Sargon I -Source code

Autor:
Data de publicació: 28-02-2022

SARGON

A COMPUTER
CHESS PROGRAM

DAN AND KATHE SPRACKLEN

Block diagram

HAYDEM BOOK COMPANY, INC.
Rochelle Park, New Jersey
ISBN 0-8104-51554
Library of Congress Catalog Card Number 78-64981

Copyright © 1978 by HAVDEN BOOK COMPANY, INC. All rights reserved.
No part of this book may be reprinted, or reproduced, or utilized in any
form or by any electronic, mechanical, or other means, now known or
hereafter invented, including photocopying and recording, or in any infor-
mation storage and retrieval system, without permission in writing from
the Publisher.

Printed in the United States of America

4 5 6 7 8 9 PRINTING
---
79 80 81 82 83 84 85 86 YEAR

CONTENTS

Meet Sargon
1

The Chessboard in Computer Graphics
3

Chess Piece Summary
6

Users Guide to Sargon
11

Notes on the Implementation of Sargon
17

Equates
19

Direction Tables
20

Point Value Array
20

Board Array
21

Attack List
22

Pinned Piece Array
22

Score and Ply Tables
23

Table Indices Section
24

Variables Section
25

Move List Section
27

Board Setup Routine
28

Path Routine
29

Piece Mover Routine
30

En Passant Routine
32

Adjust Move List Pointer
33

Castle Routine
34

Admove Routine
36

Generate Move Routine
37

Check Routine
38

Attack Routine
39

Attack Save Routine
42

Pin Check Routine
43

Pin Find Routine
44

Exchange Routine
47

Next Attacker/Defender Routine
48

Point Evaluation Routine
49

Limit Routine
51

Move Routine
54

Un-Move Routine
56

Sort Routine
58

Evaluation Routine
59

Find Move Routine
60

Ascend Tree Routine
64

One Move Book Opening
65

Graphics Data Base
66

Standard Message
67

Graphics Variables
68

I/0 Macro Definitions
68

Main Program Driver
69

Interrogation for Ply & Color
71

Computer Move Routine
73

Forced Mate Handling
75

Tab to Player's Column
76

Tab to Computer's Column
76

Tab to Player's Column w/o Move No
77

Tab to Computer's Column w/o Move No
77

Board Index to ASCII Square Name
78

Player's Move Analysis
79

ASCII Square Name to Board Index
80

Validate Move Subroutine
81

Accept Input Character
82

New Page if Needed
83

Display Mated King
84

Set Up Position for Analysis
85

Update Positions of Royalty
88

Set Up Empty Board
89

Insert Piece Subroutine
91

Board Index to Norm Address Subr
93

Positive Integer Division
94

Positive Integer Multiplication
94

Square Blinker
95

Execute Move Subroutine
97

Make Move Subroutine
98

TDL/ZILOG Mnemonics Conversion
99

Index to Subroutines
113

MEET SARGON

SARGON is a computer chess program by Dan and Kathe Spracklen. In March 1978 it took first place in the first chess tournament held strictly for microcomputers. The tournament took place during the 2-1/2 days of the 1978 West Coast Computer Faire and drew large crowds each day. When the last battle ended, SARGON had won 5 games of 5 played. A tie existed for second place, with 3 programs scoring a total of 3 points in the 5 rounds.
SARGON is written in Z-80 assembly language using the TDL Macro Assembler. The program occupies 8K of RAM, which includes 2K of data areas, 2K graphics display and user interface, and 4K move logic. The move logic is the heart of SARGON. It is displayed in the block diagram as the set of routines called by FNDMOV (Find Move). FNDMOV

controls the search for the computer's best move by performing a depth first-tree search using the techniques of alpha beta pruning. Listed first under FNDMOV's calls on the block diagram is PINFND (Pin Find Routine). PINFND produces a list of all pieces pinned against the king or queen for both white and black. Pinned pieces must be treated carefully when analyzing battles engaged on the chess board, since their attacking power may be an illusion. FNDMOV also calls POINTS (Point Evaluation Routine). POINTS performs a static evaluation and derives a score for a given board position. POINTS takes factors of material, board control, and development into account. Predominant in the evaluation is material. Material scores must be adjusted to reflect unresolved battles on the chess board. It is the function of XCHNG (Exchange Evaluation Routine) to judge the outcome of these unresolved battles. The factors of development and board control are not allowed to dominate the move choice. LIMIT is called to truncate the contribution of those factors to the score.

FNDMOV controls the generation of legal moves by GENMOV (Generate Move Routine). GENMOV produces the move set for all of the pieces of a given color. For each piece in turn, GENMOV calls MPIECE (Piece Mover Routine), which generates all the possible legal moves for a given piece. MPIECE itself calls a series of routines. PATH generates a single possible move for a given piece along its current path of motion. ADMOVE adds a move to the move list. CASTLE and ENPSNT (En Passant Pawn Capture Routine) handle the special moves. After MPIECE has produced all legal moves, GENMOV calls INCHK, which determines whether or not the king is in check.

Basic to the success of alpha beta pruning is the sorting of moves generated at each ply level. FNDMOV calls SORTM (Sort Routine) to accomplish this task. A sort is dependent on an evaluation, so SORTM calls EVAL (Evaluation Routine). To evaluate a given move on the move list, E VAL first makes the move on the board by calling MOVE. It is determined if the move is legal by calling INCHK. Then, if the move is legal, it is evaluated by calling PNFND and POINTS. Finally, EVAL restores the board position by calling UNMOVE.

The bookkeeping required by alpha beta pruning is for the most part coded in line in FNDMOV. However, FNDMOV calls ASCEND (Ascend Tree Routine) to adjust all the parameters in transferring the parameters up one ply in the tree.

At the bottom of FNDMOV's call list on the block diagram is BOOK. BOOK provides, an opening book of a single move. Ifwhite, SARGON will play P-K4 or P-Q4 at random. If black, SARGON replies to any opening move with P-K4 or P-Q4, whichever is most appropriate.

The move selection logic of FNDMOV is embedded in a whole network of routines that forms SARGON's interface to the outside world. The DRIVER routine initiates and coordinates, the entire game. First on the block diagram in DRIVER's list of calls is CHARTR (Accept Input Character). CHARTR is a totally machine-dependent input routine whose sole purpose is to accept a single character input from the keyboard. All machine-dependent aspects of SARGON have been isolated in this manner to simplify conversion to Z-80 machines running under different operating systems. Machine-dependent code appears in only two other places. The first is the macro definition area, where all the output functions are listed, and the second is in the routine DSPBRD (Display Graphics Board and Pieces), where machine-dependent lines of code are clearly marked.

Next on the block diagram is ANALYS (Set Up Position for Analysis). ANALYS allows the user to set the board to any position of his choosing. The routine blinks the graphics board squares in turn, allowing the user to input a piece ofhis choice or leave the contents unchanged. When the board has been set to the desired arrangement of pieces, play of the game may be resumed. ANALYS also provides a handy means of correcting a move entered by mistake.

As a part of game initialization, DRIVER calls INTERR (Interrogate for Ply and Color). INTERR questions the player for his choice of white or black, and allows him to select the depth of search. DSPBRD and INITBD complete initialization by setting up the graphics board display and internal board array. PGIFND (New Page if Needed) and TBCPCL (Tab to Computer's Column) are used to control spacing in the move list. The move list is displayed to the left of the graphics board on the video screen.

The most important routines called by DRIVER are, of course, CPTRMV and PLYRMV, which are control routines for the computer's and player's moves, respectively. Central to CPTRM V is FNDMOV, the logic to select the computer's move, which has already been discussed. Below FNDMOV on the block diagram is FCDMAT (Forced Mate Handling). If the computer is checkmated, it acknowledges the fact with a message displayed in the move list and by tipping over its king. Assuming the computer is not mated, MOVE makes the chosen move on the board array and EXECMV displays it on the graphics board. In displaying the move, the piece first blinks a few times, moves to its new location, and then blinks a few times again. The function of BITASN (Board Index to ASCII Square Name) is to convert the internal move into a representation in algebraic chess notation on the move list, then INCHK determines whether or not the computer should call "Check."

When the opponent is on the move, PLYRMV controls the events. It calls CHARTR to accept the move entry. ASNTBI (ASCII Square Name to Board Index) converts the move to internal representation. Then VALMOV checks the player's move for validity. If the move is legal, EXECMV displays it on the graphics board as in CPTRMV. PGIFND (New Page if Needed) aadTRPLCL (Tab to Player's Column) control spacing in the move list.

The Chess Board in Computer Graphics

A graphics display is an eye-catching addition to a chess program. For the human player, a visual display of the board is far easier to relate to than a scheme which creates an array using purely alphabetic characters. Graphics display requires specialized hardware, and degree of resolution varies with existing displays. The SARGON program features a complete graphite board display. The video screen of the Jupiter III microcomputer, on which it is implemented, has a 96 x 128 dot graphics matrix. The screen display is controlled by a 2k area of static RAM. Information may be displayed on the screen by storing the desired values in that 2k area. So only moue instructions are required for graphics display.

The SARGON display utilizes 96 x 96 dots for the graphics chess board. The remaining area is used to list the moves of the game in algebraic chess notation. The display is arranged as follows:
The empty board and move list area are displayed using the block move feature of the Z-80. It requires no stored data. The memory required to store the piece shapes has been kept to a minimum through use ofthe concept of boundary and kernel dots.

Graphics Characters

By varying and combining bits that are turned on, a total of 64 different graphics characters may be produced. For example:

Now, 1010 0101 = 165 in decimal, which can be used as the ASCII code for this character.

Pillar and Bar Formatlnq
We've seen how individual dots are grouped into blocks of six dots each. The blocks are then laid out like tiles to cover the display area. So a dot matrix that is 12 x 12 would look like:

To begin execution:
The start address of SARGON will vary depnding on the load address. It will always be the address of DRIVER. Once execution has begiun, SARGON will ask you a series of queations:

"Welcome to Chess. Care for a Game?"

To playa game of chess respond with "y." An answer of "n" will get you to the routine that allows you to set up a boats position. (See Item 5.)
"Would you like to play white (w) or black (b)?"
The player selects white by entering "w" or black by "b." Any other key defaults to black. White always moves first.
"Select look ahead (1-6),"
This allows the player to select the depth of search. For example, if you select 3 ply, SARGON will consider:

All of his possible moves.
All of your responses to those move,.
All of his possible replies to your responses.
At this point, the boats display will appear on the screen. If you choose white, SARGON will be waiting for your move entry. If you choose black, SARGON will make its move on the board, print it in the move list, and then ..it for your move entry.

To enter a move:
Moves must be entered in algebraic these notation. This means you most tell SARGON the file and rank coordinates of the squares you are moving from and to. The files are lettered a-h and the ranks are numbered 1-8. So the coordinates of the board are:

The move itself is entered as ff-tt so to play the king's pawn up two squares you would enter:

"e2-e4"
If SARGON responded with the same move, it would print:

"e7-e5"

Just enter the king's move. The rook will tag along. for example, if you are white and you wish to castle king's side, enter:

"e1-g1"
You will see both your king and rook move. When SARGON castles, he lists it as 0-0 or 0-0-0 as in normal chess notation.

To Capture En Passant
if you wish to capture one of SARGON's pawns using the en passant privilege, enter your pawn's move. After your pawn move is displayed, SARGON's pawn will blink and then vanish. When SARGON captures en peasant, his move is displayed on the graphics board in the same way. SARGON prints it in the move list as PxPep.

To play another game after checkmate:
If either your or SARGON is checkmated, and you wish to play againk just hit any key. The screen will blank out and SARGON will ask:

"Care for Another Came?"

Replies to this question are just like those to the original l"Care for a Game?"

To resign a hopless game or take back a move:
Ifyou decide your position is hopeless, or you wish to change a move entered in error, first wait until it is your turn to move. Then enter "control-R." You will immediately get the "Care for Another Game?" question. If you want to start over, type "y," but if you want to correct the board display, type "n." You will then get the routine that allows you to setup a board position.

To setup or correct a board position: If you typed "n". a "Care for a Game" question, SARGON will now ask:
"Would you like to analyse a position?"
If you answer "n" to this one, you will be out of SARGON entirely and back in the computer's monitor state. An answer of "y" will display the board just as you left it. The lower left-hand corner will blink. That, your signal that you can change the contents of that square, using one of the analysis commands.

Summary of Analysis Commands
<:CR>
A carriage return leaves the contents of the square unchanged and blinks the next square. If you are already at the upper right-hand corner, it wraps around to the lower left-hand corner and blink, that square.

Backspace
A backspace leaves the contents of the square unchanged and blinks.. square in the other direction. If, the opposite of a <:CR>, so you can go either direction.

0
An entry of 0, or of the space bar, or any key not listed in these commands will empty the square.

"Enter a Piece"
To enter a piece, type in piece-code, color-code, moved-code.

Piece-code is a letter indicating the desired piece (upper or lower case):

K -- KingQ -- QueenR -- RookB -- BishopN -- KnightP -- Pawn
Color-code is a letter indicating the side the piece belongs to (also upper or lower case):

w -- whiteb -- black
Moved-code is a number indicating whether the Piece has moved or not:

0 - Piece has never moved1 - piece has moved
Some examples:

To enter a black pawn on its original square type:

"P,b,0"
A white knight in the middle of the board would be:

"N,w,1"
A black king on its original square which has however moved:

"K,b,1"
Escape
The wscape key will terminate the blinking cycle. SARGON will ask:

"Is this right?"
An answer pf "n" wo;; gp bacl tp setting up the board. If you say "y" then SARGON will ask for information it needs to resume play from this point. The color chooice and search depth questions are the same as in Section 1. In addition SARGON must be given the answer to:

"Whose move is it?"

To terminate execution:
The way out of the SARGON program depend, on whether you're at the end of a game, in the middle of a game, or setting up a board position.

At the end of a game:

Depress any key.
SARGON responds with: "Care for Another Game?"
Answer with "n."
SARGON responds with: "Would you like to analyze a position?"
Answer with "n" and you're out.

In the middle of a game:

Wait until it's your turn.
Enter "Control-R"
SARGON responds with: "Care for Another Game?"
Follow 3-5 as for the end of a game.
Setting up a board:
Depress the escape key.
SARGON answers: "R this right?"
Respond with nswering four questions at once.
Follow 2-4 as for the middle of a game.

Notes on the Implementation of SARGON

SARGON was assembled using the TDL Macro Assembler, which does not use the ZILOG mnemonics. Conversion to ZILOG mnemonic, can be performed on instruction for instruction basis using the conversion chart included with this listing.

I/0 is based on the JOVE operation system which runs en the Wave-Mate Jupiter III computer. For ease in conversion all I/0 has been isolated to the following areas: Accept Input Character (p. 82), I/O Macro Definitions (p. 68), and Set Up Empty Board (p. 89).

SARGON must be loaded at a start address which is an even 256 byte page boundary (that is, at an address of the form XXOO hexadecimal).

Graphics routines assume a 96 by 128 dot matrix with black characters on a white background. To convert to a display with white character, on a black background, only six lines of code need be changed:

Location Is Change to
DB04 MVI M,80H MVI M,OBFH
DB08 MVI M,OBFH MVI M,80H
2 lines above IP04 JRZ IP04 JRNZ IP04
4 lines above IP18 JRNZ IPI8 JRZ IP18
2 lines above IP18 JRNZ IP2C JRZ IP2C
1 line below IP18 JRZ IP2C JRNZ IP2C

SARGON requires a minimum of 8K bytes of memory available for user programs.

Image of page 19 for reference

```
;*****************************************************************************
;           SARGON
;
; Sargon is a computer chess playing program designed and coded by Dan and
; Kathe Spracklen,
; Copyright 1978. All rights reserved. No part of this publication may be
; reproduced without the prior
; written permission.
;
;*****************************************************************************
```

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

;*****************************************************************************
; TABLES SECTION
;*****************************************************************************

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

;*****************************************************************************
; DIRECT -- Direction Table. Used to determine the direction of movement of
; each piece.
;*****************************************************************************

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
 .BYTE -10,-10,-11,-09
```

Image of page 20 for reference

```
; *******************************************************
; DPOINT -- Direction Table. Pointer. Used to determine.
; where to begin in the direction table for any
; given piece.
; *******************************************************
DPOINT = .-TBASE
.BYTE 20,16,8,0,4,0,0
; *******************************************************
; DCOUNT -- Direction Table Counter. Used to determine,
; the number of directions of movement for any
; given piece.
; *******************************************************
```

```
DCOUNT = -TBASE
.BYTE 4,4,8,4,4,8,8
; ******************************************************
; PVALUE -- Point Value. Gives the point value of each
; piece, or the worth of each piece.
; ******************************************************
PVALUE = .-TBASE-1
.BYTE 1,3,3,5,9,10
; ******************************************************
; PIECES -- The initial arrangement of the first rank of
; pieces on the board. Use to set up the board
; for the start of the game.
; ******************************************************
PIECES = .-TBASE
.BYTE 4,2,3,5,6,3,2,4
```

Image of page 21 for reference

```
; ******************************************************
; BOARD -- Board Array. Used to hold the current position
; of the board during play. The board itself
; looks like:
;  FFFFFFFFFFFFFFFFFFFF
;  FFFFFFFFFFFFFFFFFFFF
;  FF0402030506030204FF
;  FF0101010101010101FF
;  FF0000000000000000FF
;  FF0000000000000000FF
;  FF0000000000000060FF
;  FF0000000000000000FF
;  FF8181818181818181FF
;  FF8482838586838284FF
;  FFFFFFFFFFFFFFFFFFFF
;  FFFFFFFFFFFFFFFFFFFF
; The values of FF form the border of the
; board, and are used to indicate when a piece
; moves off the board. The individual bits of
; the other bytes in the board array 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
;  Bit 3 -- Piece has moved flag
;  Bits 2-0 Piece type
;   1 -- Pawn
;   2 -- Knight
;   3 -- Bishop
;   4 -- Rook
```

```
;   5 -- Queen
;   6 -- King
;   7 -- Not used
;   0 -- Empty Square
; *********************************************************

BOARD = .-TBASE
BOARDA: BLKB 120
```

Image of page 22 for reference

```
; *********************************************************
; ATKLIST -- Attack List. A two part array, the first
;  half for white and the second half for black.
;  It is used to hold the attackers of any given
;  square in the order of their value.
; WACT -- White Attack Count. This is the first
;  byte of the array and tells how many pieces are
;  in the white portion of the attack list.
; BACT -- Black Attack Count. This is the eighth byte of
;  the array and does the same for black.
; *********************************************************
WACT = ATKLST
BACT = ATKLST+7
ATKLST: WORD 0,0,0,0,0,0,0
; *********************************************************
; PLIST -- Pinned Piece Array. This is a two part array.
;  PLISTA contains the pinned piece position.
;  PLISTD contains the direction from the pinned
;  piece to the attacker.
; *********************************************************
PLIST = .-TBASE-1
PLISTD = PLIST+10
PLISTA: WORD 0,0,0,0,0,0,0,0,0,0
; *********************************************************
; POSK -- Position of Kings. A two byte area, the first
;  byte of which hold the position of the white
;  king and the second holding the position of
;  the black king.
; POSQ -- Position of Queens. Like POSK,but for queens.
; *********************************************************
POSK: BYTE 24,95
POSQ: BYTE 14,94
.BYTE -1
```

Image of page 23 for reference

```
; ******************************************************
; SCORE -- Score Array. Used during Alpha-Beta pruning to
;  hold the scores at each ply. It includes two
;  "dummy" entries for ply -1 and ply 0.
; ******************************************************
SCORE: WORD 0,0,0,0,0,0
; ******************************************************
; PLYIX -- Ply Table. Contains pairs of pointers, a pair
;  for each ply. The first pointer points to the
;  top of the list of possible moves at that ply.
;  The second pointer points to which move in the
;  list is the one currently being considered.
; ******************************************************
PLYIX: WORD 0,0,0,0,0,0,0,0,0,0
.WORD 0,0,0,0,0,0,0,0,0,0
STACK -- Contains the stack for the program.
.LOC START+2FFH
STACK:
```

Image of page 24 for reference

```
; ******************************************************
; TABLE INDICES SECTION
; Ml-M4 -- Working indices used to index into
;  the board array.
; Tl-T3 -- Working indices used to index into Direction
;  Count, Direction Value, and Piece Value tables.
; INDX1 -- General working indices. Used for various
; INDX2 purposes.
; NPINS -- Number of Pins. Count and pointer into the
;  pinned piece list.
; MLPTRI -- Pointer into the ply table which tells
;  which pair of pointers are in current use.
; MLPTRJ -- Pointer into the move list to the move that is
;  currently being processed.
:
; SCRIX -- Score Index. Pointer to the score table for
;  the ply being examined.
; BESTM -- Pointer into the move list for the move that
;  is currently considered the best by the
;  Alpha-Beta pruning process.
; MLLST -- Pointer to the previous move placed in the move
;  list. Used during generation of the move list.
; MLNXT -- Pointer to the next available space in the move
; ******************************************************
.LOC START+O
Ml: WORD TBASE
M2: WORD TBASE
M3: WORD TBASE
M4: WORD TBASE
Tl: WORD TBASE
T2: WORD TBASE
T3: WORD TBASE
INDX1: WORD TBASE
```

```
INDX2: WORD TBASE
NPINS: WORD TBASE
MLPTRI: WORD PLYIX
MLPTRJ: WORD 0
SCRIX: WORD 0
BESTM: WORD 0
MLLST: WORD 0
MLNXT: WORD MLLST
```

Image of page 25 for reference

```
; ******************************************************
; VARIABLES SECTION
; KOLOR -- Indicates computer's color. White is 0, and
;  Black is 80H.
; COLOR -- Indicates color of the side with the move.
; P1-P3 -- Working area to hold the contents of the board
;  array for a given square.
; PMATE -- The move number at which a checkmate is
;  discovered during look ahead.
; MOVENO -- Current move number.
; PLYMAX -- Maximum depth of search using Alpha-Beta
;  pruning.
; NPLY -- Current ply number during Alpha-Beta
;  pruning.
; CKFLG -- A non-zero value indicates the king is in check.
; MATEF -- A zero value indicates no legal moves.
; VALM -- The score of the current move being examined.
; BRDC -- A measure of mobility equal to the total number
;  of squares white can move to minus the number
;  black can move to.
; PTSL -- The maximum number of points which could be lost
;  through an exchange by the player not on the
;  move.
; PTSW1 -- The maximum number of points which could be won
;  through an exchange by the player not on the
;  move.
; PTSW2 -- The second highest number of points which could
;  be won through a different exchange by the player
;  not on the move.
; MTRL -- A measure of the difference in material
;  currently on the board. It is the total value of
;  the white pieces minus the total value of the
;  black pieces.
; BC0 -- The value of board control(BRDC) at ply 0.
```

Image of page 26 for reference

```
; MVO -- The value of material(MTRL) at ply 0.
; PTSCK -- A non-zero value indicates that the piece has
;  just moved itself into a losing exchange of
;  material.
; MOVES -- Our very tiny book of openings. Determines
;  the first move for the computer.
; ********************************************************
KOLOR: BYTE 0
COLOR: BYTE 0
P1: BYTE 0
P2: BYTE 0
P3: BYTE 0
PMATE: BYTE 0
MOVENO: BYTE 0
PLYMAX: BYTE 2
NPLY: BYTE 0
CKFLG: BYTE 0
MATEF: BYTE 0
VALM: BYTE 0
BRDC: BYTE 0
PTSL: BYTE 0
PTSWI: BYTE 0
PTSW2: BYTE 0
MTRL: BYTE 0
BCO: BYTE 0
MVO: BYTE 0
PTSCK: BYTE 0
MOVES: BYTE 35,55,10H
.BYTE 34,54,10H
.BYTE 85,65,10H
.BYTE 84,64,10H
```

Image of page 27 for reference

```
; ********************************************************
; MOVE LIST SECTION
; MUST -- A 2048 byte storage area for generated moves.
;  This area must be large enough to hold all
;   the moves for a single leg of the move tree.
; MLEND -- The address of the last available location
;  in the move list.
; MLPTR -- The Move List is a linked list of individual
;  moves each of which is 6 bytes in length. The
;  move list pointer(MLPTR) is the link field
;  within a move.
; MLFRP -- The field in the move entry which gives the
;  board position from which the piece is moving.
; MLTOP -- The field in the move entry which gives the
;  board position to which the piece is moving.
; MLFLG -- A field in the move entry which contains flag
;  information. The meaning of each bit is as
;  follows:
```

```
;   Bit 7 -- The color of any captured piece
;    0 -- White
;    1 -- Black
;   Bit 6 -- Double move flag (set for castling and
;    en passant pawn captures)
;   Bit 5 -- Pawn Promotion flag; set when pawn
;    promotes.
;   Bit 4 -- When set, this flag indicates that
;    this is the first move for the
;    piece on the move.
;   Bit 3 -- This flag is set is there is a piece
;    captured, and that piece has moved at
;    least once.
;   Bits 2-0 Describe the captured piece. A
;    zero value indicates no capture.
;   MLVAL -- The field in the move entry which contains the
;    score assigned to the move.
; *********************************************************
```

Image of page 28 for reference

```
.LOC START+300H
MLIST: BLKB 2048
MLEND = MLIST+2040
MLPTR = 0
MLFRP = 2
MLTOP = 3
MLFLG = 4
MLVAL = 5


; *********************************************************
PROGRAM CODE SECTION
BOARD SETUP ROUTINE
FUNCTION: To initialize the board array, setting the
pieces in their initial positions for the
start of the game.
CALLED BY: DRIVER
CALLS: None
ARGUMENTS: None
; *********************************************************
INITBD: MVI B,120 ; Pre-fill board with -1's
LXI H,BOARDA
MVI M,-1
INX H
DJNZ .-3
MVI B,8
LXI X,BOARDA
IB2: MOV A,-8(X) ; Fill non-border squares
MOV 21(X),A ; White pieces
SET 7,A ; Change to black
```

```
MOV 91(X),A ; Black pieces
MVI 31(X),PAWN ; White Pawns
MVI 81(X),BPAWN ; Black Pawns
MVI 41(X),0 ; Empty squares
MVI 51(X),0
MVI 61(X),0
MVI 71(X),0
INX X
DJNZ IB2
LXI X,POSK ; Init King/Queen position list
MVI 0(X),25
MVI 1(X),95
MVI 2(X),24
MVI 3(X),94
RET
```

Image of page 29 for reference

```
; *********************************************************
; PATH ROUTINE
; *********************************************************
; FUNCTION: To generate a single possible move for a given
;  piece along its current path of motion including:
;  Fetching the contents of the board at the new
;  position, and setting a flag describing the
;  contents:
;   0 -- New postion is empty
;   1 -- Encountered a piece of the
;    opposite color
;   2 -- Encountered a piece of the
;    same color
;   3 -- New position is off the
;    board
; CALLED BY: MPIECE
;  ATTACK
;  PINFND
; CALLS: None
; ARGUMENTS: Direction from the direction array giving the
;  constant to be added for the new position.
; *********************************************************
PATH: LXI H,M2 ; Get previous position
MOV A,M
ADD C ; Add diection constant
MOV M,A ; Save new position
LIXD M2 ; Load board index
MOV A,BOARD(X) ; Get contents of board
CPI -1 ; In boarder area ?
JRZ PA2 ; Yes - jump
STA P2 ; Save piece
ANI 7 ; Clear flags
```

```
STA T2 ; Save piece type
RZ ; Return if empty .
LDA P2 ; Get piece encountered
LXI H,P1 ; Get moving piece address
XRA M ; Compare
BIT 7,A ; Do colors match ?
JRZ PA1 ; Yes - jump
MVI A,1 ; Set different color flag
RET ; Return
PA1: MVI A,2 ; Set same color flag
RET ; Return
PA2: MVI A,3 ; Set off board flag
RET ; Return
```

Image of page 30 for reference

```
; ************************************************************************
;
; PIECE MOVER ROUTINE
;
; FUNCTION: To. generate all the possible legal moves for a given
; piece.
;
; CALLED BY: GENMOV
;
; CALLS: PATH
;  ADMOVE
;  CASTLE
;  ENPSNT
;
; ARGUMENTS: The piece to be moved.
;
; ************************************************************************
MPIECE: XRA M ; Piece to move
 ANI 87H ; Clear flag bit
 CPI BPAWN ; Is is a black Pawn ?
 JRNZ .+3 ; No-Skip
 DCR A ; Decrement for black Pawns
 ANI 7 ; Get piece type
 STA T1 ; Save piece type
 LIYD T1 ; Load index to DCOUNT/DPOINT
 MOV B,DCOUNT(Y) ; Get direction count
 MOV A,DPOINT(Y) ; Get direction pointer
 STA INDX2 ; Save as index to direct
 LIYD INDX2 ; Load index
MP5: MOV C,DIRECT(Y) ; Get move direction
 LDA Ml ; From position
 STA M2 ; Initialize to position
```

```
MP10: CALL PATH ; Calculate next position
 CPI 2 ; Ready for new direction ?
 JRNC MP15 ; Yes - Jump
 ANA A ; Test for empty square
 EXAF  ; Save result
 LDA T1 ; Get piece moved
 CPI PAWN+1 ; Is it a Pawn ?
 JRC MP20 ; Yes - Jump
 CALL ADMOVE ; Add move to list
 EXAF  ; Empty square ?
 JRNZ MP15 ; No - Jump
 LDA Tl ; Piece type
 CPI KING ; King ?
 JRZ MP15 : Yes - Jump
 CPI BISHOP ; Bishop, Rook, or Queen ?
 JRNC MP10 ; Yes - Jump
MP15: INX Y ; Increment direction index
 DJNZ MP5 ; Decr. count-jump if non-zerc
 LDA T1 ; Piece type
```

Image of page 31 for reference

```
 CPI KING ; King ?
 CZ CASTLE ; Yes - Try Castling
 RET ; Return
; ***** PAWN LOGIC *****
MP20: MOV A,B ; Counter for direction
 CPI 3 ; On diagonal moves ?
 JRC MP35 ; Yes - Jump
 JRZ MP30 ; -or-jump if on 2 square move
 EXAF ;  Is forward square empty?
 JRNZ MP15 ; No - jump
 LDA M2 ; Get "to" position
 CPI 91 ; Promote white Pawn ?
 JRNC MP25 ; Yes - Jump
 CPI 29 ; Promote black Pawn ?
 JRNC MP26 ; No - Jump
MP25: LXI H,P2 ; Flag address
 SET 5,M ; Set promote flag
MP26: CALL ADMOVE ; Add to move list
 INX Y ; Adjust to two square move
 DCR B
 LXI H,P1 ; Check Pawn moved flag
 BIT 3,M ; Has it moved before ?
 JRZ MP10 ; No - Jump
 JMP MP15 ; Jump
MP30: EXAF ;  Is forward square empty ?
 JRNZ MP15 ; No - Jump
MP31: CALL ADMOVE ; Add to move list
 JMP MP15 ; Jump
MP35: EXAF ;  Is diagonal square empty ?
 JRZ MP36 ; Yes - Jump
```

```
 LDA M2 ; Get "to" position
 CPI 91 ; Promote white Pawn ?
 JRNC MP37 ; Yes - Jump
 CPI 29 ; Black Pawn promotion ?
 JRNC MP31 ; No- Jump
MP37: LXI H,P2 ; Get flag address
 SET 5,M ; Set promote flag
 JMPR MP31 ; Jump
MP36: CALL ENPSNT ; Try en passant capture
 JMP MP15 ;  Jump
```

Image of page 32 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx**xxxxxxxxxxxxxxxxxxx
;  EN PASSANT ROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- To test for en passant Pawn capture and
;   to add it to the move list if it is
;   legal.
; CALLED BY: -- MPIECE
; CALLS: -- ADMOVE
;
; ADJPTR
;
; ARGUMENTS: --  None
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ENPSNT: LDA M1 ; Set position of Pawn
 LXI H,P1 ; Check color
 BIT 7,M ; Is it white ?
 JRZ .+4 ; Yes - skip
 ADI 10 ; Add 10 for black
 CPI 61 ; On en passant capture rank ?
 RNC ;  No - return
 CPI 69 ; On en passant capture rank ?
 RNC ;  No - return
 LIXD MLPTRJ ; Get pointer to previous move
 BIT 4,MLFLG(X) ; First move for that piece ?
 RZ ;  No - return
 MOV A,MLTOP(X) ; Get "to" postition
 STA M4 ; Store as index to board
 LIXD M4 ; Load board index
 MOV A,BOARD(X) ; Get piece moved
 STA P3 ; Save it
 ANI 7 ; Get piece type
 CPI PAWN ; Is it a Pawn ?
 RNZ ;  No - return
 LDA M4 ; Get "to" position
 LXI H,M2 ; Get present "to" position
 SUB M ; Find difference
 JP .+5 ; Positive ? Yes - Jump
 NEG ;  Else take absolute value
 CPI 10 ; Is difference 10 ?
 RNZ ;  No - return
```

```
 LXI H,P2 ; Address of flags
 SET 6,M ; Set double move flag
 CALL ADMOVE ; Add Pawn move to move list
 LDA M1 ; Save initial Pawn position
 STA M3
 LDA M4 ; Set "from" and "to" position8
   for dummy move
```

Image of page 33 for reference

```
STA M1
STA M2
LDA P3 ; Save captured Pawn
STA P2
CALL ADMOVE ; Add Pawn capture to move list
LDA M3 ; Restore "from" position
STA M1

; *********************************************************************
; ADJUST MOVE LIST POINTER FOR DOUBLE MOVE
;
; FUNCTION: -- To adjust move list pointer to link around
;   second move in double move.
;
; CALLED BY: -- ENPSNT
;  CASTLE
;  (This mini-routine is not really called,
;  but is jumped to to save time.)
;
; CALLS: --  None
;
; ARGUMENTS: --  None
;
; *********************************************************************
ADJPTR: LHLD MLLST ; Get list pointer
 LXI D,-6 ; Size of a move entry
 DAD D ; Back up list pointer
 SHLD MLLST ; Save list pointer
 MVI M,0 ; Zero out link, first byte
 INX H ; Next byte
 MVI M,0 ; Zero out link, second byte
 RET ;   Return
```

Image of page 34 for reference

```
; *********************************************************************
; CASTLE ROUTINE
;
; FUNCTION: -- To determine whether castling is legal
;   (Queen side, King side, or both) and add it
;   to the move list if it is.
;
; CALLED BY: -- MPIECE
;
; CALLS: -- ATTACK
;   ADMOVE
;   ADJPTR
;
; ARGUMENTS: --  None
;
; *********************************************************************
CASTLE: LDA P1 ; Get King
 BIT 3,A ; Has it moved ?
 RNZ ;  Yes - return
 LDA CKFLG ; Fetch Check Flag
 ANA A ; Is the King in check ?
 RNZ ;  Yes - Return
 LXI B,OFF03H ; Initialize King-side values
CA5: LDA M1 ; King position
 ADD C ; Rook position
 MOV C,A ; Save
 STA M3 ; Store as board index
 LIXD M3 ; Load board index
 MOV A,BOARD(X) ; Get contents of board
 ANI 7FH ; Clear color bit
 CPI ROOK ; Has Rook ever moved ?
 JRNZ CA20 ; Yes - Jump
 MOV A,C ; Restore Rook position
 JMPR CA15 ;  Jump
CA10: LIXD M3 ; Load board index
 MOV A,BOARD(X) ; Get contents of board
 ANA A ; Empty ?
 JRNZ CA20 ; No - Jump
 LDA M3 ; Current position
 CPI 22 ; Wh.ite Queen Knight square ?
 JRZ CA15 ; Yes - Jump
 CPI 92 ; Black Queen Knight square ?
 JRZ CA15 ; Yes - Jump
 CALL ATTACK ; Look for attack on square
 ANA A ; Any attackers ?
 JRNZ CA20 ; Yes - Jump
 LDA M3 ; Current position
CA15: ADD B ; Next position
 STA M3 ; Save as board index
 LXI H,M1 ; King position
 CMP M ; Reached King ?
```

Image of page 35 for reference

```
 JRNZ CA10 ; No - jump
 SUB B ; Determine King's position
 SUB B
 STA M2 ; Save it
 LXI H,P2 ; Address of flags
 MVI M,40H ; Set double move flag
 CALL ADMOVE ; Put king move in list
 LXI H,M1 ; Addr of King "from" position
 MOV A,M ; Get King's "from" position
 MOV M,C ; Store Rook "from" position
 SUB B ; Get Rook "to" position
 STA M2 ; Store Rook "to" position
 XRA A ; Zero
 STA P2 ; Zero move flags
 CALL ADMOVE ; Put Rook move in list
 CALL ADJPTR ; Re-adjust move list pointer
 LDA M3 ; Restore King position
 STA M1 ; Store
CA20: MOV AFB ; Scan Index
 CPI 1 ; Done ?
 RZ ;  Yes - return
 LXI B,01FCH ; Set Queen-side initial values
 JMP CA5 ;  Jump
```

Image of page 36 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  ADMOVE ROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- To add a move to the move list
;
; CALLED BY: -- MPIECE
;  ENPSNT
;  CASTLE
;
; CALLS: -- None
;
; ARGUMENT: --  None
; **x***x*x*xx*xxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ADMOVE: LDED MLNXT ; Addr of next loc in move list
 LXI H,MLEND ; Address of list end
 ANA A ; Clear carry flag
 DSBC D ; Calculate difference
 JRC AM10 ; Jump if out of space
 LHLD MLLST ; Addr of prev. list area
```

```
 SDED MLLST ; Savn next as previous
 MOV M,E ; Store link address
 INX H
 MOV M,D
 LXI H,Pl ; Address of moved piece
 BIT 3,M ; Has it moved before ?
 JRNZ .+7 ; Yes - jump
 LXI H,P2 ; Address of move flags
 SET 4,M ; Set first move flag
 XCHG  ; Address of move area
 MVI M,0 ; Store zero in link address
 INX H
 MVI M,0
 INX H
 LDA Ml ; Store "from" move position
 MOV M,A
 INX H
 LDA M2 ; Store "to" move position
 MOV M,A
 INX H
 LDA P2 ; Store move flags/capt. piece
 MOV M,A
 INX H
 MVI M,0 ; Store initial move value
 INX H
 SHLD MLNXT ; Save address for next move
 RET  ; Return
AM10: MVI M,0 ; Abort entry on table ovflow
 INX H
 MVI M,0
 DCX H
 RET
```

Image of page 37 for reference

```
; **xxxxxx*x**xxxxx*xxx*xxxx*xxx*xxxxxxxxxxxx*xxxxxxxxxxxxx*xx
;  GENERATE MOVE ROUTINE
; xxx*xx*x**xxxxxxx**xxxxxxxxx**x***x**xx*x*xx***x*x**x*xxxx*x
; FUNCTION: -- To generate the move set for all of the
;   pieces of a given color.
; CALLED BY: -- FNDMOV
; CALLS: -- MIECE
;   INCHK
; ARGUMENTS: -- None
; xxxxxxxxxxxxxxx*x*xxxx*xxxx*xxxx*xx*xxxxxxx*xxxxxxxxxx***x*x
GENMOV: CALL INCHK ; Test for King in check
 STA CKFLG ; Save attack count as flag
 LDED MLNXT ; Addr of next avail list space
 LHLD MLPTRI ; Ply list pointer index
 INX H ; Increment to next ply
 INX H
```

```
 MOV M,E ; Save move list pointer
 INX H
 MOV M,D
 INX H
 SHLD MLPTRI ; Save new index
 SHLD MLLST ; Last pointer for chain init.
 MVI A,21 ; First position on board
GM5: STA M1 ; Save as index
 LIXD Ml ; Load board index
 MOV A,BOARD(X) ; Fetch board contents
 ANA A ; Is it empty ?
 JRZ GM10 ; Yes - Jump
 CPI -1 ; Is it a boarder square ?
 JRZ GM10 ; Yes - Jump
 STA P1 ; Save piece
 LXI H,COLOR ; Address of color of piece
 XRA M ; Test color of piece
 BIT 7,A ; Match ?
 CZ MPIECE ; Yes - call Move Piece
GM10: LDA M1 ; Fetch current board position
 INR A ; Incr to next board position
 CPI 99 ; End of board array ?
 JNZ GM5 ; No - Jump
 RET ;   Return
```

Image of page 38 for reference

```
; xxxxxxxxxxxxxxxxxxxxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  CHECK ROUTINE
; xxxx*xxxxxxxxxxxxxxx-x*x*xxxxxxxxxxxxxx*xxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- To determine whether or not the
;   King is in check.
; CALLED BY: -- GENMOV
;   FNDMOV
;   EVAL
; CALLS: -- ATTACK
;
; ARGUMENTS: -- Color of King
; x****xxx*xs**xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
INCHK: LDA COLOR ; Get color
INCHK1: LXI H,POSK ; Addr of white King position
 ANA A ; White ?
 JRZ .+3 ; Yes - Skip
 INX H ; Addr of black King position
 MOV A,M ; Fetch King position
 STA M3 ; Save
 LIXD M3 ; Load board index
 MOV A,BOARD(X) ; Fetch board contents
 STA P1 ; Save
 ANI 7 ; Get piece type
 STA T1 ; Save
 CALL ATTACK ; Look for attackers on King
 RET ;   Return
```

Image of page 39 for reference

```
; ************************************************************
; ATTACK ROUTINE
; ************************************************************

; FUNCTION: -- To find all attackers on a given square
;   by scanning outward from the square
;   until a piece is found that attacks
;   that,square, or a piece is found that
;   doesn't attack that square, or the edge
;   of the board is reached.
;
;  In determining which pieces attack a square,
; this routine also takes into account the ability of
; certain pieces to attack through another attacking
; piece. (For example a queen lined up behind a bishop
; of her same color along a diagonal.) The bishop is
; then said to be transparent to the queen, since both
; participate in the attack.
;
;  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.
;
; CALLED BY: -- POINTS
;  PINFND
;  CASTLE
;  INCHK
;
; CALLS: -- PATH
;   ATKSAV
;
; ARGUMENTS: --  None
; ************************************************************
ATTACK: PUSH B ; Save Register B
 XRA A ; Clear
 MVI B,16 ; Initial direction count
 STA INDX2 ; Initial direction index
 LIYD INDX2 ; Load index
AT5: MOV C,DIRECT(Y) ; Get direction
 MVI D,0 ; Init. scan count/flags
 LDA M3 ; Init. board start position
 STA M2 ;  Save
AT10: INR D ; Increment scan count
 CALL PATH ; Next position
 CPI 1 ; Piece of a opposite color ?
 JRZ AT14A ; Yes - jump
 CPI 2 ; Piece of same color ?
 JRZ AT14B ; Yes - jump
```

Image of page 40 for reference

```
 ANA A ; Empty position ?
```

```
 JRNZ AT12 ; No - jump
 MOV A,B ; Fetch direction count
 CPI 9 ; On knight scan ?
 JRNC AT10 ; No - jump
AT12: INX Y ; Increment direction index
 DJNZ AT5 ; Done ? No - jump
 XRA A ; No attackers
AT13: POP B ; Restore register B
 RET  ; Return
AT14A: BIT 6,D ; Same color found already ?
 JRNZ AT12 ; Yes - jump
 SET 5,D ; Set opposite color found flag
 JMP AT14 ; Jump
AT14B: BIT 5,D ; Opposite color found already?
 JRNZ AT12 ; Yes - jump
 SET 6,D ; Set same color found flag

; ***** DETERMINE IF PIECE ENCOUNTERED ATTACKS SQUARE *****
AT14: LDA T2 ; Fetch piece type encountered
 MOV E,A ; Save
 MOV A,B ; Get direction-counter
 CPI 9 ; Look for Knights ?
 JRC AT25 ; Yes - jump
 MOV A,E ; Get piece type
 CPI QUEEN ; Is is a Queen ?
 JRNZ AT15 ; No - Jump
 SET 7,D ; Set Queen found flag
 JMPR AT30 ; Jump
AT15: MOV A,D ; Get flag/scan count
 ANI OFH ; Isolate count
 CPI 1 ; On first position ?
 JRNZ AT16 ; No - jump
 MOV A,E ; Get encountered piece type
 CPI KING ; Is it a King ?
 JRZ AT30 ; Yes - jump
AT16: MOV A,B ; Get direction counter
 CPI 13 ; Scanning files or ranks ?
 JRC AT21 ; Yes - jump
 MOV A,E ; Get piece type
 CPI BISHOP ; Is it a Bishop ?
 JRZ AT30 ; Yes - jump
 MOV A,D ; Get flags/scan count
 ANI OFH ; Isolate count
 CPI 1 ; On first position ?
 JRNZ AT12 ; No - jump
 CMP E ; Is it a Pawn ?
 JRNZ AT12 ; No - jump
 LDA P2 ; Fetch piece including color
```

Image of page 41 for reference

```
 BIT 7,A ; Is it white ?
 JRZ AT20 ; Yes - jump
 MOV A,B ; Get direction counter
 CPI 15 ; On a non-attacking diagonal ?
 JRC AT12 ; Yes - jump
 JMPR AT30 ; Jump
AT20: MOV A,B ; Get direction counter
```

```
 CPI 15 ; On a non-attacking diagonal ?
 JRNC AT12 ; Yes - jump
 JMPR AT30 ; Jump
AT21: MOV A,E ; Get piece type
 CPI ROOK ; Is is a Rook ?
 JRNZ AT12 ; No - jump
 JMPR AT30 ; Jump
AT25: MOV A,E ; Get piece type
 CPI KNIGHT ; Is it a Knight ?
 JRNZ AT12 ; No - jump
AT30: LDA T1 ; Attacked piece type/flag
 CPI 7 ; Call from POINTS ?
 JRZ AT31 ; Yes - jump
 BIT 5,D ; Is attacker opposite color ?
 JRZ AT32 ; No - jump
 MVI A,1 ; Set attacker found flag
 JMP AT13 ; Jump
AT31: CALL ATKSAV ; Save attacker in attack list
AT32: LDA T2 ; Attacking piece type
 CPI KING ; Is it a King,?
 JZ AT12 ; Yes - jump
 CPI KNIGHT ; Is it a Knight ?
 JZ AT12 ; Yes - jump
 JMP AT10 ;  Jump
```

Image of page 42 for reference

```
; *************************************************************
; ATTACK SAVE ROUTINE
; *************************************************************
; FUNCTION: -- To save an attacking piece value in the attack
; list, and to increment the attack count for that
; color piece.
;
; The pin piece list is checked for the attacking
; piece, and if found there, the piece is not
; included in the attack list.
;
; CALLED BY: -- ATTACK
;
; CALLS: --  PNCK
;
; ARGUMENTS: --  None
; *************************************************************
ATKSAV: PUSH B ; Save Regs BC
 PUSH D ; Save Regs DE
 LDA NPINS ; Number of pinned pieces
 ANA A ; Any ?
 CNZ PNCK ; yes - check pin list
 LIXD T2 ; Init index to value table
 LXI H,ATKLST ; Init address of attack list
 LXI B,0 ; Init increment for white
 LDA P2 ; Attacking piece
 BIT 7,A ; Is it white ?
 JRZ .+4 ; Yes - jump
```

```
 MVI C,7 ; Init increment for black
 ANI 7 ; Attacking piece type
 MOV E,A ; Init increment for type
 BIT 7,D ; Queen found this scan ?
 JRZ .+4 ; No - jump
 MVI E,QUEEN ; Use Queen slot in attack list
 DAD B ; Attack list address
 INR M ; Increment list count
 MVI D,0
 DAD D ; Attack list slot address
 MOV A,M ; Get data already there
 ANI OFH ; Is first slot empty ?
 JRZ AS20 ; Yes - jump
 MOV A,M ; Get data again
 ANI OFOH ; Is second slot empty ?
 JRZ AS19 ; Yes - jump
 INX H ; Increment to King slot
 JMPR AS20 ; Jump
AS19: RLD ;  Temp save lower in upper
 MOV A,PVALUE(X) ; Get new value for attack list
 RRD ;  Put in 2nd attack list slot
 JMPR AS25 ; Jump
AS20: MOV A,PVALUE(X) ; Get new value for attack list
 RLD ;  Put in 1st attack list slot
```

Image of page 43 for reference

```
AS25: POP D ; Restore DE regs
POP B ; Restore BC regs
RET ; Return

; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  PIN CHECK ROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- Checks to see if the attacker is in the
;   pinned piece list. If so he is not a valid
;   attacker unless the direction in which he
;   attacks in the same as the direction along
;   which he is pinned. If the piece is
;   found to be invalid as an attacker, the
;   return to the calling routine is aborted
;   and this routine returns directly to ATTACK.
;
; CALLED BY: -- ATKSAV
;
; CALLS: --  None
;
; ARGUMENTS: -- The direction of the attack. The
; pinned piece counnt.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*
PNCK: MOV D,C ; Save attack direction
 MVI E,0 ; Clear flag
 MOV C,A ; Load pin count for search
 MVI B,0
 LDA M2 ; Position of piece
 LXI H,PLISTA ; Pin list address
PCl: CCIR  ; Search list for position
```

```
 RNZ  ; Return if not found
 EXAF  ; Save search paramenters
 BIT 0,E ; Is this the first find ?
 JRNZ PC5 ; No - jump
 SET 0,E ; Set first find flag
 PUSH H ; Get corresp index to dir list
 POP X
 MOV A,9(X) ; Get direction
 CMP D ; Same as attacking direction ?
 JRZ PC3 ; Yes - jump
 NEG  ; Opposite direction ?
 CMP D ; Same as attacking direction ?
 JRNZ PC5 ; No - jump
PC3: EXAF  ; Restore search parameters
 JPE PC1 ; Jump if search not complete
 RET  ; Return
PC5: POP PSW ; Abnormal exit
 POP D ; Restore regs.
 POP B
 RET  ; Return to ATTACK
```

Image of page 44 for reference

```
; .*x**xxxxtcxx*xxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  PIN FIND ROUTINE
; .xxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*
; FUNCTION: -- To produce a list of all pieces pinned
;   against the King or Queen, for both white
;   and black.
; CALLED BY: -- FNDMOV
;   EVAL
; CALLS: -- PATH
;   ATTACK
; ARGUMENTS: -- None
; .xxxxxxxxx***x*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
PINFND: XRA A ; Zero pin count
 STA NPINS
 LXI D,POSK ; Addr of King/Queen pos list
PF1: LDAX D ; Get position of royal piece
 ANA A ; Is it on board ?
 JZ PF26 ; No- jump
 CPI -1 ; At end of list ?
 RZ  ; Yes return
 STA M3 ; Save position as board index
 LIXD M3 ; Load index to board
 MOV A,BOARD(X) ; Get contents of board
 STA P1 ; Save
 MVI B,8 ; Init scan direction count
 XRA A
 STA INDX2 ; Init direction index
 LIYD INDX2
PF2: LDA M3 ; Get King/Queen position
 STA M2 ; Save
 X RA A
 STA M4 ; Clear pinned piece saved pos
 MOV C,DIRECT.(Y) ; Get direction of scan
PF5: CALL PATH ; Compute next position
 ANA A ; Is it empty ?
```

```
 JRZ PF5 ; Yes - jump
 CPI 3 ; Off board ?
 JZ PF25 ; Yes - jump
 CPI 2 ; Piece of same color found ?
 LDA M4 ; Load pinned piece position
 JRZ PF15 ; Yes - jump
 ANA A ; Possible pin ?
 JZ PF25 ; No - jump
 LDA T2 ; Piece type encountered
 CPI QUEEN ; Queen ?
 JZ PF19 ; Yes - jump
 MOV L,A ; Save piece type
```

Image of page 45 for reference

```
 MOV A,B ; Direction counter
 CPI 5 ; Non-diagonal direction ?
 JRC PF10 ; Yes - jump
 MOV A,L ; Piece type
 CPI BISHOP ; Bishop ?
 JNZ PF25 ; No - jump
 JMP PF20 ; Jump
PF10: MOV A,L ; Piece type
 CPI ROOK ; Rook ?
 JNZ PF25 ; No - jump
 JMP PF20 ; Jump
PF15: ANA A ; Possible pin ?
 JNZ PF25 ; No - jump
 LDA M2 ; Save possible pin position
 STA M4
 JMP PF5 ; Jump
PF19: LDA P1 ; Load King or Queen
 ANI 7 ; Clear flags
 CPI QUEEN ; Queen ?
 JRNZ PF20 ; No - jump
 PUSH B ; Save regs.
 PUSH D
 PUSH Y
 XRA A ; Zero out attack list
 MVI B,14
 LXI H,ATKLST
 MOV M,A
 INX H
 DJNZ .-2
 MVI A,7 ; Set attack flag
 STA T1
 CALL ATTACK ; Find attackers/defenders
 LXI H,WACT ; White queen attackers
 LXI D,BACT ; Black queen attackers
 LDA P1 ; Get queen
 BIT 7,A ; Is she white ?
 JRZ .+3 ; Yes - skip
 XCHG  ; Reverse for black
 MOV A,M ; Number of defenders
 XCHG  ; Reverse for attackers
 SUB M ; Defenders minus attackers
 DCR A ; Less 1
 POP Y ; Restore regs.
 POP D
```

```
 POP B
 JP PF25 ; Jump if pin not valid
PF20: LXI H,NPINS ; Address of pinned piece count
 INR M ; Increment
 LIXD NPINS ; Load pin list index
 MOV PLISTD(X),C ; Save direction of pin
```

Image of page 46 for reference

```
 LDA M4 ; Position of pinned piece
 MOV PLIST(X),A ; Save in list
PF25: INX Y ; Increment direction index
 DJNZ PF27 ; Done ? No - Jump
PF26: INX D ; Incr King/Queen pos index
 JMP PF1 ; Jump
PF27: JMP PF2 ;  Jump
```

Image of page 47 for reference

```
; *************************************************************
; EXCHANGE ROUTINE
; *************************************************************
; FUNCTION: -- To determine the exchange value of a
;   piece on a given square by examining all
;   attackers and defenders of that piece.
;
; CALLED BY: -- POINTS
;
; CALLS: --  NEXTAD
;
; ARGUMENTS: --  None.
;
; *************************************************************
XCHNG:  EXX  ; Swap regs.
  LDA P1 ; Piece attacked
  LXI H,WACT ; Addr of white attkrs/dfndrs
  LXI D,BACT ; Addr of black attkrs/dfndrs
 .BIT 7,A ; Is piece white ?
  JRZ .+3 ; Yes - jump
  XCHG  ; Swap list pointers
  MOV B,M ; Init list counts
  XCHG
  MOV C,M
  XCHG
  EXX  ; Restore regs.
  MVI C,0 ; Init attacker/defender flag
  MVI E,0 ; Init points lost count
  LIXD T3 ; Load piece value index
  MOV D,PVALUE(X) ; Get attacked piece value
  SLAR D ; Double it
  MOV B,D ; Save
  CALL NEXTAD ; Retrieve first attacker
  RZ  ; Return if none
XC10:  MOV L,A ; Save attacker value
```

```
  CALL NEXTAD ; Get next defender
  JRZ XC18 ; Jump if none
  EXAF  ; Save defender value
  MOV A,B ; Get attacked value
  CMP L ; Attacked less than attacker ?
  JRNC XC19 ; No - jump
  EXAF  ; -Restore defender
XC15:  CMP L ; Defender less than attacker ?
  RC  ; Yes - return
  CALL NEXTAD ; Retrieve next attacker value
  RZ  ; Return if none
  MOV L,A ; Save attacker value
  CALL NEXTAD ; Retrieve next defender value
  JRNZ XC15 ; Jump if none
XC18:  EXAF  ; Save Defender
  MOV A,B ; Get value of attacked piece
```

Image of page 48 for reference

```
XC19: BIT 0,C ; Attacker or defender ?
 JRZ .+4 ; Jump if defender
 NEG ;  Negate value for attacker
 ADD E ; Total points lost
 MOV E,A ; Save total
 EXAF ;  Restore previous defender
 RZ ;  Return if none
 MOV B,L ; Prev attckr becomes defender
 JMP XC10 ;  Jump

; *************************************************************
; NEXT ATTACKER/DEFENDER ROUTINE
; *************************************************************
; FUNCTION: -- To retrieve the next attacker or defender
;   piece value from the attack list, and delete
;   that piece from the list.
;
; CALLED BY: -- XCHNG
;
; CALLS: --  None
;
; ARGUMENTS: -- Attack list addresses.
;  Side flag
;  Attack list counts
;
; *************************************************************
NEXTAD: INR C ; Increment side flag
 EXX  ; Swap registers
 MOV A,B ; Swap list counts
 MOV B,C
 MOV C,A
 XCHG  ; Swap list pointers
 XRA A
 CMP B ; At end of list ?
 JRZ NX6 ; Yes - jump
 DCR B ; Decrement list count
 INX H ; Increment list inter
 CMP M ; Check next item in list
```

```
 JRZ .-2 ; Jump if empty
 RRD  ; Get value from list
 ADD A ; Double it
 DCX H ; Decrement list pointer
NX6: EXX ; Restore regs.
 RET ; Return
```

Image of page 49 for reference

```
; ************************************************************
;  POINT EVALUATION ROUTINE
; ************************************************************
; FUNCTION: -- To perform a static board evaluation and
;     derive a score for a given board position
;  CALLED BY:  --  FNDMOV
;      EVAL
;  CALLS:  --  ATTACK
;      XCHNG
;      LIMIT
;  ARGUMENTS:  --   None
; ************************************************************
POINTS: XRA A ; Zero out variables
 STA MTRL
 STA BRDC
 STA PTSL
 STA PTSWI
 STA PTSW2
 STA PTSCK
 LXI H,T1 ; Set attacker flag
 MVI M,7
 MVI A,21 ; Init to first square on board
PT5: STA M3 ; Save as board index
 LIXD M3 ; Load board index
 MOV A,BOARD(X) ; Get piece from board
 CPI -1 ,Off board edge ?
 JZ PT25 ; Yes - jump
 LXI H,P1 ; Save piece, if any
 MOV M,A
 ANI 7 ; Save piece type, if any
 STA T3
 CPI KNIGHT ; Less than a Knight (Pawn) ?
 JRC PT6X ; Yes - Jump
 CPI ROOK ; Rook, Queen or King ?
 JRC PT6.B ; No - jump
 CPI KING ; Is it a King ?
 JRZ PT6AA ; Yes - jump
 LDA MOVENO ; Get move number
 CPI 7 ; Less than 7 ?
 JRC PT6A ; Yes - Jump
 JMP PT6X ; Jump
PT6AA: BIT 4,M ; Castled yet ?
 JRZ PT6A ; No - jump
 MVI A,+6 ; Bonus for castling
 BIT 7,M ; Check piece color
 JRZ PT6D ; Jump if white
 MVI A,-6 ; Bonus for black castling
```

Image of page 50 for reference

```
 JMP PT6D ; Jump
PT6A: BIT 3,M ; Has piece moved yet ?
 JRZ PT6X ; No - jump
 JMP PT6C ; Jump
PT6B: BIT 3,M ; Has piece moved yet ?
 JRNZ PT6X ; Yes - jump
PT6C: MVI A,-2 ; Two point penalty for white
 BIT 7,M ; Check piece color
 JRZ .+4 ; Jump if white
 MVI A,+2 ; Two point penalty for black
PT6D: LXI H,BRDC ; Get address of board control
 ADD M ; Add on penalty/bonus points
 MOV M,A ; Save
PT6X: XRA A ; Zero out attack list
 MVI B,14
 LXI H,ATKLST
 MOV M,A
 INX  H
 DJNZ .-2
 CALL ATTACK ; Build attack list for square
 LXI H,BACT ; Get black attacker count addr
 LDA WACT ; Get white attacker count
 SUB M ; Compute count difference
 LXI H,BRDC ; Address of board control
 ADD M ; Accum board control score
 MOV M,A ; Save
 LDA P1 ; Get piece on current square
 ANA A ; Is it empty ?
 JZ PT25 ; Yes - jump
 CALL XCHNG ; Evaluate exchange, if any
 XRA A ; Check for a loss
 CMP E ; Points lost ?
 JRZ PT23 ; No - Jump
 DCR D ; Deduct half a Pawn value
 LDA P1 ; Get piece under attack
 LXI H,COLOR ; Color of side just moved
 XRA M ; Compare with piece
 BIT 7,A ; Do colors match ?
 MOV A,E ; Points lost
 JRNZ PT20 ; Jump if no match
 LXI H,PTSL ; Previous max points lost
 CMP M ; Compare to current value
 JRC PT23 ; Jump if greater than
 MOV M,E ; Store new value as max lost
 LIXD MLPTRJ ; Load pointer to this move
 LDA M3 ; Get position of lost piece
 CMP MLTOP(X) ; Is it the one moving ?
 JRNZ PT23 ; No - jump
 STA PTSCK ; Save position as a flag
 JMP PT23 ; Jump
```

Image of page 51 for reference

```
PT20: LXI H,PTSW1 ; Previous maximum points won
 CMP M ; Compare to current value
 JRC .+4 ; Jump if greater than
 MOV A,M ; Load previous max value
 MOV M,E ; Store new value as max won
 LXI H,PTSW2 ; Previous 2nd max points won
 CMP M ; Compare to current value
 JRC PT23 ; Jump if greater than
 MOV M,A ; Store as new 2nd max lost
PT23: LXI H,Pl ; Get piece
 BIT 7,M ; Test color
 MOV A,D ; Value of piece
 JRZ .+4 ; Jump if white
 NEG ;  Negate for black
 LXI H,MTRL ; Get addrs of material total
 ADD M ; Add new value
 MOV M,A ; Store
PT25: LDA M3 ; Get current board position
 INR A ; Increment
 CPI 99 ; At end of board ?
 JNZ PT5 ; No - jump
 LDA PTSCK ; Moving piece lost flag
 ANA A ; Was it lost ?
 JRZ PT25A ; No - jump
 LDA PTSW2 ; 2nd max points won
 STA PTSW1 ; Store as max points won
 XRA A ; Zero out 2nd max points won
 STA PTSW2
PT25A: LDA PTSL ; Get max points lost
 ANA A ; Is it zero ?
 JRZ .+3 ; Yes - jump
 DCR A ; Decrement it
 MOV B,A ; Save it
 LDA PTSW1 ; Max,points won
 ANA A ; Is it zero ?
 JRZ .+11. ; Yes - jump
 LDA PTSW2 ; 2nd max points won
 ANA A ; Is it zero ?
 JRZ .+5 ; Yes - jump
 DCR A ; Decrement it
 SRLR A ; Divide it by 2
 SUB B ; Subtract points lost
 LXT 9,COLOR ; Color of side just moved
 BIT 7,M ; Is it white ?
 JRZ .+4 ; Yes - jump
 NEG ;  Negate for black
 LXI H,MTRL ; Net material on board
 ADD M ; Add exchange adjustments
 LXI H,MVO ; Material at ply 0
```

Image of page 52 for reference

```
 SUB M ; Subtract from current
 MOV B,A ; Save
 MVI A,30 ; Load material limit
 CALL LIMIT ; Limit to plus or minus value
```

```
 MOV E,A ; Save limited value
 LDA BRDC ; Get board control points
 LXI H,BCO ; Board control at ply zero
 SUB M ; Get difference
 MOV B,A ; Save
 LDA PTSCK ; Moving piece lost flag
 ANA A ; Is it zero ?
 JRZ .+4 ; Yes - jump
 MVI B,0 ; Zero board control points
 MVI A,6 ; Load board control limit
 CALL LIMIT ; Limit to plus or minus value
 MOV D,A ; Save limited value
 MOV A,E ; Get material points
 ADD A ; Multiply by 4
 ADD  A
 ADD D ; Add board control
 LXI H,COLOR ; Color of side just moved
 BIT 7,M ; Is it white ?
 JRNZ .+4 ; No - jump
 NEG ;  Negate for white
 ADI 80H ; Rescale score (neutral = 80H
 STA VALM ; Save score
 LIXD MLPTRJ ; Load move list pointer
 MOV MLVAL(X),A ; Save score in move list
 RET ;   Return
```

Image of page 53 for reference

```
; xxx*xxxxx*x*xxxxxxxxxxxxxxxx**xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  LIMIT ROUTINE
; ~*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  FUNCTION: -- To limit the magnitude of a given value
;    to another given value.
;  CALLED BY: -- POINTS
;  CALLS: -- None
;  ARGUMENTS: -- Input - Value, to be limited in the B
;      register.
;.     - Value to limit to in the A register
;    Output - Limited value in the A register.
; **xx*******xxxxx**xxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*xxxxxxx
LIMIT: BIT 7,B ; Is value negative ?
 JZ LIM10 ; No - jump
 NEG  ; Make positive
 CMP B ; Compare to limit
 RNC  ; Return if outside limit
 MOV A,B ; Output value as is
 RET  ; Return
LIM10: CMP B ; Compare to limit
 RC  ; Return if outside limit
 MOV A,B ; Output value as is
 RET
 Return
.END
```

Image of page 54 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 MOVE ROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- To execute a move from the move list on the
;   board array.
; CALLED BY: -- CPTRMV
;   PLYRMV
;   EVAL
;   FNDMOV
;   VALMOV
; CALLS: -- None
;
; ARGUMENTS: --  None
; xxxxxxxxxxx*xxxxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MOVE: LHLD MLPTRJ ; Load move list pointer
 INX H ; Increment past link bytes
 INX H
MV1: MOV A,M ; "From" position
 STA M1 ; Save
 INX H ; Increment pointer
 MOV A,M ; "To" position
 STA M2 ; Save
 INX H ; Increment pointer
 MOV D,M ; Get captured piece/flags
 LIXD M1 ; Load "from" pos board index
 MOV E,BOARD(X) ; Get piece moved
 BIT 5,D ; Test Pawn promotion flag
 JRNZ MV15 ; Jump if set
 MOV A,E ; Piece moved
 ANI 7 ; Clear flag bits
 CPI QUEEN ; Is it a queen ?
 JRZ MV20 ; Yes - jump
 CPI KING ; Is it a king ?
 JRZ MV30 ; Yes - jump
MV5: LIYD M2 ; Load "to" pos board index
 SET 3,E ; Set piece moved flag
 MOV BOARD(Y),E ; Insert piece at new position
 MVI BOARD(X),0 ; Empty previous position
 BIT 6,D ; Double move ?
 JRNZ MV40 ; Yes - jump
 MOV A,D ; Get captured piece, if any
 ANI 7
 CPI QUEEN ; Was it a queen ?
 RNZ  ; No - return
 LXI H,POSQ ; Addr of saved Queen position
 BIT 7,D ; Is Queen white ?
 JRZ MV10 ; Yes - jump
 INX H ; Increment to black Queen pos
```

Image of page 55 for reference

```
MV10: XRA A ; Set saved position to zero
 MOV M,A
 RET  ; Return
```

```
MV15: SET 2,E ; Change Pawn to a Queen
 JMP MV5 ; Jump
MV20: LXI H,POSQ ; Addr of saved Queen position
MV21: BIT 7,E ; Is Queen white ?
 JRZ MV22 ; Yes - jump
MV22: LDA M2 ; Get rnewnQueenbpositionen pos
 MOV M,A ; Save
 JMP MV5 ; Jump
MV30: LXI H,POSK ; Get saved King position
 BIT 6,D ; Castling ?
 JRZ MV21 ; No - jump
 SET 4,E ; Set King castled flag
 JMP MV21 ; Jump
MV40: LHLD MLPTRJ ; Get move list pointer
 LXI D,8 ; Increment to next move
 DAD D
 JMP MV1 ; Jump (2nd part of dbl move)
```

Image of page 56 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx'xxxxx
; UN-MOVE ROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- To reverse the process of the move routine,
;   thereby restoring the board array to its
;   previous position.
; CALLED BY: -- VALMOV
;   EVAL
;   FNDMOV
;   ASCEND
; CALLS: -- None
;
; ARGUMENTS: --  None
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
UNMOVE: LHLD MLPTRJ ; Load move list pointer
 INX H ; Increment past link bytes
 INX H
UM1: MOV A,M ; Get "from" position
 STA Ml ; Save
 INX H ; Increment pointer
 MOV A,M ; Get "to" position
 STA M2 ; Save
 INX H ; Increment pointer
 MOV D,M ; Get captured piece/flags
 LIXD M2 ; Load "to" pos board index
 MOV E,BOARD(X) ; Get piece moved
 BIT 5,D ; Was it a Pawn promotion ?
 JRNZ UM15 ; Yes - jump
 MOV A,E ; Get piece moved
 ANI 7 ; Clear flag bits
 CPI QUEEN ; Was it a Queen ?
 JRZ UM20 ; Yes - jump
 CPI KING ; Was it a King ?
 JRZ UM30 ; Yes - jump
UM5: BIT 4,D ; Is this 1st move for piece ?
 JRNZ UM16 ; Yes - jump
UM6: LIYD M1 ; Load "from" pos board index
```

```
 MOV BOARD(Y),E ; Return to previous board pos
 MOV A,D ; Get captured piece, if any
 ANI 8FH ; Clear flags
 MOV BOARD(X),A ; Return to board
 BIT 6,D ; Was it a double move ?
 JRNZ UM40 ; Yes - jump
 MOV A,D ; Get captured piece, if any
 ANI 7 ; Clear flag bits
 CPI QUEEN ; Was it a Queen ?
 RNZ ;  No - return
```

Image of page 57 for reference

```
 LXI H,POSQ ; Address of saved Queen pos
 BIT 7,D ; Is Queen white ?
 JRZ UM10 ; Yes - jump
 INX H ; Increment to black Queen pos
UM10: LDA M2 ; Queen's previous position
 MOV M,A ; Save
 RET  ; Return
UM15: RES 2,E ; Restore Queen to Pawn
 JMP UM5 ; Jump
UM16: RES 3,E ; Clear piece moved flag
 JMP UM6 ; Jump
UM20: LXI H,POSQ ; Addr of saved Queen position
UM21: BIT 7,E ; Is Queen white ?
 JRZ UM22 ; Yes - jump
 INX H ; Increment to black Queen pos
UM22: LDA M1 ; Get previous position
 MOV M,A ; Save
 JMP UM5 ; Jump
UM30: LXI H,POSK ; Address of saved King pos
 BIT 6,D ; Was it a castle ?
 JRZ UM21 ; No - jump
 RES 4,E ;. Clear castled flag
 JMP UM21 ; Jump
UM40: LHLD MLPTRJ ; Load move list pointer
 LXI D,8 ; Increment to next move
 DAD D
 JMP UM1 ; Jump (2nd part of dbl move)
```

Image of page 58 for reference

```
; *********************************************************
; SORT ROUTINE
; *********************************************************
; FUNCTION: -- To sort the move list in order of
;   increasing move value scores.
;
; CALLED BY: -- FNDMOV
;
; CALLS: --  EVAL
```

```
;
; ARGUMENTS: --  None
; ******************************************************
SORTM: LBCD MLPTRI ; Move list begin pointer
 LXI D,0 ; Initialize working pointers
SR5: MOV H,B
 MOV L,C
 MOV C,M ; Link to next move
 INX H
 MOV B,M
 MOV M,D ; Store to link in list
 DCX H
 MOV M,E
 XRA A ; End of list ?
 CMP B
 RZ  ; Yes - return
SR10: SBCD MLPTRJ ; Save list pointer
 CALL EVAL ; Evaluate move
 LHLD MLPTRI ; Begining of move list
 LBCD MLPTRJ ; Restore list pointer
SR15: MOV E,M ; Next move for compare
 INX H
 MOV D,M
 XRA A ; At end of list ?
 CMP D
 JRZ SR25 ; Yes - jump
 PUSH D ; Transfer move pointer
 POP X
 LDA VALM ; Get new move value
 CMP MLVAL(X) ; Less than list value ?
 JRNC SR30 ; No - jump
SR25: MOV, M,B ; Link new move into list
 DCX H
 MOV M,C
 JMP SR5 ; Jump
SR30: XCHG  ; Swap pointers
 JMP SR15 ; Jump
```

Image of page 59 for reference

```
; xxxx*xxx***xxxxx**xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; EVALUATION ROUTINE
; *x*rt*xxxx*xxx*xx*xxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- To evaluate a given move in the move list.
;   It first makes the move on the board, then ii
;   the move is legal, it evaluates it, and then
;   restores the boaard position.
; CALLED BY: -- SORT
; CALLS: -- MOVE
;   INCHK
;   PINFND
;   POINTS
;   UNMOV
; ARGUMENTS: -- None
; **x**x*xxx**xxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
EVAL: CALL MOVE ; Make move on the board array
```

```
 CALL INCHK ; Determine if move is legal
 ANA A ; Legal move ?
 JRZ EV5 ; Yes - jump
 XRA A ; Score of zero
 STA VALM ; For illegal move
 JMP EV10 ; Jump
EV5: CALL PINFND ; Compile pinned list
 CALL POINTS ; Assign points to move
EV10: CALL UNMOVE ; Restore board array
 RET ;   Return
```

Image of page 60 for reference

```
; *********************************************************************
; FIND MOVE ROUTINE
; *********************************************************************
;
; FUNCTION: -- To determine the computer's best move by
;   performing a depth first tree search using
;   the techniques of alpha-beta pruning.
;
; CALLED BY: -- CPTRMV
;
; CALLS: -- PINFND
;   POINTS
;   GENMOV
;   SORTM
;   ASCEND
;   UNMOV
;
; ARGUMENTS: --  None
;
; *********************************************************************
FNDMOV: LDA MOVENO ; Currnet move number
 CPI 1 ; First move ?
 CZ BOOK ; Yes - execute book opening
 XRA A ; Initialize ply number to zer
 STA NPLY
 LXI H,0 ; Initialize best move to zero
 SHLD BESTM
 LXI H,MLIST ; Initialize ply list pointers
 SHLD MLNXT
 LXI H,PLYIX-2
 SHLD MLPTRI
 LDA KOLOR ; Initialize color
 STA COLOR
 LXI H,SCORE ; Initialize score index
 SHLD SCRIX
 LDA PLYMAX ; Get max ply number
 ADI 2 ; Add 2
 MOV B,A ; Save as counter
 XRA A ; Zero out score table
 MOV M,A
 INX H
 DJNZ .-2
 STA BC0 ; Zero ply 0 board control
 STA MV0 ; Zero ply 0 material
 CALL PINFND ; Complie pin list
 CALL POINTS ; Evaluate board at ply 0
```

```
 LDA BRDC ; Get board control points
 STA BC0 ; Save
 LDA MTRL ; Get material count
 STA MV0 ; Save
FM5: LXI H,NPLY ; Address of ply counter
 INR M ; Increment ply count
```

Image of page 61 for reference

```
 XRA A ; Initialize mate flag
 STA MATEF
 CALL GENMOV ; Generate list of moves
 LDA NPLY ; Current ply counter
 LXI H,PLYMAX ; Address of maximum ply number
 CMP M ; At max ply ?
 CC SORTM ; No - call sort
 LHLD MLPTRI ; Load ply index pointer
 SHLD MLPTRJ ; Save as last move pointer
FM15: LHLD MLPTRJ ; Load last move pointer
 MOV E,M ; Get next move pointer
 INX H
 MOV D,M
 MOV A,D
 ANA A ; End of move list ?
 JRZ FM25 ; Yes - jump
 SDED MLPTRJ ; Save current move pointer
 LHLD MLPTRI ; Save in ply pointer list
 MOV M,E
 INX H
 MOV M,D
 LDA NPLY ; Current ply counter
 LXI H,PLYMAX ; Maximum ply number ?
 CMP M ; Compare
 JRC FM18 ; Jump if not max
 CALL MOVE ; Execute move on board array
 CALL INCHK ; Check for legal move
 ANA A ; Is move legal
 JRZ .+8 ; Yes - jump
 CALL UNMOVE ; Restore board position
 JMP FM15 ; Jump
 LDA NPLY ; Get ply counter
 LXI H,PLYMAX ; Max ply number
 CMP M ; Beyond max ply ?
 JRNZ FM35 ; Yes - jump
 LDA COLOR ; Get current color
 XRI 80H ; Get opposite color
 CALL INCHK1 ; Determine if King is in check
 ANA A ; In check ?
 JRZ FM35 ; No - jump
 JMP FM19 ; Jump (One more ply for check)
FM18: LIXD MLPTRJ ; Load move pointer
 MOV A,MLVAL(X) ; Get move score
 ANA A ; Is it zero (illegal move) ?
 JRZ FM15 ; Yes - jump
 CALL MOVE ; Execute move on board array
```

```
FM19: LXI H,COLOR ; Toggle color
 MVI A,80H
 X RA M
 MOV M,A ; Save new color
```

Image of page 62 for reference

```
 BIT 7,A ; Is it white ?
 RNZ .+6 ; No - jump
 XI H,MOVENO ; Increment move number
 NR M
 HLD SCRIX ; Load score table pointer
 OV A,M ; Get score two plys above
 NX H ; Increment to current ply
 NX H
 OV M,A ; Save score as initial value
 CX H ; Decrement pointer
 HLD SCRIX ; Save it
 MP FM5 ; Jump
FM25: LDA MATEF ; Get mate flag
 ANA A ; Checkmate or stalemate ?
 JRNZ FM30 ; No - jump
 LDA CKFLG ; Get check flag
 ANA A ; Was King in check ?
 MVI A,80H ; Pre-set stalemate score
 JRZ FM36 ; No - jump (stalemate)
 LDA MOVENO ; Get move number
 STA MATE ; Save
 MVI A,OFFH ; Pre-set checkmate score
 JMP FM36 ; Jump
FM30: LDA NPLY ; Get ply counter
 CPI 1 ; At top of tree ?
 RZ  ; Yes - return
 CALL ASCEND ; Ascend one ply in tree
 LHLD SCRIX ; Load score table pointer
 INX H ; Increment to current ply
 INX H
 MOV A,M ; Get score
 DCX H ; Restore pointer
 DCX H
 JMP FM37 ; Jump
FM35: CALL PINFND ; Compile pin list
 CALL POINTS ; Evaluate move
 CALL UNMOVE ; Restore board position
 LDA VALM ; Get value of move
FM36: LXI H,MATEF ; Set mate flag
 SET 0,M
 LHLD SCRIX ; Load score table pointer
FM37: CMP M ; Compare to score 2 ply above
 JRC FM40 ; Jump if less
 JRZ FM40 ; Jump if equal
 NEG  ; Negate score
 INX H ; Incr score table pointer
 CMP M ; Compare to score 1 ply above
 JC FM15 ; Jump if less than
```

```
 JZ FM15 ; Jump if equal
```

Image of page 63 for reference

```
 MOv M,A ; Save as new score 1 ply above
 LDA NPLY ; Get current ply counter
 CPI 1 ; At top of tree ?
 JNZ FM15 ; No - jump
 LHLD MLPTRJ ; Load current move pointer
 SHLD BESTM ; Save as best move.pointer
 LDA SCORE+1 ; Get best move score
 CPI OFFH ; Was it a checkmate ?
 JNZ FM15 ; No - jump
 LXI H,PLYMAX ; Get maximum ply number
 DCR M ; Subtract 2
 DCR M
 LDA KOLOR ; Get computer's color
 BIT 7,A ; Is it white ?
 RZ ;  Yes - return
 LXI H,PMATE ; Checkmate move number
 DCR M ; Decrement
 RET ;  Return
FM40: CALL ASCEND ; Ascend one ply in tree
 JMP FM15 ;  Jump
```

Image of page 64 for reference

```
; *********************************************************************
; ASCEND TREE ROUTINE
; *********************************************************************
;
; FUNCTION: -- To adjust all necessary parameters to
;   ascend one ply in the tree.
;
; CALLED BY: -- FNDMOV
;
; CALLS: --  UNMOV
;
; ARGUMENTS: --  None
; *********************************************************************

ASCEND: LXI H,COLOR ; Toggle color
 MVI A,80H
 XRA M
 MOV M,A ; Save new color
 BIT 7,A ; Is it white ?
 JRZ .+6 ; Yes - jump
 LXI H,MOVENO ; Decrement move number
 DCR M
 LHLD SCRIX ; Load score table index
```

```
 DCX H ; Decrement
 SHLD SCRIX ; Save
 LXI H,NPLY ; Decrement ply counter
 DCR M
 LHLD MLPTRI ; Load ply list pointer
 DCX H ; Load pointer to move list to
 MOV D,M
 DCX H
 MOV E,M
 SDED MLNXT ; Update move list avail ptr
 DCX H ; Get ptr to next move to undo
 MOV D,M
 DCX H
 MOV E,M
 SHLD MLPTRI ; Save new ply list pointer
 SDED MLPTRJ ; Save next move pointer
 CALL UNMOVE ; Restore board to previous pl
 RET  ; Return
```

Image of page 65 for reference

```
; ********************************************************************
; ONE MOVE BOOK OPENING
; ****************************************x***************
; FUNCTION: -- To provide an opening book of a single
;   move.
;
; CALLED BY: -- FNDMOV
;
; CALLS: --  None
;
; ARGUMENTS: --  None
; ********************************************************************

BOOK: POP PSW ; Abort return to FNDMOV
 LXI H,SCORE+1 ; Zero out score
 MVI M,0 ; Zero out score table
 LXI H,BMOVES-2 ; Init best move ptr to book
 SHLD BESTM
 LXI H,BESTM ; Initialize address of pointer
 LDA KOLOR ; Get computer's color
 ANA A ; Is it white ?
 JRNZ BM5 ; No - jump
 LDAR  ; Load refresh reg (random no)
 BIT 0,A ; Test random bit
 RZ  ; Return if zero (P-K4)
 INR M ; P-Q4
 INR M
 INR M
 RET  ; Return
BM5: INR M ; Increment to black moves
 INR M
 INR M
 INR M
 INR M
 INR M
```

```
 LIXD MLPTRJ ; Pointer to opponents 1st move
 MOV A,MLFRP(X) ; Get "from" position
 CPI 22 ; Is it a Queen Knight move ?
 JRZ BM9 ; Yes - Jump
 CPI 27 ;.Is it a King Knight move ?
 JRZ BM9 ; Yes - jump
 CPI 34 ; Is it a Queen Pawn ?
 JRZ BM9 ; Yes - jump
 RC  ; If Queen side Pawn opening -
   ; return (P-K4)
 CPI 35 ; Is it a King Pawn ?
 RZ  ; Yes - return (P-K4)
BM9: INR M ; (P-Q4)
 INR M
 INR M
 RET  ; Return to CPTRMV

; **x*********x*****x**x*xx*xx*xxxxxxxxxxxxxxxxxx*xxxxxxxxxxx
```

Image of page 66 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx*xxxxxxxx
;  GRAPHICS DATA BASE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; DESCRIPTION: The Graphics Data Base contains the
;  necessary stored data to propuce the piece
;  on the board. Only the center 4 x 4 blocks are
;  stored and only for a Black Piece on a White
;  square. A White piece on a black square is
;  produced by complementing each block, and a
;  piece on its own color square is produced
;  by moving in a kernel of 6 blocks.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
 .LOC START+384
BLBASE = START+512
BLOCK = .-BLBASE
 .RADIX 16
 .BYTE 80,80,80,80 ; Black Pawn on White square
 .BYTE 80,0A0,90,80
 .BYTE 80,0AF,9F,80
 .BYTE 80,83,83,80
 .BYTE 80,0B0,0B0,80 ; Black Knight on White square
 .BYTE OBE,0BF,0BF,95
 .BYTE OA0,0BE,0BF,85
 .BYTE 83,83,83,81
 .BYTE 80,0A0,00,80 ; Black Bishop on White square
 .BYTE OA8,0BF,0BD,80
 .BYTE 82,0AF,87,80
 .BYTE 82,83,83,80
 .BYTE 80,80,80,80 ; Black Rook on White square
 .BYTE 8A,0BE,0BD,85
 .BYTE 80,0BF,0BF,80
 .BYTE 82,83,83,81
 .BYTE 90,80,80,90 ; Black Queen on White square
 .BYTE OBF,0B4,0BE,95
```

```
 .BYTE 8B,0BF,9F,81
 .BYTE 83,83,83,81
 .BYTE 80,088,90,80 ; Black King on White square
 .BYTE OBC,0BA,0B8,94
 .BYTE OAF,0BF,0BF,85
 .BYTE 83,83,83,81
 .BYTE 90,0B0,0B0,80 ; Toppled Black King
 .BYTE OBF,0BF,0B7,80
 .BYTE 9F,0BF,0BD,80
 .BYTE 80,80,88,9D
KERNEL = .-BLBASE
 .BYTE OBF,9F,0AF,0BF,9A,0A5 ; Pawn Kernel
 .BYTE 89,0AF,0BF,9F,0B9,9F ; Knight Kernel
 .BYTE 97,0BE,96,0BD,9B,0B9 ; Bishop Kernel
 .BYTE 0B5,0A1,92,0BF,0AA,95 ; Rook Kernel
 .BYTE OA8,9B,0B9,0B6,0AF,0A7 ; Queen Kernel
 .BYTE OA3,85,0A7,9A,0BF,9F ; King Kernel
 .BYTE 0A8,0BF,89,0A2,8F,86 ; Toppled King Kernel
 .RADIX 10
```

Image of page 67 for reference

```
; ******************************************************************
;  STANDARD MESSAGES
;  ********************tr*********tt*********trtt*trtt,etttt**tr**xtr,ttt
 .LOC START+1800H
GRTTNG: .ASCII "WELCOME TO CHESS! CARE FOR A GAME?"
ANAMSG: .ASCII "WOULD YOU LIKE TO ANALYZE A POSITION?"
CLRMSG: .ASCII "DO YOU WANT TO PLAY WHITE(w) OR BLACK(b)?"
TITLEl: .ASCII "SARGON"
TITLE2: .ASCII "PLAYER"
SPACE:  .ASCII " ; For output of blank area
MVENUM: .ASCII "01 "
TITLES: .ASCII " "
 .ASCII [^H83] ; Part of TITLE 3 - Underlines
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII " "
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII [^H83]
 .ASCII " "
MVEMSG: .ASCII "al-al"
0.0: .ASCII "0-0 "
0.0.0: .ASCII "0-0-0"
CKMSG: .ASCII "CHECK"
MTMSG: .ASCII "MATE IN "
MTPL: .ASCII "2"
PCs: .ASCII "KQRBNP" ; Valid piece characters
UWIN: .ASCII "YOU WIN"
```

```
IWIN: .ASCII "I WIN"
AGAIN: .ASCII "CARE FOR ANOTHER GAME?"
CRTNES: .ASCII "IS THIS RIGHT?"
PLYDEP: .ASCII "SELECT LOOK AHEAD (1-6)"
TITLE4: -.ASCII " "
WSMOVE: .ASCII "WHOSE MOVE IS IT?"
BLANKR: .ASCII ["HlC]  ; Control-
P.PEP: .ASCII "PxPep"
INVALI: .ASCII "INVALID MOVE"
INVAL2: .ASCII "TRY AGAIN"
```

Image of page 68 for reference

```
; ********************************************************************
; VARIABLES
; ********************************************************************
BRDPOS: .BLKB 1 ; Index into the board array
ANBDPS: .BLKB 1 ; Additional index required for ANALYS
INDXER: .WORD BLBASE ; Index into graphics data base
NORMAD: .BLKW 1 ; The address of the upper left hand
   corner of the square on the board
LINECT: .BYTE 0 ; Current line number

; ********************************************************************
; MACRO DEFINITIONS
; ********************************************************************
; All input/output to SARGON is handled in the form of macro calls to
; simplify conversion to alternate systems. All of the input/output macros
; conform to the Jove monitor of the Jupiter III computer.
; ********************************************************************
;*** OUTPUT  ***
 .DEFINE CARRET=
 [RST 7
 .BYTE 92H,lAH
 .WORD  0]
;*** CLEAR SCREEN ***
 .DEFINE CLRSCR=
 [RST 7
 .BYTE OB2H,lAH
 .WORD  BLANKR,1]
;*** PRINT ANY LINE (NAME, LENGTH) ***
 .DEFINE PRTLIN[NAME,LNGTH)=
 [RST 7
 .BYTE 0B2H,lAH
 .WORD  NAME,LNGTH]
;*** PRINT ANY BLOCK (NAME, LENGTH) ***
 .DEFINE PRTBLK[NAME,LNGTH]=
 [RST 7
 .BYTE OB3H,lAH
 .WORD  NAME,LNGTH]
;*** EXIT TO MONITOR ***
 .DEFINE EXIT=
 [RST 7
 .BYTE  01FH)
```

Image of page 69 for reference

```
; ****************************************************************
; MAIN PROGRAM DRIVER
; ************************************************************
; FUNCTION: -- To coordinate the game moves.
;
; CALLED BY: -- None
;
; CALLS: -- INTERR
;   INITBD
;   DSPBRD
;   CPTRMV
;   PLYRMV
;   TBCPCL
;   PGIFND
; MACRO CALLS:  CLRSCR
;   CARRET
;   PRTLIN
;   PRTBLK
; ARGUMENTS:  None
; ************************************************************
 .LOC START+lA00H ; Above the move logic
DRIVER: LXI SP,STACK ; Set stack pointer
 CLRSCR ;  Blank out screen
 PRTLIN GRTTNG,34 ; Output greeting
DRIV01: CALL CHARTR ; Accept answer
 CARRET  ; New line
 CPI 59H ; Is it a 'Y' ?
 JNZ ANALYS ; Yes - jump
 SUB A ; Code of White is zero
 STA COLOR ; White always moves first
 CALL INTERR ; Players color/search depth
 CALL INITBD ; Initialize board array
 MVI A,1 ; Move number is 1 at at start
 STA MOVENO ; Save
 STA LINECT ; Line number is one at start
 LXI H,MVENUM ; Address of ascii move number
 MVI M,30H ; Init to '01 '
 INX H
 MVI M,31H
 INX H
 MVI M,20H
 CALL DSPBRD ; Set up graphics board
 PRTLIN TITLE4,15 ; Put up player headings
 PRTLIN TITLE3,15
DRIV04: PRTBLK MVENUM,3 ; Display move number
 LDA KOLOR ; Bring in computer's color
 ANA A ; Is it white ?
 JRNZ DR08 ; No - jump
```

Image of page 70 for reference

```
 CALL PGIFND ; New page if needed
 CPI 1 ; Was page turned ?
 CZ TBCPCL ; Yes - Tab to computers column
 CALL CPTRMV ; Make and write computers move
 PRTBLK SPACE,1 ; Output a space
 CALL PLYRMV ; Accept and make players move
 CARRET ;  New line
 JMPR DROC ; Jump
DR08: CALL PLYRMV ; Accept and make players move
 PRTBLK SPACE,1 ; Output a space
 CALL PGIFND ; New page if needed
 CPI 1 ; Was page turned ?
 CZ TBCPCL ; Yes - Tab to computers column
 CALL CPTRMV ; Make and write computers move
 CARRET ;  New line
DROC: LXI H,MVENUM+2 ; Addr of 3rd char of move
 MVI A,20H ; Ascii space
 CMP M ; Is char a space ?
 MVI A,3AH ; Set up test value
 JRZ DR10 ; Yes - jump
 INR M ; Increment value
 CMP M ; Over Ascii 9 ?
 JRNZ DR14 ; No - jump
 MVI M,30H ; Set char to zero
DR10: DCX H ; 2nd char of Ascii move no.
 INR M ; Increment value
 CMP M ; Over Ascii 9 ?
 JRNZ DR14 ; No - jump
 MVI M,30H ; Set char to zero
 DCX H ; 1st char of Ascii move no.
 INR M ; Increment value
 CMP M ; Over Ascii 9 ?
 JRNZ DR14 ; No - jump
 MVI M,31H ; Make 1st char a one
 MVI A,30H ; Make 3rd char a zero
 STA MVENUM+2
DR14: LXI H,MOVENO ; Hexadecimal move number
 INR M ; Increment
 JMP DRIV04 ;  Jump
```

Image of page 71 for reference

```
; *************************************************************
; INTERROGATION FOR PLY & COLOR
; *************************************************************
; FUNCTION: -- To query the player for his choice of ply
;   depth and color.
;
; CALLED BY: -- DRIVER
;
; CALLS: --  CHARTR
;
; MACRO CALLS: PRTLIN
;  CARRET
;
```

```
; ARGUMENTS: --  None
```

Image of page 72 for reference

```
; *************************************************************
 INTERR: PRTLIN CLRMSG,41 ; Request color choice
  CALL CHARTR ; Accept response
  CARRET  ; New line
 CPI 57H ; Did player request white ?
 JRZ IN04 ; Yes - branch
 SUB A ; Set computers color to white
 STA KOLOR
 LXI H,TITLE1 ; Prepare move list titles
 LXI D,TITLE4+2
 LXI B,6
 LDIR
 LXI H,TITLE2
 LXI D,TITLE4+9
 LXI B,6
 LDIR
 JMPR IN08 ; Jump
IN04: MVI A,80H ; Set computers color to black
 STA KOLOR
 LXI H,TITLE2 ; Prepare move list titles
 LXI D,TITLE4+2
 LXI B,6
 LDIR
 LXI H,TITLEI
 LXI D,TITLE4+9
 LXI B,6
 LDIR
IN08: PRTLIN PLYDEP,23 ; Request depth of search
 CALL CHARTR ; Accept response
 CARRET ;  New line
 LXI H,PLYMAX ; Address of ply depth variabl
 MVI M,2 ; Default depth of search
 CPI 31H ; Under minimum of 1 ?
 RM ;  Yes - return
 CPI 37H ; Over maximum of 6 ?
 RP ;  Yes - return
 SUI 30H ; Subtract Ascii constant
 MOV M,A ; Set desired depth
 RET ;   Return
```

Image of page 73 for reference

```
; ************************************************************
; COMPUTER MOVE ROUTINE
; ************************************************************
; FUNCTION: -- To control the search for the computers move
;   and the display of that move on the board
;   and in the move list.
;
; CALLED BY: -- DRIVER
;
; CALLS: -- FNDMOV
;   FCDMAT
;   MOVE
;   EXECMV
;   BITASN
;   INCHK
; MACRO CALLS:  PRTBLK
;   CARRET
;
; ARGUMENTS: --  None
************************************************************
CPTRMV: CALL FNDMOV ; Select best move
 LHLD BESTM ; Move list pointer variable
 SHLD MLPTRJ ; Pointer to move data
 LDA SCORE+1 ; To check for mates
 CPI 1 ; Mate against computer ?
 JRNZ CPOC ; No - jump
 MVI C,1 ; Computer mate flag
 CALL FCDMAT ; Full checkmate ?
CPOC: CALL MOVE ; Produce move on board array
 CALL EXECMV ; Make move on graphics board
   and return info about it
 MOV A,B ; Special move flags
 ANA A ; Special ?
 JRNZ CP10 ; Yes - jump
 MOV D,E ; "To" position of the move
 CALL BITASN ; Convert to Ascii
 S.HLD MVEMSG+3 ; Put in move message
 MOV D,C ; "From" position of the move
 CALL BITASN ; Convert to Ascii
 SHLD MVEMSG ; Put in move message
 PRTBLK MVEMSG,5 ; Output text of move
 JMPR CPlC ; Jump
CP10: BIT 1,B ; King side castle ?
 JRZ .+11 ; No - jump
 PRTBLK 0.0,5 ; Output "0-O"
 JMPR CPlC ; Jump
 BIT 2,B ; Queen side castle ?
 JRZ .+11 ; No - jump
```

Image of page 74 for reference

```
 PRTBLK 0.0.0,5 ;  Output "O-O-O"
 JMPR CPlC ; Jump
 PRTBLK P.PEP,5 ; Output "PxPep" - En passant
CPlC: LDA COLOR ; Should computer call check ?
```

```
 MOV B,A
 XRI 80H ; Toggle color
 STA COLOR
 CALL INCHK ; Check for check
 ANA A ; Is enemy in check ?
 MOV A,B ; Restore color
 STA COLOR
 JRZ CP24 ; No - return
 CARRET  ; New line
 LDA SCORE+1 ; Check for player mated
 CPI OFFH ; Forced mate ?
 CNZ TBCPMV ; No - Tab to computer column
 PRTBLK CKMSG,5 ; Output "check"
 LXI H,LINECT ; Address of screen line count
 INR M ; Increment for message
CP24: LDA SCORE+1 ; Check again for mates
 CPI OFFH ; Player mated ?
 RNZ ;  No - return
 MVI C,0 ; Set player mate flag
 CALL FCDMAT ; Full checkmate ?
 RET ;   Return
```

Image of page 75 for reference

```
; *xxxxxxxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;   FORCED MATE HANDLING
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx~xxxxxx
; FUNCTION: -- To examine situations where there exits
;   a forced mate and determine whether or
;   not the current move is checkmate. If it is,
;   a losing player is offered another game,
;   while a loss for the computer signals the
;   King to tip over in resignation.
; CALLED BY: -- CPTRMV
; CALLS: -- MATED
;   CHARTR
;   TBPLMV
; ARGUMENTS: -- The only value passed in a register is the
;   flag which tells FCDMAT whether the computer
;   or the player is mated.
; xxxxxxxxxxxx*xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
FCDMAT: LDA MOVENO ; Current move number
 MOV B,A ; Save
 LDA PMATE ; Move number where mate occurs
 SUB B ; Number of moves till mate
 ANA A ;Checkmate ?
 JRNZ FMOC ; No - jump
 BIT 0,C ; Check flag for who is mated
 JRZ FM04 ; Jump if player
 CARF9T  ; New line
 PRTLIN CKMSG,9 ; Print "CHECKMATE"
 CALL MATED ; Tip over King
 PRTLIN UWIN,7 ; Output "YOU WIN"
```

```
 JMPR FM08 ; Jump
FM04: PRTLIN MTMSG,4 ; Output "MATE"
 PRTLIN IWIN,5 ; Output "I WIN"
FM08: POP H ; Remove return addresses
 POP H
 CALL CHARTR ; Input any char to play again
FM09: CLRSCR  ; Blank screen
 PRTLIN AGAIN,22  "CARE FOR ANOTHER GAME?"
 JMP DRIV01 ; Jump (Rest of game init)
FMOC: BIT 0,C ; Who has forced mate ?
 RNZ  ; Return if player
 CARRET  ; New line
 ADI 30H ; Number of moves to Ascii
 STA MTPL ; Place value in message
 PRTLIN MTMSG,9 ; Output "MATE IN x MOVES"
 CALL TBPLMV ; Tab to players column
 RET  ; Return
```

Image of page 76 for reference

```
; *************************************************************
; TAB TO PLAYERS COLUMN
; *************************************************************
; FUNCTION: -- To space over in the move listing to the
;   column in which the players moves are being
;   recorded. This routine also reprints the
;   move number.
;
; CALLED BY: -- PLYRMV
;
; CALLS: --  None
;
; MACRO CALLS:  PRTBLK
;
; ARGUMENTS: --  None
;
; *************************************************************
TBPLCL: PRTBLK MVENUM,3 ; Reproduce move number
 LDA KOLOR ; Computers color
 ANA A ; Is computer white ?
 RNZ ;  No - return
 PRTBLK SPACE,6 ; Tab to next column
 RET ;   Return

; *************************************************************
TAB TO COMPUTERS COLUMN
; *************************************************************
; FUNCTION: -- To space over in the move listing to the
;   column in which the computers moves are
;   being recorded. This routine also reprints
;   the move number.
;
; CALLED BY: -- DRIVER
;  CPTRMV
```

```
;
; CALLS: --  None
;
; MACRO CALLS:  PRTBLK
;
; ARGUMENTS: --  None
; ************************************************************
TBCPCL: PRTBLK MVENUM,3 ; Reproduce move number
 LDA KOLOR ; Computer's color
 ANA A ; Is computer white ?
 RZ ;  Yes - return
 PRTBLK SPACE,6 ; Tab to next column
 RET ;   Return
```

Image of page 77 for reference

```
; ************************************************************
; TAB TO PLAYERS COLUMN W/0 MOVE NO.
; ************************************************************
; FUNCTION: -- Like TBPLCL, except that the move number
;   is not reprinted.
;
; CALLED BY: -- FCDMAT
; ************************************************************
; TBPLMV: PRTBLK SPACE,3
;  LDA KOLOR
;  ANA A
;  RNZ
;  PRTBLK SPACE,6
;  RET
;
; ************************************************************
TAB TO COMPUTERS COLUMN W/O MOVE NO.
; ************************************************************
; FUNCTION: -- Like TBCPCL, except that the move number
;   is not reprinted.
;
;CALLED BY: -- CPTRMV
; ************************************************************
TBCPMV: PRTBLK SPACE,3
 LDA KOLOR
 ANA A
 RZ
 PRTBLK SPACE,6
 RET
```

Image of page 78 for reference

```
; *************************************************************
; BOARD INDEX TO ASCII SQUARE NAME
; *************************************************************
; FUNCTION: -- To translate a hexadecimal index in the
;   board array into an ascii description
;   of the square in algebraic chess notation.
;
; CALLED BY: -- CPTRMV
;
; CALLS: --  DIVIDE
;
; ARGUMENTS: -- Board index input in register D and the Ascii
; square name is output in register pair HL.
; *************************************************************
BITASN: SUB A ; Get ready for division
 MVI E,10
 CALL DIVIDE ; Divide
 DCR D ; Get rank on 1-8 basis
 ADI 60H ; Convert file to Ascii (a-h)
 MOV L,A ; Save
 MOV A,D ; Rank
 ADI 30H ; Convert rank to Ascii (1-8)
 MOV H,A ; Save
 RET  ; Return
```

Image of page 79 for reference

```
; *************************************************************
; PLAYERS MOVE ANALYSIS
; *************************************************************
; FUNCTION: -- To accept and validate the players move
;   and produce it on the graphics board. Also
;   allows player to resign the game by
;   entering a control-R.
;
; CALLED BY: -- DRIVER
;
; CALLS: -- CHARTR
;   ASNTBI
;   VALMOV
;   EXECMV
;   PGIFND
;   TBPLCL
;
; ARGUMENTS: --  None
; *************************************************************
PLYRMV: CALL CHARTR ; Accept "from" file letter
 CPI 12H ; Is it instead a Control-R ?
 JZ FM09 ; Yes - jump
 MOV H,A ; Save
 CALL CHARTR ; Accept "from" rank number
 MOV L,A ; Save
 CALL ASNTBI ; Convert to a board index
 SUB B ; Gives board index, if valid
 JRZ PL08 ; Jump if invalid
```

```
 STA MVEMSG ; Move list "from" position
 CALL CHARTR ; Accept separator & ignore it
 CALL CHARTR ; Repeat for "to" position
 MOV H,A
 CALL CHARTR
 MOV L,A
 CALL ASNTBI
 SUB B
 JRZ PL08
 STA MVEMSG+1 ; Move list "to" position
 CALL VALMOV ; Determines if a legal move
 ANA A ; Legal ?
 JNZ PL08 ; No - jump
 CALL EXECMV ; Make move on graphics board
 RET  ; Return
PL08: LXI H,LINECT ; Address of screen line count
 INR M ; Increase by 2 for message
 INR M
 CARRET  ; New line
 CALL PGIFND ; New page if needed
 PRTLIN INVAL1,12 ; Output "INVALID MOVE"
 PRTLIN INVAL2,9 ; Output "TRY AGAIN"
 CALL TBPLCL ; Tab to players column
 JMP PLYRMV ;  Jump
```

Image of page 80 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  ASCII SQUARE NAME TO BOARD INDEX
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- To convert an algebraic square name in
;   Ascii to a hexadecimal board index.
;   This routine also checks the input for
;   validity.
; CALLED BY: -- PLYRMV
; CALLS: -- MLTPLY
;
; ARGUMENTS: -- Accepts the square name in register pair HL and
; outputs the board index in register A. Register
; B = 0 if ok. Register B = Register A if invalid.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
ASNTBI: MOV A,L ; Ascii rank
 SUI 30H ; Rank 1 - 8
 CPI 1 ; Check lower bound
 JM AT04 ; Jump if invalid
 CPI 9 ; Check upper bound
 JRNC AT04 ; Jump if invalid
 INR A . ; Rank 2 - 9
 MOV D,A ; Ready for multiplication
 MVI E,10
 CALL MLTPLY ; Multiply
 MOV A,H ; Ascii file letter (a - h)
 SUI 40H ; File 1 - 8
 CPI 1 ; Check lower bound
```

```
 JM AT04 ; Jump if invalid
 CPI 9 ; Check upper bound
 JRNC AT04 ; Jump if invalid
 ADD D ; File+Rank(20-90)=Board index
 MVI B,0 ; Ok flag
 RET  ; Return
AT04: MOV B,A ; Invalid flag
 RET  ; Return
```

Image of page 81 for reference

```
; *********************************************************
; VALIDATE MOVE SUBROUTINE
; *********************************************************
; FUNCTION: -- To check a players move for validity.
;
; CALLED BY: -- PLYRMV
;
; CALLS: -- GENMOV
;   MOVE
;   INCHK
;   UNMOVE
;
; ARGUMENTS: -- Returns flag in register A, 0 for valid and 1 for
; invalid move.
; *********************************************************
VALMOV: LHLD MLPTRJ ; Save last move pointer
 PUSH H ; Save register
 LDA KOLOR ; Computers color
 XRI 80H ;Toggle color
 STA COLOR ; Store
 LXI H,PLYIX-2 ; Load move list index
 SHLD MLPTRI
 LXI H,MLIST+1024 ; Next available list pointer
 SHLD MLNXT
 CALL GENMOV ; Generate opponents moves
 LXI X,MLIST+1024 ; Index to start of moves
VA5: LDA MVEMSG ; "From" position
 CMP MLFRP(X) ; Is it in list ?
 JRNZ VA6 ; No - jump
 LDA MVEMSG+l ; "To" position
 CMP MLTOP(X) ; Is it in list ?
 JRZ VA7 ; Yes - jump
VA6: MOV E,MLPTR(X) ; Pointer to next list move
 MOV D,MLPTR+1 (X)
 XRA A ; At end of list ?
 CMP D
 JRZ VA10 ; Yes - jump
 PUSH D ; Move to X register
 POP X
 JMPR VA5 ; Jump
VA7: SIXD MLPTRJ ; Save opponents move pointer
 CALL MOVE ; Make move on board array
 CALL INCHK ; Was it a legal move ?
```

```
 ANA A
 JRNZ VA9 ; No - jump
VA8: POP H ; Restore saved register
 RET  ; Return
VA9: CALL UNMOVE ; Un-do move on board array
VA10: MVI A,l ; Set flag for invalid move
 POP H ; Restore saved register
 SHLD MLPTRJ ; Save move pointer
 RET ;   Return
```

Image of page 82 for reference

```
; *********************************************************
; ACCEPT INPUT CHARATER
; *********************************************************
; FUNCTION: -- Accepts a single character input from the
;   console keyboard and places it in the A
;   register. The character is also echoed on
;   the video screen, unless it is a carrage
; return, line feed, or backspace. Lower case
; alphabetic characters are folded to upper case.
;
; CALLED BY: -- DRIVER
;  INTERR
;  PLYRMV
;  ANALYS
;
; CALLS: --  None
;
; ARGUMENTS: -- Character input is output in register A.
;
;  NOTES: -- This routine contains a reference to a
;      monitor function of the Jove monitor, there-
;    for the first few lines of this routine are
;    system dependent.
; *********************************************************
CHARTR: RST 7 ; Jove monitor single char inpt
 .BYTE  81H,0
 CPI ODH ; Carriage return ?
 RZ ;  Yes - return
 CPI OAH ; Line feed ?
 RZ ;  Yes - return
 CPI 08H ; Backspace ?
 RZ ;  Yes - return
 RST 7 ; Jove monitor single char echo
 .BYTE  81H,lAH
 ANI 7FH ; Mask off parity bit
 CPI 7BH ; Upper range check (z+l)
 RP ;  No need to fold - return
 CPI 61H ; Lower-range check (a)
 RM ;  No need to fold - return
 SUI 20H ; Change to one of A-Z
 RET ;   Return
```

Image of page 83 for reference

```
; *********************************************************
; NEW PAGE IF NEEDED
; *********************************************************
; FUNCTION: -- To clear move list output when the column
;   has been filled.
;
; CALLED BY: -- DRIVER
;  PLYRMV
;  CPTRMV
;
; CALLS: --  DSPBRD
;
; ARGUMENTS: -- Returns a 1 in the A register if a new
; page was turned.
; xx**x*xx*xx**x***xx*xx********x**x***x*xx*x*x*xxxx*xx*xx
PGIFND: LXI H,LINECT ; Addr of page position counter
 INR M ; Increment
 MVI A,1BH ; Page bottom ?
 CMP M
 RNC  ; No - return
 CALL DSPBRD ; Put up new page
 PRTLIN TITLE4,15 ; Re-print titles
 PRTLIN TITLE3,15
 MVI A,1 ; Set line count back to 1
 STA LINECT
 RET ; Return
```

Image of page 84 for reference

```
; *********************************************************
; DISPLAY MATED KING
; *********************************************************
;  FUNCTION: -- To tip over the computers King when
;    mated.
;
; CALLED BY: -- FCDMAT
;
; CALLS: -- CONVRT
;   BLNKER
;   INSPCE (Abnormal Call to IP04)
;
; ARGUMENTS: --  None
; *********************************************************
MATED: LDA KOLOR ; Computers color
 ANA A ; Is computer white ?
 JRZ .+9 ; Yes - skip
 MVI C,2 ; Set black piece flag
 LDA POSK+1 ; Position of black King
 JMPR MA08 ; Jump
```

```
 MOV C,A ; Clear black piece flag
 LDA POSK ; Position of white King
MA08: STA BRDPOS ; Store King position
 STA ANBDPS ; Again
 CALL CONVRT ; Getting norm address in HL
 MVI A,7 ; Piece value of toppled King
 MVI B,10 ; Blink parameter
 CALL BLNKER ; Blink King position
 LXI Y,MAOC ; Prepare for abnormal call
 PUSH Y
 PUSH H
 PUSH B
 PUSH D
 PUSH X
 PUSH PSW
 JMP IP04 ; Call INSPCE
MAOC: MVI B,10 ; Blink again
 LDA ANBDPS
 STA BRDPOS
 CALL BLNKER
 RET  ; Return
```

Image of page 85 for reference

```
; *********************************************************
; SET UP POSITION FOR ANALYSIS
; *********************************************************
; FUNCTION: -- To enable user to set up any position
;   for analysis, or to continue to play
;   the game. The routine blinks the board
;   squares in turn and the user has the option
;   of leaving the contents unchanged by a
;   carriage return, emptying the square by a 0,
;   or inputting a piece of his chosing. To
;   enter a piece, type in piece-code,color-code,
;   moved-code.
;
;  Piece-code is a letter indicating the
; desired piece:
;  K - King
;  Q - Queen
;  R - Rook
;  B - Bishop
;  N - Knight
;  P -  Pawn
;
; Color code is a letter, W for white, or B for
; black.
;
; Moved-code is a number. 0 indicates the piece has never
; moved. 1 indicates the piece has moved.
;
; A backspace will back up in the sequence of blinked
```

```
; squares. An Escape will terminate the blink cycle and
; verify that the position is correct, then procede
; with game initialization.
;
; CALLED BY: -- DRIVER
;
; CALLS: -- CHARTR
;   DPSBRD
;   BLNKER
;   ROYALT
;   PLYRMV
;   CPTRMV
; MACRO CALLS:  PRTLIN
;   EXIT
;   CLRSCR
;   PRTBLK
;   CARRET
;
; ARGUMENTS: --  None
; ************************************************************
```

Image of page 86 for reference

```
ANALYS: PRTLIN ANAMSG,37 ; "CARE TO ANALYSE A POSITION?"
 CALL CHARTR ; Accept answer
 CARRET ;  New line
 CPI 4EH ; Is answer a "N" ?
 JRNZ AN04 ; No - jump
 EXIT ;  Return to monitor
AN04: CALL DSPBRD ; Current board position
 MVI A,21 ; First board index
AN08: STA ANBDPS ; Save
 STA BRDPOS
 CALL CONVRT ; Norm address into HL register
 STA M1 ; Set up board index
 LIXD M1
 MOV A,BOARD(X) ; Get board contents
 CPI OFFH ; Boarder square ?
 JRZ AN19 ; Yes - jump
 MVI B,4H ; Ready to blink square
 CALL BLNKER ; Blink
 CALL CHARTR ; Accept input
 CPI 1BH ; Is it an escape ?
 JRZ ANlB ; Yes - jump
 CPI 08H ; Is it a backspace ?
 JRZ ANIA ; Yes - jump
 CPI ODH ; Is it a carriage return ?
 JRZ AN19 ; Yes - jump
 LXI B,7 ; Number of types of pieces + 1
 LXI H,PCS ; Address of piece symbol table
 CCIR ;  Search
 JRNZ AN18 ; Jump if not found
 CALL CHARTR ; Accept and ignore separator
```

```
 CALL CHARTR ; Color of piece
 CPI 42H ; Is it black ?
 JRNZ .+4 ; No - skip
 SET 7,C ; Black piece indicator
 CALL CHARTR ; Accept and ignore separator
 CALL CHARTR ; Moved flag
 CPI 31H ; Has piece moved ?
 JRNZ AN18 ; No - jump
 SET 3,C ; Set moved indicator
AN18: MOV BOARD(X),C ; Insert piece into board array
 CALL DSPBRD ; Update graphics board
AN19: LDA ANBDPS ; Current board position
 INR A ; Next
 CPI 99 ; Done ?
 JRNZ AN08 ; No - jump
 JMPR AN04 ; Jump
ANlA: LDA ANBDPS ; Prepare to go back a square
```

Image of page 87 for reference

```
 SUI 3 ; To get around boarder
 CPI 20 ; Off the other end ?
 JNC AN08 ; No - jump
 MVI A,98 ; Wrap around to top of screen
ANOB: JMP AN08 ;  Jump
ANlB: PRTLIN CRTNES,14 ; Ask if correct
 CALL CHARTR ; Accept answer
 CPI 4EH ; Is it "N" ?
 JZ AN04 ; No - jump
 CALL ROYALT ; Update positions of royalty
 CLRSCR ;  Blank screen
 CALL INTERR ; Accept color choice
ANlC: PRTLIN WSMOVE,17 ; Ask whose move it is
 CALL CHARTR ; Accept response
 CALL DSPBRD ; Display graphics board
 PRTLIN TITLE4,15 ; Put up titles
 PRTLIN TITLE3,15
 CPI 57H ; Is is whites move ?
 JZ DRIV04 ; Yes - jump
 PRTBLK MvENUM,3 ; Print move number
 PRTBLK SPACE,6 ; Tab to blacks column
 LDA KOLOR ; Computer's color
 ANA A ; Is computer white ?
 JRNZ AN20 : No - jump
 CALL PLYRMV ; Get players move
 CARRET  ; New line
 JMP DROC ; Jump
AN20: CALL CPTRMV ; Get computers move
 CARRET  ; New line
 JMP DROC ;  Jump
```

Image of page 88 for reference

```
; *********************************************************
; UPDATE POSITIONS OF ROYALTY
; *********************************************************
; FUNCTION: -- To update the positions of the Kings
;   and Queen after a change of board position
;   in ANALYS.
;
; CALLED BY: -- ANALYS
;
; CALLS: --  None
;
; ARGUMENTS: --  None
; *********************************************************
ROYALT: LXI H,POSK ; Start of Royalty array
 MVI B,4 ; Clear all four positions
 MVI M,0
 INX H
 DJNZ .-3
 MVI A,21 ; First board position
RY04: STA M1 ; Set up board index
 LXI H,POSK ; Address of King position
 LIXD M1
 MOV A,BOARD(X) ;Fetch board contents
 BIT 7,A ; Test color bit
 JRZ .+3 ; Jump if white
 INX H ; Offset for black
 ANI 7 ; Delete flags, leave piece
 CPI KING ; King ?
 JRZ RY08 ; Yes - jump
 CPI QUEEN ; Queen ?
 JRNZ RYOC ; No - jump
 INX H ; Queen position
 INX H ; Plus offset
RY08: LDA M1 ; Index
 MOV M,A ; Save
RYOC: LDA M1 ; Current position
 INR A ; Next position
 CPI 99 ; Done.?
 JRNZ RY04 ; No - jump
 RET  ; Return
```

Image of page 89 for reference

```
; *********************************************************
; SET UP EMPTY BOARD
; *********************************************************
; FUNCTION: -- Diplay graphics board and pieces.
;
```

```
; CALLED BY: -- DRIVER
;  ANALYS
;  PGIFND
;
; CALLS:  -- CONVRT
;     INSPCE
;  ARGUMENTS: --  None
; NOTES:  --  This routine makes use of several fixed
;     addresses in the video stoFage area of
;     the Jupiter III computer, and is therefor
;     system dependent. Each such reference will
;     be marked.
; ************************************************************
 DSPBRD: PUSH  B  ; Save registers
 PUSH  D
 PUSH  H
 PUSH PSW
 CLRSCR  ; Blank screen
 LXI H,OC000H ; System Dependent-First video
   address
 MVI M,80H ; Start of blank border
 LXI D,0C001H ; Sys Dep- Next boarder square
 LXI 8,15 ; Number of bytes to be moved
 LDIR  ; Blank boarder bar
 MVI M,OAAH ; First black boarder box
 INR L ; Next block address
 MVI B,6 ; Number to be moved
DB04: MVI M,80H ; Create white block
 INR L ; Next block address
 DJNZ DB04 ; Done ? No - jump
 MVI B,6 ; Number of repeats
DB08: MVI M,OBFH ; Create black box
 INR L ; Next block address
 DJNZ DB08 ; Done ? No - jump
 XCHG  ; Get ready for block move
 LXI B,36 ; Bytes to be moved
 LDIR  ; Move - completes first bar
 LXI H,0C000H ; S D - First addr to be copied
 LXI B,ODOH ; Number of blocks to move
 LDIR  ; Completes first rank
 LXI H,OC016H ; S D - Start of copy area
 LXI B,6 ; Number of blocks to move
```

Image of page 90 for reference

```
 LDIR ;  First black square done
 LXI H,OC010H ; S D - Start copy area
 LXI B,42 ; Bytes to be moved
 LDIR ;  Rest of bar done
 LXI H,OC100H ; S D - Start of copy area
 LXI B,OCOH ; Move three bars
 LDIR ;  Next rank done
 LXI H,OC000H ; S D - Copy rest of screen
 LXI B,600H ; Number of blocks
 LDIR ;  Board done
BSETUP: MVI A,21 ; First board index
```

```
BSET04: STA BRDPOS ; Ready parameter
 CALL CONVRT ; Norm addr into HL regtisters
 CALL INSPCE ; Insert that piece onto board
 INR A ; Next square
 CPI 99 ; Done ?
 JRC BSET04 ; No - jump
 POP PSW ; Restore registers
 POP H
 POP D
 POP B
 RET
```

Image of page 91 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; INSERT PIECE SUBROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- This subroutine places a piece onto a
;   given square on the video board. The piece
;   inserted is that stored in the board array
;   for that square.
; CALLED BY: -- DPSPRD
;   MATED
; CALLS: -- MLTPLY
;
; ARGUMENTS: -- Norm address for the square in register pair HL.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
INSPCE: PUSH H ; Save registers
 PUSH B
 PUSH D
 PUSH X
 PUSH PSW
 LDA BRDPOS ; Get board index
 STA M1 ; Save
 LIXD M1 ; Index into board array
 MOV A,BOARD(X) ; Contents of board array
 ANA A ; Is square empty ?
 JRZ IP2C ; Yes - jump
 CPI OFFH ; Is it a boarder square ?
 JRZ IP2C ; Yes - jump
 MVI C,0 ; Clear flag register
 BIT 7,A ; Is piece white ?
 JRZ IP04 ; Yes - jump
 MVI C,2 ; Set black piece flag
IP04: ANI 7 ; Delete flags, leave piece
 DCR A ; Piece on a 0 - 5 basis
 MOV E,A ; Save
 MVI D,16 ; Multiplier
 CALL MLTPLY ; For loc of piece in table
 M'OV A,D ; Displacement into block table
 STA INDXER ; Low order index byte
 LIXD INDXER ; Get entire index
 BIT O,M ; Is square white ?
```

```
 JRZ IP08 ; Yes - jump
 INR C ; Set compliment flag
IP08: INR L ; Address of first alter block
 PUSH H ; Save
 MVI D,® ; Bar counter
IPOC: MVI B,4 ; Block counter
IP10: MOV A,BLOCK(X) ; Bring in source block
 BIT 0,C ; Should it be complemented ?
```

Image of page 92 for reference

```
 JRZ IP14 ; No - jump
 XRI 3FH ; Graphics complement
IP14: MOV M,A ; Store block
 INR L ; Next block
 INX X ; Next source block
 DJNZ "P10 ; Done ? No - jump
 MOV A,L ; Bar increment
 ADI 3CH
 MOV L,A
 INR D ; Bar counter
 BIT 2,D ; Done ?
 JRZ IPOC ; No - jump
 POP H ; Address of Norm + 1
 BIT 0,C ; Is square white ?
 JRNZ IP18 ; No - jump
 BIT 1,C ; Is piece white ?
 JRNZ IP2C ; No - jump
 JMPR IPIC ; Jump
IP18: BIT 1,C ; Is piece white ?
 JRZ IP2C ; Yes - jump
IPlC: MVI D,6 ; Multiplier
 CALL MLTPLY ; Multiply for displacement
 MOV A,D ; Kernel table displacement
 STA INDXER ; Save
 LIXD INDXER ; Get complete index
 MOV A,L ; Start of Kernel
 ADI 40H
 MOV L,A
 MVI D,0 ; Bar counter
IP20: MVI B,3 ; Block counter
IP24: MOV A,KERNEL(X) ; Kernel block
 BIT 1,C ; Need to complement ?
 JRNZ IP28 ; No - jump
 XRI 3FH ; Graphics complement
IP28: MOV M,A ; Store block
 INR L ; Next target block
 INX X ; Next source block
 DJNZ IP24 ; Done ? No - jump
 MOV A,L ; Bar increment
 ADI 3DH
 MOV L,A
 INR D ; Bar counter
 BIT 1,D ; Done ?
```

```
 JRZ IP20 ; Repeat bar move
IP2C: POP PSW ; Restore registers
 POP X
 POP D
 POP B
 POP H
 RET
```

Image of page 93 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; BOARD INDEX TO NORM ADDRESS SUBR.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;  FUNCTION: -- Converts a hexadecimal board index into
;    a Norm address for the square.
;  CALLED BY: -- DSPBRD
;    INSPCE
;    ANALYS
;    MATED
;  CALLS: -- DIVIDE
;    MLTPLY
;ARGUMENTS: -- Returns the Norm address in register pair
;    HL.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
CONVRT: PUSH B ; Save registers
 PUSH D
 PUSH PSW
 LDA BRDPOS ; Get board index
 MOV D,A ; Set up dividend
 SUB A
 MVI E,10 ; Divisor
 CALL DIVIDE ; Index into rank and file
  ; file (1-8) & rank (2-9)
 DCR D ; For rank (1-8)
 DCR A ; For file (0-7)
 MOV C,D ; Save
 MVI D,6 ; Multiplier
 MOV E,A ; File number is multiplicand
 CALL MLTPLY ; Giving file displacement
 MOV A,D ; Save
 ADI 10H ; File norm address
 MOV L,A ; Low order address byte
 MVI A,8 ; Rank adjust
 SUB C ; Rank displacement
 ADI OCOH ; Rank Norm address
 MOV H,A ; High order addres byte
 POP PSW ; Restore registers
 POP D
 POP B
 RET  ; Return
```

Image of page 94 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; POSITIVE INTEGER DIVISION
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
DIVIDE: PUSH B
 MVI B,8
DD04: SLAR D
 RAL
 SUB E
 JM .+6
 INR D
 JMPR .+3
 ADD E
 DJNZ DD04
 POP B
 RET

; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; POSITIVE INTEGER MULTIPLICATION
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MLTPLY: PUSH B
 SUB A
 MVI B,8
ML04: BIT 0,D
 JRZ .+3
 ADD E
 SRAR A
 RARR D
 DJNZ ML04
 POP B
 RET
```

Image of page 95 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; SQUARE BLINKER
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
;
; FUNCTION: -- To blink the graphics board square to signal
;   a piece's intention to move, or to high-
; light the square as being alterable
;    in ANALYS.
;
; CALLED BY: -- MAKEMV
;  ANALYS
;  MATED
;
; CALLS: --  None
;
; ARGUMENTS: -- Norm address of desired square passed in register
; pair HL. Number of times to blink passed in
; register B.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
BLNKER: PUSH PSW ; Save registers
 PUSH B
 PUSH D
```

```
 PUSH H
 PUSH X
 SHLD NORMAD ; Save Norm address
BL04: MVI D,0 ; Bar counter
BL08: MVI C,0 ; Block counter
BLOC: MOV AIM ; Fetch block
 XRI 3FH ; Graphics complement
 MOV M,A ; Replace block
 INR L ; Next block address
 INR C ; Increment block counter
 MOV A,C
 CPI 6 ; Done ?
 JRNZ BLOC ; No - jump
 MOV A,L ; Address
 ADI 3AH ; Adjust square position
 MOV L,A ; Replace address
 INR D ; Increment bar counter
 BIT 2,D ; Done ?
 JRZ BL08 ; No - jump
 LHLD NORMAD ; Get Norm address
 PUSH B ; Save register
 LXI B,3030H ; Delay loop, for visibility
BL10: DJNZ BL10
 DCR C
 JRNZ BL10
```

Image of page 96 for reference

```
 POP B ; Restore register
 DJNZ BL04 ; Done ? No - jump
 POP X ; Restore registers
 POP H
 POP D
 POP B
 POP PSW
 RET ; Return
```

Image of page 97 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; EXECUTE MOVE SUBROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNCTION: -- This routine is the control routine for
;   MAKEMV. It checks for double moves and
;   sees that they are properly handled. It
;   sets flags in the B register for double
;   moves:
;    En Passant -- Bit 0
; 0-0 -- Bit 1
; 0-0-0 -- Bit 2
;
```

```
; CALLED BY: -- PLYRMV
;  CPTRMV
;
;  CALLS: -- MAKEMV
;
;  ARGUMENTS: -- Flags set in the B register as described
; above.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
EXECMV: PUSH X ; Save registers
 PUSH PSW
 LIXD MLPTRJ ; Index into move list
 MOV C,MLPRP(X) ; Move list "from" position
 MOV E,MLTOP(X) ; Move list "to" position
 CALL MAKEMV ; Produce move
 MOV D,MLFLG (X) ; Move list flags
 MVI B,0
 BIT 6,D ; Double move ?
 JRZ EX14 ; No - jump
 LXI D,6 ; Move list entry width
 DADX D ; Increment MLPTRJ
 MOV C,MLFRP(X) ; Second "from" position
 MOV E,MLTOP(X) ; Second "to" position
 MOV A,E ; Get "to" position
 CMP C ; Same as "from" position ?
 JRNZ EX04 ; No - jump
 INR B ; Set en passant flag
 JMPR EX10 ; Jump
EX04: CPI 1AH ; White 0-0 ?
 JRNZ EX08 ; No - jump
 SET 1,8 ; Set 0-O flag
 JMPR EX10 ; Jump
EX08: CPI 60H ; Black 0-0 ?
 JRNZ EXOC ; No - jump
 SET 1,8 ; Set 0-0 flag
 JMPR 8X10 ; Jump
EXOC: SET 2,B ; Set 0-0-0 flag
EX10: CALL MAkEMV ; Make 2nd move on board
EX14: POP PSW ; Restore registers
 POP X
 RET  ; Return
```

Image of page 98 for reference

```
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; MAKE MOVE SUBROUTINE
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
; FUNDTION: -- Moves the piece on the board when a move
;   is made. It blinks both the "from" and
;   "to" positions to give notice of the move.
;
; CALLED BY: -- EXECMV
;
; CALLS: -- CONVRT
;   BLNKER
;   INSPCE
```

```
;
; ARGUMENTS: -- The "from" position is passed in register C, and the
; "to" position in register E.
; xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
MAKEMV: PUSH PSW ; Save register
 PUSH B
 PUSH D
 PUSH H
 MOV A,C ; "From" position
 STA BRDPOS ; Set up parameter
 CALL CONVRT ; Getting Norm address in HL
 MVI B,10 ; Blink parameter
 CALL BLNKER ; Blink "from" square
 MOV A,M ; Bring in Norm 1plock
 INR L ; First change block
 MVI D,0 ; Bar counter
MM04: MVI B,4 ; Block counter
MM08: MOV M,A ; Insert blank block
 INR L ; Next change block
 DJNZ MM08 ; Done ? No - jump
 MOV C,A ; Saving norm block
 MOV A,L ; Bar increment
 ADI 3CH
 MOV L,A
 MOV A,C ; Restore Norm block
 INR D
 BIT 2,D ; Done ?
 JRZ MM04 ; No - jump
 MOV A,E ; Get "to" position
 STA BRDPOS ; Set up parameter
 CALL CONVRT ; Getting Norm address in HL
 MVI B,10 ; Blink parameter
 CALL INSPCE ; Inserts the piece
 CALL BLNKER ; Blinks "to" square
 POP H ; Restore registers
 POP D
 POP B
 POP PSW
 RET  ; Return
```

Image of page 99 for reference

TDL/ZILOG Mnemonics Conversion

symbols used

SYMBOL
OPERATION

r
one of the 8-bit registers A,B,C,D,E,H,L

n
any 8-bit absolute value

ii
an index register reference, either X or Y

d
an 8-bit index displacement, where -128< d< 127

zz
B for the BC register pair, D for the DE pair

nn
any 16-bit value, absolute or relocatable

rr
B for the BC register pair, D for the DE pair, H for the HL pair, SP for the stack pointer

qq
B for the BC register pair, D for the DE pair, H for the HL pair, PSW for the A/Flag pair

s
any of r (defined above), M, or d(ii)

IFF
interrupt flip-flop

CY
carry flip.-flop

ZF
zero flag

tt
B for the BC register pair, D for the DE pair, SP for the stack pointer, X for index register IX

uu
B for the BC register pair, D for the DE pair, SP for the stack pointer, Y for index register IY

b
a bit position in an 8-bit byte, where the bits are numbered from right to left 0 to 7

PC
program counter

b{n}
bit n of the 8-bit value or register v

vv/H
the most significant byte of the 16-bit value or register vv

vv/L
the least significant byte of the 16-bit value or register vv

Iv
an input operation on port v

Ov
an output operation on port v

w <- v
the value of w is replaced by the value of v

w <-> v
the value of w is exchanged with the value of v

Image of page 100

Image of page 101

Image of page 102

Image of page 103

Image of page 104

Image of page 105

Image of page 106

Image of page 107

Image of page 108

Image of page 109

Image of page 110

Image of page 111

ADJPTR
33

ADMOVE
36

ANALYS
85

ASCEND
64

ASNTBI
80

ATKSAV
42

ATTACK
39

BITASN
76

BLNKER
94

BOOK
65

CASTLE
34

CHARTR
82

CONVRT
93

CPTRMV
73

DIVIDE
93

DRIVER
69

DSPBRD
89

ENPSNT
32

EVAL
60

EXECMV
97

FCDMAT
75

FNDMOV
59

GENMOV
38

INCHK
38

INITBD
27

INSPCE
91

INTERR
71

LIMIT
51

MAKEMV
98

MATED
84

MLTPLY
94

MOVE
54

MPIECE
30

NEXTAD
49

PATH
29

PGIFND
83

PINFND
44

PLYRMV
79

PNCK
43

POINTS
49

ROYALT
88

SORTM
58

TBCPCL
78

TBPLCL
76

TBPLMV
78

UNMOVE
56

vVALMOV
80

XCHNG
47