THE COMPUTER HOBBYIST

Volume 1 Number 7

By the time you read this TCH will have gone to another hamfest, this one in Shelby, N.C., so Hi all you folks whom we just saw. TCH will be going to some other hamfests with our system this fall. On September 13 we will be at the ARRL national convention in Reston, VA at the Sheraton Motor Inn. Hamfests in Gaithersburg, MD and Stone Mountain, GA are also possibilities for this fall.

Upcoming articles for TCH include a series on the IMP-16 starting with the next issue. Also, TCH now has an Altair 8800, so issue 8 should usher in a series of articles on I/O and interfacing.

Now a few words about TCH's health and our future. We are slow (and how) in cranking these things out. Consequently, we are abandoning our claim to be a monthly and only claiming to be an "almost" monthly, i.e., we reserve the right to publish that often if we can manage. Don't be upset about your hard earned coins which you sent us however, for they will still get you the same number of issues (12 for $6). For those of you who are already in our cassette based record keeping system, your mailing label begins with "Sub #### Last ##". The number after "Last" is the issue number with which your subscription expires. If this number differs from your records please notify us immediately. If your mailing label has any other format, hang in there; you will be in the new system in a couple of months. As for TCH's health otherwise, its good. We now have over 1500 subscribers and have yet to see slack in our mail. We are also planning to go to 3RD class postage. This will make getting TCH slower but will allow us to put out larger issues with no increase in postage. Also TCH is now actively seeking outside authors and advertizers.

One last item, TCH wants to publish a list of clubs accross the country, so if you want your club listed drop us a card and you will be in our next issue. Be sure to include important facts such as meeting times and locations and names of club leaders.

TCH AUDIO CASSETTE STANDARD ROM

by: Richard Smith and Hal Chamberlin

The TCH audio cassette ROM provides basic support software for the TCH universal audio cassette interface. Functions include reading a record into a buffer in main memory, writing a record from a buffer, IPL (initial program load, cold start, bootstrap, etc.) from cassette tape, and interface control on both units. At the beginning of the ROM is a jump vector with an entry for each of the four main functions: CTROM - read record, CTROM+3 - write record, CTROM+6 - Control routine, and CTROM+9 - IPL routine. This jump vector allows software compatibility with possible updates to this program and similar packages for the Phi-Deck by keeping the call addresses of the routines invarient.

The read record routine reads the next record from the tape into a user specified buffer area in memory. When called, register A should contain a unit select mask specifying the tape unit to be used. A mask of all zeroes specifies unit 0 while a mask with the unit select bit (octal 020) on specifies unit 1. Equates between the unit number and the unit select mask are a convenient way to keep this straight. H and L should have the address of the buffer that will receive the data read. When the read record routine returns, register B will contain the length of the record read in and H & L will point to the next available buffer word. The condition code indicates whether a CRC error was encountered during the read. Everything is OK if the ZERO flag is on. Note that the read routine uses all of the registers and except for B, H, and L they will be in an undefined state on return.

The write record routine is the exact complement of the read routine. Register A contains the unit select parameter as before and H & L points to the buffer to be written from. Register B specifies the length of the record to be written. Since errors cannot occur when writing, the only return information is the content of H & L which points one beyond the last byte written.

At this time the control routine does nothing except reset the interface. It was included for future compatibility with a Phi-Deck interface in which case it would handle rewinding, beginning of tape search, etc.

The IPL routine can be used to read a cassette tape file into memory and branch to it. It is particularly useful after power on or a major crash since one needs only to branch to it (CTROM+9) and put the appropriate cassette into the recorder to recover. Preprogrammed into the ROM is the address in memory to read into and jump to after reading. The IPL program data format is simply memory image starting from the load address. The routine can handle multiple records so program length is unrestricted. It stops reading when an end-of-file record (zero length) is encountered and branches to the program loaded. 8080 users need not be concerned with the stack pointer since the routine initializes it to a preprogrammed address. This address should be specified when ordering the ROM. Since the 8080 predecrements the stack pointer when pushing, the address specified should be one more than the address of the top of the stack. This IPL routine requires 10 stack bytes but for future compatibility a few more should be reserved. If an I/O error is encountered during loading, the routine halts by means of an infinite loop.

Appendix I shows an example program which copies the tape on unit 0 onto unit 1. The first step taken by the program is to reset the tape interface by calling CTCN. Next, a record is read into a memory buffer starting at BUFF. This buffer should be 255 bytes long to accomodate the longest possible record. If no read errors were detected, register B is tested for zero to determine if an end-of-file record was read. If so, an end-of-file record is written by calling CTWR with a length specification of zero and the program halts. Otherwise H & L are reset to the beginning of the buffer, A is set for unit 1, B is left unchanged and CTWR is called to write the record just read. The program then loops to read the next record.

Due to Murphy's 19th law (a program is always 30 bytes too long), the coding in the ROM can be tricky in places. Hopefully this discussion, the detailed flowcharts, and the commented listings for 8008 and 8080 will help. One overriding consideration was that no memory be used for temporary storage, i.e., all counters, pointers, etc. should be kept in registers. In general the program structure follows the record structure with a separate routine for each component of the record format. Perhaps the most common trick is to fall into an adjacent subroutine rather than calling it if that is the last operation to be performed, thus saving 3 bytes. Elaborate equates are used to make the program more readable.

The control routine resets the interface by sending it a word of all zeroes. This selects unit 0, turns off both motors, and sets read mode.

After turning on the specified unit's motor, the read routine goes into a loop to search for the data ID. The ID is located by defining a 16 bit shift register using B (high part) and C. After shifting BC left by one and reading the next bit from the tape into bit 0 of C, the contents of BC are compared with the ID pattern. The only way to have an equal compare is if the last 16 bits read were the data ID pattern. Clever arrangement of the code and registers results in the CRC register (registers D and E) being zero when the ID is detected. Next, the record length is read in. Saving the record length for use by the calling program while also using it for a loop count presented some problems as all of the registers were already used. The 8080 solution is simple, push it onto the stack. The trick on the 8008 is to save the length in the buffer position reserved for the next byte. When a data byte is to be stored in the buffer, the count is loaded back into A and the byte stored from C. This works fine as long as data is read into operable memory. When the byte count reaches zero, the CRC is checked by simply reading in two more bytes but not storing them. If the CRC is zero after this, the record was OK. Finally the motor is turned off, the CRC register is tested for zero, and a return is made.

CONTINUED ON PAGE 4

LETTERS

We at TCH will publish a few of our more interesting letters each month along with comments by the staff.

Editor:

I'm curious about the May 1975 issue. I received it on July 7. Since it was first class are you getting cheated by paying first class and getting bulk service?

My main reason for writing is to compliment the writer of the May editorial. It is a damn good, well balanced piece of journalism, even though I don't totally agree with it. Hope to see more like it in future issues.

I like the graphics display and will probably try to get one together in the future. Was wondering if the MP digital display might not be interfaced with the CRT portion of the display without too much trouble.

You guys really put together good information and detailed descriptions--thanks.

Interested in your PC board for the cassette interface, but due to vacation expenses will be unable to order until the end of the month. If the supply is limited, could you save me one?

Bill Fuller

No the post office isn't cheating us, we are just slow and tried to keep the dates on the issues sequential in spite of that fact. For more comments see the discussion on the front page.

Your letter and the two following are just some of the many we received mentioning the Altair editorial. They are typical of the opinions expressed. TCH is glad you folks do read and care.

The MP digital display cannot be interfaced to the TCH graphics system, however any programs written for it should be easily convertable to run TCH's display (the reverse is NOT true).

Cassette boards are being supplied on a continuing basis. Notice will be given before they are discontinued. To date over 75 units have been shipped. Also due to our early success TCH is regrettably out of relays so do not order them now.

Gentlemen:

As an owner of an Altair 8800 and a computer hobbyist I would like to comment on your editorial in the May '75 issue of TCH.

I purchased the basic computer during the initial promotional effort by MITS. I have always considered it an excellent buy. Components are of good quality and the kit assembly instructions were above average.

MITS's marketing strategy seemed to be directed toward the average hobbyist on a limited budget. The basic 8800 fullfilled this requirement nicely. The MITS add-ons and peripherals are definitely overpriced and, I suspect beyond the means of most 8800 purchasers. I happen to agree with the "loss leader" theory.

In addition, MITS has not offered a single piece of software at a reasonable cost. Few people have the time and resources for software development. It should not be necessary to point out the usefullness of a computer without software. MIL's MONITOR-8 package, which was free for the asking, showed what can be done.

I have asked MITS for additional technical data for trouble-shooting and maintenance if this should become necessary. Their answer "data is not available at this time" is somewhat difficult to swallow. Another ripoff?

Fortunately others have tried to fill some of the gaps: The Digital Group, MiniMicroMart, Processor Technology, and TCH of course. This should not relieve MITS from their responsibilities implied in advertisements and in the PE article which started it all.

Your primary concern should be a look at a system (CPU, hardware, and software) from the standpoint of the hobbyist. Although I agree with your editorial, this is one point you obviously missed.

I enjoy your publication and it has helped me greatly getting started in the micro-computer business. Hope to see more articles on 8080 based systems. Keep up the good work.

A. P. Stumpf

Gentlemen:

I have just received, and devoured in one day, the first six issues of TCH. Excellent!

I particularly appreciated the editorial in issue #6. Your analysis displays both understanding and courage. Understanding of the interaction of economics and technology in the fast developing areas of computers and microelectronics, and courage to present unpopular facts. It is unfortunate, but many people find it easier or more comforting to blame others, often "big business", than to recognize their own motivations, or lack of foresight. We should all firmly affix in our mind that even the ALTAIR 8800 will be obsolete, though still useful, in at most two years. We should further recognize that even IBM can not limit the speed or spread of technological development.

All computer hobbyists should stop and consider where they will be, if in the next few years Digital Equipment Corp. manages to produce the LSI-11 for about $100. We are in an exciting field, but that excitment is because of, not in spite of, technological progress. If we recognize that, in the not too distant future, home computers will be as ubiquitous as component Hi-Fi equipment; we must expect tremendous improvements in both computer power and economies of production. As the market expands, more and more capital will be invested in research and production improvements, and we will be the beneficiaries.

Computer hobbyists should not glory in the exclusiveness of their hobby. They should anxiously look forward to the entrance of many participants, because these participants will attract real big business, and capital.

Micheal A. Sicilian

THE COMPUTER HOBBYIST
Founded October, 1974

Stephen C. Stallings - Managing Editor
Hal Chamberlin - Contributing editor
Jim Parker - Contributor
Edwin Tripp - Photographer
Richard Smith - Programming consultant

THE COMPUTER HOBBYIST is published at intervals approaching monthly near Raleigh, N.C. The subscription rate for 12 issues is $6.00 for persons in the U.S.A., U.S. possessions, Canada, and Mexico. Subscriptions in all other countries are $8.50 per year for surface mail or $13.50 per year for air mail. All back issues are available and cost fifty cents each for persons in the U.S.A., U.S. possessions, Canada, or Mexico. For all other countries back issues cost 75 cents each by surface mail or $1.25 each by air mail. Remittances from countries other than the U.S.A should be in the form of a bank or postal money order.

THE COMPUTER HOBBYIST is seeking paid contributors. Material to be submitted should be typed or neatly written and must not appear to be soliciting business for any firm. For details, please inquire.

Advertizing space is available to interested forms. please inquire for rates and formats.

All correspondence should be addressed to:

THE COMPUTER HOBBYIST
Box 295
Cary, NC 27511

EDITORIAL

Unlike our other articles and commentaries the editorial will be mostly opinion, sometimes that of a single staff member, and other times that of TCH as a whole. Reaction to the editorials, either supportive or dissenting, is welcome.

Are console control panels really necessary in hobby computer systems? By control panel I mean the array of toggle switches, push-buttons and lights often seen on the front panels of computers. Lets look at some history and see how the traditional control panel evolved.

Early computers were used in much the same way a lot of people are using their Mark-8's and Altair's now, programs were entered and results obtained from the front panel. One major difference though was that these early machines displayed all of their registers and major logic states as well for maintenance purpose. The control panels on these beasts were indeed large and impressive and still persist in the minds of the public through the efforts of Hollywood. Large computers now (IBM 370-168) have all of their console functions microprogrammed and channelled through an ordinary keyboard and alphanumeic CRT display, no more switchs and lights.

Minicomputrs have also gone through much the same evolution. Early ones displayed all of their registers any of which could be loaded from the console switches. Later the console was reduced to simply a data bus monitor and console access was essenially restricted to memory. Now most minicomputers are priced less console which is a $200 to $500 optional feature.

Why the trend away from consoles? The answer is that software and a basic I/O device such as a teletypewriter can provide much more convenient and extensive console functions than even an elaborate control panel. Any serious system is going to have the requesite I/O device anyway. Even a simple console emulator program (we will call it a "DEBUG" program) allows memory contents to be printed and entered at typing speed, allows registers to be examined and changed, and can start user program execution at a particular address all with simple keyboard commands.

Powerful debugging aids are available in the more advanced debug programs. These can take the form of either "trace" or "breakpoint" functions. A typical trace routine will print the instructions executed and all of the registers that are modified between trace limit addresses. The trace limits are set by the user from the keyboard. Trace routines work either by simulating execution of the user's program or by clever use of the interrupt system. The major disadvantage of trace, slow execution outside the trace limits when trace is enabled, is avoided by the breakpoint function. Typically the user can set one or more breakpoints at particular points in his program with keyboard commands. When program ex-

ecution hits a breakpoint, the breakpoint address and the registers are printed and then execution resumes. The breakpoint routine works by inserting calls to itself in the breakpoint locations and keeping track of the instructions replaced by the calls. Breakpoint routines are typically smaller but require more planning to use effectively. Either method substitutes nicely for single cycle console controls and provides a written record of program execution for detailed study. After all, large computer users have only a memory dump after catastrophic program failure to go on.

The fact is that microprocessors were never intended to have consoles. For that matter, the older ones (including the 8008 and 8080) were designed as logic replacements and dedicated controllers rather than general purpose computers for problem solving. These two facts are readily apparent to an engineer who tries to design a good console or integrate the microprocessor into a general purpose system. Nevertheless both tasks can be accomplished such as in the MITS system or the Mark-8 system. Considerable money is spent in either case on console logic, quality toggle switches and buttons, and a well labelled front panel. Often compromises are made to accomodate the single cycle function without letting the bus control logic get out of hand. As mentioned before, any serious user of these systems has or will have the keyboard-printer or display necessary to support a software console and once he tries a debug program, the hardware console may never be used again! Two interesting side points are that most complaints about existing hobby computers relate to console malfunctions and that the newer systems such as SPHERE have no control panel.

How would a consoleless computer look and operate? There would probably be three switches and two or three lights. One switch would be a power on-off toggle, one a reset button, and one an interrupt button. The first light would be a power on indicator, the second would indicate when the CPU is halted and the optional third light would indicate whether interrupts were enabled or not. Some processors may require a start button if interrupt doesn't work when the machine is halted, e.g. the IMP-16.

Now how would one get this underendowed creature started when the power is turned on? In a well designed system, a "power on reset" circuit (POR) will effectively press the console reset button when power is first applied. The reset signal should go to all of the peripherals resetting them to an idle state and it should cause the CPU to jump to a program stored in read only memory. At this point two possibilities exist. One is that enough ROM is available to hold the entire debug program in which case the restart procedure is complete. The other possibility is that a small program called a bootstrap loader is all that is in the ROM. The loader would then proceed to read the debug program into regular read-write memory from an input device such as a paper tape reader, cassette, etc. and jump to it. The bootstrap data format is generally simplified and inefficient in order to minimize the bootstrap loader size. In either case we now have control of the system through debug program commands. Ideally the console interrupt button would be seperate from the I/O interrupt system and always enabled. Pressing the interrupt button would cause an "interrupt entry" into the debug program which would save the registers, status, and return address for examination and alteration by the user. A debug command would be available for resuming execution at the point of interruption. That is all there is to it.

Some microprocessors are better adapted to this mode of operation than others. In particular the Motorola 6800 and the National PACE and IMP-16 seem to be designed with this specifically in mind. Nevertheless, any of the popular chips can be effectively operated without a console.

BOOK REVIEW by Hal Chamberlin

Machine Language Programming for the 8008, Wadsworth, Nat, Scelbi Computer Consulting Inc., 1322 Rear Boston Post Road, Milford, CT 86460

We have been constantly getting requests for information and articles on basic level machine language programming for all of those hobby computers out there. Well here is the answer! This 168 page book has all of the answers and guidance that a beginning programmer could want. It is far more detailed and down to earth than even DEC's classic Small Computer Handbook that until now had been the best tutorial publication available on the programming of small binary computers. The coverage is so broad and well done that some of the articles that we had planned on programming will have to be altered or dropped to avoid duplication.

As the title implies, the contents are directly applicable to the 8008 microprocessor. The large number of fully commented example programs and routines are directly usable on the 8008. Neither the 8080 nor any other microprocessor was mentioned but most of the concepts and techniques presented would be applicable to other machines. Many of the added instruction functions on the 8080 are developed as subroutines in the text, thus where a subroutine might be called in a sample program, the 8080 user could simply supply the appropriate instruction.

One noticable feature of the book was a complete lack of any commercialism whatsoever. As the reader may know, Scelbi is a manufacturer of 8008 based systems for hobbyists and schools and also sells extensive software for the systems. Scelbi computers are seldom mentioned and none of the text or examples made use of or even mentioned any particular features of the Scelbi system. Even the chapters on input/output programming were kept general; never mentioning the hardware or software techniques used in Scelbi I/O devices. In short, one would never suspect that the book was written by a manufacturer of the computer it describes.

The book is in the form of 8 1/2 by 11 inch pages bound in a soft cover report binder with metal binding tabs through the three hole punching. Offset printing of good contrast on one side of the pure white medium weight paper is employed. The type was obviously set on a teletype machine in all caps with a cloth ribbon so the character quality was not particularly good but nevertheless easy to read. The absence of typographical errors indicates that the author made effective use of his editor program. Space utilization on the printed side of the paper was good due to single spacing and narrow margins. Some of the simpler drawings were formed with teletype characters much like those seen in IBM manuals.

In the introduction a very convincing (and accurate) argument for machine language versus high level language programming of hobby computers is presented. The first chapter which is 21 pages long gives an original, truly readable description of the 8008 instruction set. An interesting approach is taken in explaining the op-codes. Rather than utilizing binary op-codes and then having to explain binary-to-octal conversion so early, octal op-codes are used initially. The result is that explanation of operation encoding is much simpler. For example, the load register immediate instructions have the form 0R6 where R is replaced with the register number to be loaded.

Now that the beginning programmer has been introduced to the "tools" he will use, he should be ready for the second and third chapters which discuss the steps used in program development and some necessary programming skills. A clear, accurate understanding of the problem to be programmed and the desirability of a flowchart are emphasized as prerequisites to a smooth, rewarding program development cycle. Number conversion is taken up as a programming skill along with the use of memory maps and coding sheets. Through the use of examples, the desirability of using an editor and assembler for long programs (over 100 instructions) is demonstrated. Manual coding is developed as an actual hand assembly process so that the use of an assembler should come quite naturally to the reader when he becomes advanced enough to need it.

Chapter 4 will be of great value to the beginning programmer because it is the chapter on basic programming techniques. In 37 pages the discussion proceeds from how to clear the accumulator to development of search and sort routines. Several utility routines, some of which substitute for 8080 instructions, are also developed in this section and their use is explained. Only the simplest search and sort algorithms that get the job done are presented. The more advanced and efficient methods are left to the computer science text books. The examples are always based on actual, concrete requirements, never on abstract or theoretical considerations.

I suspect that many people may buy this book solely for the contents of chapter 5. This is the section on arithmetic and is 45 pages long. The discussion starts with multiple precision add and subtract and proceeds to develop general purpose multiple precision add, subtract, and complement routines. After a general discussion of binary fractions, floating point notation is introduced. From here to the end of the chapter the six floating point operations (addition, subtraction, multiplication, division, ASCII-to-float, and float-to-ASCII) are discussed in detail and the algorithms converted into assembly language code. In other words, chapter 5 includes a floating point package. The four math routines were punched up and assembled at TCH and they appear to work correctly (be sure to consult the errata sheet supplied with the book). We have not tried the conversion routines but they should work also. Extensive use is made of subroutines developed in earlier chapters. According to the author, the code was optimized for ease of explaination and understanding and as a result is apt to be both time and space inefficient. The reader is then encouraged to rewrite the routines for greater efficiency once he thoroughly understands them in the present form. Probably the greatest speed gain will be in the multiply and divide routines and the greatest space gain in the conversion routines. Scelbi offers a listing with object code but no comments to owners of the book for $5.00. It is well worth the price if an assembler is not available.

The last four chapters discuss such diversified topics as input/output programming, real-time programming, and creative programming concepts. The depth of discussion is not very great but enough is said to get the reader thinking in the correct terms.

In summary, Machine Language Programming for the 8008 is a must acquisition for the beginning programmer, especially a hardware man. Even the experienced programmer should sit down with a copy for an hour or so because he is bound to discover something he has not known or thought of before.

CONTINUED FROM PAGE 1

The read bit routine (CTRD) waits for the clock from the interface to make a high-to-low transition and then reads a bit into the low order position of C shifting the remaining bits left. The bit read is also combined with the CRC register in DE by CTCC before returning. Thus all bits read will be factored into the CRC. The read byte routine (CTRB) simply calls the read bit routine 8 times to accumulate a new byte in register C. Due to the lack of registers, the count is done by setting a dummy bit in position 0 of C and calling CTRD until it is shifted into position 7. The routine then falls into CTRD for the eighth bit.

Operation of the write routine is similar to read. When entered, CTWR waits until the motor busy status becomes zero to insure a proper record gap after the last write operation. If more than .5 second has elapsed since the last operation, the status will already be zero so there is no waiting. The motor on the specified unit is then turned on and another wait on motor status is performed before writing is started. The 32 leading ZERO bits are written by effectively calling the write byte routine (CTWB) 4 times with zero data. The data ID is then written by two calls to CTWB with the appropriate data. At this point the CRC register in DE is zeroed in preparation for the remainder of the record. Writing of the record length is a bit tricky in order to conserve space. The length is first loaded into C from B and a jump into the middle of the write data bytes loop is taken. The byte count in B is incremented in order to compensate for the additional pass thorough the bottom of the loop. The routine still works properly for zero length and maximum length records in spite of this trick. Since HL is incremented at the top of the loop, it will point to the last byte written + 1 on exit. When writing the CRC, the low half is saved in B while writing the high half first because the act of writing the CRC also changes the CRC. Finally the trailing zeroes are written, the motor is turned off and a return is executed without waiting for the motor to stop.

The write bit routine shifts register C left by one and writes the bit shifted out. The test for write busy is done first to overlap the serialization processing with the time necessary to write a bit. It is necessary to read the the unit select bit from the interface so it can be combined with the write command bits sent to the interface. Otherwise, the unit select bit would be destroyed. Each bit written out is combined with the CRC as in the read bit routine. The write byte routine (CTWB) writes register C by calling CTWD 8 times. Bit counting is done by calling CTWD once and then appending a "stc" bit in the vacated position 0. When CTWD has been called enough times so that C contains 10 000 000, the remaining bits have been written and the routine returns.

During the course of writing these routines, several interesting observations were made. One was that the 8080 code was not significantly shorter. The two byte I/O instructions and the all-in-registers requirement were mainly responsible. Another was that there is a lot of work between an operable program and a fully optimized one (notice that we changed our minds on the organization since issue 6). Finally, this program might serve as a reasonable, notrivial benchmark for comparing microprocessors.

APPENDIX 1

```
;       EQUATES

CTROM   EQU  _______        ; CASSETTE TAPE ROM (CTROM)
CTRR    EQU  CTROM+0        ; READ RECORD ROUTINE
CTWR    EQU  CTROM+3        ; WRITE RECORD ROUTINE
CTCN    EQU  CTROM+6        ; CONTROL ROUTINE
CTIPL   EQU  CTROM+9        ; IPL ROUTINE

CTUN0   EQU  000Q           ; UNIT 0 SELECT
CTUN1   EQU  020Q           ; UNIT 1 SELECT


;       COPY PROGRAM

COPY:   MVI  A,0            ; RESET THE CASSETTE TAPE INTERFACE.
        CALL CTCN           ;
LOOP1:  LXI  HL,BUFF        ; READ INTO THE BUFFER A RECORD FROM
        MVI  A,CTUN0        ;    CASSETTE TAPE UNIT 0.
        CALL CTRR           ;
        JNZ  ERROR          ; BRANCH IF AN I/O ERROR.
        MOV  A,B            ; BRANCH IF AN END OF FILE RECORD.
        ORA  A              ;
        JZ   EOF            ;
        LXI  HL,BUFF        ; WRITE OUT THE RECORD TO CASSETTE TAPE
        MVI  A,CTUN1        ;    UNIT 1.
        CALL CTWR           ;
        JMP  LOOP1          ;
EOF:    MVI  A,CTUN1        ; WRITE THE END OF FILE RECORD TO
        CALL CTWR           ;    CASSETTE TAPE UNIT 1.
        HLT                 ; DO A DONE HALT.
        JMP  COPY           ; GO START ANOTHER COPY.
ERROR:  HLT                 ; DO A HARD ERROR HALT.
        JMP  ERROR          ;
```

NEW PRODUCTS

With this issue we are starting a new products column. In it we will list some of the new commercial products of interest to the computer hobbyist. Comments made will, in general, be condensed from the manufacturer's literature. Occasionally we may make a comment of our own if a particular feature is unusually impressive. Listing in this column, of course, does not imply endorsement of the product by TCH.

A new, nicely packaged microcomputer kit is being offered by Comp-Sultants. It is based on the Intel 4040 CPU chip. The basic machine has 256 bytes of program memory, (the 4040 has separate program and data memory) one input port, one output port, a control panel, and of course the CPU chip. An unusual feature is that the entire basic machine, including the control panel, fits on one large PC board. The machine is housed in a handsome but inexpensive metal cabinet. There is sufficient room in the cabinet to expand the memory to 8K words (each 2K memory board adds 8 I/O ports as a bonus). The basic kit costs $275 and the assembled unit costs $375. These prices were taken from a press release and do not agree with the glossy sheet price of $300 and $400 respectively.

COMP-SULTANTS, Inc.
P.O. Box 1016
Huntsville, Ala. 35807

Cramer Electronics, a well known industrial distributor of electronic components, has introduced a line of computer kits called Cramerkits. A Cramerkit consists of all of the parts (IC's, resistors, caps, etc.) necessary to build a microcomputer along with a documentation package including circuit diagrams and wirelists. NO interconnection hardware is supplied. The buyer is expected to build the system on wire-wrap cards or the equivalent. Cramerkits have or will be introduced for every major MOS microprocessor and probably some of the bipolar microprocessors. Available now are kits for the Intel 8080, the TI 8080, and the Motorola 6800. All kits have 1024 bytes of RAM, 1024 bytes of eraseable ROM (using the new 2708 8K bit easy-to-program eraseable PROMS), 4 input ports, 4 output ports, basic control panel, audio cassette interface (as published in Popular Electronics Sept. 1975), and RS-232 or TTY current loop serial I/O. The PROM comes with a debugging program and cassette read/write routine already programmed in. The TI kit includes a TMS-5501 "utopia chip" (UART, 5 interval timers, 8 level priority interrupt control, 2 I/O ports) whereas the other kits rely on software for these functions. The price of $495 is the same for any of the kits. A 2708 PROM programmer kit which connects to two of the output ports will be available shortly for around $70.

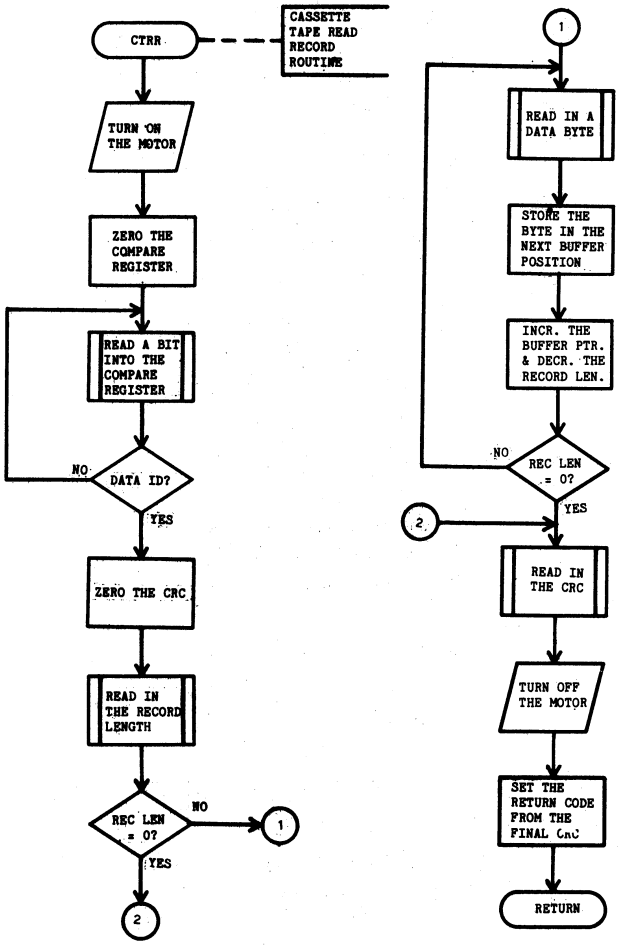
Cramer Electronics, Inc.
85 Wells Avenue
Newton, Mass 02159

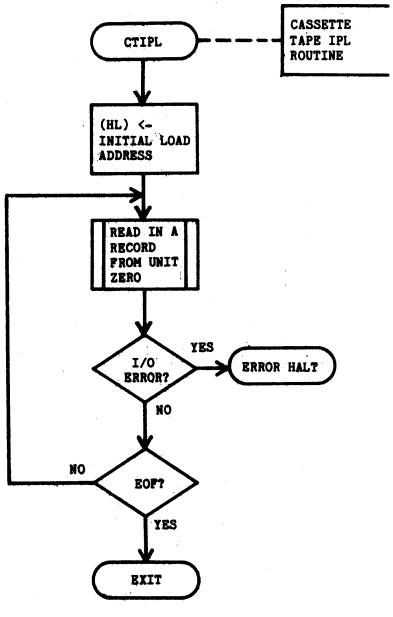
USE IT FOR WHAT IT WAS INTENDED?

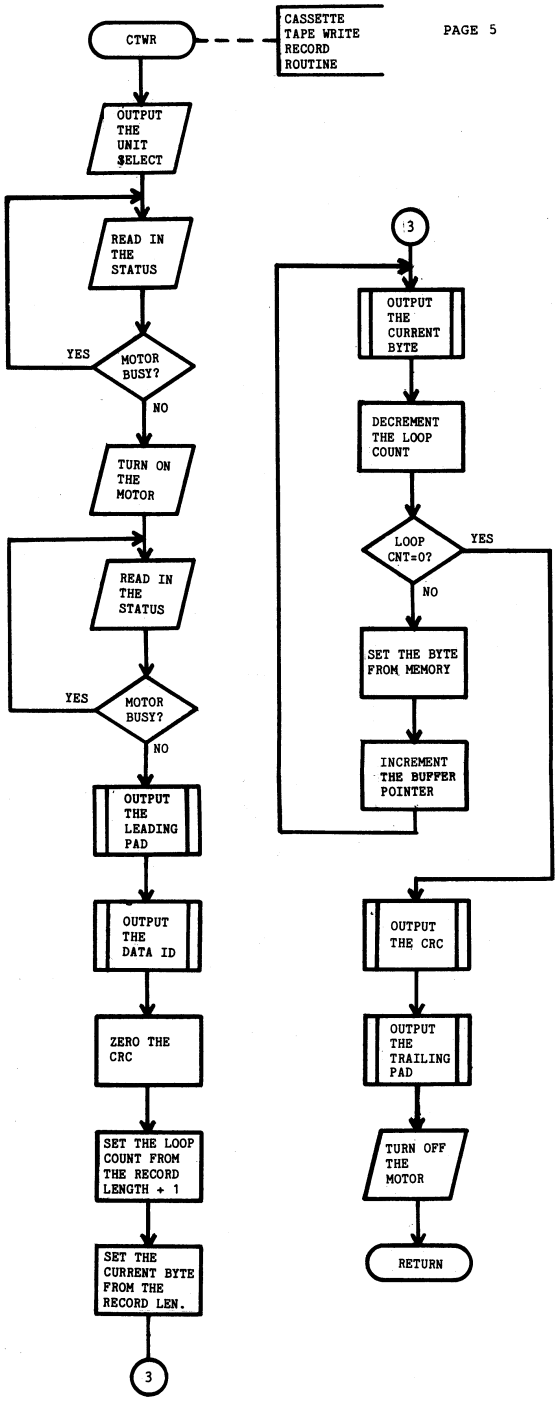
The Cyclops computer compatible TV camera (see Popular Electronics February 1975) is the best example of looking beyond the manufacturer's spec sheet we have seen in a long time. It seems that the "special image sensor chip" used is simply an MK4008 1K dynamic RAM with the metal cap replaced by a glass one. This dynamic RAM is unusual in that reading a location does not refresh it; a special refresh cycle is used instead. The rate of charge leakage from the storage capacitors in the dynamic cells is dependent on temperature and, you guessed it, ambient light. To use the modified memory as an image sensor, ONES are written into all locations. Then the memory is scanned repetitively with a scan count maintained for each bit position. The number of scans before a given bit returns to ZERO is inversely proportional to the amount of light falling on that bit.

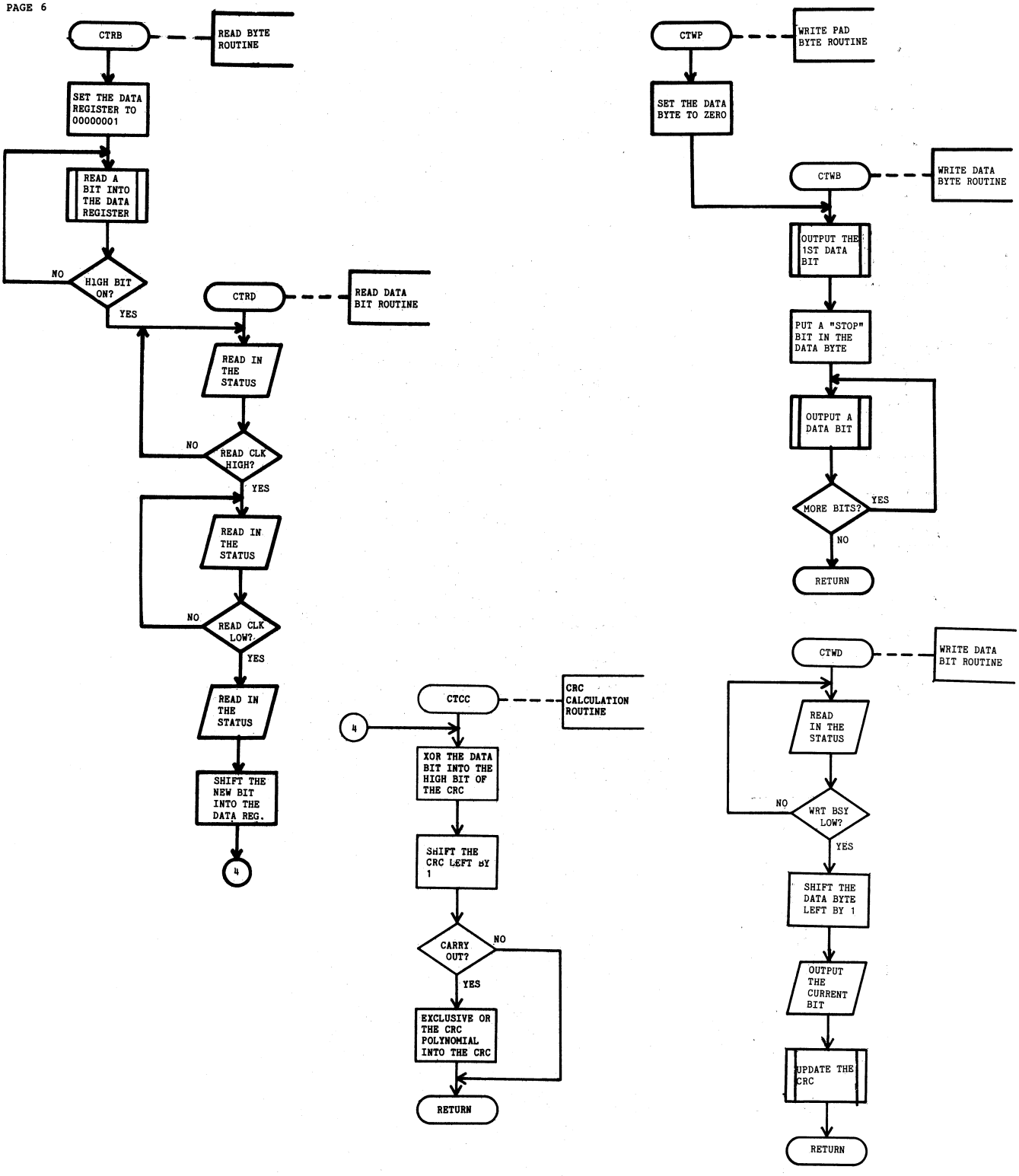
Replacing the cap on the memory requires clean room conditions and a controlled atmosphere. A kit including a PC board and all parts is available however from H. Garland, 26655 Laurel Lane, Los Altos, CA 94022 for $55.00.

I wonder if one of the new 4K RAM's could be used for a 64 X 64 image sensor?


CTRR
CASSETTE TAPE READ RECORD ROUTINE
TURN ON THE MOTOR
ZERO THE COMPARE REGISTER
READ A BIT INTO THE COMPARE REGISTER
NO
DATA ID?
YES
ZERO THE CRC
READ IN THE RECORD LENGTH
REC LEN = 0?
NO
YES
1
2
1
READ IN A DATA BYTE
STORE THE BYTE IN THE NEXT BUFFER POSITION
INCR. THE BUFFER PTR. & DECR. THE RECORD LEN.
NO
REC LEN = 0?
YES
2
READ IN THE CRC
TURN OFF THE MOTOR
SET THE RETURN CODE FROM THE FINAL CRC
RETURN


CTIPL
CASSETTE TAPE IPL ROUTINE
(HL) <- INITIAL LOAD ADDRESS
READ IN A RECORD FROM UNIT ZERO
I/O ERROR?
YES
ERROR HALT
NO
NO
EOF?
YES
EXIT


CTWR
CASSETTE TAPE WRITE RECORD ROUTINE
OUTPUT THE UNIT SELECT
READ IN THE STATUS
YES
MOTOR BUSY?
NO
TURN ON THE MOTOR
READ IN THE STATUS
YES
MOTOR BUSY?
NO
OUTPUT THE LEADING PAD
OUTPUT THE DATA ID
ZERO THE CRC
SET THE LOOP COUNT FROM THE RECORD LENGTH + 1
SET THE CURRENT BYTE FROM THE RECORD LEN.
3
3
OUTPUT THE CURRENT BYTE
DECREMENT THE LOOP COUNT
LOOP CNT=0?
YES
NO
SET THE BYTE FROM MEMORY
INCREMENT THE BUFFER POINTER
OUTPUT THE CRC
OUTPUT THE TRAILING PAD
TURN OFF THE MOTOR
RETURN


CTRB
READ BYTE ROUTINE
SET THE DATA REGISTER TO 00000001
READ A BIT INTO THE DATA REGISTER
HIGH BIT ON?
NO
YES
CTRD
READ DATA BIT ROUTINE
READ IN THE STATUS
READ CLK HIGH?
NO
YES
READ IN THE STATUS
READ CLK LOW?
NO
YES
READ IN THE STATUS
SHIFT THE NEW BIT INTO THE DATA REG.
4
CTCC
CRC CALCULATION ROUTINE
4
XOR THE DATA BIT INTO THE HIGH BIT OF THE CRC
SHIFT THE CRC LEFT BY 1
CARRY OUT?
NO
YES
EXCLUSIVE OR THE CRC POLYNOMIAL INTO THE CRC
RETURN
CTWP
WRITE PAD BYTE ROUTINE
SET THE DATA BYTE TO ZERO
CTWB
WRITE DATA BYTE ROUTINE
OUTPUT THE 1ST DATA BIT
PUT A "STOP" BIT IN THE DATA BYTE
OUTPUT A DATA BIT
MORE BITS?
YES
NO
RETURN
CTWD
WRITE DATA BIT ROUTINE
READ IN THE STATUS
NO
WRT BSY LOW?
YES
SHIFT THE DATA BYTE LEFT BY 1
OUTPUT THE CURRENT BIT
UPDATE THE CRC
RETURN

```
LOCN     CODE      SOURCE STATEMENT

                   *
                   *     TCH STANDARD CASSETTE TAPE ROM FOR THE 8008
                   *     ---
                   *
                   *     PROGRAM BY RICHARD M. SMITH
                   *


                   *     I/O DEVICE ADDRESSES

000016             CTCTL  EQU 016B        CASSETTE TAPE CONTROL REGISTER
000001             CTSTS  EQU 001B        CASSETTE TAPE STATUS REGISTER


                   *     CONTROL REGISTER BITS

000020             CTUS   EQU 020B        UNIT SELECT
000010             CTMTC  EQU 010B        MOTOR CONTROL
000004             CTWRM  EQU 004B        WRITE MODE
000002             CTWRE  EQU 002B        WRITE ENABLE
000001             CTWRD  EQU 001B        WRITE DATA


                   *     STATUS REGISTER BITS

000010             CTMTB  EQU 010B        MOTOR OPERATION BUSY
000004             CTWRB  EQU 004B        WRITE BUSY
000002             CTRDC  EQU 002B        READ CLOCK
000001             CTRDD  EQU 001B        READ DATA


                   *     UNIT NUMBERS

000000             CTUN0  EQU 000B        UNIT 0
000020             CTUN1  EQU CTUS        UNIT 1


                   *     MISCELLANEOUS EQUATES

100005             CTCRC  EQU 100005B     CRC POLYNOMIAL (CRC 16)
104657             CTDID  EQU 104657B     DATA ID
000000             CTLDA  EQU 000000B     LOAD ADDRESS FOR IPL


                   *     ORIGIN SET

037400                    ORG 037400B     ORG AT THE LAST PAGE OF MEMORY.


                   *     JUMP VECTOR

037400  104 035 077       JMP CTRR        READ RECORD ROUTINE
037403  104 213 077       JMP CTWR        WRITE RECORD ROUTINE


                   *     CASSETTE TAPE CONTROL ROUTINE

037406  250       CTCN    XRA             RESET THE CASSETTE TAPE INTERFACE.
037407  135               OUT CTCTL
037410  007               RET             RETURN.


                   *     CASSETTE TAPE IPL ROUTINE

037411  056 000   CTIPL   LHI H(CTLDA)     SET THE LOAD ADDRESS IN HL.
037413  066 000           LLI L(CTLDA)
037415  006 000   CTIP1   LAI CTUN0        READ A RECORD FROM CASSETTE TAPE
037417  106 035 077       CAL CTRR           UNIT 0.
037422  110 022 077       JFZ *           LOOP FOREVER IF AN I/O ERROR.
037425  301               LAB             LOOP IF THE RECORD IS NOT AN END OF
037426  260               ORA                FILE RECORD.
037427  110 015 077       JFZ CTIP1
037432  104 000 000       JMP CTLDA       BRANCH TO THE LOADED PROGRAM.


                   *     CASSETTE TAPE READ RECORD ROUTINE

037435  064 010   CTRR    ORI CTMTC       TURN ON THE MOTOR ON THE SPECIFIED
037437  135               OUT CTCTL          UNIT.
037440  016 000           LBI 0           ZERO THE COMPARE REGISTER IN BC.
037442  321               LCB
037443  302       CTRR1   LAC             READ THE NEXT BIT INTO THE COMPARE
037444  200               ADA                REGISTER
037445  301               LAB
037446  022               RAL
037447  310               LBA
037450  106 145 077       CAL CTRD
037453  301               LAB             LOOP IF NOT THE DATA ID. (ALSO
037454  054 211           XRI H(CTDID)       ZERO THE CRC IN DE IF THE DATA
037456  330               LDA                ID.)
037457  340               LEA
037460  302               LAC
037461  054 257           XRI L(CTDID)
037463  263               ORD
037464  110 043 077       JFZ CTRR1
037467  106 133 077       CAL CTRB        READ IN THE RECORD LENGTH AND SAVE
037472  312               LBC                IT IN B.
037473  301               LAB             LOAD IT INTO A AND BRANCH IF IT IS
037474  260               ORA                ZERO.
037475  150 117 077       JTZ CTRR3
037500  370       CTRR2   LMA             SAVE THE RECORD LENGTH IN MEMORY.
037501  106 133 077       CAL CTRB        READ IN A DATA BYTE AND SWAP IT
037504  307               LAM                WITH THE RECORD LENGTH IN
037505  372               LMC                MEMORY.
037506  060               INL             BUMP UP HL BY 1.
037507  110 113 077       JFZ *+4
037512  050               INH
037513  011               DCB             DECREMENT B AND LOOP IF IT IS NOT
037514  110 100 077       JFZ CTRR2          ZERO.
037517  310       CTRR3   LBA             SAVE THE RECORD LENGTH IN B.
037520  106 133 077       CAL CTRB        READ IN THE CRC.
037523  106 133 077       CAL CTRB


                   *     MOTOR OFF ROUTINE

037526  250       CTMOF   XRA             RESET THE CASSETTE TAPE INTERFACE.
037527  135               OUT CTCTL
037530  303               LAD             SET THE RETURN CODE FROM THE CRC.
037531  264               ORE
037532  007               RET             RETURN.


                   *     READ BYTE ROUTINE

037533  026 001   CTRB    LCI 001B        SET C TO 001B.
037535  106 145 077 CTRB1 CAL CTRD        READ IN 7 DATA BITS.
037540  302               LAC
037541  260               ORA
037542  120 135 077       JFS CTRB1


                   *     READ DATA BIT ROUTINE

037545  103       CTRD    INP CTSTS       WAIT FOR THE READ CLOCK TO GO HIGH.
037546  044 002           NDI CTRDC
037550  110 145 077       JFZ CTRD
037553  103       CTRD1   INP CTSTS       WAIT FOR THE READ CLOCK TO GO LOW.
037554  044 002           NDI CTRDC
037556  150 153 077       JTZ CTRD1
037561  103               INP CTSTS       ISOLATE THE DATA BIT IN THE CARRY.
037562  032               RAR
037563  302               LAC             ROTATE THE BIT INTO C.
037564  022               RAL
037565  320               LCA
037566  012               RRC             PUT THE BIT IN THE HIGH BIT OF A.


                   *     CRC CALCULATION ROUTINE

037567  044 200   CTCC    NDI 200B        EXCLUSIVE OR THE DATA BIT INTO THE
037571  253               XRD                HIGH ORDER BIT OF THE CRC.
037572  330               LDA
037573  304               LAE             SHIFT THE CRC LEFT BY 1.
037574  200               ADA
037575  340               LEA
037576  303               LAD
037577  022               RAL
037600  330               LDA
037601  003               RFC             RETURN IF NO CARRY OUT.
037602  303               LAD             EXCLUSIVE OR THE CRC POLYNOMIAL INTO
037603  054 200           XRI H(CTCRC)       THE CRC.
037605  330               LDA
037606  304               LAE
037607  054 005           XRI L(CTCRC)
037611  340               LEA
037612  007               RET             RETURN.


                   *     WRITE RECORD ROUTINE

037613  135       CTWR    OUT CTCTL       OUTPUT THE UNIT SELECT AND SAVE IT
037614  320               LCA                IN C.
037615  103       CTWR1   INP CTSTS       WAIT FOR THE MOTOR TO STOP.
037616  044 010           NDI CTMTB
037620  110 215 077       JFZ CTWR1
037623  302               LAC             TURN ON THE MOTOR.
037624  064 010           ORI CTMTC
037626  135               OUT CTCTL
037627  103       CTWR2   INP CTSTS       WAIT FOR THE MOTOR TO COME UP TO
037630  044 010           NDI CTMTB          SPEED.
037632  110 227 077       JFZ CTWR2
037635  106 332 077       CAL CTWP        OUTPUT THE 4 BYTES OF LEADING PAD.
037640  106 332 077       CAL CTWP
037643  106 332 077       CAL CTWP
037646  106 332 077       CAL CTWP
037651  026 211           LCI H(CTDID)    OUTPUT THE DATA ID.
037653  106 334 077       CAL CTWB
037656  026 257           LCI L(CTDID)
037660  106 334 077       CAL CTWB
037663  036 000           LDI 0           ZERO THE CRC IN DE.
037665  343               LED
037666  321               LCB             LOAD THE RECORD LENGTH INTO C, BUMP
037667  010               INB                IT UP BY 1 IN B, AND BRANCH INTO
037670  104 301 077       JMP CTWR4          THE WRITE LOOP.
037673  327       CTWR3   LCM             LOAD THE NEXT DATA BYTE INTO C.
037674  060               INL
037675  110 301 077       JFZ *+4
037700  050               INH
037701  106 334 077 CTWR4 CAL CTWB        OUTPUT THE CURRENT BYTE.
037704  011               DCB             DECREMENT B AND LOOP IF IT IS NOT
037705  110 273 077       JFZ CTWR3          ZERO.
037710  314               LBE             OUTPUT THE CRC.
```

```
037711  323              LCD
037712  106 334 077      CAL CTWB
037715  321              LCB
037716  106 334 077      CAL CTWB
037721  106 332 077      CAL CTWP        OUTPUT THE TRAILING PAD OF ZEROS.
037724  106 332 077      CAL CTWP
037727  104 126 077      JMP CTMOF       GO TURN OFF THE MOTOR.


                *       WRITE ZERO PAD BYTE ROUTINE

037732  026 000      CTWP LCI 0           ZERO C.


                *       WRITE BYTE ROUTINE

037734  106 351 077 CTWB  CAL CTWD       OUTPUT THE 1ST DATA BIT.
037737  020              INC             PUT IN A "STOP" BIT IN C.
037740  106 351 077 CTWB1 CAL CTWD       OUTPUT BITS UNTIL A STOP BIT IS THE
037743  302              LAC                ONLY BIT LEFT.
037744  200              ADA
037745  110 340 077      JFZ CTWB1
037750  007              RET             RETURN.


                *       WRITE DATA BIT ROUTINE

037751  103      CTWD    INP CTSTS       WAIT FOR THE WRITE BUSY TO GO LOW.
037752  044 004          NDI CTWRB
037754  110 351 077      JFZ CTWD
037757  302              LAC             SHIFT THE BYTE IN C LEFT BY 1.
037760  200              ADA
037761  320              LCA
037762  103              INP CTSTS       ISOLATE THE UNIT SELECT AND CURRENT
037763  032              RAR                DATA BIT IN A.
037764  002              RLC
037765  044 021          NDI CTUS+CTWRD
037767  064 016          ORI CTWRM+CTWRE+CTMTC OUTPUT THE BIT.
037771  135              OUT CTCTL
037772  012              RRC             ISOLATE THE DATA BIT IN THE HIGH BI
                *                           OF A.
037773  104 167 077      JMP CTCC        GO UPDATE THE CRC.

037776                   END
```

```
                ;       TCH STANDARD CASSETTE TAPE ROM FOR THE 8080
                ;       ---
                ;
                ;       PROGRAM BY RICHARD M. SMITH
                ;


                ;       I/O DEVICE ADDRESSES

000200          CTCTL   EQU  200Q       ; CASSETTE TAPE CONTROL REGISTER
000200          CTSTS   EQU  200Q       ; CASSETTE TAPE STATUS REGISTER


                ;       CONTROL REGISTER BITS

000020          CTUS    EQU  020Q       ; UNIT SELECT
000010          CTMTC   EQU  010Q       ; MOTOR CONTROL
000004          CTWRM   EQU  004Q       ; WRITE MODE
000002          CTWRE   EQU  002Q       ; WRITE ENABLE
000001          CTWRD   EQU  001Q       ; WRITE DATA


                ;       STATUS REGISTER BITS

000010          CTMTB   EQU  010Q       ; MOTOR BUSY
000004          CTWRB   EQU  004Q       ; WRITE BUSY
000002          CTRDC   EQU  002Q       ; READ CLOCK
000001          CTRDD   EQU  001Q       ; READ DATA


                ;       UNIT NUMBERS

000000          CTUN0   EQU  000Q       ; UNIT 0
000020          CTUN1   EQU  CTUS       ; UNIT 1


                ;       MISCELLANEOUS EQUATES

000200          CTHCRC  EQU  200Q       ; CRC POLYNOMIAL (CRC 16)
000005          CTLCRC  EQU  005Q       ;
000211          CTHDID  EQU  211Q       ; DATA ID
000257          CTLDID  EQU  257Q       ;
000000          CTLDA   EQU  000000Q    ; INITIAL LOAD ADDRESS FOR IPL
000400          CTSTE   EQU  000400Q    ; END OF STACK FOR IPL


                ;       ORIGIN SET

177400                  ORG  177400Q    ; SET THE ORIGIN TO THE LAST
                                        ;    PAGE OF MEMORY.


                ;       JUMP VECTOR

177400  303 042 377     JMP  CTRR       ; READ RECORD
177403  303 135 377     JMP  CTWR       ; WRITE RECORD
177406  303 036 377     JMP  CTCN       ; CONTROL ROUTINE


                ;       CASSETTE TAPE IPL ROUTINE

177411  041 000 000 CTIPL:  LXI  HL,CTLDA    ; INITIALIZE THE LOAD ADDRESS
177414  061 000 001          LXI  SP,CTSTE    ;    AND THE STACK POINTER.
177417  345                  PUSH HL          ; SAVE THE LOAD ADDRESS.
177420  076 000      CTIP1:  MVI  A,CTUN0     ; READ A RECORD FROM CASSETTE
177422  315 042 377          CALL CTRR        ;    TAPE UNIT 0.
177425  302 025 377          JNZ  $           ; LOOP FOREVER IF I/O ERROR.
177430  170                  MOV  A,B         ; LOOP IF THE RECORD IS NOT AN
177431  267                  ORA  A           ;    END OF FILE RECORD.
177432  302 020 377          JNZ  CTIP1       ;
177435  311                  RET              ; BRANCH TO THE LOADED PROGRAM.


                ;       CASSETTE TAPE CONTROL ROUTINE

177436  257      CTCN:   XRA  A          ; CLEAR THE CASSETTE TAPE
177437  323 200          OUT  CTCTL      ;    INTERFACE.
177441  311              RET             ; RETURN.


                ;       CASSETTE TAPE READ RECORD ROUTINE

177442  366 010  CTRR:   ORI  CTMTC      ; TURN THE SPECIFIED UNIT'S
177444  323 200          OUT  CTCTL      ;    MOTOR ON.
177446  001 000 000      LXI  BC,0       ; ZERO THE COMPARE REGISTER IN
                                        ;    BC.
177451  171      CTRR1:  MOV  A,C        ; READ THE NEXT DATA BIT INTO
177452  207              ADD  A          ;    THE COMPARE REGISTER.
177453  170              MOV  A,B        ;
177454  027              RAL             ;
177455  107              MOV  B,A        ;
177456  315 255 377      CALL CTRD       ;
177461  170          *   MOV  A,B        ; LOOP IF THE COMPARE REGISTER
177462  356 211          XRI  CTHDID     ;    DOES NOT EQUAL THE DATA
177464  127              MOV  D,A        ;    ID. (ALSO ZERO THE CRC IN
177465  137              MOV  E,A        ;    IN DE IF THE DATA ID.)
177466  171              MOV  A,C        ;
177467  356 257          XRI  CTLDID     ;
177471  262              ORA  D          ;
177472  302 051 377      JNZ  CTRR1      ;
177475  315 243 377      CALL CTRB       ; READ IN THE RECORD LENGTH AND
177500  101              MOV  B,C        ;    SAVE IT IN B AND ON THE
177501  305              PUSH B          ;    STACK.
177502  171              MOV  A,C        ; BRANCH IF IT IS ZERO.
177503  267              ORA  A          ;
177504  312 120 377      JZ   CTRR3      ;
177507  315 243 377 CTRR2:  CALL CTRB    ; READ AND STORE THE NEXT DATA
177512  167              MOV  M,A        ;    BYTE.
177513  043              INX  HL         ;
177514  005              DCR  B          ; DECREMENT B AND LOOP IF IT
177515  302 042 377      JNZ  CTRR2      ;    IS NOT ZERO.
177520  315 243 377 CTRR3:  CALL CTRB    ; READ IN THE 2 CRC BYTES.
177523  315 243 377      CALL CTRB       ;
177526  301              POP  BC         ; POP THE RECORD LENGTH INTO
                                        ;    B.


                ;       MOTOR OFF ROUTINE

177527  257      CTMOF:  XRA  A          ; RESET THE CASSETTE TAPE
177530  323 200          OUT  CTCTL      ;    INTERFACE.
177532  172              MOV  A,D        ; SET THE RETURN CODE FROM THE
177533  263              ORA  E          ;    FINAL CRC.
177534  311              RET             ; RETURN.


                ;       CASSETTE TAPE WRITE RECORD ROUTINE

177535  117      CTWR:   MOV  C,A        ; OUTPUT THE UNIT SELECT AND
177536  323 200          OUT  CTCTL      ;    SAVE IT IN C.
177540  333 200  CTWR1:  IN   CTSTS      ; WAIT FOR THE MOTOR TO STOP.
177542  346 010          ANI  CTMTB      ;
177544  302 140 377      JNZ  CTWR1      ;
177547  171              MOV  A,C        ; TURN ON THE MOTOR.
177550  366 010          ORI  CTMTC      ;
177552  323 200          OUT  CTCTL      ;
177554  333 200  CTWR2:  IN   CTSTS      ; WAIT FOR THE MOTOR TO COME
177556  346 010          ANI  CTMTB      ;    UP TO SPEED.
177560  302 154 377      JNZ  CTWR2      ;
177563  315 323 377      CALL CTWP2      ; OUTPUT THE 4 BYTES OF
177566  315 323 377      CALL CTWP2      ;    LEADING PAD.
177571  016 211          MVI  C,CTHDID   ; OUTPUT THE DATA ID.
177573  315 330 377      CALL CTWB       ;
177576  016 257          MVI  C,CTLDID   ;
177600  315 330 377      CALL CTWB       ;
177603  021 000 000      LXI  DE,0       ; ZERO THE CRC IN DE.
```

```
177606  110                     MOV  C,B            ; LOAD THE RECORD LENGTH INTO
177607  004                     INR  B              ;    C, BUMP IT UP BY 1 IN B,
177610  303 215 377             JMP  CTWR4          ;    AND BRANCH INTO THE WRITE
                                                    ;    LOOP.
177613  116             CTWR3:  MOV  C,M            ; LOAD THE NEXT DATA BYTE INTO
177614  043                     INX  HL             ;    C.
177615  315 330 377 CTWR4:     CALL CTWB           ; OUTPUT THE CURRENT BYTE.
177620  005                     DCR  B              ; DECREMENT B AND LOOP IF IT
177621  302 213 377             JNZ  CTWR3          ;    IS NOT ZERO.
177624  103                     MOV  B,E            ; OUTPUT THE CRC.
177625  112                     MOV  C,D            ;
177626  315 330 377             CALL CTWB           ;
177631  110                     MOV  C,B            ;
177632  315 330 377             CALL CTWB           ;
177635  315 323 377             CALL CTWP2          ; OUTPUT THE TRAILING PAD.
177640  303 127 377             JMP  CTMOF          ; GO TURN OFF THE MOTOR.


                                READ BYTE ROUTINE

177643  016 001         CTRB:   MVI  C,001Q         ; SET C TO 001Q.
177645  315 255 377 CTRB1:     CALL CTRD           ; READ IN 7 DATA BITS.
177650  171                     MOV  A,C            ;
177651  267                     ORA  A              ;
177652  362 245 377             JP   CTRB1          ;


                    ;           READ DATA BIT ROUTINE

177655  333 200         CTRD:   IN   CTSTS          ; WAIT FOR THE READ CLOCK TO
177657  346 002                 ANI  CTRDC          ;    GO HIGH.
177661  302 255 377             JNZ  CTRD           ;
177664  333 200         CTRD1:  IN   CTSTS          ; WAIT FOR THE READ CLOCK TO
177666  346 002                 ANI  CTRDC          ;    GO LOW.
177670  312 264 377             JZ   CTRD1          ;
177673  333 200                 IN   CTSTS          ; ISOLATE THE DATA BIT IN THE
177675  037                     RAR                 ;    CARRY.
177676  171                     MOV  A,C            ; SHIFT THE BIT INTO C.
177677  217                     ADC  A              ;
177700  117                     MOV  C,A            ;
177701  017                     RRC                 ; PUT THE NEW BIT IN THE HIGH
                                                    ;    BIT OF A FOR THE CRC
                                                    ;    UPDATE.


                    ;           CRC CALCULATION ROUTINE

177702  346 200         CTCC:   ANI  200Q           ; EXCLUSIVE OR THE DATA BIT
177704  252                     XRA  D              ;    INTO THE HIGH ORDER BIT
177705  127                     MOV  D,A            ;    OF THE CRC.
177706  353                     XCHG                ; SHIFT THE CRC LEFT BY 1.
177707  051                     DAD  HL             ;
177710  353                     XCHG                ;
177711  320                     RNC                 ; RETURN IF NO CARRY OUT.
177712  172                     MOV  A,D            ; EXCLUSIVE OR THE CRC
177713  356 200                 XRI  CTHCRC         ;    POLYNOMIAL INTO THE CRC.
177715  127                     MOV  D,A            ;
177716  173                     MOV  A,E            ;
177717  356 005                 XRI  CTLCRC         ;
177721  137                     MOV  E,A            ;
177722  311                     RET                 ; RETURN.


                    ;           WRITE ZERO PAD BYTE ROUTINE

177723  315 326 377 CTWP2:    CALL CTWP           ; OUTPUT THE 1ST PAD BYTE.
177726  016 000         CTWP:   MVI  C,0            ; ZERO C.


                    ;           WRITE BYTE ROUTINE

177730  315 345 377 CTWB:     CALL CTWD           ; OUTPUT THE 1ST DATA BIT.
177733  014                     INR  C              ; PUT A "STOP" BIT IN C.
177734  315 345 377 CTWB1:    CALL CTWD           ; OUTPUT BITS UNTIL THE STOP
177737  171                     MOV  A,C            ;   BIT IS THE ONLY BIT
177740  207                     ADD  A              ;   LEFT.
177741  302 334 377             JNZ  CTWB1          ;
177744  311                     RET                 ; RETURN.


                    ;           WRITE DATA BIT ROUTINE

177745  333 200         CTWD:   IN   CTSTS          ; WAIT FOR THE WRITE BUSY TO
177747  346 004                 ANI  CTWRB          ;   GO LOW.
177751  302 345 377             JNZ  CTWD           ;
177754  171                     MOV  A,C            ; SHIFT THE BYTE IN C LEFT BY
177755  207                     ADD  A              ;    1.
177756  117                     MOV  C,A            ;
177757  333 200                 IN   CTSTS          ; ISOLATE THE UNIT SELECT AND
177761  037                     RAR                 ;
177762  007                     RLC                 ;
177763  346 021                 ANI  CTUS+CTWRD     ;
177765  366 016                 ORI  CTWRM+CTWRE+CTMTC ; OUTPUT THE BIT.
177767  323 200                 OUT  CTCTL          ;
177771  017                     RRC                 ; POSITION THE CURRENT DATA
177772  303 302 377             JMP  CTCC           ;    BIT IN THE HIGH BIT OF A
                                                    ;    AND GO UPDATE THE CRC.


                                END                 ;
```

COMPUTER PING-PONG by Jim Parker

One of the more entertaining ways to use a graphics display is to play ping-pong on it. This program simulates a Ping-Pong game in a way very similar to the television version that has recently become popular. The game allows two players to move "paddles" up and down so as to "hit" a ball back and forth across the screen. The first serve is begun by pushing a button. A miss scores a point for the other player and automatically causes another serve to the player who missed last. The first player to score eleven points wins the game. The first game always begins with a serve to the player on the right. After that, the following games begin with a serve to the side that lost the last game.

The program runs on an 8008-1 system with at least 3 pages of RAM and Hal Chamberlin's graphics display and pot controls. The plans for these I/O devices were published in Volume I, issue numbers 1, 2, 3, and 4 of TCH. As an option, a speaker can be used to provide a pop sound when the ball and paddle collide. Readers with an 8080 CPU should also be able to use this program although it may need to be slowed down a little. More will be mentioned on this later.

Motion of the ball is done by showing successive "frames" of the ball shifted slightly over, much like a movie. An interesting feature of this game is that the ball can travel with 32 different velocities, 16 towards the left and 16 towards the right. A basic knowledge of vectors is required to understand how this is done. Simply put, the velocity (which specifies both speed and direction) can be broken up into X and Y (horizontal and vertical) components. The square root of (X**2+Y**2) determines the speed while the arc tangent of (Y/X) determines the direction angle with respect to the horizontal axis. For example, if the ball was displaced 1 unit up and one unit to the right every frame, it would appear to move at a speed of v2 units per frame at an angle of 45o. A problem arises when you try to keep the speed below 2 units per frame (as was done in this program) and still provide a large variety of direction angles. If you use only integers, the possible (X,Y) component velocity combinations are soon exhausted. For example, if you displace the ball 1 unit right and 2 units up every frame, the speed will be v5 or greater than 2 units per frame. To solve this problem, double precision (16 bit) arithmetic was used. The most significant 8 bits specify the ball's X or Y position on the display while the least significant 8 bits act as the numerator of a fraction with the implied denominator of 256. This two byte combination can be treated as if the whole thing was multiplied by 256 and converted to a 16 bit integer with an implied division by 256 to restore it back to normal. Thus the DBLADD routine can perform a simple 16 bit integer addition to compute the next ball position. Note that multiplication or division by 256 is accomplished by shifting the 16 bit integer left or right 8 bits with respect to the binary point. Thus the 16 bit ball position is displayed by regarding only the most significant byte. This has the same effect as dividing the 16 bit integer by 256 (shifting it right with respect to the binary point) and ignoring any fractional part of the quotient.

The use of fractions allow a much greater range of velocity (X,Y) components between 0 and 2 so that a wide range of angles are possible all with a speed of 2 units per frame. As mentioned before, the ball's next position is computed by adding the 16 bit X position to the 16 bit X displacement, the 16 bit Y position to the 16 bit Y displacement, and regarding only the most significant bytes of these two sums. Due to the fact that the fractional part of the ball's position is not displayed, there is a slightly ragged appearance in the ball's path across the screen although it is certainly not very noticable. To avoid complex subroutines to compute square roots and arc tangents, a simple velocity table is provided with 16 different velocities. Fifteen of these move the ball at a speed of 2 units per frame and one moves the ball horizontally at 1 unit per frame. All of the velocities move the ball towards the right. To move the ball left, the X velocity component is negated. Note that two's complement arithmetic is used to handle negative numbers. I like to negate numbers simply by subtracting them from zero which is represented the same as 2**16 or 65536 for 16 bit numbers. Thus negative 100 would be represented as 65536-100=65436. To prove that this works, try adding -100 to +300 using 16 bit unsigned integers. Well, 65436+300= 65736 but that number requires 17 bits to represent. Since the most significant bit will be lost and that bit represents 2**16=65536, we must subtract it from the above addition. Thus 65736-65536=200 which is the correct result. If you are familiar with modular arithmetic, you can think of this as MOD 2**16 arithmetic. There are other ways to deal with two's complement arithmetic but I prefer the above method.

In the velocity table, all values were computed with the help of a pocket calculator. They are listed in the order of least significant X, most significnat X, least significant Y, and most signficant Y. Multiply the most significant value by 256 and add that to the least significant value to get the numerator of a fraction in the range of 0/256 to 65535/256. If the number is greater than or equal to 2**15=32768, its most significant bit (called the sign bit) will be ONE and the number will be negative. Convert these to a positive number by subtracting them from 65536 as mentioned before. You can then confirm for yourself the ball velocities by dividing the numbers by 256 and plugging them into the square root and arc tangent formulas given previously.

When the ball is served or when the ball is hit, a random number generator indexes the ANGLE table so as to project the ball in an unpredictable manner. Bouncing the ball off the top or bottom, however, is simulated by simply negating the ball's Y velocity component. Basic physics predicts this result assuming there is no energy lost in the bounce.

The power of flow charting is well illustrated here for without the appending chart, it would be very difficult to explain or even write this program. By glancing over it, you should get a general idea of how the program runs. Please refer to the Volume 1, Number 4 issue of TCH for a detailed description of the pot controls and switch device used. Before switch 1 is pushed, the program merely draws the score from the last game, reads two pot controls and positions the two paddles accordingly, and draws the board (a square that defines the outer boundary). All this is done by calling the DRAW subroutine. If you are unfamiliar with how the CHAR, RVCD, and GRAPH subroutines work, you should refer to the example software programs listed in the first four issues of TCH. Basically the DRAW subroutine does its job by filling in the correct parameters for the other three subroutines just mentioned.

When switch 1 is pushed, the program begins a new game. It resets the score, picks a starting position for the serve somewhere near the middle of the board, and picks a serve velocity at random from the ANGLE table. The program is then ready to enter the major loop headed by the name NXPOS. The loop is passed through everytime the next position of the ball is displayed and examined. There are four tests made on the ball's position. Due to the wrap-around effect of the board the test for end zone can be reduced to one test. Similarly, so can the test for the ball touching the top or botton. It works like this:

There are two Y coordinate positions at the top and two at the bottom that indicate when the ball has reached the edge. Thus if the ball's most significant Y position is 126, 127, 128 (=-128), or 129 (=-127), then the program needs to change the ball's Y direction. This is tested by subtracting 126 from the ball's Y position. If the ball is at the edge, the resulting difference will be 0,1,2, or 3. A CPI 4 is performed on the result. This sets the flag flip-flop the same way as if 4 had been subtracted from the number. Only if the number is 0,1,2, or 3 will the carry flag (or the borrow flag in this case) be set by subtracting 4. A similar technique is used for testing the other zones. Some readers may find this technique useful in other programs for testing if a number is in a certain range. Note that the X position of the ball is not reloaded after every test. This can be done if you keep track of the amount you subtracted from the number for the previous test. Thus 104 is subtracted first to see if the ball's X position is 104 and then 44 is subtracted from that to see if the X position is 148 (=-108) or 149 (=-107).

The result of the above tests determine what the program does next. If the ball is not touching any special boundaries then the program jumps back to NXPOS and displays the next position of the ball. As long as the ball doesn't reach the paddle zones or end zone, the program will continue to loop through NXPOS, showing the ball shifted over slightly each time. This causes the ball to appear in motion. When the ball reaches a paddle zone, the program jumps to a routine that tests the position of the corresponding paddle. The paddle is 16 units high so a test is made to see if the ball's Y position minus the Y position of the bottom of the paddle is less than 16. If it is, the direction of the ball is changed and a new ball velocity is picked at random from the ANGLE table. The ball must also be cleared from the paddle zone or an interresting bug will occur. If the ball travels into the zone faster than it leaves, it will not be out of the zone on the next time around the loop. The program would the change the direction of the ball again. The ball may bounce back and forth in this zone many times until it finally escaped. To prevent this from happening, the program actually relocates the ball just outside of the paddle zone when it is hit. Also a speaker is popped. This is done by executing an input instruction to the keyboard (which has the speaker for feedback while typing) on TCH's demo system. If the ball misses the paddle then the program allows the ball to keep traveling in the same direction.

When someone misses the ball, the ball will reach the end zone. This causes the program to test the ball's direction to see who missed, and then to increment the score accordingly. Score keeping is handled somewhat unconventionally in this program in order to save on memory and program complexity. Each character can be

drawn using the minor deflection system with 16 bytes or less of data. Because of this, the data for the characters begin at memory locations separated by exactly 16 bytes per character. Since the table of characters for the score is arranged in ascending order, the score counter needs only to keep track of the low address of the character and have that value incremented by 16 every time the player scores another point. Note that ten and eleven are treated as if they were single characters. The program knows when the game has ended by testing for when the score counter addresses the character eleven. If the game is not over, the program will delay a few seconds and then jump back to the part of the program that initiates a serve. If the game is over, the program will jump back to the beginning, displaying the final score and waiting for switch 1 to be pushed to begin another game.

Readers with an 8080 machine and an assembler should have little problem using this program. Since the 8008 instruction set is a subset of the 8080 instruction set, you should be able to translate every instruction on a one to one basis. The real problem in running the program is slowing it down. If you are driving the graphics display as published in the first three issues of TCH, your main concern should be in slowing down the subroutines that draw on the display since most of the execution time is spent driving the display. This can be done by adding inefficient NOOP's (such as an even number of XTHL's) in the critical loops. If the speed of the 8080 can match the speed of Hal Chamberlin's display, then the program should be slow enough for two human players. Caution should also be used in the D/A routines for the pot controls to avoid sampling the comparators before they have settled. If you have both a fast machine and a faster graphics display, then you can either introduce delay loops, reduce the size of the entries in the ball velocity table, or make the paddles wider.

If you have a slower machine such as an 8008 instead of an 8008-1, you can improve the efficency of the display subroutines, make the paddles smaller, or increase the size of the entries in the ball velocity table. If you choose the last suggestion, you must also increase the width of the paddle zones or the ball may penetrate right through the paddles.

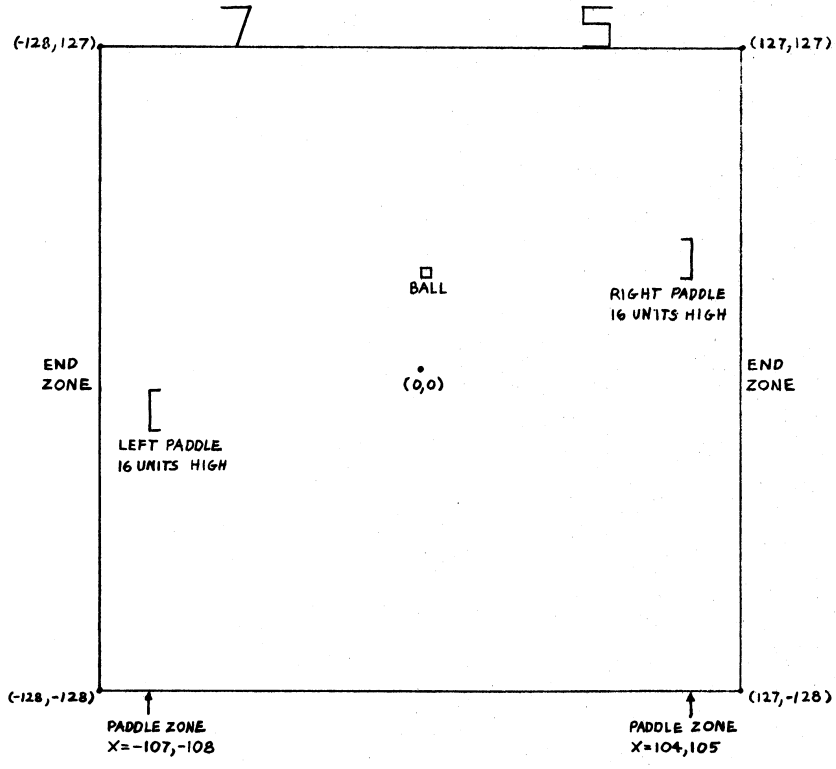
As usual, there are a large number of variations that can be tried. A simple variation is to reduce the thickness of various boundaries so as to make them penetratable 50% of the time. This is guaranteed to challenge even the most skilled players. To make even a more realistic game, the velocity of the ball after being hit could be determined by how quickly the paddle changed position prior to being hit. Even fancier versions would be a program that would occasionally make the ball loop-the-loop. A game for 1, 3, or 4 players could also be developed. In the case of a single player, provide a goal on the opposing side so that the player could hit the ball in. Hopefully this program will open the doorway to a whole series of simulation game programs. Hobbyists are encouraged to make every effort to build their own graphics display and enjoy the demo programs that are published by TCH.

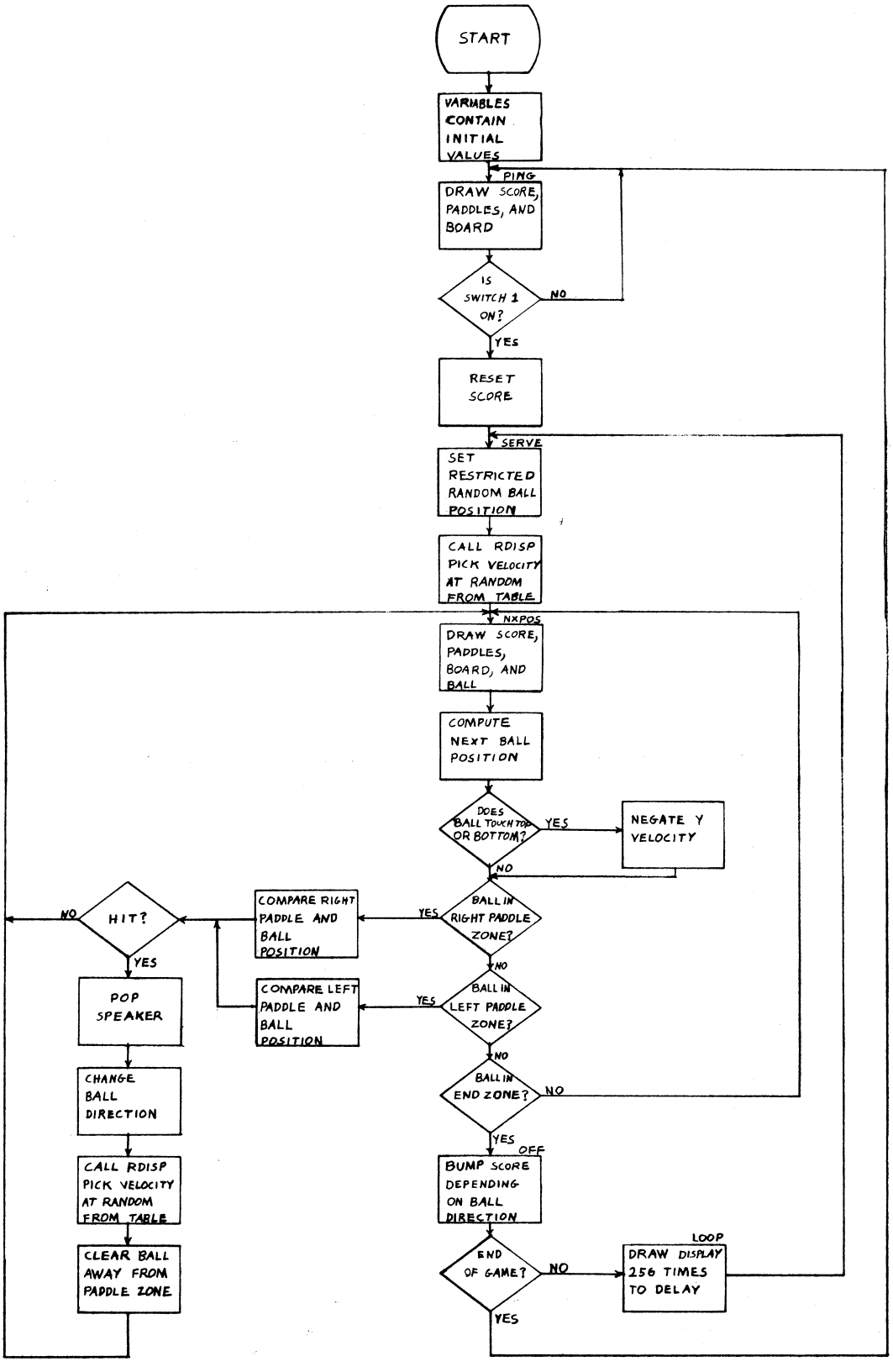
PING-PONG PROGRAM LISTING

```
LOCN    CODE        SOURCE STATEMENT

000000                      ORG   0

000006              GINP    EQU   6
000010              XMOV    EQU   10B
000011              YMOV    EQU   11B
000012              XSTOR   EQU   12B
000013              YDRAW   EQU   13B
000014              MINXY   EQU   14B
000015              MINSZ   EQU   15B

000000  106 032 001 PING    CAL   DRAW    DRAW SCORE, BOARD AND PADDLES
000003  115                 INP   GINP
000004  022                 RAL
000005  100 000 000         JFC   PING    WAIT FOR SWITCH 1 TO BE PUSHED
000010  056 002 066         SHL   SCORE
000013  025
000014  076 000             LMI   0
000016  060                 INL
000017  076 000             LMI   0       RESET SCORE TO 0

000021  106 361 000 SERVE   CAL   RAND    GET RANDOM BYTE IN ACC
000024  044 177             NDI   177B
000026  024 100             SUI   64      IN RANGE OF -64 TO 63
000030  066 052             LLI   L(BALPOS+3) PT TO MOST SIG. Y BALL POS
000032  370                 LMA           RANDOM VERTICAL SERVE POS. NEAR MIDDLE
000033  066 050             LLI   L(BALPOS+1)   PT TO MOST SIG. X BALL POS.
000035  076 000             LMI   0       HORIZONTAL SERVE POS. AT MIDDLE
000037  106 270 000         CAL   RDISP   PICK BALL VELOCITY FROM TABLE
000042  066 050     NXPOS   LLI   L(BALPOS+1)
000044  307                 LAM
000045  121                 OUT   XMOV
000046  066 052             LLI   L(BALPOS+3)
000050  307                 LAM
000051  123                 OUT   YMOV    POSITION BALL'S X AND Y COORD.
000052  066 066             LLI   L(BALL)
000054  106 127 001         CAL   CHAR    DRAW BALL
000057  106 032 001         CAL   DRAW    DRAW SCORE,BOARD, AND PADDLES
000062  036 027             LDI   L(XDISP)  D= ADDRESS OF XDISP
000064  046 047             LEI   L(BALPOS)  E= ADDRESS OF BALPOS
000066  106 333 000         CAL   DBLADD  COMPUTE NEW BALL X COORD
000071  030                 IND           D= ADDRESS OF YDISP
000072  040                 INE           E= ADDRESS OF BALPOS Y COORD
000073  106 333 000         CAL   DBLADD  COMPUTE NEW BALL Y COORD
000076  066 052             LLI   L(BALPOS+3)
000100  307                 LAM           GET BALL Y COORD
000101  024 176             SUI   126
000103  074 004             CPI   4       CHECK IF TOUCHES TOP OR BOTTOM
000105  066 031             LLI   L(YDISP)
000107  142 350 000         CTC   NEGATE  CHANGE Y DIRECTION IF SO
000112  066 050             LLI   L(BALPOS+1)
000114  307                 LAM           LOAD BALL X POSITION
000115  024 150             SUI   104
000117  074 002             CPI   2       SEE IF IT TOUCHES RIGHT PADDLE BOUNDARY
000121  140 212 000         JTC   RPAD    CHECK PADDLE POSITION IF SO
000124  024 054             SUI   44
000126  074 002             CPI   2       SEE IF IT TOUCHES LEFT PADDLE ZONE
000130  140 217 000         JTC   LPAD    CHECK PADDLE POSITION IF SO
000133  004 026             ADI   22
000135  074 004             CPI   4       SEE IF BALL REACHES END ZONE
000137  140 145 000         JTC   OFF     JUMP IF SO
000142  104 042 000         JMP   NXPOS   COMPUTE NEXT BALL POSITION

000145  066 033     OFF     LLI   L(DIRECT)
000147  307                 LAM           GET BALL DIRECTION
000150  260                 ORA           SET SIGN BIT
000151  160 161 000         JTS   OFFL    RECORD SCORE ACCORDINGLY
000154  066 025             LLI   L(SCORE)

000156  104 163 000         JMP   POINTS
000161  066 026     OFFL    LLI   L(SCORE+1)
000163  307         POINTS  LAM
000164  004 020             ADI   20B     BUMP SCORE INDEX
000166  370                 LMA
000167  074 260             CPI   260B    CHECK FOR END OF GAME
000171  150 000 000         JTZ   PING    JUMP IF SO
000174  066 034     LOOP    LLI   L(DELAY)  DELAY A WHILE BEFORE NEXT SERVE
000176  317                 LBM
000177  011                 DCB
000200  371                 LMB
000201  150 021 000         JTZ   SERVE   SERVE NEXT BALL AFTER DELAY
000204  106 032 001         CAL   DRAW    DRAW BOARD WHILE WAITING
000207  104 174 000         JMP   LOOP    DELAY LOOP

000212  006 004     RPAD    LAI   4       SET TO READ RIGHT DIAL
000214  104 221 000         JMP   HIT     CHECK FOR PADDLE HIT

000217  006 001     LPAD    LAI   1       SET TO READ DIAL 1

000221  106 137 001 HIT     CAL   RVCD    GET DIAL VALUE
000224  056 002 066         SHL   BALPOS+3
000227  052
000230  227                 SUM           GET DISTANCE BETWEEN BALL AND BOTTOM
                    *                     OF PADDLE
000231  004 020             ADI   16      SET CARRY IF BALL TOUCHES PADDLE
000233  100 042 000         JFC   NXPOS   LET BALL PASS IF NO CONTACT
000236  107                 INP   3B      POP SPEAKER
000237  066 033             LLI   L(DIRECT)
000241  307                 LAM           GET BALL DIRECTION
000242  054 200             XRI   200B
000244  370                 LMA           INVERT BALL DIRECTION
000245  106 270 000         CAL   RDISP   GET RANDOM BALL VELOCITY
000250  066 033             LLI   L(DIRECT)
000252  307                 LAM           GET BALL DIRECTION
000253  260                 ORA           SET SIGN BIT
000254  066 050             LLI   L(BALPOS+1)   PT TO  THE BALLS X POSITION
000256  076 147             LMI   103     CLEAR FROM RIGHT PADDLE
000260  160 042 000         JTS   NXPOS   LOOP IF BOUNCED OFF RIGHT PADDLE
000263  076 226             LMI   150     CLEAR FROM LEFT PADDLE
000265  104 042 000         JMP   NXPOS   LOOP

                    *       PICKS RANDOM BALL VELOCITY FROM TABLE

000270  106 361 000 RDISP   CAL   RAND    GET RANDOM BYTE
000273  044 074             NDI   74B
000275  066 272             LLI   L(ANGLE)
000277  206                 ADL
000300  360                 LLA           INDEX ANGLE TABLE AT RANDOM
000301  307                 LAM           TRANSFER 4 BYTES FROM ANGLE TABLE
000302  060                 INL           TO XDISP AND Y DISP
000303  317                 LBM
000304  060                 INL
000305  327                 LCM
000306  060                 INL
000307  337                 LDM
000310  066 027             LLI   L(XDISP)
000312  370                 LMA
000313  060                 INL
000314  371                 LMB
000315  060                 INL
000316  372                 LMC
000317  060                 INL
000320  373                 LMD
000321  066 033             LLI   L(DIRECT)
000323  307                 LAM           GET BALL DIRECTION
000324  066 027             LLI   L(XDISP)
000326  260                 ORA           SET SIGN FLAG
000327  160 350 000         JTS   NEGATE  NEGATE X DIRECTION IF NEEDED
000332  007                 RET           RETURN
```

```
                        *       16 BIT ADD ROUTINE

000333  363         DBLADD  LLD
000334  307                 LAM             LOAD LSB OF VALUE TO ADD
000335  364                 LLE
000336  207                 ADM             ADD BOTH LEAST SIG. BYTES TOGETHER
000337  370                 LMA             SAVE
000340  030                 IND
000341  363                 LLD
000342  307                 LAM             LOAD MOST SIG BYTE OF VALUE TO ADD
000343  040                 INE
000344  364                 LLE
000345  217                 ACM             ADD WITH CARRY
000346  370                 LMA             SAVE
000347  007                 RET             RETURN

                        *       16 BIT NEGATE ROUTINE

000350  250         NEGATE  XRA             ZERO REG A
000351  227                 SUM             LOAD NEGATIVE OF MEMORY
000352  370                 LMA             SAVE
000353  060                 INL
000354  006 000             LAI   0         ZERO REG A WITHOUT RESETTING CARRY
000356  237                 SBM             LOAD NEGATIVE OF MEMORY WITH BORROW
000357  370                 LMA             SAVE
000360  007                 RET

                        *       RETURNS RANDOM BYTE IN REGISTER A

000361  056 002 066 RAND    SHL   SHIFT+3       PT TO SHIFT BYTE 4
000364  375
000365  016 010             LBI   8             SET FOR 8 SHIFTS
000367  307                 LAM                 LOAD SHIFT BYTE 4
000370  002         RTOP    RLC
000371  002                 RLC                 MOVE BIT 28
000372  002                 RLC                 TO POSITION 31
000373  257                 XRM                 X OR BITS 28 & 31
000374  022                 RAL
000375  022                 RAL                 MOVE NEW BIT INTO CARRY
000376  066 372             LLI   L(SHIFT)      PT TO SHIFT BYTE 1
000400  307                 LAM                 LOAD SHIFT BYTE 1
000401  022                 RAL                 ROTATE THRU CARRY
000402  370                 LMA                 SAVE
000403  060                 INL
000404  307                 LAM                 LOAD SHIFT BYTE 2
000405  022                 RAL                 ROTATE THRU CARRY
000406  370                 LMA                 SAVE
000407  060                 INL
000410  307                 LAM                 LOAD SHIFT BYTE 3
000411  022                 RAL                 ROTATE THRU CARRY
000412  370                 LMA                 SAVE
000413  060                 INL
000414  307                 LAM                 LOAD SHIFT BYTE 4
000415  022                 RAL                 ROTATE THRU CARRY
000416  370                 LMA                 SAVE
000417  011                 DCB
000420  110 370 000         JFZ   RTOP          REPEAT 8 TIMES
000423  007                 RET

000424  307         DRAW1   LAM         LOAD SCORE INDEX
000425  360                 LLA         INDEX SCORE CHAR. TABLE
000426  006 177             LAI   177B
000430  123                 OUT   YMOV  POSITION CHARACTER AT TOP OF DISPLAY
000431  007                 RET
                        *       DRAWS BOARD, PADDLES, AND SCORE

000432  006 377     DRAW    LAI   377B
000434  133                 OUT   MINSZ   SET MINOR DEFLECTION SYSTEM FOR MAX RANGE
000435  056 002 066         SHL   SCORE
000440  025
000441  106 024 001         CAL   DRAW1
000444  006 260             LAI   260B
000446  121                 OUT   XMOV    POSITION RLEFT SCORE
000447  106 127 001         CAL   CHAR    DRAW IT
000452  066 026             LLI   L(SCORE+1)
000454  106 024 001         CAL   DRAW1
000457  006 100             LAI   64
000461  121                 OUT   XMOV    POSITION RIGHT SCORE
000462  106 127 001         CAL   CHAR    DRAW IT
000465  066 104             LLI   L(BOARD)
000467  106 200 001         CAL   GRAPH   DRAW OUTSIDE SQUARE
000472  006 001             LAI   1
000474  106 137 001         CAL   RVCD
000477  123                 OUT   YMOV    POSITION LEFT PADDLE
000500  006 224             LAI   -108
000502  121                 OUT   XMOV
000503  056 002 066         SHL   LPADD
000506  147
000507  106 127 001         CAL   CHAR    DRAW LEFT PADDLE
000512  006 004             LAI   4
000514  106 137 001         CAL   RVCD
000517  123                 OUT   YMOV    POSITION RIGHT PADDLE
000520  006 151             LAI   105
000522  121                 OUT   XMOV
000523  056 002 066         SHL   RPADD
000526  127

                        *       DRAWS WITH MINOR DEFLECTION SYSTEM

000527  307         CHAR    LAM         LOAD BYTE FROM MEMORY
000530  260                 ORA         SET SIGN BIT
000531  131                 OUT   MINXY  DRAW BYTE
000532  063                 RTS         RETURN IF LAST BYTE
000533  060                 INL         BUMP MEMORY INDEX
000534  104 127 001         JMP   CHAR    DRAW NEXT BYTE FROM TABLE

000537  310         RVCD    LBA         COMPUTE BIT MASK FROM DIAL NUMBER
000540  006 020             LAI   20B     START WITH MASK OF 00010000
000542  012         RVCD1   RRC         SHIFT MASK RIGHT UNTIL DIAL
000543  011                 DCB         NUMBER BECOMES ZERO
000544  110 142 001         JFZ   RVCD1
000547  310                 LBA         SAVE BIT MASK IN B
000550  056 200             LHI   200B  INITIALIZE TRIAL VALUE IN H
000552  365                 LLH         INITIALIZE TRIAL BIT IN L
000553  305         RVCD2   LAH         GET CURRENT TRIAL VALUE IN A
000554  256                 XRL         FLIP CURRENT TRIAL BIT
000555  121                 OUT   XMOV  SEND TO DIGITAL-TO-ANALOG CONVERTER
000556  300                 LAA         WAIT FOR 741'S TO SETTLE
000557  300                 LAA
000560  115                 INP   GINP  READ COMPARATORS & SWITCHES
000561  241                 NDB         MASK TO GET CHANNEL OF INTEREST
000562  150 170 001         JTZ   RVCD3 JUMP IF TRIAL TOO HIGH
000565  305                 LAH         RETAIN TRIAL IF TOO LOW
000566  256                 XRL
000567  350                 LHA
000570  306         RVCD3   LAL         SHIFT TRIAL BIT RIGHT 1
000571  012                 RRC
000572  360                 LLA
000573  100 153 001         JFC   RVCD2 DO ANOTHER ITERATION IF ALL BITS NOT TRIED
000576  305                 LAH         LOAD FINAL RESULT INTO A
000577  007                 RET         RETURN

                        *       DRAWS WITH MAJOR DEFLECTION SYSTEM

000600  317         GRAPH   LBM         GET TABLE COUNT OF COORD PAIRS TO DRAW
000601  250                 XRA
000602  131                 OUT   MINXY  RESET MINOR DEFLECTION SYSTEM
000603  060                 INL
000604  307                 LAM
000605  121                 OUT   XMOV  POSITION BEAM X COORDINATE
000606  060                 INL
000607  307                 LAM
000610  123                 OUT   YMOV  POSITION BEAM Y COORDINATE
000611  011         GRAPH1  DCB         DCREMENT COUNT
000612  053                 RTZ         RETURN WHEN DONE
000613  060                 INL
000614  307                 LAM
000615  300                 LAA         DELAY
000616  300                 LAA         DELAY
000617  125                 OUT   XSTOR LOAD X COORD
000620  060                 INL
000621  307                 LAM
000622  127                 OUT   YDRAW DRAW LINE
000623  104 211 001         JMP   GRAPH1 LOOP

                        *
                        *               ALL VARIABLES MUST BE LOCATED IN THE SAME
                                        PAGE

001000                      ORG   1000B
001000          DATA    EQU   *             BEGINNING OF DATA AREA

001000  040 131 136         DEF   40B,131B,136B,147B                ZERO
001003  147
001004  167 176 171         DEF   167B,176B,171B,160B
001007  160
001010  340                 DEF   340B

001020                      ORG   DATA+20B
001020  040 160 050         DEF   040B,160B,050B,157B               ONE
001023  157
001024  345                 DEF   345B
001025  000 000     SCORE   DEF   0,0     KEEPS PLAYERS' SCORE
001027  000 001     XDISP   DEF   0,1     PRESENT BALL X VELOCITY COMPONENT
001031  000 000     YDISP   DEF   0,0     PRESENT BALL Y VELOCITY COMPONENT
001033  000         DIRECT  DEF   0       INDICATES BALL DIRECTION
001034  000         DELAY   DEF   0       USED TO DELAY BETWEEN SERVES

001040                      ORG   DATA+40B
001040  026 137 157         DEF   026B,137B,157B,154B               TWO
001043  154
001044  111 110 360         DEF   111B,110B,360B
001047  000 000 000 BALPOS  DEF   0,0,0,0    PRESENT BALL POSITION
001052  000

001060                      ORG   DATA+60B
001060  030 170 177         DEF   030B,170B,177B,137B               THREE
001063  137
001064  074 334             DEF   074B,334B
001066  000 110 111 BALL    DEF   0,110B,111B,101B
001071  101
001072  300                 DEF   300B

001100                      ORG   DATA+100B
001100  050 157 124         DEF   050B,157B,124B,364B               FOUR
001103  364
```

```
001104  005          BOARD   DEF   5
001105  200 200 200          DEF   -128,-128,-128,127
001110  177
001111  177 177 177          DEF   127,127,127,-128
001114  200
001115  200 200              DEF   -128,-128

001120                       ORG   DATA+120B
001120  030 150 161          DEF   030B,150B,161B,163B      FIVE
001123  163
001124  133 136 366          DEF   133B,136B,366B
001127  000 120 127 RPADD    DEF    0B,120B,127B,307B
001132  307

001140                       ORG   DATA+140B
001140  046 136 123          DEF   046B,136B,123B,120B      SIX
001143  120
001144  150 153 323          DEF   150B,153B,323B
001147  020 100 107 LPADD    DEF    20B,100B,107B,327B
001152  327

001160                       ORG   DATA+160B
001160  050 167 327          DEF   050B,167B,327B           SEVEN

001200                       ORG   DATA+200B
001200  020 160 167          DEF   020B,160B,167B,127B      EIGHT
001203  127
001204  120 024 364          DEF   120B,024B,364B

001220                       ORG   DATA+220B
001220  050 157 137          DEF   050B,157B,137B,126B      NINE
001223  126
001224  125 134 354          DEF   125B,134B,354B

001240                       ORG   DATA+240B
001240  040 131 136          DEF   040B,131B,136B,147B      TEN
001243  147
001244  167 176 171          DEF   167B,176B,171B,160B
001247  160
001250  140 000 120          DEF   140B,000B,120B,010B
001253  010
001254  117 305              DEF   117B,305B

001260                       ORG   DATA+260B
001260  040 160 050          DEF   040B,160B,050B,157B      ELEVEN
001263  157
001264  145 000 120          DEF   145B,000B,120B,010B
001267  010
001270  117 305              DEF   117B,305B

                             *                              TABLE OF BALL VELOCITIES
001272  144 000 012 ANGLE    DEF   100,0,10,254             -79 DEG
001275  376
001276  304 000 047          DEF   196,0,39,254             -67 DEG
001301  376
001302  034 001 126          DEF   28,1,86,254              -56 DEG
001305  376
001306  152 001 226          DEF   106,1,150,254            -45 DEG
001311  376
001312  252 001 344          DEF   170,1,228,254            -34 DEG
001315  376
001316  331 001 074          DEF   217,1,60,255             -22 DEG
001321  377
001322  366 001 234          DEF   246,1,156,255            -11 DEG
001325  377
001326  000 002 000          DEF   0,2,0,0                  0 DEG FAST
001331  000
001332  000 001 000          DEF   0,1,0,0                  0 DEG SLOW
001335  000
001336  366 001 144          DEF   246,1,100,0              11 DEG
001341  000
001342  331 001 304          DEF   217,1,196,0              22 DEG
001345  000
001346  252 001 034          DEF   170,1,28,1               34 DEG
001351  001
001352  152 001 152          DEF   106,1,106,1              45 DEG
001355  001
001356  034 001 252          DEF   28,1,170,1               56 DEG
001361  001
001362  304 000 331          DEF   196,0,217,1              67 DEG
001365  001
001366  144 000 366          DEF   100,0,246,1              79 DEG
001371  001

001372          SHIFT   DST   4      RANDOM NUMBER REGISTER
                *                    SET ANY WAY EXCEPT ALL ZEROES
001376                  END   PING
```




START
VARIABLES CONTAIN INITIAL VALUES
PING
DRAW SCORE, PADDLES, AND BOARD
IS SWITCH 1 ON?
NO
YES
RESET SCORE
SERVE
SET RESTRICTED RANDOM BALL POSITION
CALL RDISP PICK VELOCITY AT RANDOM FROM TABLE
NXPOS
DRAW SCORE, PADDLES, BOARD, AND BALL
COMPUTE NEXT BALL POSITION
DOES BALL TOUCH TOP OR BOTTOM?
YES
NEGATE Y VELOCITY
NO
BALL IN RIGHT PADDLE ZONE?
YES
COMPARE RIGHT PADDLE AND BALL POSITION
HIT?
NO
YES
POP SPEAKER
CHANGE BALL DIRECTION
CALL RDISP PICK VELOCITY AT RANDOM FROM TABLE
CLEAR BALL AWAY FROM PADDLE ZONE
NO
BALL IN LEFT PADDLE ZONE?
YES
COMPARE LEFT PADDLE AND BALL POSITION
NO
BALL IN END ZONE?
NO
YES
OFF
BUMP SCORE DEPENDING ON BALL DIRECTION
LOOP
END OF GAME?
NO
DRAW DISPLAY 256 TIMES TO DELAY
YES

SURPLUS SUMMARY

Guess what! IMP-16 chip sets have hit a new low in price. This is perfect timing, next issue of TCH will begin a series on the IMP-16. The new deal is from Poly-Paks who, in their latest flyer, have the 5 chip set listed for $49.99.

Poly Paks
Box 942B
Lynnfield, Mass. 01940

Components useful for graphics displays have finally shown up as surplus. Suntronix has a batch of Sanders 720 CRT heads. These heads are "dumb" units, i.e., they have no buffer or character generator, all data comes in over a cable. However they are great for graphics because they used stroke type character generation rather than raster scan. The power supply, CRT, deflection amplifiers, and possibly even the DAC's from the unit could be utilized. Unfortunately, Suntronix is not shipping at this time due to pending court action by Sanders regarding interpretation of the word "scrap" in the contract.

Suntronix
6 King Richard Drive
Londonerry, NH 03053
Ph. 603/434-4644

Want an easy to use but somewhat weird replacement for paper tape? well Delta t has one. Its an 8 track incremental magnetic tape recorder, except the "tape" is a 16 MM magnetic film cartirdge. The unit will read and write 330 bytes per second asynchronously (you need not control data rate as long as you do not exceed 330 bytes/sec.). From their pictures the unit looks well built and they seem to have all the necessary items including interface from the 12 volt logic to TTL and the weird cartridges. The recorder goes for $250.00, the modifications for TTL (assembled, installed, and tested) adds $100, and extra cartridges go for $20.00. For details write:

Delta t
11020 Old Katy Road
Suite 204
Houston, TX 77043

TCH cassette boards are still available and will be until notice is given otherwise. The relay offer is another matter however, the supply is nearly exhausted so if you order them send two seperate checks, one for the boards, and one for the relays which we could return when the supply is gone.

CLASSIFIED ADS

There is no charge for classified ads in TCH but they must pertain to the general area of computers or electronics, and must be submitted by a non-commercial subscriber. Feel free to use ads to buy, sell, trade, seek information, announce meetings, or for any other worthwhile purpose. Please submit ads on seperate sheets of paper and include name and address and/or phone number. Please keep length down to 10 lines or less.

MEDICAL APPLICATIONS: I am interested in contacting individuals with a serious interest in the application of microprocessor technology to medical instrumentation, and automated diagnostic systems. Please contact James A. Willis, 3013 Woodlawn Ave., Falls Church, VA 22042. Ph. 703/532-8242

FOR SALE: TMS 4030 ZA0248 Dynamic RAM's. 420ns access time, 690ns cycle. Ideal for 8080 and IMP-16 microprocessors. $13 each, 8 or more $10 each. Andy Pitts, PO Box 5734, Winston Salem, NC 27103. Ph. 919/765-1277

FOR SALE: Precision 1% metal film 1/4 watt 100 ohm resistors, $2.00 per 50. Postpaid. C. Funk, 711 Eno Street, Hillsboro, NC 27278

FOR SALE: 10 CPS hard copy teleprinter with keyboard. Exact equivalent of TTY KSR. Serial in/serial out. Only one available. This will go fast at only $300. M. W. Smith, 4355 S. High Street, Englewood, CO 80110

POWER SUPPLY: I have a quantity of 5V 10 amp highly regulated power supplies taken from keyboard terminals. I will provide schematics and plans for obtaining -5V, -9V, and -12V. $25 plus postage on 15 pounds. Grant Runyan, 1146 Nirvana Rd., Santa Barbara, CA 93101

CASSETTE TAPE ROM ORDER FORM

CPU Type [_] 8008 [_] 8080

Status input device address: _____

Control output device address: _____

Memory page allocated to the ROM: _____

Memory load address for the IPL program: _____

Stack address for the IPL program (8080 only): _____

(Please specify all device and memory addresses in octal)

NAME _____

STREET _____

CITY _____ STATE _____ ZIP _____

We can program 1702, 1702A, and 5203 PROMS. At this time, TCH can only program customer supplied ROMS. The programming charge of $2.00 covers programming, verification, and return shipment. An octal listing of the ROM contents will also be enclosed. We are using an unmodified Intel PROM programmer and are strictly following the manufacturer's programming recommendations. Any PROMS that we cannot program successfully are definitely bad and will be returned. The printout will have the errors marked to aid you in obtaining a refund from your supplier. Unfortunately bad PROMS require as much effort as good ones so there will be no refund.

THE COMPUTER HOBBYIST
Box 295
Cary, NC 27511

ADDRESS CORRECTION REQUESTED

FIRST CLASS
POSTAGE PAID
CARY, NC 27511
PERMIT NO. 34

FIRST CLASS MAIL