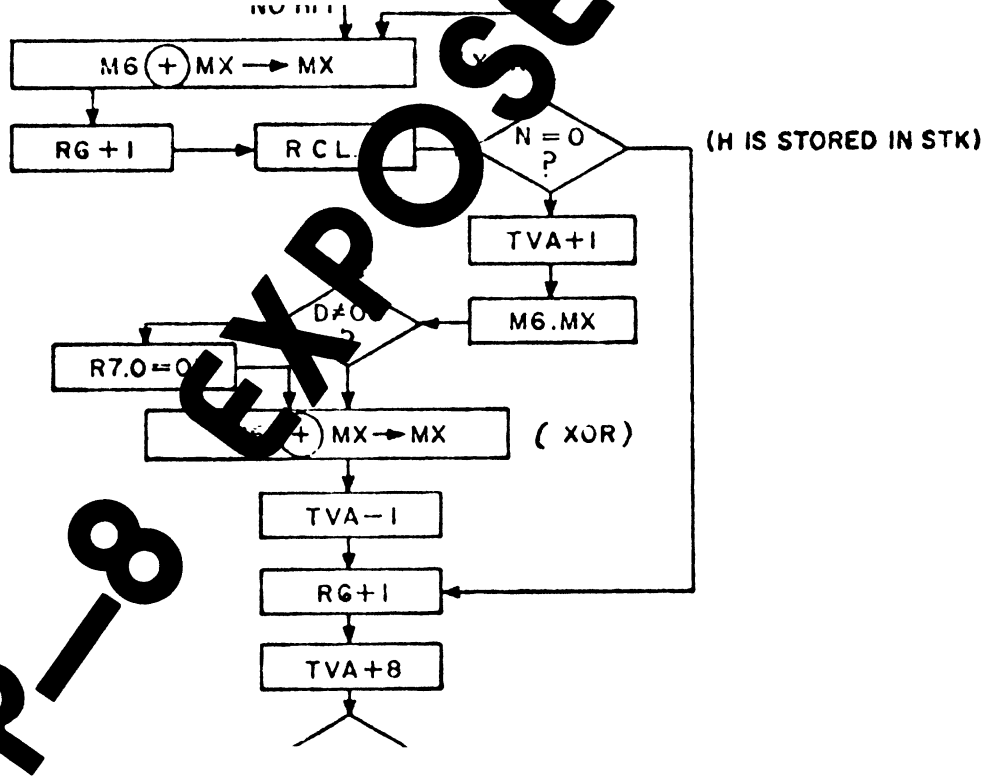
VIPER

VOLUME 1 AUGUST 1978 ISSUE 2



CHIP-8 EXPOSED!

PLUS **VIP TEXT EDITOR**

$2.00 AN ARESCO PUBLICATION

EDITORIAL

Editorials are not my strong point - and most of the VIPER issues will not have one. But I couldn't pass up the opportunity this month to tell you how much I appreciate the overwhelming response the VIPER has enjoyed from VIP owners (and prospective owners) all over the USA and Canada. In the first month alone, we've received more than twice as many subscription orders as we expected; articles, ideas, suggestions, and requests for specific information; even a few CHIP-8 programs.

I have shared your response with RCA's VIP product manager, Rick Simpson. He's as pleased and impressed as I am - as you can see in the New From RCA column in this issue, RCA has decided to support the VIP in a **big** way, and is turning out new VIP related products so fast it makes your head spin. We aren't supposed to know - or even guess - that there may be a VIP version of TINY BASIC in the works at RCA, so don't breathe a word to anyone about it - but I caught a peek at a memo which would suggest that **someone** at RCA is working very hard to get TINY BASIC up and running on the VIP by Christmas.

This issue contains the most-requested article (an indepth discussion of the CHIP-8 interpreter).
There are a few other goodies thrown in, as well. You'll see that this issue is not all prettily typeset, as issue #1 was - we couldn't take the chance of introducing errors into the manuscripts. In fact, from now on, most of the articles will be copies of the author's original work. Typists generally don't understand flowcharts, schematics, or code, and errors are remarkably easy to come by. One of the reasons this issue is two weeks late is a belated decision to forgo typesetting..... The next issue will be on time, since we already have most of the material in-house (thanks to all of you who wrote and shared your ideas and discoveries with us!)

Hope to see some of you at PC '78 in Philadelphia. Come by the RCA booth and see some of the marvelous new VIP related products

Until next month, then.

Terry

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).

The VIPER is an Aresco Publication, edited by Terry L. Laudereau. For information contact Editor, VIPER, P.O. Box 43, Audubon, PA 19407. (215)631-9052

Dear Terry,

Having for some time been fascinated by the 1802uP and by rather vague reports that it was designed to support compact interpreters, I ordered the VIP directly from RCA at PC '77 in Atlantic City. Before that time, I had breadboarded the "ELF" described in Popular Electronics.

Not being much interested in video games, my primary reason for purchasing the VIP was to learn numerical interpreter techniques; my second reason was because of the built in cassette I/O and video interface.

My video display is a 9" Hitachi black and white model PA-5 with the Pickles and Trout direct video entry conversion kit. This is a combination which I can heartily recommend to everyone. My cassette recorder is a low quality $29 model. At first, I had a great deal of difficulty with battery operation. An A.C. adapter solved those problems.

After writing a few simple CHIP-8 programs and implementing some of the games in the instruction manual, I analyzed the structure and operation of CHIP-8. In the process, I have produced a map of locations 0000-01FF and have flow charted some of the more complex subroutines (instructions) such as the DXYN instruction. I have also flow charted the ROM monitor program but much of it remains obscure to me. Although some might complain that this information should have been supplied with the VIP, I found the experience invaluable in learning machine (1802) language programming techniques. Also as a result of my analysis, I have found some possibilities in CHIP-8 which you may wish to communicate to your readers.

The 8XYN instruction (N=0,1,2,4,5) has four undocumented functions - 8XY3, 8XY6, 8XY7, and 8XYE. This is due to the fact that the 8XYN instruction operates by executing a single byte subroutine formed from the "N" digit. The description that follows applies to all values of N except N=0. In this case, the contents of VY is simply stored in VX.

The 8XYN subroutine begins execution with P=3, X=2, R5 pointing at the last byte of 8XYN, R6 pointing at VX, and R7 pointing at VY. If N is not 0, a hex "D3" is pushed on the stack -M(R2)- followed by a byte composed of the last byte of 8XYN orred with a hex "F0". X is then set to 6, the D register is loaded with the contents of VY and a SEP 2 -> P is executed. Thus, the single byte subroutine "FN" is executed, followed by a "D3" or SEP 3 -> P which returns control to the 8XYN subroutine. Following this, the contents of the D register is stored in VX and the state of DF (0 or 1) is put in VF.

Therefore, if *N=3, 6, 7, or E, the functions of exclusive or, shift right, subtract, and shift left respectively are added. This is summarized in the following table:

	INSTRUCTION	RESULT	"FN"	MNEMONIC
	8XY0	VX <- VY		
	8XY1	VX <- VX + VY (VF <- DF)	F1	OR
	8XY2	VX <- VX * VY (VF <- DF)	F2	AND
*	8XY3	VX <- VX ⊕ VY (VF <- DF)	F3	XOR
	8XY4	VX <- VX & VY (VF <- DF)	F4	ADD
	8XY5	VX <- VX - VY (VF <- DF)	F5	SD
*	8XY6	VX <- (SHR)VY (VF <- DF)	F6	SHR
*	8XY7	VX <- VX - VY (VF <- DF)	F7	SM
*	8XYE	VX <- (SHL)VY (VF <- DF)	FE	SHL

N = 8 through D or F cannot be used because these values would result in the execution of an immediate instruction with uncertain (at best) results. A CHIP-8 program to demonstrate the 8XYN instruction follows:

```
01F2  F8 LDI 81->D
01F3  81
01F4  BA PHI D->RA.1
01F5  F6 SHR 0->D->DF
01F6  F6 SHR 0->D->DF
01F7  F6 SHR 0->D->DF
01F8  F6 SHR 0->D->DF
01F9  30 BR  BR 012F
01FA  2F

      ----

0200  6370 V3=70
0202  640F V4=0F
0204  6A00 VA=00
0206  6B00 VB=00
0208  A270 I='X='
020A  2244 DO S.R.
020C  8600 V6=V0
020E  6B06 VB=06
0210  A274 I='Y='
0212  2244 DO S.R.
0214  8700 V7=V0
0216  6B0C VB=0C
0218  A279 I='N='
021A  2244 DO S.R.
021C  8042 V0=V0*V4
021E  8031 V0=V0+V3
0220  A261 I=0261
0222  F055 MI=V0
0224  6F00 VF=00
0226  2260 DO S.R.
0228  80F0 V0=VF
022A  6B12 VB=12
022C  A27E I='F='
022E  DAB5 SHOW 5 @ A,B
0230  224E DO S.R.
0232  8060 V0=V6
0234  6B18 VB=18
0236  224E DO S.R.
0238  F20A DEBOUNCE
023A  E4A1 SKIP IF KEY≠V4
023C  11FC GOTO 01FC
023E  123A GOTO CHECK KEY
0240  0000
0242  0000
0244  DAB5 SHOW 5@ A,B
0246  F00A V0=KEY
0248  F10A V1=KEY
024A  0266 DO M.L.S.R.
024C  8011 V0=V0+V1
024E  6A09 VA=09
0250  F0F2 I=MSD V0
0252  DAB5 SHOW 5 @ A,B
0254  6A0F VA=0F
0256  F029 I=LSD V0
0258  DAB5 SHOW 5 @ A,B
025A  6A00 VA=00
025C  00EE
025E  0000
0260  8600 8XYN S.R.
0262  00EE
0264  0000
0266  F8 LDI F0->D
0267  F0
0268  A6 PLO D->R6.0
0269  06 LDN M(R6)->D
026A  FE SHL DF<-D<-0
026B  FE SHL DF<-D<-0
026C  FE SHL DF<-D<-0
026D  FE SHL DF<-D<-0
026E  56 STR D->M(R6)
026F  D4 SEP 2->P
```

```
0270  88      'X='
0271  53
0272  20
0273  53
0274  88      'Y='
0275  53
0276  20
0277  23
0278  20

0279  88      'N='
027A  CB
027B  A8
027C  9B
027D  86
027E  F8      'F='
027F  83
0280  F0
0281  83
0282  80
```

Use of the program is simple - enter two digit values for X, Y, and N. These values and the resultant values of VF and VX are displayed. The first digit entered for N is ignored; the last digit of N determines the function performed - or, and, add, etc. Depressing key F restarts the program.

Note that a machine language subroutine was entered at location 01F2. This provides a new CHIP-8 instruction -FXF2- which sets I to the hex pattern of the most significant digit of VX. The instruction loads the contents of VX into D, shifts D right 4 times, then branches to the appropriate place in the FX29 subroutine. The space from 01F2 to 01FB is free for the addition of other "FX" type instructions which are found useful. For example, set timer equal VX and wait, shift VX left one digit position, and so on.

Another unused location begins at 00FC and ends at 0104. This space is suitable for often used machine language subroutines such as wait for timer equal zero. Or, by moving the two beginning bytes of the "FX" subroutine at locations 0105 and 0106 to locations 00FE and 00FF, another "FX" instruction -FX00- can be inserted at locations 0100 to 0106 in front of the FX07 instruction. A possible "FX" instruction subroutine which will fit here is 06FEFEFEFE56D4. This series of instructions will shift VX left four times or one digit position. However, if this is done, one other change must be made. The interpreter table at locations 0050 to 006F which contains the addresses of the CHIP-8 instruction subroutines must be changed to reflect the new entry point of the "FX" subroutine. Locations 005F and 006F contain 01 and 05 respectively which is the original starting address. If the bytes at 0105 and 0106 are moved to 00FE and 00FF, a 00 must be placed in 005F and an FE in 006F.

I have written a simple editor program which resides in the first two pages of RAM. It consists of a numerical interpreter in locations 0000-014F and the editor program, written the numerical language, in locations 0150-01FF. The functions of the editor allow me to display and alter any location. The display address can be rapidly or slowly incremented or decremented. There is also a copy function which will copy any range of locations to any location except 0000-01FF, of course.

I have also written an expanded CHIP-8 language which I call CHIP-8 1/2. It occupies 3 pages and although very similar, is totally

incompatible with CHIP-8. I was able to add two new op codes by putting EXA1/9E into the "FX" series of instructions and by combining 5XY0 and 9XY0 into one op code. The two new functions are branch to MM if VX = 0 or VX ≠ 0 and take the form: NXMM. Another major change over CHIP-8 was the relocation of the "FX" instructions to page 2, allowing a full page of this instruction type. Also, the display instruction was expanded to include OR, AND, XOR, and test functions.

I have witten a LIFE program which occupies practically all of my VIP's 2K of memory. It consists of a large machine language subroutine supported by CHIP-8. The LIFE grid is a 64 X 32 cell array; a new generation is displayed every 2 1/4 seconds. Page 2 is occupied by a CHIP-8 program which allows the generation of a starting pattern, clearing the array, depositing predefined patterns, and starting and stopping the LIFE process. Page 3 is occupied by the LIFE subroutine. Page 4 is a lookup table which is used to find the population count of a cell. Pages 5 and 7 are the alternate generation display buffers. Page 6 is used to store predefined patterns. This program evolved from an all CHIP-8 program to the inclusion of larger and larger machine language subroutines as I sought to decrease the cycle time from ten minutes to the present 2 1/4 seconds. I don't believe that unrolling my current LIFE subroutine any more will bring substantial gain. Possibly there is a faster algorhithm which can be employed. However, I think that the only way to gain a significant increase in speed will be by a hardware change. That is, by the addition of a line buffer to reduce the overhead of repeated DMA requests for the same 8 bytes. Such a line buffer would have the added advantage of allowing the use of three cycle instructions.

In the future, I plan to design a line buffer which will take the form of a plug-in module containing the video interface chip, a line register, and miscellaneous logic. The plug in module will replace the video IC in its present location. At the same time, I may investigate the possibility of expanding the display size to 128 by 64 or some such size.

Another hardware change that I plan to implement is the addition of some sort of primitive disk-like random access device. It will probably be an endless tape loop - cassette or cartridge.

My software plans will be combined into a single operating system, a super CHIP-X, which will include numerical programming language with immediate execution of instructions entered from the keypad, editor, tape access with file management (if I can come up with a satisfactory random access device), and perhaps program relocation. The numerical instructions will probably be three or four bytes in length with one byte op codes. Of course, more than 2K RAM will be required for all this. I have ordered the memory expansion kit from RCA. Hopefully this will be enough.

I am employed by a large computer manufacturing company headquartered in Blue Bell, Pa. My background is primarily electronics, but my software experience is catching up with that.

Most of my adult employment has been in the educational/technical writing fields. I am more than willing to join/form a VIP user's group and to help anyone who wants help with their VIP.

Please feel free to publish any or all parts of this letter.

Sincerely,

Peter K. Morrison

NEW FROM RCA

The VIP will be sporting vivid color this fall with the introduction of the VIP COLOR BOARD from RCA. You'll have program control of three background colors & eight foreground colors with CHIP-8C, the color-language addition to CHIP-8. Available late October. Priced under $80.00.

Convert the VIP single-tone output to 256 different frequencies with the new VIP TONE BOARD from RCA. With a single machine language subroutine added to either CHIP-8 or CHIP-8C, you'll be able to set the frequency and duration of the output tone. Speaker and jacks included. Available late '78; priced under $30.00.

Your VIP will be synthesizing two-part harmony with RCA's newest VIP product: the MUSIC BOARD. You'll have program control of frequency, duration, and amplitude envelope for each of two independent output channels, and an on-board potentiometer will control tempo. There will be a provision for sync output - for multi-track recording or slaving several VIPs for simultaneous play. The software, incidentally, will support the PAIA drum synthesizer which can be hooked on thru the output port. No speaker included. Under $50.00.

Add 4K of static RAM to your VIP by plugging in still another new VIP option. The MEMORY EXPANSION BOARD attaches through the expansion connector, and jumpers will address any of the first four 4K memory segments. Available by the end of the year, for under $100.00.

If you're a fan of two-player video games, this will please you! The new VIP EXPANSION KEYPAD is just what you've been waiting for. The 16-key keypad and cable connects to a socket on the color board or on its own (also new!) VIP KEYBOARD INTERFACE CARD. Instructions are included for use with either CHIP-8 or CHIP-8C. Available late October, each will be priced under $20.00.

At last you can program your own high-level language for the VIP with RCA's new EROM BOARD and the EROM PROGRAMMER. The board allows two Intel 2716 EROMs to be interfaced to the VIP and has provisi ns for placing EROMs anywhere in VIP memory space. It also allows re-allocation of on-board RAM in memory space. The programmer allows you to program the Intel 2716 EROM, and comes complete with software to program, copy, and verify EROM. All required EROM voltages are generated on board. Both should be available "soon". The EROM board is priced at under $50.00 and the Programmer will be less than $130.00

A TEXT EDITOR FOR THE VIP Part One

by Don Stein

I was tired of all my friends in the Crystal City Computer Club bragging about their big, expensive computers, and looking down their noses at my little VIP. Why, just their latest peripheral add-on board alone, they liked to tell me, cost more than my entire computer!

But I knew that my VIP was not only cheaper than their monsters, but also better. After all, my microprocessor chip was as powerful as theirs. Furthermore, they were always complaining about glitches and bus noise; I knew that since my VIP used CMOS technology instead of TTL, it didn't have any glitches. For the same reason, my VIP was a much better "hands on" computer than theirs - if I wanted to add hardware, CMOS would be much easier to work with than their TTL machines.

And my VIP had one other advantage. Since practically everything was software-driven, I could change the way the machine operated by making changes in the software - I wasn't tied down to a particular operating system or programming language.

To prove my point, I set out to write a text editor for my VIP. It would have all the bells and whistles their big machines didn't have - such as forward and backward scrolling; forward and backward paging; automatic repeat on every key, including control keys; full software motor control of two or more tape drives; and so forth. And it wouldn't require 8K or 16K of memory, either!

This series of articles describes the text editor I have developed. The reader can be the judge as to whether I was successful in proving my point.

Character Display

The first problem was how to display characters with the VIP. Clearly, the regular display operated under CHIP-8 was not high enough resolution; I would have to use full resolution (64 X 128 dots) display described on page 94 of the VIP Instruction Manual.

Even with this display, the limiting factor would be the number of dots (64) in the horizontal direction. To get even 16 characters per line, separated by spaces, it would be necessary to use character displays having only three horizontal dots per character.

I experimented with several character formats, both upper - and lower - case, and finally settled on an all-upper-case format permitting eleven rows of 16 characters per row; each character would be represented by a 3 X 8 dot matrix in a 4 X 12 field. The eleventh row of characters would just touch the bottc of the screen display area.

This format provides perfectly legible, but not always beautiful characters. Did you ever try to represent an upper-case N, for example, with a 3 X 8 matrix? Nonetheless, since the text editor would use a software character generator, I would be able to change the character display patterns at any time.

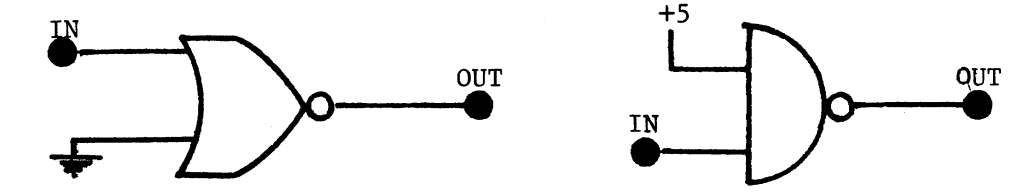
ASCII Keyboard

The next step was to hook up a typewriter-style ASCII keyboard. There are many such units available for around $50. I selected a Risk keyboard, which cost about $70 including a nice-looking cabinet.

Hooking up the keyboard was simple. I merely ran the outputs of the keyboard into the inputs of the optional VIP input port. Then the "keypressed" or "valid data

strobe" signal was run into one of the flag lines (I used EF3 the same as the hex keypad, because I wanted to keep EF4 free for other uses). The "keypressed" or "valid data strobe" also has to be run to the U25 latch input.

Note that the data inputs and the U25 latch input should be positive logic, whereas the flag input requires negative logic. I simply used a logic inverter between the data valid strobe and the flag input. It is not necessary to buy an expensive IC to get a logic inverter; a cheap 4001 or 4011 can be wired up as an inverter, using the scheme shown in figure 1. I used a ready-made $3 PC board (Radio Shack 276-154) to mount the IC and the wires. I also used this board to mount the circuitry for controlling the tape drives (described in a later installment). The complete circuit is shown in figure 2.



a) ¼ of a CD 4001 b) ¼ of a CD 4011

FIGURE 1 - Logic Inverters

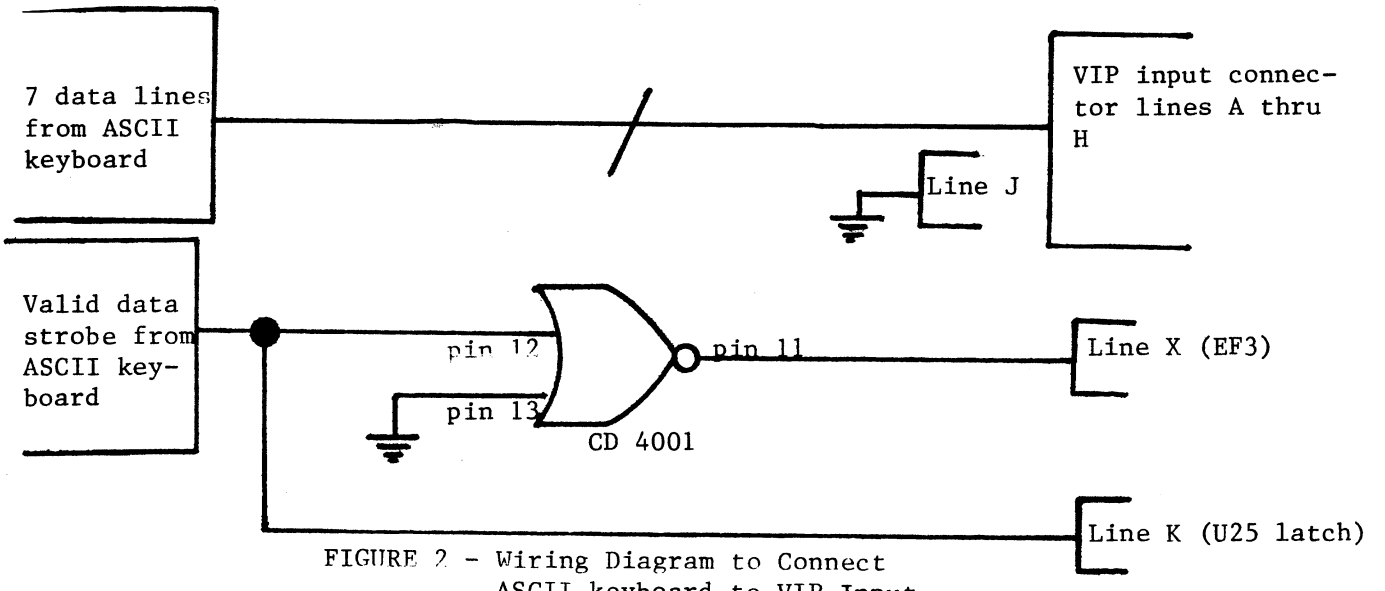


FIGURE 2 - Wiring Diagram to Connect ASCII keyboard to VIP Input Port

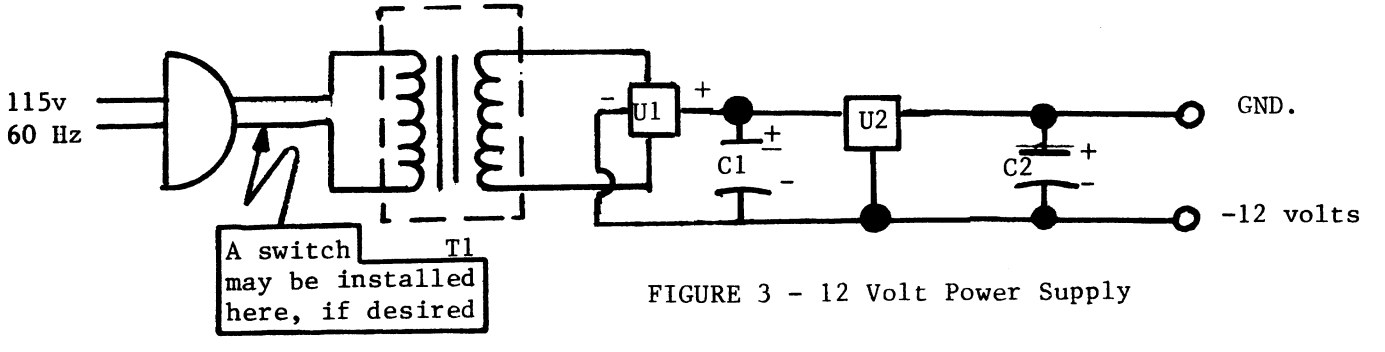


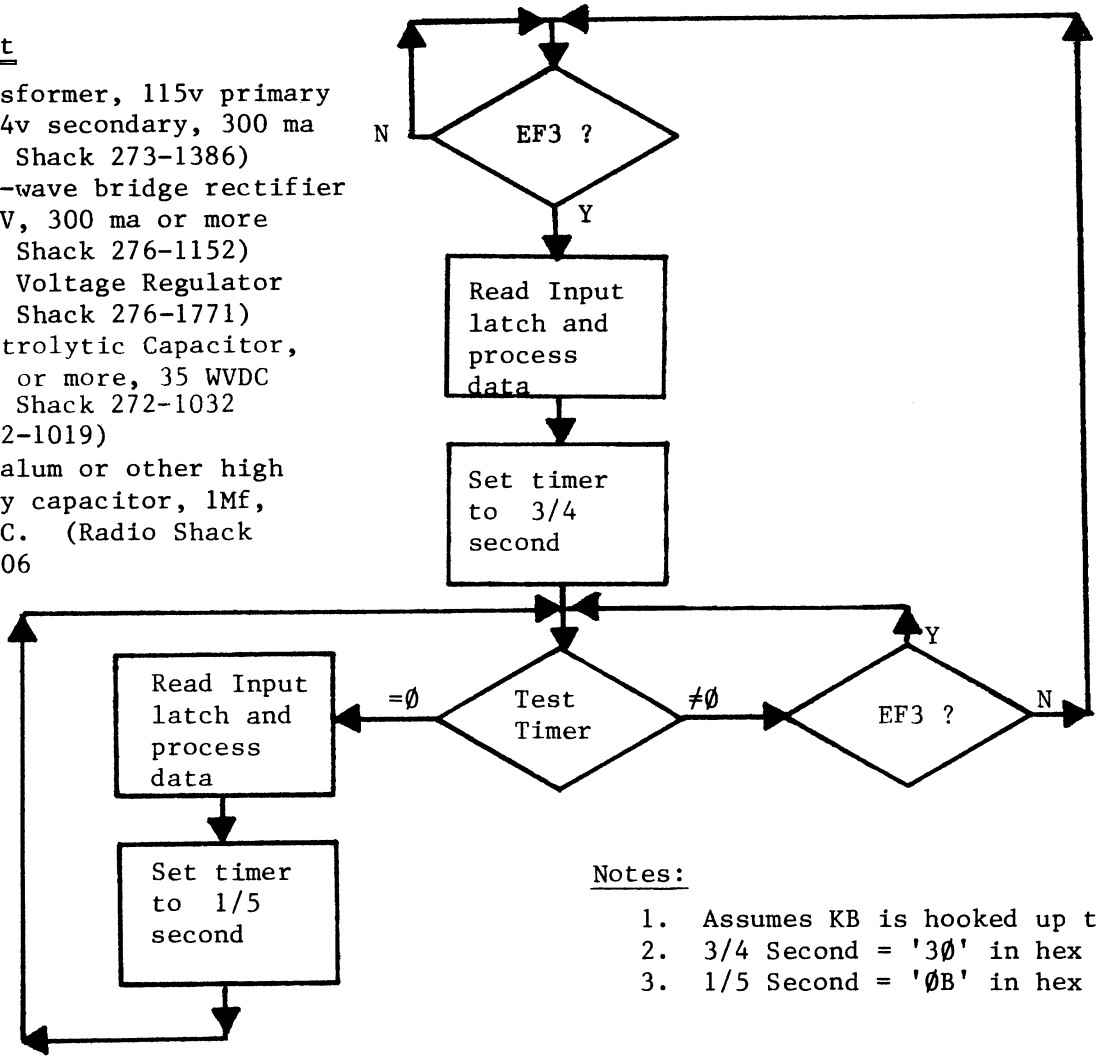
FIGURE 3 - 12 Volt Power Supply

12 Volt Power Supply

Most ASCII keyboards require a -12 volt power supply. Also, RF modulators require a negative voltage supply. Therefore, I decided to build a cheap 12V power supply. The circuit, using Radio Shack parts, is shown in figure 3. The total cost is under $10.

Parts List

T1 - Transformer, 115v primary 18 - 24v secondary, 300 ma (Radio Shack 273-1386)
U1 - Full-wave bridge rectifier 100 PIV, 300 ma or more (Radio Shack 276-1152)
U2 - 7812 Voltage Regulator (Radio Shack 276-1771)
C1 - Electrolytic Capacitor, 1000Mf or more, 35 WVDC (Radio Shack 272-1032 or 272-1019)
C2 - Tantalum or other high quality capacitor, 1Mf, 35 WVDC. (Radio Shack 272-1406



Notes:

1. Assumes KB is hooked up to flag 3
2. 3/4 Second = '3Ø' in hex
3. 1/5 Second = 'ØB' in hex

FIGURE 4 - Flowchart to Read ASCII Keyboard, With Automatic Repeat Function

---

Data Input Software

The software to read the ASCII keyboard as to test the flag line, read the latch, and wait until the flag line is no longer active before reading the next character. In addition, an automatic-repeat feature can be programmed using the VIP timer.

A simple flowchart of this software is shown in figure 4; a more detailed description of the program steps will be covered in a later installment.

Next Month

Next month I will describe the overall text editor software, along with a generalized operating system I wrote to go with it. Future installments will cover the tape input and output routines and tape drive motor control. By the way - the entire text editor fits in 3K of VIP memory.

THE CHIP-8 INTERPRETER

by Gooitzen S. van der Wal

The CHIP-8 interpreter is written in the language of the microprocessor CDP18Ø2. The microprocessor has, among others, 16 2-byte registers (R) and two 4-bit registers (P and X). P is used to point at the register which is serving as the program counter. X is used to point to another of the 2-byte registers which is serving to pont to data in memory. Initially RØ (P=Ø) is sued as the program instruction pointer. In the CHIP-8 interpreter, R4 (P=4) is used as the call-routine program counter, R3 (P=3) is the interpreter subroutine program counter, and R5 (P=5) is the CHIP-8 high-level language program counter.

Basically, the CHIP-8 interpreter sees the CHIP-8 program instructions as a data list. The call-routine takes a single CHIP-8 program instruction byte with M(R5) (= Memory contents at address R5). The call-routine recognizes the first digit and sets R3 to the address of the interpreter-subroutine (and sets P to 3). Then the other three digits of the instruction are used to execute the right subroutine. All those subroutines are ended by setting P back to 4 so the call-subroutine can take out the next CHIP-8 program instruction.

The basic flow-chart for the interpreter is then:

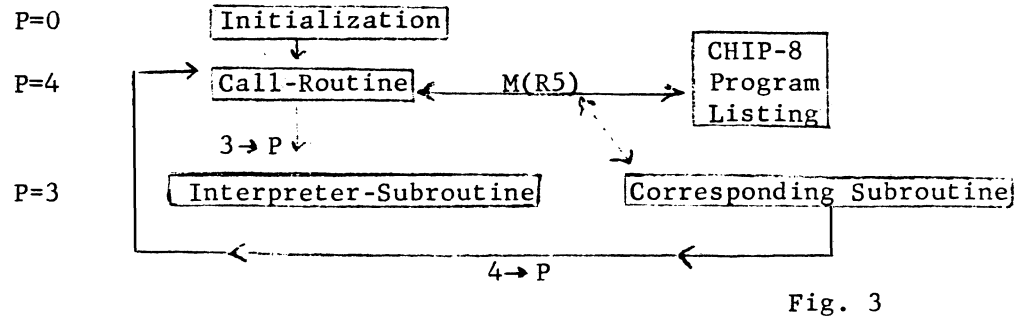


Fig. 3

For more detailed description of the CHIP-8 interpreter see flow-chart.

Flow-Chart CHIP-8 INTERPRETER.

P=0

INITIALIZATION

4→P

: RB.1 = 0L {DISPLAYPAGE}
RG.1 . 0K {R6 = VX, R7 = VY}
R2 . 0KCF {STK}
R1 . 0146 {INTERRUPT ROUTINE}
R4 . 004B {CALL ROUTINE}
PC = R5 : 01FC {CHIP-8 PROGRAM COUNTER
STARTS WITH: CLEAR DISPLAY
TURN TVON}

P=4
X=2

CALL-ROUTINE:

Get program byte → RF.0

PC +1

FIRST INSTRUCTION DIGIT = 0 ?

YES → SECOND INST. DIGIT → R3.1

NO

FIRST INSTRUCTION DIGIT +0050 → RC

SECOND INSTR. DIGIT +0KF0 → R6 =VX

Get program byte

THIRD INSTR. DIGIT +0KF0 → R7 =VY

R3 = f(FIRST INSTRUCTION DIGIT) 3→P

Go to CALL. 4→P

Get program byte → R3.0

PC +1

3→P

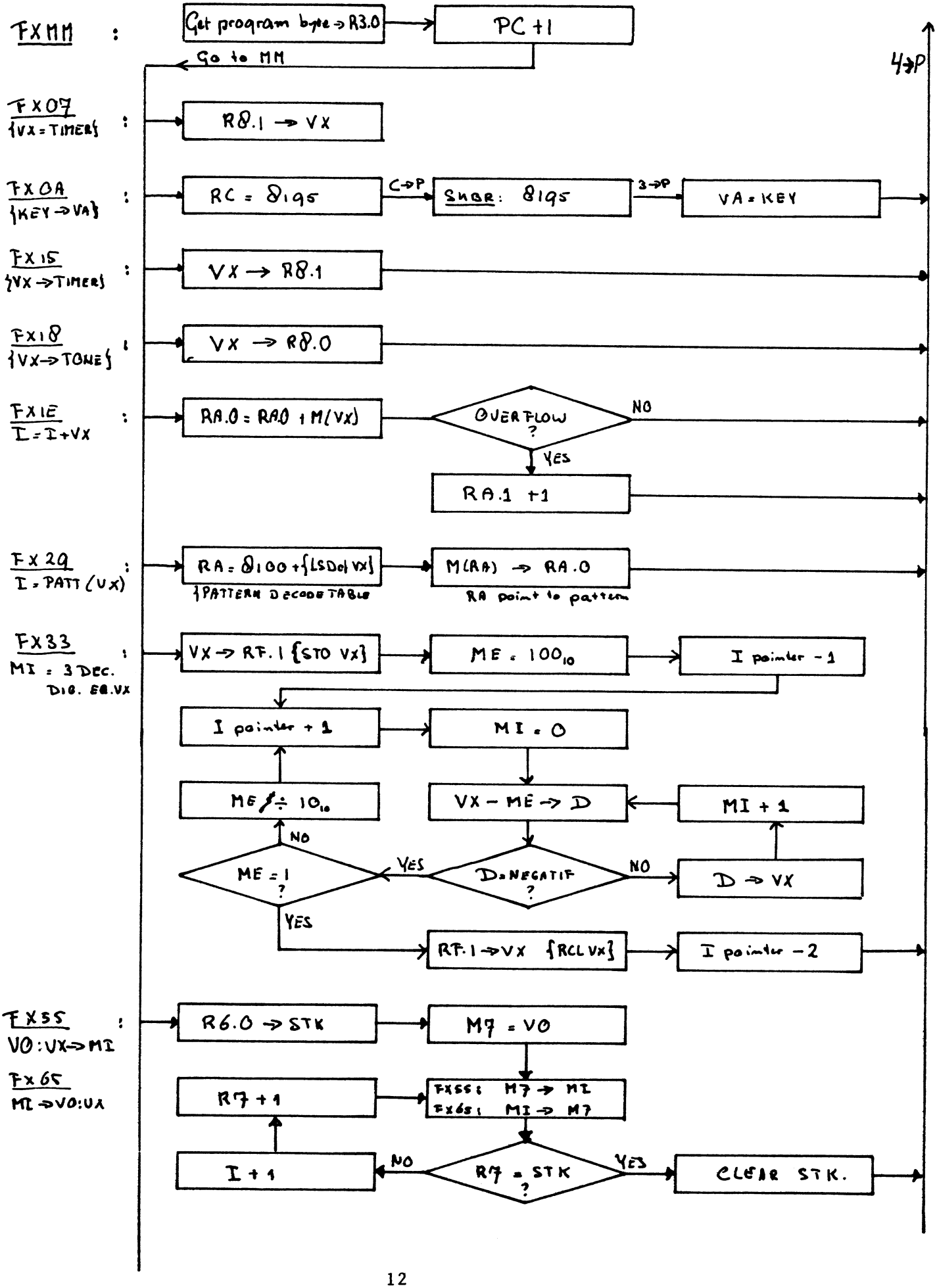
SUBROUTINE WITH
R3 AS PROGRAM COUNTER.

DECODE TABLE: FIRST INSTRUCTION DIGIT :

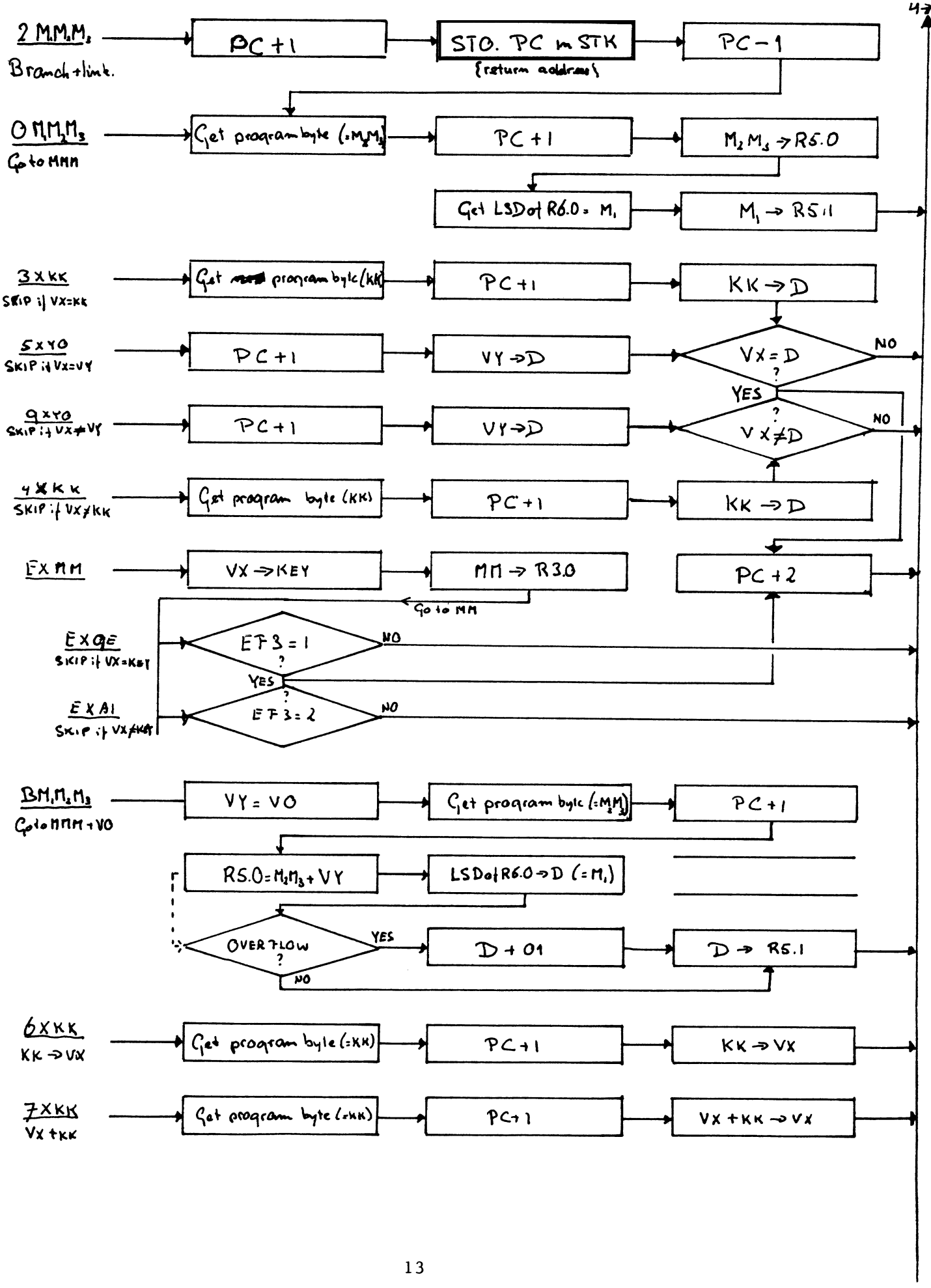
	0	1	2	3	4	5	6
R3 = f(FIRST INSTR.DIGIT):	0MMM	017C	0175	0183	018B	0195	0184

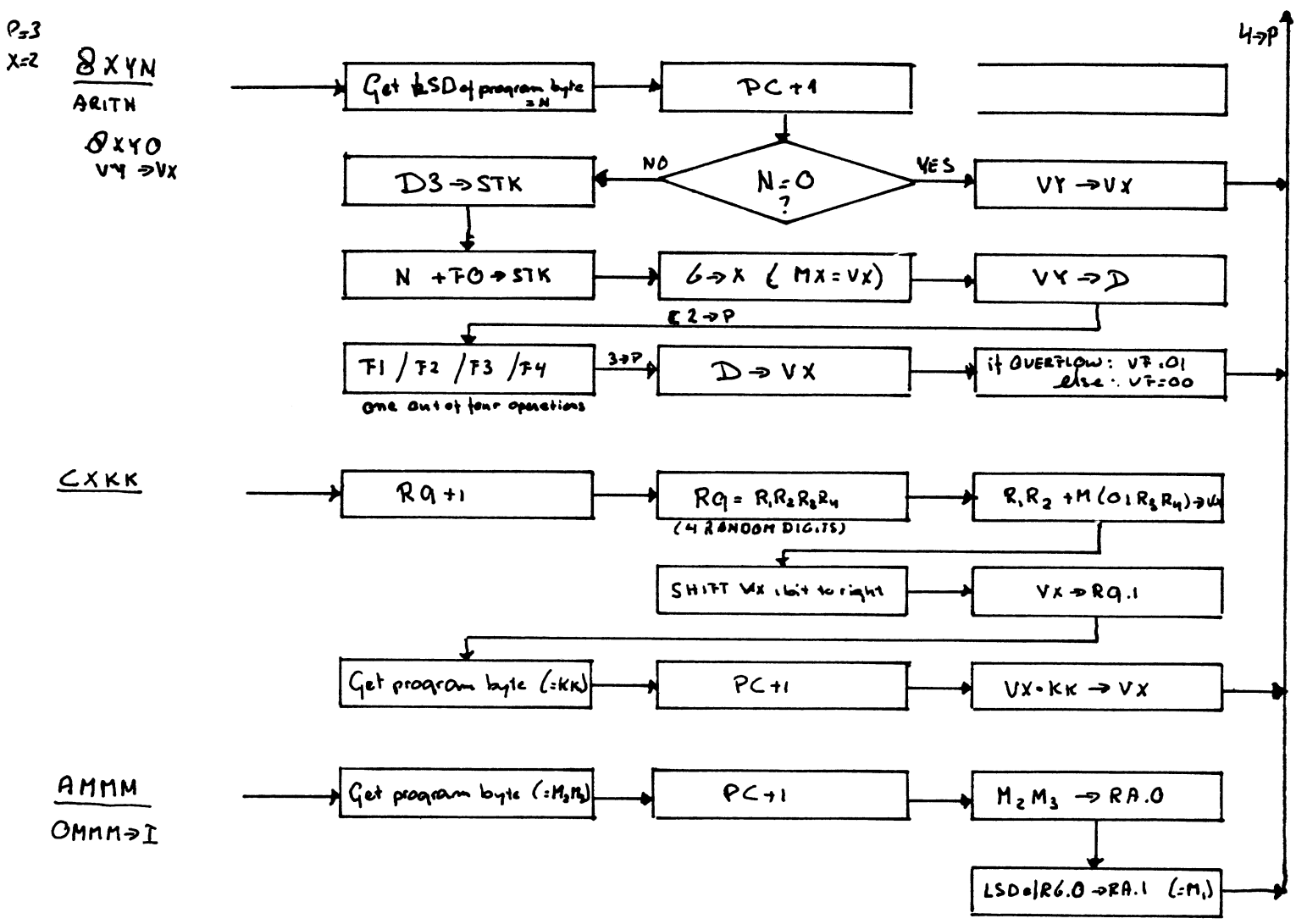
FIRST INSTR.DIGIT:	7	8	9	A	B	C	D	E	F
R3 = f( ):	0187	018C	0191	01EB	01A4	01D9	0070	0199	0105

P=3
X=2


FXMM :
Get program byte → R3.0
PC +1
Go to MM
4→P
FX07 {VX = TIMER} :
R8.1 → VX
FX0A {KEY → VA} :
RC = 8195
C→P
SUBR: 8195
3→P
VA = KEY
FX15 {VX → TIMER} :
VX → R8.1
FX18 {VX → TONE} :
VX → R8.0
FX1E I = I + VX :
RA.0 = RA.0 + M(VX)
OVERFLOW ?
NO
YES
RA.1 +1
FX29 I = PATT (VX) :
RA = 8100 + {LSDgt VX}
{PATTERN DECODE TABLE
M(RA) → RA.0
RA point to pattern
FX33 MI = 3 DEC. DIG. FR.VX :
VX → RF.1 {STO VX}
ME = 100₁₀
I pointer - 1
I pointer + 1
MI = 0
ME ÷ 10₁₀
VX - ME → D
MI + 1
NO
ME = 1 ?
YES
D = NEGATIF ?
NO
D → VX
YES
RF.1 → VX {RCL VX}
I pointer - 2
FX55 : V0 : VX → MI
FX65 MI → V0 : VX
R6.0 → STK
M7 = V0
R7 + 1
FX55 : M7 → MI
FX65 : MI → M7
I + 1
NO
R7 = STK ?
YES
CLEAR STK.

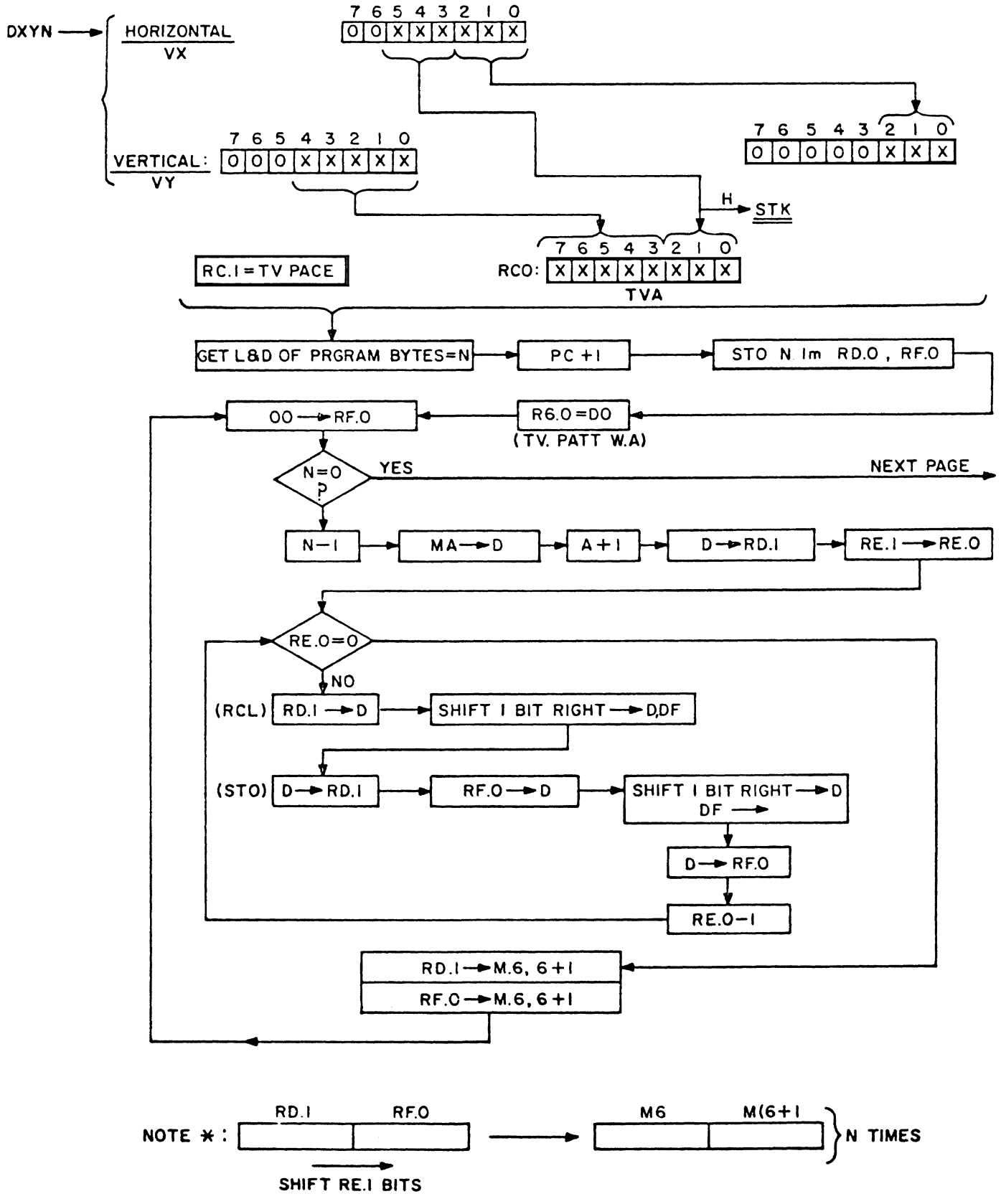
P=3
X=2

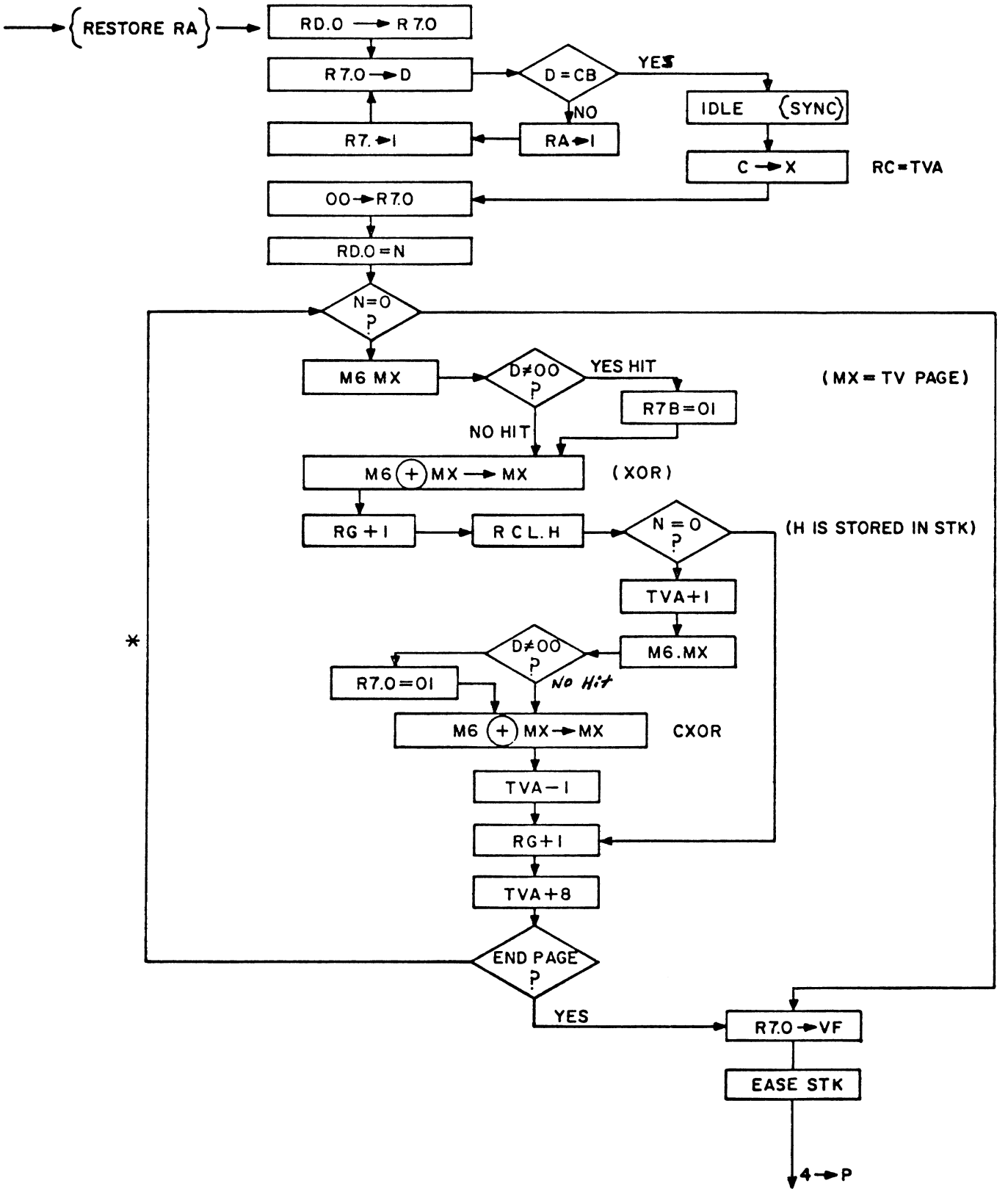

2 M1M2M3
Branch+link.
PC+1
STO. PC in STK
{return address}
PC-1
0 M1M2M3
Go to MMM
Get program byte (=M2M3)
PC+1
M2M3 → R5.0
Get LSD of R6.0 = M1
M1 → R5.1
3 X KK
SKIP if VX=KK
Get program byte (KK)
PC+1
KK → D
5 X Y 0
SKIP if VX=VY
PC+1
VY → D
VX = D ?
NO
YES
9 X Y 0
SKIP if VX≠VY
PC+1
VY → D
VX ≠ D ?
NO
4 X K K
SKIP if VX≠KK
Get program byte (KK)
PC+1
KK → D
EX MM
VX → KEY
MM → R3.0
PC+2
Go to MM
EX 9E
SKIP if VX=KEY
EF3 = 1 ?
NO
YES
EX A1
SKIP if VX≠KEY
EF3 = 2 ?
NO
BM1M2M3
Go to MMM+V0
VY = V0
Get program byte (=M2M3)
PC+1
R5.0 = M2M3 + VY
LSD of R6.0 → D (= M1)
OVERFLOW ?
YES
NO
D + 01
D → R5.1
6 X KK
KK → VX
Get program byte (=KK)
PC+1
KK → VX
7 X KK
VX + KK
Get program byte (=KK)
PC+1
VX + KK → VX
4→P



COMING NEXT MONTH

J. W. Wentworth disassembles the VIP operating system ... More about Don Stein's VIP Text Editor ... Rick Simpson modifies CHIP-8 to provide I/O instructions ... VIPpers' Letters To VIPER ... New CHIP-8 Games ... More news from RCA ...


DXYN
HORIZONTAL
VX
7 6 5 4 3 2 1 0
0 0 X X X X X X
VERTICAL:
VY
7 6 5 4 3 2 1 0
0 0 0 X X X X X
7 6 5 4 3 2 1 0
0 0 0 0 0 X X X
H
STK
7 6 5 4 3 2 1 0
RCO: X X X X X X X X
TVA
RC.1 = TV PACE
GET L&D OF PRGRAM BYTES=N
PC +1
STO N Im RD.0 , RF.0
00 → RF.0
R6.0 = D0
(TV. PATT W.A)
N=0 ?
YES
NEXT PAGE
N−1
MA → D
A+1
D → RD.1
RE.1 → RE.0
RE.0=0
NO
(RCL)
RD.1 → D
SHIFT 1 BIT RIGHT → D,DF
(STO)
D → RD.1
RF.0 → D
SHIFT 1 BIT RIGHT → D
DF →
D → RF.0
RE.0−1
RD.1 → M.6, 6+1
RF.0 → M.6, 6+1
NOTE * :
RD.1
RF.0
SHIFT RE.1 BITS
M6
M(6+1
N TIMES


RESTORE RA
RD.0 → R7.0
R7.0 → D
D = CB
YES
NO
IDLE (SYNC)
C → X
RC=TVA
RA → I
R7. → I
00 → R7.0
RD.0 = N
N=0 ?
M6 MX
D≠00 ?
YES HIT
(MX = TV PAGE)
R7B=01
NO HIT
M6 (+) MX → MX
(XOR)
RG + I
R C L. H
N = 0 ?
(H IS STORED IN STK)
TVA+I
M6.MX
D≠00 ?
No Hit
R7.0=01
M6 (+) MX → MX
CXOR
TVA−I
RG+I
TVA+8
END PAGE ?
YES
R7.0 → VF
EASE STK
4 → P
*

CODING FORM FOR RCA COSMAC PROGRAMS

Title J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 00 Programmer

00	91	0X → RB.1 (where 0X is the highest memory page no. as determined by Operating System)--designates display page.
01	BB	
02	FF	0X-1 → R2.1 (stack pointer)
03	01	
04	B2	0X-1 → R6.1 (VX pointer)
05	B6	
06	F8	CF → R2.0 (stack pointer set to 0YCF, where Y = X-1)
07	CF	
08	A2	
09	F8	Set R1 (PC for Interrupt Routine) to 8146
0A	81	
0B	B1	
0C	F8	
0D	46	
0E	A1	
0F	90	Pre-set R4 to 001B in preparation for assignment as PC
10	B4	
11	F8	
12	1B	
13	A4	
14	F8	01 → R5.1
15	01	
16	B5	FC → R5.0 (R5 now set to 01FC; will serve as PC for CHIP-8 instructions, commencing with two instructions included on page 01).
17	F8	
18	FC	
19	A5	
1A	D4	4 → P (R4 becomes PC at this point)
		FETCH AND DECODE ROUTINE
1B	96	0Y → R7.1 (high byte of VY pointer)
1C	B7	
1D	E2	2 → X
1E	94	00 → RC.1
1F	BC	

20	45	Load by R5 and advance (Fetch first byte of Chip-8 Instruction)
21	AF	Put in RF.0 (temporary storage)
22	F6	Shift right 4 times (MSD to LSD position)
23	F6	
24	F6	
25	F6	
26	32	If D = 0 (i.e., if Op Code digit is zero), branch to 0044
27	44	
28	F9	OR Immediate with 50
29	50	
2A	AC	Put in RC.0 (RC now points to 005a, where "a" is MSD of CHIP-8 Instruction)
2B	8F	Get RF.0 (high byte of instruction)
2C	FA	AND with OF, OR with F0 (thus forming byte Fb, where "b" is the second hex digit in the CHIP-8 instruction, used in some instructions to designate VX)
2D	0F	
2E	F9	
2F	F0	
30	A6	Put in R6.0 (R6 becomes VX pointer)
31	05	Load via R5 (second byte of CHIP-8 instruction)
32	F6	Shift right 4 times
33	F6	
34	F6	
35	F6	
36	F9	OR with F0 and put in R7.0 (sets VY pointer)
37	F0	
38	A7	
39	4C	Load via RC and advance (Loads high byte of address for appropriate subroutine
3A	B3	Put in R3.1
3B	8C	Get RC.0 (=5(a+1))
3C	FC	Add 0F, put in RC.0 (points to low byte of start address for appropriate subroutine to execute CHIP-8 instruction
3D	0F	
3E	AC	
3F	0C	Load via RC

CODING FORM FOR RCA COSMAC PROGRAMS

Title J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 00 Programmer

40	A3	Put in R3.0
41	D3	Call subroutine designated by first digit of CHIP 8 instruction (if not zero)
42	30	Branch to 001B to fetch next instruction
43	1B	
		ROUTINE FOR FIRST DIGIT = 0
44	8F	Get RF.0 (high byte of instruction), AND with 0F to save LSD only
45	FA	
46	0F	
47	B3	Put in R3.1 (selects page on which sub-routine will be found)
48	45	Load via R5 and advance (2nd byte of inst.)
49	30	Branch to 0040 to call subroutine (00E0 for erase page, 00EE for return from subroutine, 0MMM for machine-language subroutine)
4A	40	
		SUBROUTINE TO TURN ON DISPLAY
4B	22	Decrement R2 (stack pointer)
4C	69	Turn display ON (interrupts will occur, controlled by routine at 8146)
4D	12	Increment R2
4E	D4	Return to 0042
4F	00	Filler
50	00	Filler
51	01	High bytes for pointer to start of sub-routines selected by first digit of CHIP 8 instructions (1 through F)
52	01	
53	01	
54	01	
55	01	
56	01	
57	01	
58	01	
59	01	
5A	01	
5B	01	
5C	01	
5D	00	
5E	01	
5F	01	

60	00	Filler
61	7C	Low bytes for sub-routine pointers
62	75	
63	83	
64	8B	
65	95	
66	B4	
67	B7	
68	BC	
69	91	
6A	EB	
6B	A4	
6C	D9	
6D	70	
6E	99	
6F	05	
		DISPLAY SUBROUTINE (1st Digit = D)
70	0C	Load by R6 (VX)
71	FA	Put last 3 bits in RE.1
72	07	
73	BE	
74	06	Load by R6
75	FA	AND with 3F (save lower 6 bits)
76	3F	
77	F6	Shift right 3 times (save middle 3 digits)
78	F6	
79	F6	
7A	22	Decrement stack
7B	52	Store in stack
7C	07	Load via R7 (VY)
7D	FA	AND with 1F (save 5 lowest bits)
7E	1F	
7F	FE	

1st Digit of Instruction → Start Address for Subroutine

1	017C
2	0175
3	0183
4	018B
5	0195
6	01B4
7	01B7
8	01BC
9	0191
A	01EB
B	01A4
C	01D9
D	0070
E	0199
F	0105

CODING FORM FOR RCA COSMAC PROGRAMS

Title J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 00 Programmer

80	FE	Shift left 3 times
81	FE	
82	F1	OR with top of stack
83	AC	Put result in RC.0
84	9B	Get RB.1, put in RC.1 (0X)--RC now points to start address of first byte of pattern.
85	BC	
86	45	Load via R5 and advance--fetches 2nd byte of CHIP 8 instruction.
87	FA	Put LSD in both RD.0 and R7.0 (No. of bytes in pattern)
88	0F	
89	AD	
8A	A7	
8B	F8	D0 → R6.0
8C	D0	
8D	A6	
8E	93	00 → RF.0
8F	AF	
90	87	Get R7.0 (No. of bytes); branch if D=0 to 00F3. (When branch occurs, display bytes have been processed and stored commencing at 0YD0).
91	32	
92	F3	
93	27	Decrement R7
94	4A	Load via RA (I pointer) and advance (Loads display byte)
95	BD	Put in RD.1
96	9E	Get RE.1 (3 LSB's of VX), Put in RE.0
97	AE	
98	8E	Get RE.0; if D = 0, branch to A4. (When branch occurs, display bytes will have been split into two parts in the event that display address did not coincide with a memory byte address.)
99	32	
9A	A4	
9B	9D	Get RD.1, shift right, and return to RD.1 (left portion of split display byte)
9C	F6	
9D	BD	
9E	8F	Get RF.0, ring shift right (picking up carry, if any, from step 9C), return to RF.0--these instructions form right portion of split display byte.
9F	76	
A0	AF	
A1	2E	Decrement RE
A2	30	Branch to 0098
A3	98	
A4	9D	Get RD.1 (left portion of display byte) and store via R6
A5	56	
A6	16	Increment R6
A7	8F	Get RF.0 (right portion of display byte) and store via R6
A8	56	
A9	16	Increment R6
AA	30	Branch to 008E
AB	8E	
AC	00	Wait for display interrupt
AD	EC	C → X (C points to start address for first new byte in display page)
AE	F8	D0 → R6.0 (points to first processed display byte)
AF	D0	
B0	A6	
B1	93	00 → R7.0 (R7.0 will be used as a marker for "collisions" between new and existing patterns.)
B2	A7	
B3	8D	Get RD.0 (no. of bytes remaining); if D = 0, branch to 00D9
B4	32	
B5	D9	
B6	06	Load via R6 (processed display byte)
B7	F2	AND with contents at current address in pattern on display page
B8	2D	Decrement RD
B9	32	If D = 0 (i.e., no "collision" occurred) branch to 00BE
BA	BE	
BB	F8	01 → R7.0 (marker to indicate that a "collision" has occurred)
BC	01	
BD	A7	
BE	46	Load via R6 (processed display byte) and advance
BF	F3	XOR with existing display byte

CODING FORM FOR RCA COSMAC PROGRAMS

Title J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 00 Programmer

C0	5C	Store via RC (in display page)
C1	02	Load from top of stack (3 LSB's of display page address), XOR with 07; if result is zero, branch to 00D2 (display pattern is at right-hand edge of display "window").
C2	FB	
C3	07	
C4	32	
C5	D2	
C6	1C	Increment RC
C7	06	Load via R6 (right portion of processed display byte), AND via R6 (existing contents of display page address); branch if result is zero (i.e., if there is no "collision") to 00CE
C8	F2	
C9	32	
CA	CE	
CB	F8	01 → R7.0 (marker to indicate that a "collision" has occurred)
CC	01	
CD	A7	
CE	06	Load via R6, XOR with contents already present at designated address on display page, and store via RC (on display page).
CF	F3	
D0	5C	
D1	2C	Decrement RC
D2	16	Increment R6
D3	8C	Get RC.0, Add 08, and return to RC.0
D4	FC	
D5	08	
D6	AC	
D7	3B	If DF = 0 (i.e., if next byte location remains on display page) branch to 00B3 (Program "falls through" this point when pattern reaches bottom of screen)
D8	B3	
D9	F8	FF → R6.0 (R6 points to Variable F)
DA	FF	
DB	A6	
DC	87	Get R7.0 ("collision" marker), store via R6 (i.e., as Variable F)
DD	56	
DE	12	Increment stack
DF	D4	Return to FETCH routine at 0042

ERASE DISPLAY PAGE--Inst. 00E0

E0	9B	0X -- RF.1
E1	BF	
E2	F8	FF -- RF.0
E3	FF	
E4	AF	
E5	93	00 -- D
E6	5F	Store via RF
E7	8F	Get RF.0; if zero, branch to 00DF for exit to FETCH
E8	32	
E9	DF	
EA	2F	Decrement RF
EB	30	Branch to 00E5
EC	E5	
ED	00	Filler

INST. 00EE--Return from Subroutine

EE	42	Load from stack and advance, put in R5.1
EF	B5	
F0	42	Load from stack and advance, put in R5.0 (R5 now points to next CHIP 8 instruction)
F1	A5	
F2	D4	Return to FETCH routine at 0042

(PART OF DISPLAY ROUTINE)

F3	8D	Get RD.0 (remaining no. of bytes in pattern), put in R7.0
F4	A7	
F5	87	Get R7.0; if zero, branch to 00AC (When branch occurs, RA (I Pointer) will have returned to its initial value)
F6	32	
F7	AC	
F8	2A	Decrement RA
F9	27	Decrement R7
FA	30	Branch to 00F5
FB	F5	
FC	00	Fillers
FD	00	
FE	00	
FF	00	

CODING FORM FOR RCA COSMAC PROGRAMS

for J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 01 Programmer

00	00	Fillers
01	00	
02	00	
03	00	
04	83	FINAL DECODING OF "F" Instructions
05	A5	Load via R5 and advance (2nd byte of CHIP 8 instruction)
06	A3	put in R3 (go to designated address)
		Instruction FX07 (Let VX = Timer)
07	98	Get R8.1
08	56	Store via R6 (i.e., as VX)
09	D4	Return to 0042
		Instruction FX0A (Let VX = Hex Key)
0A	F8	RC = 8195 (keyboard scanning subroutine)
0B	81	
0C	BC	
0D	F8	
0E	95	
0F	AC	
10	22	Decrement stack pointer
11	DC	Call keyboard scanning SR at 8195 (key entry is in D upon return)
12	12	Increment stack pointer
13	56	Store via R6 (i.e., as VX)
14	D4	Return to 0042
		Instruction FX15 (Set Timer to VX)
15	F6	Load via R6 (loads VX), put in R8.1
16	A8	
17	D4	Return to 0042
		Instruction FX18 (Set tone duration = VX)
18	F6	Load via R6 (loads VX)
19	B8	Put in R8.0 (tone timer)
1A	D4	Return to 0042
1B	64	Decimal 100 (Constants needed for Instruction FX33)
1C	0A	Decimal 10
1D	01	Decimal 1
		Instruction FX1E (Let I = I + VX)
1E	26	6 → X
1F	8A	Get RA.0
20	F4	Add VX
21	AA	Put in RA.0 (as updated I pointer)
22	3B	If DF = 0 (i.e., if updated I remains on the same memory page), branch to 0128
23	28	
24	9A	Increment RA.1
25	FC	
26	01	
27	BA	
28	D4	Return to 0042
		Instruction FX29 (Let I = 5-byte display pattern for LSD of VX
29	F8	81 → RA.1
2A	81	
2B	BA	
2C	06	Load via R6 (VX)
2D	FA	AND with 0F (save last digit), put in RA.0
2E	0F	
2F	AA	
30	0A	Load via RA (start address for 5-byte pattern of hex digit), put in RA.0
31	AA	
32	D4	Return to 0042
		Instruction FX33 (Let MI = 3-decimal digit equivalent of VX
33	E6	6 → X
34	06	Load via R6 (VX), put in RF.1
35	BF	
36	93	01 → RE.1
37	BE	
38	F8	1B → RE.0 (RE = 011B)
39	1B	
3A	AE	
3B	2A	Decrement RA
3C	1A	Increment RA (cancels prev. step upon first entry into pgm loop, but needed in later "passes" around the loop)
3D	F8	Store 00 via RA (I pointer)
3E	00	
3F	5A	

CODING FORM FOR RCA COSMAC PROGRAMS

Title J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 01 Programmer

40	[illegible]	Load via RE (Decimal 100, 10 or 1)
41	[illegible]	Subtract from M(R6)--i.e., subtract from VX
42	[illegible]	Branch if Minus to 4B
43	[illegible]	
44	[illegible]	Store result via R6
45	[illegible]	Increment memory location contents pointed to by RA (= I Pointer)
46	[illegible]	
47	[illegible]	
48	[illegible]	
49	[illegible]	Branch to 0140
4A	[illegible]	
4B	[illegible]	Load via RE and advance
4C	[illegible]	Shift Right
4D	[illegible]	If DF = 0 (i.e., if decimal 100's, 10's and 1's have not been processed), branch to 3C
4E	[illegible]	
4F	[illegible]	Get RF.1 (original value of VX)
50	[illegible]	Store via R6 (restores original value of VX)
51	[illegible]	Decrement RA twice
52	[illegible]	
53	[illegible]	Return to 0042
54	[illegible]	Filler
		Instruction FX55 (Let MI = V0:VX)
55	[illegible]	Decrement stack pointer
56	[illegible]	Get R6.0 (pointer for VX) and store in stack
57	[illegible]	
58	[illegible]	
59	[illegible]	F0 → R7.0
5A	[illegible]	
5B	[illegible]	Load via R7
5C	[illegible]	Store via RA (I pointer)
5D	[illegible]	Get R7.0
5E	[illegible]	XOR with top of stack (VX pointer)
5F	[illegible]	Increment R7

60	1A	Increment RA
61	3A	If D ≠ 0 (i.e., if R7 at Step 5E has not yet reached value of VX pointer), branch to 5B
62	5B	
63	12	Increment stack pointer
64	D4	Return to 0042
		Instruction FX65 (Let V0:VX = MI)
65	22	Decrement stack pointer
66	86	Get R6.0 (VX pointer)
67	52	Store in stack
68	F8	F0 → R7.0
69	F0	
6A	A7	
6B	0A	Load via RA (i.e., via I)
6C	57	Store via R7
6D	87	Get R7.0
6E	F3	XOR with top of stack (VX pointer)
6F	17	Increment R7
70	1A	Increment RA
71	3A	If D ≠ 0 (i.e., if R7 at Step 6E has not yet reached value of VX pointer), branch to 6B
72	6B	
73	12	Increment stack pointer
74	D4	Return to 0042
		Instruction 2MMM (do subroutine at MMM)
75	15	Increment R5 (point to next CHIP 8 instruction after return)
76	85	Get R5.0
77	22	Decrement stack pointer
78	73	Store in stack and decrement
79	95	Get R5.1
7A	52	Store in stack
7B	25	Decrement R5 (points to low byte of current instruction)
7C	45	Load via R5 and advance
7D	A5	Put in R5.0
7E	86	Get R6.0 (contains 2nd digit of current instruction)
7F	FA	

CODING FORM FOR RCA COSMAC PROGRAMS

Title J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 01 Programmer

80	0F	AND with 0F (save 2nd digit of Chip 8 instruction) and put in R5.1
81	B5	(R5 now points to first instruction of subroutine commencing at 0MMM)
82	D4	Return to 0042
		Instruction 3XKK (Skip if VX = KK)
83	45	Load by R5 and advance (KK → D)
84	E6	6 → X
85	F3	XOR (operands are KK and VX)
86	3A	If D ≠ 0 (i.e., if VX ≠ KK), branch to 0182
87	82	
88	15	Increment R5 twice (causing skip of next Chip 8 instruction)
89	15	
8A	D4	Return to 0042
		Instruction 4XKK (Skip if VX ≠ KK)
8B	45	Load by R5 and advance (KK → D)
8C	E6	6 → X
8D	F3	XOR (operands are KK and VX)
8E	3A	If D ≠ 0, branch to 0188
8F	88	
90	D4	Return to 0042
		Instruction 9XY0 (Skip if VX ≠ VY)
91	45	Load by R5 and advance
92	07	Load by R7 (VY → D)
93	30	Branch to 018C (operands for subsequent XOR operation will be VY and VX)
94	8C	
		Instruction 5XY0 (Skip if VX = VY)
95	45	Load by R5 and advance
96	07	Load by R7 (VY → D)
97	30	Branch to 0184 (operands for subsequent XOR operation will be VY and VX)
98	84	
		Instructions EX9E (Skip if VX = Key) and EXA1 (Skip if VX ≠ Key)
99	E6	6 → X (RX points to VX)
9A	62	Output VX to keyboard latch, increment R6
9B	26	Decrement R6 (cancel advance of prev. step)
9C	45	Load by R5 and advance (either 9E or A1 is loaded into D)
9D	A3	Put in R3 (go to designated address)
9E	36	If EF3 = 1 (i.e., if key matching LSD of VX is down) go to 0188
9F	88	
A0	D4	Return to 0042
A1	3E	If EF3 = 0 (hex key matching LSD of VX not pressed), branch to 0188
A2	88	
A3	D4	Return to 0042
		Instruction BMMM (Go to 0MMM + V0)
A4	F8	F0 → R7.0 (R7 points to V0)
A5	F0	
A6	A7	
A7	E7	7 → X
A8	45	Load by R5 and advance (Loads 2nd byte of instruction)
A9	F4	Add V0
AA	A5	Put in R5.0
AB	86	Get LSD of R6.0 (2nd digit of instruction)
AC	FA	
AD	0F	
AE	3B	If DF=0 (i.e., if there was no carry from addition operation at Step A9), branch to 01B2
AF	B2	
B0	FC	Add 01
B1	01	
B2	B5	Put in R5.1
B3	D4	Return to 0042
		Instruction 6XKK (Let VX = KK)
B4	45	Load by R5 and advance (KK → D)
B5	56	Store via R6 (as VX)
B6	D4	Return to 0042
		Instruction 7XKK (Let VX = VX + KK)
B7	45	Load via R5 and advance (KK → D)
B8	E6	6 → X
B9	F4	Add (D = VX + KK)
BA	56	Store via R6 (as updated VX)
BB	D4	Return to 0042
		Instruction 8XYN (ALU operations with VX and VY as operands)
BC	45	Load by R5 and advance (Loads 2nd byte of instruction)
BD	FA	AND with 0F (save 2nd digit)
BE	0F	
BF	3A	If D ≠ 0, branch to 01C4

CODING FORM FOR RCA COSMAC PROGRAMS

Title J. W. Wentworth's Analysis of VIP CHIP 8 Interpreter

MEMORY PAGE 01 Programmer

C0	C4	
C1	07	Load by R7 (VY → D)
C2	56	Store via R6 (as VX)
C3	D4	Return to 0042
C4	AF	Put in RF.0
C5	22	Decrement stack pointer
C6	F8	D3 → D
C7	D3	
C8	73	Store in stack and decrement
C9	8F	Get RF.0 (last digit of instruction)
CA	F9	OR with F0 (forms an instruction code in the ALU group--codes F1, F2, F3, F4, F5, F6, F7 and FE are valid)
CB	F0	
CC	52	Store in stack (stack now holds a 2-instruction routine)
CD	E6	6 → X
CE	07	Load by R7 (VY → D)
CF	D2	2 → D (calls routine developed in stack)
D0	56	Store result via R6 (as VX)
D1	F8	FF → R6.0 (points to VF)
D2	FF	
D3	A6	
D4	F8	00 → D
D5	00	
D6	7E	Ring Shift Left (moves DF to LSB)
D7	56	Store via R6 (as VF)
D8	D4	Return to 0042
		Instruction CXKK (Let VX = Random byte, masked by KK)
D9	19	Increment R9
DA	89	Get R9.0, put in RE.0 (NOTE: R9 is a special pointer for this random-number generator, and is incremented once for every TV scan by the interrupt routine.)
DB	AE	
DC	93	01 → RE.1 (RE now points to some byte on memory page 01)
DD	BE	
DE	99	Get R9.1 (Random byte resulting from last previous use of this instruction)
DF	EE	E → X

E0	F4	Add byte pointed to by RE
E1	56	Store via R6
E2	76	Ring shift right
E3	E6	6 → X
E4	F4	Add original byte formed at Step E0 to its ring-shifted version
E5	B9	Put in R9.1 (as starting point for next use of this instruction)
E6	56	Store via R6 (byte still un-masked)
E7	45	Load via R5 and advance (Loads 2nd byte of Chip 8 instruction, KK)
E8	F2	AND with byte pointed to by R6
E9	56	Store result via R6 (as VX)
EA	D4	Return to 0042
		Instruction AMMM (Let I = 0MMM)
EB	45	Load by R5 and advance (2nd byte of instruction)
EC	AA	Put in RA.0
ED	86	Get R6.0
EE	FA	Put 2nd digit in RA.1
EF	0F	
F0	BA	
F1	D4	Return to 0042
F2	00	Fillers
F3	00	
F4	00	
F5	00	
F6	00	
F7	00	
F8	00	
F9	00	
FA	00	
FB	00	
FC	00	Preliminary CHIP 8 instruction to precede every CHIP 8 program--calls routine at 00E0 to erase display page
FD	E0	
FE	00	Second preliminary instruction to precede every CHIP 8 program--calls routine at 004B to turn on display
FF	4B	

SUMMARY OF REGISTER FUNCTIONS IN CHIP 8 PROGRAMS

		Initial Setting
R0	DMA Pointer	----
R1	Program Counter (PC) for Interrupt Routine	8146
R2	Stack Pointer	0YCF*
R3	PC for Interpreter Subroutines	----
R4	PC for Interpreter FETCH AND DECODE routine	001B
R5	Pointer for CHIP 8 Instructions	01FC
R6	VX Pointer: in DXYN (Display) Instructions, also serves as pointer for processed display bytes, later as VF pointer	0Y--*
R7	VY Pointer for instructions involving VY; V0 Pointer for BMMM Instructions; R7.0 is a "scratch pad" register in DXYN, FX55 and FX65 Instructions	0Y--*
R8	Timers controlled by Interrupt Routine (R8.1 is a general-purpose timer; R8.0 is a tone and de-bounce timer)	----
R9	Special Pointer and "Scratch Pad" used in Random Number Generator utilized in CXKK Instructions--changed by Interrupt Routine	----
RA	I Pointer for CHIP 8 Instructions	----
RB	RB.1 is Display Page Pointer; RB.0 is "Scratch Pad" for Interrupt Routine	0X--*
RC	Temporary Pointer for FETCH AND DECODE Routine; Destination Address Pointer for DXYN Instructions; PC for Keyboard Scanning Subroutine in FX0A Instructions.	00--
RD	Both Sections Used as "Scratch Pad" Registers in DXYN (Display) Instructions	----
RE	Pointer for Constants Needed in FX33 Instructions; RE.1 is a "Scratch Pad" Register for DXYN (Display) Instructions	----
RF	Display Page Address Pointer for 00E0 (Erase) Instructions; RF.0 Used as "Scratch Pad" in FETCH AND DECODE Routine and also in DXYN Instructions	----

NOTE: Registers available for machine-language subroutines are R7, RC, RD, RE and RF, but subroutines themselves must provide any initial settings required--CHIP 8 instructions may alter these register settings, as indicated above.

*In basic VIP system with 2K RAM, 0X = 07 and 0Y = 06. In general, 0X is highest memory page and 0Y = 0X - 1.

COMMENTS

YES! I'd like to subscribe to the VIPER and receive all ten issues of this year's volume! I enclose $15.00 in full payment.

Name _____
(Please print or type)

Address _____

City _____ State _____ Zip _____

Cash, Check, Money Order Enclosed _____

MC/VISA/BAC Number _____ Exp. Date _____

MC Interbank No. _____

Required Credit Card Signature _____

☐ You may let other VIP owners in my area know I have a VIP, so they can contact me.

☐ I am interested in forming/joining (circle one) a VIP Users group

☐ I'd like to see articles in the VIPER about:

_____

_____

_____

_____

MAIL TO: VIPER; P.O. Box 43; Audubon, PA; 19407

Place Stamp Here

VIPER

P.O. Box 43

Audubon, PA 19407

VIPER

P.O. Box 43

Audubon, PA 19407