown in the schematic are simply soldered to the pins on the Tektron board they refer to. The Buss Pins refer to the edge connector pins on the memory board. The IC numbers refer to the IC's on the Tektron board. Note that no trace cuts or modifications need be made to the Tektron board, except those noted below that make the board conform to the ACE buss.

To conform to the club buss structure, the TEKTRON board connections to edge connector pins 2 and B should be cut. This will not affect board operation. If you do not need the 256 byte memory on the Tektron CPU board, also cut traces going to edge connector pins 21 and Y. The board will then be completely club buss compatable.

3) CONVERTING FROM 2708's TO 2716's

Now that 2716's are cheaper by 1/2 per 1k block, I decided to convert my 2708 board to use 2716's. I did this not only to save money (as I had not yet purchased any 2708's) but to save the trouble of worrying about using three power supplies with the 2708's. In addition, I can now fit a full 16K of Eprom on what was formerly an 8K board.

The only difference between a 2708 and a 2716 is in the function of four of the pins. The rest of the pins are identical, so by changing only four lines, you can unplug a 2708 and plug in a 2716.

The pins that are different are documented below:

PIN	2708	2716
18	Program Pulse	Chip Enable- selects chip when low
19	Vdd - +12V	A10 -address line 10 (for second K)
20	CS/WE- selects chip	OE - tie low for normal read operation
21	Vbb - -5V	Vpp- Program Pulse - tie to +5v for normal read operation

A look at the above table shows that if you tie pin 20 to ground and pin 21 to +5V on your 2708 socket, then you merely have to hook up your chip select logic and the tenth address line to convert to it to a 2716 socket. The amount of modifications necessary to convert your chip select logic to 2K banks from 1K banks will depend on your circuit, but I only had to cut and jumper four wires, plus one wire per 2716 used.

4) A NOTE ON RCA PARTS-

If you are having trouble getting 1856's, and need them for an Eprom board, you can do like I did and use 1857's. **BUT** be aware that they are the same EXCEPT for pin 10, which requires an inverted $\overline{MRD}$ signal to work. You can either invert the signal with an inverter pack or, if you only require the device to transfer data from memory to the 1802, you could tie it to +5V. This would be the case with an Eprom board. Not only does this solve your chip supply problem, but 1857's are cheaper.

ERRATA Ipso Facto 19 page 29

More On "CASSETTE LOAD VOLUMN SENSITIVE?"

Post Script: After making the changes to your tape head of your cassette as mentioned in the previous artical, it may be necessary to change the poliarity output of your Giant Board as it was necessary in mine. Just follow the poliarity output recommendations mentioned in The Giant Board Instructions and you will have no problem. If any on has problems with this, you are welcome to write or call me.

Bill Eckel 7711 South 73 Avenue Omaha, Nebraska 68128 Phone: (402) 592-3044

ERRATA Ipso Facto 20 page 35

I found an error in the hex dump of the counter program by Ken Mantei on page 35 of Ipso Facto issue number 20. Memory location XX 6E reads 20. It should be 30.

With this change the program runs fine.

However I find that the display time is too fast for reading the frequency on the leds. It is probably all right using a 7 segment led display.but with the basic Tektron job, it is too fast. I therefore fiddled with the initialization from memory location XX00 to memory location XX13 as follows:

```
MA   XX00    F8 48 B1
       03    F8 D3 A1 B6
       07    F8 00 A3 B3 A4 B4 A6
       0E    90 B2
       10    F8 81 A2
       13    --------same as before to end. except
             XX6E should be 30 not 20.
```

This gives a little more than 2 seconds to read each display.

Ed.Leslie 16 Sterling Street, Hamilton Ontario, L8S 4H7,

ERRATA Ipso Facto 14 page 10

The attached errata is self explanatory. Contact with several hobbyists regarding the dynamic ram controller, and my recently building another unit utilizing another 16K of dynamics (an 1802/1861 micro) revealed numerous errors in the schematic in IPSO FACTO. I am sorry if this has inconvenienced anyone. I now see the utility of 'camera ready' material; no translation errors via the re-drafting.

Sincerely,

Harley Shanko

ERRATA - "1802 DRIVEN DYNAMIC RAM CONTROLLER"

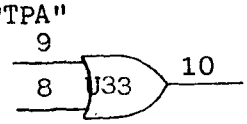
IPSO FACTO, Issue #14

Reference is to Schematic/Logic Diagram, page 10.

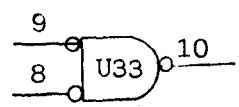
1. Lower, left: a) Signals shown; "$\overline{\text{TPA}}$" and"$\overline{\text{TPB}}$"
   should be; "TPA" and "TPB"
   b) U33 input pins; "11" and "12"
   should be; "12" and "13"

2. Center, left: a) U5 inputs : "U5-8 MUX" and "U5-12 $\overline{\text{MUX}}$"
   should be; "U5-8 $\overline{\text{MUX}}$" and "U5-12 MUX"

3. Lower, center: a) U40-8 - missing input signal
   should be; "TPA"
   b) U33 is;

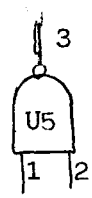


   should be;

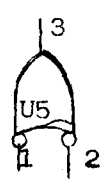


   to left and below (De Morgans theorem) is original representation to illustrate the desired function of the logic terms

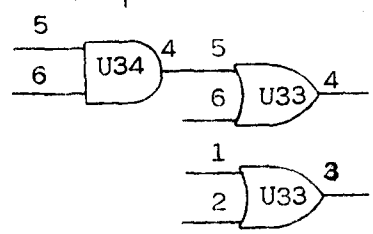
   c) U5 is;



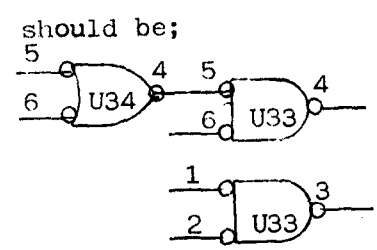
   should be;



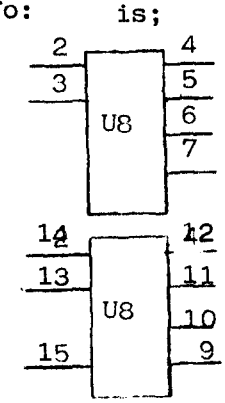
4. Lower, right a) U34,33 is;



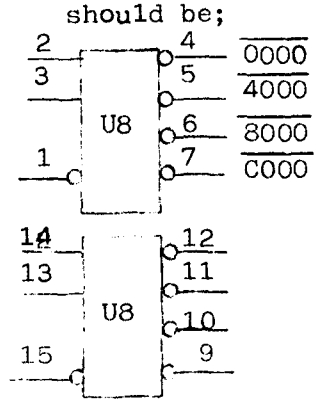
   should be;



   b) U8 missing info: is;



   should be;



INTRODUCTION

The Association of Computer Experimenters (ACE) was founded in the spring of 1977 largely as a result of the efforts of Tom Crawford and Eugene Tekatch. Eugene had been requested by the Institute of Electrical and Electronic Engineers (IEEE) Canada to produce a CPU board and to give an introductory course on micro computer technology and application to its members. He chose the RCA 1802 m.p.u. and designed the TEC 1800 system, adopting products from his company's industrial controller-processor line. The course commenced in the winter of 1977 and proved extremely popular. Some 600 people attended the three day seminars in the first few months. Tom was one of those students. Both Tom and Eugene recognized the increasing popularity of the new "micro computing" hobby, and the strong interest of course participants in carrying on beyond the seminars. The idea of a hobby users group was born. Tom co-opted the support of friends and fellow course participants, and with Eugene's help, produced issue 1 of Ipso Facto, a 12 page newsletter sent out to all course participants. The popularity of the newsletter prompted a second issue and a call to interested people to form a club.

The first meeting was held in the Dofasco steel works in Hamilton in September 1977. From the early efforts of a few interested people, ACE has grown to more than 575 members, has become a club of international stature, and publishes the most significant source of 1802 micro processing user-information in the world. The club newsletter, Ipso Facto, is published six times a year and averages 50 pages per issue.

In the intervening three years, the members have grown with their hobby. The first elementary machine language programs to turn on the "Q" light or display static bit patterns with the 1861 have given way to sophisticated control programs, higher level languages and personalized monitors. Hardware development has also advanced. No longer content to put together someone else's kits, members have designed and built memory boards, I/O devices, and currently, work is underway on a floppy disk controller board.

In order to facilitate hardware expansion, and to provide some commonality to the myriad buss and software assignments adopted by Tektron, Netronics, Quest, RCA and home brewists, ACE adopted a standard buss pin assignment and software configuration. Articles in DeFacto and Ipso Facto conform to these standards unless otherwise referenced.

The club has adopted the following 1802 signal outputs:

- 1.79 Mhz clock rate, derived by dividing a 3.58 Mhz colour burst crystal output by two
- Q and EF4 for serial I/O
- Q and EF2 for cassette I/O
- EF1 and Port 1 for 1861 video I/O
- EF3 for handshake, port status
- Port 4 for hex data imput and display

The 44 pin buss adoped by ACE utilized the following pin assignment:

Buss Pin Assignment

1	-+ 5 v.R.	A	-+ 5 v.R
2	-- 12 v.R.	B	-+ 12 v.R
3	- WAIT (RUN)	C	- CPU Clock
4	- CLEAR (LOAD)	D	- DMA IN
5	- Q	E	- DMA OUT
6	- SC1	F	- INT
7	- SC0	H	- MWR
8	- MRD	J	- TPA
9	- D7	K	- TPB
10	- D6	L	- A7
11	- D5	M	- A6
12	- D4	N	- A5
13	- D3	P	- A4
14	- D2	R	- A3
15	- D1	S	- A2
16	- D0	T	- A1
17	- EF3	U	- A0
18	- EF2	V	- N2
19	- EF1	W	- N1
20	- EF4	X	- N0
21	- I/O select	Y	- -5 v.R.
22	- Gnd.	Z	- Gnd.

The 1802 is coming of age, and the articles in DeFacto and Ipso Facto are proof of that maturing. The diversity of articles, reflecting the capabilities and interests of the members are reproduced here for all to benefit, to learn, and to enjoy.

Happy computing!

Michael E. Franklin
Editor/Publisher
23 July 1980