IPSO FACTO

Issue #12
June, 1979

(The Newsletter of the Association of Computer Experimenters)

TABLE OF CONTENTS PAGE

Editor: Bernie Murphy
Invaluable Assistants: Wayne Bowdish, Tom Crawford, Vic Sydiuk, Diane York, and all contributors to this issue.

Send Newsletter correspondence to: Bernie Murphy
102 McCrany Street,
Oakville, Ontario
Canada L6H 1H6

1979-1980 ACE EXECUTIVE COMMITTEE

PRESIDENT	Edward Fleet	5 Wells St. Toronto, ONT., M5R 1N8 416-925-5740
PAST PRESIDENT	Ken Bevis	220 Cherry Post Rd. Toronto, ONT., L5A 1H9 416-277-2495
TREASURER	Mike Franklin	24 Duby Rd, Acton, ONT., L7J 2P1 519-853-3421
ASSOCIATE EDITOR	Earle Laycock	2772 Hollington Cres., Mississauga, ONT, L5K 1E7 416-823-1345
ASSOCIATE EDITOR	Vic Kushnir	2640 Barnstone Cr., Mississauga, ONT., L5K 2C1 416-822-6505
ASSOCIATE EDITOR	Bob Silcox	1834 Green Meadow, Burlington, ONT., L7P 2Y8 416-632-6775
CONSULTING EDITOR	Bernie Murphy	102 McCraney St., Oakville, ONT., L6H 1H6 416-845-1630
DRAUGHTING	J. Myszkowski	99 Augusta St., Hamilton, ONT. 416-529-0250
MEMBERSHIP CO-ORDINATOR	Blair Gerrish	2110 Prospect St., Apt. 3, Burlington, Ont., L7R 1Y9 416-634-0268
TRAINING CO-ORDINATOR	Rod Dore	660 Oxford Rd., Unit 32, Burlington, ONT., L7N 1Y8 416-681-2456
HARDWARE CO-ORDINATOR	Fred Feaver	105 Townsend Ave., Burlington, ONT., L7T 1Y8 416-637-2513
PROGRAM CO-ORDINATOR	Fred Pluthero	1013 Upper Wellington St., Hamilton, ONT., L9A 3S4 416-389-4070
PUBLISHING COMMITTEE	Dennis Mildon	44 Wildewood Ave., Hamilton, ONT., L8T 1X3 416-385-0798
	John Hanson	955 Harvey Place, Burlingnton, ONT., L7T 3E9 416-637-1076

EDITOR'S REMARKS June 11, 1979

I am pleased to announce that a full slate of new executive was elected at the Annual General Meeting on May 8, 1979 (see page 2).

MEMBERSHIP STATUS

Issue #12 (this one), is the final issue of the 1978-1979 fiscal year. If you have already sent in $10.00 for 1979-1980, your cheque will be returned.

Because of the poor third class mail service (our current mailing practice), the 1979-1980 ACE executive has decided that IPSO FACTO will be sent via FIRST CLASS mail. I'm sure you will agree that getting your newsletter in a few days instead of 3 to 4 weeks or more is worth the extra cost.

The 1979-1980 membership dues are: $15.00 for residents of Canada and an extra $3.00 elsewhere. (ie. U.S.A dues are $18.00).

CLUB STATUS

The club is essentially shut down for the summer--no meetings, newsletters, etc. If you write us about something, please be patient as we are "on holidays" for the months of June, July and August.

KANSAS CITY CASSETTE INTERFACE STATUS

We have encountered a minor problem in getting our first interfaces going properly. The 4046 PLL seems to be a little tricky "to get going" when running at 5 volts. Once this problem is resolved (soon I hope) we will publish the schematic, artwork for a PC board and documentation on alignment and troubleshooting.

NEW EXECUTIVE

This is my final issue as Editor of IPSO FACTO. It has been fun and a lot of work! Please support your new 1979-1980 executive so that they may continue to provide you with the best microprocessor newsletter possible.

DATA ACQUISITION/CONTROLLER SUBSYSTEM

Dr. W.D. Greason
RR#8, River Road
London, Ont. N6A 4C3

This system uses 4 - 1852 ports to interface to the 1802 and allows for the A/D conversion of up to 8 input lines and the input of 16 logic level lines. Each time 16 bits of data are input, 8 control output lines are available, either in latched format or a programmed sequential format.

SYSTEM OVERVIEW

Figure no. 1 gives a block diagram of the system. Control port no. 1 is used to program the source of data to be input at data ports no. 1 & 2. It can select any 1 of 8 analog voltages to be inputted to the 3½ digit A/D converter which outputs 4 - 4 bit BCD nibbles. 2 to 1 channel multiplexers allow selection of these 16 bits or 16 bits of logic level digital signals.

The control port No. 1 uses one line to request data. In the case of analog data, the A/D converters end-of-conversion pulse latches the data; for digital data input, a strobe is output from control port No. 1 to latch the data.

Control port No. 2 allows programming of 8 output control lines. The latching of the 16 bits of data into data ports No. 1 & 2 initiates a counter which can be used to give a sequential presentation of control port No. 2's output bits. The output port can also be programmed to give a latched presentation of the 8 bits.

8 CHANNEL A/D CONVERTER

Refer to Figure No. 2. The 3½ digit A/D converter used is CMOS MC14433. It requires a 2.000V reference derived from the MC1403. It also requires a negative supply for V_{EE}. This is obtained using a MC14049 package acting as an inverter. The analog voltage to be converted is input through 8 to 1 multiplexer (MC14051) selection is done by control lines A,B,C. The inhibit forces A' high impedance state on the MUX's output line.

The A/D converter sequentially outputs the MSD through the LSD of the converted BCD data. The relevant strobe is active during the presence of the specified data. (DS1 = MSD,....DS4 = LSD). The EOC pulse goes high for one clock pulse at the end of each data conversion.

CONTROL PORT NO. 1

Refer to Figure No. 3. Control port No. 1 is an 1852 which outputs data from the 1802 data bus. 3 output bits are used to control the analogue voltage input multiplexer. One bit is used to determine if a 16 bit representation of an analogue voltage or 16 bits of logic level signals are to be the

CONTROL PORT NO. 1 (CONT'D)

input to the data ports. One line is used as a data request line. If the analogue mode is active, the first end of conversion (E.O.C.) pulse will give the data ready strobe to the rest of the system. If the digital input mode is active, an additional output line from the 1852 must be turned on and off to generate the same data ready strobe. An additional output line from the 1852 can act as a programmable strobe and can be used to latch the available 16 bits of digital data.

DATA PROCESSING SUBSECTION

Refer to Figure No. 4. The data is output from the A/D converter 4 bits at a time (BCD format) along with a strobe indicating data position (DS1 = MSD ... DS4 = LSD). The strobes are used to clock the appropriate data into the MC14042 latches. The 16 bits of latched data, representing a converted analogue voltage, form one channel of data input to the 2 to 1 multiplexer block, formed from the MC14053 triple 2 channel multiplexer/demultiplexers. 16 lines of logic level signals, D_0 to D_{15}, form the other data channel. Selection is done by the logic level applied to (X1). A logic "0" selects data from the MC14042 latches; a logic "1" selects the digital data, D_0 to D_{15}.

The output from the multiplexers is inputed into a block of HEX type D flip-flops (MC14174) which are clocked by the leading edge of a positive strobe applied to (X2). The output from the flip-flops represents 2 - 8 bit words of data to be input to the microprocessor's data bus via data ports No. 1 & 2 (1852).

DATA INPUT PORT SUBSYSTEM

Refer to Figure No. 5. Two 1852 ports are mode programmed to act as input ports. Data is strobed into the ports' 8 bit registers by a high level on the clock input (X9). The negative, high-to-low transition of the clock sets the service request flip-flop ($\overline{SR}$ = 0) and latches the data in the register. The $\overline{SR}$ output can be used to signal the microprocessor. It can be used with the interrupt line to request an interrupt or it can be connected to one of the 4 EF flag lines, which would then be sampled under software control. When CS1•CS2 = 1, the three state output drivers are enabled; the negative high-to-low transistion of CS1•CS2 resets the service request flip-flop $\overline{SR}$=1.

HARDWARE CONTROLLER

Refer to Figure No. 6. This block generates the clock required for the data input ports (X9) and generates the timing for the sequential strobe outputs.

The data request input (X3) sets flip-flop (A). The data ready input (X2) clocks the output of flip-flop (A) into flip-flop (B). This enable a clock input (X10) into the decade counter (MC14017).

The Q1 output is used as a clock for the data input ports (X9). The outputs of the counter are anded with the data programmed from control port No. 2's demultiplexer and presents a time sequential output of the 8 bits. A time sequential series of pulses is useful in hardware control applications. The Q_9 output from the counter resets both flip flops and the counter and inhibits the clock input.

In this example, the A/D's internal clock was used. However, and clock, including the microprocessor's or some division of it could be employed.

OUTPUT CONTROL PORT No. 2

Refer to Figure No. 7. This 1852 port, mode controlled as an output port, provides for a transfer of the 8 bits of data on the data bus when $\overline{CS1} \cdot CS2 \cdot CLOCK = 1$. The service request signal is generated at the termination of $\overline{CS1} \cdot CS2 = 1$ and will be present, 1 level, until the following negative, high to low transition of the clock. The service request signal is available as a hardware strobe.

The output of the port is input to a 1-to-2 demultiplexer block, configured from MC14053 devices. The least significant bit of the data is used to select the routing of the data. A logic "0" on D00 routes the data to the sequential controller for strobed sequential presentation. A logic "1" presents the data in a parallel 8 bit format (B_0 to B_7).

REFERENCES:

1. COS/MOS Integrated Circuits Data Book No. SSD-250, RCA Solid State.

2. CMOS Integrated Circuits Data Book, Motorola Semiconductor Products.

3. Use of the CDP 1852 8 Bit I/O Port with RCA Microprocessor Evaluation Kit CDP18S020 - ICAN - 6538, RCA Solid State.

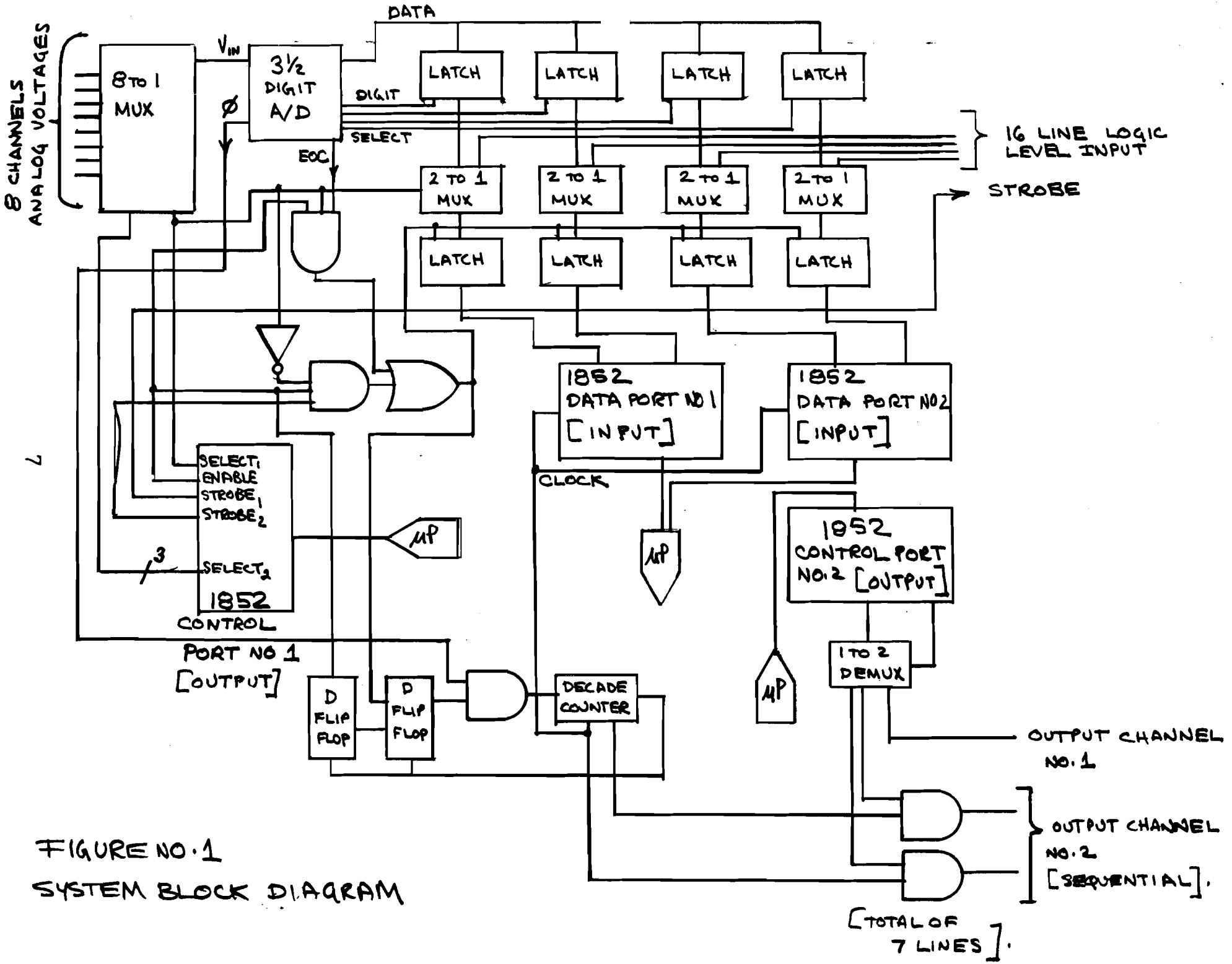
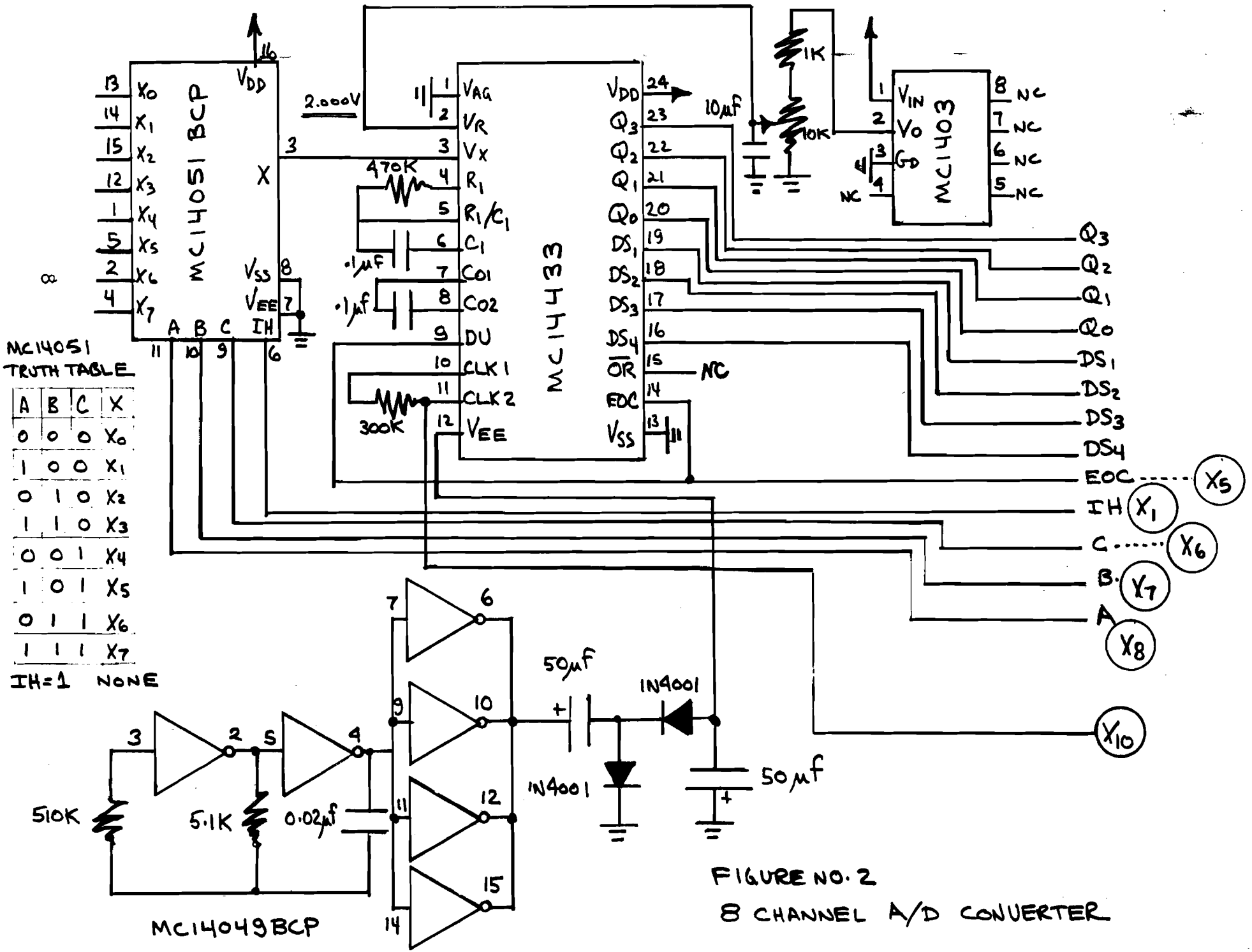


FIGURE NO. 1
SYSTEM BLOCK DIAGRAM

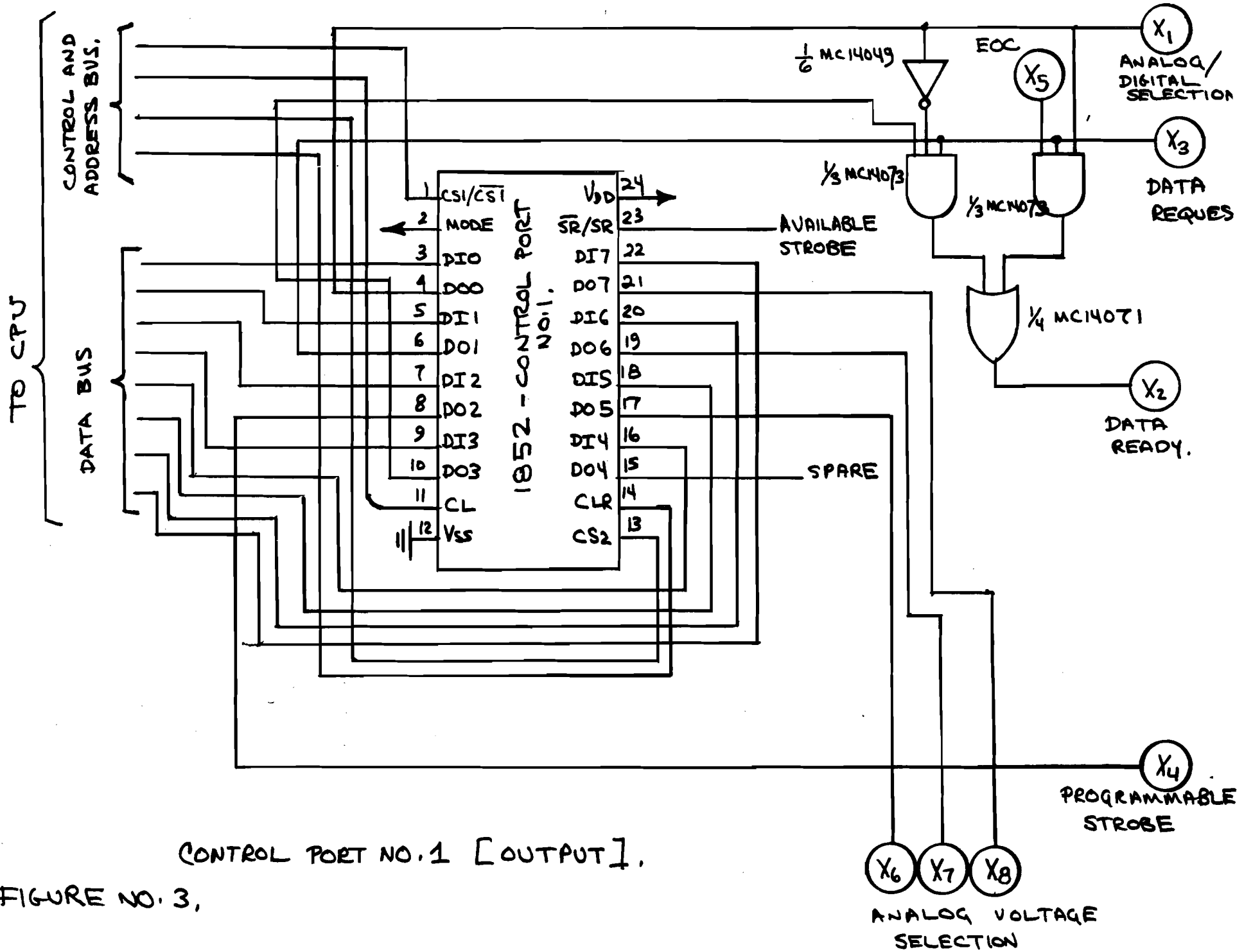


MC14051 TRUTH TABLE

A	B	C	X
0	0	0	X_0
1	0	0	X_1
0	1	0	X_2
1	1	0	X_3
0	0	1	X_4
1	0	1	X_5
0	1	1	X_6
1	1	1	X_7

IH=1 NONE

FIGURE NO. 2
8 CHANNEL A/D CONVERTER



CONTROL PORT NO. 1 [OUTPUT].

FIGURE NO. 3,

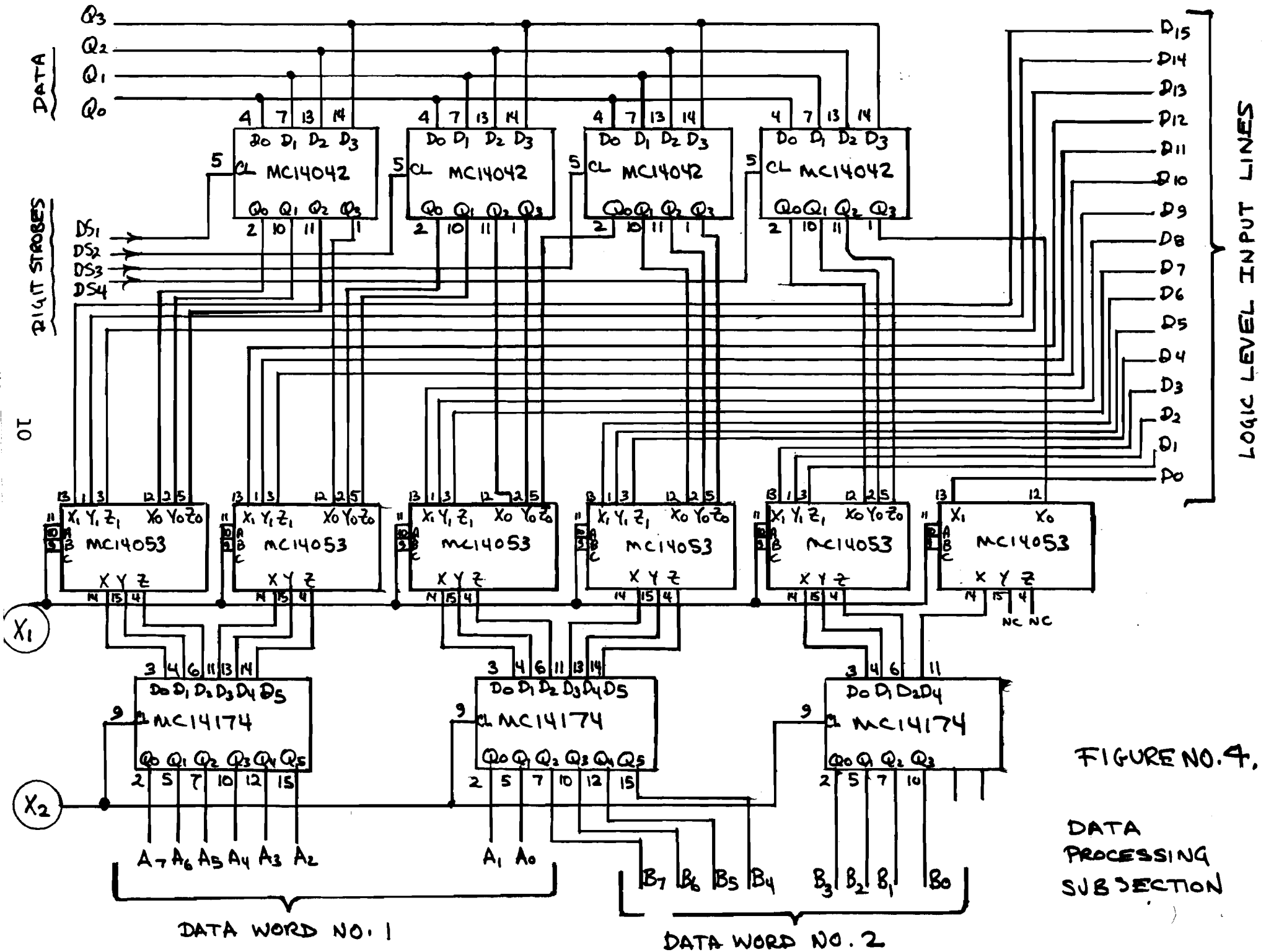
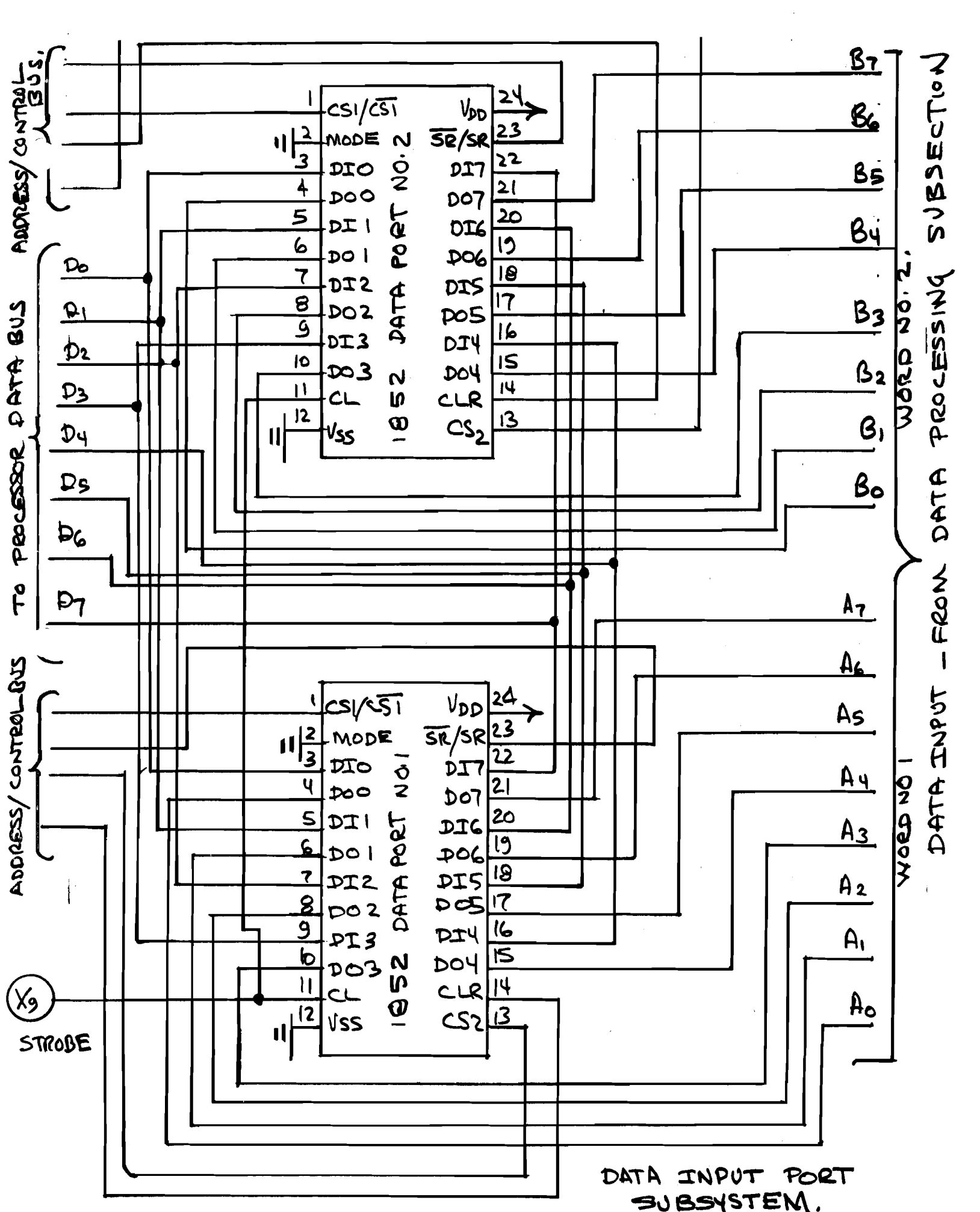


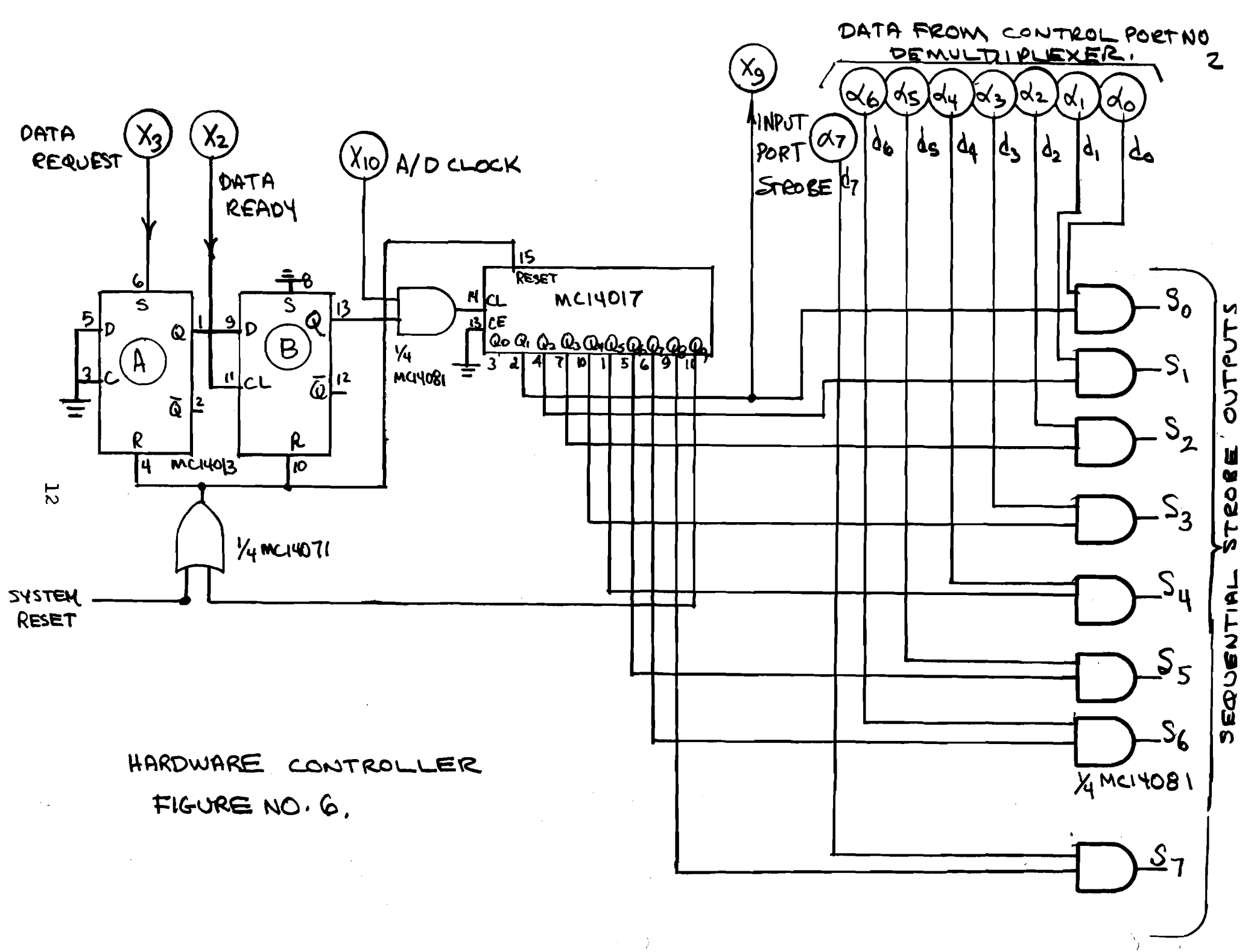
FIGURE NO. 4.

DATA PROCESSING SUBSECTION



DATA INPUT PORT SUBSYSTEM.

FIGURE NO. 5.



HARDWARE CONTROLLER

FIGURE NO. 6.

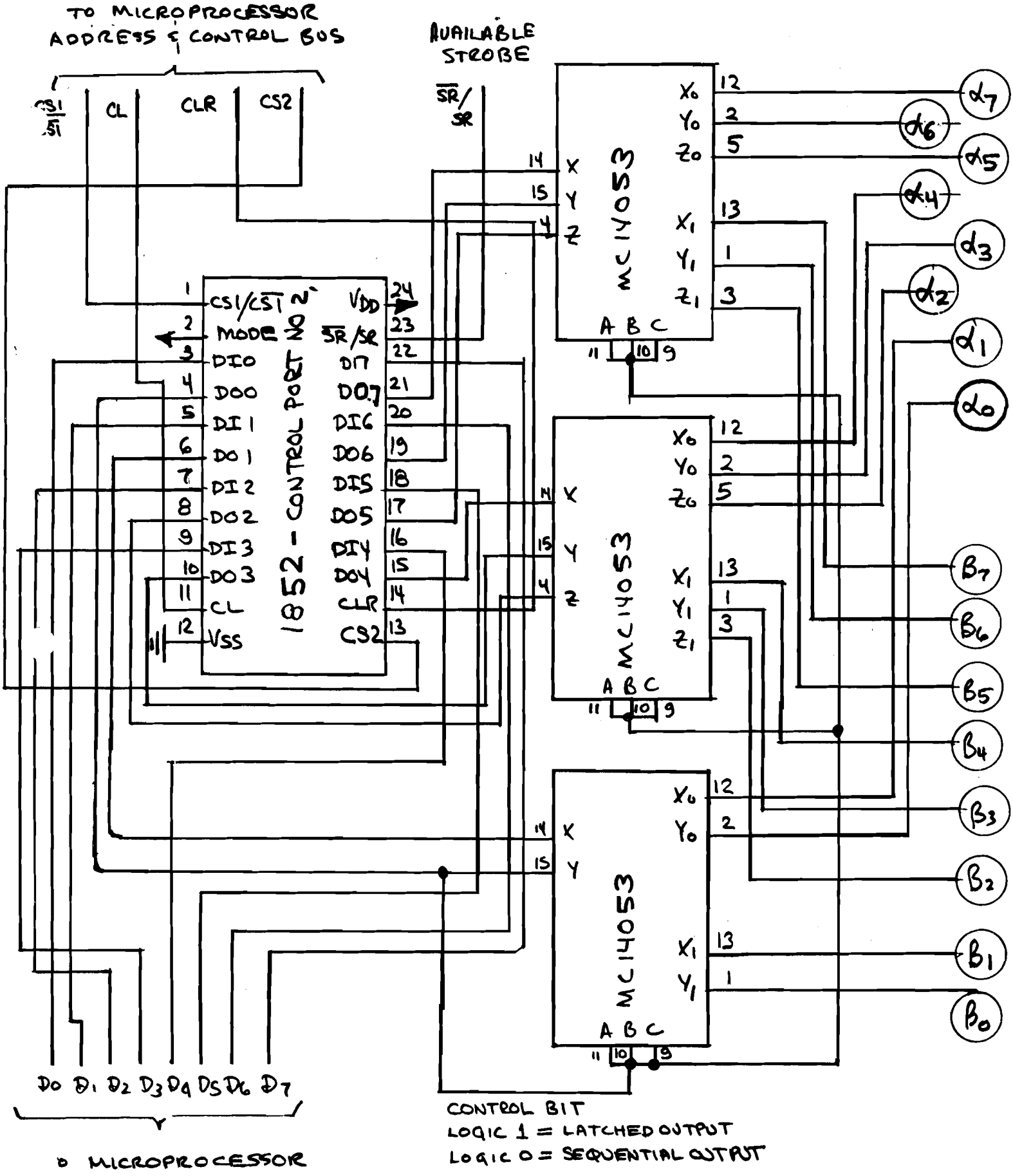


FIGURE NO 7: OUTPUT CONTROL PORT NO 2.

VIDEO VOICE

D. Roberts
660 Laurier Blvd.
Brockville, Ontario K6V 5X8

Here is the listing of a program that samples EF2 and displays transitions of it on the video display.

There are two versions: 1) For 256 Byte Systems
2) For Multi-page Systems
(NOTE: Both will work on multi-page systems.)

I have left it up to the user to optimize the program:

MULTI-PAGE VERSION

```
0000 90B1 B2B3 B4F8 2DA3 F8FF A2F8 11A1 D372;
0010 7022 7822 52C4 C4C4 F801 B0F8 00A0 80E2;
0020 E220 A0E2 20A0 E220 A03C 1E30 0FE2 69F8;
0030 00A4 F801 B4F8 08AB F800 AA8A 3540 F901;
0040 7EAA 2B8B 3A3B 8A76 5414 3032 0000 0000
```

256 BYTE VERSION

```
0000 90B1 B2B3 B4F8 2DA3 F852 A2F8 11A1 D372;
0010 7022 7822 52C4 C4C4 F800 B0F8 00A0 80E2;
0020 E220 A0E2 20A0 E220 A03C 1E30 0FE2 69F8;
0030 53A4 F800 B4F8 08AB F800 AA8A 3540 F901;
0040 7EAA 2B8B 3A3B 8A76 5414 843A 3230 2FFF
```

Have fun! (Try looking at a data cassette!)

FOR SALE

STATIC MEMORY IC'S AVAILABLE

2102's available at an outstanding price. These are sold as 1 us or better, BUT a sample batch of 12 K (96 chips) was 100% good at 575 ns, using a comprehensive test program.
When these are gone--that's it. This offer is good only as long as the supply lasts.
Price: $6.50 per K (8 pieces). For out of town, I'll likely have to add a bit for postage & packing.
Call GRAHAM IDE, (613)828-7039, 43 Quinpool Cres., Nepean, Ont., K2H 6H9.

Richard Moffie

ELFWRITER

Elfwriter is a program that will let you write and display messages or other information on the T.V. screen with your Elf system. The displayed information can be saved on tape and then read back onto the screen for display or editing.

Elfwriter has these features:

1. The program will run on a basic Elf, as long as it has 1K of RAM and the 1861 video chip.
2. It is written for hex keyboard, but an ASCII keyboard could be used with only a few minor changes.
3. 16 characters per line are displayed with either 5 or 10 lines displayed at a time.
4. The basic program fits in ½K of RAM, so that all remaining memory (2 pages in a 1K system, 14 pages in 4K system) can be used for display.
5. There are some editing features: change display page up or down, erase line, erase display area, carriage return-line feed.
6. There is some unused space for adding features and special character patterns.

USING ELFWRITER

Load the program at 0000 - 01FF. When Reset, Run is pressed, Elfwriter is ready to use, with page 2 displayed (Pages 0 and 1 contain the program - Don't write onto them!). To clear the entire display area - all pages - enter 'BB'. To write, enter the ASCII code (20-5F) of the desired character and press Input key. Continue in this manner until you have written the desired information. At the end of a line, the program will begin writing at the beginning of the next line. When a page is full, the writing will begin at the top of the next page and the display will then show the new page.

To begin a new line, or to skip lines, enter '0D' (ASCII code for carriage return). To erase the current line, enter 'EE' - a carriage return is automatic. To change the page being displayed press 'CC' for the next higher page, or 'DD' for the next lower page. To display messages after they are written, press Reset, Run and page 2 will be displayed.

A totally white screen means a non-existent page of memory is being displayed - go back (using DD) to a memory page that you can use. A display that does not show character patterns that you have written means you are on page 0 or page 1 (which contain the program). DO NOT TRY TO WRITE ON THESE PAGES.

If you 'get lost' and don't know which page you are on, pressing Reset, Run will put you on page 2 and no information will be lost. When pages are changed, the writing begins at the beginning of the top line. To space or skip over letters already written, use '20' (ASCII space) to skip over characters.

If using the 10 line display format, you may only write on the top 5 lines displayed. Change page to write on the other 5 lines. The display format can be changed from 5 to 10 lines at any time by changing byte 00D9 as indicated in the listing.

ELFWRITER - Register Usage

R0 - Display Page Pointer (Initially 0200)
R1 - Address of Video Interrupt Routine (00C3)
R2 - Stack pointer (00FF)
R3 - Main program pointer (0100)
R4 - Unused
R5 - Unused
R6 - Pointer to memory containing display page (00CB)
R7 - No. of bytes/line of characters - Used for erasing line (0028)
R8 - Counts erased bytes - for erasing display
R9 - No. of lines/character (0005)
RA - Points to current byte in character being transferred
RB - " " first " " " " "
RC - No. of bytes/line (0008)
RD - Points to first byte in current display position
RE - Flag for transfer logic
RF - Points to current byte in current display position

TRANSFER LOGIC

The program is basically an expanded version of the original T.V. Typewriter Jr. that fit in a basic 256 byte Elf, and uses much of the same logic. The program can be found in Elf of the Valley Newsletter - August 1978, or in more detail in Questdata #4.

The program stores portions of two characters in each byte. The 4 high bits store one character and the 4 low bits store the next character in sequence. Similarly, there are two characters displayed per byte across the screen to get 16 characters on a line of 8 bytes. In order to get the correct character and display it in the correct position, the Q line and E register are used as follows:

If Q=0 get high 4 bits If Q=1 get low 4 bits
If RE = 0 display on high 4 bits
If RE = 1 display on low 4 bits

These are tested and cause the program to proceed to the correct sequences:

Q	RE	Jump to
0	0	016A
0	1	0165
1	0	0179
1	1	0174

The use of the above information along with the memory map should make the logic easy to understand and the program easy to modify as desired.

ELFWRITER - MEMORY MAP

Location	Function
0000-001D	Initialize Registers
0020-00BF	Character Patterns (20-47 for ASCII 20-2F, 48-6F for ASCII 30-3F, 70-97 for ASCII 40-4F, 98-BF for ASCII 50-5F)
00C0-00DE	Video Interrupt Routine
00DF-00FB	Unused
00FC-00FF	Stack Area
0100-010F	Initialize and Reset Registers
0110-0114	Keyboard Input Routine
0115-012D	Test for special characters
012E-0139	Carriage Return/Line Feed
013A-013F	Unused
0140-015B	Routine to point to correct character pattern - Converts ASCII code to pattern location
015C-019C	Transfer logic - Selects left or right character and left or right display area, then transfers character to correct location.
01A0-01B0	Blank screen subroutine
01B1-01B7	Automatic linefeed at end of line
01B8-01C9	Erase line subroutine
01CA-01CF	Change page (higher)
01D0-01D5	Change page (lower)
01D6-01E1	Automatic change page when full
01E2-01FF	Unused

ELFWRITER - Listing

Address	Data														
0000	F8	00	A3	AE	B1	B2	B6	BA	BB	BE	F8	01	B3	F8	02	B8
0010	BD	BF	56	F8	C3	A1	F8	FF	A2	F8	CB	A6	30	C0	xx	xx
0020	02	55	75	22	22	22	00	01	02	07	61	22	41	72	00	01
0030	02	05	72	70	41	27	07	02	00	07	34	20	41	02	20	04
0040	02	05	75	20	22	02	40	24	72	77	57	77	77	00	10	47
0050	52	11	54	41	55	22	27	21	52	77	77	71	77	00	40	12
0060	52	41	11	51	51	22	27	20	72	77	17	71	77	04	10	42
0070	77	77	67	77	57	75	45	77	15	54	54	44	52	25	47	55
0080	77	74	56	65	72	26	47	55	55	54	54	45	52	25	45	55
0090	75	77	67	47	57	65	75	57	77	77	75	55	55	76	43	00
00A0	55	54	25	55	55	14	41	00	75	67	25	57	22	24	21	00
00B0	47	51	25	57	52	44	11	00	47	57	27	25	52	76	13	07
00C0	D3	72	70	22	78	22	52	C4	C4	C4	F8	02	B0	F8	00	A0
00D0	80	E2	E2	20	A0	E2	20	A0	E2	20	A0	3C	D0	30	C1	xx
00E0	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
00F0	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
0100	E2	69	F8	10	AD	F8	08	AC	8D	AF	F8	05	A9	F8	20	AB
0110	3F	10	37	12	6C	FB	BB	32	A0	F0	FB	CC	32	CA	F0	FB
0120	DD	32	D0	F0	FB	EE	32	B8	F0	FB	0D	3A	40	8D	FA	F0
0130	FC	30	AD	33	D6	F8	00	AE	30	05	xx	xx	xx	xx	xx	xx
0140	F0	FF	20	FF	10	B9	3B	4F	8B	FC	28	AB	99	30	43	02
0150	FA	0F	52	F0	F6	52	3B	5B	7B	30	5C	7A	8B	F4	AA	E2
0160	31	71	8E	3A	6A	0A	FA	F0	30	7E	0A	F6	F6	F6	F6	30
0170	7E	8E	32	79	0A	FA	0F	30	7E	0A	FE	FE	FE	FE	EF	F1
0180	5F	E2	8F	FC	08	AF	8A	FC	08	AA	29	89	3A	60	8E	FB
0190	01	AE	3A	95	1D	3A	08	2C	8C	32	B2	30	08	xx	xx	xx
01A0	F8	00	A8	F8	00	58	18	98	FB	10	3A	A3	F8	02	B8	30
01B0	02	E2	8D	FC	28	AD	30	33	F8	28	A7	8D	FA	F0	AD	A8
01C0	F8	00	58	18	27	87	3A	C0	30	35	06	FC	01	56	30	DA
01D0	06	FF	01	56	30	DA	9D	FC	01	56	BD	BF	B8	F8	00	AE
01E0	30	02	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx
01F0	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx	xx

Notes:

1. Locations marked xx are unused (except for stack - 00FB - 00FF) and can be used for additions to program.

2. For 5 line display, leave 00D9 as 20. For a 10 line display, change it to 80.

3. Location 01A9 should contain the number of pages of RAM in hex. It is set for 4K in above listing. For 1K, set to 04, etc.

Richard Moffie 7003 Longridge Ave. No. Hollywood, CA 91605

SIMON ELF

Last summer, Milton Bradley Inc. introduced an electronic game called Simon (TM of Milton Bradley) that became quite popular. Simon generated a sequence of tones and corresponding colors that must be remembered and matched by the player or players. As the number of tones gets higher and higher, it becomes more difficult to remember the correct sequence and do as 'Simon Says' without making an error and losing the game.

Here is a program that will let you 'Simonize' your Elf computer and make your Elf the life of the party. The program will run on the basic Super Elf or any similar system, provided you have a speaker attached to the Q line. No additional memory is needed since the program is quite compact and takes only 165 bytes plus the stack area.

To play the game, run the program and push the I switch. The computer will then generate one of 8 random tones and display the corresponding number (from 0 to 7). You must then press the correct key (matching the previously displayed number) to generate the same tone. If you are correct, 'charge' will be played and the number of tones in the sequence is displayed. If you choose the wrong key, a low buzz will sound and you must then start over with a single note (by pressing I). When you enter the numbers, only one key need be pressed, so that if 05 is displayed, you need only enter the 5.

To play the next round (assuming you were successful), press I and a sequence of 2 notes will be played, which you must then match in order. If these are correct, 3 notes will be played, then 4, 5 ... until you can no longer remember and match the sequence. The stack area is large enough for 40 notes, but if you can get past 9 or 10, you're doing great! You can play this game alone or with a group where players alternate turns.

There are a couple of easy modifications to the program if desired: If you wish to have only 4 different tones rather than 8 just change bytes 1C and 55 from 07 to 03. Also, if you don't want each tone played as you enter it, change byte 51 from D4 to C4.

One of the interesting features of this program is the random tone generator. This is done by repeatedly incrementing Register 7 whenever the program is waiting for I to be pushed, and happens tens of thousands of times a second, making the value of the last 3 bits (R7 is 'AND'ed with 07) impossible to determine and totally independent of any previous values. This has been used in many other programs to get random numbers, but happens to be ideal for this program since a new random number can be generated for the next round whenever a response is entered for the current round. The one additional number needed each round is generated when I is pushed to begin the new round.

Try running and examining this program. It makes use of many of the 1802's features and shows again that a lot of programming can be done even with a limit of 256 bytes. With all the extra space left, you might even try adding a few features of your own.

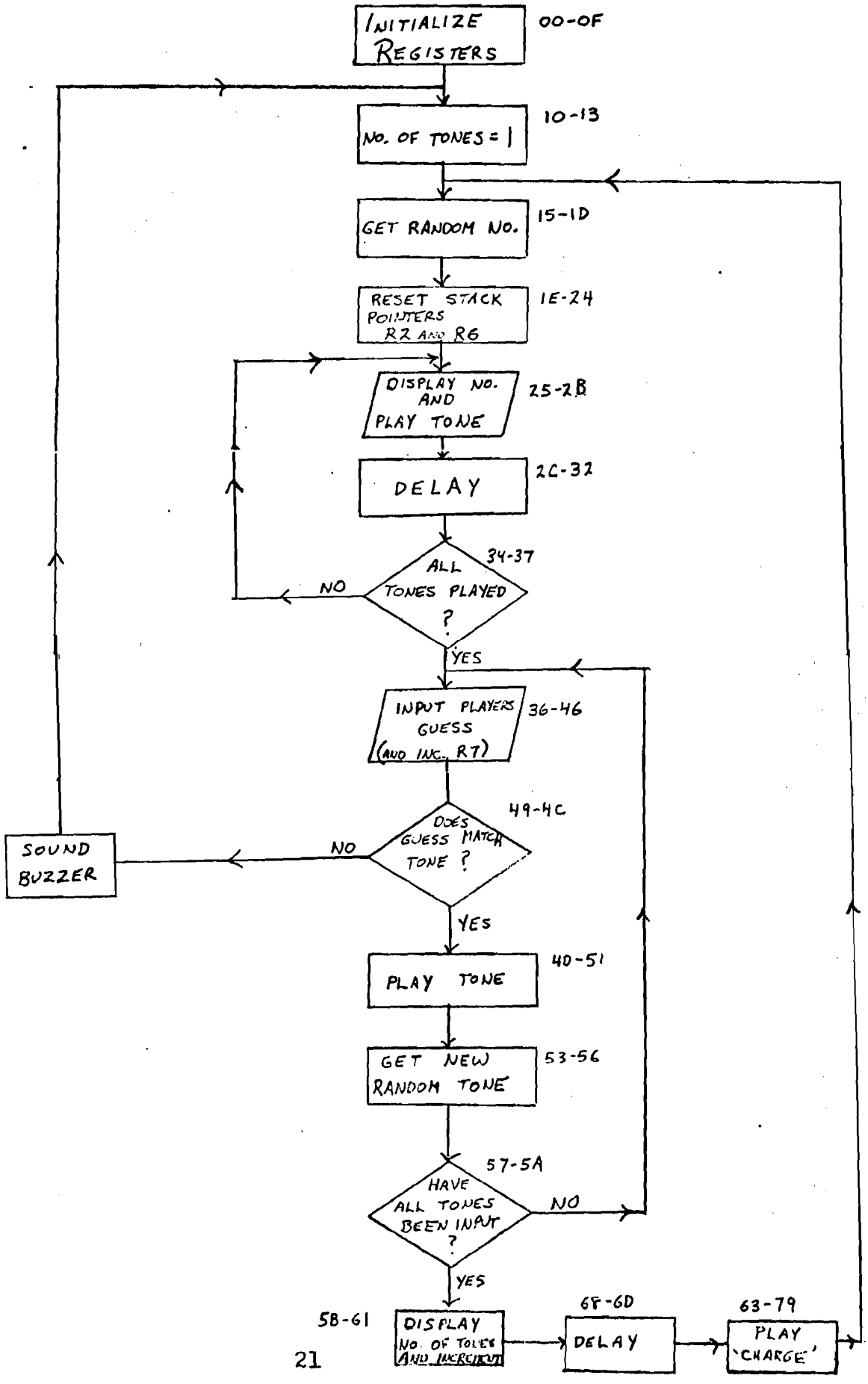
Richard Moffie 7003 Longridge Ave. No. Hollywood, CA 91605

SIMON ELF - Program Listing

Address	Data	Comments
00	90 B2 B3 B4 B6 B9	Set high register bytes to correct page
06	F8 15 A3	Set R3 to start of main program
09	F8 7B A4	R4 = Loc. of note subroutine
0C	F8 FF A6 AF	Set stack pointer & tone duration
10	F8 01 AA AB	Set no. of tones
14	D3	R3 = program counter
15	17 3F 15 37 18	Inc. R7 while waiting for I to be pushed
1A	87 FA 07 56	Get random no. 'AND' with 07 & store
1E	F8 D7 A2 F8 FF A6	Reset stack pointers
24	E6 64 26	Display tone number
27	F8 95 F4 A9 D4	Point to tone constant & call tone sub.
2C	F8 40 BE	Set delay constant
2F	2E 9E 3A 2F	Dec. RE until 0 - Delay
33	26 2A 8A 3A 25	Dec. stack ptr. Check for all tones played
38	8B AA F8 FF A6	Reset no. of tones & stack ptr.
3D	17 3F 3D 37 40	Inc. R7 while waiting for input
42	E2 6C FA 0F 52	Get guess input & store on input stack R2
47	64 22 E6	Display input, change stack ptr. to R6
4A	F3 3A 8F	XOR computer tone & input - to 8F if wrong
4D	F8 95 F4 A9 D4	Point to tone constant & call tone sub.
52	22 87 FA 07 73	Get new rand. from R7 - to stack
57	2A 8A 3A 3D	If all tones not input, go to 3D
5B	8B 56 64 26	Display no. of tones in round
5F	1B 8B AA	Inc. no. of tones
62	F8 08 A8	Set no. of tones in 'charge'
65	F8 9D A9	Pointer to 'charge' note constants
68	F8 40 BE	Set delay constant
6B	2E 9E 3A 6B	Dec. RE until 0 - Delay
6F	F8 80 AF D4	Set note duration & call tone sub.
73	19 28 88 3A 6F	If all 8 notes not played, go to 6F
78	30 15	Go back for next round
7A	D3	Return from subroutine to main
7B	09 A1	Tone constant to R1
7D	7B	Q on
7E	81 FF 01 3A 7F	R1 to D, sub. 1 until 0
82	7A	Q off
83	81 FF 01 3A 85	R1 to D, sub. 1 until 0
89	2F 8F 3A 7D	Dec. tone duration, if not 0 go to 7D
8D	30 7A	When duration = 0, return to main
8F	F8 0D A9	R9 points to 0D (which contains FF)
92	D4 30 10	Call tone sub - will sound buzzer - to star
95	58 50 48 40	Tone constants for random notes
99	38 30 28 20	
9D	40 30 25 20	Tone constants for 'charge'
A1	20 25 20 20	
B0-D7		Input stack
D8-FF		Random tone stack

Richard Moffie 7003 Longridge Ave. No. Hollywood, CA 91605

SIMON ELF - FLOWCHART



MORE ON SUBROUTINE CALLING CONVENTIONS

WAYNE BOWDISH
149 EAST 33rd
HAMILTON, ONT

This article describes a subroutine CALL/RETURN technique which may be useful in some special cases. I'll briefly describe the circumstances which caused this technique to be developed and then describe the CALL/RETURN technique.

I was writing a subroutine package which contained some 16 useful (I hope) functions. These are called by using the standard SCRT method. Arguments are passed in the bytes immediately following the call.

```
ex.  SEP  R4
     2 BYTE SUBROUTINE ADDRESS
     DATA BYTES ( 0 to 6 depending on routine )
     ... return here with SEP  R5
```

These routines share a lot of common code. For example, 9 of the routines expect the first data byte to contain a value in the range 0 to F (hex). If this value is not within this range an error return is made with DF set. Otherwise a table address is calculated and the routine continues on with its functions. Obviously this code should be made into a subroutine.

There are many ways of calling subroutines. The RCA CDP1802 Users Manual discusses several methods, each with its good and bad points. I'll discuss each one seperately.

SEP Register Technique

This requires a dedicated register for each subroutine. Registers are rather scarce and since the subroutine package does not know which registers are being used by the calling programs, these registers would have to be saved. the costs are:

```
  4  bytes to save each register ( on stack)
  6  bytes to initialize the register
  1  byte per call
  5  bytes to restore the register before returning
----
 16  bytes total
```

This code, or at least part of it would be required in each subroutine. This doesn't appear to be very space or time efficient.

MARK Subroutine Technique

This method requires the same overhead that the SEP technique required plus an extra MARK instruction per call.

SCRT Subroutine Technique

This method is very flexible and only requires 3 bytes per call but there are disadvantages, the main one being speed.The SCRT call routine requires 15 or more instructions (depending on the features) and the SCRT return routine requires 12 or more instructions. Refering back to the example of a routine to check a value and return an address,

the code consisted of 17 instructions (23 bytes), less than the SCRT CALL/RETURN overhead! Rather than double the execution time of the routine I decided to develop a different CALL/RETURN technique.

The technique had to have the following characteristics:
- be fairly space/time efficient
- not require the use of dedicated registers
- allow passing of data on the stack and in the D-register
- return data in register 14 (it was already defined for use by the subroutine package

The technique has the following restrictions which were acceptable in this special case:
- data cannot be returned in the D-register
- returning data on the stack is tricky
- the subroutine return point must be on the same page as the subroutine (sub-subroutine?) exit code

Heres how the technique works:
- the page address of the return point is placed on the stack (3 byte overhead)
- data may be placed on the stack or other functions may be performed
- the subroutine is entered by a branch instructions (long or short)
- the subroutine performs its required functions
- to exit the routine removes the return point address from the stack and places it in the low byte of the present program counter. This effectively transferres control back to the return point.

Lets look at an example. The addresses are merely for illustrative purposes.

```
0010  LDI   #30     ; LOAD RETURN POINT ADDRESS INTO D-REG.
0012  STR   SP      ; SAVE RETURN ADDRESS ON STACK
          .         ; MORE DATA MAY BE PLACED ONTO THE STACK
          .         ;  OR FUTHER COMPUTATATION MAY BE DONE
          .
0020  BR   #40      ; ENTER THE SUBROUITNE
          .
          .
          .
0030                ; ROUTINE WILL EXIT BACK TO THIS POINT
          .
          .
          .
0040                ; SUBROUTINE STARTS HERE
          .
          .         ; PERFORM REQUIRED FUNCTIONS
          .
      LDX           ; GET RETURN ADDRESS FROM STACK
      PLO   PC      ; AND SET PC TO RETURN POINT
```

The addresses were only for illustrative purposes. The only restriction is that the "PLO PC" instruction must be on the same page as the return point.

Subroutines may also be chained or threaded together using this method. For example, if you want to execute SUB1 then SUB2 then SUB3 and then return to the mainline the following code could be used:

```
     LDI   RETURN
     STXD
     LDI   SUB3
     STXD
     LDI   SUB2
     STX
     BR    SUB1
       .
       .
       .
RETURN:
```

and the return code in the subroutines would be "LDXA / PLO PC". Data could also be intermixed (carefully) with the return addresses.

In summary, this technique is not very flexible but it is fairly space/time efficient and may, in special cases, be of some use.

2708 EPROM PROGRAMMER

Berry Erick
28 Ridge Street
Dallas, Pa. 18612

I have designed a circuit and software to program 2708's from my S-100 Elf, Both hardware and software compatable with both ELF II and Super ELF. I have ELFBUG on PROM, Netronics Monitor on PROM and Quest's Super Monitor on PROM, as well as other programs I wrote. I use Netronics TINY BASIC but do not have it on PROM at this time, as I am extending it, and am also looking forward to a "full" BASIC coming forth some day. I hope that day is not too far distant. I think now that Netronics has the Video Board, and the editor software out, the Elf will gain even more as one major drawback is its' graphic type character generator.

2½ ports are used to latch the data lines for data and addresses for the 2708 being programmed. The 62,63,65 output pulses are used to latch the data, so NO,N1,N2 must be decoded, if not done so as yet. The address lines do not have to be decoded for programming, but to use the PROM they need to be.

The software:

```
0000 90  B3
0002 F8  06  A3
0005 D3                    PC=R3
0006 F8  C8  A9            ...loop 200 times
```

The software: (cont'd)

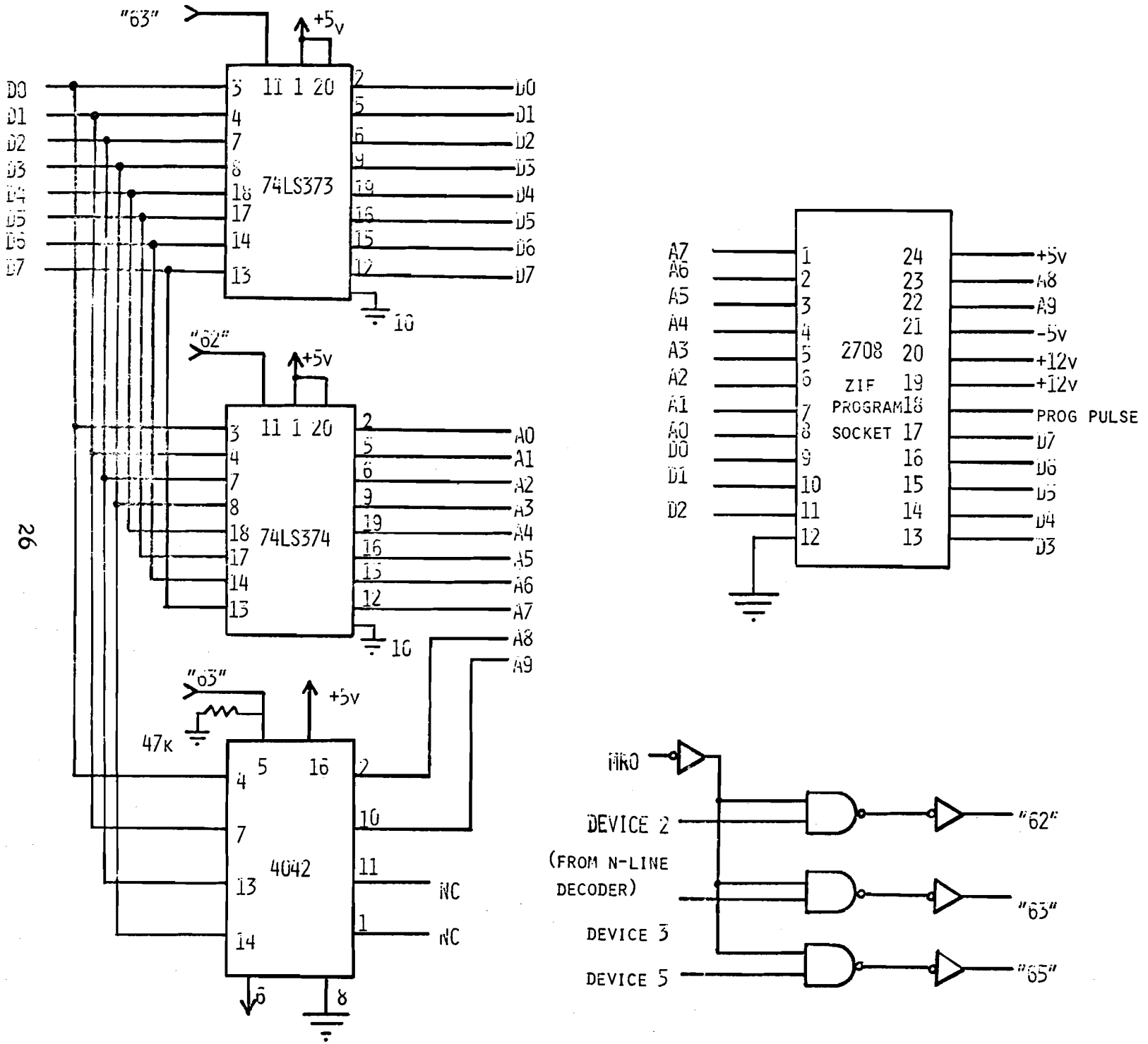
```
0009 F8  98  BD  AD      ...Stack addr=9898...change to suit system
000D F8  10  BA          ...Hi addr of data...change to suit system
0010 F8  00  AA  BC  AC  ...RA=1000(data) RC=0000(address)
0015 F8  04  BB
0018 F8  00 AB           ...RB=0400...# of bytes
001B ED
001C 89
001D 5D                  ...# of loops left to X
001E 64                  ...show # of loops left
001F 2D
0020 9C                  ...Hi addr to D
0021 5D                  ...Put it in X
0022 65                  ...Output & latch it
0023 2D
0024 8C                  ...Lo addr to D
0025 5D                  ...Put it in X
0026 62                  ...Output & latch it
0027 2D
0028 1C                  ...inc. addr
0029 EA
002A 63                  ...Output data & latch it. Also program pulse
002B F8  20  A1          ...short delay
002E 21
002F 81
0030 3A  2E
0032 2B                  ...Dec byte counter
0033 9B                  ...Hi byte counter to D
0034 3A  1B              ...Back to program next address
0036 8B                  ...LO byte counter to D
0037 3A 1B               ...Back to program next addr if not done
0039 19                  ...Dec loop counter
003A 89                  ...Loop counter to D
003B 3A  09              ...Start over if not done
003D F8  F1  B0          ...Hi ELF-BUG Monitor address
0040 F8  30  A0          ...Lo ELF-BUG Monitor address
0043 E0
0044 D0                  ...Return to monitor
```

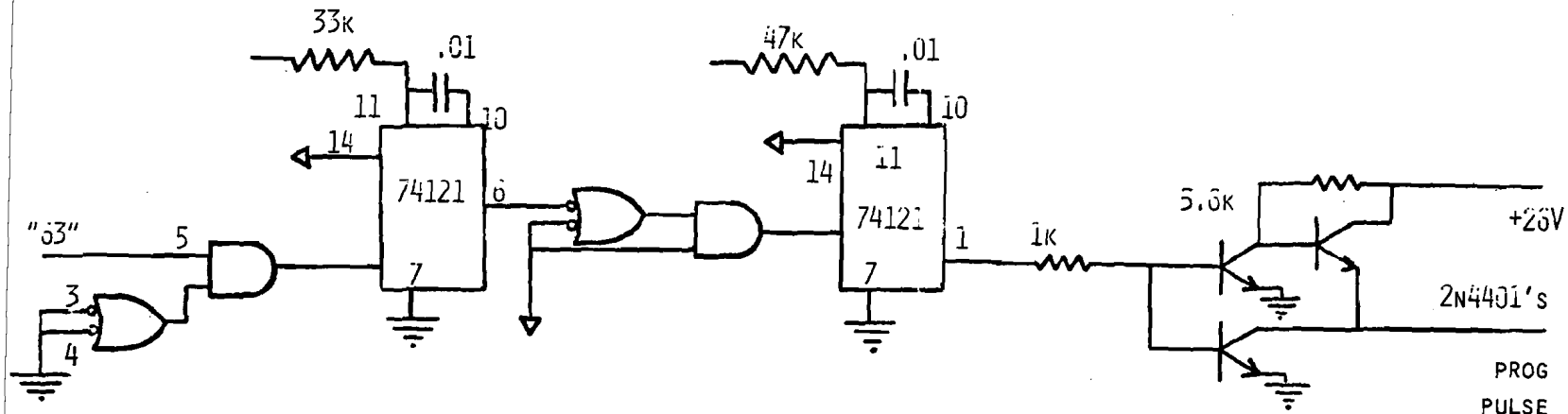
Special addresses....0009 '98' is stack addr of 9898...change to suit your system

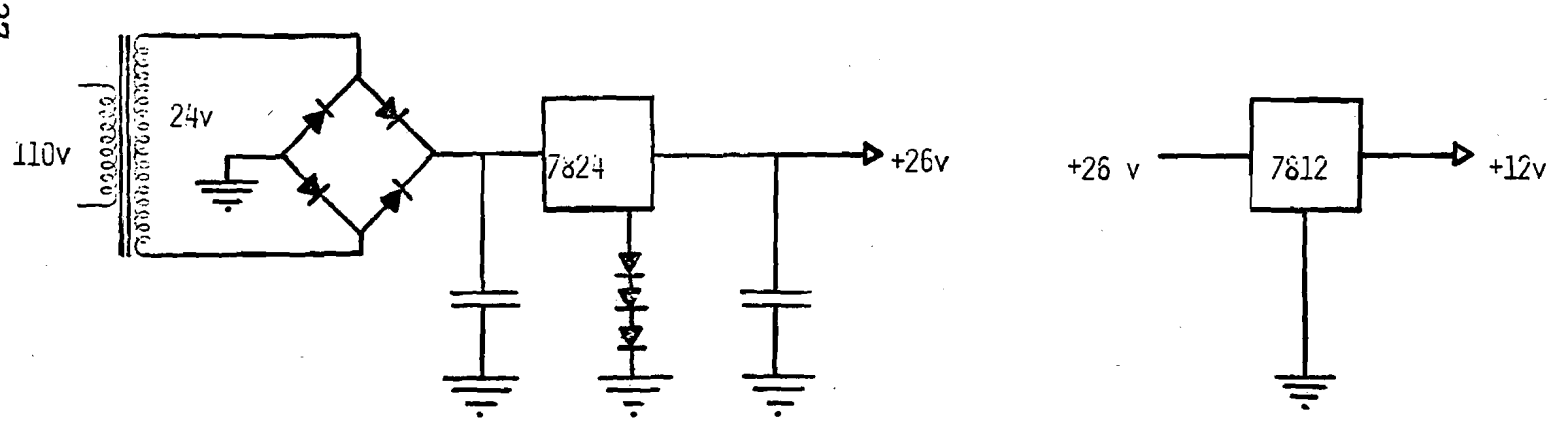
000D Hi address of where data is stored. This is page 10.

003D This starts return to Elf-Bug routine. May be 00 to end, F8 80 B0 F8 00 A0 for return to Quest monitor, or F8 F0 B0 F8 00 A0 to Netronics monitor or anything else such as Q led or warble.

Note that this code is page relocatable and also that it could be shortened by 2 bytes by including AB at the data with addr 0010 and eliminating the data at addr 0018 etc. Also, it could be shortened further by not making it page relocatable.


"63"
+5v
D0
D1
D2
D3
D4
D5
D6
D7
11 1 20
74LS373
Q0
Q1
Q2
Q3
Q4
Q5
Q6
Q7
10
"62"
+5v
11 1 20
74LS374
A0
A1
A2
A3
A4
A5
A6
A7
10
"63"
47K
+5v
5 16
4042
A8
A9
NC
NC
6
8
A7
A6
A5
A4
A3
A2
A1
A0
D0
D1
D2
2708
ZIF
PROGRAM
SOCKET
+5v
A8
A9
-5v
+12v
+12v
PROG PULSE
D7
D6
D5
D4
D3
MRQ
DEVICE 2
(FROM N-LINE
DECODER)
DEVICE 3
DEVICE 5
"62"
"63"
"65"


33K
.01
11
10
14
74121
6
"63"
5
3
4
7
47K
.01
10
14
11
74121
1
7
1K
5.6K
+26V
2N4401's
PROG
PULSE


110v
24v
7824
+26v
+26 v
7812
+12v

On Standards

In response to Tom Jones' article in IPSO FACTO #10 on standards, I would like to add my comments and ideas. I have been associated with computers for 12 years both on hardware and software sides of the business. My choice of the Elf was due to its resemblance to the larger machines (16 registers, DMA, and Interrupt driven).

First I would like to discuss a few drawbacks. As I see it, the Elf has one major drawback, that being the D-register. Due to architectural design all things must pass thru the D-reg on the way to and from registers, with few exceptions. The D-reg is also used as an accumulator which must be saved when registers are loaded or unloaded or when I/O is done. This tends to hamper register utilization. The D-reg being one byte in size, makes loading and unloading of one of the 16 registers a 4 instruction process. This creates a potential problem when developing a time-oriented subroutine where instruction path length is critical. Now, I know that most of us are not attempting to run a business or control a large TP network, but efficiency is a must. Without an efficient software system the need for additional memory comes fast and a tendency to outgrow your present system is soon realized.

Structured programming has been discussed as a standard, but as I see it, structured programming has a tendency to increase a need for additional memory by recoding common routines that could be called as subroutines. This is only practical when subroutines are shorter than the code it would take to call and return from a subroutine. Timing vs memory should be the determining factor.

When the 1861 TV chip is used, program efficiency becomes more critical, because of the number of cycles used to service it and the memory requirements for the screen display and software to control it. One page (256) display takes almost half the machine cycles available to service it, but a 4 page display would reduce machine cycles and increase memory needs.

Please do not mistake my comments as negatism towards the Elf, for that is not the case at all. They are merely considerations to keep in mind when designing the software counterpart.

Since R0, R1 and R2 have predetermined uses, R2 should be used to its fullest potential. The stack (R2) appears to be the best place to pass parms back and forth between subroutines and never needs to be reloaded, thereby eliminating at least one set of 4 instructions to load it. As long as subroutines or programs do not mickey-mouse around with R0, R1 or R2, rigid standards for register usage are not needed. Primarily because most users will have to customize new routines to fit their present system. In most cases this would just be minor register changes. Standard call and return registers would be nice, but not a necessity.

Until someone designs and markets an "Operating System" like Chip 8 that is widely utilized, rigid software standards are not needed. Since I/O configurations differ greatly and no one I/O interface (like S100) is available for the Elf, user modifications are inevitable.

A general software standards list:

R0, R1 & R2	Maintain predetermined functions, plus restrictive use of said registers.
R2	Parameter passing method whenever possible.
R3, R4	Program counter and call registers. (optional)
Idl & long branches	Restricted use when 1861 TV chip is used.
Modification of inline code	Restricted or prohibited.
Interrupt disable state	Minimum time.

(note-published programs should be complete when possible or very specific in the case of interrupt handlers as to what the program experts to be provided by or to it.)

My ideas on standards appears loose, but I feel that flexibility is a must in software as much as hardware. This, however, does not mean that there is no need for the standardization process (like tape and disk record formats and speed) when the end results are to be provided to the public. When I look at a new software routine, I evaluate its usability and adaptability to my system and determine what modifications I need to fit it in. In some cases I will redesign it, using the theme of the program and adapt it inline to my system.

Lastly, I would like to say that I really enjoy IPSO FACTO and am sorry that I have not been able to contribute in the past to such a fine newsletter. Keep up the good work and on with the 1804.

David W. Jaeger
2862 Lawrence Drive
Falls Church, Virginia
22042

ERRATA

1. Issue #11 page 26, Horse Race Program:
   M(00BA) should read 30E6
   M(01D7) should read E23CD380

2. Issue #11 page 36, VIP an ELF:
   "video chip on" should be 61 for TEC not 69.

3. Issue #11 page 37, VIP an ELF:
   M(01FA) should be 9B not 93.

Wayne Bowdish
149 East 33rd Street
Hamilton, Ontario, Canada
L8V 3T5

SOME THOUGHTS ON DEVICE INDEPENDENT I/O

As we develop more complex programs, there are several recurring operations which must be performed. One of these is I/O to our peripheral devices (keyboards, cassette types, etc.). Initially the problem was simply to input, or output, a character to the device (well, after the device was acquired it was a problem). There has been much discussion on this in IPSO FACTO. Over the last year and a half, there have been articles on proposed cassette I/O methods, BAUDOT terminal interfaces and so on. Many people have placed the basic I/O routines in ROM based monitors (refer to Tom Crawford's monitor in issue #10).

I think that it is now time to consider implementing a more generalized and powerful I/O system. The system would be installed in another 1K ROM (I hope) and would serve to simplify the I/O programming task. This should then allow the programmer to spend more time thinking about the application and less time worrying about the details of I/O.

A major consideration in the design of an I/O system is to isolate the user from specific device characteristics as much as possible. The programmer (actually the program) should not have to know which device will perform the actual I/O until the program is executed. For example, when a program is being developed (in BASIC, machine language, etc.) the programmer may want the output to go to a relatively fast CRT. Later, when the program has been debugged, the output can be redirected to a hard copy device, say a BAUDOT teletype. Ideally, this change of output devices should not require any programming changes.

Systems which isolate the applications programs from the physical device characteristics are said to implement device independent I/O. This I/O subsystem should have the following characteristics:

1. Specific device characteristics need not be considered by the programmer.
   i.e. Whether a device uses BAUDOT or ASC II codes, or whether a hardware UART is used or this function is performed by the software should not have to be considered when the program is being written.

2. Device functions which do not apply to a given device should be ignored by that device.
   i.e. If a rewind command is issued to a printer, it should be ignored.

3. Physical devices should be referred to by meaningful mnemonic names which are independent of hardware/software configurations.

4. There must be a method of referring to multiple units of a given device type.

5. Error and status information should be returned in a consistent manner. The error status should be easily tested.

6. The system should allow buffered and que'd I/O (definitions of these terms will be given later in this article).

7. The system should support block oriented and character oriented devices.

8. The system should have a simple, yet flexible interface.

9. The system should allow for future additions such as directoried devices.

10. Finally, the system should require a minimum of the computer resources (memory and execution time).

As this I/O system (let's call it a subsystem) is further developed, tradeoffs will have to be made. I am presenting my ideas at this time in the hope that readers will comment on the system. There are probably other desirable characteristics, and some will be more important than others. The tradeoffs in the design phase will have a major effect on the final performance of the system.

Proposed System

The I/O subsystem will consist of a collection of subroutines which may be called by a program. These routines will initialize various tables, allow assignments between physical devices and logical I/O channels, and route data between programs and the device drivers. These routines will be callable by applications programs and some will be called by monitor level commands.

Figure 1 diagramatically represents the I/O subsystem. Referring to the figure as well as the glossary of terms at the end of this article should help to clarify the terms used in this article.

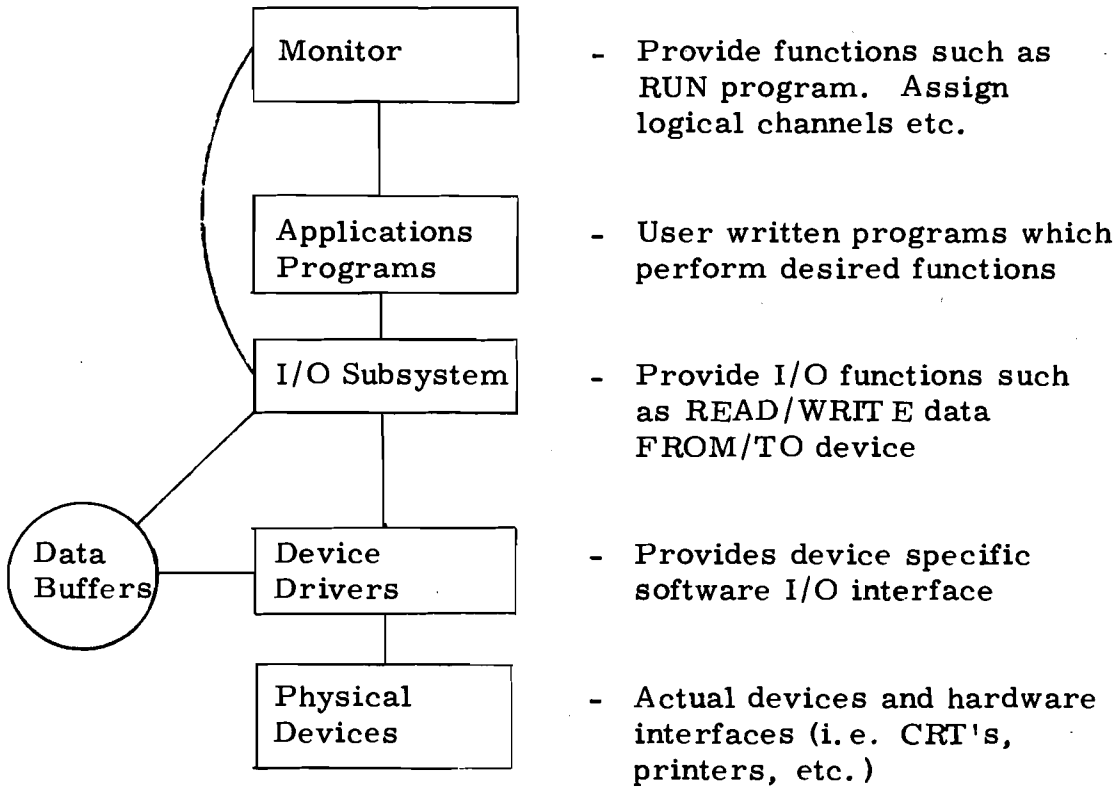


FIGURE I

I/O SUBSYSTEM OVERVIEW

Referring to Figure 1, a typical sequence of events to write data to an output device might be as follows:

1. Power up your micro and begin in "monitor mode". At this point, you might LOAD a program into memory from some device, say a cassette tape.

2. Using another monitor command, a logical channel number is assigned to a physical device.

3. Using yet another monitor command, program execution is started.

4. The program performs I/O by calls to subroutines in the I/O subsystem. Let's call them READ and WRITE.

5. The READ and WRITE calls of step 4 pass data from/to the device via the I/O subsystem and device drivers.

6. The device drivers transfer data to/from the actual physical devices via the hardware interface.

Note that step 2 is the only place where an actual physical device was referenced (step 2 could be moved into the program).

At this point I will discuss command notation. In the article commands will be specified by a meaningful name in capitals, possibly followed by some arguments. Each command would have various forms depending on how it was invoked. An appendix at the end of this article lists all of the proposed commands. As an example, refer back to step 2 of the previous sequence of events. The ASSIGN command establishes a link between a logical I/O channel and a physical device. The command notation for the ASSIGN command is:

ASSIGN D:C

where "D" is a physical device specifier and "C" is a logical channel specifier. The monitor command format might be:

AS D, C

while in assembly language, the calling sequence might be:

```
SEP  R4      ; subroutine called by SCRT
2 byte address of assign subroutine
1 byte device specifier
1 byte channel specifier
```

The assign subroutine could even be called from tiny Basic via the USR function as:

```
x = USR (A, D, C)
```

where the variable A contains the address of the ASSIGN routine.

Qued I/O

Qued I/O will be described by describing typical sequences for input and output.

On output:

- the program fills a buffer with data.
- the buffer is submitted to a device, via a channel for output.
- this buffer is placed in the device output que and will be processed when the device driver is free.
- the user may now continue processing while the device driver performs the actual output.
- when the output is completed, the program may be signaled (i.e. informed of the completion of the operation).

On input:

- the program requests input from a given device, via a logical channel, to a specified data buffer.
- the program may continue processing and even issue more input requests or wait until the transfer is complete.
- the device driver performs the data transfer and signals the program when the transfer has been completed.

In the qued I/O case, there may be several I/O requests pending at any given time. This form of I/O is generally used when blocks of data are to be transferred (i.e. cassette tape).

Buffered I/O

Buffered I/O will also be described by giving typical sequences of events.

On output:

- the applications program outputs data bytes to a buffer assigned to a channel.
- asynchronously, the device drive transfers data bytes from the buffer to the physical device.
- if the program outputs data at an average rate that is less than the device output rate, the program will not have to wait. If the buffer fills up, the program will have to wait until there is room in the buffer.

On input:

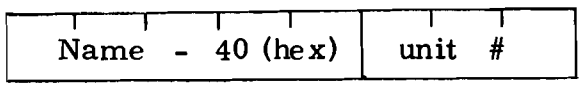
- as data bytes from a device become available, they are placed in a buffer assigned to that device (via a channel).
- the program removes data from the buffer as required. If the buffer is empty, the program may either wait for more data or proceed with some other task until more data becomes available.
- if the program is capable of removing data from the buffer at an average rate which is greater than the input rate, then no data will be lost.

Buffered I/O would normally be used for devices such as keyboards and printers, which are byte (character) oriented and which may transmit small amounts of data at irregular intervals not under control of the computer.

Device And File Specifications

Physical devices are specified by a one character (alphabetic) device name and a one digit (octal) unit number. If required, the device specifier is delimited by a full colon ":". For example, referring to the following list of proposed names, cassette type units will have the name "C". If a system has two cassette tape units, they would be referred to as C0: and C1:. To decrease the typing load a little, if the unit number is 0 then the unit number may be omitted (i.e. C0: may be specified as C:).

Internally these device specifiers could be stored in one byte as follows:



i.e. device C5 would be stored as 00011101 (in binary) since C = 63 hex.

The proposed device names are:

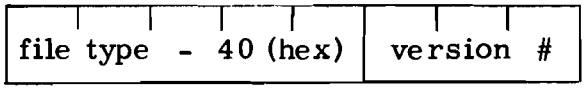
B	BAUDOT terminal
C	Cassette tape
D	Hard disk (wouldn't it be nice)
E	EBCDIC terminal
F	Floppy disk
K	Keyboard (input only device)
M	Mag tape (other than cassette)
N	Paper tape punch
P	Printer (output only)
R	Paper tape reader
S	Screen (CRT output device)
T	Terminal (input and output)
Z	Null device

Someday our systems may support files on directoried devices. File names would consist of three parts - a name, a file type extension, and a version number. The file name consists of from 1 to 6 alpha-numeric characters. Names are usually chosen to be meaningful to the user. The file type extension is a single alpha-numeric character which describes the type of data in the file. The version number is a single octal digit. For example, referring to the following list of proposed file extensions, a file LIFE.A3 is the fourth version (the first was A0) of a program written in 1802 assembly language (A file type).

Device And File Specifications

If the file is on a device it may be completely referenced by adding the device specifier (i.e. C1:LIFE.A3 refers to a file on a cassette currently on cassette tape unit 1).

File type extension and version numbers will be stored in one byte in a manner similar to device specifiers, i.e.



The proposed file types are:

A	Assembly language
B	BASIC
C	C sorce language
D	Data files
F	FORTRAN (maybe one day)
M	Memory dumps
P	PASCAL
S	Simple cassette format
T	Text (letters, books, etc.)

Qued I/O Functions

ADDQUE - address, number

Insert "number" additional que elements in the free element list. Que elements are allocated starting at "address". Each element is 6 bytes in length. Therefore, 6 * number bytes must be reserved, starting at "address".

REMQUE - address, number

Remove "number" que elements from the free list pool (previously set up by ADDQUE). These elements must not be currently in use.

Qued I/O Functions

CHNCHK - channel (qued or buffered I/O)

Return information on "channel".
On return DF set indicates channel not assigned. If DF is reset the channel is assigned and the bits in the D - register have the following meaning:

0	=	1	channel assigned
1	=	0	buffered I/O
	=	1	qued I/O
2	=	0	channel not busy
	=	1	if qued I/O then channel busy, if buffered I/O then data in buffer
3	=	1	end of file detected on channel
4	=		spare
5	=		spare
6			if bit 7 set = 0 soft error detected = 1 hard error detected
7	=	0	no errors detected on channel
	=	1	error detected on channel

WAIT - channel

Suspend the program until all I/O on "channel" is complete.
On return - DF reset indicates all ok.
- DF set indicates an error occurred on the channel (D-reg. contains error flag).

QREAD - channel, buffer, size

Request qued input of "size" bytes via "channel" to "buffer".
On return - DF reset indicates all ok.
- DF set indicates read error (D-reg. contains error flags).

QWRITE - channel, buffer, size

Request qued output of "size" bytes via "channel" from memory beginning at "buffer".
On return - DF reset indicates all ok.
- DF set indicates error detected (D-reg. contains error flags).

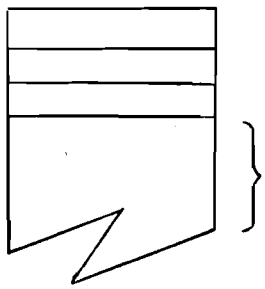
Buffered I/O Routines

BUFSET - channel, address, size, R/W

Reserve a buffer for I/O an initialize header.

channel	-	channel number to be associated with buffer.
address	-	starting address of buffer space.
size	-	number of data bytes in buffer (1 to 256).
R/W	-	read or write flag (optional for devices which are unidirectional).

Buffers have a 3 byte header. Therefore "size" + 3 bytes must be reserved for the buffer. The buffer appears as follows:



buffer size -1 (0 to 255)
relative loc of first data byte
number of data bytes in buffer

1 to 256 data bytes

Buffers are maintained in a circular fashion. Characters are added to the "end" of the buffer and removed from the "beginning".

GETCHR - channel

Input 1 byte from specified channel.
On return:

- if character available - DF reset
  - D-reg., contains byte
- if no character available - DF set
  - D-reg., contains zero
- if error detected - DF set
  - D-reg., contains error flags

Buffered I/O Routines

PUTCHR - channel

Output 1 byte to specified channel.
On return:

- if room in buffer - DF reset
- if buffer full - DF set
  - D-reg. contains zero
- if error detected - DF set
  - D-reg. contains error flags.

I/O Subsystem Tables

The I/O subsystem consists of several tables and routines.

Physical Device Tables

$DVNAM - This table contains a 1 byte entry for each physical device in the system. The byte contains the single character device names. This may be in ROM. Defined device names are:

B	baudot terminal
C	cassette tape
D	hard disk
E	EBCDIC terminal
F	floppy disk
K	keyboard (input only)
M	mag. tape (other than cassette)
N	paper tape punch
P	printer (output only)
R	paper tape reader
S	screen (output only)
T	terminal (input and output)

I/O Subsystem Tables

$DVHDL - This table contains a 2 byte entry for each corresponding physical device in $DVNAM. The double byte contains the starting address of the device handler for the device.

The device handlers must have the following capabilities:

- input/output 1 byte to device
- detect, and return, device error status
- exert any device control that is required (i.e. rewind, start/stop, etc.)

Handlers for block structural devices may read/write blocks of data if required.

Note $DVHDL may be in ROM if all device handler addresses are known at assembly time. If device handlers may be installed in RAM when required, then this table must be in RAM so that addresses may be added. In this case, a ROM copy would be transferred to RAM at monitor initialize time.

$DVFLG - This table contains a 1 byte entry for each corresponding device in $DVNAM. This table may be in ROM. The flag bits are as follows:

0	=	1	input device
1	=	1	output device
2	=	1	block structured (i.e. blocked cassette, disk)
3	=	1	non ASCII device
4	=	1	directoried device
5			
6			
7			

Channel Tables (Logical Device Tables)

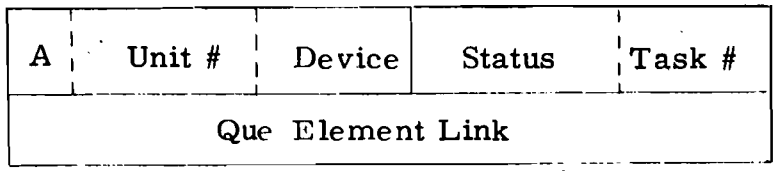
$CHANL - This table contains an entry for each logical channel in the system. This table may be set up by:

- monitor commands
- program requests

Therefore it must be RAM resident. A ROM copy of default assignments could be transferred to RAM at monitor initialize time.

Each entry contains the following:

Byte	Bits	Description
0	0-3	Physical device pointer (index to $DVxxx tables)
	4-6	Unit number
	7	Channel assigned flag
1	0-4	Task number associated with this channel (future)
	5-7	Channel status flags 5 - buffered/qued I/O 6 - channel busy 7 - error detected if 7 set (i.e. error) then 6 = 0 soft error = 1 hard error
2, 3	all	que element link

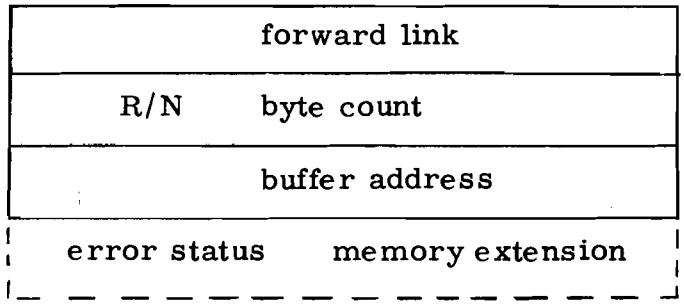


Channel Tables (Logical Device Tables)

FRELST - Two bytes which contain address of first element in free list. If there are no free que elements, this pointer contains #0000.

FREELM - Linked list of all free que elements. Elements are removed from this list and added to device ques as required.

QUE ELEMENTS



forward link = 0 last element in que
≠ 0 pointer to next element in que

byte count - number of bytes to transfer

R/W = 0 write
= 1 read

buffer address - starting address of data buffer

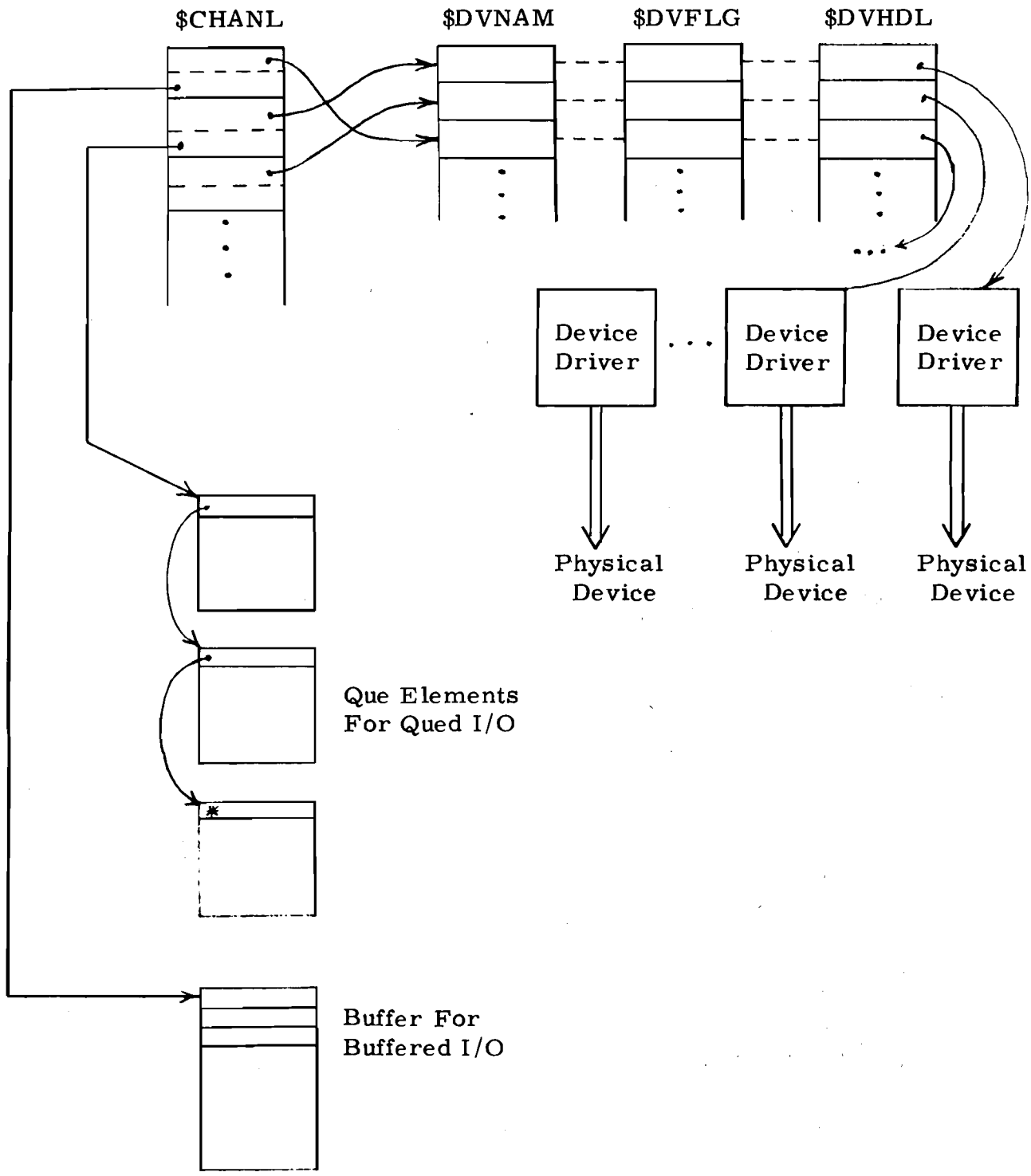


FIGURE 2

COMPONENT RELATIONSHIPS

APPENDIX A

COMMANDS

Primary Commands

ASSIGN D:C

Assign logical channel C to physical device D. The channel number is a hex digit in the range O to F. The device specifier is of the form Dn: where D is a device name (see following table) and n is a unit number in the range 0 to 7. The device name may be one of:

B - BAUDOT terminal
C - cassette tape
D - hard disk
E - EBCDIC terminal
F - floppy disk
K - keyboard
M - magnetic tape (not cassette)
N - paper tape punch
P - printer
R - paper tape reader
S - screen (CRT)
T - ascii terminal

INIT	C	Initialize the specified channel. i.e. clear buffers, screen, etc., clear error flags, etc.
INSTAL	D:addr.	Install device handler for device D currently in memory starting at "addr".
QREAD	C	Qued read from channel C.
QWRITE	C	Qued write to channel C.
GETCHR	C	Buffered character read from channel C.

Primary Commands

PUTCHR	C	Buffered character write to channel C.
WAIT	C	Wait for completion of I/O on channel C.
CHNCHK	C	Return status of specified channel.
BUFSET	C, addr, size, R/W	Reserve an I/O buffer for channel "C" of "size" bytes beginning at "addr" for I/O (type specified by R/W).
DEASSIGN	C	Remove assignments for channel C.

Future Extensions

REWIND	C	Rewind device on channel C (tape only).
SKIP	C	Skip to next input record on channel C.
BACKSPACE REREAD FWIND	C	Rewind device on channel C to beginning of file.
LOAD	D addr.	Load a device handler from device "D" into memory at "addr.".
UNLOAD	D	Unload device handler, i.e. remove from tables to memory may be reused.
LOGICAL	D:N	Assign a logical name to device D.

Future File Extensions

CLOSE	C	Close file on channel.
LOOKUP	C, D: filespec	Find "filespec".
CREATE	C, D: filespec	Create new file entry.
DELETE	C, D: filespec	Delete file.
RENAME	C, D: file 1, file 2,	Rename file 2 to file 1.
SEEK		Position head or media.

GLOSSARY

Application Program	A program which performs some desired function. i.e. the game of life or TIC-TAC-TOE or a model train controller. See utility program, monitor and operating system.
Block Structured Device	Refers to devices where the smallest data transfer consists of more than one byte. i.e. disk drives and some forms of tape drives.
Buffer	An area of memory where data is temporarily stored (buffered).

Channel	A means of forming an I/O link between a program and some device.
Device Driver	A software routine which interfaces directly to a physical device (via the computer - device hardware interface).
Monitor	A collection of routines which control the system resources and provide access to these resources (see operating system). Monitors typically allow one to load programs into memory, run programs, dump memory to a device, modify memory, etc.
Monitor Mode	The mode in which the terminal is communicating with the monitor (i.e. issuing commands to the monitor). When an application program is controlling the terminal, it is said to be in "User Mode".
Operating System	Often used interchangeably with the term Monitor.
Physical Device	A real device connected to a computer (i.e. a peripheral) such as a teletype or cassette tape unit.
Utility Program	Programs which provide commonly used functions such as text editors and language translaters (assemblers, compilers and interpreters). For example, Tiny Basic is a utility program used to create and execute application programs such as TIC-TAC-TOE.

M.E. Franklin
24 Duby Road
Acton, Ontario

1001 Options For The 1802

Well not quite, seven actually - Netronic's, Quest, Tektron, RCA, Infinite, Southern Scientific and homebrew. Thats the number of manufacturers of 1802 based micro computers at last count. Each apparently manages to use the same 40 pin chip and 91 machine language commands in a unique manner. The purpose of this article is to unravel some of the "uniqueness" of each manufacturers system, and cross reference it to the appropriate link in the other systems.

My own Netronics manuals, and those supplied by Tektron, RCA and Quest form the basis for the information pertaining to these systems, while the information on Infinite and Benchmark comes from advertising literature. At the moment, Southern Scientific is a big mystery. Those brave soles who built their own will have to add their own data. I have not attempted to qualitatively rate any of the manufacturer's equipment, nor indicate what is available. Their advertisements suffice for that. The benefits to be realized from the following information are threefold: 1) the ability to adopt another manufacturer's products to your own system by correct bus linkages (Table 1) and software linkages (Table 2); 2) the ability to choose which monitor-operating system (Table 4) and which Tiny Basic (Table 3) will meet your programming needs; and 3) the realization that you are no longer tied to one manufacturer's product or software, but, with appropriate hardware and software linkages, can adopt other products to your system.

The later is a major benefit to realize, particularily with the supply problems and poor "public relations" experienced by some ACE members with certain manufacturers, and also, no one need fear being abandoned by a manufacturer if the product line ceases to be marketed.

Table 1 is broken into 2 parts - standard functions and non-standard functions. Each function represents the use of one or more bus pins. Where pin usage changes with the bus, it has been noted. Non-standard functions are used by manufacturers to utilize their own hardware, and probably are necessary modifications for anyone adopting their equipment.

Table 2 cross references hardware usage for major functions of the systems. Changes in software will have to be made following these assignments to run one manufacturers software on another system (i.e. changing video on-off commands, use of different flag lines, etc.).

Table 3 lists the commands available from the 5 versions of Tiny available at present. At this time, no data is available on the RCA-VIP Tiny commands. All the Tiny's, except for the Palo Alto version, are basically the same and apparently all written by Tom Pittman. The Palo Alto version, see reference IPSO FACTO 9, p.19, was developed by R. Edwards of the Oak Ridge Computer Association and offers "For/Next" and an "Array", and uses a faster call/return routine than the RCA Standard.

Table 4 lists information available about the monitors and other languages available from specific manufacturers. Each should work on any system if appropriate linkages are made in either hardware or software. While all monitors offer common functions (i.e. cassette I/O, memory read/write) some offer other functions which may or may not be of use to you.

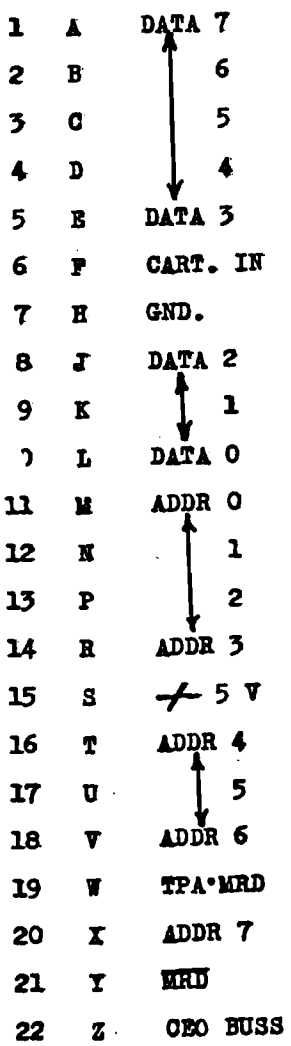
As you can see, with some work, you have several options for each hardware or language choice you may wish to make.

Table 1

1802 SYSTEMS - BUS ASSIGNMENTS (Main System Bus Only)

Function	Quest 44 Pin	Quest 50 Pin	RCA 44 Pin	TEC 44 Pin	Netronics 86 Pin
+8V - +12V	2, 27	2, 3	---	21 (bus 4, 5)	9, 11
GND	1, 26	1, A	Z, 22	22, 44	5, 7, 80, 82
+5V Regulated	-	-	Y, 21	1, 23	1, 3, 2, 4
Address 0	46	V	C	34	64
1	21	18	D	38	60
2	45	U	E	37	56
3	20	17	F	36	52
4	44	7	H	35	48
5	19	16	J	34	44
6	43	5	K	33	40
7	18	15	L	32	36
8	42	R	-*	-*	-*
9	17	14	- Systems use TPA signal		-
10	41	P	- to switch address lines to high		
11	16	13	- order addresses		-
12	40	N	-	-	-
13	15	12	-	-	-
14	39	M	-	-	-
15	14	11	-	-	-
Data 0	50	Z	M	16	62
1	25	22	W	15	58
2	49	Y	P	14	54
3	24	21	R	13	50
4	48	X	S	12	46
5	23	20	T	11	42
6	47	W	U	10	38
7	22	19	V	9	34
N0	4	4	6	42	76
N1	29	D	7	41	72
N2	5	5	8	40	68
EF1	-	-	5	19	66
EF2	28	C	-	18	70
EF3	3	3	3	17	74
EF4	-	-	-	20	78
SC0	31	F	12	7	26
SC1	6	6	14	6	22
Q	30	E	16	5	18
$\overline{\text{DMA IN}}$	8	8	17	26	12
$\overline{\text{DMA OUT}}$	33	J	15	27	16
$\overline{\text{Interupt}}$	34	K	13	28	20
Clock	32	H	-	-	8
Clock	-	-	1	25	6
TPA	35	L	B	30	28
$\overline{\text{TPB}}$	10	10	11	31	32
$\overline{\text{MRD}}$	7	7	W	8	30
$\overline{\text{MWR}}$	9	9	A	29	24

RCA STUDIO II Pin-Out



1802 SYSTEMS - BUS ASSIGNMENTS (Cont'd)

Non Standard Functions	Quest 44 Pin	Quest 50 Pin	RCA 44 Pin	TEC 44 Pin	Netronics 86 Pin
Load	-	-	-	-	10
Run	-	-	18	-	14
M	11	-	-	-	-
G	12	-	-	-	-
Chip Select	13	-	X	Z (bus 4, 5)	69
P/NP	36	-	-	-	-
RW	37	-	-	-	-
MP	38	-	-	-	-
Cassette In	-	-	-	-	61 (bus 5)
Cassette Out	-	-	-	-	59 (bus 5)
Video Off	-	-	-	-	65 (bus 5)
Video On	-	-	-	-	83 (bus 5)
Video Spot	-	-	9	-	-
Video Sync	-	-	10	-	81 (bus 5)
Video Status	-	-	X	-	-
INDIS (Ram disable)		-	19	-	-
CDEF (Ram page select)		-	20	-	-
CE1	-	-	-	21 (bus 1, 2)	-
CE2	-	-	-	43 (bus 1, 2)	-
-5V	-	-	-	43 (bus 3, 4, 5)	-
MB1E	-	-	-	2 (bus 2, 3)	-
SR	-	-	-	24 (bus 1)	-

Table 2

1802 SYSTEMS - HARDWARE ASSIGNMENTS

Function	Quest	RCA	TEC	Netronics
Video Display On	OUT1	IN1	OUT1	IN1
Video Display Off	OUT2	OUT1	OUT1	OUT1
Video Display Status	EF1	EF1	EF1	EF1
Hex Key Pad	INP4	OUT2	IN4	IN4
Hex Display	OUT4	-	OUT4	OUT4
Input Switch Status	EF4	EF4	EF4	EF4
Parallel Port In	INP5	-	IN1	IN7
Parallel In Data Ready	INCP 10 (EF2)	-	EF3	EF3
Parallel In Data Received	INCP 15	-	-	EF3
Parallel Port Out	OUT3	-	OUT4	OUT7
Parallel Out Data Ready	OUTCP 10	-	-	-
Parallel Out Data Received	OUTCP 15 (EF3)	-	-	-
Cassette Output	Q	Q	Q	Q
Cassette Input	EF3	EF2	EF2	EF2
RS 232 Output	Q	-	Q	Q
RS 232 Input	EF2	-	EF2	EF4
20 MA Output	Q	-	Q	Q
20 MA Input	EF2	-	EF2	EF4

Table 3 Tiny Basic Compared

Direct Commands	Netronics	Infinite	Super Quest & Monitor		Palo Alto	RCA
Print	X (4k)	X (2k)	X(2k)	X(3k)	X(2k)	X(4k)
Input	X	X	X	X	X	X
Let(VAR)(A-Z, 16 bit)	X	X	X	X	X	X
Goto	X	X	X	X	X	X
Gosub	X	X	X	X	X	X
Return	X	X	X	X	X	X
If Then/Let (Equality)	X	X	X	X	X	X
Rem	X	X	X	X	X	X
Clear	X	X	X	X	(New)	(new)
Run	X	X	X	X	X	X
List	X	X	X	X	X	X
End	X	X	X	X	(Stop)	X
Plot	X			X		
Load	X			X		
Save	X			X		
Poke	X			X	X	
Out	X			X		
For/Next					X	
And				X		
ABS				X	X	X
Bye				X		
Array (16 bit variables)					X	
Expressions						
Add	X	X	X	X	X	X
Subt	X	X	X	X	X	X
Div (Truncated Modulo)	X	X	X	X	X	X
Mult	X	X	X	X	X	X
#0-9, 16 bit -32767 to + 32767	X	X	X	X	X	X
Functions						
Peek	X			X	X	
INP	X			X		
USR	X		X	X		
RND	X		X	X	X	X
Clear Screen				X		X
Or				X		
Flag				X		
Stop					X	
New					X	X
Address						plus 12 colour/ sound controls
Program	0100-0F40	0100-08FF	8400-8BFF		0100-08FF	?
I/O	0000-00FF	?	9800-98FF		NO	uses OP code
User Space Starts	0F5A	0900	0003		0900	?

Table 4 Other Languages, Monitors, Debug, Etc.

Netronics - Monitor - ¼ K-address F000

- 6 Commands - 00 Execute Program
  - 01 Memory Read
  - 02 Memory Write
  - 03 Cassette Write
  - 04 Cassette Read
  - 05 Search

- Debug - ¾ K + Stack-address flexible

- 3 Commands - Works with monitor
  - 00 Execute
  - 01 Memory Read and Display
  - 02 Memory Write and Display
- Includes subroutines for Hex Character Generator, Video Display Refresh, Register Restore.
- Displays 24 consecutive addresses and contents.

RCA-VIP - Operating System - ½ K - address 8000

- 4 Commands - 0 Memory Write
  - A Memory Read
  - F Cassette Write
  - B Cassette Read
- Includes subroutines for Hex Character Generator, Video Display Refresh, Hex Key Pad Decode.
- Displays 1 address and content.

- CHIP 8 - ½ K - address 0000

- 31 Commands, used with OP CODE

- Contains macro commands for graphic manipulation with the 1861 video chip.

- CHIP 8C

- As above plus 5 commands for colour graphics control.

Quest - Monitor - 1 K - address 8000

Version 1

- 10 Commands - 00 Execute
  - 01 Memory Read
  - 02 Memory Write
  - 03 Memory Block Move
  - 04 Cassette Read
  - 05 Cassette Write
  - 06 Tape To Memory Compare
  - 07 Auto Playback Data
  - 08 Auto Tape to Memory Compare
  - 09 Video Graphic

- Inclues subroutines for - Register Save, Access Monitor, Use Monitor I/O, Call Monitor subroutines and Video Interupt Service.

Version 2

- As above
- Adds Tiny Basic Keyboard and S-100 bus driver.

Version 3

- Required for Extended Tiny Basic.

Tectronics- Monitor - 1 K - address 3800

- 10 Commands - EE Memory Read
  - CC Memory Write
  - BB Back Space 1 Location
  - CA Change All Data
  - OO Return to Monitor
  - NN Display Next Address and Data
  - DO Execute
  - CO Cassette Read
  - C1 Cassette Write
  - C5 Checksum
- Includes subroutines for Hex Character Generator, Video Display Refresh.
- Displays 10 consecutive addresses and contents.

Infinite - Keybug - 1/8 K - address flexible

- Literature Lists - Memory Write
  - Memory Read
  - Change All Memory
  - Display A Location
  - Execute
- Includes general purpose subroutines and an input buffer to access other programs.
- Literature does not specifically refer to cassette I/O, but does refer to extended programs, video, keyboards, etc. which implies mass data storage via cassette.

Benchmark - Monitor - 1 K + Stack - address flexible

- Literature Lists - Cassette Read
  - Cassette Write > Kansas City Standard
  - Input-Output
  - UART
  - Tone Generator
  - Print Memory Block
  - CR
  - LF
  - MR(N)
  - R(N)
  - ASC II - Hex Conversion
  - Memory Read
  - Memory Write

M.E. Franklin
24 Duby Road
Acton, Ontario
L7J 2P1

To VIP an ELF, Part II

Part I, in IPSO FACTO #11, gave the program changes to correct the RCA-VIP operating system and Chip 8 language to your particular 1802 micro.

The article contained one typo - in the table of functions, the TEC 1802 uses a 61, not a 69, to turn the video chip on. The program listing refering to M(0C70) is correct however, M(01FA) should be 9B NOT 93.

Now that you have Chip 8 running, the following may be of interest.

1. RCA has released a replacement manual for the VIP Instruction Manual - #311. at $5.00. It is apparently not substantially different then the 300 version.

2. RCA has released "A COSMAC VIP Users Guide", manual VIP-320 at $5.00 - detailing use of the 31 Chip 8 instructions. No games or program listings.

3. Quest has developed the "Moews Interpreter" offering an extended Operating system - Chip 8 for the Super Elf at $6.00. It is relocatable, offers multiplication, division and skip instructions. Requires 4K memory.

4. RCA has developed Chip 8C an expanded Chip 8 with 5 additional instructions to develop colour control for use with its new VP-590 colour display. Board includes CHIP-8C listing and conversion program - $69.00.

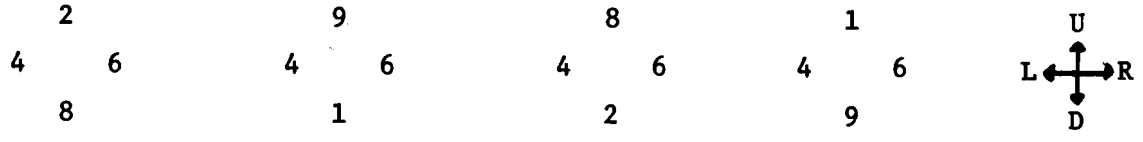
5. If you are stumped about the objective of game 18, Shooting Stars, refer to Best of Byte, Vol. 1, Nico's article on page 314 on the programs operation.

6. Finally, this last item is one of convenience for playing games 1, 2, 6, 12 and 16.

   Chip 8 uses 4 directional commands to control the graphic object - 2 up, 8 down, 4 left and 6 right. However, no other manufacturer's machine has the key pad configured in the same order, as follows:

RCA-VIP	Netronics	TEC	Quest (and Infinite)
1 2 3 C	C D E F	- - - A	0 1 2 3
4 5 6 D	8 9 A B	7 8 9 B	4 5 6 7
7 8 9 E	4 5 6 7	4 5 6 C	8 9 A B
A 0 B F	0 1 2 3	1 2 3 D	C D E F
		- 0 F E	

   A more logical control configuration would be as follows:



The following modifications convert the programs for my Netronic's Elf II key pad.

Game 1	Change	m(0232)	from	4002	to	4009
		(023E)		4008		4001
2		m(024A)		0204		0904
		(024C)		0608		0601
6		m(0226)		6003		6009
		(022C)		6006		6005
		(0232)		6009		6001
12		m(0204)		6A08		6A01
		(020A)		6702		6709
16		m(02C4)		6502		6509
		(02CA)		6508		6501

Unfortunately, I have not yet found the secret to decoding the key pad assignment for the matrix games, #11, 15 and 18. Perhaps someone else would help here.

Anyway, for your particular system, choose the desired key position value and substitute it for the byte underlined in the above listing. It makes the game more enjoyable when you don't have to hunt for the key!

For your information, Home Computer Centre, 6101 Yonge Street, Toronto, Ont., 225-1165, is now distributing RCA VIP products and the above manuals are now in stock.

LETTERS OF CONTACT

I would like a modification to Malcolm Coyne's EPROM board (IPSO FACTO #11, page 10) so that it can be used with an ELF II. Jim Anderson, 29849-24th Pl. S., Federal Way, Wash., 98003.

Do you know if there is any additional information available on the 1804 and other new chips besides the MPG-180B manual? If so, please let me know or put a note in the next IPSO FACTO. Have you had any thoughts towards forming an interface with RCA to pass on the Club's ideas for future enhancements to RCA's microprocessor line? David W. Jaeger, 2862 Lawrence Dr., Falls Church, VA., 22042.

LETTERS OF CONTACT (CONT'D)

It would be appreciated if any of the 1802 users who have built a system on S-100 boards or 44 pin vector boards could send a pin assignment list. (I have the vector boards but would rather wait to see if the S-100 has been used because memory boards and extra I/O would be cheaper.) I'd also like to know what tape recording standards have evolved. I ask this because I had hopes to pattern my starter system after the ELF II but with a few personal modifications (like a calculator chip for complex math functions). When the system is built, I'd like to be able to exchange PGM tapes with other 1802 users. Thank you for your assistance.
Roy Weidig, 731 Euclid Ct., Apt. K-2, Middletown, Ohio, USA, 45042.

LETTERS TO THE EDITOR

Dear Bernie,

The idea of a disk operating system for an 1802 system has been mentioned previously in IPSO FACTO (by me, among others). I would like to find out if anyone is interested in having one.

In order to get a disk drive interfaced to an 1802, I would propose getting a drive and SWTPC interface from Percom and making a (hopefully) simple interface between the Percom controller and the 1802 bus. (Some 1802 systems have one or two S-100 slots, but I am a bit skeptical as to whether an S-100 disk controller would work when plugged in there.) The first software I would write would be enough to save memory onto disk and read it back, with the user remembering (via pencil and paper) where on the disk (starting track and sector, number of sectors) each program is stored. Eventually, I would write a "real", file-oriented DOS.

I chose Percom in the above because it seems to be the least expensive way to get a disk drive and controller. (I would like to avoid designing my own controller, though that would probably cost less.) I believe the Percom disk unit is a mini-floppy, rather than an 8-inch model, though this is not clear from their ads. The advertised price is $599.

I may do the above for my ELF II eventually. However, if people are interested in a disk system for the 1802, I may be able to raise the priority of this project. If you are interested, or have suggestions as to hardware (or anything else), please let me know. Sincerely, Jim Howell, 5472 Playa Del Rey, San Jose, CA., 95123.

Just a quick note on your proposed Programming standards, with respect to SCRT:

if you use RF.1 to store temp acc, then RF.0 is more easily set up with status (eg. change from 00 to 01 by INC)

if you use R8 & RA instead of RA & RC then TINY BASIC can use this interface, unmodified.

Tom Pittman, ITTY BITTY COMPUTERS, P.O. Box 23189, San Jose, CA 95153.

LETTERS TO THE EDITOR (CONT'D)

In "an Automatic Program Counter Stepper" IPSO FACTO #9, the switch S2 may be removed from between pin 3 and C3 to between pins 2 and 6. Doing so will reduce trouble caused by using less expensive switches. Sincerely, Chris Airhart.

THE ASSOCIATION OF COMPUTER EXPERIMENTERS MINUTES OF CLUB MEETING 79-4 HELD AT STELCO WILCOX ST. AUDITORIUM 8 MAY, 1979, 8:00 P.M.

79-4-1 The meeting was preceded by a tutorial.

79-4-2 Motion to adopt Minutes 79-3 as included in Newsletter Issue #11.
Proposed - Tom Crawford
Seconded - Mike Franklin
Carried

79-4-3 Election of new executive--since no nominations were received a number of "volunteers" were solicited from the members present. There were no contested positions, therefore all positions were filled by acclamation. The new executive, as approved at the meeting, are:

President - Ed Fleet
Past President - Ken Bevis
Secretary/Treasurer - not filled
Newsletter Editors - Bob Silcox
Earle Laycock
Vic Kushner
Mike Franklin
Program Co-ordinator - Fred Pluthero
Training Co-ordinators - Rod Dore
Jeff Davis
Hardware Co-ordinator - Fred Feaver
Membership Co-ordinator - Blair Gerrish
Publishing Committee - Dennis Mildon
John Hanson

79-4-4 The meeting concluded at 10:30 P.M.
24 people attended the meeting.

LOGO IRON-ON TRANSFER & INSTRUCTIONS

NOTE: Use on 100% COTTON for best results or 50% Cotton, 50% Polyester materials only.

1. Set iron to cotton.
2. Place transfer ink-side down on material.
3. Hold iron on transfer for 3 seconds, with pressure.
4. Leave transfer in place until cold, then peel off slowly.

NOTE: Once iron-on transfer has been transferred on to the material do not iron directly on transfer area.

1802 EDITOR/ASSEMBLER

G. E. Millar
PO BOX 1412
Parksville, BC, V0R 2S0

The assembler and editor programs were written to run on a system with 4K bytes of memory. To conserve space in the source area, the editor drops all but one leading space when storing the buffer into the source area. Both the editor and the assembler restore the spaces when displaying the source lines. This enables one to load approximately 250 uncommented lines (based on a average of 8 characters per line) into the source area (0700 to 0F40).

EDITOR COMMANDS

The editor recognizes ten commands plus a number of internal commands in the source load and pending line routines.

F (cr); find line number (line # in HEX)
L (cr); list number of lines (# of lines in HEX)
< (cr); back up number of lines (# of lines in HEX)
> (cr); forward number of lines (# of lines in HEX)

All of the above commands take the last two HEX characters entered prior to the "cr" as the number. Line numbers start at "00" and end at "FF".

eg. F300(cr) or F00(cr) or F0(cr) will all find line "00"
L010(cr) or L10(cr) will both list 16 lines from and including the pending line.
L1(cr) will list the pending line.
L0(cr) will be ignored.

P ;pending line (has five functions).

1. control/T ;will insert the next character entered.

2. control/C ;will delete character in the same column in pending line.

3. control/T(cr) ;truncates pending line at that point.

4. any character greater than HEX "1F" and not equal to "\" (prompt) will replace the character at that point in the pending line.

5. can be used to replace the pending line.

The "\" (prompt) increments the cursor position of the buffer without changing the line.

NOTE: In the following examples, control functions will be represented by:
control/T = *
control/C = +

TO INSERT A CHARACTER:

```
 STAT1:      LDA #88 ;------comment------
    *
P\\\R(cr)
 START1:     LDA #88 ;------comment------
```

TO CHANGE A CHARACTER:

```
 START1:     LDA #88 ;------comment------
P\\\\\\\\\\\\\I\\FA(cr)
 START1:     LDI #FA ;------comment------
```

TO DELETE A CHARACTER:

```
 START1:     LDI #FA ;------comment------
P\\\\\+(cr)
 START:      LDI #FA ;------comment------
```

TO TRUNCATE A LINE:

```
 START:      LDI #FA ;------comment------
P\\\\\\\\\\\\\\\\\*(cr)
 START:      LDI #FA
```

TO CHANGE A LINE:

```
 START:      LDI #FA
PSTOP:       XRI #88 ;------comment------
 STOP:       XRI #88 ;------comment------
```

If the new line is shorter than the pending line, it will have to be truncated; if it is longer, the command will automatically terminate.

I ; insert line function. Inserts line after the pending line.

D ; delete pending line (last line displayed).

W ; locate the next occurance of characters on these columns.

```
eg:  W\\\RT(cr)
 START1:    LDI #88 ;------comment------
```

S ; type in new source program. If this routine is entered directly on start up, it will point at the beginning of source (location 0700) and work up as lines are entered. If entry to the source loader (S) is entered from another editor function, the source pointer will be pointing to the start of the pending line. Any lines entered from the source loader will over write the pending line and will work up from that point. Exit from the source loader routine terminates the program and re-enters the editor. There are three internal

commands:

1. If the first character is a "space", the editor will generate a software TAB (seven spaces).
2. BS (back space), will decrement buffer and the next character entered will over write the previous entry.
3. ESC, terminates source program entry and re-enters the editor routine.

E ; End editing and return to the system monitor. The source pointer is displayed on exit. The pointer is used to find the end of the source program for the purpose of taping. Typing FFF(cr) will produce error #4. This action moves the pointer to the end of the source area. By entering "E", you will get the location of the last source byte in the editor work area.

EDITOR ERROR CODES

1. - Not an editor command.
2. - Not an ASCII HEX character.
3. - Line requested is less than zero.
4. - Line requested does not exist.
5. - Buffer is full.
7. - Source memory area is full.
8. - Input requires at least on HEX character.

ASSEMBLER

The assembler recognizes all 87 mnemonics contained in the 1802 users manual plus an additional 7 mnemonics. There are no expressions or macro instructons. The 7 additonal mnemonics are:

ORG ; program origin. This must be followed by an opfield consisting of a "#" and the HEX address of the program origin (start).

CALL ; call subroutine (SEP R4,"address").

RETN ; return from subroutine (SEP R5).

EQL ; must be preceeded by a label and followed by a "#" and the HEX address of the label.

BYTE ; call be followed by a label or can be followed by a "#" a and a HEX number (one byte). BYTE produces a single byte equal to the low address byte of the label or equal to the HEX number.

DBYT ; same as above put produces 2 bytes from the label address or HEX number (2 bytes).

END ; must be the last mnemonic in the program. This is used to terminate the program.

Example:

```
0000                      ORG #0100 ; pgm begins at "0100"
0100  D40F00 START:       CALL INIT ; jump to subroutine "INIT"
0103  F81C                LDI TAB1  ; continue with program
                          .
                          .
013A  C08051 STOP:        LBR MONITOR ; end of main program
013D         MONITOR:     EQL #8051 ; monitor entry "8051"
013D         ;
013D         INIT:        ORG #0F00 ; origin of subroutine is "0F00"
0F00  F8FF                LDI #FF
0F02                      .
                          .
0F1B  D5                  RETN        ; exit subroutine
0F1C  0100   TAB1:        DBYT START ; table of 5 bytes.
0F1E  FA88                DBYT #FA88 ;
0F20  3D                  BYTE INIT
0F21  ;
0F21                      END         ;terminates program
```

LABELS

If column 1 contains anything other than a "space" or a semicolon (;) , then a label is assumed. All labels must start in column 1 and must be terminated by a colon (:). Labels can be from 1 to 9 characters but the assembler only differentiates on the first 6 characters. The assembler does not check for duplicate labels. Labels can be used in the opfield (see opfield).

OPCODE MNEMONICS

If column 1 contains a "space" and the first non space character along the line is not a semicolon (;), it must be an opcode mnemonic. All labels must be followed by an opcode.

OPFIELD

All opcodes requiring an opfield must have at least one "space" separating the opcode and the opfield. The opfield can consist of a "#" followed by a HEX number or a label with exception of ORG and EQL which must have an opfield consisting of a "#" and a HEX number (2 bytes).

All register opcodes must be followed by at least one "space", the character "R" and a single HEX character representing the register number.

All short branch opcodes followed by a label use the low order byte of the label address and the branch is tested for out of page branch. An out of page condition generates an error #3 along with the source address of the error.

Note: Opcodes requiring a single byte opfield, when using a label for the opfield use only the low order byte of the label address.

```
Examples: LDA RB     ; load and advance register B
          LDI #F4    ; single byte HEX number
          LDI TAB1   ; low order used only
          ANI START  ; same as above
          BR  #03
          BZ  STOP   ; tested for out of page branch
          LBR MONITOR
          LBR #8051  ; double byte HEX number
          LBNF #0F00; there must be a least one or more "spaces" between
                     ; the opcode and the opfield.
```

COMMENTS

If the first non space character in a line is a semicolon (;), the rest of the line is comments and is ignored. The semicolon can follow the opcode or opfield -- the remainder of the line will be ignored.

ERROR MESSAGES

1. - not an opcode.
2. - opcode mnemonic not recognized.
3. - short branch out of page.
4. - label table full (PASS1); no label found (PASS2)
5. - opcode requires opfield.
6. - opcode does not require an opfield.
7. - labeled line requires an opcode.
8. - register opfield requires "R" or I/O number is incorrect.
9. - no colon (:) found after a label.

RUNNING THE ASSEMBLER

The assembler has 4 PASS2 options. After you get a "#" when PASS1 is done, type 0(cr) for a syntax check only, type 1(cr) to generate code only, type 2(cr) to get a full listing and generate code, and type 3(cr) to get a HEX listing and generate code. Remember that when you generate object code, the assembler is overlaying the source area from location 0700 on.

IMPLEMENTATION NOTE

Both the editor and assembler assume that the TTY output routines do not destroy D. If you get garbage when doing output, you will have to modify your I/O driver to save D on entry and restore D on exit.

MACHINE READABLE CODE

Object code is available on cassette in Kansas City "S" format from myself or Bernie Murphy. Please send a cassette, SASE, and sufficient postage.

RESIDENT EDITOR 1802

```
0000    F803 A3F8 00B3 D3B9 BBA8 A9AA ABF8 07B8
0010    F6B4 B5F8 04BA F80D A4F8 1FA5 F80F B2F8
0020    FDA2 D402 59F8 3AD4 0263 D402 66FB 4632
0030    71FB 7A32 C9FB 0232 D59F FB4C 32DF FB09
0040    C203 AEFB 1632 919F FB50 327B FB13 C800
0050    00FB 0A32 839F FB44 32BF FB13 C202 CF9F
0060    FD1F 3322 F831 C8F8 33C8 F836 D401 6030
0070    22D4 023F 8FBF D401 6E30 25D4 0398 D401
0080    9B30 25D4 00F8 D401 EF93 A9D4 0119 1B30
0090    25D4 0259 98FB 0F32 B7F8 21D4 0263 D401
00A0    EF87 FB1B 32B0 8FA9 D401 19D4 00F8 3094
00B0    58A8 F807 B830 25F8 37D4 0160 9330 B0D4
00C0    0398 89B9 D402 8E30 22D4 023F 8B52 8FF5
00D0    BF3B 6730 76D4 023F 8FA7 D401 7D30 25D4
00E0    023F 30E7 D400 F88F 3225 2F93 A7D4 017D
00F0    0832 251B 30E4 0000 48FB 0D3A F8D5 0000
0100    F800 AFAA B7BB 0AFB 203A 121F 4AFB 2032
0110    0C2A 4A1F FB0D 3A12 D5D4 0100 8952 8FF7
0120    3326 8FF5 B99B A932 2CD4 0269 9932 32D4
0130    028E 9BAA 98B7 88A7 0AFB 203A 444A FB20
0140    323D 2A2A 4A57 17FB 0D3A 44D5 D402 61D4
0150    0261 F800 AA4A D402 63FB 0D3A 55C0 0259
0160    BBD4 0259 F83F D402 639B D402 63D5 9F52
0170    8BF5 337D F807 B8F8 00A8 AB30 6EA7 0832
0180    9487 328D 48FB 0D3A 8427 1B30 7ED4 0371
0190    D401 4CD5 F834 D401 6030 5DF8 00AA D402
01A0    66FB 5C32 E8FB 5F32 C1FB 1732 D2FB 1932
01B0    B99F FD1F 339E 9F30 E7C0 03BC D402 5930
01C0    908A 52A7 9AB7 1A4A 5717 FB0D 3AC7 02AA
01D0    309E 8A52 9AB7 F83F AAA7 2A0A 572A 2787
01E0    F33A DB1A D402 665A 1AFB 0D32 B930 9EF8
01F0    00B9 AA8A FD3E CB02 2FD4 0266 FB20 C203
0200    64FB 2832 28FB 0532 2BFB 1632 399F FD1F
0210    C301 F39F 5A1A C001 F3F8 07A7 F820 5A1A
0220    D402 6327 873A 1C38 2A30 161B 9F30 36F8
0230    35D4 0160 F80D 5A1A 389F A7C0 036B 00D4
0240    02A7 FBCE 32C8 9FAF FEFE FEFE 73D4 02A7
0250    60FB CE32 599F F430 47F8 0DD4 0263 F80A
0260    C8F8 20C0 830F C083 0A88 7398 7348 3A6D
0270    98B7 8852 89F4 A73B 7D97 FC01 B760 2728
0280    0857 98F3 3A7E 1288 F322 3A7E 12D5 98B7
```

```
0290   7388 5299 F4A7 3B9C 97FC 01B7 4758 183A
02A0   9C58 42A8 02B8 D5D4 0266 FB0D 32CD 9FFF
02B0   303B C5FF 0A3B C2FF 073B C5FF 0633 C5FC
02C0   10D5 FC0A D5F8 32C8 F838 C001 9683 D5F8
02D0   00AA D402 665A 1AFB 0D3A D2AA 08FB 203A
02E0   EB4A FB5C 3AFE 8AFB 073A E10A FB5C C203
02F0   4AFB 51C2 034F EA48 F3E2 1A32 EB28 4832
0300   5FFB 0DCA 02FE 1BC0 02DB 0000 D3BF E296
0310   7386 7393 B683 A646 B346 A39F 300C D3BF
0320   96B3 86A3 6072 A6F0 B69F 301E BBF8 02A7
0330   9BF6 F6F6 F6FC F63B 3BFC 07FF C6D4 0263
0340   2787 3249 9BFA 0F30 35D5 1A18 C002 EB28
0350   08FB 0D3A 4F18 D403 71D4 01BC C000 25F8
0360   35C0 006C 8AC2 0219 C002 13D4 0259 C001
0370   00F8 00AA B9A9 98B7 88A7 07FB 203A 8AF8
0380   07A9 F820 5A1A 2989 3A82 475A 1A19 FB0D
0390   3A8A D500 0000 0000 F800 A998 B788 A747
03A0   FB0D 193A 9FD5 0000 0000 0000 0000 D402
03B0   5998 D403 2C88 D403 2CC0 8051 8A52 08FB
03C0   203A C789 FC07 3889 F53B CEF8 0D5A D401
03D0   19C0 01BC 0000 0000 0000 0000 0000 0000

0400 ---- 043F (Buffer)


0263   C0----  (Long branch to output routine)
0266   C0----  (Long branch to input routine)
03B9   C0----  (Long branch to system monitor)
```

```
0000  C002 11D4 02A8 08FB 2032 42FB 1B32 31F8
0010  06AD 9AFB 0732 7048 5A1A FB3A 3236 2D8D
0020  3A17 F804 AD48 FB3A 3236 2D8D 3A25 306B
0030  28D4 028D 3006 8AFA F8FC 06AA 975A 1A87
0040  5A1A D402 2CA3 1909 325E FE32 83FB C032
0050  6317 49FA 0332 30F6 325B 1717 3030 D402
0060  E730 30D4 0261 3030 F8FA A616 1616 1616
0070  1616 86D4 02BA D401 B498 B788 A7D4 029A
0080  C080 51F8 23D4 01B6 D401 B9FB 0D32 959F
0090  FF30 BD30 88D4 02A8 30CC D402 E7D4 0287
00A0  9DFB 023A D560 72B8 72A8 87F7 FD03 FEFA
00B0  06AF D401 B48F 2F3A B208 FB20 3AC4 18F8
00C0  0730 B1AF 48D4 01B6 FB0D 3AC4 D402 9787
00D0  7388 7398 7308 FB20 32E9 FB1B 329D 48FB
00E0  3A32 E9FB 3E32 6D30 DED4 022C FB46 32F6
00F0  FB77 329D 9FA3 49AF 0932 9AFE 3280 09A3
0100  30C8 13C4 D402 F509 FA03 AD32 88D4 01C1
0110  28FB 2DCE FB36 C200 6FFB 1832 6397 7398
0120  5288 BFEA F806 AF0A FB3A 3A34 08FB 20CE
0130  FB2D 323D 48F3 3A71 602F 8F3A 27E2 8AFA
0140  F8FC 06AA 8DF6 324E 4AAF D402 F513 1A0A
0150  AFD4 02F5 6009 FBC9 3A60 2A0A F3CA 0071
0160  C000 9D18 D402 D8AF D402 F52D 8D3A 6430
0170  609F A802 B88A FAF8 FC08 AA3B 249A FC01
0180  BAFB 07C2 0070 3024 D401 C1FB 1B28 CEFB
0190  36CA 006E 3060 0000 D401 C1FB 72CA 006C
01A0  D401 D952 8FF4 AFD4 02F5 3060 F80D D401
01B0  B6F8 0AC8 F820 C083 0FC0 830A 09FC 0130
01C0  0848 FB20 32C1 D500 C830 04D4 01D5 52FD
01D0  083B A430 9BD4 01C1 2848 FF30 CB00 68FF
01E0  0A3B EFFF 073B DCFF 06C3 0068 FC10 D5FC
01F0  0AD5 D3BF 96B3 86A3 6072 A6F0 B69F 30F2
0200  D3BF 9673 8673 93B6 83A6 46B3 46A3 9F30
0210  00F8 14A3 F802 B3D3 B4A6 F6B5 A4F8 F3A5
0220  A2F8 0FB2 E2C0 0003 60F0 A7D5 D401 C1FB
0230  2D32 5BFB 3632 5E28 F86D AFD4 0275 AF32
0240  5849 AF09 5289 FD2F 334D F804 B902 A9D4
0250  0275 8F32 58F8 46C8 F868 C8F8 6DC8 F831
0260  D597 BB87 AB2A 2AD4 02E7 975A 1A87 5A1A
0270  9BB7 8BA7 D58F 5289 F5CB 0502 8973 9873
0280  8852 93AD C004 E0F8 05BA F810 AA48 FB0D
0290  3A8D A9F8 03B9 D5D4 01AC 97D4 02BA 87D4
02A0  02BA D401 B4C0 01B4 F807 BBB8 F805 BAF8
02B0  10AA F800 B7A7 A8AB 3092 AF87 7393 A78F
02C0  F6F6 F6F6 FCF6 3BCA FC07 FFC6 D401 B627
02D0  8732 288F FA0F 30C4 D401 D9FE FEFE FE73
02E0  D401 D960 F4D5 0048 FB23 3AE7 D402 D8B7
02F0  D402 D8A7 D59D 32FE 8F5B 1BD4 02BA 17D5
```

```
0300    5000 746C 5800 7C74 4341 807C 4442 8480
0310    4400 8C84 4500 948C 4700 9C94 4900 AC9C
0320    4C44 C0AC 4C42 DCC0 4C53 F8DC 4D41 00F8
0330    414E 0800 4144 1808 4259 706C 424E 6448
0340    4200 4818 4E4C 6864 4E00 7068 4F00 8070
0350    5245 8C80 5253 948C 534D B4A4 5354 D0C8
0360    5348 E0D0 5344 C4B4 5300 C894 4C4F A098
0370    4849 B098 4F52 F304 5249 FB01 4C4C D402
0380    5954 00BD 4953 7104 4543 2098 514C 0060
0390    4E44 0080 4C4F 8098 4849 9098 4E50 68C8
03A0    5258 6004 4E43 1098 444C 0004 5841 7204
03B0    4920 F801 4E20 0098 4120 4098 5820 F004
03C0    5A20 C202 5220 C002 4E5A CA02 4446 C302
03D0    4E46 CB02 5120 C102 4E51 C902 4B50 C804
03E0    5A20 CE04 4E5A C604 4446 CF04 5120 CD04
03F0    4E51 C504 4945 CC04 4E46 C704 524B 7904
0400    4420 F204 4920 FA01 4420 F404 4943 7C01
0410    4920 FC01 4320 7404 5220 30C9 5A20 32C9
0420    4446 33C9 505A 33C9 4745 33C9 4D20 3BC9
0430    4C20 3BC9 5120 31C9 3120 34C9 3220 35C9
0440    3320 36C9 3420 37C9 5A20 3AC9 4620 3BC9
0450    5120 39C9 3120 3CC9 3220 3DC9 3320 3EC9
0460    3420 3FC9 4252 C804 4950 C404 5445 00BC
0470    5247 0000 5249 F901 5220 F104 5554 60C8
0480    544E D504 5420 7004 5120 7A04 484C 7E04
0490    4852 7604 4550 D098 4551 7B04 4156 7804
04A0    4B50 3804 4249 7F01 4220 7704 4920 FF01
04B0    20F7 0400 4249 7D01 4220 7504 4920 FD01
04C0    20F5 0400 4558 E098 5844 7304 5220 5098
04D0    4C43 7E04 5243 7604 4C20 FE04 5220 F604
04E0    E909 32FE FB20 3AF3 08FB 20CE FB2D CEFB
04F0    3632 FE48 F3CA 0504 602D 8D3A E138 60E2
0500    6072 AFD5 E272 A872 B8F0 FC04 A9C0 0275

0510 ----- 06FF  (Lable table)

0080    C0____   (Long branch to system monitor)
01B6    C0____   (Long branch to output routine)
01B9    C0____   (Long branch to input routine)
```

"NBR" has been removed from the table.

MEMBERSHIP RENEWAL

Your 1978-9 club membership expired on May 31, 1979. Issue #12 is the last issue of the 78-9 club year.

The dues for the new club year, June 1979 - May 1980, are now due. Dues for the new year are as follows: Canadian $15.00, American and overseas $18.00, including FIRST CLASS POSTAGE for 6 issues.

Please make cheque payable to "The Association of Computer Experimenters" and mail to c/o M.E. Franklin, Treasurer, A.C.E., 24 Duby Road, Acton, Ontario, L7J 2P1. (519-853-3421)

Please will out the membership application/renewal form below and attach mailing label (or reproduce it exactly).

NEW MEMBERS

New members are required to pay the full dues for the current year and will receive all the IPSO FACTO issues of the current year (June 79-May 80). Back issues may be purchased for $30.00 (Issues 1-12, May 1977 - May 79).

MEMBERSHIP: NEW 1979-80_____ RENEWAL 1979-80_____

DUES ENCLOSED: 1979-80 $_____ BACK ISSUES $_____ TOTAL $_____

Family Name First Names Membership No.

Address Apt. No.

City Province/State Country Postal Code

MEMBERSHIP QUESTIONAIRE

There were about 500 members at the end of the last fiscal year. Approximately 250 lived in Toronto/Hamilton and surrounding area. Of these about 50 are active in the club. Therefore, the club activities itself are directly of interest to about 10% of the club, while the newsletter affects 100% of the membership. Please take a few minutes to complete the attached questionaire to assist the IPSO FACTO editors in determining and meeting your needs.

As you may know, the six major 1802 manufacturers utilize different bus sizes and pin assignments and different hardware assignments. By letting us know what you have, IPSO FACTO will be able to ensure that coverage is given to all areas of interest and conversion data is available for all, or most, equipment in use.

Please check (✓) for existing working systems and use (D) to indicate a system/component in development.

Micro Computer		Memory		I/O	
Homebrew	____	Ram	____	Keyboard	____
Homebrew-Popular Science	____	Size	____K	Hex Pad	____
Netronics ELF II	____	(EP) ROM	____	Leds	____
Quest Super ELF	____	Size	____K	Hex Led Display	____
Infinite 1800	____			1861 Video	____
TEC 1800	____	Monitor		Video Board	____
RCA-VIP	____			Cassette	____
RCA - Studio II	____	Netronics	____	Cas. Kansas City Standard	____
Southern Scientific	____	TEC	____	Printer	____
Other - Specify	____	Quest	____	Flopy Disk	____
_____	____	Infinite	____	Other - Specify	____
_____	____	Benchmark	____	_____	____
_____	____	RCA Opcode	____	_____	____
_____	____	Other-Spec.	____	_____	____
_____	____	_____	____	_____	____

Language Please indicate order of preference/use		Application - Use - Orientation Please indicate order of preference/use	
Machine Language	____	Hardware Experimentation	____
RCA - VIP - CHIP 8	____	Software Development	____
Tiny Basic - Netronics	____	Games - Entertainment	____
- Infinite	____	Business/Personal Financial, Etc.	____
- Quest	____	Security Alarm/Sensor Control	____
- RCA	____	Industrial Controlling	____
- Itty Bitty	____	HAM Radio	____
Palo Alto T.B.I.	____	Music/Sound Generator	____
Other - Specify	____	Medical	____
_____	____	Other - Specify	____
_____	____	_____	____