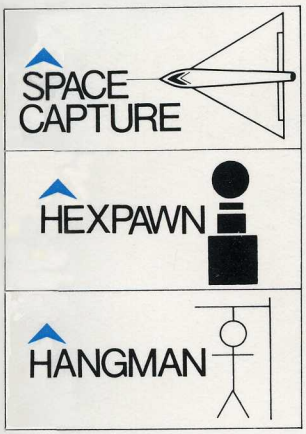
SCELBI's FIRST BOOK of COMPUTER GAMES for the "8008"/"8080"



SCELBI COMPUTER CONSULTING INC.

SCELBI'S FIRST BOOK OF COMPUTER GAMES
for the
'8008/8080'

AUTHORS:

Nat Wadsworth
and
Robert Findley

Scelbi Computer Consulting, Inc.
1322 Rear - Boston Post Road
Milford, CT. 06460

IMPORTANT NOTICE

The information in this manual has been carefully reviewed and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies or for the success or failure of various applications to which the information contained herein might be applied.

Acknowledgement

Composing and typesetting material while working from rough drafts, and painstakingly copying source listings and the myriads of octal numbers from assembled listings, is a lot of tedious work which demands a high degree of accuracy. We wish to express our special appreciation for the services of:

Ms. Gabrielle Tingley

who performed this task for the material in this publication in a most efficient manner.

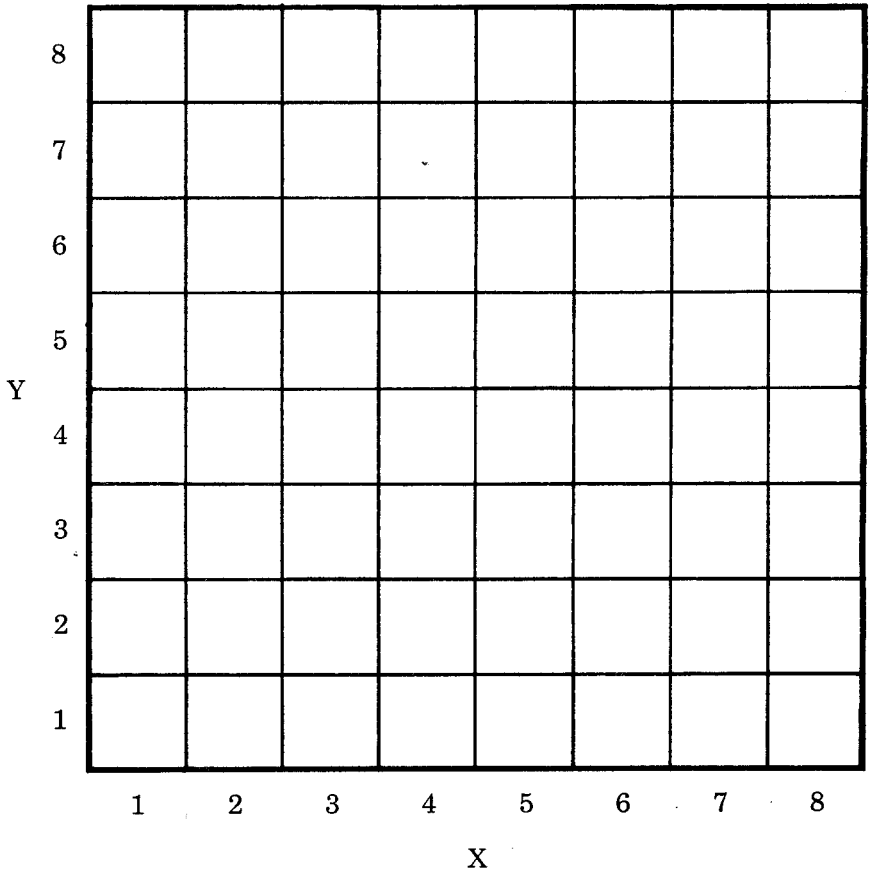
The Authors

SCELBI'S FIRST BOOK OF COMPUTER GAMES
for the
'8008/8080'

****
TABLE OF CONTENTS
****

SPACE CAPTURE

Space Capture is a game of skill and chance. The object of the game is to capture an imaginary space ship by destroying all the possible sectors that it might attempt to travel in. The game as presented herein utilizes a game board consisting of a grid containing 64 squares or sectors. The sectors are identified by X and Y coordinates. A pictorial of the playing board is illustrated below.

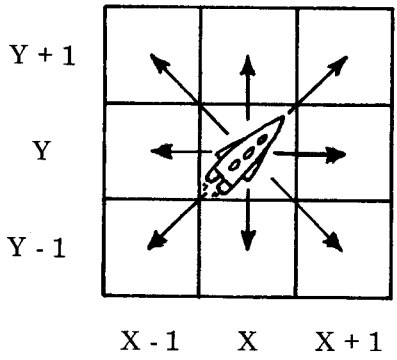


To start the game, the computer informs the player of a location

where the space ship was last observed. The player is then allowed to take one PHASOR SHOT by giving the X and Y coordinates of a SECTOR. The phasor shot will destroy the sector specified thereby preventing the space ship from traveling therein in the future. However, the player must be careful! If the space ship happens to be residing in a sector at the time it is destroyed by a phasor shot, then the space ship itself is considered destroyed. Since the object of the game is to CAPTURE the ship (for its cargo of course!), destroying the vessel is a losing move for the player.

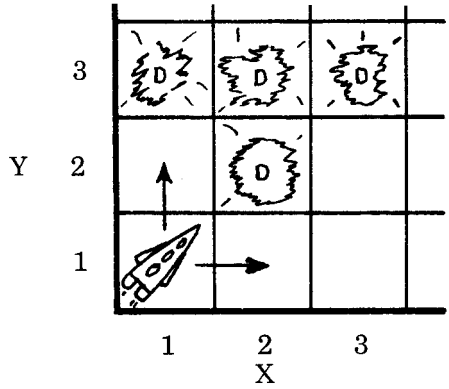
The space ship is limited to moving only one sector at a time. As already mentioned, it may not move into an area that has been previously hit by a phasor shot. The ship's movement is also restricted to the boundaries of the eight by eight grid on which the game is played.

The maximum number of different moves the space ship has to choose from at any given time is thus eight. This is illustrated in the diagram shown below. This maximum number of possible moves, for instance, would be the case at the start of a game before a player had destroyed any sectors.



However, once the game is underway, the number of possible directions in which the space ship may move can be reduced. The example illustrated next shows the space ship in a position where only two moves are possible. This is because it is bounded on two sides by the edges of the playing grid. Additionally, the diagram

shows several sectors marked by a D. These represent sectors that have been destroyed by the phasor shots of the player. The space ship may not enter into those areas. Thus, in the example, the space ship is only able to move up to the position X = 1 and Y = 2 or to the right into the position X = 2 and Y = 1. The space ship would be CAPTURED in the illustration if those two sectors had also been destroyed so that it could not move out of the indicated position X = 1 and Y = 1.



The game is relatively simple as far as computer games go, but it is a lot of fun because the moves of the space ship are made essentially random by the program. One may create strategies to attempt to use to capture the space ship, but one can never be certain where the next move will be until the space ship is captured. Also, if one does not take care where one shoots the phasor shots, the elements of chance can again enter the game. Remember, destroying a sector with the space ship traveling in it at the time ends the game!

The program for the game as it will be presented here will reside with room to spare in about 5 pages (256 bytes per page) of an '8008' or '8080' microcomputer system. If a person has even less memory available, the program can readily be compacted by the removal of some non-essential text messages. More room could be saved by more effective subroutining and attention to the program's organization to reduce the number of times pointers are altered. These techniques would allow the program to fit easily in less than three pages of memory. The reader should remember that the follow-

ing program was designed so that the operation of the program could be easily followed. It was definitely not designed to minimize memory usage other than in the sense that this machine language version is many times more compact that would be required if the program utilized a higher level language for compilation or interpretation!

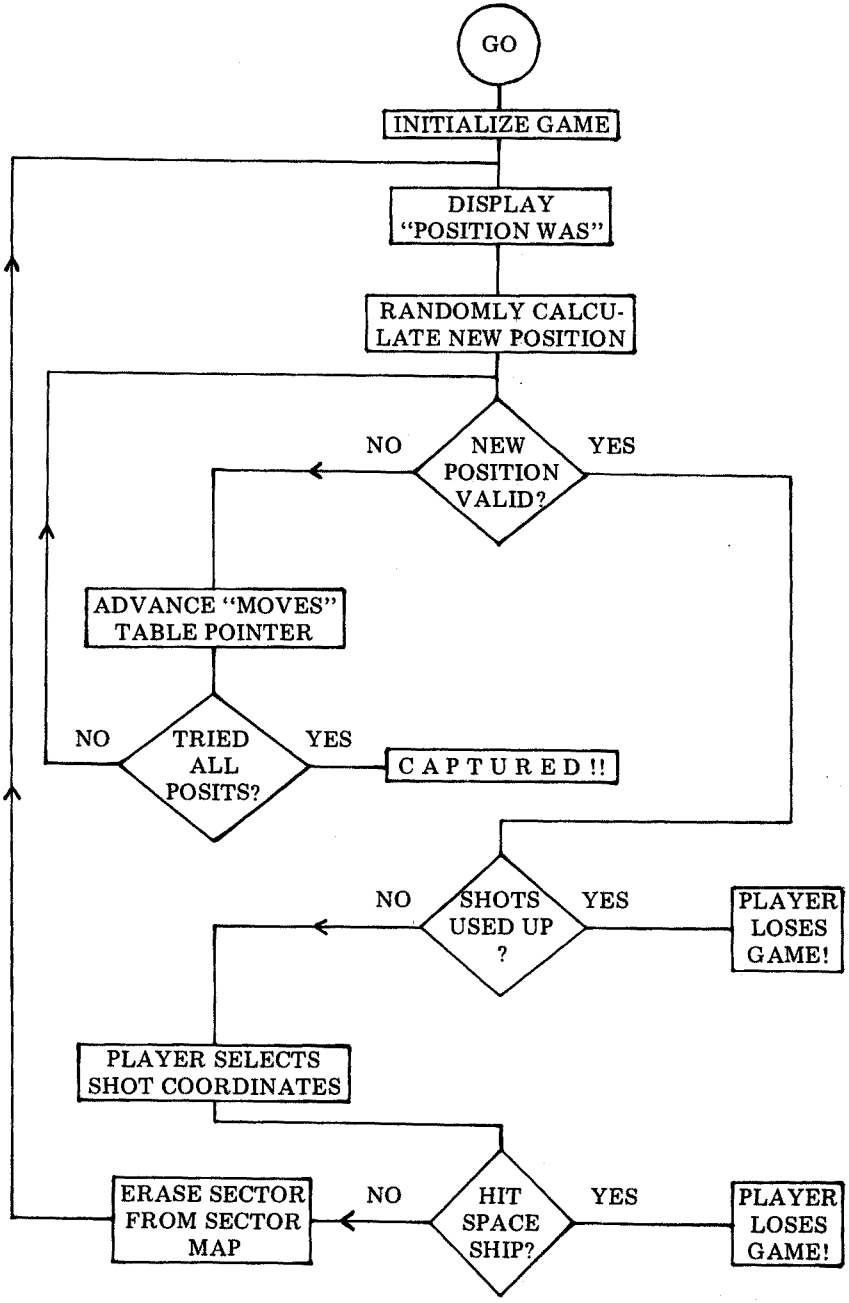
The fundamental operation of the program is outlined in the flow chart that appears on the next page. A brief verbal explanation of that chart will follow. Then the various portions of the program will be presented and discussed in detail.

OUTLINE OF THE PROGRAM'S OPERATION

At the start of the program, a brief message is presented to explain the game to a new player. Next, the program determines if the operator desires to play a game. If not, a closing message is displayed and the program ends.

Assuming that a person elects to play a game, the program proceeds to select a semi-random starting point as the initial position of the space ship that is to be captured. The position of the space ship is then displayed as the WAS position to the operator. That is, the operator is informed of the LAST POSITION in which the space ship was observed. The operator then knows one position in which the space ship cannot be because the actual current position of the ship when the player fires a phasor shot will be in a sector adjacent to its last announced position. (Unless, of course, the ship has been captured.)

Once the WAS position has been displayed, the program proceeds to calculate a new position for the space ship to move into using an essentially random method. Whenever a calculation to move into a new sector has been made, the program must perform several tests. It must make sure that the new sector is within the bounds of the playing grid. And, it must make sure that it is not moving into a sector that has been destroyed by a player's phasor shot. If either of these tests fail, a new calculation is made to try


GO
INITIALIZE GAME
DISPLAY "POSITION WAS"
RANDOMLY CALCULATE NEW POSITION
NEW POSITION VALID?
NO
YES
ADVANCE "MOVES" TABLE POINTER
TRIED ALL POSITS?
NO
YES
CAPTURED!!
SHOTS USED UP ?
NO
YES
PLAYER LOSES GAME!
PLAYER SELECTS SHOT COORDINATES
HIT SPACE SHIP?
NO
YES
ERASE SECTOR FROM SECTOR MAP
PLAYER LOSES GAME!

another one of the eight possible adjacent sectors. If the program finds that all eight possible moves are blocked, the player wins. An appropriate message is then displayed.

Assuming that the space ship does have a valid move, the new position that it occupies is saved temporarily. If the player has not already expended an allotted number of shots, the program allows the operator to enter the coordinates of the sector that is to be eliminated from further occupation by the space ship. Once the coordinates have been entered, a test is made to see whether the space ship is presently in that sector. If so, the player loses as the space ship was destroyed versus being captured. If the space ship was not hit by the phasor shot, then the sector area is erased from a SECTOR MAP. Once an entry in the sector map has been erased, the space ship will be prevented from entering that sector in the future.

The game continues until the player uses up the allotted number of phasor shots, hits the space ship, or obtains a CAPTURE. At the conclusion of a game, the program queries the player as to whether a new game is to be played. Appropriate action is then taken as indicated above.

TEXT MESSAGES USED BY THE PROGRAM

Close to a third of the memory space utilized by the program is for storing the ASCII code for various messages that are displayed during the program's operation. These messages are of esthetic value particularly if the game is to be enjoyed by those who may not be familiar with the operation of a computer. The contents of these messages may be altered by the reader as desired, including complete deletion in many instances if one desires to conserve memory space. The various message strings that are used in the program being presented are listed next.

"SPACESHIP CAPTURE. YOU HAVE 15 PHASOR SHOTS WITH WHICH TO DESTROY MY TRAVEL SECTORS. IF ALL MY ADJACENT SECTORS ARE DESTROYED I AM CAPTURED. IF YOU HIT ME OR RUN OUT OF PHASOR ENERGY, THEN YOU LOSE!"

"WANT TO PLAY?"

"POOR SPORT!"

"MY LAST POSITION WAS: X = "

", Y = "

"YOU ARE FIRING TO: X = "

YOU HIT ME!! YOU LOSE!"

YOU ARE OUT OF PHASOR ENERGY, YOU LOSE!"

"#!0# DARN! YOU HAVE ME C A P T U R E D !!"

The introductory message in particular takes almost a page of storage in memory and may readily be deleted if memory is at a premium in the user's system. The reader who wants to reduce the memory requirements some more can abbreviate the other messages if desired.

The text messages shown above are all stored in one continuous section in memory in the form of ASCII codes for the various characters in each string. (Note: In this manual the standard seven bit ASCII code will be shown with the code augmented by an eighth bit commonly referred to as the PARITY bit. The parity bit will always be assumed to be in the logic one or marking condition unless otherwise noted.)

A subroutine frequently referred to by the game program is shown

below. It is labeled MSG.

```
MSG,    LAM         Fetch a character
        NDA         Set flags
        RTZ         Finished if have zero byte
        CAL PRINT   Else print character
        INL         Advance low addr pointer
        JFZ MSG     Continue display
        INH         Or adv page addr pointer
        JMP MSG     And then continue display
```

The MSG subroutine is quite simple. The calling program simply sets up the H and L memory pointing registers to the starting address of a string of characters that are to be outputted. Then, when the MSG subroutine is executed, the routine proceeds to fetch the characters from memory and output them until a zero byte is encountered. The subroutine itself calls on a subroutine labeled PRINT which must be provided by the program user. The PRINT subroutine must be an actual device operating routine that will cause the ASCII character in the accumulator to be transmitted to the output device being used by the system. The PRINT subroutine provided by the user may use the CPU registers B through E if required but it should not alter the contents of the H and L CPU registers. (Unless, of course, in doing so it is able to restore them to their original values before returning to the calling program.)

The reader will see that the MSG subroutine is used through-out the program being described. Prior to calling the subroutine, the main program will always setup the H and L registers to the starting address of the character string that is to be displayed. The character strings that will be used in the example program have been presented previously. It hardly goes without saying, that if a reader desires to modify the text messages, and by doing so alters their starting addresses, that appropriate modifications must be made to the setup address values whenever the MSG subroutine is used. This is also the case if the user decides to store the text messages at locations other than those shown in the program provided herein.

The reader may refer to the flow chart presented earlier as the discussion of the actual operating portions of the program proceeds.

The first few procedures in the program consist of merely displaying the introductory message and then asking the prospective player if the playing of a game is desired.

Following the WANT TO PLAY query, the program then waits for a response from the system's input device which must be in the form of a letter Y for YES or N for NO. If a NO response is received at this point, then no game is to be played. The program will display the closing message and end the program. These first few operations are illustrated in the program listing below.

```
START,  LHI 000      Pointer to introductory
        LLI 000      Message
        CAL MSG      Display introductory message

OVER,   LHI 000      Pointer to WANT TO PLAY
        LLI 325      Message
        CAL MSG      Display message
INAGN,  CAL CKINP    See if have input
        NDA          Set flags
        CFS INPUTN   Fetch character if ready
        INL          Increment RANDOM cntr
        CPI 316      If input, was it N?
        JFZ NOTNO    Jump ahead if not N
        LHI 000      Pointer to POOR SPORT
        LLI 350      Message
        CAL MSG      Display message
        HLT          End of session
```

There are several instructions in the above routine that require elaboration. The reader may observe that there are references to two input subroutines. One is labeled CKINP. The other is designated

INPUTN. The subroutine labeled INPUTN is a user created subroutine that will accept a character from the system's input device. Typically, this would be an ASCII encoded keyboard. The subroutine is expected to return the inputted character in the accumulator. This subroutine is free to use CPU registers B through E in performing its function. The INPUTN subroutine should also provide an echo capability by sending the character inputted out to the system's display device. This is done so the operator may verify the character inputted. (This might be accomplished by simply calling the previously mentioned PRINT subroutine.)

The CKINP subroutine is a user provided routine that simply performs a check to see if the input device has a character waiting to be inputted. If so, the subroutine must return with the MSB of the accumulator set to '0.' If a character is not ready, the subroutine should return with the most significant bit of the accumulator set to a logic '1' state.

The importance of having a separate subroutine (CKINP) that merely ascertains if a character is waiting will be explained here. The reader can see in the previous routine that if a character is ready to be received, the INPUTN subroutine will be called to actually obtain the data. However, whether or not a character is received, CPU register L will be incremented. CPU register L is actually used as a sort of random counter. The final value in register L will be determined by how long it takes for the player to respond with an input after a query from the program. This is because if there is no input the first time the instruction sequence is executed, the routine will eventually loop back to the point in the program labeled INAGN. Each time the program has to wait and goes back through the loop, the contents of register L are incremented. Naturally, this looping operation is being performed at a many-thousands-per-second rate.

How the final value in register L is used to form an essentially random number (when a valid input finally occurs) will be illustrated shortly. It is important to note that the inclusion of the separate CKINP subroutine is vital to the proper operation of the program being described. To reiterate, the CKINP subroutine must only ascertain if the input device has a character for the computer! It does not itself form a waiting loop for such a signal. That is accomplished

by the previous routine in the manner described!

The program continues with a portion to be illustrated next starting with the label NOTNO. The first part of this sequence completes the test of the player's response to the WANT TO PLAY? query. This is done by testing to see if the character Y for YES was inputted. If not, the program loops back to the label INAGN just described to continue looking for a valid input.

When a Y response is received, the routine continues in the following manner. The value in register L is transferred to the accumulator. It is trimmed by a masking operation to leave only the three least significant bits. This would leave an octal number in the range of zero to seven. A count of one is added to this value to give a number in the range 01 to 10 octal. This is the equivalent of decimal 1 to 8, or the allowed coordinates along either the X or Y coordinate of the playing grid! This value is then saved in two temporary storage locations in memory to serve as the initial position of the space ship at the start of the game. Thus, the space ship will always have a starting point along a line corresponding to the diagonal where both coordinates are the same value. However, the actual value will vary with each game because of the random manner in which the number is generated (in register L) as alluded to previously. This gives some added variety to the game right from the beginning move!

The routine then continues by taking the value in the accumulator and reducing it by a masking operation back to the octal range 0 to 7. The value is then multiplied by 2 (RLC instruction) so that it will represent an even number in the range 00 to 16 octal.

At this point the value is converted to the low portion of an address. For this particular version of the program this is accomplished by the ORI 260 command which will form a value in the range 260 to 277 (octal). This address is stored temporarily in memory for use by a routine that will be explained in detail further on. Suffice it to say at this point that the address refers to a table that will contain the possible moves that the space ship might try to take.

The routine then continues by initializing a phasor shots taken counter that will keep track of how many shots the player has fired.

Because of the point in the overall program at which the counter is decremented, this counter is initialized to a value one greater than the number of shots that the player is to be allowed.

The routine concludes by filling a block of 64 (decimal) locations with all ones. This block of memory will serve as a shots taken map. Its use will be explained in detail later.

```
NOTNO,   CPI 331       If input, was it Y?
         JFZ INAGN     If not, get a new input
         LAL           Else, move random counter
         NDI 007       To ACC & trim ASCII code
         ADI 001       Add 1 to get 1 - 8
         LHI 001       Range and set up pointer
         LLI 372       To LAST position storage
         LMA           Initialize X WAS value
         INL           Advance pointer
         LMA           Initialize Y WAS value
         LLI 377       Pointer to random cntr storage
         NDI 007       Reduce size
         RLC           Make it an even value
         ORI 260       Form table pointer
         LMA           And save table pointer
         LHI 001       Set pointer to shot counter
         LLI 376       Storage location
         LMI 020       Initialize to 16 decimal
         LHI 003       Set pointer to start of
         LLI 300       Shots taken map
         LAI 377       Fill accumulator with 1's

FILOOP,  LMA           Initialize shots taken
         INL           Map to all ones condition
         JFZ FILOOP    Until map completed
```

The next portion of the program starts by displaying the message MY LAST POSITION WAS: to the player. The routine then fetches the values of the X and Y coordinates that have been previously

stored in memory and outputs those values by forming the ASCII code for the appropriate numerical values and displaying them via the output display subroutine PRINT provided by the user.

Next, a subroutine termed TRYMOV is called. The TRYMOV subroutine, which will be discussed shortly, will attempt to move the space ship into an available free sector using a technique that selects a new location in an essentially random manner. If the TRYMOV subroutine cannot move the space ship, the program will not return in the normal manner as the space ship will have been captured. If, however, the space ship is able to move to a new sector, the program will continue as illustrated in the routine. At this point, the phasor shots taken counter will be decremented in value. If the player has not used up the allotted shots, the game continues.

If the player has used up the number of allotted phasor shots, the program continues to the label PHASOR. Here the program will display the message indicating that the player is out of phasor energy and has lost the game. The program will then loop back to the label OVER presented earlier to see if the player wants to start a new game.

```
PLAYIN,   LHI 000      Set pntr to POSITION WAS:
          LLI 367      X = message
          CAL MSG      Display message
          LHI 001      Set pointer to X WAS
          LLI 372      Storage location
          LAM          Fetch value
          ORI 260      Form ASCII code
          CAL PRINT    Display position value
          LHI 001      Set pointer to Y =
          LLI 026      Message
          CAL MSG      Display message
          LHI 001      Set pointer to Y WAS
          LLI 373      Storage location
          LAM          Fetch value
          ORI 260      Form ASCII code
          CAL PRINT    Display position value
          CAL TRYMOV   Move the spaceship!
```

```
            LHI 001         Pointer to shots taken
            LLI 376         Counter storage
            LBM             Fetch counter
            DCB             Decrement value
            LMB             Restore counter
            JFZ CONTIN      Jump ahead if counter not 0

PHASOR,     LHI 001         If shots counter = 0,
            LLI 125         Set pointer and display
            CAL MSG         OUT OF ENERGY message
            JMP OVER        Go see if want new game
```

Provided that the player still has shots available with which to destroy travel sectors, the program continues at the label CONTIN. Here the message YOU ARE FIRING TO: is displayed. The program then allows the player to enter first the X and then the Y coordinate of the sector that the player wishes to destroy.

When obtaining the X coordinate, the program simply calls the subroutine INPUTN to obtain a character from the input device. The character obtained is checked to see if it is in the range of one to eight decimal. If not, the routine loops back to wait for a valid input. If so, the ASCII code is trimmed down to four bits and the value saved in a temporary location as the new value of X in memory.

The program then prepares to receive the Y coordinate from the player.

```
CONTIN,     LHI 001         Set pointer to
            LLI 036         FIRING TO message
            CAL MSG         Display message

INX,        CAL INPUTN      Fetch X value
            CPI 261         See if input is a
            JTS INX         Digit in the range
            CPI 271         of one to eight decimal
            JFS INX         Ignore input if not
            LHI 001         If valid input set pointer
```

```
        LLI 370         To new X storage location
        NDI 017         Trim off ASCII part
        LMA             Save the new X value
        LHI 001         Set pointer to
        LLI 026         Y = message
        CAL MSG         Display message
```

The input for obtaining the Y coordinate from the player is handled in the same manner that was described for the portion of the program where the player responds to the WANT TO PLAY? query. Register L is again used as a counter whose final value will depend on how long it takes the player to enter the Y coordinate. When a valid character is received (in the range one to eight decimal), the trimmed number is saved in memory as the new coordinate along the Y axis. The value in register L is then processed in the same manner as before to form an address that will be utilized by the TRYMOV subroutine that has already been referred to, and which will be described soon.

```
INY,    CAL CKINP       See if have input
        NDA             Set flags
        CFS INPUTN      Fetch character if ready
        INL             Advance random counter
        CPI 261         See if input
        JTS INY         In decimal range
        CPI 271         One to eight
        JFS INY         Else ignore input
        LBL             Save random counter value
        LHI 001         Set pointer to new Y
        LLI 371         Storage location
        NDI 017         Trim ASCII part off
        LMA             Save the new Y value
        LAB             Move random counter to ACC
        NDI 007         Reduce it in size
        RLC             Make it an even value
        ORI 260         Form random table pointer
        LHI 001         Set pointer to random table
        LLI 377         Pointer storage location
        LMA             Save random pointer
```

After the X and Y shot coordinates have been obtained from the player, the program continues at the point labeled HITEST. At this time the program must perform a check to determine whether the sector which the player has just destroyed is the same one in which the space ship might have been in. (Which is determined by the TRYMOV subroutine. Remember, the TRYMOV subroutine has already been called by the program, even though it has not yet been presented in detail in this discussion.) This is determined by testing to see if the coordinates of the player's phasor shot (stored in memory) match with the new location of the space ship (also stored in memory). If a match occurs here, then the space ship has been hit. That is a losing move for the player, and the program will display the appropriate message and return to see if the player wants to try a new game.

```
HITEST,   LHI 001       Set pointer to
          LLI 370       X phasor shot storage
          LAM           Fetch X shot value
          INL           Advance pointer to new
          INL           X spaceship location value
          CPM           Compare shot with location
          JFZ ZERSEC    If not a match, no hit
          DCL           Set pointer to Y phasor
          LAM           Shot storage and fetch
          INL           Advance pointer to new
          INL           Y spaceship location value
          CPM           Compare shot with location
          JFZ ZERSEC    If not a match, no hit

BOMB,     LHI 001       Shot hit spaceship - set
          LLI 072       Pointer to HIT message
          CAL MSG       Display message
          JMP OVER      See if want new game
```

The next section of the program begins at the label ZERSEC. This portion of the program serves to zero-out a sector in the sector map whenever a sector is destroyed by a phasor shot made by the player.

As the reader knows, there are 64 different sectors in the 8 by 8

grid on which the game is played. At the beginning of a game, an area in memory is assigned as a sector map. This area consists of 64 consecutive locations in memory. In the sample program, the memory area is assigned to locations 300 to 377 (octal) on page 03. The area is initialized (as mentioned earlier) by loading the value 377 into all 64 locations. Now, each time the player specifies a grid location by designating an X and Y coordinate, a memory location in the sector map is changed to be in a 000 (octal) condition. The location that is to be zeroed out is ascertained by performing a simple calculation. The fundamental calculation made may be expressed by the following formula:

$$V = [(X - 1) \times 8] + (Y - 1)$$

where X and Y are the respective coordinates given by the player when firing a phasor shot. The value V obtained by the calculation is then added to a base value (300 in the example program) to give an effective address in the sector map. By reviewing the above formula, the reader may verify that the calculation will yield the values from 0 to 63 decimal or 0 to 77 octal when all the possible coordinate values are considered. When added to the base value (300 in the example) this will yield a low address in the range 300 to 377 (octal).

A portion of the program (to be presented later) will prevent the space ship from moving into any location that has been zeroed-out in the sector map.

The ZERSEC portion of the program is presented next.

```
ZERSEC,   LLI 370      Get X shot value
          LAM          From storage
          SUI 001      Subtract '1'
          RLC
          RLC          Multiply by eight
          RLC
          LDA          Save in register D temporarily
          INL          Get Y shot value
          LAM          From storage
```

```
SUI 001        Subtract '1'
ADD            Add to previous calculations
ORI 300        Form shot table address
LLA            Set low address pointer
LHI 003        And page address of shot table
LMI 000        Zero the entry in shot table
JMP PLAYIN     Continue with game
```

The next portion of the program is the subroutine TRYMOV. This subroutine is the most complicated portion of the program. The subroutine serves the function of attempting to find a new location for the space ship. It does this by attempting to find a sector adjacent to the last position of the space ship that has not been destroyed by a phasor shot. The sector must also be within the boundaries of the 8 by 8 playing grid. Additionally, the direction of movement is accomplished in an essentially random manner so that the player will not be able to detect a reliable pattern of movement for the space ship!

The first few instructions of the subroutine fetch the address that points to the move table. This address is set up each time the player specifies the Y coordinate of a phasor shot (or answers the WANT TO PLAY? query at the beginning of a game). As discussed earlier, the address each time the subroutine is entered will have been selected in an essentially random manner.

The address refers to a position in a table which holds all the possible moves the space ship can make to an adjacent sector. A pictorial near the beginning of this article illustrated the eight possible moves the space ship could make if it was not bounded by the edges of the playing grid, or sectors that had been destroyed. Referral to that pictorial will show that the possible moves may be referenced by a value of -1, 0, or +1 from its present position along each axis. One can convert this information into a table that holds all the possible moves. The table used consists of eight groups of two bytes per group. The first byte in a group holds a move along the X axis. The second stores a move along the Y axis. In the example program, the table is stored in locations 260 through 277 on page 03, and appears as shown on the following page.

```
377     MOVES TABLE (X = -1)
001     Y = +1
000     X =  0
001     Y = +1
001     X = +1
001     Y = +1
377     X = -1
000     Y =  0
001     X = +1
000     Y =  0
377     X = -1
377     Y = -1
000     X =  0
377     Y = -1
001     X = +1
377     Y = -1
```

The initial value of the address to the moves table is transferred from a temporary location in memory to the accumulator at the start of the TRYMOV subroutine. Also, a moves tried counter, which will be maintained in CPU register C during the subroutine, is initialized to a value of eight (decimal). The routine then goes on to the point labeled TRYSEC which marks a looping point within the subroutine.

At TRYSEC, the address value originally in the accumulator is moved to CPU register L to set up the low portion of the address to the moves table. This value is also saved for possible later use in CPU register B. The high address of the moves table is set up in CPU register H, and an X move value fetched from the table. The X move value obtained from the table is then added to the X coordinate value representing the previous position of the space ship to form a new value along the X coordinate. At this point some tests must be made to ensure that the new value is within the boundaries of the playing grid. This is readily accomplished by checking to see that the new coordinate is between the range of 1 to 8 decimal. If the new value is within the playing grid, it is saved in a temporary location in memory. If it is not valid, the program jumps ahead to a routine that will advance the moves table pointer to the next X entry in the table

(by going to the label NOGDX which will advance the address temporarily stored in CPU register B TWO locations!)

If the new X value is O.K., the routine proceeds to advance the pointer to the moves table one location to obtain a Y move. A similar boundary checking procedure is performed again. If the new Y coordinate is valid, it too is saved in a temporary location in memory. If not, the program jumps ahead to the label NOGDY which will advance the moves table pointer just one location to the next X entry.

If the new X and Y coordinates are within the boundaries of the playing grid, the program continues by executing the portion of the subroutine labeled CHECK. This part of the program ascertains whether the new sector the space ship is attempting to move into is available. That is, that it has not been destroyed by a previous phasor shot made by the player! This is readily accomplished by using the new coordinate values to once again calculate a position in the sector map and checking to see that the position has not been zeroed out. The calculation technique is exactly the same as that used by the routine ZERSEC explained earlier. If the sector is available, the program jumps ahead to the routine labeled SAVPOS. If not, the space ship must try to find another position in the grid. This is attempted by proceeding to the point labeled NOGDY.

The portion of the routine beginning with the label NOGDX serves to advance the moves table address pointer to the next X entry in the table. Since the table occupies just 20 (octal) locations in memory (in the example program locations 260 through 277 on page 03), and since the initial value may have been at any even valued address within that range, some special operations must be performed when advancing the pointer to the next X entry. First, it may be necessary to try every possible move in the table. Since the first position tried may have been the eighth entry in the table (four least significant bits of the address equal to 16 octal), one must keep the pointer in the range of 00 to 17 octal (for the four LSB's). This is readily accomplished by a masking operation that removes the four most significant bits. Then, since the table does not reside in the first 20 (octal) locations on a page in memory, the base address value of 260 (in the example) must be tacked back on to form the complete

address value. This procedure will force the moves table pointer to loop back to the value 260 when it reaches a value of 277 rather than going to 300 which would be outside the range of the table.

When the address of the next entry in the moves table has been set up, the routine checks to see if all eight possible moves have been tried by decrementing the counter being maintained in CPU register B. If not, the routine loops back to the point labeled TRYSEC. If, however, all eight possible moves have been tried, then the space ship has not been able to find a new position and it is captured. A message of defeat is then issued to the player. From that point, the program will go back to see if a new game is desired.

The portion of the routine labeled SAVPOS is executed if the space ship successfully finds a new sector to move into. This routine simply moves the X and Y coordinate values being temporarily saved in memory during the TRYMOV subroutine into the storage locations used to hold the new location. (This move actually places the new coordinates into the memory locations that are referenced the next time the message MY LAST LOCATION WAS: is displayed.)

Note that the SAVPOS routine ends with a RET instruction to conclude the subroutine. Thus, a return from the TRYMOV subroutine indicates that the space ship completed a successful move. If the space ship does not find a new sector to move into, the subroutine is not actually completed. Instead, a jump out of the subroutine to start a new game is executed.

The various portions of the TRYMOV subroutine are presented below.

```
TRYMOV,   LHI 001     Set pointer to
          LLI 377     Random counter storage
          LAM         Fetch value
          LCI 010     Set a loop counter

TRYSEC,   LLA         Set pointer to moves table
          LBA         Save pointer in B too
          LHI 003     Set pg pointer to moves table
```

```
        LAM         Fetch an X move
        LHI 001     Change pointer to
        LLI 372     X WAS storage
        ADM         Add X move to form new loc
        CPI 001     Now make boundaries test
        JTS NOGDX   No good if less than one
        CPI 011     Or more than
        JFS NOGDX   Eight decimal
        LHI 001     If OK, save in X temporarily
        LLI 374     Storage location
        LMA         For awhile
        INB         Advance pointer stored in B
        LLB         And load new pointer
        LHI 003     To Y move location
        LAM         Fetch a Y move
        LHI 001     Change pointer to
        LLI 373     Y WAS storage
        ADM         Add Y move to form new loc
        CPI 001     Now make boundaries test
        JTS NOGDY   No good if less than one
        CPI 011     Or more than
        JFS NOGDY   Eight decimal
        LHI 001     If OK, save in Y temporarily
        LLI 375     Storage location
        LMA         For awhile

CHECK,  DCL         Decrement pointer back to
        LAM         X temp storage and fetch
        SUI 001     Subtract '1'
        RLC
        RLC         Multiply by eight
        RLC
        LDA         Save in D temporarily
        INL         Advance pointer to
        LAM         Y temp storage and fetch
        SUI 001     Subtract '1'
        ADD         Add to previous calculations
        ORI 300     Form shot table address
        LLA         Set low address pointer
        LHI 003     And page address pointer
```

```
        LAM         Fetch entry from shot table
        NDA         Set flags, now see if location
        JFZ SAVPOS  Previously fired into
        JMP NOGDY   Try another location if yes

NOGDX,  INB         Advance move table pointer
NOGDY,  INB         As required to get to next
        LAB         X move, move value to ACC
        NDI 017     And make sure it is kept
        ORI 260     In bounds
        DCC         Decrement loop counter
        JFZ TRYSEC  If not 0, try another location
        LHI 001     Else set pointer
        LLI 201     To CAPTURED message
        CAL MSG     Display message
        JMP OVER    See if want new game

SAVPOS, LHI 001     Set pointer to X temporarily
        LLI 374     Storage location
        LDM         Save value in D temporarily
        INL         Advance pointer to Y temp
        LEM         Save in E temporarily
        LLI 372     Change pointer to X WAS
        LMD         Storage and set new value
        INL         Do likewise for Y WAS
        LME         Too!
        RET         Return to calling program
```

That is all there is to the program! That is not so difficult, eh?

An assembled listing of the program for running on a 8008 system will be presented on the following pages. The program in the listing has been assembled to reside in pages 01 to 03 with page 04 reserved for the user's I/O routines. Page 00 and most of page 01 will be used to hold the various message strings mentioned previously. The ASCII data that should be stored in those locations is shown next. (Assuming the user is satisfied with the message strings illustrated.)

000	000	215	212	212	323	320	301	303	305
000	010	323	310	311	320	240	303	301	320
000	020	324	325	322	305	256	240	331	317
000	030	325	240	310	301	326	305	240	261
000	040	265	240	320	310	301	323	317	322
000	050	215	212	323	310	317	324	323	240
000	060	327	311	324	310	240	327	310	311
000	070	303	310	240	324	317	240	304	305
000	100	323	324	322	317	331	240	315	331
000	110	240	324	322	301	326	305	314	215
000	120	212	323	305	303	324	317	322	323
000	130	256	240	311	306	240	301	314	314
000	140	240	315	331	240	301	304	312	301
000	150	303	305	316	324	240	323	305	303
000	160	324	317	322	323	215	212	301	322
000	170	305	240	304	305	323	324	322	317
000	200	331	305	304	240	311	240	301	315
000	210	240	303	301	320	324	325	322	305
000	220	304	256	240	311	306	240	331	317
000	230	325	215	212	310	311	324	240	315
000	240	305	240	317	322	240	322	325	316
000	250	240	317	325	324	240	317	306	240
000	260	320	310	301	323	317	322	240	305
000	270	316	305	322	307	331	254	215	212
000	300	324	310	305	316	240	331	317	325
000	310	240	314	317	323	305	241	215	212
000	320	212	000	000	000	000	215	212	212
000	330	327	301	316	324	240	324	317	240
000	340	320	314	301	331	277	240	240	000
000	350	215	212	212	320	317	317	322	240
000	360	323	320	317	322	324	241	000	215
000	370	212	212	315	331	240	314	301	323
001	000	324	240	320	317	323	311	324	311
001	010	317	316	240	327	301	323	272	240
001	020	240	330	240	275	240	000	254	240
001	030	240	331	240	275	240	000	215	212
001	040	212	331	317	325	240	301	322	305
001	050	240	306	311	322	311	316	307	240
001	060	324	317	272	240	240	330	240	275
001	070	240	000	215	212	212	331	317	325

```
001 100   240 310 311 324 240 315 305 241
001 110   241 240 240 331 317 325 240 314
001 120   317 323 305 241 000 215 212 212
001 130   331 317 325 240 301 322 305 240
001 140   317 325 324 240 317 306 240 320
001 150   310 301 323 317 322 240 305 316
001 160   305 322 307 331 254 240 240 331
001 170   317 325 240 314 317 323 305 241
001 200   000 215 212 212 243 241 260 243
001 210   207 207 207 240 240 304 301 322
001 220   316 241 240 240 331 317 325 240
001 230   310 301 326 305 240 315 305 240
001 240   240 303 240 301 240 320 240 324
001 250   240 325 240 322 240 305 240 304
001 260   240 241 241 000
```

The starting address for each message string may be located from the above data presentation. The beginning of each message string has been underlined. The reader may desire to change some of the messages. If the reader elects to do so, and by so doing changes the starting address of a character string, then the appropriate pointer instruction in the operating portion of the program must be modified accordingly. The assembled program for an 8008 system is presented on the next several pages.

```
001 350   307                    MSG,  LAM
001 351   240                          NDA
001 352   053                          RTZ
001 353   106 200 004                  CAL PRINT
001 356   060                          INL
001 357   110 350 001                  JFZ MSG
001 362   050                          INH
001 363   104 350 001                  JMP MSG

001 370   000                          000
001 371   000                          000
001 372   000                          000
001 373   000                          000
```

```
001 374   000                          000
001 375   000                          000
001 376   000                          000
001 377   000                          000

002 000   056 000              START,  LHI 000
002 002   066 000                      LLI 000
002 004   106 350 001                  CAL MSG

002 007   056 000               OVER,  LHI 000
002 011   066 325                      LLI 325
002 013   106 350 001                  CAL MSG

002 016   106 000 004          INAGN,  CAL CKINP
002 021   240                          NDA
002 022   122 020 004                  CFS INPUTN
002 025   060                          INL
002 026   074 316                      CPI 316
002 030   110 043 002                  JFZ NOTNO
002 033   056 000                      LHI 000
002 035   066 350                      LLI 350
002 037   106 350 001                  CAL MSG
002 042   000                          HLT

002 043   074 331              NOTNO,  CPI 331
002 045   110 016 002                  JFZ INAGN
002 050   306                          LAL
002 051   044 007                      NDI 007
002 053   004 001                      ADI 001
002 055   056 001                      LHI 001
002 057   066 372                      LLI 372
002 061   370                          LMA
002 062   060                          INL
002 063   370                          LMA
002 064   066 377                      LLI 377
002 066   044 007                      NDI 007
002 070   002                          RLC
002 071   064 260                      ORI 260
002 073   370                          LMA
002 074   056 001                      LHI 001
```

```
002 076    066 376                      LLI 376
002 100    076 020                      LMI 020
002 102    056 003                      LHI 003
002 104    066 300                      LLI 300
002 106    006 377                      LAI 377

002 110    370                  FILOOP, LMA
002 111    060                          INL
002 112    110 110 002                  JFZ FILOOP

002 115    056 000              PLAYIN, LHI 000
002 117    066 367                      LLI 367
002 121    106 350 001                  CAL MSG
002 124    056 001                      LHI 001
002 126    066 372                      LLI 372
002 130    307                          LAM
002 131    064 260                      ORI 260
002 133    106 200 004                  CAL PRINT
002 136    056 001                      LHI 001
002 140    066 026                      LLI 026
002 142    106 350 001                  CAL MSG
002 145    056 001                      LHI 001
002 147    066 373                      LLI 373
002 151    307                          LAM
002 152    064 260                      ORI 260
002 154    106 200 004                  CAL PRINT
002 157    106 002 003                  CAL TRYMOV
002 162    056 001                      LHI 001
002 164    066 376                      LLI 376
002 166    317                          LBM
002 167    011                          DCB
002 170    371                          LMB
002 171    110 206 002                  JFZ CONTIN

002 174    056 001              PHASOR, LHI 001
002 176    066 125                      LLI 125
002 200    106 350 001                  CAL MSG
002 203    104 007 002                  JMP OVER

002 206    056 001              CONTIN, LHI 001
```

```
002 210   066 036                    LLI 036
002 212   106 350 001                CAL MSG

002 215   106 020 004          INX,  CAL INPUTN
002 220   074 261                    CPI 261
002 222   160 215 002                JTS INX
002 225   074 271                    CPI 271
002 227   120 215 002                JFS INX
002 232   056 001                    LHI 001
002 234   066 370                    LLI 370
002 236   044 017                    NDI 017
002 240   370                        LMA
002 241   056 001                    LHI 001
002 243   066 026                    LLI 026
002 245   106 350 001                CAL MSG

002 250   106 000 004          INY,  CAL CKINP
002 253   240                        NDA
002 254   122 020 004                CFS INPUTN
002 257   060                        INL
002 260   074 261                    CPI 261
002 262   160 250 002                JTS INY
002 265   074 271                    CPI 271
002 267   120 250 002                JFS INY
002 272   316                        LBL
002 273   056 001                    LHI 001
002 275   066 371                    LLI 371
002 277   044 017                    NDI 017
002 301   370                        LMA
002 302   301                        LAB
002 303   044 007                    NDI 007
002 305   002                        RLC
002 306   064 260                    ORI 260
002 310   056 001                    LHI 001
002 312   066 377                    LLI 377
002 314   370                        LMA

002 315   056 001          HITEST,   LHI 001
002 317   066 370                    LLI 370
002 321   307                        LAM
```

```
002 322   060                          INL
002 323   060                          INL
002 324   277                          CPM
002 325   110 352 002                  JFZ ZERSEC
002 330   061                          DCL
002 331   307                          LAM
002 332   060                          INL
002 333   060                          INL
002 334   277                          CPM
002 335   110 352 002                  JFZ ZERSEC

002 340   056 001              BOMB,   LHI 001
002 342   066 072                      LLI 072
002 344   106 350 001                  CAL MSG
002 347   104 007 002                  JMP OVER

002 352   066 370              ZERSEC, LLI 370
002 354   307                          LAM
002 355   024 001                      SUI 001
002 357   002                          RLC
002 360   002                          RLC
002 361   002                          RLC
002 362   330                          LDA
002 363   060                          INL
002 354   307                          LAM
002 365   024 001                      SUI 001
002 367   203                          ADD
002 370   064 300                      ORI 300
002 372   360                          LLA
002 373   056 003                      LHI 003
002 375   076 000                      LMI 000
002 377   104 115 002                  JMP PLAYIN

003 002   056 001              TRYMOV, LHI 001
003 004   066 377                      LLI 377
003 006   307                          LAM
003 007   026 010                      LCI 010

003 011   360                  TRYSEC, LLA
003 012   310                          LBA
```

```
003 013   056 003                 LHI 003
003 015   307                     LAM
003 016   056 001                 LHI 001
003 020   066 372                 LLI 372
003 022   207                     ADM
003 023   074 001                 CPI 001
003 025   160 125 003             JTS NOGDX
003 030   074 011                 CPI 011
003 032   120 125 003             JFS NOGDX
003 035   056 001                 LHI 001
003 037   066 374                 LLI 374
003 041   370                     LMA
003 042   010                     INB
003 043   361                     LLB
003 044   056 003                 LHI 003
003 046   307                     LAM
003 047   056 001                 LHI 001
003 051   066 373                 LLI 373
003 053   207                     ADM
003 054   074 001                 CPI 001
003 056   160 126 003             JTS NOGDY
003 061   074 011                 CPI 011
003 063   120 126 003             JFS NOGDY
003 066   056 001                 LHI 001
003 070   066 375                 LLI 375
003 072   370                     LMA

003 073   061             CHECK,  DCL
003 074   307                     LAM
003 075   024 001                 SUI 001
003 077   002                     RLC
003 100   002                     RLC
003 101   002                     RLC
003 102   330                     LDA
003 103   060                     INL
003 104   307                     LAM
003 105   024 001                 SUI 001
003 107   203                     ADD
003 110   064 300                 ORI 300
003 112   360                     LLA
```

```
003 113   056 003                    LHI 003
003 115   307                        LAM
003 116   240                        NDA
003 117   110 152 003                JFZ SAVPOS
003 122   104 126 003                JMP NOGDY

003 125   010                NOGDX,  INB
003 126   010                NOGDY,  INB
003 127   301                        LAB
003 130   044 017                    NDI 017
003 132   064 260                    ORI 260
003 134   021                        DCC
003 135   110 011 003                JFZ TRYSEC
003 140   056 001                    LHI 001
003 142   066 201                    LLI 201
003 144   106 350 001                CAL MSG
003 147   104 007 002                JMP OVER

003 152   056 001            SAVPOS, LHI 001
003 154   066 374                    LLI 374
003 156   337                        LDM
003 157   060                        INL
003 160   347                        LEM
003 161   066 372                    LLI 372
003 163   373                        LMD
003 164   060                        INL
003 165   374                        LME
003 166   007                        RET


003 260   377                        377
003 261   001                        001
003 262   000                        000
003 263   001                        001
003 264   001                        001
003 265   001                        001
003 266   377                        377
003 267   000                        000
003 270   001                        001
003 271   000                        000
```

```
003 272    377                    377
002 273    377                    377
003 274    000                    000
003 275    377                    377
003 276    001                    001
003 277    377                    377

004 000                    CKINP,

004 020                    INPUTN,

004 200                    PRINT,
```

Do not forget that the program as presented will be using locations 300 through 377 on page 03 as a sector map. The reader should also make sure that the user provided I/O routines are loaded into memory at the indicated locations before attempting to operate the program!

OPERATING THE SPACE CAPTURE PROGRAM

Once the program has been loaded into memory it is ready for operation. The program is started by executing a jump to location 000 on page 02 for the illustrated program, and placing the computer in the normal program execution run mode. From there on the program effectively guides the player. The program will continue to operate, playing game after game, until the player responds with a N for NO to the WANT TO PLAY? query.

The player will want to have a supply of paper with 8 by 8 grids marked out to keep track of the space ship's movements and sectors in which shots have been fired as the games progresses. If the game is to be used frequently, it is probably worthwhile to make up a good supply of the grid forms using a mimeograph or duplicating machine.

In case the reader has any doubts as to how the game is played,

the following illustrates an actual game played using the program. At the end of the illustration showing the dialogue between the computer and the player is a grid illustrating how the game progressed. The progress of the space ship is shown as a series of arrows indicating the direction of each movement. The phasor shots fired by the player are shown as a circled number in various sectors. The number refers to the actual shot number as the game progressed.

```
SPACESHIP CAPTURE. YOU HAVE 15 PHASOR
SHOTS WITH WHICH TO DESTROY MY TRAVEL
SECTORS. IF ALL MY ADJACENT SECTORS ARE
DESTROYED, I AM CAPTURED. IF YOU HIT ME
OR RUN OUT OF PHASOR ENERGY, THEN
YOU LOSE!

WANT TO PLAY?        Y

MY LAST POSITION WAS:    X= 7,    Y = 7

YOU ARE FIRING TO:    X = 7,    Y = 7

MY LAST POSITION WAS:    X = 8,    Y = 6

YOU ARE FIRING TO:    X = 6,    Y = 6

MY LAST POSITION WAS:    X = 8,    Y = 7

YOU ARE FIRING TO:    X = 6,    Y = 8

MY LAST POSITION WAS:    X = 8,    Y = 6

YOU ARE FIRING TO:    X = 6,    Y = 5

MY LAST POSITION WAS:    X = 8,    Y = 5

YOU ARE FIRING TO:    X = 6,    Y = 4

MY LAST POSITION WAS:    X = 7,    Y = 6
```

YOU ARE FIRING TO: X = 6, Y = 7

MY LAST POSITION WAS: X = 8, Y = 7

YOU ARE FIRING TO: X = 7, Y = 4

MY LAST POSITION WAS: X = 7, Y = 6

YOU ARE FIRING TO: X = 8, Y = 4

MY LAST POSITION WAS: X = 8, Y = 6

YOU ARE FIRING TO: X = 8, Y = 6

MY LAST POSITION WAS: X = 7, Y = 5

YOU ARE FIRING TO: X = 8, Y = 7

MY LAST POSITION WAS: X = 7, Y = 6

YOU ARE FIRING TO: X = 7, Y = 6

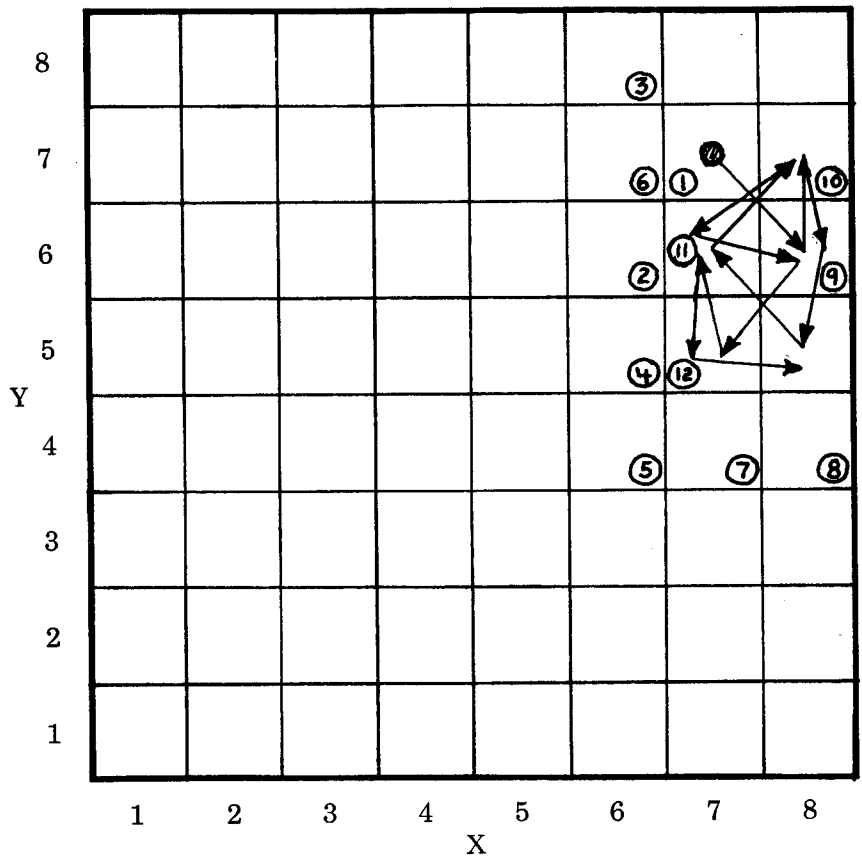
MY LAST POSITION WAS: X = 7, Y = 5

YOU ARE FIRING TO: X = 7, Y = 5

MY LAST POSITION WAS: X = 8, Y = 5

#!0#! DARN! YOU HAVE ME CAPTURED!!

WANT TO PLAY?



PICTORIAL OF THE MOVES MADE IN THE ILLUSTRATIVE SPACE CAPTURE GAME

The following is a listing of the program for an 8080 system. Only a few minor changes have been made in the program. Notably, the inclusion of stack pointer initializing instructions (required by the 8080 since it does not have a program counter stack on the CPU chip) at the labels START and OVER. Additionally, the double register (H and L) load instruction has been utilized when applicable instead of the individual commands required in an 8008 unit. Several other minor changes have been made to make use of the more powerful 8080 instruction set, but the basic structure of the program has not been altered so that the explanations of the various routines made earlier need not be elaborated upon.

```
001 350    176                    MSG,  LAM
001 351    247                          NDA
001 352    310                          RTZ
001 353    315 200 004                  CAL PRINT
001 356    043                          INXH
001 357    303 350 001                  JMP MSG

001 370    000                          000
001 371    000                          000
001 372    000                          000
001 373    000                          000
001 374    000                          000
001 375    000                          000
001 376    000                          000
001 377    000                          000

002 000    061 350 001          START,  LXS 350 001
002 003    041 000 000                  LXH 000 000
002 006    315 350 001                  CAL MSG

002 011    061 350 001          OVER,   LXS 350 001
002 014    041 325 000                  LXH 325 000
002 017    315 350 001                  CAL MSG
```

```
002 022   315 000 004   INAGN,  CAL CKINP
002 025   247                   NDA
002 026   364 020 004           CFS INPUTN
002 031   054                   INL
002 032   376 316               CPI 316
002 034   302 046 002           JFZ NOTNO
002 037   041 350 000           LXH 350 000
002 042   315 350 001           CAL MSG
002 045   166                   HLT

002 046   376 331       NOTNO,  CPI 331
002 050   302 022 002           JFZ INAGN
002 053   175                   LAL
002 054   346 007               NDI 007
002 056   306 001               ADI 001
002 060   041 372 001           LXH 372 001
002 063   167                   LMA
002 064   054                   INL
002 065   167                   LMA
002 066   056 377               LLI 377
002 070   346 007               NDI 007
002 072   007                   RLC
002 073   366 260               ORI 260
002 075   167                   LMA
002 076   041 376 001           LXH 376 001
002 101   066 020               LMI 020
002 103   041 300 003           LXH 300 003
002 106   076 377               LAI 377

002 110   167           FILOOP, LMA
002 111   054                   INL
002 112   302 110 002           JFZ FILOOP

002 115   041 367 000 PLAYIN,   LXH 367 000
002 120   315 350 001           CAL MSG
002 123   041 372 001           LXH 372 001
002 126   176                   LAM
002 127   366 260               ORI 260
002 131   315 200 004           CAL PRINT
```

```
002 134   041 026 001            LXH 026 001
002 137   315 350 001            CAL MSG
002 142   041 373 001            LXH 373 001
002 145   176                    LAM
002 146   366 260                ORI 260
002 150   315 200 004            CAL PRINT
002 153   315 363 002            CAL TRYMOV
002 156   041 376 001            LXH 376 001
002 161   065                    DCM
002 162   302 176 002            JFZ CONTIN

002 165   041 125 001   PHASOR,  LXH 125 001
002 170   315 350 001            CAL MSG
002 173   303 011 002            JMP OVER

002 176   041 036 001   CONTIN,  LXH 036 001
002 201   315 350 001            CAL MSG

002 204   315 020 004      INX,  CAL INPUTN
002 207   376 261                CPI 261
002 211   372 204 002            JTS INX
002 214   376 271                CPI 271
002 216   362 204 002            JFS INX
002 221   041 370 001            LXH 370 001
002 224   346 017                NDI 017
002 226   167                    LMA
002 227   041 026 001            LXH 026 001
002 232   315 350 001            CAL MSG

002 235   315 000 004      INY,  CAL CKINP
002 240   247                    NDA
002 241   364 020 004            CFS INPUTN
002 244   054                    INL
002 245   376 261                CPI 261
002 247   372 235 002            JTS INY
002 252   376 271                CPI 271
002 254   362 235 002            JFS INY
002 257   105                    LBL
002 260   041 371 001            LXH 371 001
002 263   346 017                NDI 017
```

```
002 265   167                        LMA
002 266   170                        LAB
002 267   346 007                    NDI 007
002 271   007                        RLC
002 272   366 260                    ORI 260
002 274   041 377 001                LXH 377 001
002 277   167                        LMA

002 300   041 370 001        HITEST, LXH 370 001
002 303   176                        LAM
002 304   054                        INL
002 305   054                        INL
002 306   276                        CPM
002 307   302 333 002                JFZ ZERSEC
002 312   055                        DCL
002 313   176                        LAM
002 314   054                        INL
002 315   054                        INL
002 316   276                        CPM
002 317   302 333 002                JFZ ZERSEC

002 322   041 072 001          BOMB, LXH 072 001
002 325   315 350 001                CAL MSG
002 330   303 011 002                JMP OVER

002 333   056 370              ZERSEC, LLI 370
002 335   176                        LAM
002 336   326 001                    SUI 001
002 340   007                        RLC
002 341   007                        RLC
002 342   007                        RLC
002 343   127                        LDA
002 344   054                        INL
002 345   176                        LAM
002 346   326 001                    SUI 001
002 350   202                        ADD
002 351   366 300                    ORI 300
002 353   157                        LLA
002 354   046 003                    LHI 003
002 356   066 000                    LMI 000
```

```
002 360   303 115 002                  JMP PLAYIN

002 363   041 377 001        TRYMOV,   LXH 377 001
002 366   176                          LAM
002 367   016 010                      LCI 010

002 371   157                TRYSEC,   LLA
002 372   107                          LBA
002 373   046 003                      LHI 003
002 375   176                          LAM
002 376   041 372 001                  LXH 372 001
003 001   206                          ADM
003 002   376 001                      CPI 001
003 004   372 101 003                  JTS NOGDX
003 007   376 011                      CPI 011
003 011   362 101 003                  JFS NOGDX
003 014   041 374 001                  LXH 374 001
003 017   167                          LMA
003 020   004                          INB
003 021   150                          LLB
003 022   046 003                      LHI 003
003 024   176                          LAM
003 025   041 373 001                  LXH 373 001
003 030   206                          ADM
003 031   376 001                      CPI 001
003 033   372 102 003                  JTS NOGDY
003 036   376 011                      CPI 011
003 040   362 102 003                  JFS NOGDY
003 043   041 375 001                  LXH 375 001
003 046   167                          LMA

003 047   055                CHECK,    DCL
003 050   176                          LAM
003 051   326 001                      SUI 001
003 053   007                          RLC
003 054   007                          RLC
003 055   007                          RLC
003 056   127                          LDA
003 057   054                          INL
003 060   176                          LAM
```

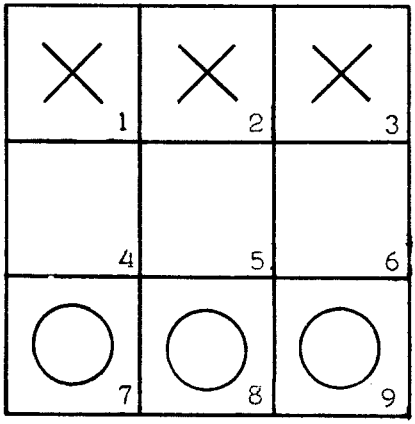
```
003 061    326 001                      SUI 001
003 063    202                          ADD
003 064    366 300                      ORI 300
003 066    157                          LLA
003 067    046 003                      LHI 003
003 071    176                          LAM
003 072    247                          NDA
003 073    302 125 003                  JFZ SAVPOS
003 076    303 102 003                  JMP NOGDY

003 101    004                 NOGDX,   INB
003 102    004                 NOGDY,   INB
003 103    170                          LAB
003 104    346 017                      NDI 017
003 106    366 260                      ORI 260
003 110    015                          DCC
003 111    302 371 002                  JFZ TRYSEC
003 114    041 201 001                  LXH 201 001
003 117    315 350 001                  CAL MSG
003 122    303 011 002                  JMP OVER

003 125    041 374 001         SAVPOS,  LXH 374 001
003 130    126                          LDM
003 131    054                          INL
003 132    136                          LEM
003 133    056 372                      LLI 372
003 135    162                          LMD
003 136    054                          INL
003 137    163                          LME
003 140    311                          RET



003 260    377                          377
003 261    001                          001
003 262    000                          000
003 263    001                          001
003 264    001                          001
003 265    001                          001
003 266    377                          377
```

```
003 267    000                    000
003 270    001                    001
003 271    000                    000
003 272    377                    377
003 273    377                    377
003 274    000                    000
003 275    377                    377
003 276    001                    001
003 277    377                    377

004 000                  CKINP,

004 020                  INPUTN,

004 200                  PRINT,
```

HEXPAWN - A MINI CHESS GAME

The possibility of playing a game of chess against a computer has undoubtably crossed the minds of most people that have had exposure to computers in one way or another. However, the game's near-limitless number of possible board configurations and moves makes it impossible to program on a small computer system. An alternative is to simplify the game to allow it to be programmed for the small computer system. Hexpawn is one such game.

Hexpawn consists of a 3 x 3 playing board and six pawns, three pawns for each player. The starting configuration is illustrated below. The pawns move in a manner similar to their moves in chess. A pawn can move one square forward, provided the square it is moving to is vacant, or one square diagonally to capture an opponent's pawn. A diagonal move cannot be made if an opponent's pawn is not captured by the move. The object of the game is to move a pawn to the opponent's side of the board while blocking the opponent from doing so, or capture all of the opponent's pawns. A game is a draw when no one can make a legal move.



The game starts by the current board configuration being printed followed by a request for a human to enter the first move. Each move is made by entering the number of the square which contains the pawn to be moved, followed by the number of the square to

which the pawn is to be moved. The computer then makes its move, and prints the new board configuration. It then waits for the human's next move. After each move by the human and by the computer, the board is examined to determine if the game has been won by either side. When this occurs, an appropriate message is printed to indicate the end of the current game, and a new game is started. Should the human make an illegal move, the computer rejects the move and requests a new one.

As one can see, the game is fairly simple and requires no more than three or four moves by each side to complete. Thus, to make the game interesting, the program is written to provide the computer with ARTIFICIAL INTELLIGENCE. The computer is given the ability to decide which move it should make in an effort to win the game. That is, after each move is made by the human player, the computer examines the board and decides which, of all possible moves, it will make. If a move is made by the computer which results in the computer losing the game, that move is noted as an undesired move which should not be made again when that same board configuration if encountered. Thus, the computer learns from its mistakes, and eventually becomes so efficient in its ability to play the game that the best one can hope for is to play to a draw with the computer.

This version of Hexpawn is written to reside in five pages (256 bytes per page) of an 8008 or 8080 microcomputer system. The program may be reduced somewhat be revising or removing some of the text messages, if the user is limited in the amount of memory available. There are also several portions of the program which could be rearranged into subroutines, allowing for further compression of the program. Also, the table which is used to restore the program to its initial state of ignorance may be deleted along with the associated restoration program steps. If this is done, the program can be restored by simply reloading the program into memory. Making such changes to the program can reduce the memory required to less than four pages. The program was written to make it easy to follow the logic rather than minimize the amount of memory required. The reader can see that the memory required is, however, considerably less than that needed to duplicate the same game using a higher level language.

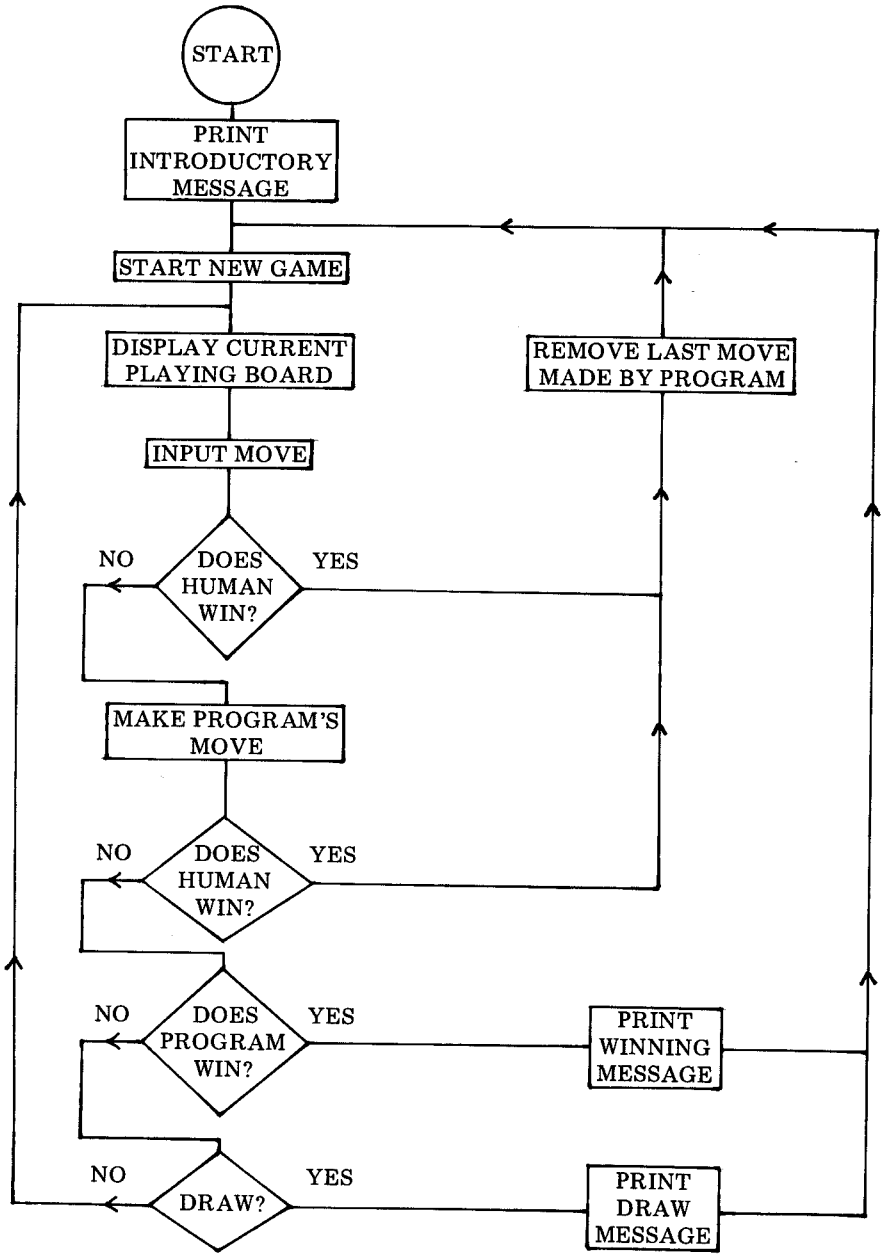
The flow chart on the following page illustrates the basic flow of the program. As one may observe, the game progresses in the same manner as a chess game. Each side makes one move at a time, and checks the board at the completion of each move to determine whether there is a winner. A verbal description of the flow chart will now be presented.

FUNDAMENTAL OPERATION OF THE HEXPAWN PROGRAM

The program starts by printing an introductory message which describes the operation of the game for a person that is playing the game for the first time. A game is then started by displaying the playing board along with the opening positions of the pawns for the human player to examine.

The program next requests the human to input a move. When the move has been received, it is checked to determine whether the player's move was to the opposite side of the board. That would indicate the challenger had won the game. If not, the move is entered on the current board. The program then examines the board and determines which move it will make in response to the player's move. If the human's move was a winning move, the program will remove the last move that it made from its list of possible moves. Since the last move that the computer made resulted in a win by the human, it does not want to make the same mistake twice. One may note that it requires at least two moves by a human to win a game, so that there will always be an initial move by the program which can be deleted.

When the program examines the current playing board, it selects a move from a list of possible moves which it may make for the current board configuration. If this list has had all of its entries removed, because the human has won as a result of making those moves, the game is conceded to the human and the move that the program just made will be removed. Otherwise, the program makes the move indicated in the list. It then determines whether its move has won the game or has resulted in a draw (all remaining pawns are blocked from making a move). If the game is a win or draw, an appropriate message is printed. A new game is then started. If not, the game is continued and the program returns to print the current


START
PRINT INTRODUCTORY MESSAGE
START NEW GAME
DISPLAY CURRENT PLAYING BOARD
INPUT MOVE
DOES HUMAN WIN?
NO
YES
MAKE PROGRAM'S MOVE
DOES HUMAN WIN?
NO
YES
DOES PROGRAM WIN?
NO
YES
PRINT WINNING MESSAGE
DRAW?
NO
YES
PRINT DRAW MESSAGE
REMOVE LAST MOVE MADE BY PROGRAM

playing board for the player to examine.

TABLES USED BY THE HEXPAWN PROGRAM

The most important portion of a program such as Hexpawn is that which decides what move is to be made for a specific board configuration. This operation is performed through the use of four tables in this program. These tables are used to: Find the matching board configuration, direct the program to the list of possible moves for each configuration, select the move to be made, and provide the actual codes for making the move. Each of these tables are presented in the listing at the end of this chapter. Due to the size of these tables, only sample entries of each will be presented in the following discussion.

The MODEL table is a table used by the program to determine the current board configuration. It consists of 33 pairs of bytes which define all the possible board configurations immediately following a human's move. The first byte of each pair indicates the positions of the program's pawns on the board. For each pawn in a square, the bit corresponding to that square is set to '1.' If a pawn is not in a square, the bit corresponding to that square is '0.' The squares of the board defined in the first byte are as follows:

BIT POSITION	B7	B6	B5	B4	B3	B2	B1	B0
BOARD POSITION	1	2	3	4	5	6	X	X

The reader may note that bits B1 and B0 do not have any position on the playing board defined for them. The reason for this is that if any of the program's pawns reach position 7, 8, or 9, the game will be over with the computer winning. It will not be necessary to store the fact that the program's pawn has reached the last row. Consequently, bits B1 and B0 will always be set to a zero. The same is true for the human's pawns, which are defined by the following bit definitions in the second byte of the model pair.

BIT POSITION	B7	B6	B5	B4	B3	B2	B1	B0
BOARD POSITION	X	X	4	5	6	7	8	9

Bits B7 and B6 are not defined for any position on the board, since a move to position 1, 2, or 3 is a winning move for the human. Bits B7 and B6 of the second byte are always a zero.

For example, suppose the first move made by the human was from square 7 to square 4. The current board configuration would be represented by the following byte pair (using octal notation):

PGM'S PAWNS	340
HUMAN'S PAWNS	043

This corresponds to a physical board configuration of:

```
X|X|X
O|  |
 |O|O
```

When the program finds the byte pair in the model table that matches the current board, it sets up a pointer to the MODEL-TO-MOVE INDEX table. The model-to-move index table consists of a list of pointers. Each entry in this table points of a list of possible moves for the current board configuration. The list of moves is contained in the MOVE INDEX table.

The move index table contains a list of moves which may be made for each of the 33 models in the model table. Each list contains from one to three numbers which indicate a possible move. Each list is terminated by a 200 octal entry. The move numbers range in value from 1 to 15, and indicate possible moves taken from the list on the following page.

The move number is a number which is contained in the move index table. The number in the FROM column is the square on the playing board that the program's pawn is to be moved from. The TO column contains the square that the program's pawn is to be moved to. The result of the move is indicated in the last column. When a number of a move is read from the move index table, it is used to set up a pointer to the MOVE table.

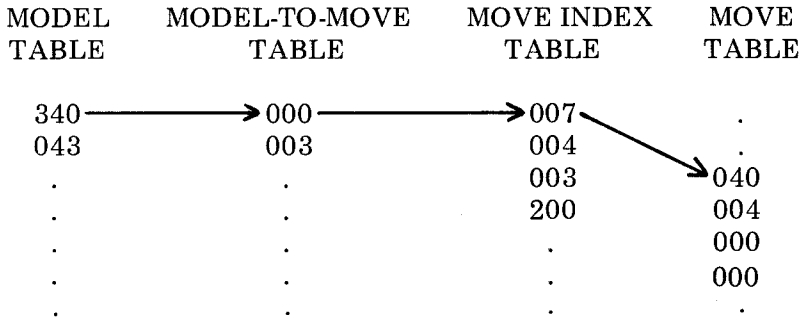
The move table contains a four byte grouping for each of the

MOVE NO.	FROM	TO	RESULT
1	1	4	---
2	1	5	CAPTURE
3	2	4	CAPTURE
4	2	5	---
5	2	6	CAPTURE
6	3	5	CAPTURE
7	3	6	---
8	4	7	COMPUTER WINS
9	4	8	COMPUTER WINS
10	5	7	COMPUTER WINS
11	5	8	COMPUTER WINS
12	5	9	COMPUTER WINS
13	6	8	COMPUTER WINS
14	6	9	COMPUTER WINS
15	-	-	DRAW

moves in the table above. The first byte has one bit set. This bit represents the original position of the program's pawn which is to be moved (as discussed previously). The second byte has a bit set to represent the position to which the program's pawn will be moved. The third byte indicates the position of the player's pawn which will be captured by the move, if the move results in a capture. If no capture will be made, the third byte will be all zeros. The fourth byte depicts the result of the move. If the fourth byte is zero, the program will continue with the game by requesting a move by the human. If the move results in the computer winning, the fourth byte will contain the ASCII code for the number of the square the computer will move into to win the game. This code is used in setting up the winning message. If the move will result in a draw, the fourth byte will contain the octal value 100. The following example contains the four byte group used to define move number three. The reader may note that bytes 2 and 3 both correspond to square 4 of the playing board. Byte 2 indicates the program's pawn moving to square 4, and byte 3 indicates the human's pawn at square 4 being captured by the move.

MOVE NO. 3	BYTE NO. 1	100
	BYTE NO. 2	020
	BYTE NO. 3	040
	BYTE NO. 4	000

The move index table is the means by which the program learns to play the game. Each time a move is selected from the move index table that location in the table is saved. After every move, a test for a possible win by the human is made. If the human wins, the location in the move index table which was saved by the program is zeroed. Once zeroed, the program will not be able to make the same move. The program will skip that location and make the next move indicated in the list. For example, suppose in the first game the player's first move was from 7 to 4. The program would find the first model in the model table as a match. It would fetch the pointer from the first location in the model-to-move index table and select move 7 as its first move. This sequence is illustrated below.



From this sequence it may be observed that the program has moved from square 3 to 6. This move allows the human to win easily by moving from square 4 to 2 thus capturing the program's pawn. The program then removes 007 from the list of moves for the first model by loading a 000 in the first location of that list. Now, if the human makes the same opening move of square 7 to square 4, the program will skip the first location in the list of moves for the first model, since it is zero. It will then make the second move, which is

move number 4. Thus, the program has learned that move number 7 was not the proper move to make for that particular board model.

Another manner in which the program learns is when all the moves for a specific model have been zeroed. When the program searches the move index table for a move, and it reaches the 200 byte, which terminates the list, it concedes the game to the human. The program knows that on the next move, no matter what move it makes, it will lose the game. At this point the program will go back to the previous move that it made before getting into the predicament and zero it in the move index table. In this way, the program is prevented from making the same move which brought it to the point of having to concede the game.

TEXT MESSAGES USED IN HEXPAWN

There are several messages used in Hexpawn to inform the human player of the setup of the game, to indicate when the human has input an illegal move, to display the current board configuration, and to signify the outcome of the game. These messages are of variable length, and may require more than one line of output. The content of these messages may be altered by the reader, if desired, to reflect greater emotion by the program at winning and losing. The messages presented here were kept fairly low key to conserve memory space, since the operating program and associated tables alone require almost 1K bytes of memory. The message strings are presented next.

"HERE'S THE BOARD
1|2|3
4|5|6
7|8|9
I'M X, YOU'RE O
MOVE ONE SQUARE FORWARD IF VACANT
OR ONE SQUARE DIAGONALLY TO CAPTURE
YOU START."

"NO ! NO !"

"I CONCEDE!"

"YOU WIN."

"NO GOOD! MUST START AGAIN"

"DRAW, NO ONE WINS."

"I MOVE TO .
I WIN! YOU LOSE!"

"I WIN! YOU HAVE NO PAWNS."

These text messages are stored as a continuous string of ASCII characters with each message terminated by a zero byte. The playing board, however, is stored on the same page as the model table and temporary data, as it is updated after each move by the program. The board output, FROM and TO messages are stored as shown next.

```
"X ! X ! X
    !   !
 O ! O ! O
 FM " " " TO "
```

The program uses a common subroutine to output these messages to the user's output device. This subroutine is called with memory pointer registers H and L set to the starting address of a message. It fetches each character from the storage area and presents it to the user's output routine in the accumulator. When it encounters the zero byte, it returns to the calling program. The listing for this routine is presented next. It is labeled MSG.

```
MSG,    LAM         Fetch character to print
        NDA         End of message?
        RTZ         Yes, return
        CAL PRINT   No, print character
        INL         Incr low addr msg pointer
        JFZ MSG     If non-0, continue output
        INH         Else, incr page addr pointer
        JMP MSG
```

As one can see, the MSG subroutine is quite straight-forward. It simply fetches characters from memory starting at the location set up by the calling program in the H and L registers, and calls the user provided subroutine PRINT to output the character to the system output device. The PRINT subroutine must take the character in the accumulator and perform whatever is required to output that character to the printer or display device. This routine is free to use registers B through E in outputting the character. The only requirement is that if registers H and L must be used, they must be restored to their initial value before returning to the MSG subroutine.

The MSG subroutine is called to output every message by this program. Therefore, in order to change the messages, the reader simply stores the ASCII codes for the messages desired in a continuous string in memory, and then stores a zero byte to terminate the message. To output a message, the program sets the pointer to the starting address of the message and calls the MSG subroutine.

THE HEXPAWN PROGRAM

The reader may refer to the flow chart presented previously during the following discussion of the operating portion of the program.

The first part of the Hexpawn program outputs the introductory message and resets the move index table to its initial state of intelligence. The move index table is initialized by transferring the contents of the RESTORE MOVE list into the ACTIVE MOVE list. The restore move list is a copy of the move index table with all the possible moves contained in it. The active move list is actually the move index table, which will have its contents zeroed as the program learns to play the game. This restore routine transfers the upper half of page 04, which contains the restore list and the beginning of the text messages, down to the lower half of page 04, which contains the active move list, or move index table. The actual list, however, only requires approximately 3/8 of the page. This should be noted so that one does not try to store messages in the area from 137 through 177 on page 04.

This restore routine is performed only when the program is started at the beginning. After each game is completed, the program returns

to the next section, starting at the label AGAIN so that the move index table will not be reset. The only way that the move index table can be reset is for the operator to restart the program.

The listing for the initial portion of the Hexpawn program is presented below.

```
START,  LLI 076      Set pointer to intro. msg
        LHI 005
        CAL MSG      Print introduction
        LHI 004      Set pointer to move index pg
        LDI 000      Init. actv move list pointer
        LEI 200      Init. rstr move list pointer
RSTR,   LLE          Set restore list pointer
        LAM          Fetch restore list entry
        LLD          Set pointer to active list
        LMA          Store entry in active list
        IND          Increment active list pointer
        INE          Increment restore list pointer
        JFZ RSTR     Done? No, cont. transfer
```

The next section of the program is the one which prepares the board output to display the current setup of the playing board. There are two points at which the program enters this routine. The first is at the instruction labeled AGAIN. This entry point resets the playing board, as stored in locations 03 000 for the X pawns, and 03 001 for the O pawns. This is the starting setup as shown in the figure on the first page of this chapter. It then proceeds to the other entry point of this routine. The second entry point is labeled PBD. This point is entered when the program is in the middle of a game. This portion of the routine sets up the board output message to display the current positions of the pawns in the following manner.

First, the board output message is cleared by storing space characters, ASCII code 240, at the locations in the board output that represent the possible pawn positions. The current positions of the X pawns and O pawns are then determined by the subroutines STX and STO.

When the STX subroutine is first called, the accumulator contains

the current X board as stored in location 03 000. This is rotated left one bit to load the CARRY with the condition of square 1 with respect to the presence of an X pawn. If the carry is set to 1, the ASCII code for an X is stored in the location of square 1 in the board output message. If the carry is reset to 0, the routine simply returns and the contents of square 1 remain a space character. The next time STX is called, the accumulator contains the current X board rotated to the left one bit so that when it is rotated to the left again the carry will indicate the presence, or absence, of an X pawn for square 2. Each time the STX routine is called, the memory pointer registers H and L are set to indicate the location in the board output message where the X is to be stored if it is present at that location. The STO subroutine stores the ASCII code for the 0 character in the locations in the board output message where there are 0 pawns present in a manner similar to the STX subroutine. When the board output message is set to reflect the current board set up, the MSG subroutine is called to display the board for the player to examine. The following is the listing of this board set up and display routine.

```
AGAIN,  LLI 000         Set pointer to current board
        LHI 003
        LMI 340         Set board to starting setup
        INL
        LMI 007
PBD,    LLI 302         Set pointer to brd printout
        LBI 240         Set space char to clear board
        LMB             Store space in '1'
        LLI 304
        LMB             Store space in '2'
        LLI 306
        LMB             Store space in '3'
        LLI 311
        LMB             Store space in '4'
        LLI 313
        LMB             Store space in '5'
        LLI 315
        LMB             Store space in '6'
        LLI 320
        LMB             Store space in '7'
        LLI 322
```

```
LMB         Store space in '8'
LLI 324
LMB         Store space in '9'
LLI 000     Set pointer to current X board
LAM         Fetch X board
INL         Advance to O board
LBM         Fetch current O board
LLI 302     Set pointer to 1 position
CAL STX     If X here, store character
LLI 304     Set pointer to 2 position
CAL STX     If X here, store character
LLI 306     Set pointer to 3 position
CAL STX     If X here, store character
LLI 311     Set pointer to 4 position
CAL STX     If X here, store character
LCA         Save X board
LAB         Fetch O board
RLC         Position to 4
RLC
CAL STO     If O here, store character
LBA         Save O board
LLI 313     Set pointer to 5 position
LAC         Fetch X board
CAL STX     If X here, store character
LCA         Save X board
LAB         Fetch O board
CAL STO     If O here, store character
LBA         Save O board
LLI 315     Set pointer to 6 position
LAC         Fetch X board
CAL STX     If X here, store character
LAB         Fetch O board
CAL STO     If O here, store character
LLI 320     Set pointer to 7 position
CAL STO     If O here, store character
LLI 322     Set pointer to 8 position
CAL STO     If O here, store character
LLI 324     Set pointer to 9 position
CAL STO     If O here, store character
LLI 300     Set pointer to board printout
```

```
        CAL MSG         Print current board

STX,    RLC             Bit set?
        RFC             No, return
        LMI 330         Yes, put X in board
        RET

STO,    RLC             Bit set?
        RFC             No, return
        LMI 317         Yes, put O in board
        RET
```

After the current board is outputted, the program requests the player to enter a move by first entering the number of the square which contains the pawn to be moved, and then the number of the square to which the pawn is to be moved. The input request is indicated by the output of the message FM which is output as part of the board output message. The program then calls the user supplied input routine to accept a character from the system input device.

The input subroutine is a user provided routine which must input a character from the keyboard device and return with the ASCII code for that character in the accumulator. This subroutine is free to use registers A through E in inputting the character. If registers H and L are required to be used, they must be restored to their original contents before returning to the calling program. If the system's input device does not provide automatic echo of the inputted character to the output device, this input routine should include some provision for echoing the character received to the output device. This subroutine is called only in the move input routine being presented here.

When the FROM square is received, it is checked first to determine whether it is a valid number from 1 to 9, since these are the only valid entries expected. This is checked by calling the FNUM subroutine. If the input is not within these limits, the ERROR routine is entered. The error routine is called at several points in the next group of routines whenever the move which has been input is found to be illegal. The error routine prints the message "NO! NO!"

and then jumps to the PBD entry point of the program to request a new input.

If the FROM input is valid, the 260 portion of the ASCII code is removed, and the binary value of the number entered is stored in location 03 002. The current O board is then checked to determine whether an O pawn does reside in the square designated by the input. The binary value of the FROM square is used as a counter by the RTAL subroutine which rotates the current O board until the carry bit indicates the presence or absence of an O pawn in that position. If there is no O pawn, the error routine is entered.

The message TO is then output and the INPUT routine is called to input the square to which the pawn is to be moved. This input is also checked by calling the FNUM routine to determine whether it is within the limits of the expected input. If it is valid, it is changed to its binary value and stored in location 03 003. The program then proceeds to the next routine which checks that the move is legal.

The listing for this portion of the program is presented below.

```
LLI 002        Set pointer to input storage
CAL INPUT      Input FM move
LMA            Save input
CAL FNUM       Number valid?
JTS ERROR      No, error
LAM            Fetch number
NDI 017        Delete ASCII code
LMA            Save FM location
LBA            Save bit count for RTAL
DCL            Set pointer to O board
LAM            Fetch O board
CAL RTAL       Is pawn in FM position?
JFC ERROR      No, illegal move
LLI 333        Set pointer to TO msg
CAL MSG        Print TO
LLI 003        Set pointer to input storage
CAL INPUT      Input TO move
LMA            Save TO input
```

```
        CAL FNUM        Input valid?
        JTS ERROR       No, error
        LAM             Fetch number
        NDI 017         Delete ASCII code
        LMA             Save TO location

ERROR,  LLI 364         Set pointer to error message
        LHI 005
        CAL MSG         Print error message
        LHI 003
        JMP PBD         Print current board

RTAL,   DCB             Decrement bit count
        RTZ             If zero, return
        RLC             Else, rotate left
        JMP RTAL

FNUM,   LAM             Fetch ASCII number
        CPI 261         Is number valid?
        RTS             No, return with S flag set
        SUI 272         If number is valid, return
        ADI 200         With S flag set
        RET
```

Once the FROM and TO values are received, the move must be checked to determine whether it is legal. First, a move forward is checked by subtracting 3 from the FROM value and checking it with the value stored for the TO value. If the move is forward one square, the position of the X pawns must be checked to make sure an X pawn is not blocking the move. The BLK routine is entered to check the forward move. BLK sets the TO value as a counter and fetches the current X board, which is then rotated left by the RTAL subroutine, placing the bit corresponding to the location the O pawn is to be moved, to the sign position in the accumulator. If this bit is set, the move is blocked by an X pawn and the error routine is entered. If not, the BLK routine returns to the mainstream of the program at the HMV label to make the move as entered.

If the move is not forward, a diagonal move to the left or right is

examined. By adding 1 to the forward move, the new value indicates a diagonal move to the right. If this matches the TO value, and is not equal to 7, which would be an illegal move from 9 to 7, the capture of an X pawn is checked, since a diagonal move must capture an opponent's pawn. If the move is not to the right, 2 is subtracted from the forward move and a diagonal move to the left is tested. If this matches and it is not equal to 3, indicating an illegal move from 7 to 3, an X pawn capture will be checked. If any of the above illegal conditions occur, the error routine is entered. When a capture move is indicated, the bit position in the current X board of the bit to be deleted is set up by the RTLP subroutine. The presence of an X pawn is checked, and if not there, the error routine is entered.

Once the preliminaries are complete, the move is checked for a win by the human player. If the move is to square 1, 2, or 3, the human has won the game. The HWIN routine is then entered to perform the required steps to teach the program. This is accomplished by zeroing the last move made by the program in the move index table. The HWIN routine then outputs a congratulatory message and starts a new game.

If the move does not result in a win, the move is entered in the current O board by resetting the bit indicating the FROM position and setting the bit indicating the TO position. If a capture was made, the bit in the current X board in the TO position is reset.

The listing for this routine is shown below.

```
DCL         Set FM pointer
LAM         Fetch FM
SUI 003     Is move forward?
INL         Check against TO
CPM
JTZ BLK     Yes, check if legal
ADI 001     No, move right 1 square
CPM         Is TO here
JTZ CKCAP   Yes, check for capture
SUI 002     No, move left 1 square
CPI 003     Is move from 7 to 3?
```

```
        JTZ ERROR       Yes, illegal
        CPM             Is TO here?
        JFZ ERROR       No, illegal move
CKCAP,  CPI 007         Is move to 7?
        JTZ ERROR       Yes, error
        LBM             Fetch TO move
        LAI 200         Set up to calculate capture
        CAL RTLP        Bit by rotating right
        LEA             Save capture bit
        LLI 000         Set X board pointer
        NDM             Capture?
        JTZ ERROR       No, illegal move
HMV,    LLI 002         Set pointer to FM
        LAM             Fetch FM location
        CAL RTAR        Set up FM bit
        LDA             Save FM bit
        INL             Set pointer to TO
        LAM             Fetch TO location
        CPI 004         Human wins?
        JTC HWIN        Yes, zero last move
        CAL RTAR        Set up TO bit
        LCA             Save TO bit
        LLI 001         Set pointer to current O board
        LAM             Fetch current board
        XRD             Clear old set
        ORC             Set new position
        LCA             Save new O board
        LMA             Save current board
        DCL
        LAE             Fetch capture bit
        NDA             Capture?
        JTZ NOCP        No, skip
        XRM             Yes, delete piece
        LMA             Save current X board

RTAR,   LBA             Set bit count
        LAI 001         Set bit to rotate
RTLP,   DCB             Decrement bit count
        RTZ             If zero, return
        RRC             Else, rotate right
```

```
            JMP RTLP

BLK,        LBM         Fetch TO move
            LLI 000     Set pointer to X board
            LAM         Fetch X board
            CAL RTAL    Check for blocked move
SET,        NDA         Is move blocked?
            JTS ERROR   Yes, illegal move
            LEI 000     Set for no capture
            JMP HMV     Return to make human move

HWIN,       LLI 004     Set pointer to last move
            LHI 003
            LLM         Fetch last move address
            LHI 004
            LMI 000     Zero last move
            LLI 315     Set pointer to lose message
            LHI 005
            CAL MSG     Print lose message
            JMP AGAIN   Start new game
```

Now that the human's move has been entered, it is time for the program to show what it knows about the game. This is the portion of the program which performs the table search and makes the resultant move that the program believes correct at the time for the given board configuration. The program first searches the model table for a matching model. If none is found, it is assumed that the move just input by the human is invalid. Some of the conditions which were not checked in the move input routine include a move forward into a square already occupied by an O pawn. (This results in the human eliminating one of his own pawns.) Or, a move from square 6 to square 4 in which an X pawn is captured. These moves will slip through the initial validity tests, but they do result in illegal board setups which are caught here. The only recourse for the program at this time is to start the game over again because the current board is unrecognizible.

When a model is found where the current X board and current O board match a byte pair in the model table, a pointer is calculated

from the relative position in the model table to the model-to-move index table. This pointer is calculated by dividing the low address of the X byte of the matching model by 2 and adding 106 to the result. The pointer is used to fetch the starting address of the list of possible moves in the move index table from the model-to-move index table.

The designated list of moves is then examined, and the first non-zero entry is used as the move the program will make in response to the current board model. If the program encounters a 200 byte in searching the table, it jumps to the ONO (OH! NO!) routine. Reaching a 200 byte indicates the program has made every move it can for the model, and they all lead to defeat. The ONO routine concedes defeat and also goes to the HWIN routine to eliminate the previous move with the intent that this model will not be encountered again.

When a move is found, a pointer is set up using the move number and the four bytes of the move are fetched from the move table. The last byte of this group is examined first to determine whether the program has won the game, indicated by the sign bit set in the last byte. Or, if the game is a draw, indicated by a 100 stored in the last byte. If the game is won by the computer, the WIN routine is entered to print the winning message and start a new game. If the game is a draw, the DRAW routine is entered, to print the draw message and start a new game. If the last byte is zero, the move is made as indicated by the first three bytes. The first byte has the bit set which is the FROM location of the X move, the second byte has the bit set which is the TO location of the X board and the third byte has the bit set which indicates which location in the O board has been captured. If the third byte is zero, the move does not capture any O pawn, and the game is continued by jumping to PBD.

When a capture is made, the O board is checked to determine whether all the O pawns have been captured. If so, the program has won the game. At this point, the program deletes the move that it has just made from the move table because it knows that if there was only one O pawn on the board, the program has a move open which will allow it to win by moving to the opponent's side rather than gobbling up the last pawn. The program will then print a message to inform the human that he has no more pawns!

The listing for this final routine of the Hexpawn program is presented next. The reader will note the common MSG call followed by a jump to AGAIN which starts a new game. This instruction pair, labeled CMSG, was set up to conserve program space, as it is a common sequence used by several routines to print game concluding messages and then begin a new game.

```
NOCP,   LDM         Save new X board
        LLI 010     Set pointer to model table
SMDL,   LAD         Fetch X board
        CPM         X board match model?
        JTZ OHLF    Yes, try O half
        INL         Advance table pointer
SMD1,   INL
        LAL         Check for end of table
        CPI 112     End of table?
        JFZ SMDL    No, continue search
        LLI 340     No match, illegal move made
        LHI 004     Print "NO GOOD!"
CMSG,   CAL MSG     Print message
        JMP AGAIN   Start new game

OHLF,   INL         Advance pointer to O board
        LAC         Fetch current O board
        CPM         O boards match?
        JFZ SMD1    No, continue search

        DCL         Move pointer to X board
        LAL         Set up to calculate pointer
        RRC         Divide by 2
        ADI 106     Add to start of mdl index tbl
        LLA         Set pointer to mdl index tbl
        LLM         Fetch pntr to move index tbl
        LHI 004     Set pntr to move index table
MFD1,   LAM         Fetch move number
        NDA         Move number here?
        JTS ONO     No move avail. Human wins
        JFZ MOVE    Move found, make it
        INL         Move zeroed, try next location
```

```
                JMP MFD1

MOVE,   LEL     Save move location
        LLI 004 Set pointer to last move
        LHI 003
        LME     Save location as last move
        RLC     Set up pointer to move
        RLC     Storage table
        ADI 174
        LLA     Set pointer
        LDM     Fetch FM bit
        INL     Advance pointer
        LCM     Fetch TO bit
        INL     Advance pointer
        LEM     Fetch capture bit
        INL     Advance pointer
        LAM     Fetch contest bit
        NDA     Is game over?
        JTS WIN Yes, computer wins
        JFZ DRAW Yes, draw
        LLI 000 Set pointer to X board
        LAM     Fetch current X board
        XRD     Clear old position
        ORC     Set new position
        LMA     Save new X board
        INL     Advance pointer to O board
        LAE     Fetch capture bit
        NDA     Capture?
        JTZ PBD No, print new board
        XRM     Yes, delete piece
        LMA     Save new O board
        JFZ PBD Non-0, continue game

        LLI 004 Set pointer to last move
        LLM     Fetch last move location
        LHI 004 Set pointer to active move list
        LMI 000 Cancel last move
        LHI 005 Set pointer to msg "I WIN
        LLI 330 YOU HAVE NO PAWNS!"
        CAL CMSG Print msg and start again
```

```
WIN,    LLI 025         Set pointer to store win move
        LHI 005
        LMA             Store win move in message
        LLI 011         Print "I MOVE TO  ."
        JMP CMSG        "I WIN, YOU LOSE"

DRAW,   LLI 052         Prnt 'DRAW, NO ONE WINS'
        LHI 005
        JMP CMSG        Print draw message

ONO,    LLI 374         Print "I CONCEDE!"
        LHI 004
        CAL MSG         Then zero last move
```

Well! That's it! Now the Hexpawn program is presented in its final assembled form to be loaded into an 8008 based microcomputer system. The operating portion of the program resides on pages 01 and 02 and the tables and messages are on pages 03 through 05. Due to the length of assembled listings for the tables and messages, they will be presented as an octal dump.

```
001 000   066 076              START,  LLI 076
001 002   056 005                      LHI 005
001 004   106 171 002                  CAL MSG
001 007   056 004                      LHI 004
001 011   036 000                      LDI 000
001 013   046 200                      LEI 200
001 015   364                  RSTR,   LLE
001 016   307                          LAM
001 017   363                          LLD
001 020   370                          LMA
001 021   030                          IND
001 022   040                          INE
001 023   110 015 001                  JFZ RSTR
001 026   066 000                AGAIN,  LLI 000
001 030   056 003                      LHI 003
001 032   076 340                      LMI 340
```

```
001 034   060
001 035   076 007
001 037   066 302
001 041   016 240
001 043   371
001 044   066 304
001 046   371
001 047   066 306
001 051   371
001 052   066 311
001 054   371
001 055   066 313
001 057   371
001 060   066 315
001 062   371
001 063   066 320
001 065   371
001 066   066 322
001 070   371
001 071   066 324
001 073   371
001 074   066 000
001 076   307
001 077   060
001 100   317
001 101   066 302
001 103   106 157 002
001 106   066 304
001 110   106 157 002
001 113   066 306
001 115   106 157 002
001 120   066 311
001 122   106 157 002
001 125   320
001 126   301
001 127   002
001 130   002
001 131   106 164 002
001 134   310
001 135   066 313
```

```
          INL
          LMI 007
PBD,      LLI 302
          LBI 240
          LMB
          LLI 304
          LMB
          LLI 306
          LMB
          LLI 311
          LMB
          LLI 313
          LMB
          LLI 315
          LMB
          LLI 320
          LMB
          LLI 322
          LMB
          LLI 324
          LMB
          LLI 000
          LAM
          INL
          LBM
          LLI 302
          CAL STX
          LLI 304
          CAL STX
          LLI 306
          CAL STX
          LLI 311
          CAL STX
          LCA
          LAB
          RLC
          RLC
          CAL STO
          LBA
          LLI 313
```

```
001 137    302                LAC
001 140    106 157 002        CAL STX
001 143    320                LCA
001 144    301                LAB
001 145    106 164 002        CAL STO
001 150    310                LBA
001 151    066 315            LLI 315
001 153    302                LAC
001 154    106 157 002        CAL STX
001 157    301                LAB
001 160    106 164 002        CAL STO
001 163    066 320            LLI 320
001 165    106 164 002        CAL STO
001 170    066 322            LLI 322
001 172    106 164 002        CAL STO
001 175    066 324            LLI 324
001 177    106 164 002        CAL STO
001 202    066 300            LLI 300
001 204    106 171 002        CAL MSG
001 207    066 002            LLI 002
001 211    106 000 006        CAL INPUT
001 214    370                LMA
001 215    106 303 002        CAL FNUM
001 220    160 207 002        JTS ERROR
001 223    307                LAM
001 224    044 017            NDI 017
001 226    370                LMA
001 227    310                LBA
001 230    061                DCL
001 231    307                LAM
001 232    106 223 002        CAL RTAL
001 235    100 207 002        JFC ERROR
001 240    066 333            LLI 333
001 242    106 171 002        CAL MSG
001 245    066 003            LLI 003
001 247    106 000 006        CAL INPUT
001 252    370                LMA
001 253    106 303 002        CAL FNUM
001 256    160 207 002        JTS ERROR
001 261    307                LAM
```

```
001 262    044 017                  NDI 017
001 264    370                      LMA
001 265    061                      DCL
001 266    307                      LAM
001 267    024 003                  SUI 003
001 271    060                      INL
001 272    277                      CPM
001 273    150 242 002              JTZ BLK
001 276    004 001                  ADI 001
001 300    277                      CPM
001 301    150 317 001              JTZ CKCAP
001 304    024 002                  SUI 002
001 306    074 003                  CPI 003
001 310    150 207 002              JTZ ERROR
001 313    277                      CPM
001 314    110 207 002              JFZ ERROR
001 317    074 007          CKCAP,  CPI 007
001 321    150 207 002              JTZ ERROR
001 324    317                      LBM
001 325    006 200                  LAI 200
001 327    106 234 002              CAL RTLP
001 332    340                      LEA
001 333    066 000                  LLI 000
001 335    247                      NDM
001 336    150 207 002              JTZ ERROR
001 341    066 002          HMV,    LLI 002
001 343    307                      LAM
001 344    106 231 002              CAL RTAR
001 347    330                      LDA
001 350    060                      INL
001 351    307                      LAM
001 352    074 004                  CPI 004
001 354    140 323 002              JTC HWIN
001 357    106 231 002              CAL RTAR
001 362    320                      LCA
001 363    066 001                  LLI 001
001 365    307                      LAM
001 366    253                      XRD
001 367    262                      ORC
001 370    320                      LCA
```

```
001 371    370                    LMA
001 372    061                    DCL
001 373    304                    LAE
001 374    240                    NDA
001 375    150 002 002            JTZ NOCP
002 000    257                    XRM
002 001    370                    LMA
002 002    337            NOCP,   LDM
002 003    066 010                LLI 010
002 005    303            SMDL,   LAD
002 006    277                    CPM
002 007    150 034 002            JTZ OHLF
002 012    060                    INL
002 013    060            SMD1,   INL
002 014    306                    LAL
002 015    074 112                CPI 112
002 017    110 005 002            JFZ SMDL
002 022    066 340                LLI 340
002 024    056 004                LHI 004
002 026    106 171 002    CMSG,   CAL MSG
002 031    104 026 001            JMP AGAIN
002 034
002 034    060            OHLF,   INL
002 035    302                    LAC
002 036    277                    CPM
002 037    110 013 002            JFZ SMD1
002 042
002 042    061                    DCL
002 043    306                    LAL
002 044    012                    RRC
002 045    004 106                ADI 106
002 047    360                    LLA
002 050    367                    LLM
002 051    056 004                LHI 004
002 053    307            MFD1,   LAM
002 054    240                    NDA
002 055    160 314 002            JTS ONO
002 060    110 067 002            JFZ MOVE
002 063    060                    INL
002 064    104 053 002            JMP MFD1
```

```
002 067    346                    MOVE,  LEL
002 070    066 004                       LLI 004
002 072    056 003                       LHI 003
002 074    374                           LME
002 075    002                           RLC
002 076    002                           RLC
002 077    004 174                       ADI 174
002 101    360                           LLA
002 102    337                           LDM
002 103    060                           INL
002 104    327                           LCM
002 105    060                           INL
002 106    347                           LEM
002 107    060                           INL
002 110    307                           LAM
002 111    240                           NDA
002 112    160 262 002                   JTS WIN
002 115    110 274 002                   JFZ DRAW
002 120    066 000                       LLI 000
002 122    307                           LAM
002 123    253                           XRD
002 124    262                           ORC
002 125    370                           LMA
002 126    060                           INL
002 127    304                           LAE
002 130    240                           NDA
002 131    150 037 001                   JTZ PBD
002 134    257                           XRM
002 135    370                           LMA
002 136    110 037 001                   JFZ PBD
002 141
002 141    066 004                       LLI 004
002 143    367                           LLM
002 144    056 004                       LHI 004
002 146    076 000                       LMI 000
002 150    056 005                       LHI 005
002 152    066 330                       LLI 330
002 154    106 026 002                   CAL CMSG
002 157
002 157    002                    STX,   RLC
```

```
002 160    003                          RFC
002 161    076 330                      LMI 330
002 163    007                          RET
002 164
002 164    002                  STO,    RLC
002 165    003                          RFC
002 166    076 317                      LMI 317
002 170    007                          RET
002 171
002 171    307                  MSG,    LAM
002 172    240                          NDA
002 173    053                          RTZ
002 174    106 100 006                  CAL PRINT
002 177    060                          INL
002 200    110 171 002                  JFZ MSG
002 203    050                          INH
002 204    104 171 002                  JMP MSG
002 207
002 207    066 364              ERROR,  LLI 364
002 211    056 005                      LHI 005
002 213    106 171 002                  CAL MSG
002 216    056 003                      LHI 003
002 220    104 037 001                  JMP PBD
002 223
002 223    011                  RTAL,   DCB
002 224    053                          RTZ
002 225    002                          RLC
002 226    104 223 002                  JMP RTAL
002 231
002 231    310                  RTAR,   LBA
002 232    006 001                      LAI 001
002 234    011                  RTLP,   DCB
002 235    053                          RTZ
002 236    012                          RRC
002 237    104 234 002                  JMP RTLP
002 242
002 242    317                  BLK,    LBM
002 243    066 000                      LLI 000
002 245    307                          LAM
002 246    106 223 002                  CAL RTAL
```

```
002 251   240                    SET,  NDA
002 252   160 207 002                  JTS ERROR
002 255   046 000                      LEI 000
002 257   104 341 001                  JMP HMV
002 262
002 262   066 025                WIN,  LLI 025
002 264   056 005                      LHI 005
002 266   370                          LMA
002 267   066 011                      LLI 011
002 271   104 026 002                  JMP CMSG
002 274
002 274   066 052                DRAW, LLI 052
002 276   056 005                      LHI 005
002 300   104 026 002                  JMP CMSG
002 303
002 303   307                    FNUM, LAM
002 304   074 261                      CPI 261
002 306   063                          RTS
002 307   024 272                      SUI 272
002 311   004 200                      ADI 200
002 313   007                          RET
002 314
002 314   066 374                ONO,  LLI 374
002 316   056 004                      LHI 004
002 320   106 171 002                  CAL MSG
003 323
002 323   066 004                HWIN, LLI 004
002 325   056 003                      LHI 003
002 327   367                          LLM
002 330   056 004                      LHI 004
002 332   076 000                      LMI 000
002 334   066 315                      LLI 315
002 336   056 005                      LHI 005
002 340   106 171 002                  CAL MSG
002 343   104 026 001                  JMP AGAIN
002 346
```

TEMPORARY DATA

```
003 000   000 000 000 000 000
```

MODEL TABLE

003 010	340	043	340	016	340	025	260	021
003 020	150	041	240	062	300	051	150	014
003 030	160	034	260	012	304	061	140	054
003 040	140	021	140	024	240	041	070	010
003 050	200	070	120	030	104	060	230	010
003 060	240	014	210	042	054	040	060	020
003 070	110	040	110	010	220	020	044	020
003 100	200	060	240	032	100	020	250	052
003 110	040	070						

MODEL-TO-MOVE INDEX TABLE

003 112			000	004	010	013	017	023
003 120	027	033	036	041	043	046	051	054
003 130	057	061	063	065	070	073	075	077
003 140	101	103	107	112	115	120	123	125
003 150	131	133	135					

MOVE TABLE

003 200	200	020	000	000	200	010	020	000
003 210	100	020	040	000	100	010	000	000
003 220	100	004	010	000	040	010	020	000
003 230	040	004	000	000	020	000	000	267
003 240	020	000	000	270	010	000	000	267
003 250	010	000	000	270	010	000	000	271
003 260	004	000	000	270	004	000	000	271
003 270	000	000	000	100				

BOARD OUTPUT MESSAGE

003 300	215	212	330	336	330	336	330	215
003 310	212	240	336	240	336	240	215	212
003 320	317	336	317	336	317	215	212	306
003 330	315	240	000	240	324	317	240	000

MOVE INDEX TABLE

```
004 000   007 004 003 200 001 004 005 200
004 010   001 002 200 007 006 010 200 007
004 020   003 013 200 007 002 006 200 003
004 030   004 017 200 005 012 200 005 006
004 040   200 010 200 002 003 200 005 017
004 050   200 006 017 200 006 007 200 017
004 060   200 010 200 002 200 005 010 200
004 070   003 016 200 013 200 017 200 017
004 100   200 016 200 007 006 010 200 003
004 110   013 200 005 013 200 002 010 200
004 120   006 016 200 002 200 001 006 017
004 130   200 017 200 017 200 006 200
```

RESTORE LIST

```
004 200   007 004 003 200 001 004 005 200
004 210   001 002 200 007 006 010 200 007
004 220   003 013 200 007 002 006 200 003
004 230   004 017 200 005 012 200 005 006
004 240   200 010 200 002 003 200 005 017
004 250   200 006 017 200 006 007 200 017
004 260   200 010 200 002 200 005 010 200
004 270   003 016 200 013 200 017 200 017
004 300   200 016 200 007 006 010 200 003
004 310   013 200 005 013 200 002 010 200
004 320   006 016 200 002 200 001 006 017
004 330   200 017 200 017 200 006 200
```

MESSAGE STORAGE

```
004 340   215 212 316 317 240 307 317 317
004 350   304 241 240 315 325 323 324 240
004 360   323 324 301 322 324 240 301 307
004 370   301 311 316 000 215 212 311 240

005 000   303 317 316 303 305 304 305 241
```

005	010	000	215	212	311	240	315	317	326
005	020	305	240	324	317	240	240	254	215
005	030	212	311	240	327	311	316	241	240
005	040	331	317	325	240	314	317	323	305
005	050	241	000	215	212	304	322	301	327
005	060	254	316	317	240	317	316	305	240
005	070	327	311	316	323	256	000	215	212
005	100	310	305	322	305	247	323	240	324
005	110	310	305	240	302	317	301	322	304
005	120	215	212	261	336	262	336	263	215
005	130	212	264	336	265	336	266	215	212
005	140	267	336	270	336	271	215	212	311
005	150	247	315	240	330	254	240	331	317
005	160	325	247	322	305	240	317	215	212
005	170	315	317	326	305	240	317	316	305
005	200	240	323	321	325	301	322	305	240
005	210	306	317	322	327	301	322	304	240
005	220	311	306	240	326	301	303	301	316
005	230	324	215	212	317	322	240	317	316
005	240	305	240	323	321	325	301	322	305
005	250	240	304	311	301	307	317	316	301
005	260	314	314	331	240	324	317	240	303
005	270	301	320	324	325	322	305	215	212
005	300	331	317	325	240	323	324	301	322
005	310	324	256	215	212	000	215	212	331
005	320	317	325	240	327	311	316	256	000
005	330	215	212	311	240	327	311	316	241
005	340	240	331	317	325	240	310	301	326
005	350	305	240	316	317	240	320	301	327
005	360	316	323	256	000	215	212	316	317
005	370	241	240	316	317	241	000		

006 000	000	INPUT
006 100	000	PRINT

After loading the Hexpawn program into memory, the program execution is begun by jumping to the start address of the program which is at location 000 on page 01. The program will print the introductory message followed by the starting position of the playing board. When the FM is displayed, the player enters the number of the square from which the pawn is to be moved. The program then prints TO, and the player enters the number of the square to which the pawn is to go. The program then makes its move, and the new board configuration is displayed. When the outcome of the game is evident to the program, a message is printed to indicate win, lose, or draw. A sample of three consecutive games is listed below. Note how the program goes through its learning process as the human player makes the same sequence of moves in each game.

```
HERE'S THE BOARD
1|2|3
4|5|6
7|8|9
I'M X, YOU'RE O
MOVE ONE SQUARE FORWARD IF VACANT
OR ONE SQUARE DIAGONALLY TO CAPTURE
YOU START.

X|X|X
 | |
O|O|O
FM 8 TO 5
 |X|X
X|O|
O| |O
FM 9 TO 6
 | |X
X|O|X
O| |
FM 5 TO 2
YOU WIN.
```

```
X|X|X
 | |
O|O|O
FM 8 TO 5
 |X|X
X|O|
O| |O
FM 9 TO 6
 |X|
X|X|O
O| |
FM 7 TO 5
 | |
X|O|X
 | |
FM 5 TO 2
YOU WIN.
X|X|X
 | |
O|O|O
FM 8 TO 5
 |X|X
X|O|
O| |O
FM 9 TO 6
 |X|
X|X|O
O| |
FM 7 TO 5
I MOVE TO 7,
I WIN! YOU LOSE!
```

AN 8080 LISTING OF THE HEXPAWN PROGRAM

This final listing is the operating portion of the Hexpawn program written for an 8080 based system. The 8080 version makes use of the more powerful instruction set of the 8080 mainly in setting up pointers, and includes instructions to set up the stack pointer, a function not required by the 8008. The operating portion of the 8080

version resides on pages 01 and 02 with the stack beginning at location 377 on page 02. The tables and messages are located on pages 03, 04, and 05 exactly as defined for the 8008 version. The user defined I/O routines should be set up as defined previously. The functional operation of the program is exactly as described in the text, and, therefore, need not be expanded upon. So, for those readers with 8080 based systems, here is the listing for the Hexpawn program.

```
001 000   061 000 003        START,  LXS 000 003
001 003   041 076 005                LXH 076 005
001 006   315 165 002                CAL MSG
001 011   041 000 004                LXH 000 004
001 014   021 200 004                LXD 200 004
001 017   032                 RSTR,  LDAD
001 020   167                        LMA
001 021   054                        INL
001 022   034                        INE
001 023   302 017 001                JFZ RSTR
001 026   041 000 003        AGAIN,  LXH 000 003
001 031   066 340                    LMI 340
001 033   054                        INL
001 034   066 007                    LMI 007
001 036   056 302              PBD,  LLI 302
001 040   006 240                    LBI 240
001 042   160                        LMB
001 043   056 304                    LLI 304
001 045   160                        LMB
001 046   056 306                    LLI 306
001 050   160                        LMB
001 051   056 311                    LLI 311
001 053   160                        LMB
001 054   056 313                    LLI 313
001 056   160                        LMB
001 057   056 315                    LLI 315
001 061   160                        LMB
001 062   056 320                    LLI 320
001 064   160                        LMB
001 065   056 322                    LLI 322
```

```
001 067    160                  LMB
001 070    056 324              LLI 324
001 072    160                  LMB
001 073    056 000              LLI 000
001 075    176                  LAM
001 076    054                  INL
001 077    106                  LBM
001 100    056 302              LLI 302
001 102    315 153 002          CAL STX
001 105    056 304              LLI 304
001 107    315 153 002          CAL STX
001 112    056 306              LLI 306
001 114    315 153 002          CAL STX
001 117    056 311              LLI 311
001 121    315 153 002          CAL STX
001 124    117                  LCA
001 125    170                  LAB
001 126    007                  RLC
001 127    007                  RLC
001 130    315 160 002          CAL STO
001 133    107                  LBA
001 134    056 313              LLI 313
001 136    171                  LAC
001 137    315 153 002          CAL STX
001 142    117                  LCA
001 143    170                  LAB
001 144    315 160 002          CAL STO
001 147    107                  LBA
001 150    056 315              LLI 315
001 152    171                  LAC
001 153    315 153 002          CAL STX
001 156    170                  LAB
001 157    315 160 002          CAL STO
001 162    056 320              LLI 320
001 164    315 160 002          CAL STO
001 167    056 322              LLI 322
001 171    315 160 002          CAL STO
001 174    056 324              LLI 324
001 176    315 160 002          CAL STO
001 201    056 300              LLI 300
```

```
001 203    315 165 002          CAL MSG
001 206    056 002              LLI 002
001 210    315 000 006          CAL INPUT
001 213    167                  LMA
001 214    315 270 002          CAL FNUM
001 217    372 177 002          JTS ERROR
001 222    176                  LAM
001 223    346 017              NDI 017
001 225    167                  LMA
001 226    107                  LBA
001 227    055                  DCL
001 230    176                  LAM
001 231    315 212 002          CAL RTAL
001 234    322 177 002          JFC ERROR
001 237    056 333              LLI 333
001 241    315 165 002          CAL MSG
001 244    056 003              LLI 003
001 246    315 000 006          CAL INPUT
001 251    167                  LMA
001 252    315 270 002          CAL FNUM
001 255    372 177 002          JTS ERROR
001 260    176                  LAM
001 261    346 017              NDI 017
001 263    167                  LMA
001 264    055                  DCL
001 265    176                  LAM
001 266    326 003              SUI 003
001 270    054                  INL
001 271    276                  CPM
001 272    312 231 002          JTZ BLK
001 275    306 001              ADI 001
001 277    276                  CPM
001 300    312 316 001          JTZ CKCAP
001 303    326 002              SUI 002
001 305    376 003              CPI 003
001 307    312 177 002          JTZ ERROR
001 312    276                  CPM
001 313    302 177 002          JFZ ERROR
001 316    376 007      CKCAP,  CPI 007
001 320    312 177 002          JTZ ERROR
```

```
001 323   106                        LBM
001 324   076 200                    LAI 200
001 326   315 223 002                CAL RTLP
001 331   137                        LEA
001 332   056 000                    LLI 000
001 334   246                        NDM
001 335   312 177 002                JTZ ERROR
001 340   056 002              HMV,  LLI 002
001 342   176                        LAM
001 343   315 220 002                CAL RTAR
001 346   127                        LDA
001 347   054                        INL
001 350   176                        LAM
001 351   376 004                    CPI 004
001 353   332 307 002                JTC HWIN
001 356   315 220 002                CAL RTAR
001 361   117                        LCA
001 362   056 001                    LLI 001
001 364   176                        LAM
001 365   252                        XRD
001 366   261                        ORC
001 367   117                        LCA
001 370   167                        LMA
001 371   055                        DCL
001 372   173                        LAE
001 373   247                        NDA
001 374   312 001 002                JTZ NOCP
001 377   256                        XRM
002 000   167                        LMA
002 001   126              NOCP,     LDM
002 002   056 010                    LLI 010
002 004   172              SMDL,     LAD
002 005   276                        CPM
002 006   312 032 002                JTZ OHLF
002 011   054                        INL
002 012   054              SMD1,     INL
002 013   175                        LAL
002 014   376 112                    CPI 112
002 016   302 004 002                JFZ SMDL
002 021   041 340 004                LXH 340  004
```

```
002 024   315 165 002          CMSG,  CAL MSG
002 027   303 026 001                 JMP AGAIN
002 032
002 032   054                  OHLF,  INL
002 033   171                         LAC
002 034   276                         CPM
002 035   302 012 002                 JFZ SMD1
002 040
002 040   055                         DCL
002 041   175                         LAL
002 042   017                         RRC
002 043   306 106                     ADI 106
002 045   157                         LLA
002 046   156                         LLM
002 047   046 004                     LHI 004
002 051   176                  MFD1,  LAM
002 052   247                         NDA
002 053   372 301 002                 JTS ONO
002 056   302 065 002                 JFZ MOVE
002 061   054                         INL
002 062   303 051 002                 JMP MFD1
002 065
002 065   135                  MOVE,  LEL
002 066   041 004 003                 LXH 004 003
002 071   163                         LME
002 072   007                         RLC
002 073   007                         RLC
002 074   306 174                     ADI 174
002 076   157                         LLA
002 077   126                         LDM
002 100   054                         INL
002 101   116                         LCM
002 102   054                         INL
002 103   136                         LEM
002 104   054                         INL
002 105   176                         LAM
002 106   247                         NDA
002 107   372 251 002                 JTS WIN
002 112   302 262 002                 JFZ DRAW
002 115   056 000                     LLI 000
```

```
002 117    176                          LAM
002 120    252                          XRD
002 121    261                          ORC
002 122    167                          LMA
002 123    054                          INL
002 124    173                          LAE
002 125    247                          NDA
002 126    312 036 001                  JTZ PBD
002 131    256                          XRM
002 132    167                          LMA
002 133    302 036 001                  JFZ PBD
002 136
002 136    056 004                      LLI 004
002 140    156                          LLM
002 141    046 004                      LHI 004
002 143    066 000                      LMI 000
002 145    041 330 005                  LXH 330 005
002 150    315 024 002                  CAL CMSG
002 153
002 153    007                    STX,  RLC
002 154    320                          RFC
002 155    066 330                      LMI 330
002 157    311                          RET
002 160
002 160    007                    STO,  RLC
002 161    320                          RFC
002 162    066 317                      LMI 317
002 164    311                          RET
002 165
002 165    176                    MSG,  LAM
002 166    247                          NDA
002 167    310                          RTZ
002 170    315 100 006                  CAL PRINT
002 173    043                          INXH
002 174    303 165 002                  JMP MSG
002 177
002 177    041 364 005            ERROR, LXH 364 005
002 202    315 165 002                  CAL MSG
002 205    046 003                      LHI 003
002 207    303 036 001                  JMP PBD
```

```
002 212   005                 RTAL,  DCB
002 213   310                        RTZ
002 214   007                        RLC
002 215   303 212 002                JMP RTAL
002 220
002 220   107                 RTAR,  LBA
002 221   076 001                    LAI 001
002 223   005                 RTLP,  DCB
002 224   310                        RTZ .
002 225   017                        RRC
002 226   303 223 002                JMP RTLP
002 231
002 231   106                  BLK,  LBM
002 232   056 000                    LLI 000
002 234   176                        LAM
002 235   315 212 002                CAL RTAL
002 240   247                  SET,  NDA
002 241   372 177 002                JTS ERROR
002 244   036 000                    LEI 000
002 246   303 340 001                JMP HMV
002 251
002 251   062 025 005          WIN,  STA 025 005
002 254   041 011 005                LXH 011 005
002 257   315 024 002                CAL CMSG
002 262
002 262   041 052 005         DRAW,  LXH 052 005
002 265   303 024 002                JMP CMSG
002 270
002 270   176                 FNUM,  LAM
002 271   376 260                    CPI 260
002 273   370                        RTS
002 274   326 272                    SUI 272
002 276   306 200                    ADI 200
002 300   311                        RET
002 301
002 301   041 374 004          ONO,  LXH 374 004
002 304   315 165 002                CAL MSG
002 307
002 307   041 004 003         HWIN,  LXH 004 003
002 312   156                        LLM
```

```
002 313    046 004                    LHI 004
002 315    066 000                    LMI 000
002 317    041 315 005                LXH 315 005
002 322    315 165 002                CAL MSG
002 325    303 026 001                JMP AGAIN
002 330

006 000    000                INPUT,

006 100    000                PRINT,
```

HANGMAN!

HANGMAN is a word game with which most readers are probably well acquainted. The object of the game is to determine what word a player is thinking of by guessing the letters that make up the word. When characters contained in the word are correctly identified, the positions of the letters that have been ascertained are disclosed. The goal of the game is to ascertain all the letters making up the concealed word with the least amount of incorrect guesses. The game in the form to be presented traditionally received its name from the practice of creating a sketch of a stick figure being hung from a hangman's scaffold. A portion of the stick figure, such as a head, arms, torso, or legs, would be drawn in each time an incorrect letter guess was made. If the stick figure was completed before the entire word had been correctly identified, the player lost.

In the computerized version of the game to be presented here, the computer will select a word from a list of words (which may be created by the reader if desired). The computer will then allow a player to enter guesses as to the letters contained in the word selected. Each time the player correctly identifies a letter contained in the word, the characters that have been ascertained will be displayed in their proper location within the word. Each time a guess is incorrect, the computer will add a letter towards the spelling of Hangman! A game is finished when a player correctly identifies the word selected by the computer. Or, when eight incorrect letter guesses result in the complete spelling of Hangman!

The game is relatively simple to implement on a computer. However, despite its relative simplicity programming wise, the game can be surprisingly fun and challenging. This is due primarily to the nature of the game, augmented by the fact that the programmer has a virtually unlimited reservoir of alternatives to use when creating a list of words for the computer to select from when playing a game.

Besides its use as a pure fun game, the program can also be applied to more serious considerations such as making it a learning or teaching tool. Since the level of the vocabulary that is placed in the

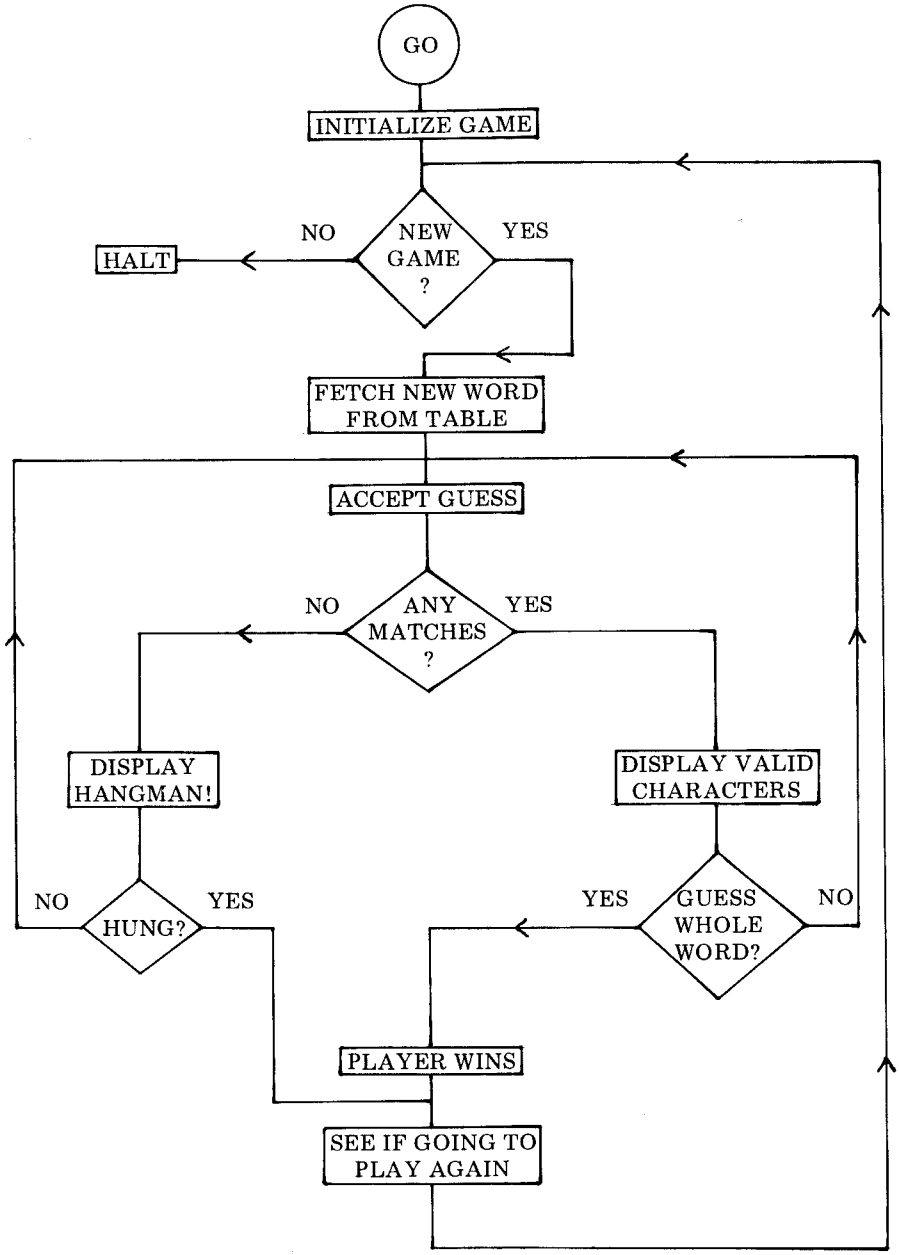
computer memory may be set as desired by the programmer, the game can be applied towards helping students develop vocabularies in virtually any subject. Additionally, one may readily change the language with which the game is played! French, Spanish, German, Maylasian! The computer will not care at all! The human player, though, may be suitably impressed with such variations!

FUNDAMENTAL STRUCTURE OF THE PROGRAM

The structure of the program is straightforward. Essentially, the computer is directed to select a word from a list of words in memory. A selected word is transferred to a buffer storage area. The player is asked to guess the letters in the selected word. Each time the player makes an entry, the buffer is scanned for any matches with the letter entered by the player. Appropriate matches are transferred to a working buffer that keeps track of all correct letter entries made by the player. Correct entries result in the contents of the working buffer being displayed to show the correct locations of letters properly identified by the player. Incorrect guesses by the player result in successive portions of the dreaded HANGMAN! being displayed. The overall flow of the program to be described here is illustrated in the flow chart on the following page.

DETAILS OF THE PROGRAM

The operating portion of the program described here fits easily into less than 1 K of memory excluding a variable length word table. The word table is simply a list of words. The list may extend as far as the user desires in available memory. A sample word table is included in this article. However, the reader may create a new list of words for the game. The word list provided uses about four pages of memory if the entire list is used. However, the list may be shortened if memory space is at a premium. A version of the program assembled to reside on pages 02 through 04 (operating portion) with the word table starting on page 05 will be provided as part of this article.


GO
INITIALIZE GAME
NEW GAME ?
NO
YES
HALT
FETCH NEW WORD FROM TABLE
ACCEPT GUESS
ANY MATCHES ?
NO
YES
DISPLAY HANGMAN!
DISPLAY VALID CHARACTERS
HUNG?
NO
YES
GUESS WHOLE WORD?
YES
NO
PLAYER WINS
SEE IF GOING TO PLAY AGAIN

The first several routines in the program are used to initialize pointer storage locations and display a WANT A NEW WORD message to the system operator. Messages to be displayed by the program are stored as text strings in memory terminated by a zero byte. Text strings are displayed by calling a subroutine labeled MSG. MSG will output a string of characters pointed to by the H and L registers until a zero byte is detected. The actual MSG subroutine will be presented later. Suffice it to say at this point that one need only set up the H & L CPU registers to the starting address of a text string stored in memory, then call the MSG subroutine when it is desired to display such messages.

Following the WANT A NEW WORD message display, the program waits for a response from the operator by calling a user defined input subroutine labeled INPUTN. INPUTN should be designed by the reader to accept a character from the system's imput device (such as a keyboard) and return the character in the accumulator to the calling program. The INPUTN subroutine should also perform an echo display function so that the operator may verify the input character. The subroutine is free to utilize CPU registers A through E as far as this program is concerned. The user should note that this program expects the eighth bit in the accumulator to be in a '1' condition when the remaining seven bits represent an ASCII encoded character.

If the operator responds to the WANT A NEW WORD query by entering the letter N for no, the program terminates after displaying an appropriate response. If a Y for yes is entered, the program continues by calling upon a subroutine called MOVTAB. This subroutine will fetch a word from the program's word table. It will then transfer the word into a buffer. The buffer it is transferred into will be referred to as the word buffer in this article. The actual operation of the MOVTAB subroutine will be presented later. Suffice it to note at this time that upon return from the MOVTAB subroutine, a new word will be residing in the word buffer. The program will then be ready to start the playing of a game of Hangman!

```
START,    LHI 003        Set pointer to
          LLI 350        Number of guesses counter
          LMI 001        Initialize counter
```

```
        LLI 356     Pointer to word table pointer
        LMI 000     Initialize to
        INL         Start of
        LMI 005     Word table
NEWONE, LHI 004     Pointer to WANT A
        LLI 000     NEW WORD? message
        CAL MSG     Display message
        CAL INPUTN  Fetch answer
        CPI 316     Was it NO?
        JTZ NOMORE  Say GOODBYE if no
        CPI 331     Else was it a YES?
        JFZ NEWONE  Ignore input otherwise
        CAL MOVTAB  If Y, fetch a word
        JMP GUESS   Go play the game

NOMORE, LHI 004     Pointer to GOODBYE
        LLI 025     Message
        CAL MSG     Display message
        HLT         End of playing session
```

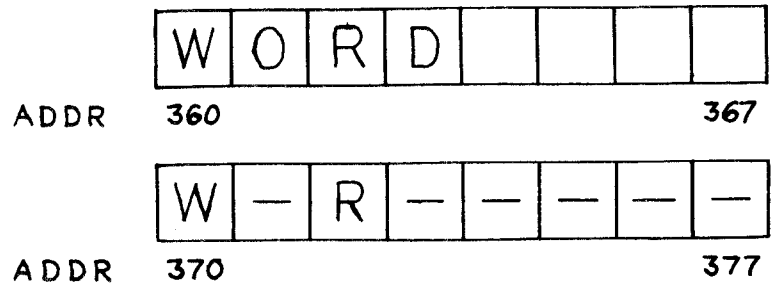
The next portion of the program begins by sending a message telling the operator to GUESS A LETTER. The program then accepts an input from the player. The input is expected to be any alphabetical character.

The program will then scan the word buffer to see if the letter received from the player matches with any of the letters in the word. Whenever a match is detected, the letter is stored in the same position in a second buffer called the guesses buffer.

It should be pointed out that the word buffer and the guesses buffer are identical in length. (Eight bytes in this program.) In the example program, the word buffer starts at location 360 on page 03. The guesses buffer starts at location 370 on page 03. At the start of each game, the word buffer, as previously mentioned, will be loaded with a word taken from the word table. A word may be up to eight letters in length. If there are not eight letters in a word, the balance of the word buffer will be filled with zero bytes. Furthermore, at the time the word buffer obtains a new word, the guesses buffer is filled

with hyphens.

It thus becomes an easy matter to keep a record of correct guesses as the Hangman game progresses. Each time a position in the word buffer matches with the letter guessed by the player, the identical position in the guesses buffer is changed from a hyphen to the actual letter! The addressing scheme used in the program makes it easy to accomplish the objective. When a match is found in the word buffer, it is only necessary to add 010 (octal) to the buffer pointer to reach the corresponding position in the guesses buffer. The pictorial below should clarify the relationship.



If there is a match between the character inputted and any position in the word buffer, a flag is set (using register B). The word in the word buffer is scanned for a matching character until a zero byte is detected or the buffer pointer reaches the last address allocated for the buffer.

```
GUESS,    LHI 004       Pointer to GUESS A
          LLI 037       LETTER message
          CAL MSG       Display message
          CAL INPUTN    Fetch a letter
          LCA           Save letter in C

SCAN,     LBI 000       Clear B for a flag register
          LHI 003       Set pointer to
          LLI 360       Word buffer

CKMTCH,   CPM           Look for a match
          JFZ NOMTCH    Skip ahead if not a match
          INB           Set B as a flag
```

```
        LAI 010     Advance pointer
        ADL         To the guesses
        LLA         Buffer
        LMC         And deposit character
        LAL         Decrease pointer
        SUI 010     Back to
        LLA         Word buffer

NOMTCH, INL         Advance buffer pointer
        LAM         See if next character
        NDA         Is a zero byte
        JTZ EOWORD  End of word if so
        LAI 007     See if at
        NDL         End of word buffer
        JTZ EOWORD  End of word if so
        LAC         Restore character to ACC
        JMP CKMTCH  Check next position
```

When the entire word buffer has been searched, the program checks the flag mentioned previously (in CPU register B) to determine if the player had made a correct letter guess. The flag will be set (have a value) if such was the case. It will still be zero if no match was detected.

If the flag was set during the SCAN operation, then the player has correctly determined a letter that exists in the word that the player is trying to identify. The program must now show the player how much of the word has been correctly identified. Thus, the program will first display an encouraging message. Then, the routine simply outputs the contents of the guesses buffer. The guesses buffer will contain all the locations of the letters that have been correctly identified during the game. Unidentified locations will still contain a hyphen. Thus, if a player had correctly identified the letters W and R in the spelling of WORD, the computer would output: W-R-.

As the program outputs the contents of the guesses buffer, it checks to see if any hyphens (referred to in the listing as dashes) are displayed. A software flag mechanism is used for this purpose. At the end of the guesses buffer outputting operation, the flag is tested. If

it is zero, then the player has identified the entire word. The program will then display a congratulations message and go back to see if the player wants to continue the game with a new word. If there are any hyphens left in the guesses buffer as indicated by the software flag being set, then the program loops back to allow the player to guess another letter.

The reader may note that the routine examines the word buffer to determine when to stop outputting the contents of the guesses buffer. This is because the word buffer will contain a zero byte at the end of the word if the word is less than eight characters in length. The guesses buffer does not contain such an indicator.

The portion of the program just discussed is presented next.

```
EOWORD,   INB          At end of word, exercise
          DCB          The MATCH flag
          JTZ HANGIT   If = 0, no matches
          LHI 004      If match(es), set pointer
          LLI 074      To GOOD. YOU HAVE: msg
          CAL MSG      Display message
          LHI 003      Set pointer to
          LLI 353      Dashes counter storage
          LMI 000      Clear dashes counter
          LLI 370      Pointer to guesses buffer

NOTEND,   LAM          Fetch a character
          CPI 255      See if it is a dash
          JFZ AHEAD2   Skip next instruction if not
          LEL          Save pointer temporarily
          LLI 353      Set pointer to dashes counter
          LBM          Fetch dashes value
          INB          Increment
          LMB          Restore to memory
          LLE          Restore saved pointer

AHEAD2,   CAL PRINT    Print the character
          INL          Advance the buffer pointer
          LAL          Decrease pointer
          SUI 010      To word buffer
```

```
            LLA         Here
            LAM         Fetch data from word buffer
            NDA         And see if it is
            JTZ ENDAGN  Zero byte, jump if so
            LAL         Fetch pointer
            NDI 007     See if at end of word
            JTZ ENDAGN  Buffer, jump if so
            LAI 010     Else restore pointer
            ADL         Ahead to
            LLA         The guesses buffer
            JMP NOTEND  Do next character in buffer

ENDAGN,     LLI 353     Pointer to dashes counter
            LBM         Fetch value
            INB         Exercise the dashes
            DCB         Flag register
            JFZ GUESS   Word not completed
            LHI 004     If reach here, set pointer
            LLI 120     To congratulations message
            CAL MSG     Display message
            JMP NEWONE  Go play with a new word
```

For the case when the player has inputted a letter that does not exist in the word in the word buffer, the program must take a different course of action. This case is handled by a portion of the program that starts at the label HANGIT. Here the operator is informed of the incorrect guess by the display of the message NOPE. This message is then followed by the display of a portion or all of the HANGMAN! message.

Each time the player guesses incorrectly during a game, a letter is added to the message spelling out the word HANGMAN! In order to do this properly, the program maintains a counter of the number of incorrect guesses made. Then, the computer is simply used to determine how many letters of the HANGMAN! message to display. (The HANGMAN! message is simply stored in a buffer, starting at location 340 on page 03 in the example program.) If the entire HANGMAN! statement is not displayed, the balance of the message is shown as dash signs (hyphens). The number of dash signs to display is cal-

culated by subtracting the value of the counter (of incorrect guesses made) from 10 (octal) which is the number of characters in the HANGMAN! message. Thus, as the game progresses, the message HANGMAN! will appear more and more complete with each incorrect guess as illustrated below.

```
H- - - - -      (First incorrect guess)
HA- - - -       (Second incorrect guess)
HAN- - -        (Third incorrect guess)
   .
   .
HANGMAN!        (Eighth incorrect guess)
```

When eight incorrect guesses have been made in a game, the entire HANGMAN! message will be displayed. The player then loses the game. The program will then go back and see if the player wants to start with a new word.

The listing for the portion of the program that displays the dreaded HANGMAN! message is shown next.

```
HANGIT,   LHI 004      Pointer to NOPE
          LLI 062      Message
          CAL MSG      Display message
          LHI 003      Set pointer to
          LLI 350      Number of guesses in counter
          LBM          Fetch counter
          INB          Increment it
          LMB          Restore the counter
          INL          Advance pointer
          LMB          Save it again
          LAI 010      Calculate number of
          SUB          Dashes left in HANGMAN!
          INL          Advance pointer
          LMA          Save the value in memory
          LEI 340      Init. pntr. to HANGMAN bfr

HANGMR,   LLE          Set pntr to HANGMAN bfr
          LAM          Fetch a character from buffer
          CAL PRINT    Display it
```

```
        INL         Advance the buffer pointer
        LEL         Save temporarily in E
        LLI 351     Pointer to counter in memory
        LBM         Fetch counter value
        DCB         Decrement
        LMB         Restore to memory
        JFZ HANGMR  Continue if counter not zero
        LLI 352     Pntr to second cntr in memory
        LCM         Fetch counter value
        INC         Exercise counter value
        DCC         To see if it is zero
        JTZ NEWONE  Start new game if so
        LAI 255     Else load code for "-"

MRDASH, CAL PRINT   Display a dash
        LCM         Fetch counter
        DCC         Decrement
        LMC         Restore
        JFZ MRDASH  Until counter is zero
        JMP GUESS   Then continue game
```

The next portion of the program is a subroutine mentioned previously called MOVTAB. The primary function of this subroutine is to fetch a new word from a list or table of words stored in memory. However, the subroutine also performs a few other functions that need to be performed each time a new word (actually representing the start of a new game) is obtained.

The first thing the subroutine does is fetch the value of the counter used for keeping track of how many incorrect guesses were made during the last game played. This value will be used to determine how many words in the word table to skip over when selecting a new word. This method is used so that words will be selected from the table in a rather arbitrary fashion rather than simply taking the next word in the list. If the next word in the list was always taken, players might soon start remembering certain words or sequences of words which would soon make the game somewhat boring!

The program then initializes the guesses buffer to the all hyphens

condition by loading the ASCII code for the dash sign (255 octal) into all the locations in the buffer. In a similar fashion, the word buffer is cleared to the all zeroes condition in preparation for its receiving a new word from the word table.

These initial functions of the subroutine are shown below.

```
MOVTAB,   LHI 003       Pointer to number
          LLI 350       Of guesses counter
          LBM           Fetch and save in B
          LLI 370       Pointer to guesses buffer
          LCI 010       Set a loop counter
          LAI 255       Set code for "-"

DASHFL,   LMA           Fill guesses buffer
          INL           With dashes
          DCC           Until counter
          JFZ DASHFL    Is zero

NXWORD,   LLI 360       Set pointer to word buffer
          LCI 010       Set a loop counter
          XRA           Clear the accumulator

ZEROFL,   LMA           Fill word buffer
          INL           With zero bytes
          DCC           Until counter
          JFZ ZEROFL    Is zero
```

Before explaining the operation of the portion of the subroutine that extracts a new word from the word table, it will be beneficial to explain the organization of the table.

The table consists of a list of words stored in memory in the following format.

```
A       1st letter of a word
A+1     2nd letter of a word
A+2     3rd letter of a word
```

.	.
.	.
A+N	Nth letter of a word
A+N+1	000 word terminator code
B	1st letter of a word
B+1	2nd letter of a word
.	.
.	.
B+N	Nth letter of a word
B+N+1	000 word terminator code
C	1st letter of a word
.	.
.	.
C+N+1	000 word terminator code
D	000 end of table terminator

The reader should notice that each word in the list must be terminated by a zero byte. Words must also be limited to eight or less letters in length (or they would overflow the word buffer). The table is terminated by placing an additional zero byte immediately following the zero byte word terminator after the last word in the table. The word table in the program provided in this manual starts on page 05 at location 000. The table may extend for as long as the user desires within available memory. (In the sample word list about 100 words are provided. These require about three pages of memory. Of course, the list may be shortened if necessary. Or the user may provide a completely original table of words.)

A word is extracted from the word table through the following procedure. First, a word table pointer is extracted from its storage location in memory and loaded into CPU registers H and L. This pointer will initially point to the first letter of a word in the table. Next, a character is extracted from the word table. The character obtained is first tested to see if it is a zero byte. A zero byte in place of an expected letter (as the first letter in a word) indicates that the end of the table has been reached. In that case, the pointer in H and L is reset back to the start of the word table.

Next, a second pointer is established in CPU registers D and E.

This pointer will be used to point to the word buffer during transfer operations from the word table. Now a character is fetched from the word table. Then the pointers in H and L and D and E are swapped and the character is transferred into the word buffer. (Unless a zero byte indicating the end of a word is detected. In that case no transfer takes place.) Next, the two sets of pointers are advanced. The process is then repeated until a whole word has been loaded into the word buffer.

When an entire word has been transferred into the word buffer, the routine advances the word table pointer once more. (This is so it will be advanced over the end of word terminator and be pointing at the first letter in the next word in the table.) Then the pointer is restored to its storage location in memory. Next, the routine fetches the number of guesses counter. It decrements the value of that counter. If the value is not zero after the decrement operation, then the routine loops back (to the label NXWORD), and proceeds to read the next word in the word table into the word buffer. (The reason for following this procedure was presented earlier.) When the counter reaches zero, it is stored in memory (at its 000 value) for use during the next game. The subroutine is then exited.

```
            LLI 356         Set pointer to word table pntr
            LAM             Fetch the low address
            INL             Advance pointer
            LHM             Set the page address
            LLA             And low address
            XRA             Clear the accumulator
            CPM             See if first entry is zero
            JFZ AHEAD3      Skip ahead if not
            LHI 005         Reset pointer to start
            LLI 000         Of word table if so

AHEAD3,     LDI 003         Set pointer to word
            LEI 360         Buffer in D and E

BUFFMR,     LAM             Fetch a character from table
            NDA             Exercise flags
            JTZ NEXT        If zero, have whole word
```

```
            CAL SWITCH      Else swap pointers
            LMA             Dep character in word buffer
            INL             Advance word buffer pointer
            CAL SWITCH      Swap pointers
            INL             Advance word table low pntr
            JFZ NOHIGH      If not zero, skip next
            INH             Advance table high pointer

NOHIGH,     JMP BUFFMR      Continue transfer from table

NEXT,       INL             Advance table pointer
            JFZ NOTHI       Low address
            INH             And high address if required

NOTHI,      CAL SWITCH      Save pointer in D and E
            LLI 356         Set pointer to table pointer
            LME             Save table pointer low
            INL             Address and
            LMD             High address
            DCB             Decr number guesses counter
            JFZ NXWORD      If not zero, get next word
            LLI 350         Else set pointer to guesses
            LMI 000         Counter and zero counter
            RET             Then exit subroutine
```

That completes the discussion of the major routines in the program. There are two more minor utility subroutines used in the program. One of these is simply a subroutine called SWITCH that is used to exchange the contents of CPU registers H and L with D and E. During this operation, CPU register C is used as a temporary register.

```
SWITCH,     LCH             Put H into C temporarily
            LHD             Load D into H
            LDC             Now orig H from C to D
            LCL             Put L into C temporarily
            LLE             Load E into L
            LEC             Now orig L from C to E
            RET             Swapping oper. completed
```

The other is the subroutine mentioned earlier called MSG. MSG simply outputs a string of characters from memory to an output device until it detects a zero byte.

```
MSG,    LAM         Fetch a character
        NDA         See if a zero byte
        RTZ         Indicating end of string
        CAL PRINT   If not, display character
        INL         Increment low address pointer
        JFZ MSG     Get next character unless
        INH         Need to advance page address
        JMP MSG     Then get next character
```

The MSG subroutine above calls on another subroutine which has been termed PRINT. The PRINT subroutine must be an actual device operating subroutine that will cause the ASCII character in the accumulator to be transmitted to the output device being used by the system. The PRINT subroutine, which must be provided by the user, may use the CPU registers B through E if required. It should not alter the contents of the H and L CPU registers (unless the subroutine is able to restore those registers to their original values at the conclusion of the process).

ASSEMBLED LISTING OF THE "HANGMAN!" PROGRAM FOR AN '8008'

An assembled listing of the program for operation on an 8008 system is presented next. The operating portion of the program has been assembled to reside in pages 02 and 03. Page 04 is reserved for the various message strings used by the program plus the user provided I/O subroutines. The word table for the program is assumed to start on page 05. A sample list of words for use with the program is provided in ASCII form at the end of the assembled listing.

```
002 000    056 003              START,  LHI 003
002 002    066 350                      LLI 350
002 004    076 001                      LMI 001
002 006    066 356                      LLI 356
002 010    076 000                      LMI 000
002 012    060                          INL
002 013    076 005                      LMI 005

002 015    056 004              NEWONE, LHI 004
002 017    066 000                      LLI 000
002 021    106 110 003                  CAL MSG
002 024    106 200 004                  CAL INPUTN
002 027    074 316                      CPI 316
002 031    150 047 002                  JTZ NOMORE
002 034    074 331                      CPI 331
002 036    110 015 002                  JFZ NEWONE
002 041    106 347 002                  CAL MOVTAB
002 044    104 057 002                  JMP GUESS
002 047    056 004              NOMORE, LHI 004
002 051    066 025                      LLI 025
002 053    106 110 003                  CAL MSG
002 256    000                          HLT

002 057    056 004              GUESS,  LHI 004
002 061    066 037                      LLI 037
002 063    106 110 003                  CAL MSG
002 066    106 200 004                  CAL INPUTN
002 071    320                          LCA

002 072    016 000              SCAN,   LBI 000
002 074    056 003                      LHI 003
002 076    066 360                      LLI 360

002 100    277                  CKMTCH, CPM
002 101    110 116 002                  JFZ NOMTCH
002 104    010                          INB
002 105    006 010                      LAI 010
002 107    206                          ADL
002 110    360                          LLA
002 111    372                          LMC
```

```
002 112   306                        LAL
002 113   024 010                    SUI 010
002 115   360                        LLA

002 116   060              NOMTCH,   INL
002 117   307                        LAM
002 120   240                        NDA
002 121   150 136 002                JTZ EOWORD
002 124   006 007                    LAI 007
002 126   246                        NDL
002 127   150 136 002                JTZ EOWORD
002 132   302                        LAC
002 133   104 100 002                JMP CKMTCH

002 136   010              EOWORD,   INB
002 137   011                        DCB
002 140   150 253 002                JTZ HANGIT
002 143   056 004                    LHI 004
002 145   066 074                    LLI 074
002 147   106 110 003                CAL MSG
002 152   056 003                    LHI 003
002 154   066 353                    LLI 353
002 156   076 000                    LMI 000
002 160   066 370                    LLI 370

002 162   307              NOTEND,   LAM
002 163   074 255                    CPI 255
002 165   110 177 002                JFZ AHEAD2
002 170   346                        LEL
002 171   066 353                    LLI 353
002 173   317                        LBM
002 174   010                        INB
002 175   371                        LMB
002 176   364                        LLE

002 177   106 300 004      AHEAD2,   CAL PRINT
002 202   060                        INL
002 203   306                        LAL
002 204   024 010                    SUI 010
002 206   360                        LLA
```

```
002 207    307                          LAM
002 210    240                          NDA
002 211    150 231 002                  JTZ ENDAGN
002 214    306                          LAL
002 215    044 007                      NDI 007
002 217    150 231 002                  JTZ ENDAGN
002 222    006 010                      LAI 010
002 224    206                          ADL
002 225    360                          LLA
002 226    104 162 002                  JMP NOTEND

002 231    066 353          ENDAGN,     LLI 353
002 233    317                          LBM
002 234    010                          INB
002 235    011                          DCB
002 236    110 057 002                  JFZ GUESS
002 241    056 004                      LHI 004
002 243    066 120                      LLI 120
002 245    106 110 003                  CAL MSG
002 250    104 015 002                  JMP NEWONE

002 253    056 004          HANGIT,     LHI 004
002 255    066 062                      LLI 062
002 257    106 110 003                  CAL MSG
002 262    056 003                      LHI 003
002 264    066 350                      LLI 350
002 266    317                          LBM
002 267    010                          INB
002 270    371                          LMB
002 271    060                          INL
002 272    371                          LMB
002 273    006 010                      LAI 010
002 275    221                          SUB
002 276    060                          INL
002 277    370                          LMA
002 300    046 340                      LEI 340

002 302    364              HANGMR,     LLE
002 303    307                          LAM
002 304    106 300 004                  CAL PRINT
```

```
002 307   060                          INL
002 310   346                          LEL
002 311   066 351                      LLI 351
002 313   317                          LBM
002 314   011                          DCB
002 315   371                          LMB
002 316   110 302 002                  JFZ HANGMR
002 321   066 352                      LLI 352
002 323   327                          LCM
002 324   020                          INC
002 325   021                          DCC
002 326   150 015 002                  JTZ NEWONE
002 331   006 255                      LAI 255

002 333   106 300 004        MRDASH,   CAL PRINT
002 336   327                          LCM
002 337   021                          DCC
002 340   372                          LMC
002 341   110 333 002                  JFZ MRDASH
002 344   104 057 002                  JMP GUESS

002 347   056 003            MOVTAB,   LHI 003
002 351   066 350                      LLI 350
002 353   317                          LBM
002 354   066 370                      LLI 370
002 356   026 010                      LCI 010
002 360   006 255                      LAI 255

002 362   370                DASHFL,   LMA
002 363   060                          INL
002 364   021                          DCC
002 365   110 362 002                  JFZ DASHFL

002 370   066 360            NXWORD,   LLI 360
002 372   026 010                      LCI 010
002 374   250                          XRA

002 375   370                ZEROFL,   LMA
002 376   060                          INL
002 377   021                          DCC
```

```
003 000   110 375 002            JFZ ZEROFL
003 003   066 356                LLI 356
003 005   307                    LAM
003 006   060                    INL
003 007   357                    LHM
003 010   360                    LLA
003 011   250                    XRA
003 012   277                    CPM
003 013   110 022 003            JFZ AHEAD3
003 016   056 005                LHI 005
003 020   066 000                LLI 000

003 022   036 003        AHEAD3, LDI 003
003 024   046 360                LEI 360

003 026   307            BUFFMR, LAM
003 027   240                    NDA
003 030   150 053 003            JTZ NEXT
003 033   106 101 003            CAL SWITCH
003 036   370                    LMA
003 037   060                    INL
003 040   106 101 003            CAL SWITCH
003 043   060                    INL
003 044   110 050 003            JFZ NOHIGH
003 047   050                    INH

003 050   104 026 003    NOHIGH, JMP BUFFMR

003 053   060              NEXT, INL
003 054   110 060 003            JFZ NOTHI
003 057   050                    INH

003 060   106 101 003     NOTHI, CAL SWITCH
003 063   066 356                LLI 356
003 065   374                    LME
003 066   060                    INL
003 067   373                    LMD
003 070   011                    DCB
003 071   110 370 002            JFZ NXWORD
003 074   066 350                LLI 350
```

```
003 076    076 000                        LMI 000
003 100    007                            RET

003 101    325                 SWITCH,    LCH
003 102    353                            LHD
003 103    332                            LDC
003 104    326                            LCL
003 105    364                            LLE
003 106    342                            LEC
003 107    007                            RET

003 110    307                    MSG,    LAM
003 111    240                            NDA
003 112    053                            RTZ
003 113    106 300 004                    CAL PRINT
003 116    060                            INL
003 117    110 110 003                    JFZ MSG
003 122    050                            INH
003 123    104 110 003                    JMP MSG

003 340    310                            310
003 341    301                            301
003 342    316                            316
003 343    307                            307
003 344    315                            315
003 345    301                            301
003 346    316                            316
003 347    241                            241

003 350    000                            000
003 351    000                            000
003 352    000                            000
003 353    000                            000

003 356    000                            000
003 357    000                            000

003 360    000                            000
003 361    000                            000
003 362    000                            000
```

```
003 363    000                                   000
003 364    000                                   000
003 365    000                                   000
003 366    000                                   000
003 367    000                                   000

003 370    255                                   255
003 371    255                                   255
003 372    255                                   255
003 373    255                                   255
003 374    255                                   255
003 375    255                                   255
003 376    255                                   255
003 377    255                                   255


        004 000   215 212 212 327 301 316 324 240
        004 010   301 240 316 305 327 240 327 317
        004 020   322 304 277 240 000 215 212 307
        004 030   317 317 304 302 331 241 000 215
        004 040   212 307 325 305 323 323 240 301
        004 050   240 314 305 324 324 305 322 272
        004 060   240 000 215 212 316 317 320 305
        004 070   241 240 240 000 215 212 307 317
        004 100   317 304 256 240 331 317 325 240
        004 110   310 301 326 305 272 240 240 000
        004 120   215 212 303 317 316 307 322 301
        004 130   304 325 314 301 324 311 317 316
        004 140   323 241 000

004 200                                   INPUTN,

004 300                                   PRINT,


        005 000   310 305 314 314 317 000 301 322
        005 010   322 317 327 000 303 317 315 320
        005 020   325 324 305 322 000 320 322 305
        005 030   315 311 325 315 000 316 317 324
        005 040   311 303 305 000 306 325 316 000
```

005	050	310	305	301	326	331	000	322	325
005	060	302	302	305	322	000	322	325	323
005	070	324	314	305	000	324	310	327	301
005	100	322	324	000	317	331	323	324	305
005	110	322	000	317	330	311	304	311	332
005	120	305	000	317	323	323	311	306	331
005	130	000	317	320	311	316	311	317	316
005	140	000	317	317	332	331	000	317	316
005	150	305	322	317	325	323	000	316	317
005	160	315	301	304	000	316	317	303	324
005	170	325	322	316	305	000	316	317	315
005	200	311	316	301	324	305	000	316	325
005	210	315	323	313	325	314	314	000	304
005	220	301	306	306	317	304	311	314	000
005	230	323	311	304	305	322	305	301	314
005	240	000	303	322	311	303	313	305	324
005	250	000	303	317	325	322	311	305	322
005	260	000	303	317	323	315	317	323	000
005	270	303	310	305	315	311	323	324	000
005	300	303	310	305	315	311	303	301	314
005	310	000	303	310	311	303	317	322	331
005	320	000	303	310	314	317	322	311	316
005	330	305	000	303	311	324	311	332	305
005	340	316	000	303	311	324	322	325	323
005	350	000	303	314	317	323	305	324	000
005	360	303	317	307	305	316	324	000	302
005	270	311	322	304	000	302	305	305	324
006	000	314	305	000	302	305	314	311	305
006	010	326	305	000	302	301	324	310	324
006	020	325	302	000	302	301	323	313	305
006	030	324	000	302	301	316	321	325	305
006	040	324	000	302	301	302	302	311	324
006	050	324	000	302	301	303	313	302	317
006	060	316	305	000	301	325	304	311	302
006	070	314	305	000	301	323	320	311	322
006	100	311	316	000	301	323	324	305	322
006	110	317	311	304	000	301	320	320	322
006	120	317	326	301	314	000	301	320	317
006	130	307	305	305	000	301	316	316	325
006	140	311	324	331	000	301	316	317	304

006	150	311	332	305	000	301	314	325	315
006	160	311	316	325	315	000	301	311	322
006	170	000	301	311	323	314	305	000	301
006	200	304	312	317	311	316	000	301	302
006	210	331	323	323	000	301	302	317	314
006	220	311	323	310	000	321	325	305	325
006	230	305	000	321	325	311	326	305	322
006	240	000	321	325	301	314	315	000	321
006	250	325	311	324	305	000	321	325	311
006	260	330	317	324	311	303	000	321	325
006	270	317	311	316	000	321	325	317	311
006	300	324	000	321	325	317	324	311	305
006	310	316	324	000	322	301	304	311	317
006	320	000	322	301	311	323	311	316	000
006	330	322	301	320	324	000	322	301	324
006	340	311	317	000	322	301	325	303	317
006	350	325	323	000	322	301	331	317	316
006	360	000	322	301	332	317	322	000	322
006	370	305	301	314	315	000	322	305	305
007	000	313	000	322	305	307	311	323	324
007	010	305	322	000	322	311	326	305	324
007	020	000	323	303	310	317	317	316	305
007	030	322	000	323	301	325	316	301	000
007	040	323	301	324	311	316	000	323	303
007	050	305	320	324	305	322	000	323	303
007	060	311	305	316	303	305	000	323	303
007	070	322	311	302	302	314	305	000	302
007	100	305	310	311	316	304	000	304	311
007	110	307	316	311	306	331	000	305	314
007	120	314	311	320	324	311	303	000	305
007	130	314	317	321	325	305	316	324	000
007	140	305	314	325	323	311	326	305	000
007	150	306	305	301	324	310	305	322	000
007	160	307	301	314	314	317	327	323	000
007	170	307	301	322	304	305	316	000	307
007	200	301	332	305	314	314	305	000	315
007	210	301	303	301	302	322	305	000	326
007	220	301	314	311	301	316	324	000	326
007	230	305	316	311	323	317	316	000	326
007	240	311	326	311	304	000	327	305	311

```
007 250   307 310 324 000 327 305 311 322
007 260   304 000 327 311 323 305 000 332
007 270   305 322 317 000 332 317 317 314
007 300   317 307 331 000 332 305 316 311
007 310   324 310 000 331 301 327 316 000
007 320   331 317 314 313 000 331 305 314
007 330   314 317 327 000 331 325 314 305
007 340   000 324 322 311 303 313 314 305
007 350   000 305 316 304 000 000
```

A list of the messages used in the game (which reside on page 04 in the assembled listing just presented) is shown below in the order in which they appear in the messages table.

WANT A NEW WORD?

GOODBYE!

GUESS A LETTER:

NOPE!

GOOD. YOU HAVE:

CONGRATULATIONS!

For those that want to use the word list supplied as an example, (pages 05 through 07 in the listing just presented) the list on the following page will serve as a reference. The words appear in the same order as they are stored in the list. (Remember, however, that the program will skip around the list as it selects the next word that will be played!)

HELLO
ARROW
COMPUTER
PREMIUM
NOTICE
FUN
HEAVY
RUBBER
RUSTLE
THWART
OYSTER
OXIDIZE
OSSIFY
OPINION
OOZY
ONEROUS
NOMAD
NOCTURNE
NOMINATE
NUMSKULL
DAFFODIL
SIDEREAL
CRICKET
COURIER
COSMOS
CHEMIST
CHEMICAL
CHICORY
CHLORINE
CITIZEN
CITRUS
CLOSET
COGENT
BIRD
BEETLE
BELIEVE
BATHTUB
BASKET
BANQUET
BABBITT
BACKBONE
AUDIBLE
ASPIRIN
ASTEROID
APPROVAL
APOGEE
ANNUITY
ANODIZE
ALUMINUM
AIR
AISLE
ADJOIN
ABYSS
ABOLISH
QUEUE
QUIVER
QUALM
QUITE
QUIXOTIC
QUOIN
QUOIT
QUOTIENT
RADIO
RAISIN
RAPT
RATIO
RAUCOUS
RAYON
RAZOR
REALM
REEK
REGISTER
RIVET
SCHOONER
SAUNA
SATIN
SCEPTER
SCIENCE
SCRIBBLE
BEHIND
DIGNIFY
ELLIPTIC
ELOQUENT
ELUSIVE
FEATHER
GALLOWS
GARDEN
GAZELLE
MACABRE
VALIANT
VENISON
VIVID
WEIGHT
WEIRD
WISE
ZERO
ZOOLOGY
ZENITH
YAWN
YOLK
YELLOW
YULE
TRICKLE
END

Once the program has been loaded into memory (along with the user provided I/O routines!) the program is ready to operate. Simply start program execution at page 02 location 000. Operation from then on is directed by the program. A sample of the program's operation is illustrated below.

```
WANT A NEW WORD?  Y
GUESS A LETTER:   A
NOPE!  H-----
GUESS A LETTER:   E
GOOD. YOU HAVE:  -E---
GUESS A LETTER:   R
NOPE!  HA----
GUESS A LETTER:   L
GOOD. YOU HAVE:  -ELL-
GUESS A LETTER:   B
NOPE!  HAN---
GUESS A LETTER:   S
NOPE!  HANG--
GUESS A LETTER:   O
GOOD. YOU HAVE:  -ELLO
GUESS A LETTER:   H
GOOD. YOU HAVE:  HELLO
CONGRATULATIONS!

WANT A NEW WORD?  Y
GUESS A LETTER:   A
NOPE!  H-----
GUESS A LETTER:   E
GOOD. YOU HAVE:  -----E
GUESS A LETTER:   I
GOOD. YOU HAVE:  ---I-E
GUESS A LETTER:   O
GOOD. YOU HAVE:  -O-I-E
GUESS A LETTER:   U
NOPE!  HA----
```

```
GUESS A LETTER:   R
NOPE!  HAN- - -
GUESS A LETTER:   T
GOOD. YOU HAVE:  - OTI - E
GUESS A LETTER:   N
GOOD. YOU HAVE:  NOTI - E
GUESS A LETTER:   C
GOOD. YOU HAVE:  NOTICE
CONGRATULATIONS!


WANT A NEW WORD?  Y
GUESS A LETTER:   W
NOPE!  H- - - - -
GUESS A LETTER:   Y
NOPE!  HA- - - -
GUESS A LETTER:   P
NOPE!  HAN- - -
GUESS A LETTER:   N
NOPE!  HANG- -
GUESS A LETTER:   G
NOPE!  HANGM- -
GUESS A LETTER:   V
NOPE!  HANGMA-
GUESS A LETTER:   C
NOPE!  HANGMAN-
GUESS A LETTER:   X
NOPE!  HANGMAN!


WANT A NEW WORD?  N
GOODBYE!
```

The program will continue to operate until a player responds with a N for NO to the WANT A NEW WORD query. At that time, the program will halt. If it is desired to continue playing after a NO response to that question, the program may simply be restarted at the starting address (page 02 location 000).

ASSEMBLED LISTING OF THE PROGRAM FOR AN 8080 SYSTEM

The following is an assembled listing of the HANGMAN! program designed to run on an 8080 system. Only minor changes have been made in the program to take advantage of some of the special capabilities of the 8080 instruction set. However, the basic organization of the program has not been altered so that the previous detailed discussion of the program's operation still applies. The message table and word list would be in the same format as the 8008 version. Of course, the user will need to provide the appropriate I/O routines for either version of the program. They can be placed in the same memory locations for the following 8080 version as was suggested for the 8008 example (on page 04 starting at locations 200 (input) and 300 (output)).

```
002 000   041 350 003        START,  LXH 350 003
002 003   066 001                    LMI 001
002 005   056 356                    LLI 356
002 007   066 000                    LMI 000
002 011   054                        INL
002 012   066 005                    LMI 005

002 014   061 200 004        NEWONE, LXS 200 004
002 017   041 000 004                LXH 000 004
002 022   315 032 003                CAL MSG
002 025   315 200 004                CAL INPUTN
002 030   376 316                    CPI 316
002 032   312 050 002                JTZ NOMORE
002 035   376 331                    CPI 331
002 037   302 014 002                JFZ NEWONE
002 042   315 327 002                CAL MOVTAB
002 045   303 057 002                JMP GUESS

002 050   041 025 004        NOMORE, LXH 025 004
002 053   315 032 003                CAL MSG
002 056   166                        HLT
```

```
002 057   041 037 004          GUESS,  LXH 037 004
002 062   315 032 003                  CAL MSG
002 065   315 200 004                  CAL INPUTN
002 070   117                          LCA

002 071   006 000              SCAN,   LBI 000
002 073   041 360 003                  LXH 360 003

002 076   276                  CKMTCH, CPM
002 077   302 114 002                  JFZ NOMTCH
002 102   004                          INB
002 103   076 010                      LAI 010
002 105   205                          ADL
002 106   157                          LLA
002 107   161                          LMC
002 110   175                          LAL
002 111   326 010                      SUI 010
002 113   157                          LLA

002 114   054                  NOMTCH, INL
002 115   176                          LAM
002 116   247                          NDA
002 117   312 134 002                  JTZ EOWORD
002 122   076 007                      LAI 007
002 124   245                          NDL
002 125   312 134 002                  JTZ EOWORD
002 130   171                          LAC
002 131   303 076 002                  JMP CKMTCH

002 134   004                  EOWORD, INB
002 135   005                          DCB
002 136   312 243 002                  JTZ HANGIT
002 141   041 074 004                  LXH 074 004
002 144   315 032 003                  CAL MSG
002 147   041 353 003                  LXH 353 003
002 152   066 000                      LMI 000
002 154   056 370                      LLI 370

002 156   176                  NOTEND, LAM
002 157   376 255                      CPI 255
```

```
002 161   302 171 002              JFZ AHEAD2
002 164   135                      LEL
002 165   056 353                  LLI 353
002 167   064                      INM
002 170   153                      LLE

002 171   315 300 004    AHEAD2,   CAL PRINT
002 174   054                      INL
002 175   175                      LAL
002 176   326 010                  SUI 010
002 200   157                      LLA
002 201   176                      LAM
002 202   247                      NDA
002 203   312 223 002              JTZ ENDAGN
002 206   175                      LAL
002 207   346 007                  NDI 007
002 211   312 223 002              JTZ ENDAGN
002 214   076 010                  LAI 010
002 216   205                      ADL
002 217   157                      LLA
002 220   303 156 002              JMP NOTEND
002 223   056 353        ENDAGN,   LLI 353
002 225   064                      INM
002 226   065                      DCM
002 227   302 057 002              JFZ GUESS
002 232   041 120 004              LXH 120 004
002 235   315 032 003              CAL MSG
002 240   303 014 002              JMP NEWONE

002 243   041 062 004    HANGIT,   LXH 062 004
002 246   315 032 003              CAL MSG
002 251   041 350 003              LXH 350 003
002 254   064                      INM
002 255   176                      LAM
002 256   054                      INL
002 257   167                      LMA
002 260   076 010                  LAI 010
002 262   226                      SUM
002 263   054                      INL
002 264   167                      LMA
```

```
002 265    036 340                          LEI 340

002 267    153                    HANGMR,   LLE
002 270    176                              LAM
002 271    315 300 004                      CAL PRINT
002 274    054                              INL
002 275    135                              LEL
002 276    056 351                          LLI 351
002 300    065                              DCM
002 301    302 267 002                      JFZ HANGMR
002 304    056 352                          LLI 352
002 306    064                              INM
002 307    065                              DCM
002 310    312 014 002                      JTZ NEWONE
002 313    076 255                          LAI 255

002 315    315 300 004            MRDASH,   CAL PRINT
002 320    065                              DCM
002 321    302 315 002                      JFZ MRDASH
002 324    303 057 002                      JMP GUESS

002 327    041 350 003            MOVTAB,   LXH 350 003
002 332    106                              LBM
002 333    056 370                          LLI 370
002 335    016 010                          LCI 010
002 337    076 255                          LAI 255

002 341    167                    DASHFL,   LMA
002 342    054                              INL
002 343    015                              DCC
002 344    302 341 002                      JFZ DASHFL

002 347    041 360 003            NXWORD,   LXH 360 003
002 352    016 010                          LCI 010
002 354    257                              XRA

002 355    167                    ZEROFL,   LMA
002 356    054                              INL
002 357    015                              DCC
002 360    302 355 002                      JFZ ZEROFL
```

```
002 363   052 356 003                  LHLD 356 003
002 366   257                          XRA
002 367   276                          CPM
002 370   302 376 002                  JFZ AHEAD3
002 373   041 000 005                  LXH 000 005

002 376   021 360 003          AHEAD3, LXD 360 003

003 001   176                  BUFFMR, LAM
003 002   247                          NDA
003 003   312 014 003                  JTZ NEXT
003 006   022                          STAD
003 007   023                          INXD
003 010   043                          INXH
003 011   303 001 003                  JMP BUFFMR

003 014   043                    NEXT, INXH
003 015   042 356 003                  SHLD 356 003
003 020   005                          DCB
003 021   302 347 002                  JFZ NXWORD
003 024   041 350 003                  LXH 350 003
003 027   066 000                      LMI 000
003 031   311                          RET

003 032   176                    MSG,  LAM
003 033   247                          NDA
003 034   310                          RTZ
003 035   315 300 004                  CAL PRINT
003 040   043                          INXH
003 041   303 032 003                  JMP MSG
```

PUBLICATIONS FROM SCELBI COMPUTER CONSULTING, INC.

MACHINE LANGUAGE PROGRAMMING for the '8008'
(and similar microcomputers). $19.95

ASSEMBLER PROGRAMS FOR THE '8008'. $17.95

AN '8008' EDITOR PROGRAM. $14.95

'8008' MONITOR ROUTINES $11.95

AN '8080' ASSEMBLER PROGRAM. $17.95

AN '8080' EDITOR PROGRAM. $14.95

'8080' MONITOR ROUTINES $11.95

SCELBI'S FIRST BOOK OF COMPUTER GAMES
FOR THE '8008'/'8080'. $14.95

SCELBI'S GALAXY GAME FOR THE 8008/8080 . . . . . $14.95

THE ABOVE PUBLICATIONS MAY BE ORDERED
DIRECTLY FROM:

SCELBI COMPUTER CONSULTING, INC.
1322 Rear - Boston Post Road
Milford, CT 06460