IPSO FACTO

MERRY CHRISTMAS

AND A HAPPY NEW YEAR

ISSUE 15

TABLE OF CONTENTS PAGE

A.C.E. EXECUTIVE COMMITTEE.....2
EDITORIAL COMMENTS.....3
HOW TO BOOT YOUR 1802.....4
AFFILIATED CLUBS.....7
LETTERS TO THE EDITOR.....34,8
THE MACROSUB SUBROUTINE TECHNIQUE.....10
SOME ELFISH IDEAS.....26
THE ELF HAS COME OF AGE.....28
S-100 INTERFACE.....29
REVIEW OF RCA'S UT-4.....31
SOME MISCELLANEOUS THOUGHTS.....35
ERRATA.....39
COSMAC GRAPHICS PROGRAM.....40
LIFE FOR 1802.....46
TINY BASIC CORNER.....51
MORE ON THE 16K DYNAMIC RAM.....53
MORE ON 8-LEVEL INTERRUPTS.....54

Information furnished by IPSO FACTO is believed to be accurate and reliable. However, no responsibility is assumed by IPSO FACTO or the Association of Computer Experimenters for its use; nor for any infringements of patents or other rights of third parties which may result from its use.

Send Newsletter correspondence to:
BERNIE MURPHY
102 McCrany Street
Oakville, Ontario
CANADA L6H 1H6

JANUARY 1980

1979-1980 ACE EXECUTIVE COMMITTEE

PRESIDENT	Ken Bevis	220 Cherry Post Rd. Toronto, ONT., L5A 1H9 416-277-2495
TREASURER	Mike Franklin	24 Duby Rd, Acton, ONT., L7J 2P1 519-853-3421
ASSOCIATE EDITOR	Vic Kushnir	2640 Barnstone Cr., Mississauga, ONT., L5K 2C1 416-822-6505
ASSOCIATE EDITOR	Earle Laycock	2772 Hollington Cres., Mississauga, ONT, L5K 1E7 416-823-1345
ASSOCIATE EDITOR	Bob Silcox	562 Forestwood Cr., Burlington, ONT., L7L 4K3 416-681-2848
CONSULTING EDITOR	Bernie Murphy	102 McCraney St., Oakville, ONT., L6H 1H6 416-845-1630
DRAFTING	J. Myszkowski	99 Augusta St., Hamilton, ONT. 416-529-0250
MEMBERSHIP CO-ORDINATOR	Blair Gerrish	2110 Prospect St., Apt. 3, Burlington, Ont., L7R 1Y9 416-634-0268
TRAINING CO-ORDINATOR	Position Vacant	
HARDWARE CO-ORDINATOR	Fred Feaver	105 Townsend Ave., Burlington, ONT., L7T 1Y8 416-637-2513
PROGRAM CO-ORDINATOR	Fred Pluthero	1013 Upper Wellington St., Hamilton, ONT., L9A 3S4 416-389-4070
PUBLISHING COMMITTEE	Dennis Mildon	44 Wildewood Ave., Hamilton, ONT., L8T 1X3 416-385-0798
	John Hanson	955 Harvey Place, Burlington, Ont. L7T 3E9 416-637-1076

>

Editorial Comments

As this issue marks not only the beginning of a new year but also the start of the 1980's, best wishes go to our membership for both a Happy New Year and New Decade too.

The microcomputer "revolution" is certainly in full swing as more companies come out with new products both in basic systems and support peripherals. CPU manufacturers are upgrading designs and producing chips that can directly access megabytes of memory with a more powerful instruction set. This tremendous growth in the industry and the subsequent availability of so many diverse options has put a lot of people into future shock as far as planning goes. More questions are created than answers provided. As a result this might be an appropriate time to review possible directions to go with your 1802 system.

What is the best use for an 1802? Can it be expanded to the point of being competitive with larger systems currently available? Or is it most efficient as a dedicated processor for a specific function? What are the limits, if any, set by the inherent design of the chip beyond which the 1802 should not be pushed? Does it make sense to plan for interfacing sophisticated peripherals? What are YOUR answers to these questions and how far do you plan to expand your system? We would be interested to hear from you on these issues.

We hope you will continue your experimentation with the 1802 and share your results and ideas with us. We would particularly like to hear from anyone who has done any work an A/D or D/A conversion or interfacing with various types of sensors.

Please keep in mind that anyone who has equipment to sell, swap or give away is welcome to write their own mini-ad, submit it to us, and we'll publish it in a subsequent issue of Ipso Facto. This offer does not apply to commercial ventures although we will accept material concerning new product announcements.

Please note that ACE meetings for February 12th and March 11th will be at Sheridan College in Oakville in Room B123 at 7:30.

You may wonder why Santa Claus is on our front cover and why we seem to have abandoned the unique cover design of Issue 14. Since John had done such an excellent job on the cover design, we said "what the heck. Let's go with it!" After all, we intended to get the issue out by Christmas (of 1979) even if, in the holiday rush, it didn't quite work out that way.

On the news front, good news and bad news. The good is that our next issue (#16) should be out soon. The bad news is that we're out of back issues 7-12. However, within a few months, we hope to be able to offer 'Best of Ipso Facto' (issues 1-12). Watch future issues of Ipso Facto for further news on this point.

>

HOW TO BOOT YOUR 18Ø2 SYSTEM

Tom J. Crawford,
5Ø Brentwood Dr.,
Stoney Creek, Ont.,
Canada. L8G 2W8

One of the first pieces of software most people write or obtain for their computer system is a monitor routine. This allows much easier use of the computer system for operations such as program entry, checking, cassette save, load and etc. For most people this monitor resides in a ROM or RAM in high memory. Therefore it is still necessary to manually key in a long branch instruction at location ØØØØ, and to execute it each time the monitor is to be re-entered.

One way around this extra work is to put the monitor at location ØØØØ. Unfortunately, just about every piece fo software written for the 18Ø2 is designed to start at ØØØØ. It would therefore be impossible to use the monitor while using any of this other software. Thus the requirement to "BOOT" the CPU up to the monitor starting address upon RESET and RUN.

Perhaps the simplest way to do this was explained to me by Harley Shanko in California. Simply force a few of the high address bits (A15, A14) to 1's for a few hundred microseconds after inbitializing the CPU. Thus, even though the CPU is addressing ØØØØ, the rest of the system sees address CØØØ. The first thing your monitor should do is change the PC register value to correct address, so that when the address forcing timer times out, the monitor will continue to run.

This mechanism is very easy to implement with a few OR gates and an RC network, if your system has a de-multiplexed address bus available on the back-plane. If not, then it is still possible to use this method, but with rather unorthodox BOOT addresses. The reason for this is that the high bits of both the high and low bytes of the address will be forced to 1's. For example, if the 2 hi bits of the multiplexed address bus are forced to 1's the boot address is CØCØ.

Figure 1 shows a suitable Boot circuit using this method. Listing 1 is typical of the code which should be entered in your monitor upon Booting your system. Notice that the code first sets up the Program Counter value, then delays in a loop for a while. This loop is fairly important to the operation of the Boot circuit. Since an RC network is used to produce the forcing timing, it is difficult to set the time accurately. The Boot must be present long enough to set up the Program Counter value, but must be gone before the CPU attempts to execute any code requiring a Ø on one of the forced address lines. In the case of booting to CØCØ, this could be simply executing past the end of page CØ to C1ØØ, or attempting to branch back to an address in page CØ below CØCØ. The delay loop is used to ensure that the CPU stays in a suitable address range until the forcing timing expires. For the code of Listing 1 and a 1 Mhz clock, the minimum and maximum force times are about 64 microseconds and 12

milliseconds. Thus the forcing timer is set for about 1 millisecond.

The problem with this Boot mechanism is the restriction it places on the starting address of your monitor. Assuming you want your monitor in the upper 16K of memory, you are pretty well restricted to these 49 addresses: CØCØ, C1C1, C2C2,...,FØFØ. You may not be able to insert the correct code at precisely the right spot in your Monitor; or you may not be in a position to re-work and re-assemble a purchased Monitor. Under these circumstances, it would be nice to be able to Boot to any page boundary address. This is the function of the circuit shown in Figure 2.

Notice the circuit is somewhat more complicated than that of Figure 1. It only requires one trace to be cut on the CPU board, however. And it requires no extra software in your Monitor to support the Boot function, as does the previous circuit. The Boot timing is synchronous; it lasts exactly as long as required, then quits. The Boot mechanism itself works by forcing the CPU to execute a Long Branch (CØ) instruction to the desired page boundary address; hence the PC register is properly set up at the end of the Boot. And the Boot address can be changed at any time via the DIP switch, without any of the hardware changes required of Figure 1.

The circuit operates by switching a ROM chip into the memory address space at ØØØØ whenever a Boot (RESET-->RUN) is requested. This ROM contains a 3 byte Long Branch instruction. The ROM is switched back out after exactly 3 machine cycles, in order to allow the Long Branch to occur. Also, while the ROM is switched in, the MRD line from the CPU is disabled, so that the memory normally mapped into address ØØØØ will not respond during the Boot.

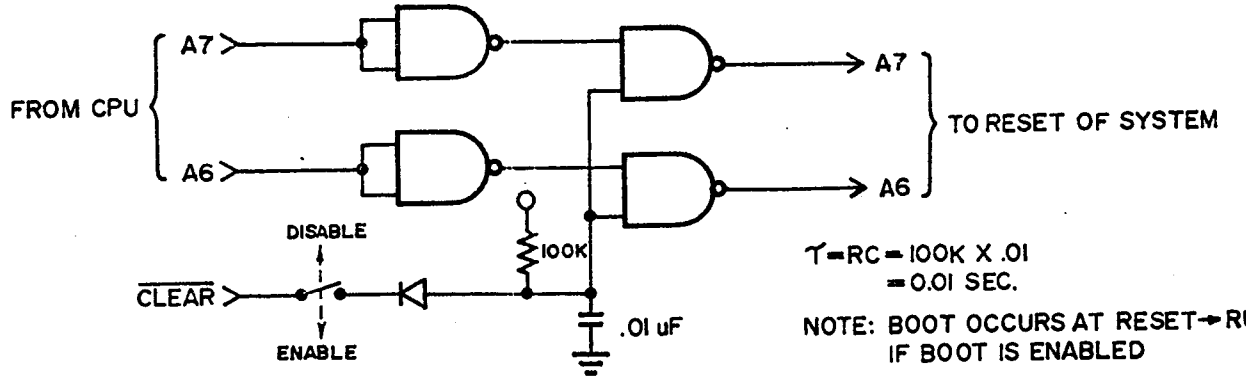
Since only 3 bytes of the 1Ø24 byte ROM are required for a Long Branch instruction, it is convenient to program the ROM with 256 different Long Branches, 1 to each of the 256 different page boundaries. These instructions can be selected with switches connected to the high 8 address lines of the ROM, independently of the rest of the Boot circuit.

One final word; these circuits are untested at the present time. I will let you know the results of building and testing the circuit in Figure 2 in a future issue of IPSO FACTO.

LISTING OF TYPICAL BOOT SOFTWARE (BOOT=CØCØ HEX)

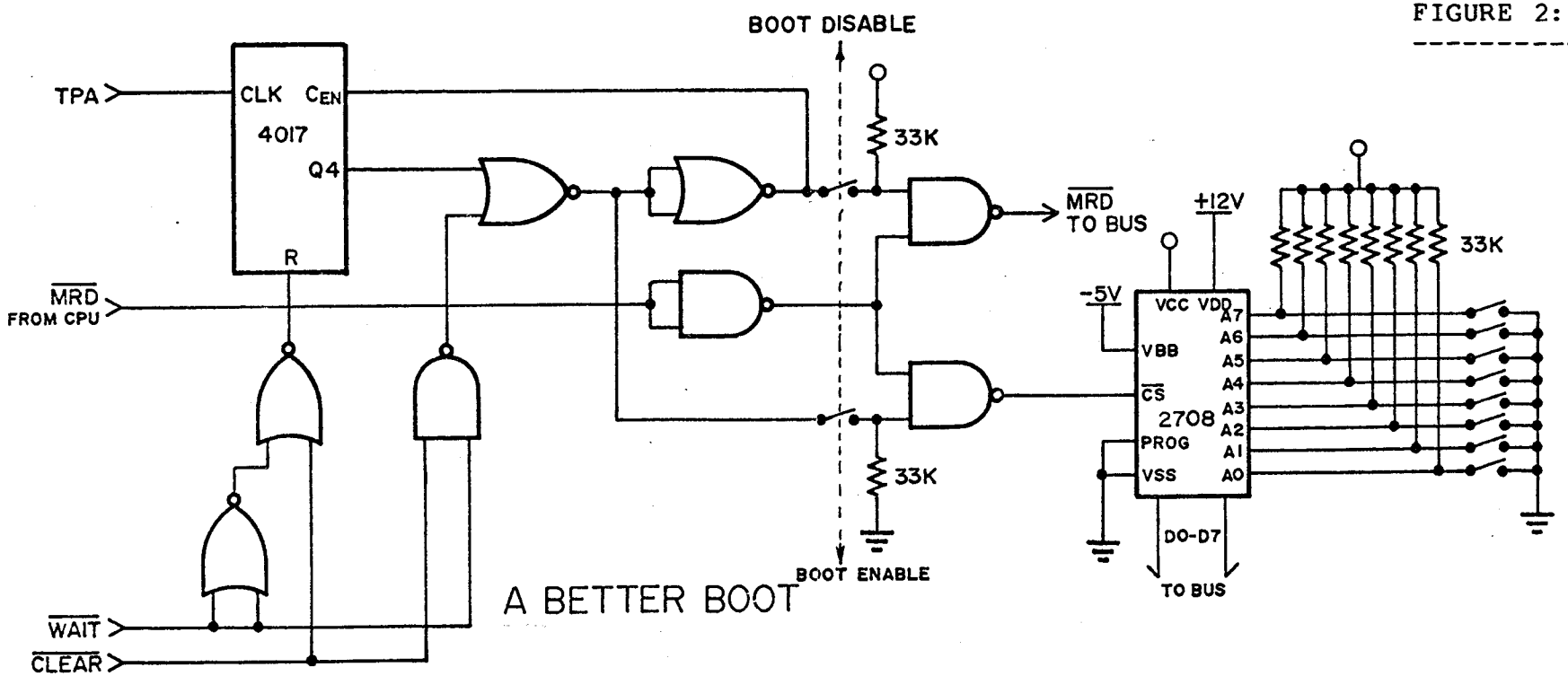
```
C0C0  F8 C0          LDI  INIT.1
C0C2  B0             PHI  R0          ;ENSURE PC IS SET
C0C3  F8 C6          LDI  INIT.0      ;TO TAKE OVER WHEN
C0C5  A0             PLO  R0          ;BOOT TIMER TIMES
C0C6  F8       INIT: PDI #FF          ;OUT.
C0C8  A3             PLO  R3          ;DELAY FOR A WHILE
C0C9  23       INITIO: DEC R3
C0CA  83             GLO  R3
C0CB  3A C9          BNZ  INITIO      ;DONE YET? NO.

                                      ;YES.
```

FIGURE 1:



TYPICAL BOOT HARDWARE (BOOT=M(COCO))

FIGURE 2:



A BETTER BOOT

AFFILIATED CLUBS

In issue 14 we asked people who would be willing to act as the foci for potential affiliated clubs to submit their names, addresses, and phone numbers for publication for this purpose.

There were quite a number who accepted the invitation, and they are listed below. If you would be interested in joining such a club, please contact one of the people listed below if they are in your general geographic area. If there is nobody who lives close to you, then we would be glad to print your name in an upcoming issue so that you can get together with other 1802 enthusiasts in your area.

Lloyd R. Dickinson
Institute of Economic Analysis
44 Montgomery St., 5th Floor
San Francisco, Calif., 94104

Eugene E. Jackson
3637 Snell Ave., #385
San Jose, Calif., 95136
(408) 224-0837

John R. Powers, III
CyberVision, Inc.
6660 Doubletree Ave., Suite 16
Columbus, Ohio, 43229
Phone (800) 848-4377

Michael Olah
2892 Mogadore Rd., Apt. 1-B
Akron, Ohio, 44312
Phone (216) 867-7290

James E. Tate
4122B El Lago Blvd.
Indianapolis, Ind., 46227
Phone (317) 783-1156 (Home)
(317) 545-6631 (Work)

>

FOR SALE

1) Tektron 1802, with 8K (7K RAM + 1K 2708 ROM) - $500 or best offer
- with SWTP TVT (capable of 64 characters per line)
- with numerous software items (including QUEST 12K BASIC)
2) 4116 dynamic RAM chips (350 ns.)
32 (i.e. 16K bytes) available - $8 each

Contact John Rogers
114 Rand Street
Hamilton, Ont. Phone (416) 561-3121

>

John McLaren,
41 Walker St.,
Coorparoo, Queensland,
Australia. 4151

Dear Editor:

In regard to your last article in Ipso #14 regarding "Club Sponsored Hardware Projects", I am writing my response at speed to be with you before the December meeting. Apologies for the scrawl, mistakes and etc...

(i) I agree with the concept as outlined and the several implications you have neatly sidestepped by using the 44 pin buss. Good!

(ii) Many of your members are like me--complete mugs around circuit diagrams. In fact, I don't know which end the bullets come out on a soldering gun! I believe that the level of documentation should be high for all the bits i.e. several orders of magnitude higher than a commercially available kit. Generally speaking, the "doers" of the world aren't the "teachers" of the world so I must assume that documentation won't be acceptable to people of my technical skill. However, I am prepared (in fact, I would be delighted to help) to correct slash enlareg upon slash illustrate the assembly instructions for each board in the course of assembling each prototype.

The correcting/etc. would include spelling and grammar typing of the new instructions, hand illustrations of "bumbler" points (e.g. how to solder an IC socket correctly; how to apply solder, etc.) which are vital knowledge to people like me, sentence construction and idiom evaluation, real HTG (Honest To God) photographs when necessary screened for printing), inclusion of test programs (e.g. how do I tell if the RAM and etc. work HTG); and setup of the whole bag ready for transferring to paper plates (cheap) for printing on an A.B. Dick (or similar) offset printing press.

Response time from date you dispatched my order (I am prepared to buy the boards) would be about five weeks until you received the new new instructions ready for printing.

(iii) It is not obvious to me how you intend all the existing 18Ø2 busses to be interfaced to your buss. I think (and would like to see) that you should supply these details in these ways:

+An article in IPSO FACTO for really poor members.
+Detailed instructions and circuits included in each kit for the ribbon cable/44 pin socket backframe. This is for people on limited budgets (such as me) who want a board developed by ACE but can't afford the full backplane as detailed in IPSO FACTO #14.
+The ACE backplane as a kit.

As regards the backplane, no termination method was discussed in IPSO #14. Do you intend to use it? I hope so and I hope it is active termination. The theory is fully discussed somewhere in the year's issue of Kilobaud (just a minute.....April, '79 has two articles). I seem to have 87, 293, 427.5 phantom signals galloping up and down my ELF II buss and will have to do something about it shortly.

(iv) I have no comments on the pin assignments of the proposed buss--about par with my knowledge of electronics.

(v) Whenever possible, I would like to see program subroutines for the boards e.g. the colour video board, the K.C. tape board, the digital I/O board, ect. This is really for mugs like me who want these types of boards but also need software to use them and have difficulty in developing some!!

(vi) There are several power systems and TV systems used on planet Earth. Please don't forget the other when designing your boards. You have no idea just how frustrating it is to be locked out of using a board because we have the PAL TV system and the boardd is designed for NTSC or because we use the "other" power system.

In regard to out 240VAC, 50 Hz. power system. Please bear in mind that it delivers TWICE the power of 115V 60 Hz. systems with a consequent massive rise in heat dissipation requirements and lowered component life. Please ensure that your boards can carry sufficiently large heatsinks to dissipate the heat generated by the other power system.

Finally, a few points regarding the projects. I think the idea is excellent and I hope my offer of assistance will be of help to you. Unfortunately space (12,000 miles or so) precludes closer co-operation, however I thank you may find the offer attractive--its a boring job and very rarely done well for kits; and it is important.

The boards discussed in IPSO #14 almost form a full micro. Why not to the whole way and develop a CPU board?? and floppies??

As regards the club projects, I would like to see some software oriented projects e.g. a decent monitor; some decent languages e.g. a Microsoft Basic interpreter equivalent, Pascal, Fortran; a strong DOS; graphics. I would be happy to help.

Note: MUGS -noun, plural. Australian for Mainly Useless Gentlepersons

>

Russell H. Dowling, Ph.D.
P.O. Box 900
Seroe Colorado 321
Aruba
Netherlands Antilles

Chester W. Vlaun
P.O. Box 624
Seroe Colorado 1528
Aruba
Netherlands Antilles

The MACROSUB subroutine technique

During the development of our homebrew 1802 one of our concerns was the inefficiency of RCA's SCRT subroutine technique. While other subroutine techniques have been documented elsewhere, none measure up to the flexibility and ease-of-use of the SCRT technique. Towards this end we developed a simple, easy-to-use technique, whose overall efficiency far exceeds that of the SCRT technique. Our goal was a method for the efficient calling of especially short unnested macro-instruction sequences, so from now on we'll refer to it as the "MACROSUB" technique. In what follows we discuss not only how the technique works, but also illustrate its use via some examples. In addition, the basic MACROSUB technique, and its two variations, are compared with several other subroutine calling techniques.

The MACROSUB code had to satisfy several objectives, these being:

1. Minimize the number of inline bytes in the calling program;
2. Keep the call overhead in MACROSUB at a minimum;
3. Keep MACROSUB's return overhead at a minimum;
4. Both the DF and X registers must be undisturbed for optional use by the subroutine;
5. The D-Register (Accumulator) need be saved only when so required;
6. Inline bytes should be easily transferable from calling program to subroutine;
7. The subroutine, or macro should be easy to implement with either a macro-assembler or without one.

Briefly, to introduce the technique, the heart of it is what you FORTRAN buffs know as a (very long) computed go to statement. In other words, a multi-conditional branch is used to access a particular subroutine. This feat is accomplished by assigning a fixed code to each subroutine. This code is then used to access a branch instruction, which transfers program control to a subroutine. To embody the above objectives into a program the following code would arise in the calling program. Register R7 should be dedicated to the technique for the sake of efficiency. Initialization of register R7 to the entry point, i.e., address of MACROSUB need be done only once. The following group of initialization instructions should be located in the same section of code which initializes the SCRT dedicated registers.

```
LDI A.0(MACROSUB)   Initialize R7 to point to the entry point
PLO R7              of MACROSUB
LDI A.1(MACROSUB)
PHI R7
```

Calling any macro-subroutine then requires only two bytes inline in the calling program:

```
SEP  R7          Branch to MACROSUB
BYTE $XY         XY=address of branch to a particular macro
 .  }
 .  }            Inline bytes (if any) to be transferred to
 .  }            the macro-subroutine should follow here
```

We can simplify the above even further by realizing that for a given macro these two bytes are fixed, regardless of where the actual subroutine is located in memory. A more compact, and less error-prone way to implement the technique makes use of the EQU (Equate) and DBYT (Double Byte) assembler directives. Somewhere in the program (or in the assembler's instruction table) define the macro MACR by the following statement:

```
MACR    EQU  D7XY          XY=address of branch to subroutine MACR
```

Every call is replaced by the single statement

```
        DBYT MACR          Execute MACR subroutine
```

If the D-Register has to be saved for any reason whatsoever, then this must be done prior to calling the macro-subroutine. Any of the four standard instructions may be used for this purpose: STXD,STR,PHI,PLO.

The MACROSUB routine makes the assumption that the calling program counter always equals register R3.The following code implementing MACROSUB's conditional branch is then straightforward and fast.

```
 LOW
ADDRESS

    FD MACRORET SEP  R3          Return to calling program with R7 pointing to
                                 MACROSUB
    FE MACROSUB LDA  R3          Get address of branch = first inline byte
    FF          PLO  R7          Effect a branch to an LBR instruction in the
                                 following table of LBR's
    00          LBR  MACROS00    Branch to off-page subroutine MACROS00
    03          LBR  MACROS01    Branch to off-page subroutine MACROS01
    06          LBR  MACROS02    Branch to off-page subroutine MACROS02
     .          .                etc. .
     .          .                .
     .          .                .
    FC          LBR MACROS84     Branch to off-page subroutine MACROS84
    FF          LBR MACROS85     Branch to off-page subroutine MACROS85
    02          .                Any other code
```

In studying the above code, we see how the technique functions. Register R7 has been initialized previously, and it addresses MACROSUB's entry point. The subroutine is entered by changing the calling program counter to register R7. Since the SCRT program counter (the calling program counter) is register R3, execution of the LDA R3 instruction causes the inline byte to be loaded into the D-register. Register R3, having been incremented by the LDA R3 instruction now points to the return point in the calling program (or arguments as the case may be). Of course the previous contents of the D-register is hereby destroyed. In the PLO R7 instruction that follows, the above inline byte is stored into the low order half of the current program counter (R7). This causes a branch to a location on the page containing the branch table. This location's low address is the inline byte, which therefore must point to an LBR instruction. The particular LBR instruction is executed creating an off-page branch to a specific subroutine. Notice that extreme havoc and chaos will most likely be created if the inline byte does not point to an LBR instruction!

The code below for a subroutine, say for example MACRO26, is executed. Note that MACRO26 may be located anywhere within memory, because of the use of the LBR MACRO26 instruction in the branch table. If subroutine MACRO26 expects inline bytes, then these may be transferred by executing one LDA R3 instruction for each inline byte to transfer. Upon completion of the subroutine instruction sequence, an unconditional branch is made to the common return point for all macro-subroutines via the statement LBR MACRORET. The execution of a SEP R3 instruction returns program control back to the calling program. Register R7 at this time points the entry of the MACROSUB subroutine, thus preparing R7 for subsequent calls to MACROSUB.

```
MACRO26  .                  Instructions particular to subroutine
         .                  MACRO26, i.e., the actual task to be
         .                  performed by MACRO26
         LDA  R3            Pick up an inline byte
         .   }
         .   }              More code in MACRO26
         .   }
         LBR MACRORET       Branch to common return point
```

Pay special attention to the address of MACROSUB's PLO R7 statement. Since we are changing the value of the current program counter (R7) with the PLO R7 statement, it is crucial that when this instruction is executed, R7 points to the same page as the LBR instructions table. In other words, the PLO R7 statement must be placed at the location shown (last byte on a page), or on the same page as the LBR instructions table. By locating the PLO R7 instruction at location FF as we did, one more subroutine may be accessed than would otherwise be possible.

A slight increase in speed over the standard version shown above is obtainable. One of the two faster versions available has no penalties associated with it, whilst the the other does. In both cases this involves the substitution of an LBR instruction with a BR instruction.

In "speedy" Version 1, LBR MACRORET is changed to BR MACRORET. This is possible only if the BR MACRORET statement is located on the same page as the SEP R3 instruction. Not only is the code shortened by one byte, but it executes one machine cycle faster than the standard version. We recommend that Version 1 be reserved solely for the very shortest of subroutines. This is by virtue of the fact that all fast (short) routines' code must be "essentially" limited to a single page. We say essentially for the following reason that is best illustrated by the first example below. With the exception of a single "fast" subroutine, all code for the short subroutines should be placed on the same page as SEP R3. Just one subroutine may actually cross this page's boundary as shown. Even though the starting point of this single routine is off the SEP R3 page, what really counts is that the BR MACRORET is on the SEP R3 page. Clearly a mild increase in speed was obtained while no penalties were incurred. Notice from the example that the last fast subroutine need not have the BR MACRORET!

The concept of Version 2 is similar to that of Version 1: LBR MACRORET becomes BR MACRORET. In this case the twist involves deleting LBR MACROAB from the branch table and replacing it with subroutine MACROAB's code. As the second example below shows, the SEP R3, LDA R3 and PLO R7 instructions must now also be on the branch table page. In this fashion the efficiency is improved by 4 bytes and 4 machine cycles. The consequent drawback, however, is that the room available for LBR's in the branch table is greatly reduced. Version 2 merits consideration, and hence is recommended only if: a) time is critical, or b) your application is limited in scope to a few short subroutines and not too many long ones.

Illustration of use of Version 1 of the MACROSUB technique

```
 LOW
ADDRESS
    .  FASTSUB1 NOP              Kill time in subroutine FASTSUB1, which starts
    .           .                on any page previous to the MACRORET SEP  R3
    .           .                statement
    FF          .                One or more page boundaries may be crossed
    00          .
    .           .
    .           BR   MACRORET    Branch to common return point
    .           .
    .           .                Other "fast" subroutines, for example: FASTSUB5
    .           .
    .  FASTSUB5 NOP              Kill time
    .           NOP              Kill more time
    .           BR   MACRORET    Branch to common return point
    .           .                More fast subroutines. Notice that the
    .           .                last fast code subroutine doesn't need BR MACRORET.
    FD MACRORET SEP  R3          Common return point to calling program
    FE MACROSUB LDA  R3          Start of MACROSUB code
    FF          PLO  R7          Branch to branch instruction
    00          .                Long Branch table starts here
    .           .
    .           LBR  FASTSUB5    Off-page branch to subroutine FASTSUB5
    .           .
    .           .
    .           .
    .           LBR  FASTSUB1    Off-page branch to subroutine FASTSUB1
    .           .
    FF          .                End of page with table of branches to subroutines
    .           .
```

Illustration of use of Version 2 of the MACROSUB technique

```
 LOW
ADDRESS
    .  SLOWSUB3 NOP              Kill time in subroutine SLOWSUB3 located anywhere
    .           .                in memory
    .           .
    .           LBR  MACRORET     Branch to common return point
    .           .
    .           .
    .           .
    00 MACRORET SEP  R3          Common return point to calling program
    01 MACROSUB LDA  R3          Start of MACROSUB code
    02          PLO  R7          Branch to branch instruction
    03          .                LBR table starts here.
    .           LBR SLOWSUB3     Branch to off-page subroutine SLOWSUB3
    .           .
    .  FASTSUB1 NOP              Fast subroutine FASTSUB1 code starts here with
    .           .                killing time
    .           BR   MACRORET    Branch to common return point
    .           .
    .           .
    .  FASTSUB8 .                Code for fast subroutine FASTSUB8 starts here
    .           .
    .           BR   MACRORET    Branch to common return point
    FF          .                End of page with branch table and fast subroutines
```

The MACROSUB routine introduced above has the capability of addressing or calling up to 86 individual macro-subroutines. Several valuable subroutines are included in Appendix A as sample cases. Besides meeting our objectives, MACROSUB is characterized by easy-of-use and low overhead in all three variations of the technique. Although straight-code (unnested) subroutines are readily implemented with MACROSUB, nested subroutines are a bit elusive to it. In fact, it is somewhat tedious and awkward, but not impossible, to accomplish multi-level nesting using a fixed program counter. For you brave at heart Appendix B shows how to change program counters to Register R3 as required for the above type of multi-level nesting. Nesting of this nature is not recommended since it defeats the purpose for which MACROSUB was designed, and because there are better ways to do this (e.g., SCRT).

Though the 1802 CPU is endowed with an array of 16 general purpose registers, exactly half of these should be reserved for a (full-blown) micro-computer system. The following table lists the registers' dedicated service.

R0-DMA service pointer
R1-Interrupt program counter
R2-Stack pointer
R3-SCRT program counter
R4-SCRT call routine program counter
R5-SCRT return routine program counter
R6-SCRT return and arguments location pointer
R7-MACROSUB subroutines program counter

As pointed out by others, registers can become scarce at times. None the less we did not consider it such a serious handicap or drawback to dedicate a single register to MACROSUB in view of its many superior qualities.

Finally the table below shows some comparative data from which we may draw a few objective conclusions regarding the advantages, disadvantages and pitfalls of the listed techniques. The RCA 1802 User's Manual lists both the most and least efficient techniques, respectively SEP and SCRT. Wayne Bowdish recently (IPSO FACTO No. 12, pg 22-24, June 1979) published a procedure roughly as efficient as the MACROSUB technique documented in this article. We didn't think it was essential to go into a detailed discussion comparing the various techniques, since the self-explanatory table already does so. A careful examination of the table's contents will suggest which subroutine techniques are best suited for certain applications. We propose the following:

SCRT-Bigger subroutines which must call other subroutines, especially off-page ones;
BOWDISH-Short (on-page) subroutines requiring nesting capabilities;
Standard MACROSUB-Long unnested subroutines;
Version 1 MACROSUB-Short unnested subroutines;
Version 2 MACROSUB-Very short unnested subroutines;
SEP-Very few subroutines, time critical applications.

Table With Comparative Data On Various Subroutine Techniques

	SCRT	BOWDISH	Standard MACROSUB	Version 1 MACROSUB	Version 2 MACROSUB	SEP
	Bytes/Cycles	Bytes/Cycles	Bytes/Cycles	Bytes/Cycles	Bytes/Cycles	Bytes/Cycles
Initialization	12/16	0/0	6/8	6/8	6/8	6/8
*Inline Call Statements	3/2	5/6***	2/2	2/2	2/2	1/2
*Fixed Call Overhead	16/30	0/0	5/7	5/7	2/4	0/0
*Fixed Return Overhead	14/26	2/4	4/5	3/4	3/4	2/2
Stack Bytes	2/-	1/-	0/-	0/-	0/-	0/-
Minimum Call/Return Overhead = Total Of Items Marked With *	36/60 ***	9/10	11/14	10/13	7/10	3/4
Minimum Bykes (Product Of Bytes And Cycles)	2160	90	154	130	70	12
**Registers Required	5	2	2	2	2	2-16***
Relative Overall Inefficiency, %	100.0/100.0	25.00/16.67	30.56/23.33	27.78/21.67	19.44/16.67	8.33/6.67
Relative Bykes Inefficiency, %	100.00	4.17	7.13	6.02	3.24	0.56
Easy-To-Use	YES	YES	YES	YES	YES	YES
Nesting Capability (Unlimited Nesting)	EASY	EASY	DIFFICULT***	DIFFICULT***	DIFFICULT***	DIFFICULT
Number Of Subroutines	LIMITED BY STACK LENGTH	LIMITED BY STACK LENGTH	≤ 86	≤ 86	≤ 85	≤ 15

**Includes program counter, stack pointer, and pointers to subroutines.
***Most significant disadvantage of technique.

APPENDIX B:

MULTI-LEVEL NESTING USING THE MACROSUB TECHNIQUE WITH FIXED SCRT PROGRAM COUNTER

Multi-level nesting would require that the MACROSUB program counter (R7) be returned to the original SCRT program counter (register R3) for each level of nesting. Basically this involves saving the calling program's program counter R3 on the stack, setting up a new SCRT program counter R3 and changing the program counter from R7 to R3 while setting up R7 for future MACROSUB calls. Upon return from the nested subroutine the reverse procedure is taken: setting up the new R7 program counter, changing program counter from the R3 back to R7 and finally restoring the saved value of R3 from the stack. The code looks like the following for macro-subroutine MACROAB.

```
MACROAB   .                    Begin execution of subroutine MACROAB and load
          .                    inline bytes, if any.
          .
          GLO  R3              Save return address of calling program (R3=calling
          STXD                 program counter) on the stack prior to the first
          GHI  R3              nesting. This need be done only once in
          STXD                 subroutine MACROAB.
          .
          .                    More code
          .
          LDI  A.0(PARTA)      Set up fake pointer to continue MACROAB
          PLO  R3              subroutine with R3 as program counter
          LDI  A.1(PARTA)
          PHI  R3
          .
          .                    More code
          .
          LBR  MACRORET        Change program counter to R3 leaving R7 pointing
                               to entry of MACROSUB
PARTA     .                    Continuation of subroutine with R3 as new
          .                    program counter in subroutine MACROAB; execute
          .                    any code including SCRT, MACROSUB, etc. calls.
          .
          LDI  A.0(PARTB)      At end of subroutine MACROAB prepare to change
          PLO  R7              the program counter from R3 back to R7
          LDI  A.1(PARTB)
          PHI  R7
          SEP  R7              Change program counter to R7 (pointing to the
                               following instruction)
PARTB     IRX                  Advance stack pointer
          LDXA                 Restore the original value of register R3 from
          PHI  R3              the stack to prepare for return to program which
          LDX                  called MACROAB
          PLO  R3
          LBR  MACRORET        Branch to common MACROSUB return point
```

```
LOCN OBJ CODE   STMT    SOURCE STATEMENT     1802 VER 1.7

                   1 *
                   2 *       SUBROUTINE ***MACROSUB*** EFFECTS A CALL TO A MACRO-SUBROUTINE
                   3 *
                   4 *       TO CALL MACROSUB REQUIRES THE FOLLOWING STATEMENTS IN THE CALLING
                   5 *       PROGRAM
                   6 *
                   7 *            SEP   R7              BRANCH TO DEDICATED MACRO CALLING SUBROUTINE
                   8 *            DC    #XY             LOW ADDRESS BYTE FOR SPECIFIC MACRO-SUBROUTINE
                   9 *                                  WHERE XY= A HEXADECIMAL BYTE
                  10 *
                  11 *       ENTER WITH...
                  12 *       R3=CALLING PROGRAM COUNTER POINTING TO LOW ADDRESS BYTE
                  13 *       R7=MACRO-SUBROUTINE PROGRAM COUNTER
                  14 *
                  15 *       RETURNS WITH...
                  16 *       R3=RETURN ADDRESS OF CALLING PROGRAM
                  17 *       D-REGISTER=DESTROYED
                  18 *
A3FD              19            ORG   $A3FD
A3FD D3           20 MACRORET   SEP   R3              RETURN VIA SCRT STANDARD PROGRAM COUNTER
A3FE 43           21 MACROSUB   LDA   R3              PICK UP MACRO LOW ADDRESS FROM CALLING PROGRAM
A3FF A7           22            PLO   R7              STORE IN MACRO-SUBROUTINE PROGRAM COUNTER TO
                  23 *                                EFFECT A BRANCH TO APPROPRIATE MACROSUBROUTINE
A400 C0A502       24            LBR   SAVREG8B        SAVE REGISTERS 8-B ON STACK
A403 C0A515       25            LBR   RSTREG8B        RESTORE REGISTERS 8-B FROM STACK
A406 C0A529       26            LBR   SAVREGCF        SAVE REGISTERS C-F ON STACK
A409 C0A53C       27            LBR   RSTREGCF        RESTORE REGISTERS C-F FROM STACK
A40C C0A550       28            LBR   SAVREG8F        SAVE REGISTERS 8-F ON STACK
A40F C0A573       29            LBR   RSTREG8F        RESTORE REGISTERS 8-F FROM STACK
A412 C0A597       30            LBR   MOVSBLOK        MOVE SHORT MEMORY BLOCK
A415 C0A5A2       31            LBR   MOVLBLOK        MOVE LONG MEMORY BLOCK
A418 C0A5BD       32            LBR   DUPSBLOK        DUPLICATE A BYTE IN A SHORT MEMORY BLOCK
A41B C0A5BB       33            LBR   DUPLBLOK        DUPLICATE A BYTE IN A LONG MEMORY BLOCK
A41E C0A5C9       34            LBR   MULTIBR         MULTIPLE CONDITIONAL SHORT BRANCH
A421 C0A5D1       35            LBR   MULTILBR        MULTIPLE CONDITIONAL LONG BRANCH
A424 C0A5E6       36            LBR   HEXASCII        CONVERT HEXADECIMAL BYTE TO ASCII


LOCN OBJ CODE   STMT    SOURCE STATEMENT     1802 VER 1.7

A502              37            ORG   $A502
                  38 *       MACRO-SUBROUTINE ***SAVREG8B*** SAVES THE CONTENTS OF REGISTERS 8
                  39 *       THROUGH B ON A STACK
                  40 *
                  41 *       ENTER WITH...
                  42 *       RX=ADDRESS OF FREE LOCATION ON THE BOTTOM OF THE STACK
                  43 *
                  44 *       RETURNS WITH...
                  45 *       RX=ADDRESS OF FREE LOCATION ON THE TOP OF THE STACK
                  46 *       R8,R9,RA,RB=UNCHANGED AND SAVED ON THE STACK
                  47 *
                  48 *       CAUTION: RX SHOULD NOT BE ONE OF R8-R9
                  49 *
                  50 *       EXECUTION TIME=46 MACHINE CYCLES
                  51 *
A502 8B           52 SAVREG8B   GLO   RB              GET LSB
A503 73           53            STXD
A504 9B           54            GHI   RB              GET MSB
A505 73           55            STXD                  SAVE REGISTER B ON THE STACK
A506 8A           56            GLO   RA              GET LSB
A507 73           57            STXD
A508 9A           58            GHI   RA              GET MSB
A509 73           59            STXD                  SAVE REGISTER A ON THE STACK
A50A 89           60            GLO   R9              GET LSB
A50B 73           61            STXD
A50C 99           62            GHI   R9              GET MSB
A50D 73           63            STXD                  SAVE REGISTER 9 ON THE STACK
A50E 88           64            GLO   R8              GET LSB
A50F 73           65            STXD
A510 98           66            GHI   R8              GET MSB
A511 73           67            STXD                  SAVE REGISTER 8 IN THE STACK
A512 C0A3FD       68            LBR   MACRORET        RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE   STMT    SOURCE STATEMENT    1802 VER 1.7

                  69 *        MACRO-SUBROUTINE ***RSTREG8B*** RESTORES THE CONTENTS OF REGISTERS
                  70 *        THROUGH 8, WHICH WERE SAVED ON A STACK
                  71 *
                  72 *        ENTER WITH...
                  73 *        RX=ADDRESS OF FREE LOCATION ON THE TOP OF THE STACK
                  74 *
                  75 *        RETURNS WITH...
                  76 *        RX=ADDRESS OF (LAST) FREE LOCATION ON THE BOTTOM OF THE STACK
                  77 *        R8,R9,RA,RB=CONTENTS OF THE STACK
                  78 *
                  79 *        CAUTION: RX SHOULD NOT BE ONE OF R8-RB
                  80 *
                  81 *        EXECUTION TIME=48 MACHINE CYCLES
                  82 *
A515 60           83 RSTREG8B IRX                 POINT REG. X TO FIRST BYTE ON THE STACK
A516 72           84          LDXA                GET SAVED REGISTER 8 FROM THE STACK
A517 B8           85          PHI   R8            STORE MSB
A518 72           86          LDXA
A519 A8           87          PLO   R8            STORE LSB
A51A 72           88          LDXA                GET SAVED REGISTER 9 FROM THE STACK
A51B B9           89          PHI   R9            STORE MSB
A51C 72           90          LDXA
A51D A9           91          PLO   R9            STORE LSB
A51E 72           92          LDXA                GET SAVED REGISTER A FROM THE STACK
A51F BA           93          PHI   RA            STORE MSB
A520 72           94          LDXA
A521 AA           95          PLO   RA            STORE LSB
A522 72           96          LDXA                GET SAVED REGISTER B FROM THE STACK
A523 BB           97          PHI   RB            STORE MSB
A524 F0           98          LDX                 LEAVE RX POINTING TO LAST FREE LOCN. ON STACK
A525 AB           99          PLO   RB            STORE LSB
A526 C0A3FD      100          LBR   MACRORET      RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE   STMT    SOURCE STATEMENT    1802 VER 1.7

                 101 *        MACRO-SUBROUTINE ***SAVREGCF*** SAVES THE CONTENTS OF REGISTERS C
                 102 *        THROUGH F ON A STACK
                 103 *
                 104 *        ENTER WITH...
                 105 *        RX=ADDRESS OF FREE LOCATION ON THE BOTTOM OF THE STACK
                 106 *
                 107 *        RETURNS WITH...
                 108 *        RX=ADDRESS OF FREE LOCATION ON THE TOP OF THE STACK
                 109 *        RC,RD,RE,RF=UNCHANGED AND SAVED ON THE STACK
                 110 *
                 111 *        CAUTION: RX SHOULD NOT BE ONE OF RC-RF
                 112 *
                 113 *        EXECUTION TIME= 46 MACHINE CYCLES
                 114 *
A529 8F          115 SAVREGCF GLO   RF            GET LSB
A52A 73          116          STXD
A52B 9F          117          GHI   RF            GET MSB
A52C 73          118          STXD                SAVE REGISTER F ON THE STACK
A52D 8E          119          GLO   RE            GET LSB
A52E 73          120          STXD
A52F 9E          121          GHI   RE            GET MSB
A530 73          122          STXD                SAVE REGISTER E ON THE STACK
A531 8D          123          GLO   RD            GET LSB
A532 73          124          STXD
A533 9D          125          GHI   RD            GET MSB
A534 73          126          STXD                SAVE REGISTER D ON THE STACK
A535 8C          127          GLO   RC            GET LSB
A536 73          128          STXD
A537 9C          129          GHI   RC            GET MSB
A538 73          130          STXD                SAVE REGISTER C ON THE STACK
A539 C0A3FD      131          LBR   MACRORET      RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CO    STMT    SOURCE STATEMENT.    1802 VER 1.7

                132 *       MACRO-SUBROUTINE ***RSTREGCF*** RESTORES THE CONTENTS OF REGISTERS C
                133 *       THROUGH F, WHICH WERE SAVED ON A STACK
                134 *
                135 *       ENTER WITH...
                136 *       RX=ADDRESS OF FREE LOCATION ON THE TOP OF THE STACK
                137 *
                138 *       RETURNS WITH...
                139 *       RX=ADDRESS OF (LAST) FREE LOCATION ON THE BOTTOM OF THE STACK
                140 *       RC,RD,RE,RF= CONTENTS OF THE STACK
                141 *
                142 *       CAUTION: RX SHOULD NOT BE ONE OF RC-RF
                143 *
                144 *       EXECUTION TIME= 48 MACHINE CYCLES
                145 *
A53C 60         146 RSTREGCF IRX                 POINT REG. X TO FIRST BYTE ON THE STACK
A53D 72         147          LDXA                GET SAVED REGISTER C FROM THE STACK
A53E BC         148          PHI   RC            STORE MSB
A53F 72         149          LDXA
A540 AC         150          PLO   RC            STORE LSB
A541 72         151          LDXA                GET SAVED REGISTER D FROM THE STACK
A542 BD         152          PHI   RD            STORE MSB
A543 72         153          LDXA
A544 AD         154          PLO   RD            STORE LSB
A545 72         155          LDXA                GET SAVED REGISTER E FROM THE STACK
A546 BE         156          PHI   RE            STORE MSB
A547 72         157          LDXA
A548 AE         158          PLO   RE            STORE LSB
A549 72         159          LDXA                GET SAVED REGISTER F FROM THE STACK
A54A BF         160          PHI   RF            STORE MSB
A54B F0         161          LDX                 LEAVE RX POINTING TO LAST FREE LOCN. ON STACK
A54C AF         162          PLO   RF            STORE LSB
A54D C0A3FD     163          LBR   MACRORET      RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT    1802 VER 1.7

                164 *       MACRO-SUBROUTINE ***SAVREGBF*** SAVES THE CONTENTS OF REGISTERS 8
                165 *       THROUGH F ON A STACK
                166 *
                167 *       ENTER WITH...
                168 *       RX=ADDRESS OF FREE LOCATION ON THE BOTTOM OF THE STACK
                169 *
                170 *       RETURNS WITH...
                171 *       RX=ADDRESS OF FREE LOCATION ON THE TOP OF THE STACK
                172 *       R8,R9,RA,RB,RC,RD,RE,RF=UNCHANGED AND SAVED ON THE STACK
                173 *
                174 *       CAUTION: RX SHOULD NOT BE ONE OF R8-RF
                175 *
                176 *       EXECUTION TIME=78 MACHINE CYCLES
                177 *
A550 8F         178 SAVREGBF GLO   RF              GET LSB
A551 73         179          STXD
A552 9F         180          GHI   RF              GET MSB
A553 73         181          STXD                  SAVE REGISTER F ON THE STACK
A554 8E         182          GLO   RE              GET LSB
A555 73         183          STXD
A556 9E         184          GHI   RE              GET MSB
A557 73         185          STXD                  SAVE REGISTER E ON THE STACK
A558 8D         186          GLO   RD              GET LSB
A559 73         187          STXD
A55A 9D         188          GHI   RD              GET MSB
A55B 73         189          STXD                  SAVE REGISTER D ON THE STACK
A55C 8C         190          GLO   RC              GET LSB
A55D 73         191          STXD
A55E 9C         192          GHI   RC              GET MSB
A55F 73         193          STXD                  SAVE REGISTER C ON THE STACK
A560 8B         194          GLO   RB              GET LSB
A561 73         195          STXD
A562 9B         196          GHI   RB              GET MSB
A563 73         197          STXD                  SAVE REGISTER B ON THE STACK
```

```
A564 8A         198         GLO   RA          GET LSB
A565 73         199         STXD
A566 9A         200         GHI   RA          GET MSB
A567 73         201         STXD              SAVE REGISTER A ON THE STACK
A568 89         202         GLO   R9          GET LSB
A569 73         203         STXD
A56A 99         204         GHI   R9          GET MSB
A56B 73         205         STXD              SAVE REGISTER 9 ON THE STACK
A56C 88         206         GLO   R8          GET LSB
A56D 73         207         STXD
A56E 98         208         GHI   R8          GET MSB
A56F 73         209         STXD              SAVE REGISTER 8 ON THE STACK
A570 C0A3FD     210         LBR   MACRORET    RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE   TMT    SOURCE STATEMENT    1802 VER 1.7

                211 *       MACRO-SUBROUTINE ***RSTREG8F*** RESTORES THE CONTENTS OF REGISTERS 8
                212 *       THROUGH F, WHICH WERE SAVED ON A STACK
                213 *
                214 *       ENTER WITH...
                215 *       RX=ADDRESS OF FREE LOCATION ON THE TOP OF THE STACK
                216 *
                217 *       RETURNS WITH...
                218 *       RX=ADDRESS OF (LAST) FREE LOCATION ON THE BOTTOM OF THE STACK
                219 *       R8,R9,RA,RB,RC,RD,RE,RF= CONTENTS OF THE STACK
                220 *
                221 *       CAUTION: RX SHOULD NOT BE ONE OF R8-RF
                222 *
                223 *       EXECUTION TIME= 80 MACHINE CYCLES
                224 *
A573 60         225 RSTREG8F IRX                POINT REG. X TO FIRST BYTE ON THE STACK
A574 72         226          LDXA               GET SAVED REG. 8 FROM THE STACK
A575 B8         227          PHI   R8           STORE MSB
A576 72         228          LDXA
A577 A8         229          PLO   R8           STORE LSB
A578 72         230          LDXA               GET SAVED REG. 9 FROM THE STACK
A579 B9         231          PHI   R9           STORE MSB
A57A 72         232          LDXA
A57B A9         233          PLO   R9           STORE LSB
A57C 72         234          LDXA               GET SAVED REGISTER A FROM THE STACK
A57D BA         235          PHI   RA           STORE MSB
A57E 72         236          LDXA
A57F AA         237          PLO   RA           STORE LSB
A580 72         238          LDXA               GET SAVED REGISTER B FROM THE STACK
A581 BB         239          PHI   RB           STORE MSB
A582 72         240          LDXA
A583 AB         241          PLO   RB           STORE LSB
A584 72         242          LDXA               GET SAVED REGISTER C FROM THE STACK
A585 BC         243          PHI   RC           STORE MSB
A586 72         244          LDXA
A587 AC         245          PLO   RC           STORE LSB
A588 72         246          LDXA               GET SAVED REGISTER D FROM THE STACK
A589 BD         247          PHI   RD           STORE MSB
A58A 72         248          LDXA
A58B AD         249          PLO   RD           STORE LSB
A58C 72         250          LDXA               GET SAVED REGISTER E FROM THE STACK
A58D BE         251          PHI   RE           STORE MSB
A58E 72         252          LDXA
A58F AE         253          PLO   RE           STORE LSB
A590 72         254          LDXA               GET SAVED REGISTER F FROM THE STACK
A591 BF         255          PHI   RF           STORE MSB
A592 F0         256          LDX                LEAVE RX POINTING TO LAST FREE LOCN. ON STACK
A593 AF         257          PLO   RF           STORE LSB
A594 C0A3FD     258          LBR   MACRORET     RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT     1802 VER 1.7

                259 *         MACRO-SUBROUTINE ***MOVSBLOK*** MOVES (TRANSFERS) A SHORT BLOCK OF
                260 *         MEMORY FROM ONE LOCATION TO ANOTHER LOCATION IN MEMORY
                261 *
                262 *         ENTER WITH...
                263 *         LSB R9=NUMBER OF BYTES TO BE TRANSFERRED MINUS 1
                264 *         (MINIMUM NO. OF BYTES=1,MAXIMUM NO. OF BYTES=256)
                265 *         RA=STARTING ADDRESS OF MEMORY BLOCK TO TRANSFER INTO
                266 *         RX=STARTING ADDRESS OF MEMORY BLOCK TO BE TRANSFERRED
                267 *
                268 *         RETURNS WITH...
                269 *         LSB R9=0
                270 *         MSB R9=UNCHANGED
                271 *         RA=ADDRESS OF FIRST BYTE FOLLOWING MEMORY BLOCK TO TRANSFER INTO
                272 *         RX=ADDRESS OF FIRST BYTE FOLLOWING MEMORY BLOCK TO BE TRANSFERRED
                273 *
                274 *         EXECUTION TIME=16+2N MACHINE CYCLES
                275 *               WHERE N=NUMBER OF BYTES TO BE TRANSFERRED
                276 *
A597 72         277 MOVSBLOK  LDXA                  GET BYTE TO TRANSFER, POINT TO NEXT BYTE
A598 5A         278           STR   RA              TRANSFER TO NEW LOCATION
A599 1A         279           INC   RA              POINT TO NEXT LOCATION TO TRANSFER INTO
A59A 89         280           GLO   R9              GET BYTE COUNT
A59B 29         281           DEC   R9              DECREMENT BYTE COUNT
A59C 3A97       282           BNZ   MOVSBLOK        IF BYTE COUNT NON-ZERO MORE BYTES TO TRANSFER
A59E 19         283           INC   R9              INCREMENT BYTE SO MSB R9 REMAINS UNCHANGED
A59F C0A3FD     284           LBR   MACRORET        RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT     1802 VER 1.7

                285 *         MACRO-SUBROUTINE ***MOVLBLOK*** MOVES (TRANSFERS) A LONG BLOCK OF
                286 *         MEMORY FROM ONE LOCATION TO ANOTHER LOCATION IN MEMORY
                287 *
                288 *         ENTER WITH...
                289 *         R9=NUMBER OF BYTES TO BE TRANSFERRED MINUS 1
                290 *               (MINIMUM NO. OF BYTES=1,MAXIMUM NO. OF BYTES=65536)
                291 *         RA=STARTING ADDRESS OF MEMORY BLOCK TO TRANSFER INTO
                292 *         RX=STARTING ADDRESS OF MEMORY BLOCK TO BE TRANSFERRED
                293 *
                294 *         RETURNS WITH...
                295 *         R9=0
                296 *         RA=ADDRESS OF FIRST BYTE FOLLOWING MEMORY BLOCK TO TRANSFER INTO
                297 *         RX=ADDRESS OF FIRST BYTE FOLLOWING MEMORY BLOCK TO BE TRANSFERRED
                298 *
                299 *         EXECUTION TIME=16+12L+1076M MACHINE CYCLES
                300 *               WHERE L=LSB R9, AND M=MSB R9
                301 *
A5A2 1C         302 MOVLBLOK  INC   RC              INCREMENT BYTE COUNT TO NUMBER OF BYTES
A5A3 72         303 MOVLBL20  LDXA                  GET BYTE TO TRANSFER, POINT TO NEXT BYTE
A5A4 5A         304           STR   RA              TRANSFER TO NEW LOCATION
A5A5 1A         305           INC   RA              POINT TO NEXT LOCATION TO TRANSFER INTO
A5A6 29         306           DEC   R9              DECREMENT BYTE COUNT
A5A7 89         307           GLO   R9              GET LSB OF BYTE COUNT
A5A8 3AA3       308           BNZ   MOVLBL20        IF NON-ZERO MORE BYTES TO TRANSFER
A5AA 99         309           GHI   R9              GET MSB OF BYTE COUNT
A5AB 3AA3       310           BNZ   MOVLBL20        IF NON-ZERO MORE BYTES TO TRANSFER
A5AD C0A3FD     311           LBR   MACRORET        RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT      1802 VER 1.7

                312 *         MACRO-SUBROUTINE ***DUPSBLOK*** DUPLICATES A BYTE ON A STACK INTO A
                313 *         SHORT BLOCK OF MEMORY
                314 *
                315 *         ENTER WITH...
                316 *         LSB R9=NUMBER OF BYTES TO BE DUPLICATED MINUS 1
                317 *              (MINIMUM NO. OF BYTES=1, MAXIMUM NO. OF BYTES=256)
                318 *         RA=STARTING ADDRESS OF MEMORY BLOCK INTO WHICH TO STORE DUPLICATE BYTES
                319 *         RX=ADDRESS OF BYTE TO DUPLICATE
                320 *         M(RX)=BYTE TO DUPLICATE
                321 *
                322 *         RETURNS WITH...
                323 *         LSB R9=0
                324 *         MSB R9=UNCHANGED
                325 *         RA=ADDRESS OF FIRST BYTE FOLLOWING MEMORY BLOCK IN WHICH TO STORE
                326 *              DUPLICATE BYTES
                327 *         RX=UNCHANGED
                328 *         M(RX)=UNCHANGED
                329 *
                330 *         EXECUTION TIME=16+12N MACHINE CYCLES
                331 *              WHERE N=NUMBER OF BYTES TO BE DUPLICATED
                332 *
A5B0 F0         333 DUPSBLOK LDX                    GET DUPLICATE BYTE FROM STACK
A5B1 5A         334          STR   RA               TRANSFER TO NEW LOCATION
A5B2 1A         335          INC   RA               POINT TO NEXT LOCATION TO TRANSFER INTO
A5B3 89         336          GLO   R9              GET BYTE COUNT
A5B4 29         337          DEC   R9              DECREMENT BYTE COUNT
A5B5 3AB0       338          BNZ   DUPSBLOK        IF NON-ZERO MORE BYTES TO TRANSFER
A5B7 19         339          INC   R9              INCREMENT BYTE COUNT LEAVING MSB R9 UNCHANGED
A5B8 C0A3FD     340          LBR   MACRORET        RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT      1802 VER 1.7

                341 *         MACRO-SUBROUTINE ***DUPLBLOK*** DUPLICATES A BYTE ON A STACK INTO A
                342 *         LONG BLOCK OF MEMORY
                343 *
                344 *         ENTER WITH...
                345 *         R9=NUMBER OF BYTES TO BE TRANSFERRED MINUS 1
                346 *              (MINIMUM NO. OF BYTES=1, MAXIMUM NO. OF BYTES=65536)
                347 *         RA=STARTING ADDRESS OF MEMORY BLOCK INTO WHICH TO STORE DUPLICATE BYTES
                348 *         RX=ADDRESS OF BYTE TO DUPLICATE
                349 *         M(RX)=BYTE TO DUPLICATE
                350 *
                351 *         RETURNS WITH...
                352 *         R9=0
                353 *         RA=ADDRESS OF FIRST BYTE FOLLOWING MEMORY BLOCK IN WHICH TO STORE
                354 *              DUPLICATE BYTES
                355 *         RX=UNCHANGED
                356 *         M(RX)=UNCHANGED
                357 *
                358 *         EXECUTION TIME=16+12L+1076M MACHINE CYCLES
                359 *              WHERE L=LSB R9, M=MSB R9
                360 *
A5B8 19         361 DUPLBLOK INC   R9               INCREMENT  BYTE COUNT TO NUMBER OF BYTES
A5BC F0         362 DUPLBL20 LDX                    GET DUPLICATE BYTE FROM STACK
A5BD 5A         363          STR   RA               TRANSFER TO NEW LOCATION IN BLOCK
A5BE 1A         364          INC   RA               POINT TO NEXT LOCATION TO TRANSFER INTO
A5BF 29         365          DEC   R9              DECREMENT BYTE COUNT
A5C0 89         366          GLO   R9              GET LSB BYTE COUNT
A5C1 3ABC       367          BNZ   DUPLBL20        IF NON-ZERO MORE BYTES TO TRANSFER
A5C3 99         368          GHI   R9              GET MSB OF BYTE COUNT
A5C4 3ABC       369          BNZ   DUPLBL20        IF NON-ZERO MORE BYTES TO TRANSFER
A5C6 C0A3FD     370          LBR   MACRORET        RETURN VIA COMMON MACRO-SUBROUTINE
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT      1802 VER 1.7

               371 *        MACRO-SUBROUTINE ***MULTIBR*** EFFECTS A MULTIPLE CONDITIONAL SHORT
               372 *        BRANCH (I.E., AN ON-PAGE COMPUTED GO TO) DEPENDING ON A VALUE STORED
               373 *        IN MEMORY
               374 *
               375 *        ENTER WITH...
               376 *        R3=CALLING PROGRAM COUNTER POINTING TO THE TABLE WITH ON-PAGE
               377 *        ADDRESSES TO WHICH THE MULTIPLE CONDITIONAL BRANCH MUST BE MADE
               378 *        THE ADDRESS TABLE MUST BE AS FOLLOWS:
               379 *        LSB OF FIRST ON-PAGE BRANCH ADDRESS FOR WHICH M(RX)=0
               380 *        LSB OF SECOND ON-PAGE BRANCH ADDRESS FOR WHICH M(RX)=1
               381 *        ETC.
               382 *        RX=ADDRESS OF VALUE OF VARIABLE DEPENDING ON WHICH TO SHORT BRANCH
               383 *        M(RX)=VALUE OF VARIABLE
               384 *
               385 *        RETURNS WITH...
               386 *        RX=UNCHANGED
               387 *        R3=BRANCH ADDRESS
               388 *        D,DF=DESTROYED
               389 *
               390 *        CAUTION: THE ENTIRE CODE FOR CALLING MULTIBR (INCLUDING THE BRANCH
               391 *              TABLE) MUST BE LOCATED ON THE SAME PAGE.
               392 *
               393 *        EXECUTION TIME=24 MACHINE CYCLES
               394 *
A5C9 83        395 MULTIBR  GLO   R3          GET LSB OF SCRT PROGRAM COUNTER
A5CA F4        396          ADD               ADD DISPLACEMENT IN ADDRESS TABLE
A5CB A3        397          PLO   R3          SAVE AS LSB OF ADDRESS IN ADRESS TABLE
A5CC C3        398          LDN   R3          PICK UP LSB OF BRANCH ADDRESS
A5CD A3        399          PLO   R3          STORE AS NEW SCRT PROGRAM COUNTER
A5CE C0A3FD    400          LBR   MACRORET    RETURN VIA COMMON MACRO-SUBROUTINE RETURN


LOCN OBJ CODE  STMT    SOURCE STATEMENT      1802 VER 1.7

               401 *        MACRO-SUBROUTINE ***MULTILBR*** EFFECTS A MULTIPLE CONDITIONAL LONG
               402 *        BRANCH (I.E., A COMPUTED GO TO) DEPENDING ON A VALUE STORED IN MEMORY
               403 *
               404 *        ENTER WITH...
               405 *        R3=CALLING PROGRAM COUNTER POINTING TO THE TABLE WITH ADDRESSES TO
               406 *        WHICH THE MULTIPLE CONDITIONAL LONG BRANCH MUST BE MADE
               407 *        THE ADDRESS TABLE MUST BE AS FOLLOWS:
               408 *        MSB OF FIRST BRANCH ADDRESS FOR WHICH M(RX)=0
               409 *        LSB OF FIRST BRANCH ADDRESS
               410 *        MSB OF SECOND BRANCH ADDRESS FOR WHICH M(RX)=1
               411 *        LSB OF SECOND BRANCH ADDRESS
               412 *        ETC.
               413 *        RX=ADDRESS OF VALUE OF VARIABLE DEPENDING ON WHICH TO LONG BRANCH
               414 *        M(RX)=VALUE OF VARIABLE. IF THE VALUE IS GREATER THAN 127, THEN 128 IS
               415 *              SUBTRACTED FROM THE VALUE
               416 *
               417 *        RETURNS WITH...
               418 *        RX=UNCHANGED
               419 *        R3=BRANCH ADDRESS (I.E., NEW CALLING PROGRAM COUNTER)
               420 *        D,DF,M(RX)=DESTROYED
               421 *
               422 *        EXECUTION TIME= 48 MACHINE CYCLES
               423 *
A5D1 F0        424 MULTILBR LDX               PICK UP VALUE OF VARIABLE
A5D2 FE        425          SHL               MULTIPLY BY TWO FOR ADDRESS TABLE DISPLACEMENT
A5D3 73        426          STXD              STORE INTO SAME LOCATION
A5D4 60        427          IRX               POINT TO VARIABLE
A5D5 83        428          GLO   R3          GET LSB OF SCRT PROGRAM COUNTER
A5D6 F4        429          ADD               ADD DISPLACEMENT IN ADDRESS TABLE
A5D7 A3        430          PLO   R3          SAVE AS LSB OF ADDRESS IN ADDRESS TABLE
A5D8 93        431          GHI   R3          GET MSB OF SCRT PROGRAM COUNTER
A5D9 7C00      432          ADCI  0           ADD IN CARRY BIT
A5DB B3        433          PHI   R3          SAVE AS MSB OF ADDRESS IN ADDRESS TABLE
A5DC 43        434          LDA   R3          PICK UP MSB OF BRANCH ADDRESS
A5DD 73        435          STXD              STORE IN LOCATION OF VARIABLE
A5DE 60        436          IRX               POINT X-REG. BACK TO VARIABLE
A5DF C3        437          LDN   R3          PICK UP LSB OF BRANCH ADDRESS
A5E0 A3        438          PLO   R3          SAVE AS LSB OF NEW SCRT PROGRAM COUNTER
A5E1 F0        439          LDX               PICK UP MSB FROM STACK
A5E2 B3        440          PHI   R3          SAVE AS MSB OF NEW SCRT PROGRAM COUNTER
A5E3 C0A3FD    441          LBR   MACRORET    RETURN VIA COMMON MACRO-SUBROUTINE RETURN
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT     1802 VER 1.7

                442 *       SUBROUTINE ***HEXASCII*** CONVERTS A HEXADECIMAL VALUE TO ITS
                443 *       EQUIVALENT ASCII CODE
                444 *
                445 *       ENTER WITH...
                446 *       RX=REGISTER POINTING TO THE ADDRESS OF THE BYTE TO CONVERT
                447 *       M(RX)=BYTE TO CONVERT FROM HEXADECIMAL TO ASCII
                448 *
                449 *       RETURNS WITH...
                450 *       RX=DECREMENTED BY 1
                451 *       M(RX)=BYTE CONVERTED TO ASCII CODE ACCORDING TO THE FOLLOWING TABLE:
                452 *       HEXADECIMAL VALUE     ASCII CODE
                453 *       $00-$09               $30-$39
                454 *       $0A-$0F               $41-$46
                455 *
                456 *       CAUTION: NO CHECK IS MADE TO SEE IF M(RX) IS A VALID HEXADECIMAL
                457 *       VALUE (0-F IS VALID)
                458 *
                459 *       EXECUTION TIME: 27 MACHINE CYCLES FOR VALUES BETWEEN $00-$09, OR
                460 *                       29 MACHINE CYCLES FOR VALUES BETWEEN $0A-$0F
                461 *
A5E6 F0         462 HEXASCII LDX                   LOAD BYTE TO CONVERT VIA X-REGISTER
A5E7 FD09       463          SDI   9               IF GREATER THAN 9 THEN DF=1
A5E9 F830       464          LDI   $30             LOAD ASCII DISPLACEMENT FOR DECIMAL BYTE
A5EB CF         465          LSDF                  IF DF=1 THEN BYTE BETWEEN 0-9
A5EC F837       466          LDI   $37             LOAD ASCII DISPLACEMENT FOR HEXADECIMAL BYTES A-F
A5EE F4         467          ADD                   ADD BYTE TO ASCII DISPLACEMENT
A5EF 73         468          STXD                  STORE ASCII EQUIVALENT OF BYTE,POINT TO FREE LOCN.
A5F0 C0A3FD     469          LBR   MACRORET        RETURN VIA COMMON MACRO-SUBROUTINE
```

Following is an example program, using the techniques and sub-routines discussed above.

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT     1802 VER 1.7

                470 *                          APPENDIX A
                471 *
                472 *
                473 *
                474 *       THE FOLLOWING PROGRAM ILLUSTRATES AN EXAMPLE OF HOW TO USE THE MACROSUB
                475 *       SUBROUTINE TECHNIQUE AND SOME OF THE SUBROUTINES INCLUDED IN THIS
                476 *       APPENDIX.
                477 *
                478 *
                479 *       ASSUME THE SYSTEM HAS ALREADY BEEN INITIALIZED WITH REGISTERS R0-R7
                480 *       POINTING TO ADDRESSES ACCORDING TO THE TABLE WITH THESE REGISTERS'
                481 *       DEDICATED SERVICES (SEE PAGE 5 OF MACROSUB ARTICLE)
                482 *
0000            483          ORG   $0000
0000 F8F2       484          LDI   $F2             LOAD DUMMY VALUE $F29A INTO
0002 B8         485          PHI   R8              REGISTER 8
0003 F89A       486          LDI   $9A
0005 A8         487          PLO   R8
0006 D4000A     488          CALL  STR8ASCI        STORE R8 IN ASCII CODE INTO TELETYPE BUFFER
0009 00         489          IDL                   WAIT
```

```
LOCN OBJ CODE  STMT    SOURCE STATEMENT     1802 VER 1.7

                490 *       EXAMPLE SUBROUTINE ***STR8ASCII*** STORES THE CONTENTS OF REGISTER 8
                491 *       INTO THE TELETYPE BUFFER AS A HEXADECIMAL VALUE IN ASCII CODE
                492 *
                493 *       ENTER WITH...
                494 *       R8=VALUE TO BE STORED IN BUFFER
                495 *       RX=R2=STACK POINTER
                496 *
                497 *       RETURNS WITH...
                498 *       ALL REGISTERS UNCHANGED
                499 *
                500 *       THE FORMAT IN WHICH R8 IS STORED IN THE BUFFER IS 3 LEADING BLANKS
                501 *       FOLLOWED BY 4 HEXADECIMAL CHARACTERS
                502 *
000A     =4008  503 TTYBUF  EQU  $4008          STARTING ADDRESS OF TELETYPE BUFFER
000A     =0020  504 BLANK   EQU  $20           ASCII CODE FOR BLANK
000A D7        505 STR8ASCI SEP  R7            SAVE REGISTERS 8-R ON STACK VIA
000B 00        506          DC   $00          MACRO-SUBROUTINE SAVREG88
000C F840      507          LDI  A.1(TTYBUF)  LOAD STARTING ADDRESS OF TELETYPE OUTPUT BUFFER
000E BA        508          PHI  RA           INTO REGISTER RA
000F F808      509          LDI  A.0(TTYBUF)
0011 AA        510          PLO  RA
0012 F820      511          LDI  BLANK        STORE BLANK IN FREE LOCATION
0014 52        512          STR  R2           ON STACK
0015 F802      513          LDI  2            STORE NO. OF LEADING BLANKS MINUS 1
0017 A9        514          PLO  R9           IN LSB R9
0018 D7        515          SEP  R7           DUPLICATE LEADING BLANKS INTO BUFFER VIA
0019 18        516          DC   $18          MACRO-SUBROUTINE DUPSBLOK
001A 88        517          GLO  R8           STORE 4 LS BITS OF R8.0 ON STACK
001B FA0F      518          ANI  $0F
001D 52        519          STR  R2
001E D7        520          SEP  R7           CONVERT BYTE ON STACK TO ASCII VIA
001F 24        521          DC   $24          MACRO-SUBROUTINE HEXASCII
0020 88        522          GLO  R8           STORE 4 MS BITS OF R8.0 ON STACK
0021 F6        523          SHR
0022 F6        524          SHR
0023 F6        525          SHR
0024 F6        526          SHR
0025 52        527          STR  R2
0026 D7        528          SEP  R7           CONVERT BYTE ON STACK TO ASCII VIA
0027 24        529          DC   $24          MACRO-SUBROUTINE HEXASCII
0028 98        530          GHI  R8           STORE 4 LS BITS OF R8.1 ON THE STACK
0029 FA0F      531          ANI  $0F
002B 52        532          STR  R2
002C D7        533          SEP  R7           CONVERT BYTE ON STACK TO ASCII VIA
002D 24        534          DC   $24          MACRO-SUBROUTINE HEXASCII
002E 98        535          GHI  R8           STORE 4 MS BITS OF R8.1 ON STACK
002F F6        536          SHR
0030 F6        537          SHR
0031 F6        538          SHR
0032 F6        539          SHR
0033 52        540          STR  R2
0034 D7        541          SEP  R7           CONVERT BYTE ON THE STACK TO ASCII VIA
0035 24        542          DC   $24          MACRO-SUBROUTINE HEXASCII
0036 F803      543          LDI  3            LOAD NO. OF ASCII CHARACTERS TO TRANSFER MINUS 1
0038 A9        544          PLO  R9           INTO LSB R9
0039 60        545          IRX               POINT X-REG. TO FIRST BYTE ON STACK
003A D7        546          SEP  R7           TRANSFER ASCII CHARACTERS FROM STACK TO BUFFER VIA
003B 12        547          DC   $12          MACRO-SUBROUTINE MOVSBLOK

LOCN OBJ CODE  STMT    SOURCE STATEMENT     1802 VER 1.7

003C 22        548          DEC  R2           POINT TO FREE LOCATION ON STACK
003D D7        549          SEP  R7           RESTORE REGISTERS 8-R FROM STACK VIA
003E 03        550          DC   $03          MACRO-SUBROUTINE RSTREG88
003F D5        551          RETURN            RETURN TO MAIN VIA SCRT RETURN SUBROUTINE
```

SOME ELFISH IDEAS

Paul St. Pierre,
15 Reddington St.,
Coventry, R.I. USA.
02816

I have a Super Elf by Quest Electronics. It didn't take me long to become frustrated with the basic single board Super Elf. After all with only 256 bytes of memory and no storage device, it soon became work rather than pleasure. So I took the first step to expansion. I purchased the 4K Super Expansion Board with the 1K ROM monitor, S-100 interface, cassette interface and parallel I/O ports. A couple of words on the expansion board are in order. The board uses 2102's for the 4K memory with adequate bypass capacitors. It connects to the basic board through a 50 pin ribbon cable (not provided, in fact no connector is provided, it must be ordered at extra cost). The ribbon cable cannot be longer than six inches. The pin out of the 50 pin connector is not included in the expansion board documentation but rather is part of the basic kit documentation.

The memory can be located on any 4K boundary. The original power supply cannot be used with the expansion board. I found this out the hard way. The 1K ROM monitor is on a 2708 and requires +/-5V and +12V. The +5 volts can be regulated on the expansion board but the -5 and +12 must be regulated off the board.

I have had good success with the cassette interface in conjunction with the monitor. It runs at about 133 bytes per second.

The parallel I/O ports are 8212's. The expansion board has room for 2 S-100 conectors. Not all of the S-100 signals are provided. The following signals are provided:

1	+8	51	+8
2	+16	52	-16
		68	MWR
20	NP	70	P
		77	PWR
29	A5	79	A0
30	A4	80	A1
31	A3	81	A2
32	A15	82	A6
33	A12	83	A7
34	A9	84	A8
35	DO1	85	A13
36	DO0	86	A14
37	A10	87	A11
38	DO4	88	DO2
39	DO5	89	DO3
40	DO6	90	DO7
41	DI2	91	DI4
42	DI3	92	DI5
43	DI7	93	DI6
		94	DI1

45	GND	95	DIØ
46	GND	96	
47	MRD		
		99	RESET
5Ø	GND	1ØØ	GND

QUEST recommends that no more than one LS board be put on the buss.

Because of the power supply requirements, I had to buy another power supply so I bought the one Quest offers. It uses a +/-12V transformer to derive all the voltages necessary and is working fine for me.

The first project I undertook was to get RCA's Chip 8 up and running. The first hurdle I had to overcome was the keyboard entry problem. The VIP has a very dumb keyboard that latches one nibble at a time. The Super Elf keyboard latches a full byte at a time and does the key encoding and debounce in hardware.

To make my keyboard look more like a VIP keyboard, I instituted my first hardware mod. When a key is pressed on the VIP, one of the EF lines is forced low. To do this on the Super Elf, I added jumper wire to pin 13 of U25 (the DATA ACCEPT pulse of the 74C923 keyboard encoder) through a diode to a switch, then to pin 13 of U21 (NAND gate used by Input switch on the Super Elf to force EF4 down). With this mod and the appropriate software changes to CHIP 8 (see listing) I can select normal operation or CHIP 8.

All the other changes I made to CHIP 8 are straight forward and mostly have to deal with the relocation of the required operating system routines.

I use page ØFØØ for the display page, ØEØØ for the relocated VIP operating system routines.

The only VIP operating system routines I needed were the bit encoded digits and the interrupt routine.

Another change I made was to the tone generating routine. Since, the Super Elf has the Q line tied to a speaker, I can generate a tone. so I rewrote the tone routine to accept a tone frequency and a tone duration. The disadvantage to this routine is that it ties up the CPU causing any motion on the TV to stop. However, I have found this to be and advantage. In target games the stopping of the action is further identification that the target has been hit. When it is not desirable to stop the action, a judicious selection of frequency and duration will make the pause unnoticeable.

With these changes, I have been able to run the CHIP 8 programs in the VIP manual, most of them without change (except for the tone variable values) and the whole family is having fun with them.

My next project is to build an alphnumeric ASCII display. I am planning on using a Motorola 6847 video graphics chip which will simplify the design. I plan to wire wrap it on a Vector board and plug it in to one of my S-1ØØ slots. If anyone else has done this I would like to hear from them. (Paul, see IPSO FACTO Issues #13 and #14 for 6847 based displays--Editor.)

"The Elf Has Come of Age"

R. C. Francis
111 Guelph St.
Oshawa, Ontario
L1H 6H8
Canada

Since its introduction by Joseph Weisbecker in 1976 as a construction project (Popular Electronics, August 1976, page 33), the RCA 1802 based Cosmac "ELF" microcomputer has enjoyed increasing popularity among Microcomputer hobbyists and electronics experimenters alike. It has been utilized in a wide range of applications encompassing many fields of endeavour - from energy conservation in the home to industrial process control. Because of its simplicity and easy expansion capability the 1802 provides an inexpensive way to get one's feet wet in the sea of microcomputers. Among the factors contributing to its popularity are listed the following:

- minimal power supply requirements (single +5V)
- wide operating voltage range (3V to 12V)
- wide operating frequency (DC - 6.4 MHz)
- wide temperature range (-55° to +125°C) capability
- high noise immunity
- simple adaptation to control applications
- CMOS, TTL, NMOS compatibility
- single phase clock; optional on-chip crystal-control oscillator
- simple control of reset, start and pause
- easy expansion

However, despite this popularity among both the uninitiated and professionals, the 1802 uP has not achieved any real measure of commercial success when compared with such uP's as the 6800, Z80, 8080, 6502 and other frontrunners. As a result, there is a severe lack of available support - hardware and software - for the Cosmac Elf. The only commercial interest in the 1802 has been shown by RCA with the VIP and Cosmac Development System (CDS), Netronics R&D of Connecticut with their Elf II, Quest Electronics of California offering both a Basic Elf and Super Elf microcomputers and Infinite in Florida with their UC 1800 microcomputer. No major software house has provided any support for the Elf as far as can be determined and whatever is available is provided by the aforementioned manufacturers to support their hardware as well as by the few Elfers dedicated to the pleasurable pastime of "Elfing".

One possible explanation for the acute lack of support for the Elf might be that it is different architecturally from most of the CPU's available today. The 1802 CPU is register oriented and lacks the familiar dedicated registers such as Index, Stack Pointer, Condition Code and Program Counter. The principal feature of the 1802 is its register matrix (R) which consists of sixteen 16-bit scratch pad registers. Individual registers in the array (R) are designated by the 4-bit binary code from one of the designators P, X or N. Each register in (R) can be used as either Program Counter, Data Pointer (i.e. Index Register) or as scrathpad locations (data register) for temporary storage of two bytes of data. This flexibility of register manipulation and assignment is indeed, a major programming advantage of the 1802.

With the introduction of RCA's Cosmac Microboard Computer System Modules the Elf has finally come of age. The Microboard family line includes:

(a) Three basic single board computers
(b) Four memory boards (3 RAM + 1 ROM/PROM)
(c) A UART Interface board
(d) A combination memory and I/O board
(e) D/A converter board
(f) A/D converter board
(g) Control and display module
(h) A 5 volt power supply card
(i) Two card cages (5 slots and 25 slots)
(j) A PROM Programmer

Each module is a single 4.5" by 7.5" card having 22 contacts per side on standard .156" centres. The universal backplane incorporates a BUS which was designed to save hardware headaches by allowing any module to be plugged into any slot at any time. The backplane also has full swing logic for that high CMOS noise immunity, and there is no complex logic required to talk to the backplane. The Microboards are also plug-in compatible with all the RCA COSMAC development tools and follow the same pin-out as that of the 22-pin expansion connector on the VIP.

One advantage of CMOS is its low power consumption. This feature lends itself very well to battery operation. The CDP18S622, for example, is an 8k RAM board with rechargeable Ni-cad batteries on board. The batteries can provide 96-hour memory retention with external power off or they can be used optionally for powering other microboard modules on the universal backplane. Great idea. It should be pointed out, however, that these great features of the Microboard family do not come cheaply. The 25-card chassis, for instance, can be obtained at a cost of $594.00 (Can.) from local RCA distributors while the 5-card version will leave a $125.40 hole in any Canadian account. Further details on each module is provided in a 12-page colour brochure CMB-250 available from your nearest RCA sales office.

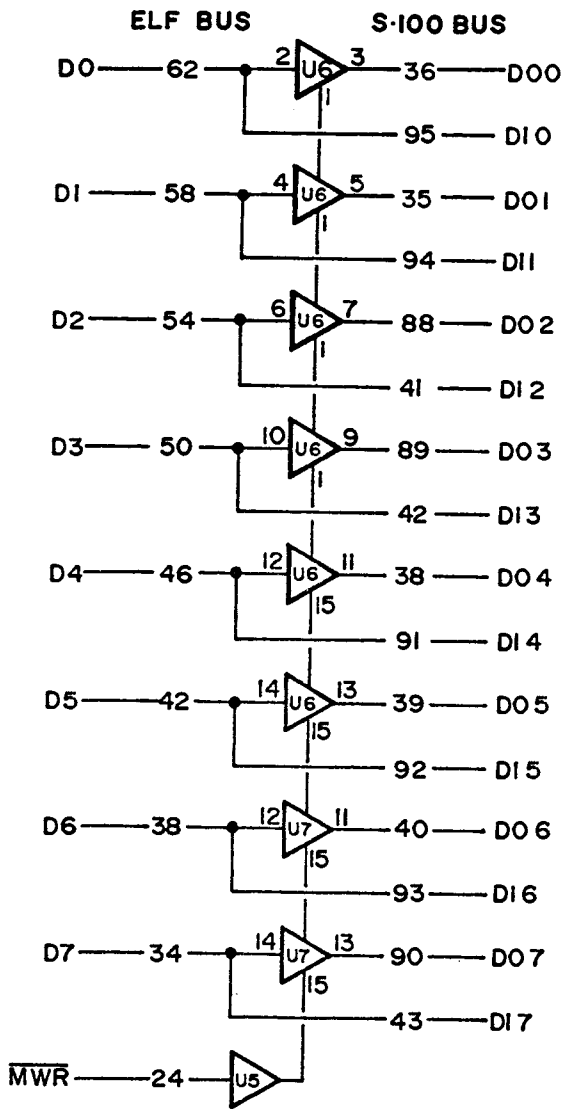
S100 INTERFACE

Charles L. Kimtantas,
1906 Baltimore,
Richardson, TX USA 7508

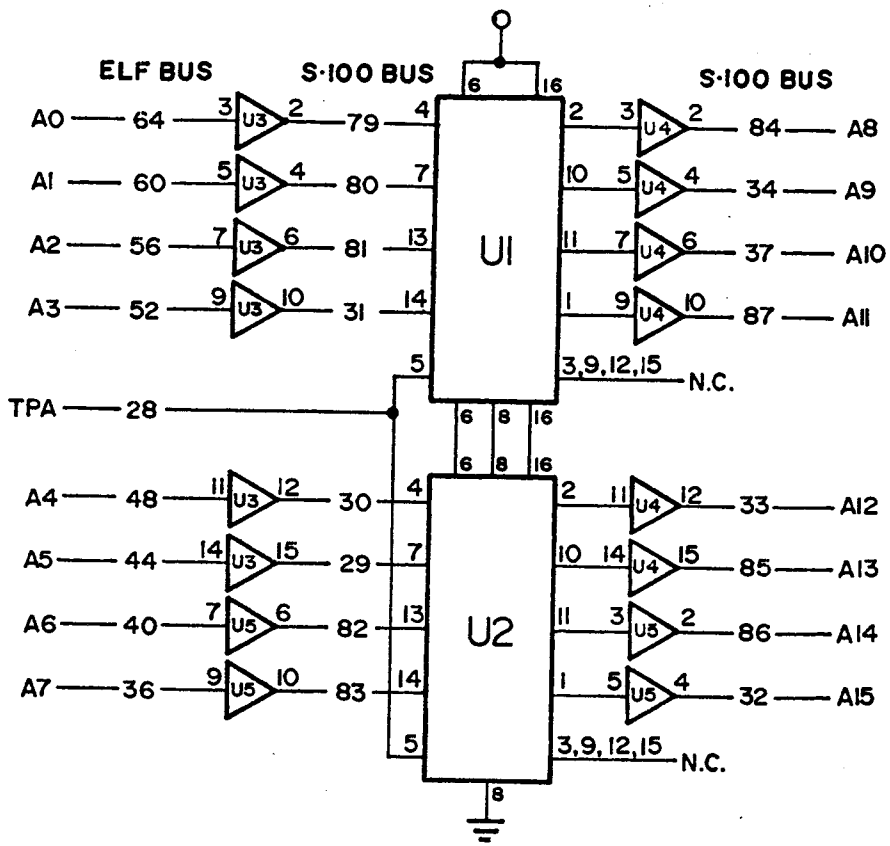
In Issue #7 there was a request for S-100 i;nterface circuits. I have successfully interfaced both an 8K and a 16K static memory board to my ELF. Figure I shows the address latching and buffers required. If only one memory board is used the buffers may not be required. Figure II shows the data line buss drivers and buss connections. Figure III shows the control lines used. The PRESET and SOUT were required for my 16K memory board as it had bank select capabilities. I currently have 32k of S-100 memory in system and the above interface works very well.

There was an article in Kilobaud-Microcomputing June 1979 page 68, that had a slightly different interface and may be of interest.
>

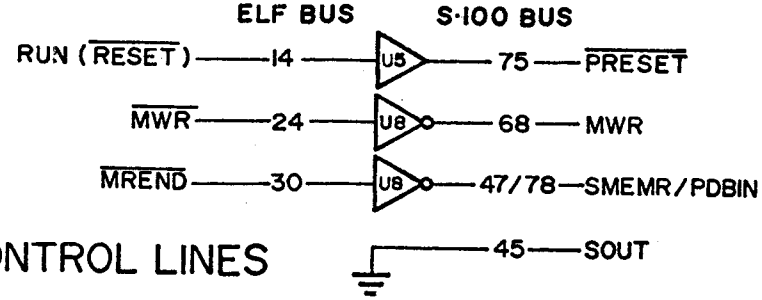
S-100 INTERFACE



DATA INTERFACING



ADDRESS INTERFACING



CONTROL LINES

M.E. Franklin, 24 Duby Rd., Acton, Ont., L7J 2P1

A REVIEW OF RCA UTILITY MONITOR 4 (UT 4).

UT 4 is a 512 byte serial monitor designed to manipulate hex data for an 1802, equiped with a keyboardand video or printer display. RCA provides the monitor as a ROM or as a source listing in several publications. The monitor is written with the following requirements: output data via Q, input via inverted EF 4, program address 8000-81FF(H), with an isolated RAM stack addressed at 8C00-8C1F(H). The monitor is capable of running up to 300 baud with a CPU clock of 1.758Mhz.

Entry to the monitor is via a hardware or software jump to 8000(H). The operator is required to enter a line feed (LF) or carriage return (CR) to designate half or full duplex respectively. Half duplex will not echo commands upon the display devise, while full duplex will echo all keyboard input. UT 4 responds with a prompt (*), and awaits command entry. Baud rate timing is set by the monitor during this time.

The monitor has only three commands, and are used in the following format:

-$P hex address CR - execute program begining at hex address.

-!M hex address space hex data optional , or ; or space CR -
enter data to memory (write memory)

-?M hex address space # of bytes to be displayed plus 1 CR -
read data from memory (read memory)

The use of a comma in the !M command permits the continuous entry of sequential data without the need to enter new !M address commands after every 16 bytes or at the end of your TVT's line buffer. A semi-colon permits entry of data at varying address locations without inputing the !M command at each line input, however, the address must be input of course. No input or a space followed by a CR terminates the memory write command.

The two "M" commands have dual function, in that they will also drive or receive data to/from a terminal devise, such as a printer, cassette recorder, or paper punch. Input is via the inverted EF 4 line.

This monitor offers considerable versatility, considering its small size. I use it to display data on my Elf II system, including Cybernex TVT and H-14 printer. The monitor displays 15 lines of 16 bytes each, for a total of 240 bytes, almost a full page.

The monitor is available from RCA suppliers on a 512 byte ROM, #CDP **R512** at about $20., or a listing can be obtained from RCA publications MPM 203A or MPM 224 at about $15. apiece. The two publications also contain a wealth of information about other RCA products, including a listing of RCA Tiny Basic in MPM 224.

Loading the program listing offers the user the opportunity to relocate the monitor wherever one chooses, un invert EF 4, and remove or relocate the RAM stack.
UT 4 may be relocated by changing only three addresses, M(8002) and M(81FD)H to the new first page high address, and M(80D0)H for the new second page high address.

The input flag may be un inverted (or changed) by changing the following address contents:

	from 37 to 3F	from 3F to 37
M	8109	8107
	8117	8114
	-	**8126**
	814F	814D
	8153	-
	8161	-
	-	8173

Using an inverted EF 4 would be compatible with Netronics Tiny Basic (origional version), but a normal EF 4 is compatible with Netronics Video Board and Tiny, the Text Editor, Dissassembler and Assembler.

The provision of the RAM stack at memory location 8C00 -8C1F can present some difficulty. The stack has only one function - to store the contents of the registers at monitor initialization. The stack fills memory from highest to lowest RAM available, repeating itself over and over until it runs out of RAM or runs into ROM.

UT 4 will operate properly without the RAM stack, so one solution would be to either omit the RAM at address 8C00, or relocate the stack to a portion of memory containing no RAM or into ROM. UT 4 will write out the register contents, test to see if it can recover them, and failing to do that, will end the routine and continue initializing. To relocate the monitor, just change the contents of M(8005) and M(8007)H to the new high page address and end of stack low address respectively. If you choose to relocate the stack to another RAM location, the new RAM must have either ROM or no memory below it, else the register save routine will write backwards throughout your entire memory. If your system employs a floating 256 byte RAM, it must be disabled else the stack will follow it through your memory and over write your program. The 256 byte Ram will always appear to be at any address the monitor tests, since it is enabled anywhere other memory is not located. It may be disabled by taking its input enable to ground -(210ls on a ELF11, use a SPDP switch on buss line 69).

Another alternative is to dedicate a small unit of RAM for stack or array storage purposes, and relocate the stack to that address. The 256 byte RAM may be useful for this purpose, or a 1 "k" RAM block located in the middle of a ROM monitor block.

Dave Taylor, one of our members from San Fransisco, has written about his modifications to UT 4. His solution to avoiding the RAM stack problem was to insert a Branch to M(8029) at M(8004)H. He also found the TIMALC routine, the one that determines the baud timing rate, to be picky at 300 baud on his ELF 11. His solution was to re write UT 4 to ignor the TIMALC routine and use a predetermined value in RE.1 to set the baud rate. His changes are as follows:

M(8000)H	F8	80	B3	B5	F8	08	A5	D5	E5	71	55	61	01
	F8	**FE**	A3										
M(8010)H	D3	F8	xx	BE	30	39	C4	C4	C4				

where xx is set to 24 for full duplex at 300 baud, or 64 for full duplex at 110 baud. Corresponding half duplex values are 25 and 65 respectively. I found that my micro worked properly with values 26 rather than 24 and 66 rather than 64, so you may have to experiment a bit to determine the correct value for your micro's particular clock rate. Of course, if you try the first route, relocating the RAM stack to no RAM and have no trouble initializing, then the above changes are unnecessary.

UT 4 will accept full cursor control when not executing one of its three instructions. The format of the output of UT 4 can not be programmed by use of the cursor control, nor can corrections be made to data listing by use of back space or delete or cancel controls. The use of a cursor command while executing a program will automatically terminate execution of the command and return the monitor to the ready state, with a new line prompt. My keyboard encoder, an AY 5-2376, accepts the following cursor commands:

"control" and H backspace
I forward space
J line feed
K home cursor
L home cursor and clear page
M return to start of line
N reverse line feed
O clear to end of line

While cursor commands can not be used with UT 4 executed commands directly, they can be used successfully with assembly languages utilizing UT 4.

W. Petty has written three good articles on applications of UT 4 and the 1802 in KILOBAUD. The first describes putting the 1802 on the S-100 buss, with UT 4 as the system monitor(June 1979). The second describes using UT 4 with Tiny Pilot, listing of Tiny Pilot included(July 1979), and the third lists a Tiny Text Editor (Dec, 1979).

UT 4 is a versatile monitor and a welcome addition to my system. Since it is EPROMABLE, I have placed UT 4 and RCA VIP Op Code together in a 2708 located at M(FC00)H. The "F" memory block is designated as my monitor area, and contains a 1"k" block of RAM for stack, video display and array purposes.

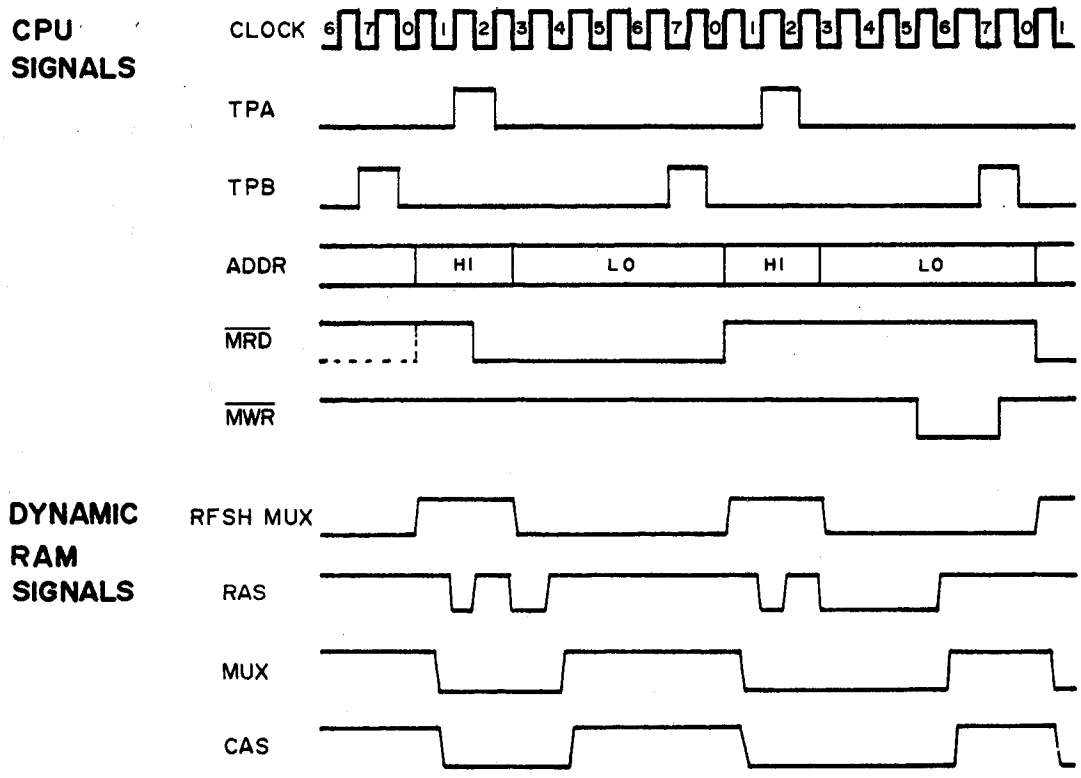
I hope that the above review will help you determine whether UT 4 could be useful to your programming applications, and if so, to help you get it running.

MORE ON THE 16K DYNAMIC RAM

Harley Shanko,
Apt. 209,
15025 Vanowen St.,
Van Nuys, Calif.,
USA 91405

The following timing diagram should be of value to anyone building the 16K Dynamic RAM published in Issue 14 of IPSO FACTO. Please refer to page 53 for the power supply used with the RAM board.

1802/DYNAMIC RAM TIMING



Richard N. Thornton
1403 Mormac Road
Richmond, Va. 23229

Dear Bernie,
I've been reading the 12 back issues of IPSO Facto which arrived about a month ago. They're great! These back issues prompted this letter.

There are two minor things I'd like to see in future issues, if not too much trouble. One would be to include the current number of members somewhere on the front page. Second, I wish the editors would sign their remarks, as I'm never sure who is speaking with a number of editors on the team.

I noted a number of requests by the editors of back issues for more support, articles, nominations, help with the newsletter, etc. While distance poses a problem for those of us in other countries, I'm sure you would be flooded with volunteers if you find something we might be able to do.

Some of the letters to the editor in back issues contained questions which were not answered by the editor. I assume the editor did not know the answer. I further assume that some of these questions were in fact answered by others by direct communication. What bothers me is that these answers were not shared with the members at large via IPSO FACTO. I wish that people who offer help would send a copy to you for the newsletter so that we might all benefit.

I recently tried to program 74S287 PROM's using a haywired homebrew circuit, and really fried them good! Now I think I'll try EPROM, and follow more experienced people. The 2708 EPROM programmers shown in issues 11 and 12 are both very ingenious, and I think I'll try one of them. In reviewing the circuits, I believe there are a couple of minor errors in issue 11, but may be wrong:

issue 11, page 13: 2708 pins 22 and 23 are reversed (a nit); NO and N1 should be reversed if the program on page 11 is to be used?

I am sending along several pages of stuff you may be able to use in the newsletter if there is a gap needs filling. Anyhow, I've contributed.

Thanks very much for the best newsletter in the world!

Dear Richard:

If questions in letters are not answered, we are inviting other readers to contact the letter writer. Our current membership is about 450.

As for signing our comments individually, there has generally been enough 'group-think' (and double-talk?) going into them to justify taking group responsibility. -Editor

>

An Information Index System

Millions of words are written about microcomputers and other electronic systems every year. Some of this is valuable to you as you read it, but much does not reach you when you are ready to use it. The trick is to be able to locate it when you are ready. Annual indexes are published by many magazines, which range from simple lists of article titles to detailed topical indexes. Few, however, will give you the detail you may need.

My solution to this problem is to index everything I read in 3X5 inch cards. The card index is in two parts, a general index of articles, and a detailed subject index. The content of each of these parts will be unique to the individual.

In the general article index, I have arrived at a number of topics, and enter each article in one of the topic areas. One card will contain about 10 article entries. As each article is entered on the card, it is assigned a code number which is used in the detailed subject index. The full title of the article, newsletter, book, etc., is entered on the general index card with the publication name and date.

The detailed index contains one card for each information element I find useful. To minimize problems, I always use the full name of the element rather than abbreviations. Thus I have Read Only Memory, but not ROM. Duplicate entries under varying element names results in partial listing on any one card, as you can never remember all the similar element names.

As example, my general index has 23topics, which include computer circuits(CC), digital circuits (DD), 1802 microcomputer (EL), interfacing (IF), Keypads and keyboards (KP), linear devices (LD), miscellaneous circuits (MC), memory (MM), and others. As you can see, my interests are hardware oriented. As each new article is entered on a card, it is assigned the next higher number with the prefix indicated in parentheses above. The last article in the digital circuits topic is identified as DD26. This identifier is used on the detailed index cards for brevity.

While reading new material, I make short notes showing the subjects which contain substantial information in the article. When devices are mentioned, I index by the number only, disregarding the prefix and suffix letters, if any. For example, uA723 and LM723 are both indexed as 723. After finishing the article, I enter the information on the cards. Som of my detailed index subjects include: analog multiplier, assembler, avalanche transistor, bi-directional bus, break beam detector, cassette tape interface, debounce, edge detector, frequency doubler, tone sequencer, vectored interrupt, 1861 video controller, 339 quad comparator, etc. The detailed index card for touch controlled latch contains the following: TS05, TU37, KK06, LD22.

ELECTRICAL POWER MONITOR SYSTEM

With electrical power as expensive as it is today, it would be beneficial to to be able to monitor use of this commodity in order to minimize its use and detect the least efficient devices in the home. The microcomputer should be very useful in this application. Unfortunately, I do not know enough to accomplish this, but would be very interested in anything which may have been done, or in current development by others.

I would conceive the system to contain a microcomputer (1802, of course) which read the output of an analog-to-digital converter which produced a value proportional to the current in the attached circuit. The value read, and time of day would then be written to tape. The readings would be taken periodically for one or more days. The recorded data could then be reduced by a program to print a graph, display on a CRT, etc. By checking each line at the fuse or circuit breaker box, and perhaps even at individual appliances, some very valuable information might be obtained.

The analog-to-digital converter is the difficult part of the system, of course. To be effective, it should be capable of reporting how many amperes of current are flowing in the connected circuit (0-256 amperes should be sufficient for most homes.) The converter should not affect the circuit being measured, and should be inexpensive.

I believe a coil of a few turns of wire around one side of the ac line might provide sufficient pickup to be amplified by a differential amplifier and applied to an ADC. Design and calibration of this circuit, or its equivalent are quite beyond me, however. I would like to hear from anyone who might have the solution to this problem. Perhaps he might be willing to write it up for IPSO FACTO.

SOME THOUGHTS ON A MONITOR

Years ago, I worked with a Honeywell 120 computer with 12K of memory, which did all the accounting and production scheduling for a manufacturer with 15 branch offices. This computer had no resident monitor, and we had to key in a "bootstrap" program to load programs. While there, I collaborated with the Honeywell representative in creating a primitive operating system. This system included a program loader, and a number of commonly used service routines. In order to minimize the effects of monitor changes on our applications programs, we used a branch table which was located immediately after the program loader. If I wanted to branch to the tape read routine, I branched to a fixed location which contained a branch to the tape read routine. Now, if monitor changes resulted in a new location for the tape read routine, my program was unaffected. I plan to use this technique in the monitor I'll eventually have in EPROM.

I feel the monitor should include operator utility functions, such as memory examination, memory change, memory scan, program relocation, program execution from any address, register display, as well as program services which might include tape read, tape record scan, tape write, tape record header creation, compare memory fields, move memory fields, add and subtract with multiple precision, radix conversion, and perhaps multiply and divide. The mix of routines would depend upon the hardware configuration and usual applications of the owner, of course.

Cassette Tape Subroutine Library

In large computer systems, subroutine libraries are provided which contain subroutines which have been written to provide common functions. The programmer inserts "CALL" statements in his programs which result in a standard linkage. After assembly, a "LINKAGE EDITOR" program is run which locates the called subroutines, incorporates them into the program, and relocates everything. This is a bit beyond most of us, but we can save a lot of redundant coding by placing commonly used routines on separate tape files, and reading them into the appropriate location in programs being created.

I have one cassette which contains nothing but subroutines. Each subroutine is documented on paper, showing which file on the tape contains the routine, and identifying all locations which require change: branch addresses, address constants, etc. As written on the subroutine tape, all locations which must be changed contain a 68, which is an unusual byte value. After loading the subroutine, the monitor scan routine is used to locate each 68, which is then changed to the correct value using the memory change function of the monitor. I find this reduces programming effort, keying errors, and logic errors.

At present my subroutine library consists of only four modules: an initialization routine which initializes registers to zeros, points R2 to the stack area, points R1 to the interrupt routine, etc. The second routine is a 64X32 display interrupt routine for the 1861, third is a routine to read a byte from the data entry switches and echo it back via a 64 instruction. Last is a short routine to read the monitor in from tape. I find this useful for those large programs which require all of my memory, and destroy the monitor, usually using the area as a display page.

Even with a very small library, it rapidly becomes evident that you need to establish some conventions on register use so that subroutines know how to exit, and which registers can be freely used for work, which must be preserved, etc. So far, I have decided to use R1 for the interrupt routine, R2 for the stack pointer, R3 as the main program instruction register, R4 will contain the primary display page address, R8 is the X register for input-output, R5, R6, and R7 are subroutine instruction registers. The remaining registers are available for work. I'm sure these conventions will change as my experience and equipment configuration improves.

The main difficulty I've found with the subroutine library so far is that I have to locate each subroutine manually, as my monitor has no ability to identify files. This is something I'll include in the monitor when I get some EPROM.

MINI TAPE LOADER

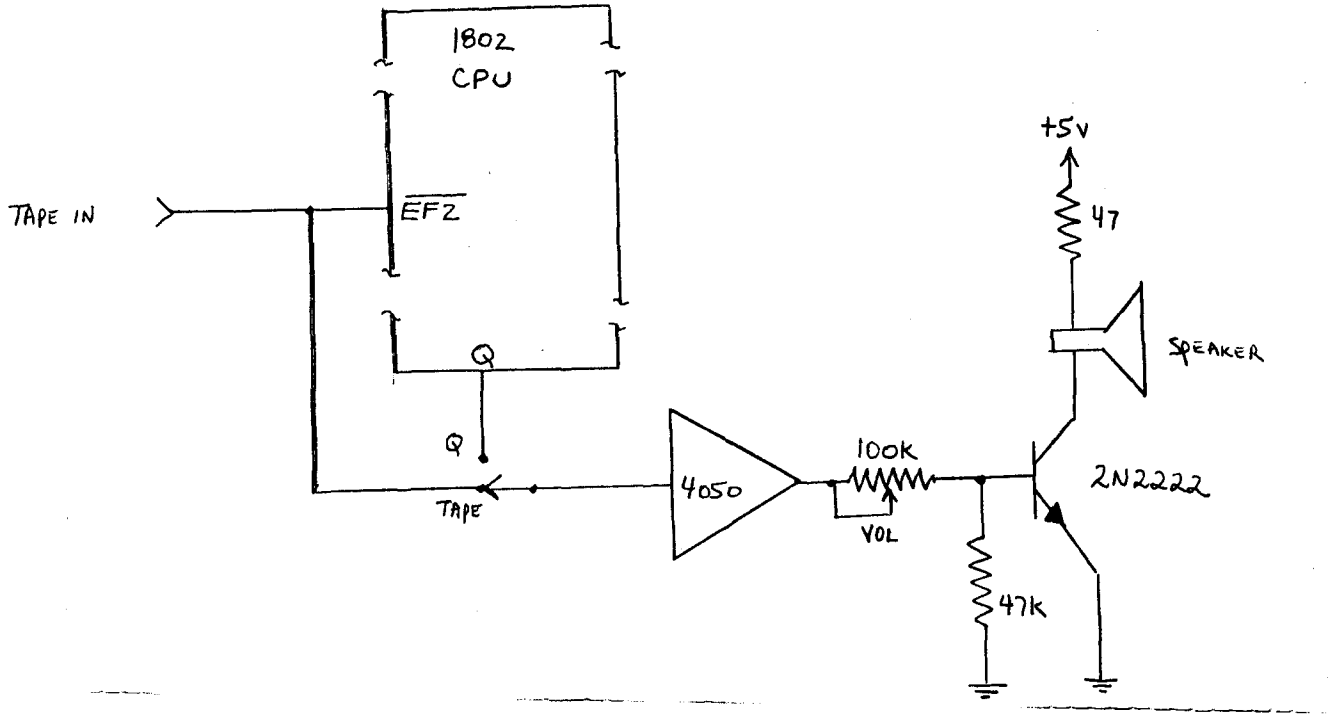
I have a homebrew P.E. Elf with 1K of Ram, but without a ROM monitor. After trying the monitor included in the article "Expanding the Elf II" by Martin Meyer in the March, 1978 issue of Popular Electronics, I realized that cassette tape routines and the ability to quickly inspect and modify memory are essential to any serious attempt to program the Elf. The 256 byte monitor is too long to key in before each session, however, so I extracted the tape read routine, and use it to load the monitor. The principal goal was to produce a workable loader in the fewest possible bytes. In use, the program is keyed in, the tape is positioned to the leader tone, and the program is run. On completion, The program branches to the monitor, and turns the "Q" light on. Each byte is displayed using a 64 instruction so that you can see it is working. I load the monitor at 0300. If another page is desired, change the bytes at locations 0004 and 001F. It is best to enter a long branch to the monitor (C0 0300) into memory locations 0000-0002 so that you will branch to the monitor after the next RESET - RUN sequence.

LOC.	LABEL	INSTRUCTION	MNEMONICS	COMMENTS
0000		90 AA B7	GHI PLO PHI	00 to RA.0 and R7.1
0003		F803 BA	LDI PLO	Start address to RA (0300)
0006		F822 A7	LDI PLO	READBIT address to R7 (0022)
0009		EA	SEX	
000A	LEADER	D7	SEP	Perform READBIT
000B		330A	BDF LEADER	Loop past leader 1 bits
000D		F801 BD	LDI PHI	01 to RD.1
0010	NEXTBIT	D7	SEP	Perform READBIT
0011		9D	GHI	Get byte in progress
0012		7E	RSHL	Shift bit into byte
0013		BD	PHI	And put it back in RD
0014		3B10	BNF NEXTBIT	Continue until byte filled
0016		D7	SEP	Bypass parity bit
0017		9D	GHI	Get completed byte
0018		5A	STR	And store at current address
0019		64	OUT4	Display and increment RA
001A		8A	GLO	Get low byte of address
001B		3A0A	BNZ LEADER	To next byte if not end
001D		7B	SEQ	End, tell user via "Q"
001E		C00300	LBR MONITOR	And go to the monitor
0021	RBEXIT	D0	SEP	Subroutine return
0022	READBIT	F80E	LDI	Get timer constant
0024	WAITLOW	3524	B2 WAITLOW	Wait until tape goes high
0026	BITCHEK	3521	B2 RBEXIT	Exit with DF=1 if tape goes low here (1-bit)
0028		FF01	SMI	Decrement timer count
002A		3326	BDF BITCHEK	Continue while count non-zero
002C	PASSZERO	3D2C	BN2 PASSZERO	Wait for rest of zero bit
002E		3021	B RBEXIT	Exit with DF=0, zero-bit

Before using this program, you must first have loaded the monitor and written it to tape using the 03 function.

Cassette Tape Monitor

Since my cassette tape speaker does not work when the computer is plugged into the EAR jack, I was never sure what was happening when reading tape. I always had to let it run for a long time to be sure leader was past before stopping to see what went wrong. This could be solved if I could hear the tape as is was being read. One solution would have been to modify the tape machine, but I was reluctant to do that. Instead, I added a volume control and switch to the speaker connected to the "Q" line for music. The switch permits the speaker to be used to monitor either the "Q" line or the tape in signal. The volume control is necessary if you have sensitive ears, or the leader tone will drive you batty.



ERRATA

There are two mistakes on the VDU article in issue 13. The first is that an INVERTER buffer was placed directly on the WRITE line of the 2102's (i.e. pin 3) from IC 18 (i.e. pin 3 of a NAND gate. This INVERTER buffer does not belong anywhere. The second mistake was in the blown up diagram found on page. The unlabeled AND gate should be a NAND gate (part of IC 18).

In 'Four-Byte Delight' there were two transcription errors in the program listing. The instructions at locations 003D and 0060 should both reference register 3 instead of 2 (which is uninitialized, and points nowhere).

Specifically at 003D 83 instead of 82
and at 0060 93 instead of 92.

COSMAC GRAPHICS PROGRAM

Jerry Steinberg,
4361 Cambie St.,
Vancouver, B.C.,
Canada. V5Z 2Y6

What started out as a program to facilitate creating new "life-forms" for Conway's game of Life (see Scientific American Oct. 1970 and Ipso Facto June 78) ended up as a graphics program that was interesting in its own right. The program written for the COSMAC ELF with the CDP 1861 Video Chip uses about 1K of memory.

When the program is run, a blinking cursor is displayed in the center of the screen. By using any of 8 move keys (hex keypad), the cursor can be moved up or down, sideways or diagonally. To display a dot anywhere on the screen, the cursor can be moved to the desired location and a write command given (commands also entered by the hex keypad). When the cursor is moved away, the dot will remain. Conversely to remove a dot, the cursor is moved to the desired location and an erase command given. To make it easier to display or remove a connected series of dots, additional commands can be given that will cause all the positions the cursor vacates to be either written into or erased. The cursor moves one position each time a move key is pressed, however it can be made to move throughout the time that the move key is depressed. This automatic mode can be enabled or disabled by pressing the correct command keys. The program also has a feature which remembers up to the first 1024 commands keyed in and allows these commands to be cycled through repeatedly. For those users with more than 1K of memory, data is included for a demonstration run; details on how to see the demonstration are given later in this article.

Memory (address 0900 to 09FF) is displayed on the monitor (via the interrupt routine) and the cursor is controlled by using a hexadecimal key pad. The following commands are recofnized:

Control Keys		Move Keys		
0	Erase	1	Move Cursor	S W
5	Write	2	" "	S
A	Erase screen and display cursor in the center	3	" "	S E
		4	" "	W
B	Constant write	6	" "	E
C	Constant Erase	7	" "	N W
D	Continual Read from depressed key pad (automatic mode)	8	" "	N
		9	" "	N E
E	Disable D			
F	Return to main and use stored commands			

Commands are read in by subroutine MAIN and then handed to subroutine CURSOR where they are interpreted. Many versions of MAIN can be used; the one in this article includes a blinking cursor as well as the ability to redraw the picture using up to the first 1024

commands processed by subroutine CURSOR. In this version, commands are packed 2 per byte and are stored in memory starting at address 0700. When more than 1024 commands are processed, the commands will start writing into the memory that is being displayed i.e. 0900 to 09FF. However when command F is given only those commands stored from 0700 to 08FF are cycled through. If the user wishes to store and recycle over more than 1024 commands, the contents of 066B (this location holds a constant that allows branching when end of storage is reached) should be changed to a larger number and a different area of memory displayed. The high order starting address of the display (in this case 09) will have to be changed in three locations: 040A (to clear correct page) 0421 (to display initial cursor position) and 05B5 (for the interrupt routine to know what page to display). For example, if an additional 512 commands are to be stored, then store 0A at 066B, 040A, 0421 and 05B5.

The following is an example of how the commands can be used to draw a square:

Press the following keys: A,4,4,4,B,8,8,8,6,6,6,6,6,6,
2,2,2,2,2,2,4,4,4,4,4,4,8,8,8

An extension can be made to one side of the square and then erased as follows:

4,4,4,0,6,0,6,0,6

That could also have been done in this way:

B,4,4,4,C,6,6,6

To place a dot in the center of the square:

5,6,6,6,5

Move the cursor up and out of the square:

8,8,8,8,8,8

Pressing key D causes any depressed key (except key E) to be continually read in, therefore the combination of:

B,D,9 (keeping key 9 depressed)

will cause the screen to fill with diagonals. To erase these diagonals press the following keys:

E,C,D,1 (Keeping key 1 depressed for
as long as key 9 was depress

Command E was given so that command C wouldn't be continually read in by the program thus uselessly filling the space reserved for storing commands.

Now press key F. The entire sequence of commands should be

repeated again and again until the program is stopped. A more interesting result can be seen if the above sequence of commands is repeated except this time omitting to kpress key A. The same commands will be cycled over, only this time the screen won't be erased at the end of each cycle and the cursor won't start in the middle of the screen, but instead, at the last position it occupied before key F was pressed. If the program is stopped, the same sequence of commands can be seen by restarting the program at the address used for the demonstration run, 0670.

Running the Program and the Demonstration Run:

To run the program start running at address 0400. To see the demonstration, load the demonstration data into address 0700 to address 07F4 and start running ar address 0670. The words COSMAC ELF should be displayed on the screen. When the program is run from 0400, the domenstration data wil be replaced by whatever commands are used. Users must have more than 1K of memory (starting from 0400) to see the demonstration.

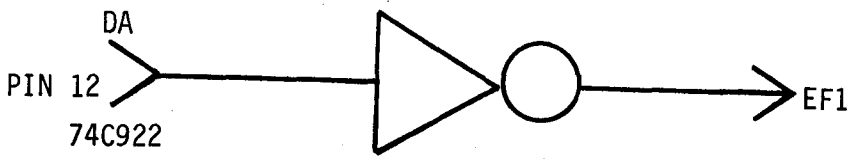
Some modifications will have to be made to the program for users who only have 1K of memory. Memory from 0700 to 07FF will have to be used for the display, and there will be no room to store commands used. Kay F will also have to be disabled. Change the high order starting address of the display to 07 by loading 07 into locations 040A, 0421 and 05B5. Bypass storing commands by loading 30 into locations 061C and 063Am, 21 into location 0610 and 3E into location 063B. Disable key F by loading D3 into locations 0627, 0628, 0644 and 0645 (D3 is a 2 cycle NOOP).

REGISTER	USE
0	address of display page
1	interrupt routine address
2	stack pointer
3	program counter
4	subroutine call link address
5	subroutine return link address
6	temporary variable storage
7	stores input from keypad
8	temporary variable storage
9	address where commands are to be st
A	temporary variable storage
B	free
C	temporary variable storage
D	2 flags
E	address of cursor
F	2 flags

Notes:
------

(1) Register 3 must be used as the program counter before the program is run.

(2) To make the graphics program easier to use, one small hardware modification was made to our otherwise standard ELF. The hex keypad was made 'self-strobing', that is, after depressing one of the keys on the pad it is not necessary to press the input switch to enter the data into the computer. Our hex keypad uses a 74C922 encoder chip, and we connect D A (data available pin 12) through an inverter to EF1, thus making the keypad 'self-strobing' (see figure below). For those that don't have a hex keypad or for those who don't want to modigy their computers, strobe data into the ELF with the input switch. If you wish to make our change, all B4's in the program must be changed to B1's (there are 6 locations: 058E, 059F, 060B, 0615, 0633 and 629).



(In our circuit, the inverter is really a multi-input NOR gate because we use EF1 for more than one purpose.)

(3) The time delay for the automatic made can be changed by loading a different number into location 0497. For example, if 06 replaced the 02 (which is currently at this location), the cursor will move more slowly.

(4) The program can be coupled to the game of life by making the game use the display page as the initial generation.

(NOTE: Included in this article were Assembler listings of all mainline and subroutines. Due to space restrictions, we were not able to reproduce the listings. For those interested, we suggest contacting Jerry directly. Thanks Jerry for the well documented article. --Editor)

COSMAC GRAPHICS PROGRAM

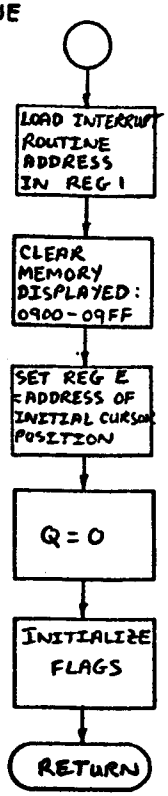
```
0400  C005 EDF8 05B1 F8AD A1F8 09BA F8FF AAEA
  10  F800 738A FBFF 3A10 F805 B7F8 CBA7 E769
  20  F809 BEF8 7BAE F801 AF5E 7AF8 00BF ADBD
  30  D5E7 F0FD 0D3A 3BF8 01BF D5F0 FD0E 3A44
  40  F800 BFD5 F0FD 053A 4F7B F800 BDAD D5F0
  50  FD00 3A5A 7AF8 00BD ADD5 F0FD 0B3A 677B
  60  F801 BDF8 00AD D5F0 FD0C 3A70 F801 ADD5
  70  8D3A 7531 80D4 0566 FBFF EEF2 5EE7 3084
  80  9D3A 847A F0FD 0A3A 8DD4 0403 D5F0 FD0F
  90  3A93 D59F 329D F802 BC2C 9C3A 99F0 FD01
  A0  3AA7 D405 3B30 EEF0 FD02 3AB1 D405 4E30
  B0  EEF0 FD03 3ABB D405 5330 EEF0 FD04 3AC5
  C0  D405 1D30 EEF0 FD06 3ACF D405 2C30 EEF0
  D0  FD07 3AD9 D404 F230 EEF0 FD08 3AE3 D405
  E0  0530 EEF0 FD09 3AED D405 0A30 EED5 D405
  F0  76D5 8FFD 083A FF8E FF09 AEF8 01AF D58E

0500  FF08 AE1F D58E FF08 AED5 8FFD 013A 178E
  10  FF07 AEF8 08AF D58E FF08 AE2F D58F FD08
  20  3A2A 8EFF 01AE F801 AFD5 1FD5 8FFD 013A
  30  398E FC01 AEF8 08AF D52F D58F FD08 3A48
  40  8EFC 07AE F801 AFD5 8EFC 08AE 1FD5 8EFC
  50  08AE D58F FD01 3A60 8EFC 09AE F808 AFD5
  60  8EFC 08AE 2FD5 8FAC F801 BC2C 8C32 749C
  70  FEBC 306B 9CD5 D405 66EE F1AC F53A 807B
  80  8C5E F802 BC2C 9C3A 85D5 F810 BC2C 37A4
  90  9C3A 8DD4 0566 FBFF EEF2 5EF8 10BC 2C37
  A0  A49C 3A9E D405 66EE F15E D572 7022 7822
  B0  52C4 C4C4 F809 B0F8 00A0 80E2 E220 A0E2
  C0  20A0 E220 A03C BA30 ABC4 C400 D3B8 E296
  D0  7386 7393 B683 A646 B346 A398 30CC D3B8
  E0  96B3 86A3 E212 72A6 F0B6 9830 DEF8 05B4
  F0  F8CD A4F8 05B5 F8DF A5F8 06B2 F8FF A2C0

0600  0602 D404 03F8 07B9 F800 A937 12D4 058A
  10  300B 9F3A 1737 15E7 6CFA 0F57 FEFE FEFE
  20  59D4 0431 07FD 0F32 4837 30D4 058A 3029
  30  9F3A 3537 33E7 6CFA 0F57 E9F4 5919 D404
  40  3107 FD0F 3248 300B F807 B9F8 00A9 09F6
  50  F6F6 F657 D404 3107 FD0F 3248 09FA 0F57
  60  D404 3107 FD0F 3248 1999 FD09 3248 304E
  70  F805 B4F8 CDA4 F805 B5F8 DFA5 F806 B2F8
  80  FFA2 D404 03D4 0648
```

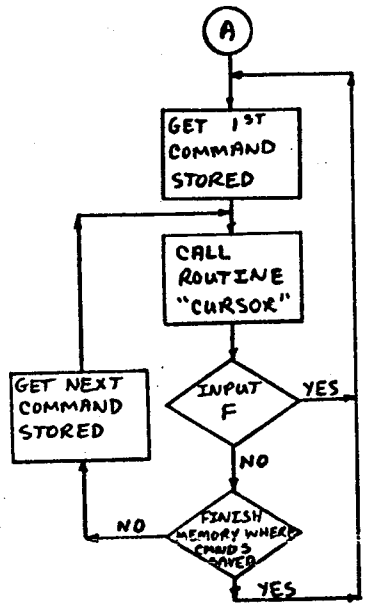
```
700  44 44 44 44 44 44 44 44
708  44 44 48 88 88 88 66 66
710  68 44 44 42 22 22 22 26
718  66 66 56 68 66 66 68 88
720  88 88 84 44 44 22 22 22
728  22 56 66 66 66 88 82 36
730  66 88 77 77 88 66 63 25
738  22 22 22 66 88 88 88 88
740  86 33 99 62 22 22 22 25
748  66 88 88 88 89 96 33 22
750  22 22 86 84 44 45 66 66
758  22 26 66 66 66 84 44 44
760  88 88 88 88 66 66 65 44
```

DATA FOR DEMONSTRATION

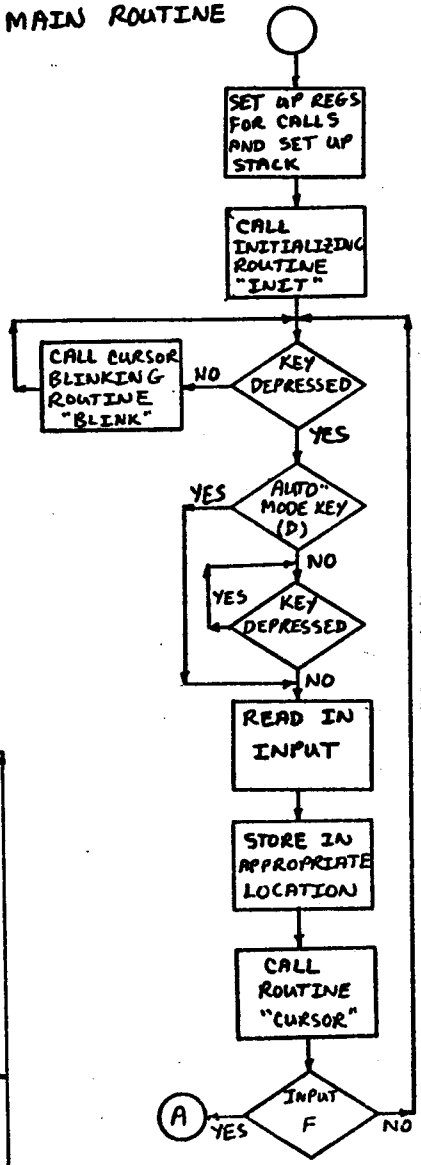
768	44	42	22	22	22	22	24	44
770	44	44	44	44	44	44	44	44
778	02	22	26	6B	44	88	64	88
780	66	56	6B	22	22	66	56	6B
788	88	64	88	66	54	48	44	44
790	44	44	44	44	44	48	88	88
798	88	86	66	11	11	66	66	77
7A0	77	73	3E	E6	61	11	22	99
7A8	94	44	99	94	44	33	31	11
7B0	99	97	77	66	61	11	66	66
7B8	88	88	44	44	42	22	66	62
7C0	22	44	88	88	66	62	22	22
7C8	26	88	88	88	83	44	46	22
7D0	22	44	66	66	D8	82	82	82
7D8	82	82	82	82	82	82	82	82
7E0	82	82	82	82	82	82	82	82
7E8	82	82	82	82	82	82	82	82
7F0	82	82	82	FF	FF			

SUBROUTINE "INIT"

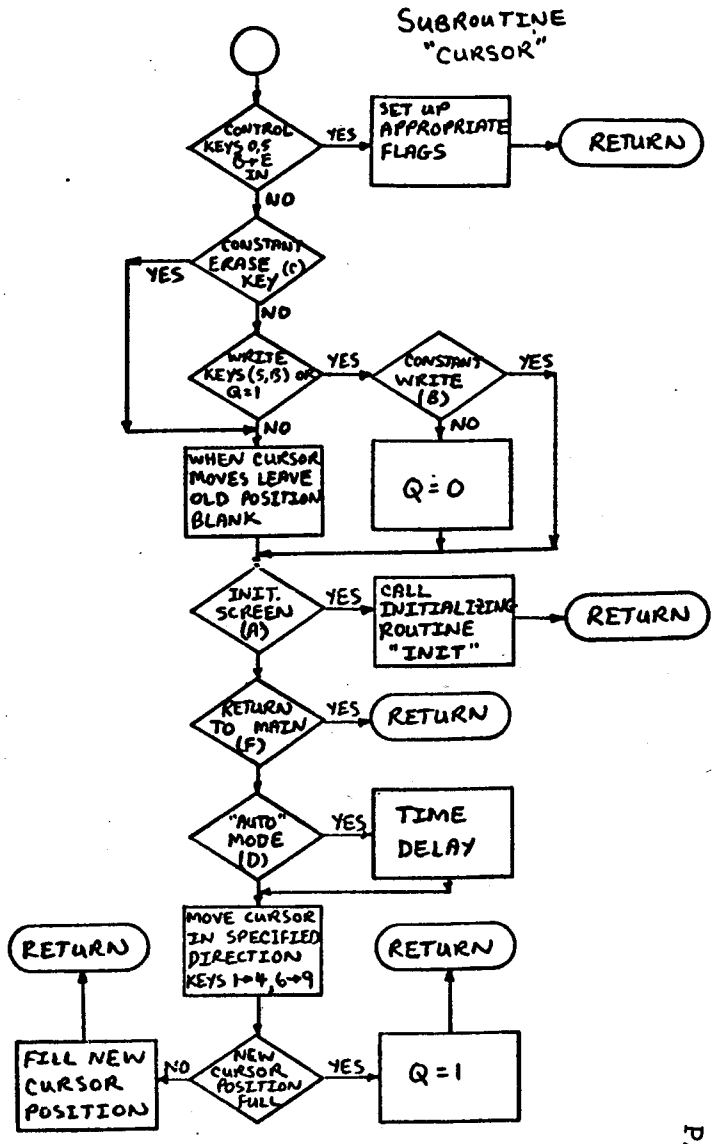




"MAIN" ROUTINE



SUBROUTINE "CURSOR"



LIFE FOR THE 1802!

G.E. Millar,
P.O. Box 1412,
Parksville, B.C. Canada.
V0R 2S0

The following is a copy of "LIFE" which I have rewritten from the April '79 Byte, page 166, to work on the 1802 system. I've written a loader with prompts and I have included a loop test for number of iterations. The program requires the SCRT and the loader requires one subroutine similar to READAH (UT4) to input ASCII Hex. If the system does not have a similar routine, one will have to be written and called at location 0156. The program uses location 0900 to 09FF as the playing board. It is very similar to the TBI version I sent in but runs much faster.

With a few changes, the playing area could be expanded to take in 1024 bytes (32 x 32) instead of the 256 bytes (16 x 16) it presently uses.

LIFE See page 166 in the April 79 issue of "BYTE". The following program is my version of the "LIFE" program written for the 1802. I have included a loader with prompts and a loop test for number of iterations. The program requires the SCRT and the loader requires one subroutine similar to the READAH in (UT4) to input ASCII Hex.

```
0000   F800   BDINIT:   LDI    #00    ;  Uses SCRT registers plus
0002   AA               PLO    RA     ;  RA, RB, RD & RF
0003   AB               PLO    RB
0004   F809             LDI    #09    ;  Playing area 0900 to 09FF.
0006   BA               PHI    RA
0007   8B       CLR:    GLO    RB     ; Clear matrix.
0008   5A               STR    RA
0009   1A               INC    RA
000A   8A               GLO    RA
000B   3A07             BNZ    CLR
000D   F809             LDI    #09
000F   BA               PHI    RA
0010   2B               DEC    RB     ; Set RB.0 =FF
0011   8B       TOPROW: GLO    RB
0012   5A               STR    RA     ; Draw top line.
0013   1A               INC    RA
0014   8A               GLO    RA
0015   FB11             XRI    #11
0017   3A11             BNZ    TOPROW
0019   2A               DEC    RA
001A   8B       EDGE:   GLO    RB     ; Draw edge.
001B   5A               STR RA
001C   8A               GLO    RA
001D   FC10             ADI    #10
001F   AA               PLO    RA
0020   3A1A             BNZ    EDGE
```

```
0022  D40100 LOAD:   CALL  LOADER ; Load initial cells & iterations.
0025  F811   WRITE:  LDI   #11
0027  AA             PLO   RA     ; Point to first cell.
0028  F809           LDI   #09
002A  BA             PHI   RA
002B  F80E   BWRITE: LDI   #0E    ; Set counter for line length.
002D  AB             PLO   RB
002E  8A             GLO   RA
002F  BB             PHI   RB     ; Save cell location.
0030  D40120         CALL  CRLF   ; Output "cr" & "lf".
0033  F82A           LDI   #2A
0035  D4830F         CALL  VIDEO  ; Output "*" (boarder).
0038  F800   TEST:   LDI   #00
003A  AF             PLO   RF
003B  EA             SEX   RA
003C  1A     TS1:    INC   RA
003D  8F             GLO   RF
003E  F1             OR
003F  AF             PLO   RF
0040  2B             DEC   RB
0041  8B             GLO   RB     ; Is it the end of the line?
0042  3A3C           BNZ   TS1    ; No.
0044  8F             GLO   RF     ; Yes.
0045  E2             SEX   R2     ; Does line contain cells?
0046  324B           BZ    TS2    ; No.
0048  D40053         CALL  WRTLIN ; Yes, print line.
004B  1A     TS2:    INC   RA
004C  1A             INC   RA
004D  9A             GHI   RA     ; Reached end of matrix?
004E  F6             SHR
004F  3B81           BNF   MOVE   ; Yes.
0051  302B           BR    BWRITE ; No.
0053  9B     WRTLIN: GHI   RB     ; Restore cell location.
0054  AA             PLO   RA
0055  F80E           LDI   #0E    ; Set counter for line length.
0057  AB             PLO   RB
0058  4A     NEXT:   LDA   RA     ; Check for "cell","nocell" or
0059  3276           BZ    NOCELL ;      "boarder".
005B  F6             SHR
005C  326A           BZ    ALVE
005E  F82A   SLASH:  LDI   #2A    ; Boarder "**"
0060  D4830F         CALL  VIDEO
0063  F82A           LDI   #2A
0065  D4830F         CALL  VIDEO
0068  307C           BR    OVER
006A  F85B   ALVE:   LDI   #5B    ; Cell "[]"
006C  D4830F         CALL  VIDEO
006F  F85D           LDI   #5D
0071  D4830F         CALL  VIDEO
0074  307C           BR    OVER
0076  D40134 NOCELL: CALL  SPACE  ; No cell "  "
0079  D40134         CALL  SPACE
007C  2B     OVER:   DEC   RB     ; Line completed?
007D  8B             GLO   RB
007E  3A58           BNZ   NEXT   ; No.
0080  D5             RETN         ; Yes, return.
```

```
0081  F811    MOVE:   LDI   #11
0083  AA              PLO   RA     ; Point to first cell in matrix.
0084  AB              PLO   RB
0085  F809            LDI   #09
0087  BA              PHI   RA
0088  BB              PHI   RB
0089  D400AB  NXCELL: CALL  VALCK ; Check cells around this cell.
008C  D400E5          CALL  ESET  ; Set next generation value of cell.
008F  1B              INC   RB
0090  8B              GLO   RB     ; Completed matrix?
0091  3A89            BNZ   NXCELL ; No.
0093  F811    UPDATE: LDI   #11    ; Yes, now update board.
0095  AA              PLO   RA
0096  F809            LDI   #09
0098  BA              PHI   RA     ; Point to first board position.
0099  0A      LOOP:   LDN   RA
009A  FE              SHL          ; Is it a boarder position?
009B  33A0            BDF   BRDR   ; Yes, skip it.
009D  0A              LDN   RA     ; No, Shift next generation into
009E  F6              SHR          ;  this generation.
009F  5A              STR   RA
00A0  1A      BRDR:   INC   RA
00A1  8A              GLO   RA     ; Completed?
00A2  3A99            BNZ   LOOP   ; No.
00A4  2D              DEC   RD
00A5  8D              GLO   RD     ; Completed iterations?
00A6  3A25            BNZ   WRITE  ; No, do next generation.
00A8  C00139          LBR   ITERN  ; Yes.
00AB  F800    VALCK:  LDI   #00    ; Check value of cell pointed to
00AD  AF              PLO   RF     ; by RB. Clear counter.
00AE  0B              LDN   RB
00AF  FBFF            XRI   #FF    ; Check for boarder.
00B1  32E4            BZ    NC2    ; Yes, return.
00B3  8B              GLO   RB     ; No.
00B4  FF11            SMI   #11    ; Check cell above & to the left.
00B6  AA              PLO   RA
00B7  D400D3          CALL  NCOUNT
00BA  1A              INC   RA     ; Check cell directly above.
00BB  D400DB          CALL  NCOUNT
00BE  1A              INC   RA     ; Check cell above & to right.
00BF  D400DB          CALL  NCOUNT
00C2  8B              GLO   RB     ; Check cell directly to left.
00C3  AA              PLO   RA
00C4  2A              DEC   RA
00C5  D400DB          CALL  NCOUNT
00C8  1A              INC   RA     ; Check cell directly to right.
00C9  1A              INC   RA
00CA  D400DB          CALL  NCOUNT
00CD  8B              GLO   RB     ; Check cell below & to the left.
00CE  FC0F            ADI   #0F
00D0  33E4            BDF   NC2    ; On last line ignore these cells.
00D2  AA              PLO   RA
00D3  D400DB          CALL  NCOUNT
00D6  1A              INC   RA     ; Check cell directly below.
00D7  D400DB          CALL  NCOUNT
00DA  1A              INC   RA     ; Check cell below & to right.
```

```
00DB  0A      NCOUNT: LDN   RA
00DC  FBFF            XRI   #FF    ; Check for boarder?
00DE  32E4            BZ    NC2    ; Yes, return.
00E0  F6              SHR          ; Is it = 1 or 3?
00E1  33E4            BDF   NC2    ; No, return.
00E3  1F              INC   RF     ; Yes, inc counter.
00E4  D5      NC2:    RETN         ;   Return.
00E5  0B      ESET:   LDN   RB     ; Set value of cell (as
00E6  FBFF            XRI   #FF    ;  determined by VALCK)
00E8  32E4            BZ    NC2
00EA  0B              LDN   RB
00EB  FA01            ANI   #01    ; Remove any second generation
00ED  5B              STR   RB     ;  bits.
00EE  8F              GLO   RF     ; Get VALCK counter.
00EF  FB02            XRI   #02    ; Is it = 02?
00F1  32F7            BZ    SRVIV  ; Yes.
00F3  F6              SHR          ; No, is it = 01?
00F4  32FA            BZ    ALIVE  ; Yes.
00F6  D5              RETN         ; No, return.
00F7  0B      SRVIV:  LDN   RB     ; Is cell alive this generation?
00F8  32E4            BZ    NC2    ; No, return.
00FA  0B      ALIVE:  LDN   RB     ; Set cell alive next generation.
00FB  F902            ORI   #02
00FD  5B              STR   RB
00FE  D5              RETN
00FF  00              BYTE  #00    ; Spare byte.
0100  F809    LOADER: LDI   #09
0102  BA              PHI   RA     ; Point RA.1 to board.
0103  D4015C          CALL  TEXT
0106  0176            DBYT  TAB1   ; "ITERATIONS?"
0108  D4012C          CALL  CRLF   ; Output "cr" & "lf".
010B  D40153          CALL  INPUT  ; Input # of iterations.
010E  FB0D            XRI   #0D    ; Completed?
0110  3A00            BNZ   LOADER ; No, loop again.
0112  8D              GLO   RD     ; Yes, save number.
0113  BB              PHI   RB
0114  D4015C          CALL  TEXT   ; "INPUT CELLS "
0117  0183            DBYT  TAB2
0119  D40153  LDR1:   CALL  INPUT  ; Input cell location.
011C  FB0D            XRI   #0D    ; 1st nonHex not = to "cr"
011E  3A2A            BNZ   LDR2   ;   terminates loader.
0120  8D              GLO   RD
0121  AA              PLO   RA     ; Point RA at cell location
0122  F801            LDI   #01    ;  and set cell alive.
0124  5A              STR   RA
0125  D4012C          CALL  CRLF   ; Output "cr" & "lf".
0128  3019            BR    LDR1   ; Go get next cell location.
012A  9B      LDR2:   GHI   RB     ; Restore RD with # of iterations.
012B  AD              PLO   RD
012C  F80D    CRLF:   LDI   #0D    ; Output "cr".
012E  D4830F          CALL  VIDEO
0131  F80A            LDI   #0A    ; Output "lf".
0133  C8              LSKP
0134  F820    SPACE:  LDI   #20    ; Output "space".
0136  C0830F          LBR   VIDEO
```

```
0139   D4012C ITERN:   CALL   CRLF
013C   D4015C          CALL   TEXT
013F   0171            DBYT   TAB3   ; "MORE ITERATIONS? "
0141   D40153          CALL   INPUT  ; Get number.
0144   FB40            XRI    #40    ; Is it a "M"?
0146   C28051          LBZ    MONITOR ; Yes, return to MONITOR.
0149   FB4D            XRI    #4D    ; No, is it a "cr"?
014B   3A39            BNZ    ITERN  ; No, try again.
014D   8D              GLO    RD     ; Yes, ensure iterations #0.
014E   CA0025          LBNZ   WRITE  ; Okay.
0151   3039            BR     ITERN  ; No. Is = 0, go try again.
0153   F800    INPUT:  LDI    #00
0155   AD              PLO    RD     ; Clear RD.
0156   D4814C IN1:     CALL   READAH ; (UT4 location 813B)
0159   3356            BDF    IN1
015B   D5              RETN
015C   9B      TEXT:   GHI    RB     ; Output text.
015D   73              STXD          ; Save RB.
015E   8B              GLO    RB
015F   73              STXD
0160   46              LDA    R6     ; Point RB to TAB.
0161   BB              PHI    RB
0162   46              LDA    R6
0163   AB              PLO    RB
0164   4B      TX1:    LDA    RB     ; Output TEXT from Table.
0165   D4830F          CALL   VIDEO
0168   0B              LDN    RB     ; Is it = 0?
0169   3A64            BNZ    TX1    ; No, loop again.
016B   60              IRX           ; Yes, restore RB and return.
016C   72              LDXA
016D   AB              PLO    RB
016E   F0              LDX
016F   BB              PHI    RB
0170   D5              RETN
0171   4D4F    TAB3:   DBYT   #4D4F ; "MORE "
0173   5245            DBYT   #5245
0175   20              BYTE   #20
0176   4954    TAB1:   DBYT   #4954 ; "ITERATIONS? "
0178   4552            DBYT   #4552
017A   4154            DBYT   #4154
017C   494F            DBYT   #494F
017E   4E53            DBYT   #4E53
0180   3F20            DBYT   #3F20
0182   00              BYTE   #00
0183   494E    TAB2:   DBYT   #494E ; "INPUT CELLS "
0185   5055            DBYT   #5055
0187   5420            DBYT   #5420
0189   4345            DBYT   #4345
018B   4C4C            DBYT   #4C4C
018D   5320            DBYT   #5320
018F   00              BYTE   #00
```

1. Label "VIDEO" refers to system output routine, either video or printer.
2. Label "READAH" refers a routine which will input Hex characters into RD.0, such as "READAH" in UT4.
3. Label "MONITOR" is the exit from the LIFE program to your system moniter routine.

TINY BASIC CORNER

Micheal Mirsky, VE3 INC,
10 De Quincy Blvd.,
Downsview, Ont., Canada.
M3H 1Y5

Here is a program I thought I'd share with you. It finds the day of the week for any date after 1541 (that's the year the Gregorian calendar started). The program uses the famous formula known as Zeller's Congruence.

```
10 REM   THIS PROGRAM CALCULATES THE DAY OF THE
20 REM   WEEK FOR ANY DATE AFTER 1541.
25 REM
30 REM   ENTER THE NUMBER OF THE DAY, MONTH AND YEAR.
40 REM   E.G. JAN.=1, FEB=2 .....
45 REM
46 Y=999
50 PR"THIS PROGRAM FINDS THE DAY OF THE WEEK FOR ANY DATE"
60 PR"ENTER THE DAY, MONTH, YEAR";
70 INPUT D,M,P
80 K=(60+10000/(100*M)))/100
90 L=P-K
100 O=M+12*K
110 Z=(L/400)-(L/100)+((5*L)/4)+(13*(O+1)/5)+D-1
120 Z=(Z-(7*(Z/7)))+1
130 PR"THE DAY OF THE WEEK IS ";
140 GOSUB Z*1000
145 PR
150 PR"DO YOU WANT TO ENTER ANOTHER DATE (Y OR N)";
160 INPUT A
180 END
1000 PR"SUNDAY"
1010 RETURN
2000 PR"MONDAY"
2010 RETURN
3000 PR"TUESDAY"
3010 RETURN
4000 PR"WEDNESDAY"
4010 RETURN
5000 PR"THURSDAY"
5010 RETURN
6000 PR"FRIDAY"
6010 RETURN
7000 PR"SATURDAY"
7010 RETURN
```

I hope to send in some of my machine language programs in the future. I find Basic easy to program with but is very slow so sometimes I find it's worth the effort to use machine language.

G.E. Millar,
P.O. Box 1412,
Parksville, B.C. Canada.
VØR 2SØ

The following two programs illustrate the use of two dimentional arrays in Tiny Basic (TBI Palto Alto).

```
  5   REM*  SAMPLE PROGRAM. 2 DEMENTIONAL ARRAYS.
 10   REM*  INPUT QUANTITY OF STOCK TO WARE HOUSES.
 20   FOR J=1 TO 3; FOR I=0 TO 9 STEP 3
 30   PR.#0,"INPUT QTY OF ITEM#",I/3+1," IN WHS#",J,
 40   INPUT@(J+I); NEXT I; NEXT J
 50   REM*  CHECK WARE HOUSES FOR STOCK ITEMS
 60   REM*  LESS THAN TEN IN QUANTITY.
 70   PR."WHS.# ITEM#  QTY."
 80   PR."-----------------"
 90   FOR J=1 TO 3; FOR I=0 TO 9 STEP 3
100   IF @(J+I)>9 GOTO 120
110   PR.#2,J,#5,I/3+1,@(J+I)
120   NEXT I; NEXT J; PR."-----------------"
130   STOP
```

```
  5   REM*  SAMPLE PROGRAM. BOOKING SEAT RESERVATIONS.
 10   REM*  I= ROWS(1 TO 5), J= SEATS(1 TO 4).
 20   Y=1; N=0
 25   REM*  CLEAR MATRIX.
 30   FOR I=0 TO 25; @(I)=0; NEXT I
 35   REM*  INPUT RESERVATIONS.
 40   INPUT"SEAT#"S; IF S>4 GOTO 40
 45   INPUT"ROW#"R; IF R>5 GOTO 45
 50   IF @(R*5-5+S)=1 PR."SORRY SEAT IS RESERVED."; GOTO 70
 60   PR.#0,"YOUR SEAT (",S,",",R,") IS RESERVED."
 65   @(R*5-5+S)=1
 70   INPUT"DO YOU WANT ANOTHER (Y/N)"X
 80   IF X=1 GOTO 40
 90   INPUT"DO YOU WANT A SEAT LISTING (Y/N)"Z; IF Z=0 STOP
100   PR.#5,1,#6,2,3,4
110   FOR I=0 TO 20 STEP 5; PR.#0,I/5+1,
120   FOR J=1 TO 4; IF @(J+I)=0 PR.#2,,"EMPTY",; GOTO 140
130   PR.#2,,"RESVD",
140   NEXT J; PR.; NEXT I; STOP
```

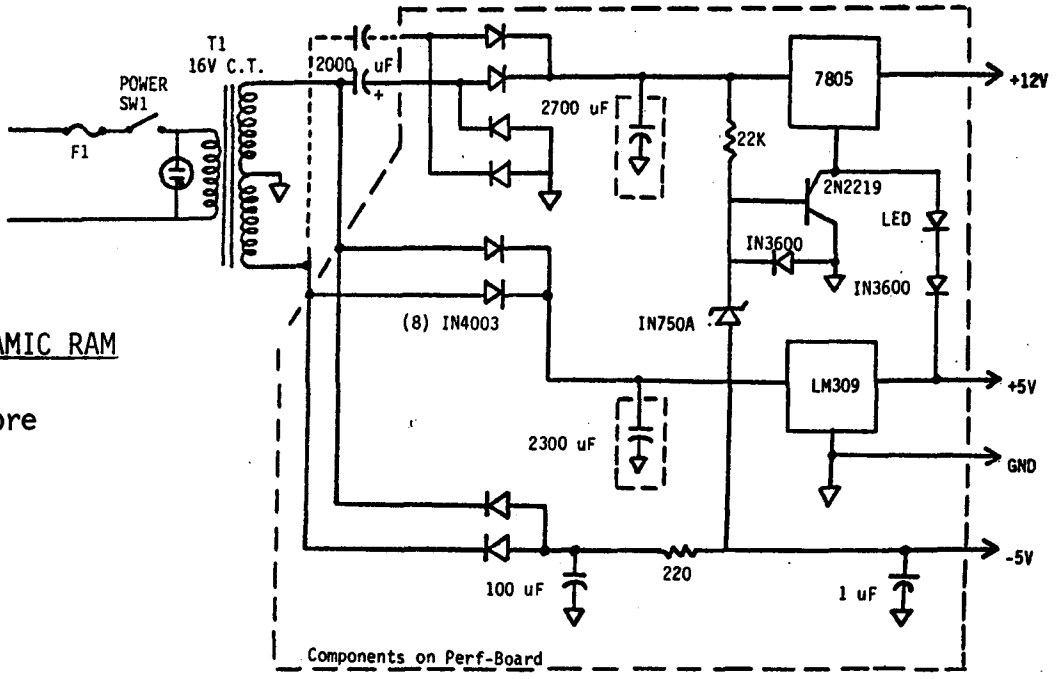
The following is the "HIDDEN PEA GAME" which is to be run on a video display only. Line 75 outputs a VT (vertical tab) so that the next output over writes the previous one and line 8Ø clears the line before you enter your guess. Line 7Ø is a delay loop which is determined by line 1Ø input. At line 9Ø the input will be either 1, 2, or 3.

e.g. *+( )*( )*(+)*+ pea is in location 3

```
  5  REM*  HIDDEN PEA GAME.
 10  INPUT"ENTER DIFFICULTY FACTOR (1,2,3)"T;T=T*T*T
 15  C=1; D=0; N=0; Y=1
 20  INPUT"READY (Y/N)"Z; IF Z#1 GOTO 20
 25  FOR I=1 TO 10
 30  B=RND(3)*10+40; GOTO B
 40  PR."*+(+)*( )*( )*+"; GOTO 70
 50  PR."*+( )*(+)*( )*+"; GOTO 70
 60  PR."*+( )*( )*(+)*+"
 70  FOR Z=1 TO T; NEXT Z
 75  PR.^ K,; NEXT I
 80  PR."                "
 90  INPUT"WHERE'S THE PEA"Z
100  IF Z#(B-40)/10+1 PR."SORRY YOU LOSE"; GOTO 120
110  PR."LUCKY GUESS!!"; D=D+1
120  C=C+1; IF C<4 GOTO 20
130  PR.#0,"YOU WON ",D," OUT OF 3."
140  INPUT"NEW GAME (Y/N)"Z; IF Z#0 GOTO 15
150  STOP
```

MORE ON THE 16K DYNAMIC RAM

(See page 33 for more information.)



16K DYNAMIC RAM POWER SUPPLY

Steve Nies
2510 Deas Street
Bossier City, LA 71111

After reading my article on "1802 8-Level Interrupts", I noticed one particularly disturbing fact. Namely, the program locations mentioned in the article did not match with the actual locations in the program. After I saw this error I realized what happened.

After I first wrote this program (along with the article), I discovered a bug in it. I then rewrote the program to kill the bug. Somehow, I managed to submit my new program along with the old article.

However, my problems are not yet over. While I was trying to figure out the first problem, another slid into view. The reference manuel I used in designing this program was wrong! The table in the book that explains the generated vector code was in error. According to the schematic of the inner workings of the IC, the vector code should have been complimented. The table did not show this. Consequently, the program submitted is also wrong!

Yet, that was not all of the problems my research raised. Somehow, the wires on the schematic connecting the 8214 with the data buss were drawn reversed (Strike three, I'm out).

After all this, I threw out the idea of trying to patch it all together and started fresh. First, since the program was the easiest to change, I rewrote it. All that was required here was to eliminate the bytes FB FF at location XX17 and switch the remaining code around a little. The complete program is shown in Figure 1.

Second, the locations referenced by the original article must also be changed. Please refer to Table 1 for the corrected locations.

Paragraph	Line	Old Value	New Value
2	5	XX1D	XX1E
2	9	XX2B	XX2C
3	3	XX20	XX1E

Table 1.

Finally, a new schematic is shown in Figure 2. Please note that three new NAND gates were added. After talking to a friend, he pointed out that my notation on the original article was confusing, since he did not have the same I/O setup that I did. Therefore, I went ahead and redesigned this section to accommodate the "standard" way of interfacing I/O ports.

I am sorry for any confusion these errors might have caused. However, after extensive testing, these errors have been eliminated and the Interrupt package should run as designed.

```
                    /* INTERRUPT */

XX00  70 C8                          Go Execute Program
XX02  70 23                          Go Execute Subroutine
XX04  E2 22 78 22 73 7E 73           Save T, D, and DF
XX0B  93 73 83 73 96 73 86 73        Save R(3) and R(6)
XX13  F8 S B6                        Set Up Interrupt Pointer
XX16  6F FA 07 52 67 22
XX1C  FE FC 70 A6                    S 70 is the beginning of the
XX20  46 B3 46 A3                         interrupt table in my system
XX24  91 B6 F8 2C A6                 Set Up Return Pointer
XX29  E1 30 02
XX2C  E3 71 23                       Disable Interrupts
XX2F  93 B1 F8 35 A1 D1              Set PC to R(1)
XX35  F8 FF 52 67
XX39  42 A3 42 B3                    Restore R(3)
XX3D  42 F6 42                       Restore DF, D
XX40  30 00

                    /* SWI */

XX42  93 B0 F8 48 A0 D0              Set PC to R(0)
XX48  12 42 A6 42 B6                 Restore R(6)
XX4D  42 A3 42 B3                    Restore R(3)
XX51  42 F6 42                       Restore DF, D
XX54  12 12                          Restore R(2)
XX56  C0 ZZ ZZ                       Jump to User's Program
                                           (In my case, the monitor)
```

Figure 1.

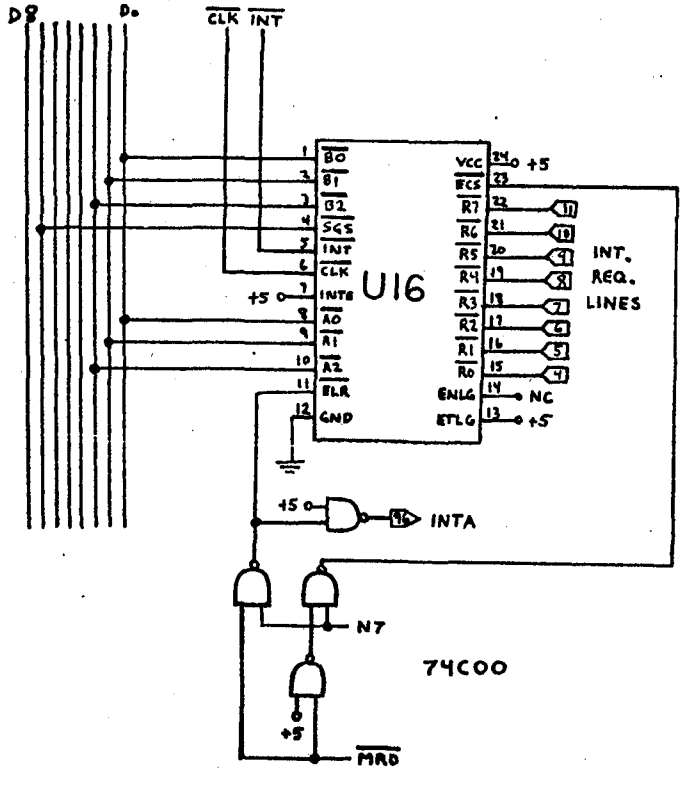


Figure 2.

MEMBERSHIP RENEWAL/CHANGE OF ADDRESS

The dues for the new club year, September 1979 to August 1980 are as follows:

Canada	$15.00	
Elsewhere.....	$18.00	(Canadian)

These new rates include provision for FIRST CLASS POSTAGE which should ensure that your copy of IPSO FACTO arrives as quickly as possible.

Do you have any friends? If so, they might well appreciate a gift subscription to IPSO FACTO.

New members are required to pay the full dues for the current year and will receive all the IPSO FACTO issues for the current year (September 1979 to August 1980). Back issues may be purchased for $15.00 (issues 1-6, May 1977 - May 1978).

Please make cheque or postal money order payable in Canadian funds and send to:

CHANGE OF ADDRESS ☐

NEW SUBSCRIPTION ☐

RENEWAL ☐

"THE ASSOCIATION OF COMPUTER EXPERIMENTERS",
c/o M.E. Franklin, Treasurer,
A.C.E.,
24 Duby Road,
ACTON, Ontario, Canada.
L7J 2P1

FIRST NAME

LAST NAME

FIRST LINE OF ADDRESS

SECOND LINE OF ADDRESS

CITY & PROVINCE

POSTAL CODE

TELEPHONE NUMBER

ATTACH OLD MAILING LABEL
HERE FOR CHANGE OF ADDRESS