VIPER

VOLUME 1 OCT 1978 ISSUE 4

EDITORS EDITION

$2.00

AN ARESCO PUBLICATION

E D I T O R I A L

Welcome to issue number four! Now that we have provided the two most-requested pieces of information for the VIP (the disassembled listings of CHIP-8 and the ROM operating system), we are ready to get into some of the exciting applications for CHIP-8 that our readers have provided. This issue continues Don Stein's information on his Editor program, and Bill Barrett has an article on a breakpoint processor for CHIP-8 which will allow you to set 'stopping points' in your CHIP-8 program so you can examine register contents before continuing. The bigger a program gets, the harder it gets to debug, and Bill's program should prove most useful for system development.

At the New York Personal Computing Show, RCA demonstrated a model railroad being controlled by a VIP. What have you interfaced your VIP to? Send us your interfacing schematic so our readers who are not hardware-oriented can work on control applications too.

VIP to VIP COMMUNICATION!

Last week I saw a demonstration which amazed me and opens up a whole new world of applications for VIPers. By adding a simple circuit card to the VIP, you can take the tape output line and transmit it through a normal telephone line to a similarly-equipped VIP, which receives the signal through the tape input jack. Using this simple connection, we quickly transferred several VIP programs from one machine to another. It would be a simple step to program the tape routines to transmit messages or data between two VIPs anywhere in the country. The only drawback is that you have to manually switch the circuit from transmit to receive. Thus, two VIPs can't "talk back and forth" without a lot of switch-throwing.

I am trying now to get the rights to publish the schematic in the VIPER, so that we can all experiment with this new capability. By the next issue we should know whether we can transmit programs to any similarly-equipped VIP. If you are especially interested in this capability, write to me and we'll see if we can start a 'network'. If there is enough interest, we might get RCA to provide the board as a VIP option!

SUBSCRIPTION RATES, ADVERTISING RATES AND OTHER ESSENTIAL INFORMATION

The VIPER is published ten times per year and mailed to subscribers on the 15th day of each month except June and December. Single copy price is $2.00 per issue, subscription price is $15.00 per year (all ten issues of one volume.) Dealer prices upon request. Outside of Continental U.S. and Canada, add $10.00 per subscription for postage ($1.00 for single copy).

Readers are encouraged to submit articles of general interest to VIP owners. Material submitted will be considered free of copyright restrictions and should be submitted by the 1st day of the month in which publication is desired. Non-profit organizations (i.e., computer clubs) may reprint any part of the VIPER without express permission, provided appropriate credit is given with the reprint. Any other persons or organization should contact the editor for permission to reprint VIPER material.

Advertising rates are as follows:

1/4 page – 25.
1/2 page – 45.
3/4 page – 65.
full page – 85.

Less than 30% of the VIPER will be available for advertising. Please send camera ready copy in the exact page size of your ad on 8-1/2 x 11 white stock by the 1st day of the month in which you'd like the ad to appear. Photos should be glossy black & white in the exact size to be printed. Payment required with copy.

The VIPER is an Aresco Publication, edited by Terry L. Laudereau. For information contact Editor, VIPER, P.O. Box 43, Audubon, PA 19407.

The VIPER is not associated with RCA in any way, and RCA is not responsible for its contents. Inquiries should be directed to ARESCO at the address above or by telephone to (215) 631-9052.

DEAR VIPER —

I am glad to see that your magazine exists, and am very interested and excited about your plans. I am a writer, and computer programming has proven to be the ideal outlet hobby for me.

I have developed my own editor (written in 1802 machine language) for writing Chip-8 programs. It is capable of entering new instructions with an automatic scroll, scrolling and paging forwards and back, and relocating any block of memory forwards or back to any other location. This resides (with much room to spare) in the first two pages of memory, allowing Chip-8 programs to be written and edited, then run after loading the interpreter which I store on tape immediately following the editor. I've also written games -- sur - round, Life (in 2 pages of machine language -- with a new generation every 3 sec.) and am currently working on checkers and Othello. I am stalled in the checkers program while I wait to receive the expansion 2K from RCA, but the program is written.

If RCA is listening, it would be a big help to have a new video chip -- that would plug right in -- with the ability to compress the resolution horizontally as well as vertically!

I'd love to write for the VIPER if you are interested. Please let me know if I may submit articles to you on the programs I have written.

Some of the things I'd like to see are: Interfacing with keyboards and other devices; math techniques in machine language; Chip-8 modifications; questions and answers.

One thing that I've had trouble with is performing long branches in machine language with the video interface on. One way out is:

Use a utility register (I always reserve R_3 for these things) as a temporary program counter. Let's say you are using R as your program counter (P=3) and want to jump from 0250 back to 0165:

```
024A  F8  LDI  Load D with
  4B  01       High order of jump address and
  4C  BF  PHI  put into RF.1
  4D  F8  LDI  Load D with
  4E  65       low order of jump address and
  4F  AF  PLO  put into RF.0
 0250 DF  SEP Set program counter = RF.
```

Control will now jump to 0165 with RF as the program counter. At the jump-to point, it will be necessary to repeat the above routine setting R3 equal to the appropriate address, and restoring it with a D3 instruction as the program counter. A more complex answer would be to use a subroutine to manage the exchange of program counters, and control the long jumps.

Also, you could turn off the video, perform the long branch, then turn it back on at the jump-to point; or just use the long branch instructions, though the screen will flicker each time the instruction is encountered as it takes three machine cycles to execute, and messes up the video timing.

A question I would really like to see answered concerning the use of a higher resolution interrupt routine for a Chip-8 program display, is: How can the X Y coordinates be modified to allow points to be continued below the bottom of the first page displayed on the screen?

In other words, I have written a two page interrupt routine including subroutines used by the Chip-8 interpreter. All my programs operate fine in the higher resolution, but only in the top half of the display! Obviously some change to the high order portion of the display address must be made, but where can this be handled in the interpreter? Last month's breakdown of the interpreter gives a start, but the answer to my question still floats out of my reach.

Any help would be appreciated. And, if any of the above seems worthwhile to include in the VIPER, you have my permission to print it. Thanks.

Tom Swan
San Antonio, TX

Tom:
You sure have done some interesting things with your VIP! Please do send us your editor. Hope you enjoy the variety of other editors in this issue. We'd also like to print your games. Othello will be in the new RCA game book. RCA is planning several new video chips, but none are pin-compatible with the 1861. The long-branch problem you mention is what makes it so difficult to interface Pittman's Tiny BASIC to the VIP - it's full of LBR's. I hope the two-page display in the last issue helped you.

A TEXT EDITOR FOR THE VIP Part Two

by Don Stein

This installment will describe, in general terms, the software for my text editor. Space does not permit the reproduction of all of the flow charts and detailed program listings for all of the subroutines, which number almost 50. However, I have no proprietary interest in the text editor, so that if there is sufficient interest among readers, I would be happy to work out with the editor of VIPER an arrangement to make full details available.

Operating System

The first step was to select an operating system to run the text editor. I was greatly tempted to use CHIP-8, because of the ease of programming. However, I finally settled on machine-language programming, to be run under my own operating system. This was necessary because CHIP-8 is too wasteful of precious memory space for this sort of application, and because several of the subroutines would have to be written in machine language, anyway.

The operating system would not be limited to the text editor; would be capable of future expansion; and would be easily modifiable. With these goals in mind, I set to work.

Before writing programs, several decisions had to be made. The first was to write everything in small, easily-modifiable modules (this technique is called "structured programming"). The second was to use the "SCRT" subroutine call and return method described on pages 61 to 64 of the COSMAC User Manual, because it was the most flexible and would permit jumping around the memory space at will.

In order to conserve precious registers, the VIP timer was modified slightly so that R8 was decrimented each 1/60 second, only. The separate timer for the tone generator was deleted.

The operating system would use two stacks: a data stack, and a subroutine linkage stack. (A stack is a dedicated memory area with associated pointer, following the "last-in-first-out" protocol.) Since the display area would require a full 1K of memory, the last page would not be suitable for the display area; therefore, the unused portion of the last page of memory was reserved for the two stacks.

Nine of the sixteen registers are required by the operating system; the register assignments are shown in Table 1.

When the VIP switch is set to "RUN", an operating-system executive routine initializes the registers, then waits for the operator to key in (using the hex keyboard) the address of the program to be executed. For the text editor, this address is 0400.

Text Editor

The text editor uses a data field which is separate from the display field. The characters to be displayed are stored in ASCII format in the data field, and then are translated to the necessary bit patterns for displays; the display bit patterns are stored in the display field. This arrangement permits complete separation between the data manipulation operations such as insert and delete, and the data display operations; this makes programming a lot easier and more flexible, and permits simple up- and down- scrolling operations.

Because my VIP has only 3K of memory, I set up a data field of two pages. There is nothing magic about this size -- it can be larger or smaller. A two page data field will fill approximately three screens, using the 11-line by 16-character format described in part one of the article.

At present, I am using an underline cursor, which is not completely satisfactory. This is because it was easy to program, but there is no room under the bottom row of characters to display the underline. If desired, one could use a reverse-field cursor instead.

Four of the registers are used for text-editor pointers; the remaining three registers are available as working registers. The text-editor pointers consist of a pointer to the place in the display field where one is working; a pointer to the position of the data field which corresponds to the first character currently being displayed; an offset pointer to the data field which tells how many characters into the current display one is working; and a pointer to the software character generator. These pointers were slected to make programming easy.

Control Codes

The ASCII character set is divided into 32 control characters and 96 display characters. The control characters begin with '000' or '001'.

The ASCII control characters were not designed for text editing, so I just plunged ahead and assigned them to the functions I needed in an arbitrary function. Since the control-character-interpreter subroutine is table-driven, these assignments can be changed at will.

Four control characters, A, S, W, and Z, form a quadrant on the left side of the keyboard, right next to the 'control' key. Therefore, I used them for cursor movement (A for left, W for up, etc.).

A listing of control code assignments is shown in Table 2. All of the listed control codes have been implemented in the text editor.

Character Display

It was necessary to convert each ASCII code (which was not a control character) to a pattern of dots to be displayed. For this purpose, I coded each character into a 4x8 pattern, the right-hand dots of which were always zeros, thereby providing a blank area to the right or each character to separate one character from the next. Thus, four bytes would be required to store the display pattern for each character.

Eliminating lower-case characters, there are precisely 64 remaining ASCII characters. Thus, exactly one page of memory (256 bytes)is required to store the character generator.

The text editor converts the binary ASCII codes to relative addresses in the character-generator memory page, through a complex (but efficient) series of binary shifts, ands, and ors. The display bits stored at these locations then are unpacked and stored in the display field.

Whenever the data field is modified, the text editor automatically corrects the display field accordingly.

Memory Map

The memory map thus breaks down as follows:

Operating system -- page 0
Text editor programs -- pages 1, 3, and 4
Character generator -- page 2
Data field -- pages 5 and 6
Display field -- pages 7, 8, 9, and A
Stacks -- top of page B
(note: the VIP operating system uses the bottom of page B)

This map, of course, is for a 3K memory. In a larger memory, some of the above fields may be shifted.

Next Installment

The next installment of this article will describe the cassette tape interfaces used by the text editor.

TABLE 1 -- REGISTER ASSIGNMENTS

Operating System Registers

RO	DMA pointer
R1	Interrupt subroutine pointer
R2	Data stack pointer
R3	Program counter
R4	Call subroutine program counter
R5	Return subroutine program counter
R6	Subroutine linkage stack pointer
R7	Display page pointer
R8	Clock (free-running, decrementing)

Test Editor Registers

RA	Display field pointer
RB	Data field base pointer
RC.0	Data field offset pointer
RC.1	Even-odd flag (not explained in article)
RD	Character generator pointer

Working Registers

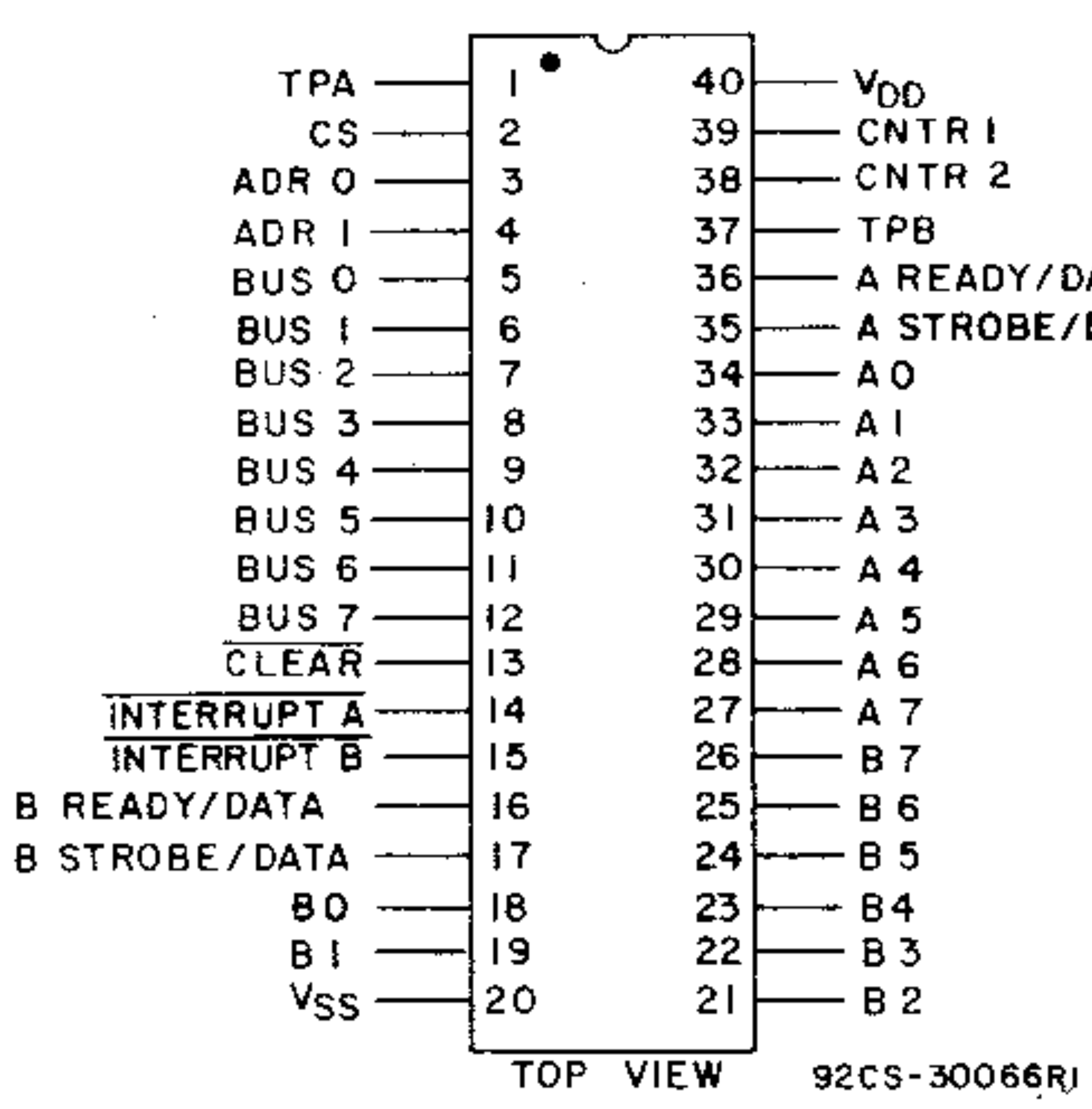
R9
RE
RF

TABLE 2 -- CONTROL CODE ASSIGNMENTS

Up cursor	control	W	(see text)
Down cursor (new line)	"	Z	" "
Left cursor (backspace)	"	A	" "
Right cursor	"	S	" "
Clear screen & home cursor	"	C	
Home cursor	"	H	
Carriage return	"	M	(separate key on keyboard)
Scroll up	"	U	
Scroll down	"	D	
Advance to next page	"	P	
Back up one page	"	B	
Insert line	"	L	
Delete line	"	X	
Input from tape	"	I	
Output to tape	"	O	
Tape unit control	"	T	(explained in the next part of article)

Here are two new 1800-family chips just announced by RCA

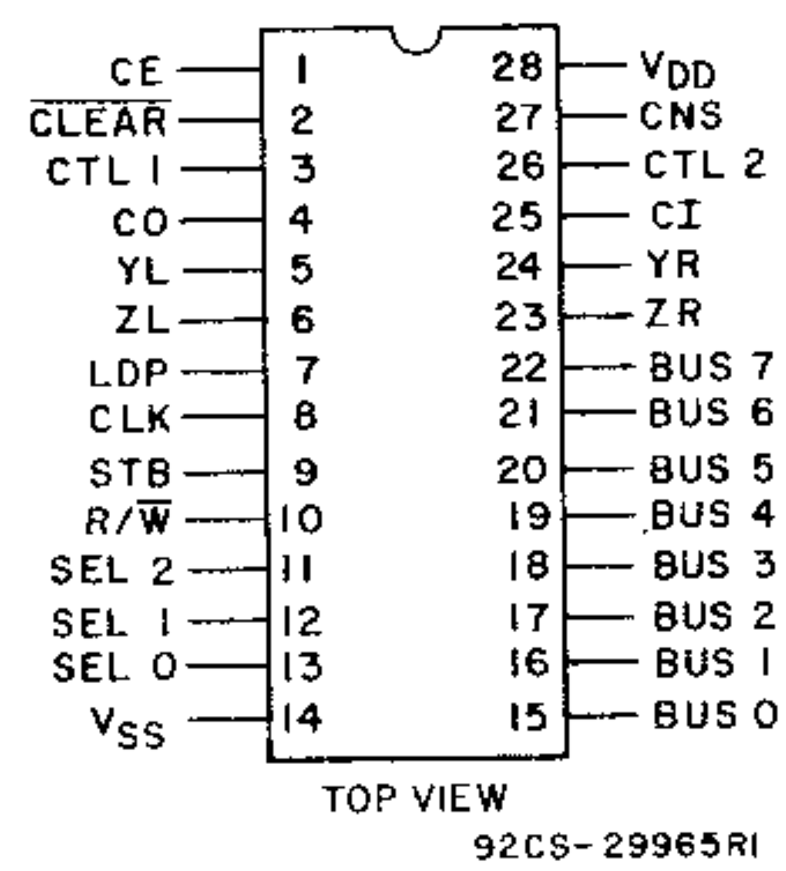
NEW PROGRAMMABLE I/O INTERFACE CDP1851*



Silicon-gate CMOS circuitry
Single voltage supply
Operates in I/O or memory spaces
20 programmable I/O lines
Programmable to operate in 1 of 4 modes
Input
Output
Bidirectional
Bit programmable

* Available fourth quarter 1978.

NEW MULTIPLY, DIVIDE UNIT CDP1855*



Low-power static CMOS
Performs binary multiplication or division under microprocessor control
8-bit by 8-bit multiply or divide in 2.5 μs at 10 V
Interfaces directly with CDP1802
Single supply voltage, 3-12 V
Cascadable up to 4 units for 32-bit by 32-bit = 64-bit operation
May be used as an I/O or memory device
Full temperature range (-55 to +125°C)

* Available fourth quarter 1978.

Reproduced from the new COSMAC Microprocessor Product Guide (MPG-180B) available for $.50 from RCA Solid State, Box 3200, Somerville, NJ 08876.

NON VIDEO OPERATING SYSTEM

by Joseph Czajkowski

INTRODUCTION

Wouldn't it be nice if you could see the VIP operating system data on Hexadecimal LED displays rather than on your bulky TV set? Here's an operating system that simulates the VIP operating system with hex displays and adds some other extra features. The software fits in less than one page of memory and is easily modified with the instructions included here to be the way you want it. The hardware is simple enough to be constructed by persons who have completed a few small projects on their own. The hardware can be also tailored to suit the user's requirements. In simple terms, the key is flexibility, you get just what you want.

Features of the operating system compare with those of the VIP operating system. One additional command, Execute, has been added, as well as automatic control of the recorder motor and display of the mode of operation. Additionally you can "backstep" one byte to correct an error in the memory write mode and also restart the operating system in any other program mode.

OPERATION

Following is a description of the operation of each command:

MEMORY WRITE - Key 0 - Operation is the same as the VIP Memory Write except for the "backstep" feature. If you make a mistake keying in a byte, press the Restart button and hold it while you reenter the byte. This avoids having to reset, run, rekey in the address, select memory write, and then rekey the byte.

MEMORY READ - Key A - Same as the VIP Routine. Each key depression steps through memory. This routine also uses the RESTART Feature. Press and hold the button, and then press any hex key and the operating system will restart.

TAPE READ - Key B - Same as the VIP Routine. 1-15 pages of memory can be read in off a tape which was written by the original VIP operating system or by the TAPE WRITE routine in this operating system - compatability is retained. The relay will turn on the tape recorder, read in the tape and shut off the recorder automatically. Pressing RESTART at the end of the routine will restart the operating system.

TAPE WRITE - Key F - Same as the VIP routine. 1-15 pages can be recorded on tape. VIP compatibility is retained. The recorder is turned on and off by the relay automatically. At the end of the routine, pressing the RESTART button will return to the start of the operating system.

EXECUTE - Key E - Pressing key E after keying in an address will execute a program starting at that address. Execution begins with P=0 and X=0.

SOFTWARE MODIFICATIONS

The following guidelines will allow you to modify the software of the operating system. Some modifications are not necessary if you choose not to include their hardware partners. This includes all Restart commands and the Backstep feature in memory write mode. An interesting modification allows you to tape, write and read more than the $0F_{16}$ pages allowed by the VIP (If You Have The Memory).

The high starting address of the program is initially set for page 06. This is at 01 in the program. Changing this byte can relocate the program in any page of memory.

The stack is located at 07F0 in the initial program. Changing the bytes at 08 and 0B respectively, will change the high and low address of the stack.

The program is started at 00 of the selected page; any program counter can enter the program, which sets its own parameters.

MEMORY WRITE - To eliminate the "backstep" feature, substitute the following instruction -

Address	Instruction
5F	C4

MEMORY READ - To eliminate the RESTART feature, substitute the following instruction in the program -

Address	Instruction
68	69

TAPE READ - The program can be modified to read in $1\text{-}255_{10}$ pages of memory, however, other VIPs without this program will not be able to read in more than the 15 pages that the VIP is limited to. Keep this in mind if you plan to exchange long programs. To obtain this expansion substitute the following instruction in the program -

Address	Instruction
6C	D5

The program now requires a two key depression for the number of pages to read.

The RESTART feature can be eliminated by substituting the following instructions in the program -

Address	Instruction
87	C4
88	C4

The program will enter an endless loop after doing a tape read.

The relay control of the recorder can be eliminated by substituting the following instruction in the program -

Address	Instruction
75	OB

TAPE WRITE - The program can be modified to write $1\text{-}255_{10}$ pages of memory to tape. VIPs without this program will not be able to read more than 15 pages of memory written by this program. To obtain this expansion substitute the following instruction in the program -

Address	Instruction
91	D5

A two key depression is now required for the number of pages to write.

The RESTART feature can be eliminated by substituting the following instructions in the program -

Address	Instruction
AC	C4
AD	C4

At the end of a tape write the program will enter an endless loop.

The relay control of the recorder can be eliminated by substituting the following instruction in the program -

Address	Instruction
9A	OF

A Special Note Concerning Tape Quality:

Quality tapes and a good recorder are necessary for error-free storage of programs on the Standard VIP. With the extended number of pages, you may now store up to 16_{10} times more data. It is imperative that you use the best tape you can find. Follow the suggestions on page 32 of the VIP manual concerning recording guidelines - it will pay off in the long run.

HARDWARE

For basic operation, only the circuitry in figure 1 is necessary, i.e. the Hexadecimel displays. If you so desire, the current operating system mode (tape read, execute, etc.) can be displayed using the circuit in figure 2. Figure 3 shows a 1 bit latch, which, when used with the relay circuit in figure 4 (or a circuit of your own), will control the operation of the recorder's motor. The circuit in figure 4 allows manual override of the recorder control. (Adapted from Viper, Volume 1, Issue 1). Figure 5 shows a power supply which is recommended for use with the circuitry above since the VIP power supply cannot handle the extra load these circuits provide. Be sure to attach the ground on the VIP to the ground on every piece of the above mentioned circuitry. Never tie the +5 output from the power supply in figure 5 to the +5 power buss of the VIP if another supply is attached to the VIP; SEVERE CIRCUIT DAMAGE may result.

The type of Hexidecimal LED display can be selected by the user. The author used HP 50 82-7340 which are quite expensive in unit quantities and may not be readily available in all areas. Yet, they are quite impressive with their dot matrix format. TI 311 Hex displays can be substituted with appropriate wiring charges. Additional information concerning Hexadecimal displays for computer readout can be found in the April 1978/Issue #16 Kilobaud on page 104; "Displaying Hexadecimal" by Dave Maciorowiski. The article shows many other alternatives to those presented here, quite economically. The RESTART button is also the BACKSTEP button. Its hook up is shown in figure 6.

CONCLUSION

The hardware added to the VIP is not limited to just the operating system. Many uses can be found with a little ingenuity; such as:

- Use the relay as a telephone dialer.
- Use the relay as the output of an electronic lock.
- Use the displays as a programable digital clock.
- Use the displays as a timer.
- Use the displays in game applications.

All of the above need software support which can be provided by the user; work slowly at first, set your goals, develop your program and you will be on your way.

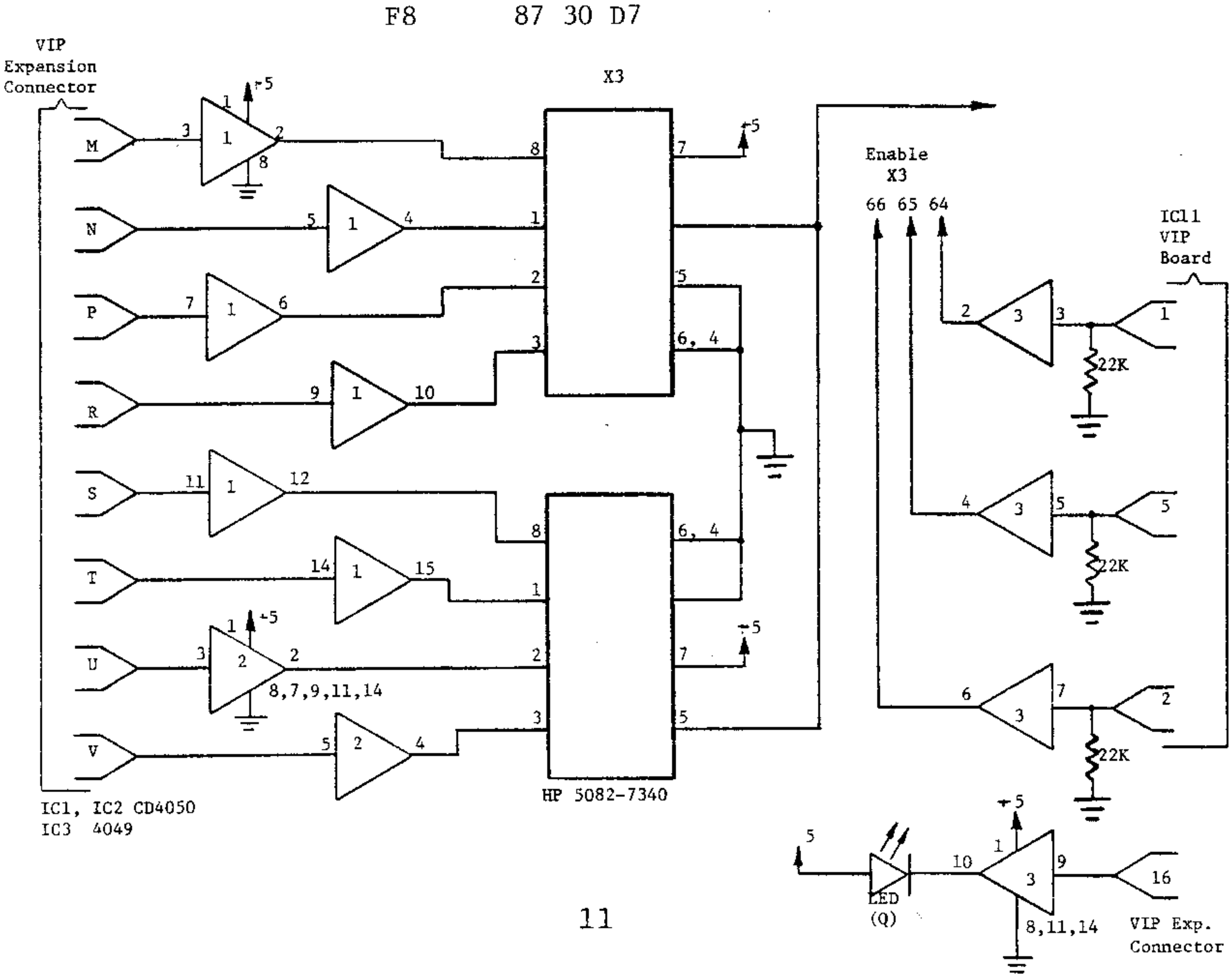
So, there you have it - VIP operating system basics - improved with Hexadecimal readout on LED displays and other hardware add-ons. I hope that with this article your usefulness and enjoyment derived from the VIP is increased. In the future, more articles using the Hexadecimal displays are planned - watch for them!

****

Editor's Note:
Several subscribers have written The VIPER mentioning that they found it difficult to read the code on the left edge of pages 18-25 of the #2 issue. This was due to a poor original, not bad printing. You can 'ressurect' this code in your copy by using the VIP monitor to step through the ROM code in your system and copy down the illegible entries. Sorry for the problem!

PROGRAM LISTING

Address								
00	F8	06	B4	F8	07	A4	D4	F8
08	07	B2	F8	FO	A2	E2	F8	05
10	52	67	22	94	B5	BA	BC	BD
18	F8	81	B1	F8	46	A1	F8	80
20	B3	F8	CO	A5	F8	D9	AA	F8
28	B1	AC	F8	CF	AD	D5	B6	52
30	64	22	D5	A6	52	65	22	06
38	52	66	22	DD	AE	52	67	22
40	32	5C	FB	01	32	07	8E	FB
48	OA	32	65	8E	FB	OB	32	6C
50	8E	FB	OE	32	8B	8E	FB	OF
58	32	91	30	3B	D5	3F	60	26
60	56	DC	16	30	5C	DD	DC	37
68	07	16	30	65	DD	AE	F8	C2
70	A3	F8	81	BC	F8	1B	52	67
78	22	D3	61	22	F8	B1	AC	94
80	BC	DC	F8	OB	52	67	22	37
88	07	30	87	96	BO	86	AO	EO
90	DO	DD	AE	F8	91	A3	F8	81
98	BC	F8	1F	52	67	22	D3	61
AO	22	F8	B1	AC	94	BC	DC	F8
A8	OF	52	67	22	37	07	30	AC
BO	D4	96	52	64	22	86	52	65
B8	22	06	52	66	22	30	BO	D4
CO	84	A4	DA	FE	FE	FE	FE	AB
C8	DA	52	8B	F1	30	BF	D4	F8
DO	00	A8	DA	1A	87	30	CE	DD
D8	D5	F8	10	A7	27	87	52	62
EO	22	36	E7	32	D9	30	DC	7B
E8	F8	12	B9	29	99	3A	EB	36
FO	E8	7A	88	32	F8	87	30	D8
F8	87	30	D7					



Function Display

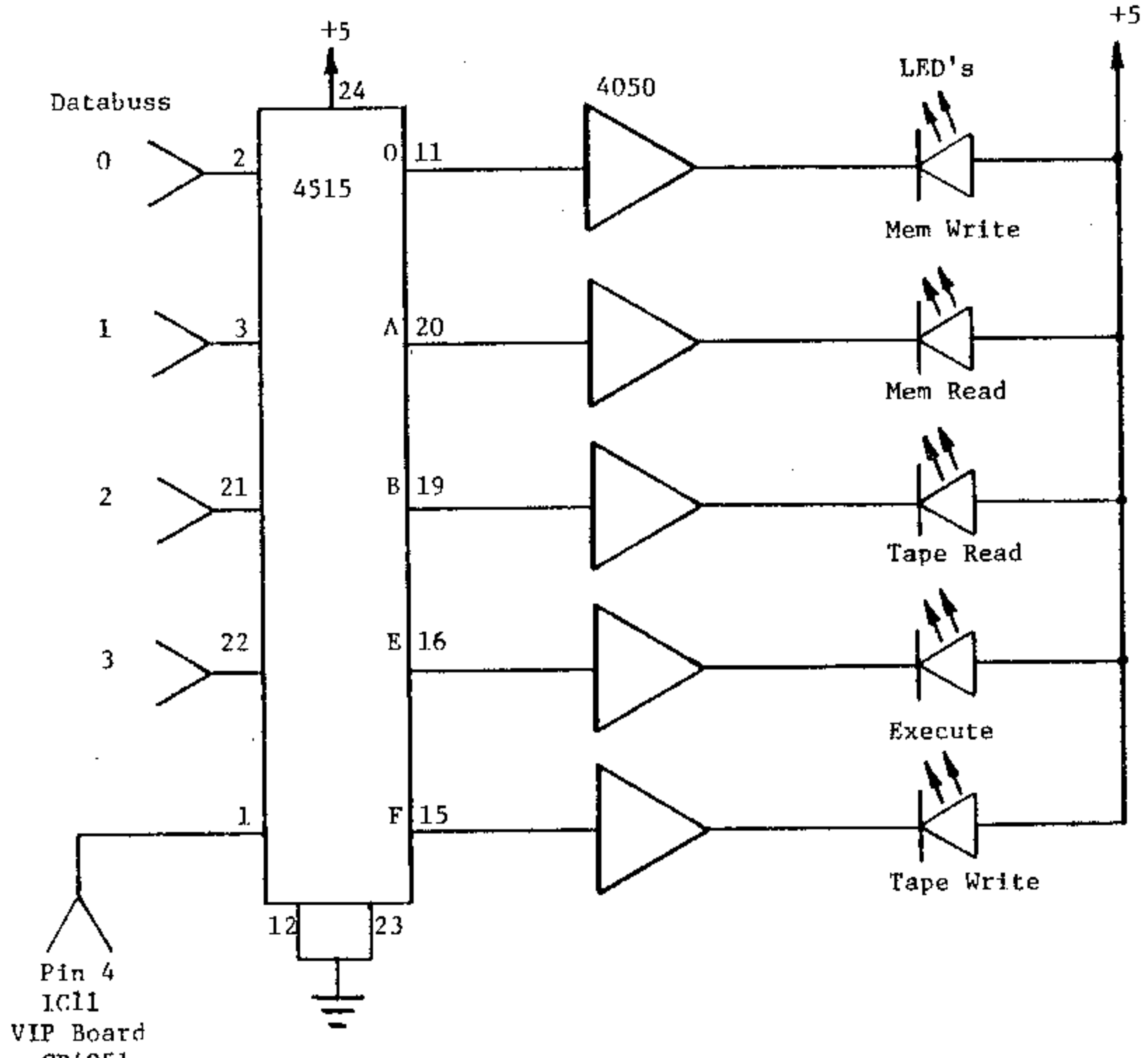


Figure 2

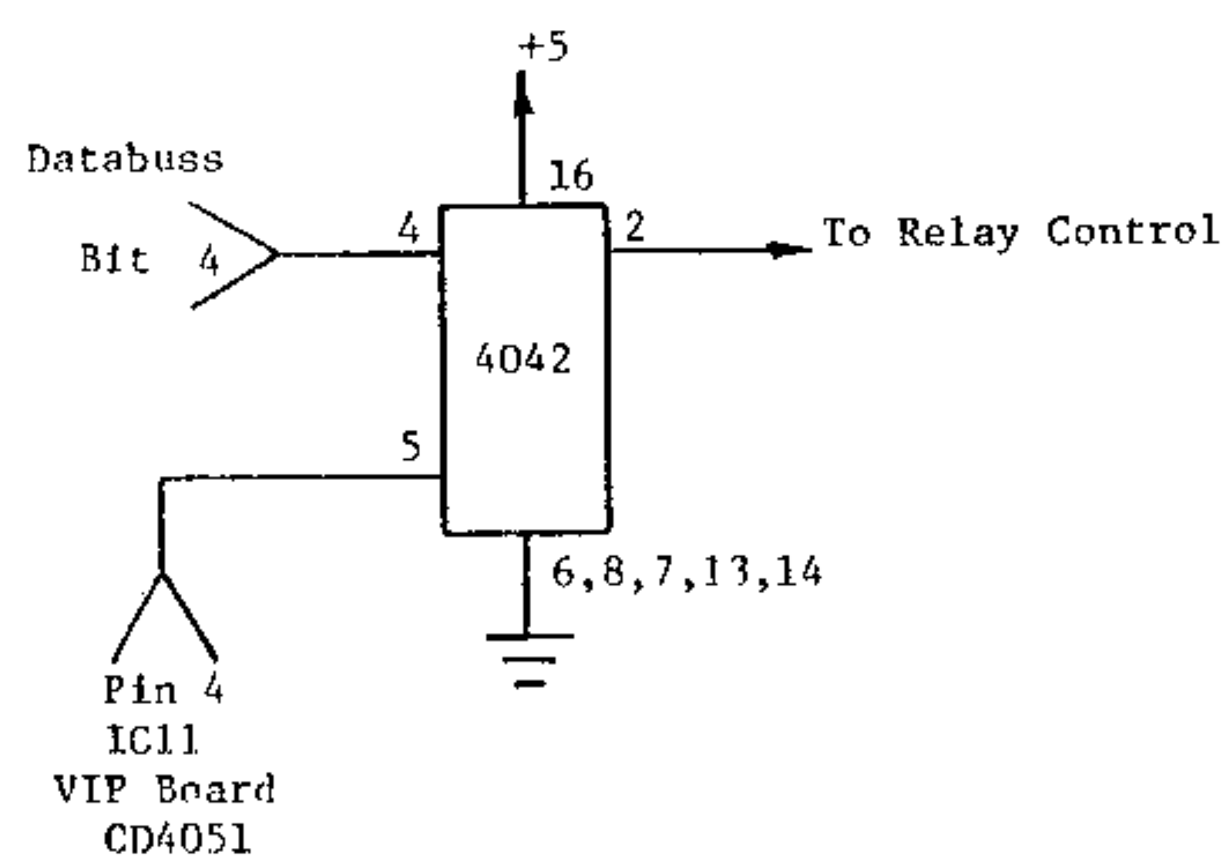


Figure 3

Relay Control

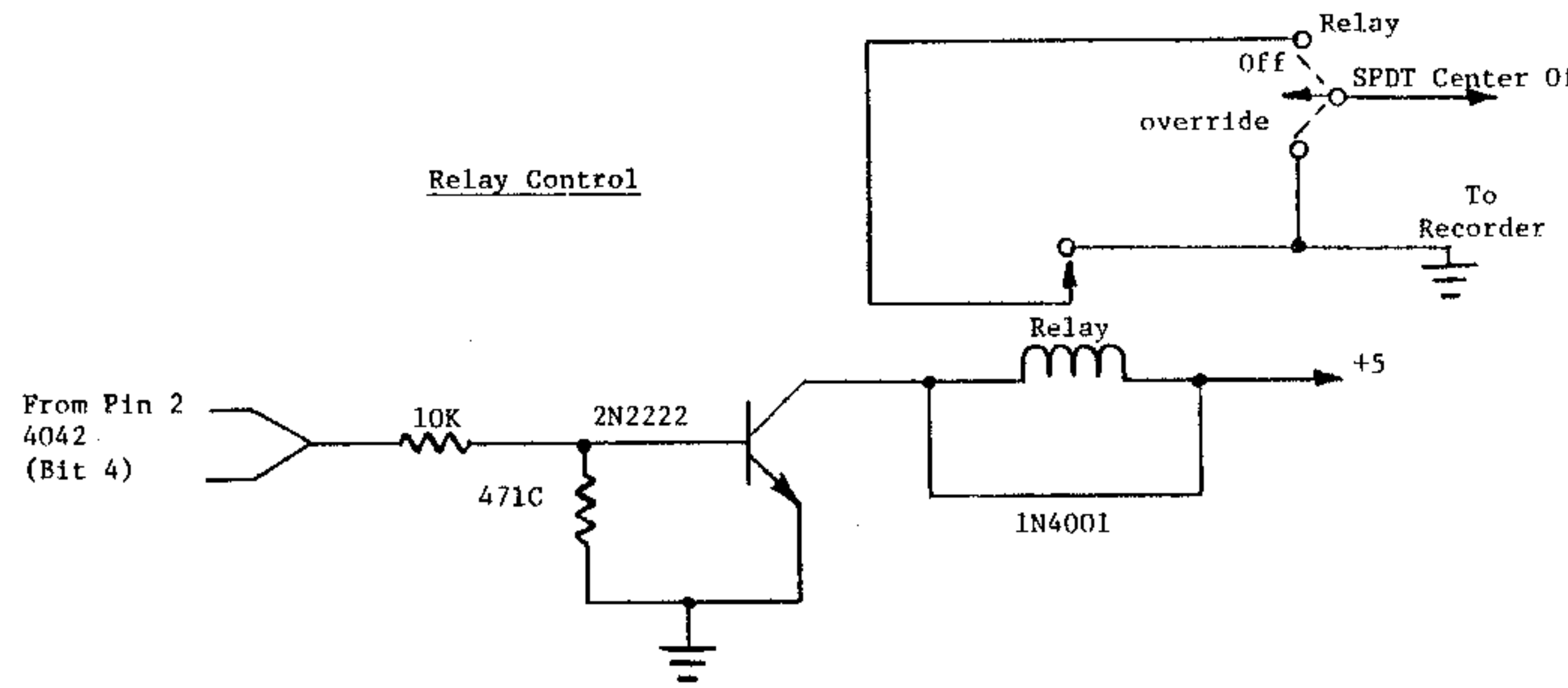


Figure 4

Power Supply

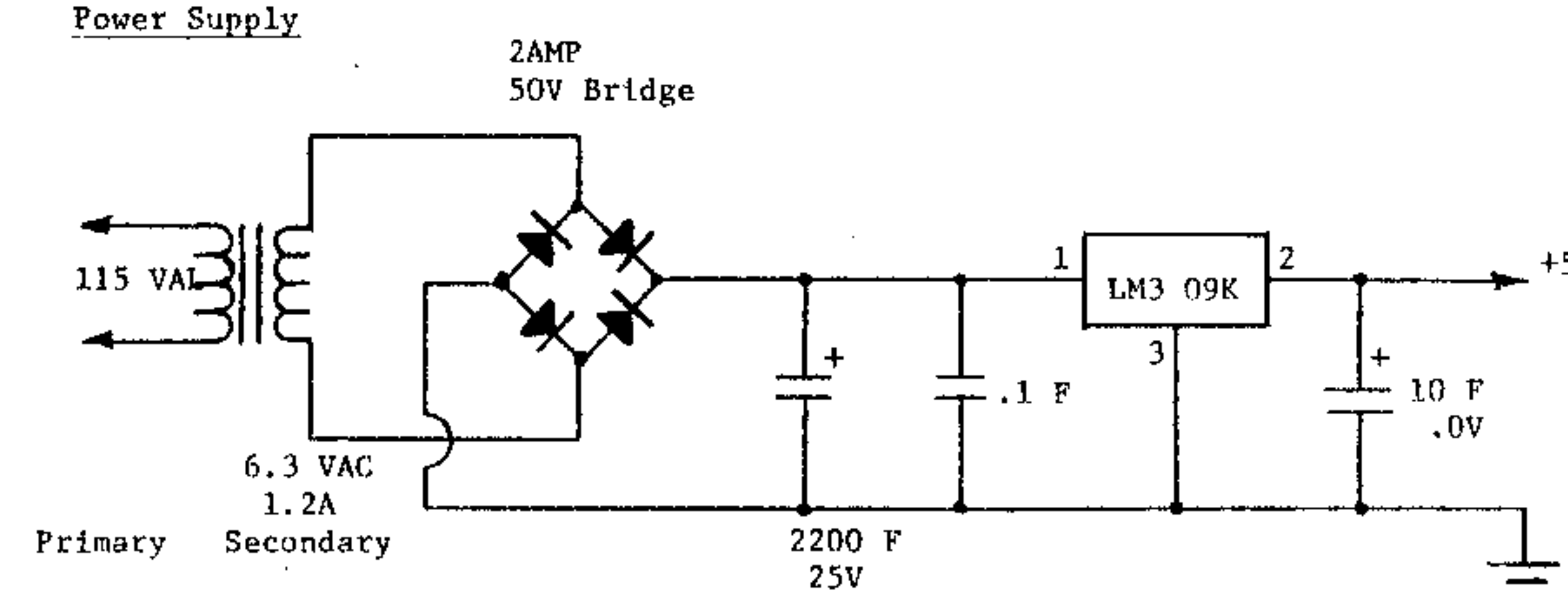


Figure 5

Restart & Backstep

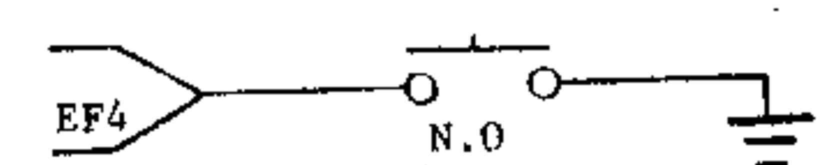


Figure 6

A VIP Breakpoint and Register Display System

W. A. Barrett

The basic VIP kit provides some very nice tools for filling and displaying memory, one byte at a time. Unfortunately, it does not provide much in the way of register display. About the only way a machine-language program can be debugged is through a more expensive development station, or through a simulator. The operating system does show the contents of registers 3 through F, but it is necessary to plant a BR * somewhere, start the program, reset, then start the operating system. You have one chance to read off the register values, by halves.

The system described in this paper can change that. Three pages of RAM memory are needed to display all the registers, including X, P, Q, DF and D. A breakpoint is easily inserted in the program, and one may step through a program in some detail using the breakpoint system as a tool for following the register assignments.

How to use it

The first program instruction should be a LBR DEBUG (C0 500), in location 0000. Control will return to location 3, with initial conditions as explained later.

A breakpoint is triggered by an IDL (00) instruction. Your program will be periodically interrupted, however, a special interrupt program will examine the instruction interrupted and resume execution if the instruction is not IDL. If it is an IDL, the breakpoint system is entered, yielding a register display and an opportunity to change certain memory locations, including the breakpoint location.

Upon hitting an IDL instruction, a register display will appear on the TV screen, as follows:

X,P,D	Q,DF,Q,DF
R0	R1
R2	R3
R4	R5
R6	R7

Q and DF are repeated to make the display come out on even register boundaries. Key 1 will yield the rest of the registers:

R8	R9
RA	RB
RC	RD
RE	RF

Successive pushes of key 1 will display the register pages, alternately.

Whenever the register display is on screen, several choices are available:

Key B causes a resume of the interrupted program, just after the IDL instruction.

Key F causes a resume of the interrupted program, ON the IDL instruction. The instruction presumably has been changed to something else, otherwise another breakpoint at the same location will occur immediately.

Key A (MR) permits displaying memory. Enter a four character address X, then the TV will display X and M(X). Push A to increment X and display successive locations. Push 1 to return to register display mode.

Key 0 (MW) permits altering selected memory locations. Enter a four character address X, then a two character byte value, B. B will be stored into M(X), and the TV will display X, B. Push Key 0 again to repeat this operation--enter an address, then a byte, etc. Push key 1 to return to register display mode.

If the code under test has no IDL instructions written into it, then an instruction must be replaced with an IDL. When the breakpoint is hit, check its location through P and R(P). R(P) will point to the location FOLLOWING the IDL. Use the MW feature to change the IDL back to the original instruction, then push Key F to resume on the original instruction.

If the code has IDL instructions written in, then key B will resume correctly with no further ado.

More breakpoints may be entered through the MW feature at any time. Of course, the operating system services may also be invoked, but upon resuming from the operating system, control passes to the origin.

This code assumes that the breakpoint software goes into pages 5 and 6. The top of page 4 is used for the stack. Page 7 is used for the TV display area. Your program goes in pages 0 through 4, with the top of page 4 allocated to the stack.

Changing register values

Any of the register values may be changed during a break by finding its stored value in memory. The displayed R2 value points to one byte above the X,P byte, and the other registers are stored in memory below these, in the same order as displayed. Thus M(R2-1) carries X,P, M(R2-2) carries D, etc. Upon a resume, the registers will carry the values displayed, whatever they are. However--don't change R1, and be sure the stack conventions are observed if you expect debug to work after resumption.

Endless loops

If your program is caught in an endless loop, you may interrupt it by pushing Key C. This will break out at some location and go into the usual register display, from which you can determine the nature of the loop, or plant more breakpoints.

Stack conventions

Register R(2) must point to a stack top at all times. An interrupt can potentially occur anywhere within the program; it first decrements R(2), then writes information in this location, and in lower locations. Upon resuming the interrupt, the stack pointer is restored to its original position, but ANY INFORMATION IN ADDRESSES LESS THAN R(2) are subject to being overwritten randomly.

Limitations

Don't try to debug programs that use the TV interface, or that make use of program interrupts or the DMA system. Don't use register 0 as a program counter. However, its value is preserved by the debug system, if used for anything but the PC.

Be sure that the stack conventions are satisfied at all times. Any bytes below R2 are subject to being changed by a debug interrupt at any time.

Your program will run about half as fast under debug, even when no breakpoints are being serviced, since it is interrupted once every TV frame (about 60 times/second), and held in debug during the TV display interval (about 1/120 second).

Don't use register R1 for anything. It holds an interrupt routine address. If changed, the debug system will fail.

Locations '500 through '7FF are used by the debug system. Page 7 ('700-'7FF) is used for a TV display, and may be shared with debug if you don't mind seeing your display wiped out on a breakpoint. The stack starts at '4FF and grows downward. Debug needs about 10 bytes of the stack to do its thing.

Don't try to place the debug program in ROM. It alters certain of its own instructions.

The debug system uses some features of the operating system, so don't scrap it and expect this to work.

Initial conditions

Your program must have a LBR DEBUG (C0 500) in location 0. Debug returns control to location 3, with the following initial conditions: PC=3, X=2, IE=1, TV is on, R1=trap address, R2='4FF, R8.H=top memory page, Q=0, DF=1.

How it works

The initial LBR DEBUG enables interrupts with a special interrupt procedure, and turns on the TV. It also initializes the stack pointer R2 to 04FF.

Upon an interrupt, the special interrupt routine examines the instruction interrupted and resumes immediately if not IDL. If it is IDL, all the registers are saved in the stack, a display and memory alteration routine is entered. Upon a resume, the registers are restored, with attention paid to restoring the original P register correctly, then a return executed.

Every interrupt is serviced by the debug system. An interrupt occurs just before a series of DMA bursts. If a normal return is needed, the debug system holds off a return until the end of the DMA bursts. This is essential to protect program IDL instructions from being skipped over by a DMA burst. Any return from debug, and the initial entry into the program is synchronized to the end of the DMA bursts to avoid missing any IDL's. The IDL instructions must be detected by an interrupt, not a DMA.

The Machine Language Program

This is designed for the basic system with minimum memory, the one I have:

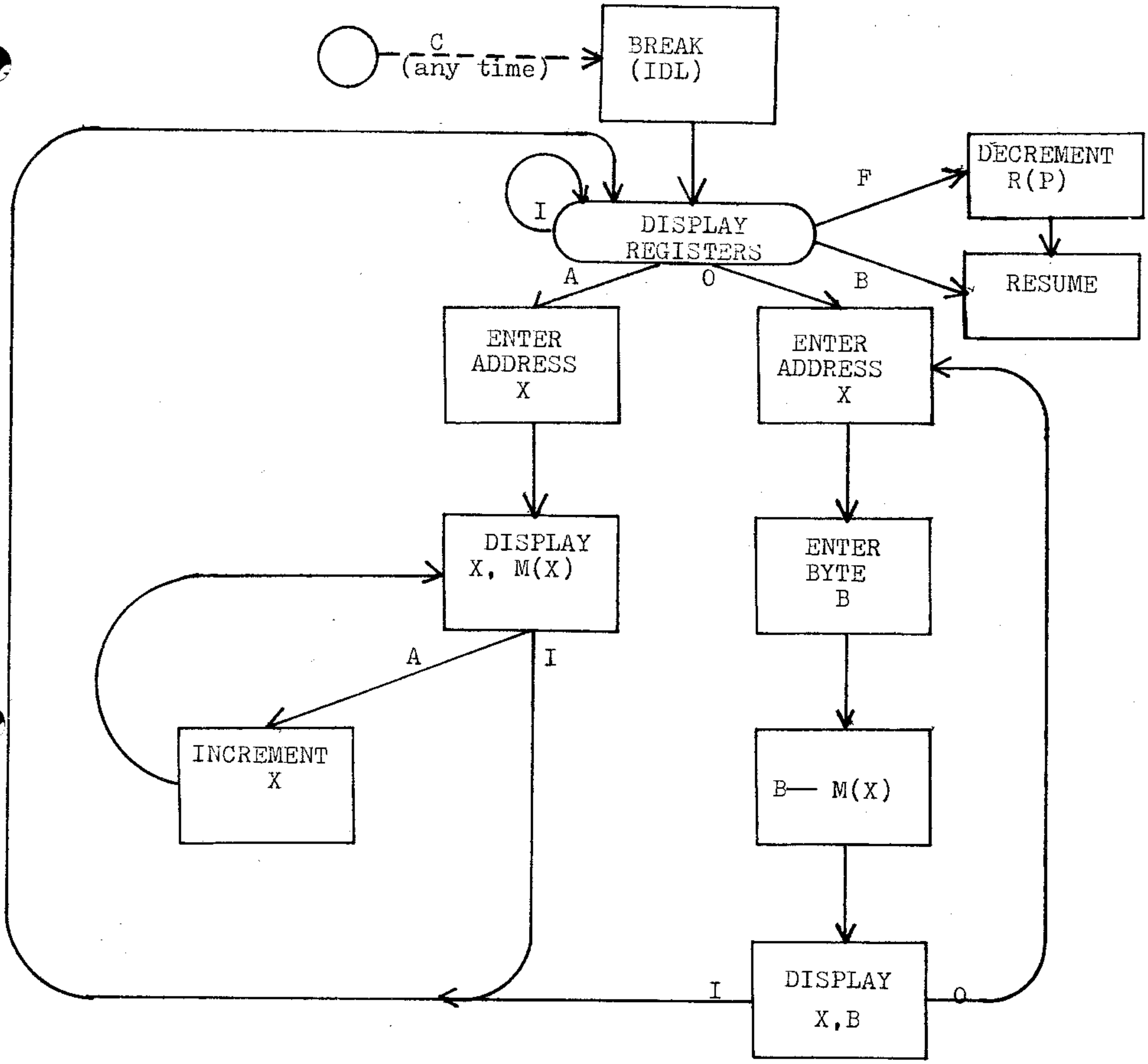
```
0500  90 BB F8 08 A3 E0 71 BB      0580  FC 07 A0 92 7C 00 B0 40
0508  91 BB F8 68 A1 F8 05 B1      0588  FA 0F 32 8A F9 40 73 FF
0510  F8 FF A2 F8 04 B2 69 00      0590  20 52 D2 22 32 9D F8 0C
0518  3C 18 34 1A 70 23 C0 00      0598  52 62 22 3E 55 91 73 F8
0520  03 E3 71 23 F8 2B A1 F8      05A0  68 73 82 FC 09 A0 92 7C
0528  05 B1 D1 E2 60 F8 05 BB      05A8  00 73 80 73 98 73 83 73
0530  F8 B8 A3 F8 AF 53 FE 72      05B0  91 83 7E F8 B9 A3 F8 94
0538  00 03 F8 B4 32 4A 03 BB      05B8  53 00 73 03 33 C7 FC F0
0540  46 FC 10 53 30 B7 FE 11      05C0  53 FB A0 32 CB 30 B9 FF
0548  30 43 72 A3 72 B3 60 60      05C8  EF 30 C0 A8 A3 F8 06 B3
0550  60 3C 51 34 53 69 BC 56      05D0  D3 03 9B BD F8 00 AA F8
0558  34 58 60 72 A0 72 B0 60      05D8  FF AD ED 8A 73 8D 3A DB
0560  72 F6 7A 32 66 7B 72 70      05E0  5D E7 F8 C6 A5 F0 F5 F6
0568  22 61 22 78 22 E2 73 F8      05E8  F5 F6 D5 F0 27 FA 0F D5
0570  00 39 75 F8 08 7E 73 73      05F0  2E 8E 32 D1 8D FA 07 3A
0578  9D 73 80 73 F8 01 73 82      05F8  E5 8D FC 28 AD 30 E5 00
```

0600	F8	06	B1	BC	F8	C6	A1	F8
0608	07	B3	F8	D2	A4	F8	05	B4
0610	F8	B8	AC	F8	95	A6	F8	81
0618	B6	B5	3A	69	E3	70	23	82
0620	FC	24	A7	92	7C	00	B7	E2
0628	F8	24	A9	89	FF	14	3B	34
0630	F8	14	30	35	89	AE	D4	D5
0638	FB	01	3A	44	89	FF	14	A9
0640	BB	1F	30	2B	02	32	77	FF
0648	0B	32	74	FF	FF	32	77	FF
0650	05	3A	37	82	FC	24	A7	92
0658	7C	00	B7	07	FA	0F	FE	FC
0660	05	52	87	E2	F7	A7	97	7F
0668	00	B7	E7	72	AC	F0	BC	2C
0670	9C	73	8C	73	C0	05	21	E2
0678	82	A7	92	B7	22	DC	DC	0C
0680	B9	DC	DC	DC	A9	07	27	32
0688	A0	E2	09	73	F8	03	AE	D4
0690	D6	FB	01	32	B1	12	19	17
0698	17	89	57	17	99	57	30	89
06A0	DC	DC	DC	59	F8	03	AE	D4
06A8	12	12	12	12	D6	32	77	30
06B0	1F	12	12	12	12	30	1F	D3
06B8	D5	FE	FE	FE	FE	AE	D6	BE
06C0	F1	73	30	B7	42	70	22	78
06C8	22	52	C4	C4	C4	9B	B0	F8
06D0	00	A0	80	E2	E2	20	A0	E2
06D8	20	A0	E2	20	A0	3C	D2	7A
06E0	88	32	C4	7B	28	30	C4	

VIP Breakpoint System for 16 Page Memory

The conventions are essentially the same as for the 8 page system. The stack starts at 0CFF and grows downward. The top page, 0F00-0FFF is used for the TV display, and the program occupies most of 0D00-0EFF.

0D00	90	B3	F8	08	A3	E0	71	33
0D08	91	BB	F8	68	A1	F8	0D	B1
0D10	F8	FF	A2	F8	0C	B2	69	00
0D18	3C	18	34	1A	70	23	C0	00
0D20	03	E3	71	23	F8	2B	A1	F8
0D28	0D	B1	D1	E2	60	F8	0D	B3
0D30	F8	38	A3	F8	AF	53	FE	72
0D38	00	03	FB	B4	32	4A	03	3B
0D40	46	FC	10	53	30	37	FF	11
0D48	30	43	72	A3	72	B3	50	60
0D50	60	3C	51	34	53	59	3C	56
0D58	34	58	60	72	A0	72	B0	60
0D60	72	F6	7A	32	66	7B	72	70
0D68	22	61	22	78	22	E2	73	F8
0D70	00	39	75	F8	08	7E	73	73
0D78	90	73	80	73	F8	D1	73	82
0D80	FC	07	A0	92	7C	00	B0	40
0D88	FA	0F	32	8A	F9	40	73	FF
0D90	20	52	D2	22	32	9D	F8	0C
0D98	52	62	22	3E	55	91	73	F8
0DA0	68	73	82	FC	09	A0	92	7C
0DA8	00	73	80	73	93	73	83	73
0DB0	91	B3	7E	F8	B9	A3	F8	94
0DB8	53	00	73	03	33	C7	FC	F0
0DC0	53	FB	A0	32	CB	30	39	FF
0DC8	EF	30	C0	A8	A3	F8	0E	B3
0DD0	D3	D3	9B	BD	F8	00	AA	F8
0DD8	FF	AD	E0	8A	73	8D	3A	DB
0DE0	5D	E7	F8	C6	A5	F0	F5	F6
0DE8	F6	F5	D5	F0	27	FA	0F	D5
0DF0	2E	3E	32	D1	8D	FA	07	3A
0DF8	E5	8D	FC	28	A0	30	E5	00
0E00	F8	0E	B1	BC	F8	C6	A1	F8
0E08	0F	BB	F8	D2	A4	F8	0D	B4
0E10	F8	B8	AC	F8	95	A6	F8	81
0E18	B6	B5	BA	69	E3	70	23	82
0E20	FC	24	A7	92	7C	00	B7	E2
0E28	F8	24	A9	89	FF	14	3B	34
0E30	F8	14	30	35	89	AE	D4	D6
0E38	FB	01	3A	44	89	FF	14	A9
0E40	3B	1F	30	2B	02	32	77	FF
0E48	0B	32	74	FF	FF	32	77	FF
0E50	05	3A	37	82	FC	24	A7	92
0E58	7C	00	B7	07	FA	0F	FE	FC
0E60	05	52	87	E2	F7	A7	97	7F
0E68	00	B7	E7	72	AC	F0	BC	2C
0E70	9C	73	8C	73	C0	0D	21	E2
0E78	82	A7	92	B7	22	DC	DC	0C
0E80	B9	DC	DC	DC	A9	07	27	32
0E88	A0	E2	09	73	F8	03	AE	D4
0E90	D6	FB	01	32	B1	12	19	17
0E98	17	89	57	17	99	57	30	89
0EA0	DC	DC	DC	59	F8	03	AE	D4
0EA8	12	12	12	12	D6	32	77	30
0EB0	1F	12	12	12	12	30	1F	D3
0EB8	D5	FE	FE	FE	FE	AE	D6	BE
0EC0	F1	73	30	B7	42	70	22	78
0EC8	22	52	C4	C4	C4	9B	B0	F8
0ED0	00	A0	80	E2	E2	20	A0	E2
0ED8	20	A0	E2	20	A0	3C	D2	7A
0EE0	88	32	C4	7B	28	30	C4	



State Diagram - VIP Breakpoint System

Cosmac 1802 Editor

Sam Hersh
Stanford University

Editor reads and writes non-adjacent locations in ram without repeated use of the run switch, displays the contents of ten addresses simultaneously and starts program execution.

Program Operation:

Initially the editor is in command mode and will accept A, B, F, E, D and ignore all other entries:

'A'ddress enters a new address
'B'ack decrements address pointer by 2
'F'orward increments address pointer by 2
'E'nter writes a 2 byte instruction
'D'o begins executing at 0400*

*This address may be changed by modifying the Chip-8 instruction at 0280

Commands 'A' & 'E' place the editor into insertion mode.
2 bytes (4 hex digits) are then accepted for insertion.
An underlining cursor shows where insertion will occur.
Errors can only be corrected by reentry or by switching to reset and then back to run before finishing an insertion.
Of course, Chip-8 Variables (OYDO-OYFF) cannot be meaningfully examined as they are used by the editor.

Starting Address: 0200
Main Program: 0200-0281

Subroutines:

Vload	028E-02A8	Loads an address or instruction into V0-V3
Display	02AA-02C3	Displays V0-V3
Inst	02C4-02D1	Loads an instruction into 0204-5
Addr-2	02D2-02E1	Adds 2 to the address scratchpad
Addr-N	02E2-02F1	Subtracts 2 from the address scratchpad
Rewrite	02F2-030B	Enters a new address or instruction
E&D	0314-032B	Erases and displays a new digit
Pack	032C-0347	Packs contents of V0-V3 into 0202-3 or 0204-5

Reserved Ram:

Address Scratchpad	0202-3
Instruction Scratch	0204-5
Cursor	030C-0313

Cosmac Editor

	0200	1206	Go 0206	
	2	address		
	4	instruction		
5T:	6	00E0	Erase	
	8	6A00	VA=0	X-axis=0
	A	6B00	VB=0	Y-axis=0
	C	6C00	VC=0	line count=0
	E	02E2	Subr Addr-N	
	0210	0800	N=8	
Next Line:	2	028E	Subr Vload	V0:V3=address
	4	0203		
	6	22AA	Subr Display	display address
	8	02C4	Subr Inst	
	A	028E	Subr Vload	V0:V3=instruction
	C	0205 0205		
	E	22AA	Subr Display	display Chip-8 instruction
	0220	6A00	VA=0	⟨CR⟩
	2	7C01	VC=VC+1	line count + 1
	4	4C05	Skip YC=5	
	6	122E	Go Command:	get command
	8	7B06	VB=VB + 6	⟨LF⟩
	A	02D2	Subr Addr + 2	
	C	1212	Go Next Line:	
Command:	E	F40A	V4=Keypress	
	0230	340B	Skip V4=B	'B' ?
	2	123A	GO 023A	
	4	02E2	Subr Addr+N	
	6	0200	N=2	
	8	1206	Go 5T:	
	A	340F	Skip V4=F	'F' ?
	C	1242	GO 0242	
	E	02D2	Subr Addr + 2	(Ø2D2)
	0240	1206	Go 5T:	
	2	340A	Skip V4=A	'A' ?
	4	1252	Go E:	
	6	028E	Subr Vload	V0:V3=address
	8	0203		
	A	22F2	Subr Rewrite	address
	C	032C	Subr Pack	address
	E	0202		
	0250	1206	Go 5T:	
	2	340E	Skip V4=E	'E' ?
	4	127A	Go D:	
	6	02C4	Subr Inst	
	8	028E	Subr Vload	Instruction
	A	0205		
	C	7A19	VA=VA+19	point to 1st digit of instruction
	E	22F2	Subr Rewrite	instruction

Cosmac Editor

0260	0264	Subr 0264	
2	1272	Go 1272	
4	F8, 02		Machine code subr. which generates appropriate parameter for subr pack at 0274
6	BC, AC		
8	BD, F8		
A	74, AD		
C	4C, 5D		
E	1D, 4C		
0270	5D, D4		
2	032C	Subr Pack	instruction
4	xxxx		initially, may be anything
6	02D2 (0202)	Subr Add+2	(Ø2D2)
8	1206	Go 5T:	
A	340D	Skip V4=D	'D'?
C	1206	Go 5T:	
E	00E0	Erase	
0280	1400		execute starting at 0400

Subr VLoad

Loads the contents of the address parameter following the Vload call into V0:V3. Uses 0203 to load an address. Uses 0205 to load an instruction.

028E	45	LDA MR5	load 1st byte of parameter
	BC	PHI RC	
	45	LDA MR5	load 2nd byte of parameter
	AC	PLO RC	RC=address parameter
	96	GHI R6	
	C4 ← E0	NOP	
	BD	PHI RD	
	F8	LDI F3	
	F3		
	AD	PLO RD	RD=0EF3=address of Chip-8 variable V3 say A_1, A_2, A_3, A_4
	ED	SEX=D	
	0C	LDN MRC	
	73	STXD	V3=A_3A_4
	F6	SHR	
	F6	"	
	F6	"	
	F6	"	
	73	STXD	V2=$0A_3$
	2C	DEC RC	only the LSD will get displays
	0C	LDN MRC	
	73	STXD	V1=A_1A_2
	F6	SHR	
	F6	"	
	F6	"	
	F6	"	
	5D	STR MRD	V0=$0A_1$
	D4	Return	

Subr Display

```
02AA   F029      I=V0 (LSDP)
   C   DAB5      Show 5MI@ VA,VB
   E   7A05      VA=VA+5
02B0   F129
   2   DAB5
   4   7A05
   6   F229
   8   DAB5
   A   7A05
   C   F329
   E   DAB5
02C0   7A0A
   2   00EE      Return
```

Subr Rewrite

```
02F2   8D00      VD=V0          VD is Subr E & D parameter
   4   2314      Subr E&D
   6   80D0      V0=VD          restore V0
   8   8D10      VD=V1
   A   2314      Subr E&D
   C   81D0      V1=VD          restore V1
   E   8D20      VD=V2
0300   2314      Subr E&D
   2   82D0      V2=VD          restore V2
   4   8D30      VD=V3
   6   2314      Subr E&D
   8   83D0      V3=VD          restore V3
   A   00EE      Return
```

Subr E & D

```
        030C   0000
           E   0000      Cursor
        0310   0000
           2   6000
Entry:     4   A30C      I=030C          point to cursor
           6   DAB7      Show 7MI@VA,VB  show cursor
           8   FE0A      VE=Hex Keypress
           A   FD29      I=VD(LSDP)
           C   DAB5      Show 5MI        erase
           E   FE29      I=VE(LSDP)
        0320   DAB5      Show 5MI        display keypress
           2   A30C      I=030C
           4   DAB7      Show 7MI        erase cursor
           6   7A05      VA=VA+5
           8   8DE0      VD=VE
           A   00EE      Return
```

Subr List

```
02C4   F8      LDI 02
   5   02
   6   BC      PHI RC
   7   AC      PLO RC       RC=0202
   8   4C      LDA MRC
   9   BD      PHI RD
   A   4C      LDA MRC
   B   AD      PLO RD       RD=address in 0202, RC=0204
   C   4D      LDA MRD
   D   5C      STR MRC
   E   1C      INC RC                            RC=0205
   F   0D      LDN MRD
02D0   5C      STR MRC      M(0204-5)=instruction
   1   D4      Return
```

Addr+2

```
02D2   EC      SEX=C
   3   F8      LDI=02
   4   02
   5   BC      PHI RC
   6   F8      LDI 03
   7   03
   8   AC      PLO RC       RC=0203
   9   F8      LDI 02
   A   02
   B   F4      ADD
   C   73      STXD
   D   F8      LDI 00
   E   00
   F   74      ADC
02E0   5C      STR MRC
   1   D4      Return
```

Addr-N

decrements address scratchpad by the byte following ADDR-N call

```
02E2   EC      SEX=C
   3   F8
   4   02
   5   BC      PHI RC
   6   F8
   7   03
   8   AC      PLO RC       RC=0203
   9   45      LDA MRS      load parameter N
   A   F5      SD
   B   73      S7XD
   C   F8
   D   00
   E   75      SDB
   F   5C      STR MRC
   0   15      INC R5       skip byte following N
   1   D4      Return
```

```
            Subr Pack                              Call followed by 2 bytes

032C        96              GHI R6
            C4   EC         NOP
            BC              PH1 RC
            F8              LDI FO
0330        FO
            AC              PLO RC                 RC=OEFO=address of Chip-8  VO
            45              LDA MR5
            BD              PHI RD
            45              LDA MR5
            AD              PLO RD                 RD=0202 or address of instruc-
                                                   tion to be changed
            EC              SEX=C
            4C              LDA MRC                load VO
            FE              SHL
            FE               "
            FE               "
            FE               "                     shift left 4 times
            F4              ADD MRC                add V1
            5D              STR MRD
            1C              INC RC
            1D              INC RD
0340        4C              LDA MRC                load V2
            FE              SHL
            FE               "
            FE               "
            FE               "                     shift left 4 times
            F4              ADD MRC                add V3
            5D              STR MRD
   7        D4              Return
```

The complete code for this program can be obtained from THE VIPER for a handling charge of $5.00. The program is available on VIP compatible cassette tape only. This is an experiment - to see if any of you would prefer to get tapes from THE VIPER rather than type in all the code yourself. If it's successful, we'll start a cassette exchange library!

Tiny BASIC

Tiny BASIC ROM Implements Fourteen Standard BASIC Commands for the COSMAC VIP

The Tiny BASIC ROM Board VP-700 provides the COSMAC VIP user with the fundamental functions of the high-level programming language called BASIC. The Tiny BASIC interpreter is resident on the ROM and requires only a user-supplied ASCII-coded keyboard to implement the fourteen standard BASIC commands. Twelve additional commands are provided to augment the capabilities of the COSMAC VIP.

- •ROM resident—no need to load BASIC into RAM
- •SHOW command for graphics
- •COLOR command for color display using the VP-590 Color Board
- •FQ and TO commands for control of the VP-595 Simple Sound Board
- •Twenty-six 16-bit integer variables plus one array

Standard BASIC Commands:

NEW
LIST
RUN
GO TO
GO SUB
IF...THEN
INPUT
LET
PRINT
REM
RETURN
ABS
RND
END

Plus...

SAVE and LOAD	For cassette program storage
CLS	Clear screen
COLOR	Sets color map (requires VP-590)
GO KEY	Branch on keypress
TI	Sets timer, reads timer
FQ	Sets tone frequency (requires VP-595)
TO	Sets tone duration
PT	Sets pattern for SHOW command
SHOW	Displays PT pattern at coordinates
TV ON, TV OFF	Turns display on or off
HIT	Detects pattern intersection
MEM	Program storage remaining

Note: An external, ASCII-coded keyboard is required but not supplied.

This is not an ad, just a reproduction of the data sheet distributed by RCA at the New York and Chicago personal computing shows where they demonstrated the BASIC.

Although very slow, due to the amount of processing time 'stolen' by the video display and the use of a two-level interpreter, the provision for color, sound, and graphic display really take it out of the 'tiny' category.

RCA says the ROM version will be available in January, and rumor has it that a companion ASCII keyboard will be available shortly afterward for less than $50!