First Steps in Computer Chess Programming

The fascination of chess gains a new dimension with microcomputer chess. No longer are the struggles confined to giant machines. With the advent of the Chess Mate, Chess Challenger, Boris, and Computer chess, as well as some custom software packages, the day of microcomputer chess has dawned. Writing a program to play chess on a small system is no small matter, though. Consider just for a start the challenge of meaningfully representing the board and its pieces in computer memory: there are 64 squares, 32 pieces, 6 piece types and 2 piece colors. Since the machine is a microcomputer, storage requirements must be kept to a minimum. Next comes the job of moving the pieces. Only when these first problems of piece representation and move generation have been solved can the chess programmer go on to consider strategy.

Sargon, a chess playing program we developed for Z-80 machines, solves the representation problem through the use of a board array. Move generation is accomplished through a network of routines diagrammed in figure 1. The functions of the routines are as follows:

- **GENMOV**: Generate move routine. Generates the move set for all of the pieces of a given color.
- **MPIECE**: Piece mover routine. Generates the move set for a given piece.
- **INCHK**: Check routine. Determines whether or not the King is in check.
- **PATH**: Path routine. Generates a single possible move for a given piece along its current path of motion.
- **ADMOVE**: Admove routine. Adds a move to the move list.
- **CASTLE**: Castle routine. Determines whether castling is legal and adds it to the move list if it is.
- **ENPSNT**: En passant routine. Tests for an en passant pawn capture and adds it to the move lists if it is legal.
- **ATTACK**: Attack routine. Finds all the attackers on a given square.
- **ADJPTR**: Adjust move list pointer. Links around the second move in a double move (i.e., castle or en passant pawn capture).
- **ATKSAV**: Attack save routine. Saves attacking piece value in the attack list and increments the attack count for that color piece.
- **PNCK**: Pin check routine. Checks to see if an attacking piece is in the pinned piece list.

Several of the routines involved are multipurpose routines. Their involvement in move generation is incidental to a main function elsewhere in the move selection logic. The key routines in move generation are MPIECE, PATH, CASTLE and ENPSNT. Of these,
Figure 2: Decimal (a) and hexadecimal (b) representations of the chessboard used in the Sargon program. Each square of the board is represented by a single byte in memory. Border squares are assigned a flag value of hexadecimal FF. The use of the border simplifies move generation, since it becomes easy to determine when a piece moves off the board.

MPiece and PATH will be discussed here. The routines will be described in a language independent narrative. The Z-80 assembler code in which they are implemented will also be presented and exhaustively commented.

The Board in Memory

The chessboard in memory is an array of 120 bytes that can be visualized as in figure 2. Each square of the board is represented in memory by a single byte. Border bytes are assigned a flag value of hexadecimal FF. The border simplifies move generation, since it becomes easy to determine when a piece moves off the board.

The Pieces in Memory

Each piece is represented in memory by one byte of data. The meaning and function of the bits are as follows:

- **Bit 7** - color of the piece.
  - 1 - Black
  - 0 - White

- **Bit 6** - not used.

- **Bit 5** - not used.

- **Bit 4** - castle flag for Kings only.
  - Set if the King has castled.

- **Bit 3** - moved flag.
  - Set if the piece has moved.

- **Bits 2-0** - Piece type.
  - 1 Pawn
  - 2 Knight
  - 3 Bishop
  - 4 Rook
  - 5 Queen
  - 6 King

The pieces in play occupy squares of the board. If a board square is empty, it has the value 00. Thus the board setup for play would be as shown in figure 3.

**Piece Mover Data Base**

In order to generate moves for the pieces on the board, data must be maintained to describe the possibilities for each piece. This is accomplished through the use of three tables. Values for the tables are given in table 1.

<table>
<thead>
<tr>
<th>TABLE</th>
<th>FUNCTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIRECT</td>
<td>Direction Table. Used to determine the direction of movement of each piece.</td>
</tr>
<tr>
<td>DPOINT</td>
<td>Direction Table Pointer. Used to determine where to begin in the direction table for any given piece.</td>
</tr>
<tr>
<td>DCOUNT</td>
<td>Direction Table Counter. Used to determine the number of directions of movement for any given piece.</td>
</tr>
</tbody>
</table>

Figure 3: Representation of the pieces on their home squares. Pieces are identified by means of unique byte values.

---

About the Authors

Dan and Kathe Spracklen are the creators of Sargon, the microcomputer chess program that won the microcomputer chess tournament at the 1978 West Coast Computer Faire. Dan Spracklen is a 13 year programming veteran. His experience ranges from scientific simulation programs to real time commercial applications. He is currently a senior applications analyst for Sperry-Univac. Kathe Spracklen is a graduate student in computer science at San Diego State University. An experienced tournament player, Kathe provided the chess background for Sargon.
Listing 1: The Sargon move generation routine, written in Z-80 assembly language.

; EQUATES
PAWN = 1
KNIGHT = 2
BISHOP = 3
ROOK = 4
QUEEN = 5
KING = 6
WHITE = 0
BLACK = 80H
BPAWN = BLACK+PAWN

; TABLES SECTION

START:
.LOC START=80H
TBASE = START+100H

; DIRECT - DIRECTION TABLE
DIRECT = -TBASE

.BYTE +09,+11,-11,-09
.BYTE +10,-10,+01,-01
.BYTE -21,-12,+08,+19
.BYTE +21,+12,-08,-19
.BYTE +10,+10,+11,+09

.DPOINT-DIRECTION TABLE POINTER
DPOINT = -TBASE

.BYTE 20,16,8,0,4,0,0

.DCOUNT-DIRECTION TABLE COUNTER
DCOUNT = -TBASE

.BYTE 4,4,4,4,4,8,8

; BOARD - BOARD ARRAY
BOARD = -TBASE
BOARDA .BLKB 120

; TABLE INDICES SECTION

.LOC START+0
M1: .WORD TBASE
M2: .WORD TBASE
M3: .WORD TBASE
M4: .WORD TBASE
T1: .WORD TBASE
T2: .WORD TBASE
T3: .WORD TBASE

; VARIABLES SECTION

P1: .BYTE 0
P2: .BYTE 0
P3: .BYTE 0

; PIECE MOVER ROUTINE

MPIECE: XRA M

ANI 87H
CPI BPAWN

Equate statements supply symbolic equivalents for the piece types and colors.

Start is the first address in Sargon and should lie on an even 256 byte page boundary.

Indexing in the Z-80 makes use of an address, contained in either the IX or IV index registers, plus a displacement. The displacement is a signed number +127 to −128. Thus a 256 byte area of memory centered on the index address is accessible. For this reason TBASE is placed in the middle of the tables section.

The value of "I" is the current program counter. Direct is now the displacement of the direction table from the table base. So if the value of TBASE is loaded in the IV index register, "DIRECT(Y)" will reference the first element in the direction table.

Diagonal directions used for Bishop, Queen, and King. Rank and file directions used for Rook, Queen, and King. Knight move directions.

White pawn directions including two forward moves and two diagonal moves for captures. Black pawn directions.

Displacement from table base.
Starting point in direction table. In the order BP, WP, N, B, R, Q, K.

Number of directions to use from table. In the same order as DPOINT.

The board array consists of 120 bytes in memory.

Uses the area of memory between START and START+80H.
These indices are used to index into the various tables. Since TBASE is on an even boundary, its address is of the form XXX0, where XX depends on the load address. The table address needed for a particular routine is formed by storing a one byte value in the 00 portion. Since addresses are stored in memory with the low order byte first, XXX0 would be stored as 00XX. Then changing the 00 portion is simply a matter of storing a one byte value in the index.

Working storage area to hold the contents of the board array for a given square.

Gets the piece to be moved into register A. In GENMOV, the routine which calls MPIECE, the piece value in register A, had been exclusive ORed with COLOR, the color of the piece to determine whether or not to call MPIECE. Another exclusive OR restores the piece.
This clears all the flag bits and leaves just piece type and color.
Is it a Black pawn?
Listing 1, continued (Listing 1 is concluded on page 95):

<table>
<thead>
<tr>
<th>Instruction</th>
<th>MNEMONIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>JRNZ</td>
<td>MP2</td>
</tr>
<tr>
<td>DCR</td>
<td>A</td>
</tr>
<tr>
<td>ANI</td>
<td>7</td>
</tr>
<tr>
<td>STA</td>
<td>T1</td>
</tr>
<tr>
<td>LIYD</td>
<td>T1</td>
</tr>
<tr>
<td>MOV</td>
<td>B,DCOUNT(Y)</td>
</tr>
<tr>
<td>MOV</td>
<td>A,DPPOINT(Y)</td>
</tr>
<tr>
<td>STA</td>
<td>INDX2</td>
</tr>
<tr>
<td>LIYD</td>
<td>INDX2</td>
</tr>
<tr>
<td>MOE</td>
<td>C,DIRECT(Y)</td>
</tr>
<tr>
<td>LDA</td>
<td>M1</td>
</tr>
<tr>
<td>STA</td>
<td>M2</td>
</tr>
<tr>
<td>CALL</td>
<td>PATH</td>
</tr>
<tr>
<td>CPI</td>
<td>2</td>
</tr>
<tr>
<td>JRNZ</td>
<td>MP15</td>
</tr>
<tr>
<td>ANA</td>
<td>A</td>
</tr>
<tr>
<td>EXAF</td>
<td></td>
</tr>
<tr>
<td>LDA</td>
<td>T1</td>
</tr>
<tr>
<td>CPI</td>
<td>PAWN+1</td>
</tr>
<tr>
<td>JRC</td>
<td>MP20</td>
</tr>
<tr>
<td>CALL</td>
<td>ADMOVE</td>
</tr>
<tr>
<td>EXAF</td>
<td></td>
</tr>
<tr>
<td>JRNZ</td>
<td>MP15</td>
</tr>
<tr>
<td>LDA</td>
<td>T1</td>
</tr>
<tr>
<td>CPI</td>
<td>KING</td>
</tr>
<tr>
<td>JRZ</td>
<td>MP15</td>
</tr>
<tr>
<td>CPI</td>
<td>BISHOP</td>
</tr>
<tr>
<td>JRNZ</td>
<td>MP10</td>
</tr>
<tr>
<td>INX</td>
<td>Y</td>
</tr>
<tr>
<td>DJNZ</td>
<td>MP5</td>
</tr>
<tr>
<td>LDA</td>
<td>T1</td>
</tr>
<tr>
<td>CPI</td>
<td>KING</td>
</tr>
<tr>
<td>SET</td>
<td>CASTLE</td>
</tr>
</tbody>
</table>

**********PAWN LOGIC**********

<table>
<thead>
<tr>
<th>Instruction</th>
<th>MNEMONIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MP20:</td>
<td>MOV A,B</td>
</tr>
<tr>
<td>CPI</td>
<td>3</td>
</tr>
<tr>
<td>JRC</td>
<td>MP35</td>
</tr>
<tr>
<td>JRZ</td>
<td>MP30</td>
</tr>
<tr>
<td>EXAF</td>
<td></td>
</tr>
<tr>
<td>JRNZ</td>
<td>MP15</td>
</tr>
<tr>
<td>LDA</td>
<td>M2</td>
</tr>
<tr>
<td>CPI</td>
<td>91</td>
</tr>
<tr>
<td>JRNZ</td>
<td>MP25</td>
</tr>
<tr>
<td>CPI</td>
<td>29</td>
</tr>
<tr>
<td>JRNZ</td>
<td>MP26</td>
</tr>
<tr>
<td>MP25:</td>
<td>LXI H,P2</td>
</tr>
<tr>
<td>SET</td>
<td>5,M</td>
</tr>
</tbody>
</table>

No—Skip.
Decrement, making piece type a 0 for a Black pawn. Clears color bit and leaves just the piece type.
This is the first step in forming the index into DPOINT and DCOUNT. T1 contains the value of TBASE (XX00) stored in low-high order (00XX). After storing the piece type (0–6) in T1, it contains the address of TBASE + TYPE.
This operation loads the entire TBASE + TYPE address into the IY index register.
DCOUNT is the replacement from TBASE to the start of the direction count table. So DCOUNT + TBASE is the starting address of the direction count table. Then DCOUNT(Y) is:

DCOUNT + CONTENTS IY Register = DCOUNT + TBASE + TYPE (0–6)

This move instruction pulls the direction count for the given piece type and places it in register B.
Similarly, this instruction pulls the direction table pointer for the given piece type and places it in register A.
The direction table pointer will be used to index into the direction table.
GETS the direction and places it in register C.
The "from" position which was stored in M1 in GENMOV.
Save in M2 to form the address of the current position.
Generate a single move in the given direction.
Did the moving piece encounter a piece of the same color, or is new position off the board?
Jump if yes to either question. No move to add to move list. Ready for new direction.
Was the square moved to empty?
Save the answer to this question by swapping flag register for alternate flag register.
Get type of moving piece.
Is it a pawn?
If so, jump to special pawn handling logic. PAWN+1 is equal to the number 2. A White pawn would be of type 1 while a Black pawn would have type set to 0. In either case the carry flag would be set upon a comparison to a value of 2.
Valid move, so add it to the move list.
Restore the answer to the empty square question.
If it is not empty, go get ready for next direction. No further moves are possible in this direction.
Get piece type. Some pieces may only make one move in a given direction.
The King is such a piece. Is this a piece a King?
If so, go get ready for a new direction.
Compare piece type to a Bishop.
If piece type is bishop or greater (ie: Bishop, Rook, or Queen) go make another move in this same direction.
Increment direction index for next direction in the direction table.
Decrement the direction count (in register B). If count is not yet 0, go back and repeat this process for the new direction. Otherwise all of the directions have been considered.
Fetch piece type again.
Is it a King?
If so, call castle to add it to the move list if legal.
Return to GENMOV.

Get the number of move directions left to consider. If this is the first direction, register A = 4.
Are there three directions left to look at?
A carry on this compare indicates a diagonal move. If so, branch to diagonal logic.
Equality on this compare indicates a forward move of two squares.
Branch to check for legality.
Otherwise this is a forward move of one square. Restore the answer to the empty square question.
If the square is not empty, this is not a valid move. Go check the next direction.
Get the "to" position of the move.
Is it on the last rank and therefore a promotion of a White pawn?
If so, go set promotion flag.
Otherwise, is it on the first rank and therefore a promotion of a Black pawn?
If no, skip setting flag.
Load the address of the promotion flag.
Set the flag (bit 5 of P2).
**3 Reasons To Always Choose Computer Enterprises:**

1. Quality Merchandise
2. Lowest Prices
3. NEW! No-Risk Guarantee

**Take A Look At Our Latest Line Of Products From MICROPOLIS**

**ALL ASSEMBLED:**

<table>
<thead>
<tr>
<th>DISKETTES</th>
<th>Credit Price</th>
<th>Cash Discount Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1081-05 Package of 5 Micropolis diskettes (5-1/2&quot;) for use with both Mod I and Mod II drives.</td>
<td>33</td>
<td>32</td>
</tr>
<tr>
<td>DAISY CHAIN CABLES AND ACCESSORIES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1083-02 Daisy chain interface cable B, with 3 connectors for use with 2 storage modules attached to controller.</td>
<td>33</td>
<td>32</td>
</tr>
<tr>
<td>1083-04 Daisy chain interface cable D, with 5 connectors for use with 4 storage modules attached to controller.</td>
<td>61</td>
<td>59</td>
</tr>
</tbody>
</table>

**METAFLOPPY DISK SUBSYSTEMS FOR THE S-100 BUS.**

| 1053 Mod II Two-disk system with 630,000 bytes (formatted) | 1,774 | 1,706 |
| 1043 Mod II One-disk system with 315,000 bytes (formatted) | 1,072 | 1,031 |

**METAFLOPPY ADD-ON STORAGE MODULES**

<table>
<thead>
<tr>
<th>METAFLOPPY ADD-ON STORAGE MODULES</th>
<th>Credit Price</th>
<th>Cash Discount Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1023 Mod II One-disk 315,000 byte add-on storage module with enclosure and power supply. Requires Daisy chain cable.</td>
<td>604</td>
<td>581</td>
</tr>
</tbody>
</table>

**MACROFLOPPY DISK SUBSYSTEMS FOR THE S-100 BUS.**

<p>| 1042 Mod I One-disk system with 143,000 bytes (formatted) | 744 | 716 |</p>
<table>
<thead>
<tr>
<th>MACROFLOPPY ADD-ON STORAGE MODULES</th>
<th>Credit Price</th>
<th>Cash Discount Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>1022 Mod I One-disk 143,000 byte add-on storage module with enclosure and power supply. Requires Daisy chain cable.</td>
<td>510</td>
<td>491</td>
</tr>
</tbody>
</table>

**NO-RISK GUARANTEE**

- Cancellation With No Obligation If WE Don't Deliver in 90 Days.
- Full Satisfaction Or You May Return Product.
- Plus All Manufacturers' Factory Guarantees.

Shipping charges: $10 per CPU on larger units; $1.50 per kit. $2.00 min. per order. Delivery is stock to 30 days on most items. Shipment is immediate for payment by cashier's check, money order or charge card. Allow 3 weeks for personal checks to clear. N.Y. State residents add appropriate sales tax. Availability, prices and specs may change without notice.

**Operating Hours:**

Mon - Fri 10-5 EST
Thu - Fri 10-9 EST
Closed Sat & Sun

P. O. Box 71 Fayetteville, N.Y. 13066
Phone (315) 637-6208 Today!

---

The subroutines in Sargon that handle the actual move generation rely heavily on the indexing capabilities of the Z-80 microprocessor. For this purpose several sets of indices are maintained to access elements of the tables. The piece mover routines depend especially on the following groups of indices.

- **M1-M4:** Working indices used to index into the board array.
- **T1-T3:** Working indices used to index into direction count, direction value, and piece value tables.
- **INDX1:** General working indices.
- **INDX2:** Used for various purposes.

Variables and constants used in the routines PATH and MPiece include:

- **PAWN:** Identification of the piece types is made.
- **BISHOP:** Through use of.
- **ROOK:** Equate statements.
- **QUEEN:** Numbers are hexadecimal.
- **WHITE:** Decimal.
- **BLACK:** 80
- **BPawn:** Black + pawn
- **PI-P3:** Working area to hold the contents of the board array for a given square.

**Table 1:** Direction table (a) and direction table pointer and counter (b). In order to generate moves for the chess pieces, data describing the possibilities for each piece is kept in Table 1a. Table 1b shows the direction table pointer, which tells where to start in the table for a given piece, and the direction table counter, which determines the number of directions of movement for a given piece.
KNIGHT BEGINS AT WHITE'S QB3
DECIMAL BOARD ARRAY INDEX = 43

<table>
<thead>
<tr>
<th>Direction Table Values</th>
<th>Direction Calculation</th>
<th>New Board Array Index</th>
<th>Resulting Square</th>
</tr>
</thead>
<tbody>
<tr>
<td>-21</td>
<td>43 - 21</td>
<td>22</td>
<td>QN1</td>
</tr>
<tr>
<td>-12</td>
<td>43 - 12</td>
<td>31</td>
<td>QN2</td>
</tr>
<tr>
<td>+9</td>
<td>43 + 9</td>
<td>51</td>
<td>QN4</td>
</tr>
<tr>
<td>+19</td>
<td>43 + 19</td>
<td>51</td>
<td>QN5</td>
</tr>
<tr>
<td>+21</td>
<td>43 + 21</td>
<td>64</td>
<td>K4</td>
</tr>
<tr>
<td>+12</td>
<td>43 + 12</td>
<td>65</td>
<td>K5</td>
</tr>
<tr>
<td>-21</td>
<td>43 - 21</td>
<td>24</td>
<td>Q1</td>
</tr>
<tr>
<td>-19</td>
<td>43 - 19</td>
<td>24</td>
<td>Q1</td>
</tr>
</tbody>
</table>

Figure 4: Generating all the possible Knight moves from the Queen Bishop 3 (QB3) square. The Knight is piece type 2 (see text) and has a DPOINT (direction table pointer) value of 8 and a DCOUNT (direction table counter) value of 8 also. So in generating the Knight's moves, DIRECT+8 will be the starting point in the direction table, and 8 values will be used: -21, -12, +08, +19, +21, +12, -08 and -19. The Knight starts at White's QB3 square, which is square 43 (see figure 2a, decimal representation). Thus the first possible Knight move is 43 - 21 = 22 (QN1), and so on.

Sample Move Generation

Suppose a Knight occupies the QB3 square. A Knight is piece type 2 and has a DPOINT of 8 and a DCOUNT of 8 (see table 1b). So in generating the Knight's moves, DIRECT + 8 will be the starting point in the direction table and 8 values will be used. Those values are -21, -12, +08, +19, +21, +12, -08, and -19. The Knight starts at White's QB3, which is square 43 (see figure 2a, decimal representation). Thus the first possible Knight move is 43 - 21 = 22. Now 22 is QN1, so the first Knight move returns the Knight to its starting square. Figure 4 summarizes all possible Knight moves from QB3.

Move Generation—
The Algorithms Explained

Move generation is controlled by GEN-MOV, which scans the board array and calls MPiece for each piece encountered.

PUT THIS IN YOUR PIPE
AND 'SMOKE' IT:
$150 less
for the best!

SMOKE SIGNAL BROADCASTING'S
M-16A STATIC MEMORY SYSTEM
IS NOW BEING OFFERED AT A NEW
LOW PRICE OF $379.00 — $150.00 LOWER
THAN EVER BEFORE!

FEATURES:

- SWTPC 6800 Plug Compatible
- STATIC — No refresh required
- Allows SWTPC 6800 expansion to 48K
- Low Power
- Uses Single +8 Volt Supply

The M-16A STATIC random access memory system, with a total storage capacity of 16834 words of 8 bits each, is switch selectable to any 4K starting address, and a hardware write protect switch is also included. The system's storage elements are 4K by 1 STATIC memory chips which store 4 times as much in only 12% more space than the low power 2102's. Typical access time is fast enough to work with a 6800 based computer operating at 2 MHz and all systems are factory tested at 2 MHz.

Send for FREE NEW
Computer Products Catalog
includes our complete line of 6800
Super Software products.

SMOKE SIGNAL BROADCASTING
6304 Yuca/Hollywood, CA 90028
(213) 462-5652

Circle 320 on inquiry card.
Then MPiece, the piece mover routine, generates all possible legal moves for that piece (moves that place the King in check are eliminated later in the program). The piece is brought in from memory. It is a one byte data value, as previously discussed, which contains piece type, flags, and color. The flags are deleted from the piece before checking for type. Basic piece types are indicated by values from 1 to 6. Except for pawns, White and Black pieces move alike. So a special case is needed for the Black pawn. If the given piece is a Black pawn, the piece type is decremented, making it type 0.

The type of the piece, now one from 0 to 6, is used as an index into the DCOUNT, direction table count, and DPOINT, direction table pointer arrays. The values for the given piece are fetched. The direction table pointer is then used as an index into DIRECT, the direction table, and the first move direction is fetched. The "from" position of the piece is the square on which the piece currently stands. This "from" board index and the direction table value are passed as parameters to the routine PATH.

PATH generates the move indicated and returns a flag which describes the status of the "to" position of the piece. Flag values are:
0 "to" position is empty.
1 "to" position contains a piece of the opposite color.
2 "to" position contains a piece of the same color.
3 "to" position is off the board.

PATH accomplishes its task by fetching the "from" position, adding the direction counter, and storing the result as the "to" position. It then uses the "to" position to form an index into the board array. The current contents of the square are fetched. If the square contains a hexadecimals FF, it is off the board. The off board flag is set and control is returned to MPiece.

If the square is on the board, the contents of the square are saved in memory location P2. The color and flag bits are then cleared and the remaining piece type is saved in T2. If the square is empty, control is returned to MPiece with the flag value still set to 0. Otherwise the color of the piece on the "to" square is compared with that of the moving piece. The appropriate flag is set to indicate whether or not the pieces are of the same color, and control is returned to MPiece.

Upon return from PATH, piece mover
checks to see if the square is occupied by a piece of the same color or is off the board. If so, this cannot be a legal move, so a check for further moves must follow a new direction. Otherwise a check is made to see if the square is empty. The answer is saved. A check is made to see if the piece being moved is a pawn. If so, control is passed to the special pawn logic. Otherwise the move generated must be added to the move list. ADMOVE is called for the job. After the move has been added to the move list, the answer to the empty square question is recovered. If the square is empty and the piece is a Bishop, Rook, or Queen, it is possible to continue moving in the same direction. In this case control passes back to the call to PATH for another move in that direction. Kings and Knights may make only one move in a given direction.

When the time comes to consider a new direction of movement for the piece, the index into the direction table is incremented. DCOUNT, the number of directions to con-

Listing 1, continued:

MP26: CALL ADMOVE
INX Y
DCR B
LXI H,P1
BIT 3,M
JRZ MP10
JMP MP15

*******MOVE OF 2 SQUARES*********
MP30: EXAF
JRNZ MP15
MP31: CALL ADMOVE
JMP MP15

*******DIAGONAL MOVE*************
MP35: EXAF
JRZ MP36
LDA M2
CPI 91
JRNC MP37
CPI 29
JRNC MP31
MP37: LXI H,P2
SET 5,M
JMPR MP31

*******DIAGONAL SQUARE EMPTY *****
MP36: CALL ENPSINT
JMP MP15

PATH ROUTINE
**********************************************************************
PATH: LXI H,M2
MOV A,M
ADD C
MOV M,A
LXI M2
MOV A,BOARD(X)
CPI OFFH
JRZ FA2
STA F2
ANI 7
STA T2
RZ
LDA P2
LXI H,P1
XRA M
BIT 7,A
JRZ FA1
MVI A,1
RET

************SAME COLOR************
MVI A,2
RET

************OFF BOARD************
MVI A,3
RET

Add this move to the move list.
Increment direction index for two square move direction.
Decrement the direction count.
Load the address where the piece was saved.
Check the flag in the piece which tells whether it has moved before.
If the pawn has never moved, go generate a second forward move. (The pawn can move two squares on the first move.) Otherwise go get new direction, skipping second forward move.

Restore the answer to the empty square question.
If the square is not empty, this is not a valid move. Go check the next direction.
Otherwise add this move to the move list.
Go check the next direction.

Restore the answer to the empty square question.
If the square is empty, it is not a normal pawn capture. Go try en passant.
Get the "to" position of the move.
If the board index is 91 or greater, this is the last rank and a White pawn promotion.
If so, go set promote flag.
Otherwise, if the board index is less than 29, this is the first rank and a Black pawn promotion.
If not, just add the move to the move list.
Load the address of the promotion flag.
Set the flag (bit 5 of P2), and go add the move to the move list.

Check for possible en passant capture and add it to the move list if legal.
Go check the next direction.

Get the address of the location where the "from" position was stored.
Get the "from" position from that memory location.
Add in the direction from the direction table, giving the "to" position.
Use "to" position to form an index into the board array.
Load the board index.
Get the contents of the board at the "to" square.
Is the "to" position off the board?
If so, go set off-board flag.
Save contents of the board at "to" square.
Isolate piece type.
Save piece type.
Return if the square is empty. The flag value is returned in the A register and it is already 0.
Get piece again.
Load the address of the moving piece.
Compare the pieces.
Check to see if the colors match. If so, after the exclusive OR the color bit will be 0.
If they match, go set match flag.
Otherwise, set different color flag and return.

Set same color flag and return.
Set off-board flag and return.
sider, is decremented. When DCOUNT reaches zero, all the moves for the piece have been generated. If the piece involved is the King, a call to castle will add any legal castling moves. Then control is returned to GENMOV.

All that remains is to discuss the special pawn logic. Pawns are peculiar in that they capture diagonally, but move straight ahead. They also have the option of moving one or two squares forward on their first move. Furthermore, if they reach the eighth and final rank, they may be promoted to another piece. Sargon always promotes its own pawns to Queens. A flag in variable P2 indicates pawn promotion.

The pawn logic in MPiece first checks to see if the direction of movement is along a diagonal. If so, the square must be occu-

Figure 5: Flowchart of the piece mover routine, MPiece.
ped by an enemy piece. It may also be possible to move to the eighth rank in capturing, so pawn promotion must be considered here as well. Another type of diagonal pawn move is the en passant capture. It must be considered by a call to ENPSNT. Finally it is time to consider a new direction, as is done for the other piece types.

If, however, the direction of movement is forward, the “to” square must be empty. Pawn promotion must be checked for on forward moves. If the piece has never moved before, another move in the same direction is a possibility. Otherwise it is time to consider a new direction. Figures 5 and 6 are flowcharts of MPiece and PATH, respectively.

The Other Move Generation Routines

The move generation driver is GENMOV, the generate move routine. The basic function of GENMOV is to generate the move set for all pieces of a given color. It scans the board checking for a piece of the same color and calls MPiece, the piece mover routine.

CASTLE and ENPSNT are also key routines in move generation. CASTLE checks the legality of both King side and Queen side castling. It adds them to the move list if legal. Basic checks must include:

- Has King moved?
- Is King in check?
- Has Rook moved?
- Are the intervening squares empty?
- Are any squares that the King passes through under attack?

ENPSNT checks for any en passant pawn captures and adds them to the move list if legal. The tests must include:

- On the fifth rank?
- Was previous move the first move for the enemy pawn?
- Is the enemy pawn on an adjacent file?

INCK, the check routine, performs the function of determining whether or not the King is in check. The basic method used is to scan outward from the King looking for attackers, by calling ATTACK.

The attack routine finds all attackers on a given square by scanning outward from the square until one of the following occurs:

- A piece is found that attacks this square.
- A piece is found that doesn’t attack this square.
- The edge of the board is reached.

---

**LSI-11 TIME**

It's TIME you brought your LSI-11 up to DATE TIME and DATE. Two important parameters in the computer world, are available to your LSI-11 on one DUAL SIZE BOARD. When requested, the TCU-50D will present you with the date (month and day), time (hour and minutes), and seconds. Turn your computer off and forget about the time — your battery supported TCU-50D won't, not for 3 months anyway. The correct date and time will be there when you power up.

The TCU-50D is shipped preset to your local time, but can be set to any time you want by a simple software routine.

**AT $295**

**YOU CAN’T AFFORD TO IGNORE TIME**

Time is only one way we can help you upgrade your LSI-11 or PDP-11 system. We’d also like to tell you about the others. So contact Digital Pathways if you’re into 11's. We are too.

---

Circle 383 on inquiry card.

Circle 89 on inquiry card.
In the case where this routine is called by CASTLE or INCHK, the routine is terminated as soon as an attacker of the opposite color is encountered.

ADMOVE adds a move to the move list. The move list is a linked list. Each move in the move list is stored in a 6 byte area. The meaning of each byte is as follows:

0 & 1 MLPTR — Move list pointer. A pointer to the next move in the move list. Used to facilitate sorting the list.

2 MLFRP — Move list from position. The board position from which the piece is moving.

3 MLTOP — Move list to position. The board position to which the piece is moving.

4 MLFLG — Move list flags.

5 MLVAL — Move list value. Contains the score assigned to the move in evaluation.

It is hoped that this introductory discussion will assist potential chess programmers in getting started. With the essentials of move generation out of the way, the fun part of evaluation can begin.

BIBLIOGRAPHY

In writing Sargon, it was our original intention to put together the first version without any research into the attempts made by others. In this respect Sargon is a unique creation. After competing in the Second West Coast Computer Faire, we began to investigate some of the literature. This bibliography presents some of the references we found most helpful, together with our evaluations.

1. Michie, Donald, On Machine Intelligence, Edinburgh University Press, 1974. Michie's book provides an excellent treatment of exchange evaluation. He uses the concept of an exchange polynomial for accurately determining the outcome of battles engaged on the board. The basic approach we used in XCHNG, the Sargon exchange evaluator, turned out to be surprisingly similar. Sargon's approach, however, is far less computationally complex. We highly recommend this reference to anyone planning to write a chess program without look-ahead.


3. Fine, Reuben, Ideas Behind the Chess Openings, David McKay, New York, 1943. Fine's book makes a great starting point for anyone contemplating the addition of an opening book. Although Fine does not present enough lines of play for a complete book, it does provide a good orientation to other references.
